PopPUNK Documentation

Release 2.6.3

John Lees and Nicholas Croucher

Dec 16, 2023

CONTENTS:

1 Installation 3

2 Overview 5

3 Sketching (`--create-db`) 9

4 Data quality control (`--qc-db`) 17

5 Fitting new models (`--fit-model`) 19

6 Distributing PopPUNK models 45

7 Query assignment (`poppunk_assign`) 47

8 Creating visualisations 53

9 Minimum spanning trees 63

10 Subclustering with PopPIPE 67

11 Using GPUs 71

12 Troubleshooting 75

13 Scripts 79

14 Iterative PopPUNK 83

15 Citing PopPUNK 89

16 Reference documentation 91

17 Roadmap 139

18 Miscellaneous 141

19 Why use PopPUNK? 145

20 Citation 147

21 Index 149

Python Module Index 151

Index 153

PopPUNK is a tool for clustering genomes. We refer to the clusters as variable-length-k-mer clusters, or **VLKCs**. Biologically, these clusters typically represent distinct strains. We refer to subclusters of strains as lineages.

If you are new to PopPUNK, we'd recommend starting on *Installation*, then by reading the *Overview*.

The first version was targeted specifically as bacterial genomes, but the current version has also been used for viruses (e.g. enterovirus, influenza, SARS-CoV-2) and eukaryotes (e.g. *Candida* sp., *P. falciparum*). Under the hood, PopPUNK uses pp-sketchlib to rapidly calculate core and accessory distances, and machine learning tools written in python to use these to cluster genomes. A detailed description of the method can be found in the paper.

Important: Looking for older versions of the documentation? For previous versions with the old API (`--assign-query`, `--refine-fit` etc) see v2.2.0. For older versions which used mash, see v1.2.0.

CHAPTER

ONE

INSTALLATION

The easiest way to install is through conda, which will also install the dependencies:

```
conda install poppunk
```

Then run with `poppunk`.

Important: From v2.1.0 onwards, PopPUNK requires python3.8 to run (which on many default Linux installations is run using `python3` rather than `python`).

Important: From v2.1.2 onwards, PopPUNK no longer supports `mash`. If you want to use older databases created with `mash`, please downgrade to <v2

1.1 Installing with conda (recommended)

If you do not have `conda` you can install it through miniconda and then add the necessary channels:

```
conda config --add channels defaults
conda config --add channels bioconda
conda config --add channels conda-forge
```

Then run:

```
conda install poppunk
```

If you want to use GPUs, take a look at *Using GPUs*.

If you are having conflict issues with conda, our advice would be:

- Remove and reinstall miniconda.
- Never install anything in the base environment
- Create a new environment for PopPUNK with `conda create -n pp_env poppunk`

If you have an older version of PopPUNK, you can upgrade using this method – you may also wish to specify the version, for example `conda install poppunk==2.3.0` if you wish to upgrade.

conda-forge also has some helpful tips: https://conda-forge.org/docs/user/tipsandtricks.html

1.2 Installing with pip

If you do not have conda, you can also install through pip:

```
python3 -m pip install poppunk
```

This may not deal with all necessary dependencies, but we are working on it and it should again be possible in an upcoming release.

1.3 Clone the code

You can also clone the github to run the latest version, which is executed by:

```
git clone https://github.com/bacpop/PopPUNK.git && cd PopPUNK
python3 setup.py build
python3 setup.py install
python3 poppunk-runner.py
```

This will also give access to the *Scripts*.

You will need to install the dependencies yourself (you can still use conda or pip for this purpose). See `environment.yml`:

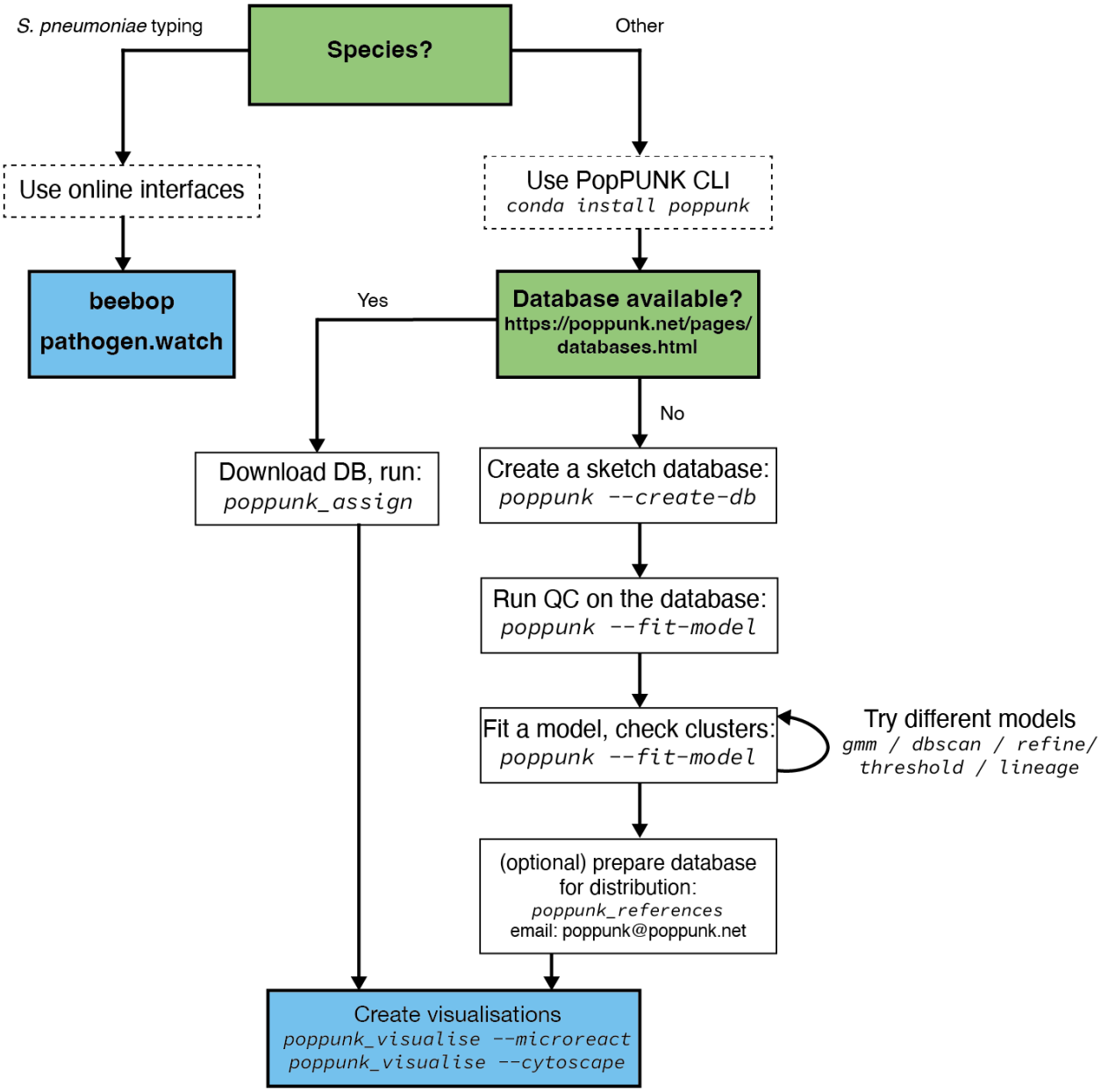
```
mamba env create -f environment.yml
conda activate pp_env
```

CHAPTER

TWO

OVERVIEW

This page details the way in which we would advise that you *should* use and run PopPUNK, if possible.



2.1 Use the command line interface

2.1.1 Installation and version

Install via conda if possible. Version 2.5.0 of PopPUNK and version 2.0.0 of pp-sketchlib are the current minimum supported versions.

2.1.2 Use query assignment mode

If a database is available for your species (see https://www.bacpop.org/poppunk/) we would strongly recommend downloading it to use to cluster your genomes. This has many advantages:

- No need to run through the potentially complex model fitting.
- Assured model performance.
- Considerably faster run times.
- Use existing cluster definitions.
- Use the context of large, high quality reference populations to interpret your genomes' clusters.

See *Query assignment (poppunk_assign)* for instructions on how to use this mode.

You can think of this as being similar to using an existing MLST/cgMLST/wgMLST scheme to define your sample's strains.

2.1.3 Fit your own model

If a database isn't available for your species, you can fit your own. This consists of three steps:

1. Sketch your genomes (see *Sketching (--create-db)*).
2. Quality control your database (see *Data quality control (--qc-db)*).
3. Fit a model (see *Fitting new models (--fit-model)*).
4. Repeat step two, until you have a model which works for your needs.

After getting a good fit, you may want to share it with others so that they can use it to assign queries. See *Distributing PopPUNK models* for advice. We would also be interested to hear from you if you'd like to add your new model to the pre-fit databases above – please contact poppunk@poppunk.net.

2.1.4 Create visualisations

A number of plots are created by default. You can also create files for further visualisation in microreact, cytoscape, grapetree and phandango. We have found that looking at the appearance of clusters on a tree is always very helpful, and would recommend this for any fit.

Older versions of PopPUNK mandated this be chosen as part of the main analysis, and then with `--generate-viz` mode. This is now run separately, after the main analysis, with `poppunk_visualise`.

See *Creating visualisations* for details on options.

2.2 Use an online interface

If available, you may want to use one of the browser-based interfaces to PopPUNK. These include beebop and pathogen.watch (*S. pneumoniae* only).

Using these interfaces requires nothing to be installed or set up, doesn't require any genome data to be shared with us, and will return interactive visualisations. If your species isn't available, or you have large batches of genomes to cluster you will likely want to use the command line interface instead.

CHAPTER

THREE

SKETCHING (--CREATE-DB)

The basis of all analysis is estimation of core and accessory genome distances between samples. PopPUNK uses genome sketching to make analysis more efficient. In previous versions we used mash, however the current version now requires pp-sketchlib.

This page details options related to sketching and distance calculation, and is relevant to both *Query assignment (poppunk_assign)* and *Fitting new models (--fit-model)*.

3.1 Overview

Any input given to `--r-files` or `--q-files` will be sketched using the following steps:

1. Run pp-sketchlib to sketch all input genomes.
2. (r-files only) Run *Data quality control (--qc-db)* on the sketches. Remove, ignore or stop, depending on `--qc-filter`.
3. (r-files only) Calculate random match chances and add to the database.
4. Save sketches in a HDF5 datbase (the .h5 file).
5. (r-files only) Calculate core and accessory distances between every pair of sketches, save in .npy and .pkl.
6. (q-files only) Calculate core and accessory distances between query and reference sketches.
7. Report any core distances greater than `--max-a-dist` (and quit, if an r-file).

To run this before *Fitting new models (--fit-model)*, use `--create-db`:

```
poppunk --create-db --output database --r-files rlist.txt --threads 8
```

As noted elsewhere, the input is file which lists your sample names and paths to their sequence data. This file has no header, is tab separated, and contains the sample name in the first column. Subsequent columns may contain paths to either assembled or raw read data (the type will automatically be inferred by checking for the presence of quality scores). Data may be gzipped or uncompressed:

```
MS1 ms1_assembled.fa
MS2 ms2_assembled.fa
SM14        SM14_1.fq.gz SM14_2.fq.gz
```

The rest of this page describes options to further control this process.

Note: Sketching considers k-mers on both the forward and reverse by default, as typically reads and assemblies are not aligned to a single strand. For genomes where input is always on the same (forward) strand, as may be the case

with single-stranded viral genomes, use the `--strand-preserved` option to ignore the reverse strand k-mers.

3.1.1 Using pp-sketchlib directly

You can use pp-sketchlib directly to create sketches, though functionality is identical to doing this through PopPUNK. You will need to run both sketch and query modes to generate the sketch database and the distance files as in `--create-db`:

```
sketchlib sketch -l rfiles.txt -o database -s 10000 -k 15,31,2 --cpus 4
sketchlib query dist database -o dists --cpus 4
```

You may want to do this if you anticipate trying different k-mer sizes, are using the databases for other purposes, or running a very large analysis where it is useful to split up the sketching and distance steps. Useful options include:

- `sketchlib query jaccard` – will output Jaccard distances at each k-mer length, rather than core and accessory distances.
- `--subset` – to only calculate distances for a subset of the genomes in the reference database.

Warning: Some options have slightly different names. See the pp-sketchlib README for full details.

3.1.2 Viewing information about a database

Use `poppunk_info` on a HDF5 file:

```
> poppunk_info --db e_coli --simple

PopPUNK database:           ecoli.h5
Sketch version:                     9314bda28ed25a60dd40f9b9e896c0b269500fec
Contains random matches:    True
Number of samples:          10287
K-mer sizes:                        15,18,21,24,27
Sketch size:                        9984
```

Sketch size is always rounded to the nearest 64.

Warning: The sketch version should match between databases you are comparing, but the program will still run with a warning if they don't. Check results carefully.

Without `--simple` to get further information for every sample (`--output` to save results to a file; `--network` to use a non-default network file):

```
Sample,Length,Missing_bases,Frequency_A,Frequency_C,Frequency_G,Frequency_T,Component_
↪label,Component_size,Node_degree
11657_5#1,4673808,2879,0.24679,0.257679,0.249401,0.24555,0,258,15
11657_5#10,4702152,4024,0.244373,0.252315,0.252617,0.249008,1,17,5
11657_5#12,5024448,5852,0.247563,0.245727,0.255955,0.24988,2,464,445
11657_5#13,5180468,5010,0.248152,0.251625,0.254893,0.244739,3,784,733
11657_5#14,5329972,13419,0.243371,0.260128,0.251946,0.244246,4,177,128
```

3.2 Choosing the right k-mer lengths

To get a sensitive estimate of accessory distance independent from core distance, a small a k-mer size as possible needs to be included in the fit. However, for longer genomes too small a k-mer size will result in biased estimates of distances as small k-mers will match at random. pp-sketchlib now includes a correction for random matches, but there is still a lower limit at which this can work. A simple formula for estimating this is:

$$r = 1 - (1 - 2 \cdot 4^{-k})^{-l}$$
$$J_r = \frac{r^2}{2r - r^2}$$

where k is the k-mer length, l is the length of the genome and J_r is the Jaccard distance expected by chance. When J_r approaches 1, estimation will begin to fail.

Note: For genomes on a single strand, the factor of two in the first formula above should be excluded.

At the other end, choosing a k which is too long will result in all k-mers mismatching. The greater the core distance π, the lower the allowable maximum.

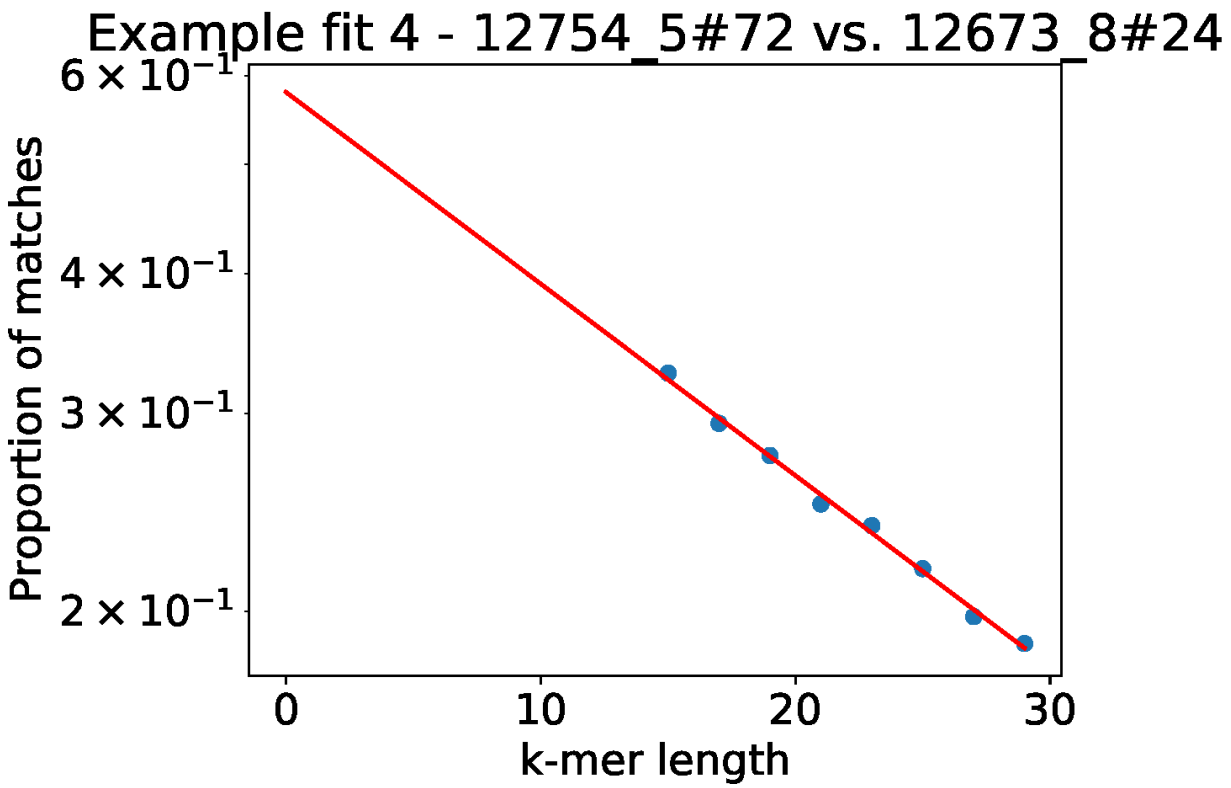
Some k-mer ranges for `--min-k` and `--max-k` we have found to work for various genomes:

Table 1: k-mer lengths by domain

Domain/pathogen	Typical l	π	min-k	max-k
Beta-coronaviruses	20kb	0.1	6	15
Bacteria	2-5Mb	~0.01-0.04	13	29
Fungi	16Mb	~0.01	15	31
Plasmodium	23Mb	0.0005	17	31

A `--k-step` of four is usually sufficient, but drop this to two or three to give the best accuracy at the expense of increased execution time.

A good model will have a straight line fit between $\log(J)$ and k. Run with the `--plot-fit` option to randomly choose a number of sample pairs to plot the relation between k-mer distances and core and accessory fits. This plot does not have to be perfectly straight, but the general trend should be followed. If you have a point at the end going off the scale, you will want to adjust your k-mer range.



3.2.1 Choosing the sketch size

The default sketch size s is 10000. Note that this is 10-fold greater than the mash default of 1000 – this is required to get sufficient resolution on π. For closely related genomes with smaller π, you may need to increase the sketch size.

As a rule of thumb, choose $s = \frac{1}{\pi}$ based on the minimum resolution in π you need to observe.

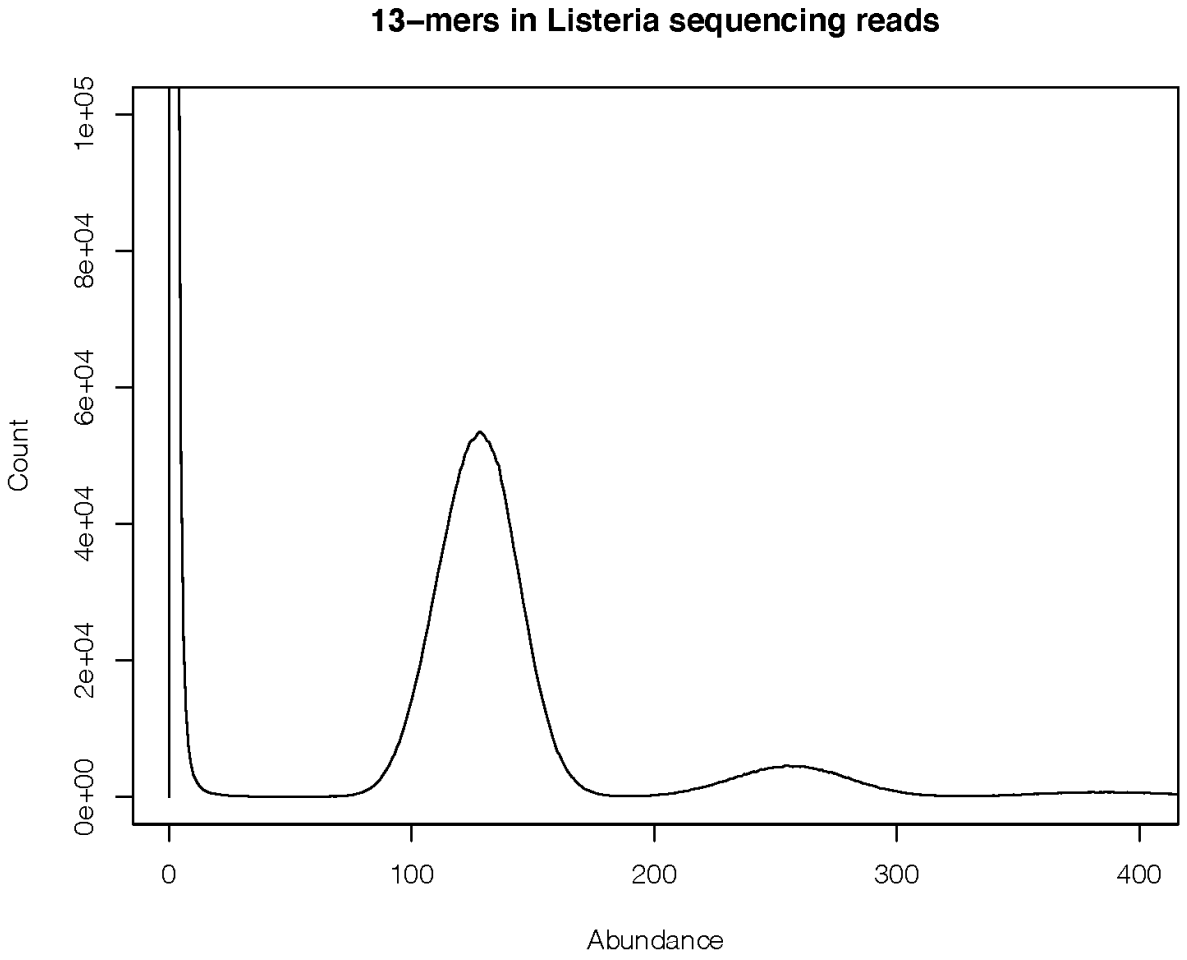
Important: Any Jaccard distances $< \frac{5}{s}$ will be ignored in the fit of core and accessory distances. This prevents spurious matches between very close sketches dominating, when a poor minimum k-mer length has been chosen.

Note that a larger sketch size will result in a linear increase in database size and distance calculation time.

3.3 Sketching from read data

You can also use sequence reads rather than assemblies as input. The main differences are that this data is typically a lot larger, and may contain false k-mers as the result of sequencing errors.

Read data is automatically detected for each input file. It may be interleaved, or given as forward and reverse reads. Low frequency k-mers, which are assumed to be the result of sequencing error, will be filtered out automatically. Use the `--min-kmer-count` option to set the minimum number of k-mers needed to be observed to include these. Most error k-mers will appear only once, but ideally set this somewhere between 1 and the coverage:



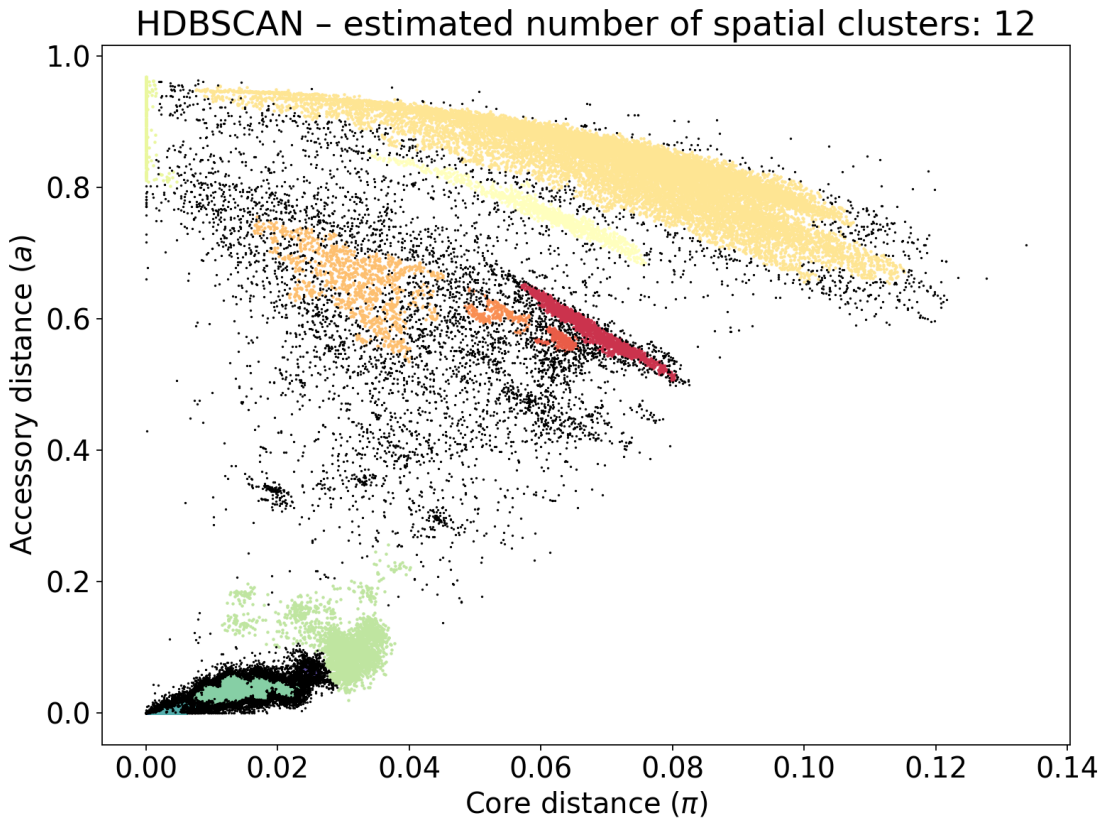
In this example the coverage is around 150x, so most correct k-mers have a frequency centred around this point (there is a second peak at twice this value, which are repeats). There is a large peak at a frequency of one, which are the error k-mers. In this example any filter between 15-75 would be appropriate.

The default filter is a probabilistic countmin filter, assuming up to 134M unique k-mers. If you expect significantly more k-mers than this, for example with longer genomes, you should add the `--exact-count` argument to use a hash table instead. This is exact, but may use more memory.

3.4 Sketching RNA viruses

Firstly, if your viral genomes are single stranded, you probably need to add the `--strand-preserved` option.

For small genomes where strong selection is present, in the example here shown with influenza genomes, the third codon bias may be so great that 6-mers (or any multiple of three) have fewer matches than 7-mers. In a mostly coding genome the third codon position across a gene is more free to mutate, as it can cause non-synonymous changes, whereas the first and second codons always cause coding changes. This can cause issues with the core-accessory regression pushing some core distances to 0:

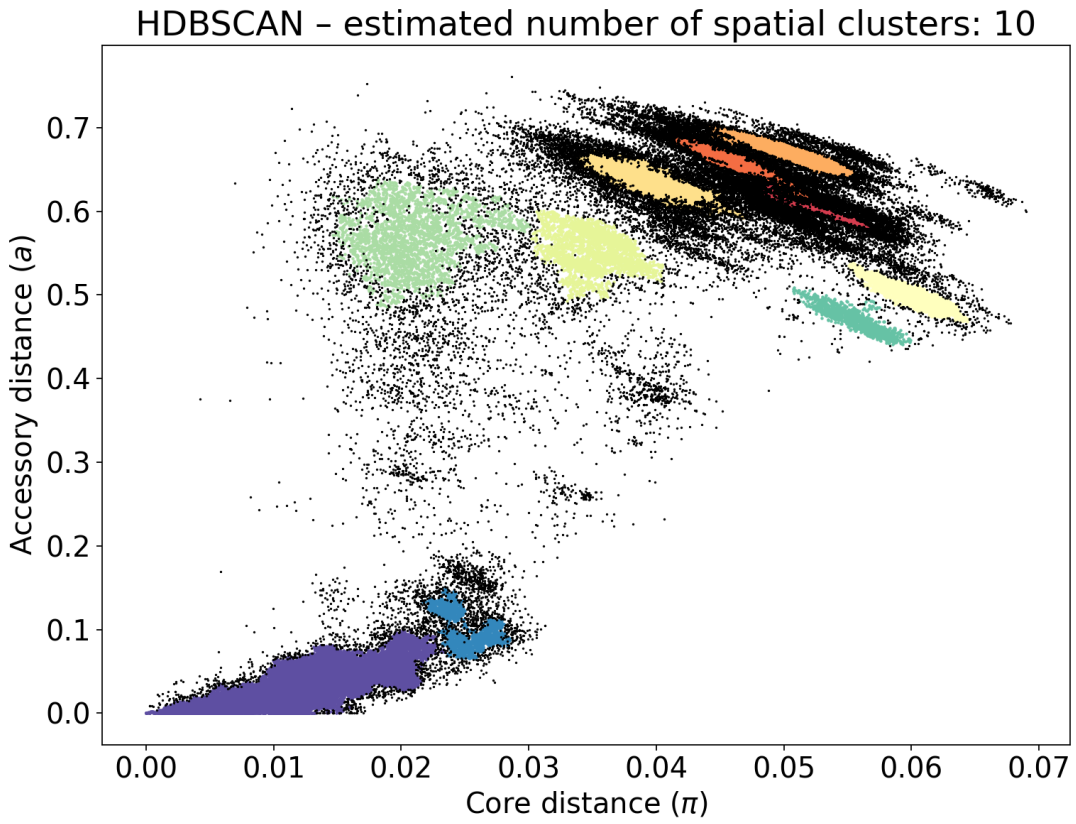


A solution to this is to use k-mers with spaced seeds, where only every third base is added to the k-mer. This prevents multiples of the codon size lining up with heavily mutated bases.

Table 2: Codon phased seeds

k-mer	dense	Phased seed
3	XXX	X–X–X
4	XXXX	X–X–X–X
5	XXXXX	X–X–X–X–X

Add the `--codon-phased` option to enable this. This fixes the above example:



Note: When using a database constructed with codon phased seeds for *Query assignment (poppunk_assign)*, codon phased seeds will automatically be turned on for the query sequences too.

3.5 GPU acceleration

There are two pieces of heavy computation that can be accelerated with the use of a CUDA-enabled GPU:

- Sketching read data `--gpu-sketch`.
- Calculating core and accessory distances `--gpu-dist`.

We assume you have a GPU of at least compute capability v7.0 (Tesla) with drivers correctly installed. You do not need the CUDA toolkit installed, as all libraries are included with the pp-sketchlib executable.

Note: You will see 'GPU' in the progress message if a GPU is successfully being used. If you see the usual CPU version your install may not have been compiled with CUDA.

Sketching read data with the GPU is a hybrid algorithm which can take advantage of CPU threads too (which are used to read and process the fastq files). You can add up to around 16 `--threads` to keep a typical consumer GPU busy. The sequence data must fit in device memory, along with a 2Gb countmin filter. The countmin filter is 134M entries wide. If you expect your reads to have more unique k-mers than this you may see an increased error rate.

Typical output will look like this:

```
Sketching 128 read sets on GPU device 0
also using 16 CPU cores
Sketching batch: 1 of 9
k = 29   (100%)
k = 29   (100%)
k = 29   (100%)
k = 29   (100%)
k = 29   (100%)
k = 29   (100%)
k = 29   (100%)
k = 29   (100%)
k = 29   (100%)
k = 29   (100%)
k = 29   (100%)
k = 29   (100%)
k = 29   (100%)
k = 29   (100%)
k = 29   (100%)
k = 29   (100%)
k = 29   (100%)
Sketching batch: 2 of 9
k = 29   (100%)
k = 29   (100%)
k = 29   (100%)
k = 29   (100%)
....
```

Calculating distances with the GPU will give slightly different results to CPU distances, but typically within 1%, which should not usually affect downstream results. The sketches, random matches and distances must fit in the device memory. Around 35k bacterial genomes uses around 10Gb of device memory, typical for a high-end consumer device. If the device memory is exceeded the calculation will automatically be split into chunks, at only slightly reduced efficiency. The amount of memory available and needed will be estimated at the start:

```
Calculating distances on GPU device 0
Estimated device memory required: 565Mb
Total device memory: 11019Mb
Free device memory: 10855Mb
Progress (GPU): 100.0%
```

Important: The GPU which is device 0 will be used by default. If you wish to target another GPU, use the `--deviceid` option. This may be important on computing clusters where you must use your job's allocated GPU.

CHAPTER

FOUR

DATA QUALITY CONTROL (--QC-DB)

PopPUNK now comes with some basic quality control options, which you should run on your sketch database made with `--create-db` by running `--qc-db` as follows:

```
poppunk --qc-db --ref-db example_db --type-isolate 12754_4_79 --length-range 2000000␣
↪3000000
```

For `poppunk_assign`, instead add `--run-qc`:

```
poppunk_assign --query queries.txt --db example_db --run-qc --max-zero-dist 1 --max-
↪merge 3
```

The following criteria are available:

- Outlying genome length (calculated during sketching, for assemblies or reads) with `--length-range` and/or `--length-sigma`.
- Too many 'N's with `--prop-n` and/or `--upper-n`.
- Outlying core or accessory distances with `--max-pi-dist` and `--max-a-dist` respectively.
- Too many zero distances with `--max-zero-dist`.

For `poppunk --create-db` only:

- Names of samples to remove (e.g. failing external QC) with `--remove-samples`.

For `poppunk_assign` only:

- Maximum number of clusters a single isolate can cause to merge with `--max-merge`.
- Betweenness of queries (not automated, just reported) with `--betweenness`.

In all cases a file will be written at `qcreport.txt` which lists the failing samples, and the reasons why they failed. Adding `--qc-keep` will only write the file and not remove failing samples. You may also add `--retain-failures` to write a separate sketch database with the failed samples.

Random match chances in PopPUNK are only calculated and added to the database after the chosen QC step. If you use `sketchlib` directly, they will be added without any automated QC.

4.1 QC of input sequences

The first QC step is applied directly to the input sequences themselves, to identify poor quality sequences.

You can change the genome length cutoff with `--length-sigma` which sets the maximum number of standard deviations from the mean, and `--length-range` which sets an absolute range of allowable sizes.

Ambiguous bases are controlled by `--prop-n` which gives the maximum percentage of Ns, and `--upper-n` which gives the absolute maximum value.

4.2 QC of pairwise distances

The second QC step uses the pairwise distances, to enable the removal of outlier samples that may not be part of the taxon being studied. This is with reference to a type isolate. The type isolate will be selected by PopPUNK, unless specified using `--type-isolate`.

By default, the maximum allowed accessory distance is 0.5 to ensure you check for contamination. However, many species do really have high accessory values above this range, in which case you should increase the value of `--max-a-dist`.

The maximum allowed core distance is also 0.5, by default. This can be altered with `--max-pi-dist`.

All sequences differing from the type isolate by distances greater than either threshold will be identified by the analysis.

Each isolate may have a proportion of distances that are exactly zero as set by `--max-zero-dist`.

4.3 QC of the network (assign only)

Finally, you may also check network properties.

Maximum number of clusters a single isolate can cause to merge is set with `--max-merge`. More than this number of links across the original clusters will result in removal of the isolate.

Betweenness of queries can be reported with `--betweenness`, which may be useful to prune the input in more complex cases. This does not cause automated removal as it's difficult to set a sensible threshold across datasets. You will therefore need to re-run and remove samples yourself.

CHAPTER

FIVE

FITTING NEW MODELS (`--FIT-MODEL`)

If you cannot find an existing model for your species in the list you will want to fit your own. This process is flexible, and there are five different models you can use depending on the population structure of your dataset.

Note: After fitting a model to a new species we would like to share it on our website, so others can use it for assigning queries. If you are open to this, please read *Distributing PopPUNK models* after this page.

5.1 Overview

First, use `poppunk --create-db` to sketch your input data and calculate distances between all samples. This is detailed in *Sketching (--create-db)*.

Then, use `poppunk --fit-model <model_name>` with one of the following model names:

- `bgmm` – Bayesian Gaussian Mixture Model. Best for small sample collections with strain-structure. Works best when distance distribution components are clearly separated.
- `dbscan` – HDBSCAN. A good general method for larger sample collections with strain-structure. Some points will always be designated as noise, so a subsequent run of model refinement may help improve the fit.
- `refine` – Model refinement. Requires a model already fitted with `bgmm` or `dbscan` and attempts to improve it by maximising the network score. Particularly useful when components overlap significantly (often due to recombination), or when the strain boundary is thought to lie somewhere within a component.
- `threshold` – Apply a given core or accessory distance threshold to define VLKCs. Useful if a cutoff threshold is already known/calculated, is estimated from a plot, or to compare a threshold between datasets or species.
- `lineage` – Lineage clustering. To find lineages within a strain (subclustering), or find clusters in a population without strain structure. Uses a simple nearest neighbour approach so is more of a heuristic. Network scores are not meaningful in this mode.

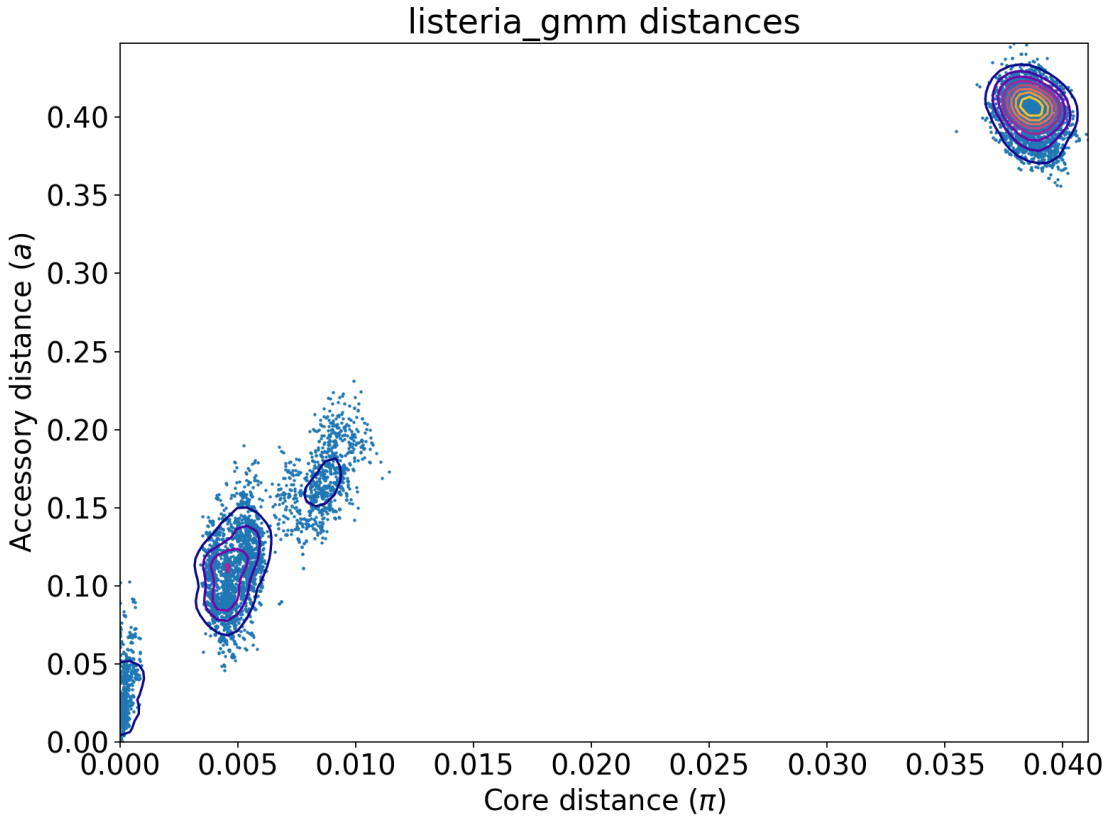
The most useful guide to deciding which model to use is the `_distanceDistribution.png` file showing the core and accessory distances. More details on each of these models is given further down this page.

A completed fit will consist of:

- A `_clusters.csv` file, which gives the VLKC (strain) for each sample in the database.
- A `_unword_clusters.csv` file, which gives an English-pronounceable name instead of a number to each VLKC.
- `_fit.npz` and `_fit.pkl` files, which contain numeric data and metadata for the fit.
- A `_graph.gt` file, which is the network defining the fit in graph-tool format.

- Some plots of the fit, which depend on the specific model used.
- A `.refs` file, which lists the samples kept as 'references' for assigning future samples (see *Distributing PopPUNK models* for more details).

This page will use 128 *Listeria monocytogenes* genomes from Kremer et al, which can be downloaded from figshare. The distribution of core and accessory distances from the `--create-db` step is as follows:



We also show some examples with 616 *Streptococcus pneumoniae* genomes, which are more complex. These genomes were collected from Massachusetts, first reported here and can be accessed here.

5.2 Common arguments

- `--ref-db`: the output prefix used with `--create-db` i.e. the directory where the .h5 file is located
- `--output`: where to save the model. If not specified this defaults to `ref-db`.
- `--overwrite`: overwrite any existing files in the output directory.
- `--external-clustering`: any additional labels to add to the cluster output.
- `--graph-weights`: save the edges weights in the network as their Euclidean core-accessory distances, rather than as 0 or 1 (useful for visualising the network).

External clusters may be other cluster names, such as serotype, sequence type, cgMLST etc. VLKCs are mapped as one-to-many, so that each strain is labelled with all of the clusters any of its members is assigned to in this file. This input file must be comma separated, one sample per line, with the sample name as the first column, and other clusters

as subsequent columns. A header line with 'sample' and the names of other cluster types is required. Output is to `output/output_external_clusters.csv`.

5.3 How good is my fit?

We have found the best way to assess this is to use *Creating visualisations* on your output and look at your assigned VLKCs against a tree, to determine whether they have the specificity required.

You can also compare models with their network score, and whether the output plots look as expected. Typically the key thing is that **your spatial component nearest the origin is accurate**. More detail is given for each model below.

5.3.1 Interpreting the network summary

All fits will output a network summary which looks similar to this:

```
Network summary:
    Components                          59
    Density                             0.0531
    Transitivity                        0.9966
    Mean betweenness                    0.0331
    Weighted-mean betweenness           0.0454
    Score                               0.9438
    Score (w/ betweenness)              0.9126
    Score (w/ weighted-betweenness)     0.9009
```

- Components are the number of VLKCs (strains) found using this model.
- Density is the proportion of distances assigned as 'within-strain'. Generally smaller is better as this gives more specific clusters, but too close to zero may be an over-specific model.
- Transitivity measures whether every member of each strain is connected to every other member. Closer to 1 is better, but this can be achieved with very loose fits.
- Score synthesises the above as $(1 - \text{density}) * \text{transitivity}$, which gives a single number between 0 (bad) and 1 (good) which in many cases is at a maximum when it accurately describes strains in the data.
- Two further scores for larger networks. See *Alternative network scores* for more information on these.

5.4 bgmm

This mode fits a Bayesian Gaussian mixture model to the core and accessory distances. With few points, methods such as DBSCAN may struggle to find clusters due to the sparsity, whereas a BGMM can often find a good fit. A further advantage is that the equation for the posterior is known, so all points will have an assignment and a non-linear boundary found exactly.

However, when there are a very large number of points the likelihood has a tendency to totally override the prior in the estimated posterior, meaning many overlapping components may be fitted, which may give poor clusters, and is less robust to adding more data. It is possible for this mode to fail to converge, but it is more likely to produce a bad fit in difficult cases.

The key parameter to specify is the maximum number of components `--K`. You should choose a number based on the number of components you can see on your distance plot. This may be automatically reduced if there is insufficent

evidence for this many components. As a rule of thumb, if you have under 150 samples or under 1000 samples and clear components then this mode should give a good fit.

A better network score is evidence of a better fit, but the output files should also be used to judge this. With the test dataset, four components are visible:

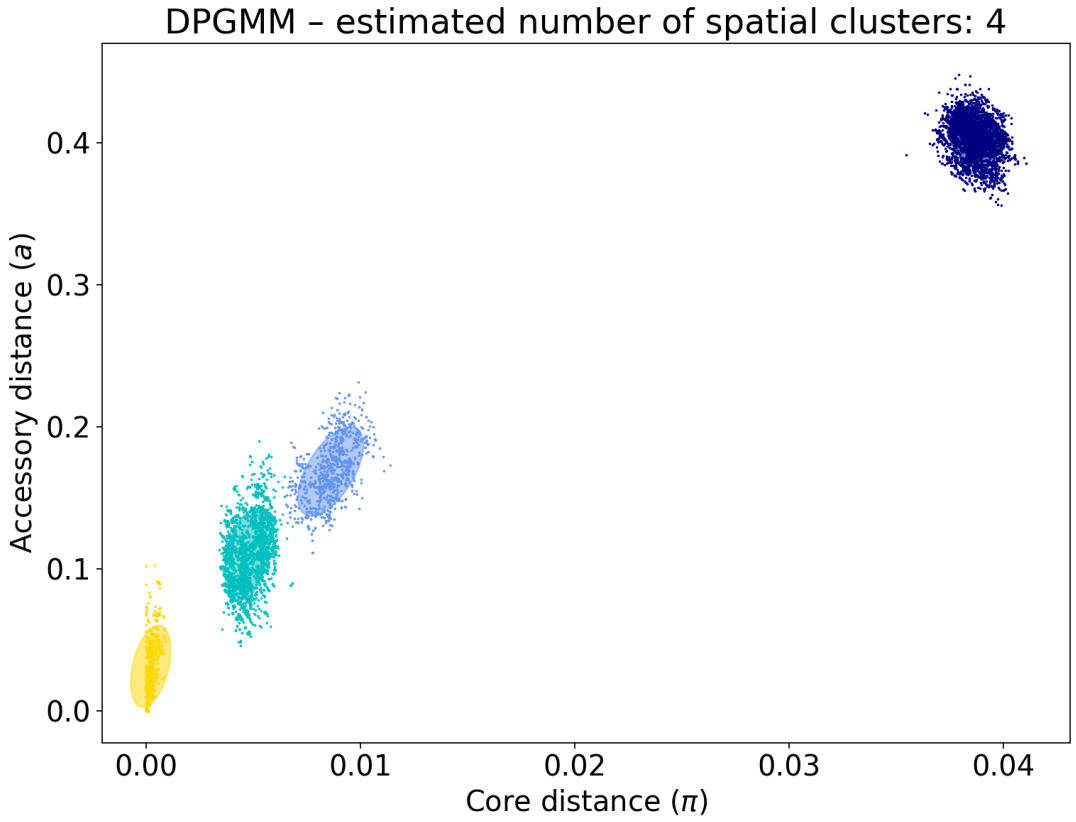
```
poppunk --fit-model bgmm --ref-db listeria --K 4
PopPUNK (POPulation Partitioning Using Nucleotide Kmers)
    (with backend: sketchlib v1.6.0
    sketchlib: /Users/jlees/miniconda3/envs/pp-py38/lib/python3.8/site-packages/pp_
↪sketchlib.cpython-38-darwin.so)

Graph-tools OpenMP parallelisation enabled: with 1 threads
Mode: Fitting bgmm model to reference database

Fit summary:
    Avg. entropy of assignment      0.0042
    Number of components used       4

Scaled component means:
    [0.9415286  0.90320047]
    [0.11542755 0.24570244]
    [0.20966101 0.37694884]
    [0.00527421 0.07043826]

Network summary:
    Components      31
    Density 0.0897
    Transitivity    1.0000
    Score   0.9103
Removing 97 sequences

Done
```

In the output to the terminal:

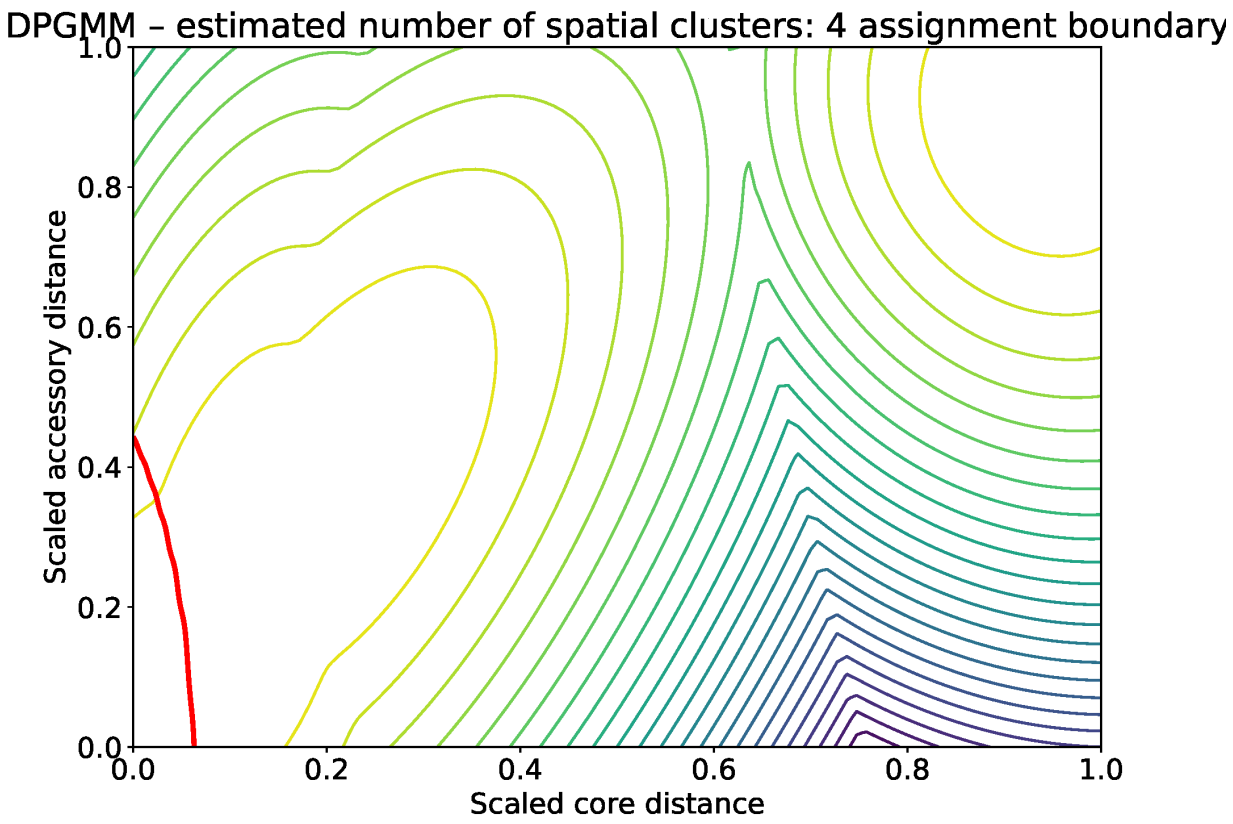
- The average entropy of assignment is a measure of the certainty of assignment of each point. Lower is better. Higher values may indicate overlapping components, perhaps due to high amounts of recombination between strains.
- Number of components used is how many components from `K` were actually used in the spatial fit. This is usually equal to `K`, but may be reduced in small datasets.
- Scaled component means are the centres of the fitted components in the model, where the core and accessory distances have been rescaled between 0 and 1. These can be used with *Using fit refinement when mixture model totally fails*.

The fit actually just uses the component closest to the origin – any distances assigned to this component are within-strain. This is the most important part of the fit in this mode.

You can see that this gives a good network score, and fits the data well:



The position of the boundary is also produced (in red), along with contours of the fitted mixture components:



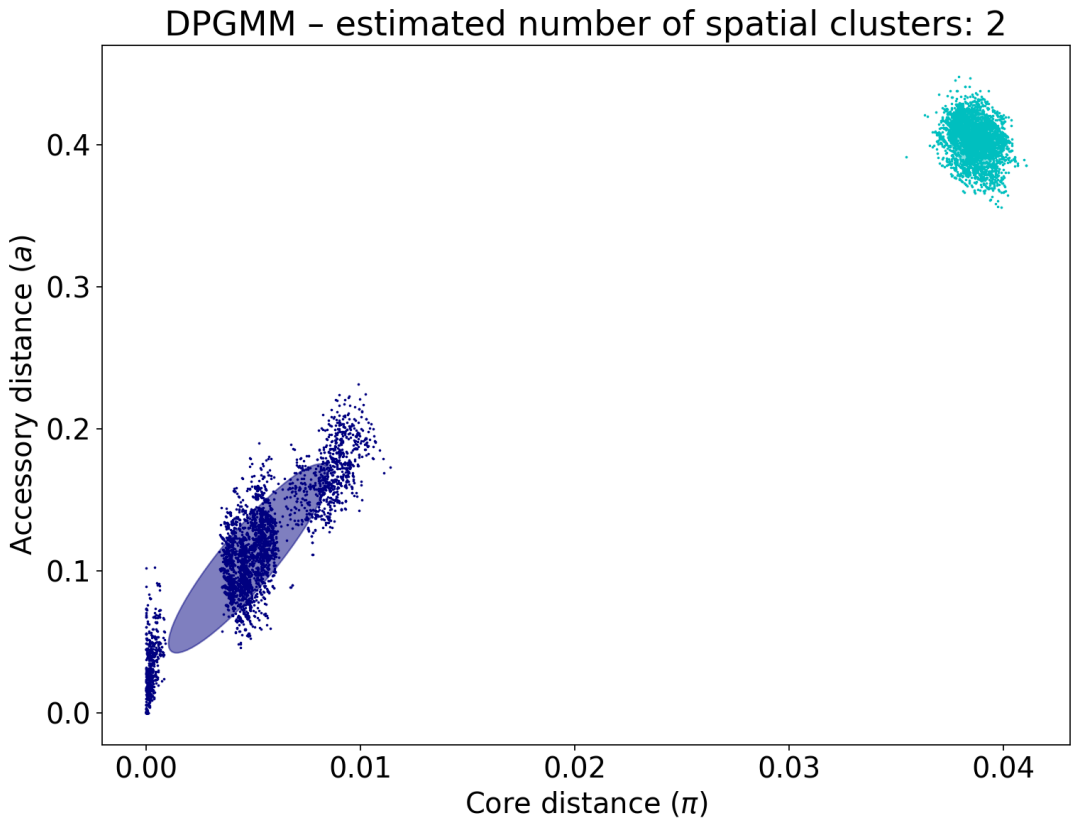
If you make K too low, some components will be merged, resulting in a less-specific fit with fewer clusters, that do not fully delineate all of the strains (in this case just finding the two main lineages of *Listeria* in this data):

```
poppunk --fit-model bgmm --ref-db listeria --K 2
PopPUNK (POPulation Partitioning Using Nucleotide Kmers)
    (with backend: sketchlib v1.6.0
    sketchlib: /Users/jlees/miniconda3/envs/pp-py38/lib/python3.8/site-packages/pp_
↪sketchlib.cpython-38-darwin.so)

Graph-tools OpenMP parallelisation enabled: with 1 threads
Mode: Fitting bgmm model to reference database

Fit summary:
    Avg. entropy of assignment      0.0007
    Number of components used       2

Scaled component means:
    [0.11627304 0.2432584 ]
    [0.9415286  0.90320047]

Network summary:
    Components      2
    Density 0.5405
    Transitivity    1.0000
```

(continues on next page)

(continued from previous page)

```
    Score   0.4595
Removing 126 sequences

Done
```



Too many components in a small dataset are automatically reduced to an appropriate number, obtaining the same good fit as above:

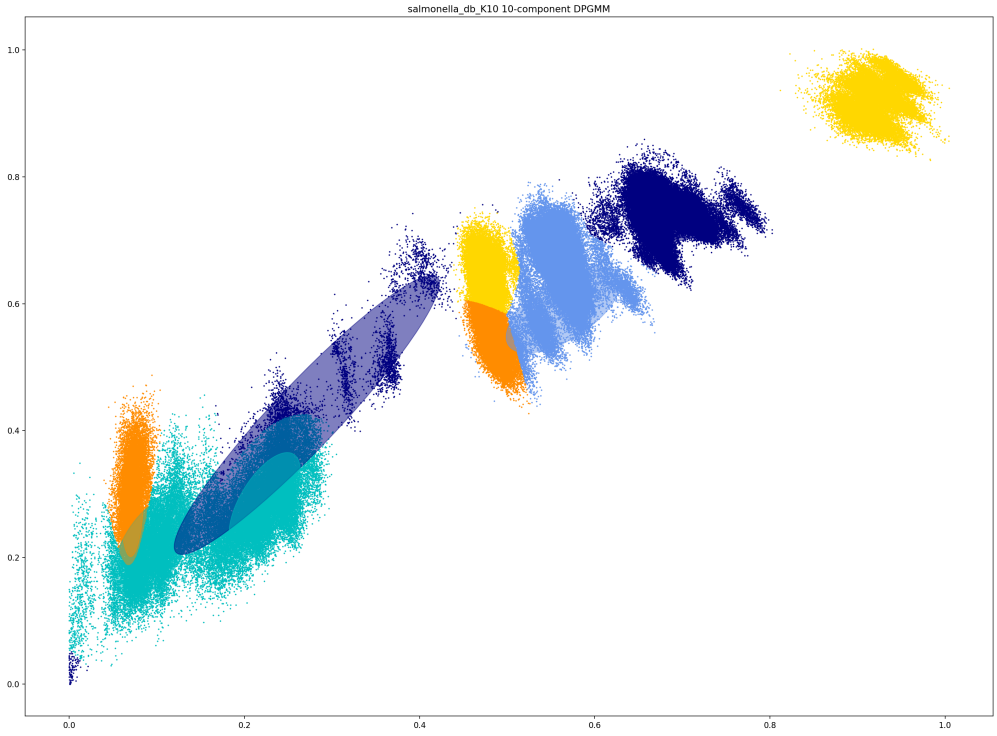
```
poppunk --fit-model bgmm --ref-db listeria --K 10
PopPUNK (POPulation Partitioning Using Nucleotide Kmers)
    (with backend: sketchlib v1.6.0
     sketchlib: /Users/jlees/miniconda3/envs/pp-py38/lib/python3.8/site-packages/pp_
↪sketchlib.cpython-38-darwin.so)

Graph-tools OpenMP parallelisation enabled: with 1 threads
Mode: Fitting bgmm model to reference database

Fit summary:
    Avg. entropy of assignment      0.3195
    Number of components used       4

Scaled component means:
    [0.9415286  0.90320047]
```

(continues on next page)

(continued from previous page)

```
    [3.72458739e-07 4.73196248e-07]
    [0.00527421 0.07043826]
    [0.20966682 0.37695524]
    [0.11542849 0.2457043 ]
    [1.68940242e-11 2.14632815e-11]
    [7.66987488e-16 9.74431443e-16]
    [3.48211781e-20 4.42391191e-20]
    [1.58087904e-24 2.00845290e-24]
    [7.17717973e-29 9.11836205e-29]

Network summary:
    Components      31
    Density 0.0897
    Transitivity    1.0000
    Score   0.9103
Removing 97 sequences

Done
```

In a dataset with more points, and less clear components, too many components can lead to a bad fit:



This is clearly a poor fit. The real issue is that the component whose mean is nearest the origin is unclear, and doesn't include all of the smallest distances.

5.5 dbscan

This mode uses HDBSCAN to find clusters in the core and accessory distances. This is a versatile clustering algorithm capable of finding non-linear structure in the data, and can represent irregularly shaped components well. Possible drawbacks are that a fit cannot always be found (this can happen for small datasets with sparse points, or for datasets without much structure in the core and accessory), and that some points are classified as 'noise' so not all of their edges are included in the network (these are the small black points).

Warning: HDBSCAN models are not backwards compatible from sklearn v1.0 onwards. We would recommend using at least this version. Even better would be to then run model refinement (*refine*) to get a simpler and faster model for onward query assignment.

dbscan usually needs little modification to run:

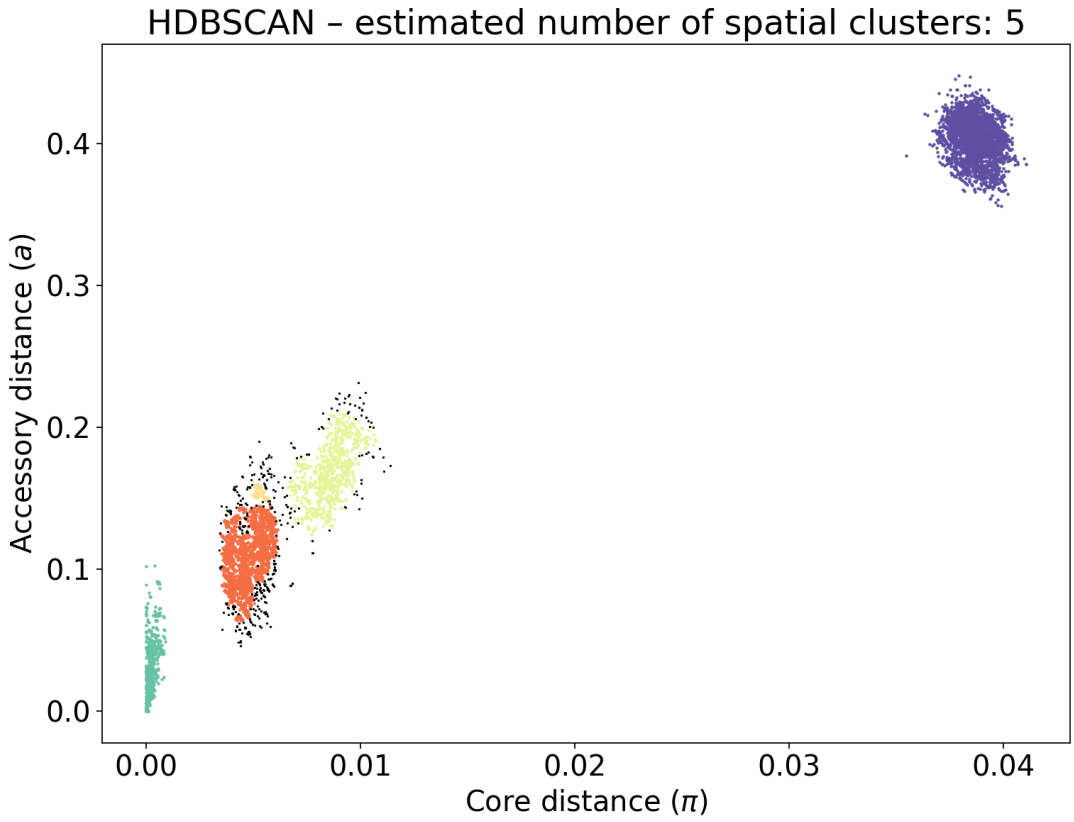
```
poppunk --fit-model dbscan --ref-db listeria
PopPUNK (POPulation Partitioning Using Nucleotide Kmers)
    (with backend: sketchlib v1.6.0
     sketchlib: /Users/jlees/miniconda3/envs/pp-py38/lib/python3.8/site-packages/pp_
↪sketchlib.cpython-38-darwin.so)

Graph-tools OpenMP parallelisation enabled: with 1 threads
Mode: Fitting dbscan model to reference database

Fit summary:
    Number of clusters      5
    Number of datapoints    8128
    Number of assignments   7804

Scaled component means
    [0.94155383 0.90322459]
    [0.00527493 0.07044794]
    [0.20945986 0.37491995]
    [0.12876077 0.34294888]
    [0.11413982 0.24224743]

Network summary:
    Components      31
    Density 0.0897
    Transitivity    1.0000
    Score   0.9103
Removing 97 sequences

Done
```

In the output to the terminal:

- The number of clusters is the number of spatial components found in the data.
- Number of datapoints is the number of points used (all-vs-all distances), which may have been subsampled from the maximum.
- Number of assignments is the number of points assign to one of the spatial components, so excluding noise points.

- Scaled component means are the centres of the fitted components in the model, where the core and accessory distances have been rescaled between 0 and 1. These can be used with *Using fit refinement when mixture model totally fails*.

The fit actually just uses the component closest to the origin – any distances assigned to this component are within-strain. This is the most important part of the fit in this mode. In this case the identification of this component is identical to the bgmm fit, so they produce the same strains. Note there is a small yellow cluster which is poorly defined, but as it does not impact the within-strain cluster the fit is unaffected:



You can alter the fit with `--D`, which sets a maximum number of clusters, and `--min-cluster-prop` which sets the minimum number of points a cluster can have (as a proportion of 'Number of datapoints). If the means of both of the core and accessory are not strictly increasing between the within-strain and next further component, the clustering fails. In this case the minimum number of samples per cluster is halved, and the fit is tried again. If this goes below ten, no fit can be found.

Increasing `--min-cluster-prop` or decreasing `--D` gets rid of the errant cluster above:

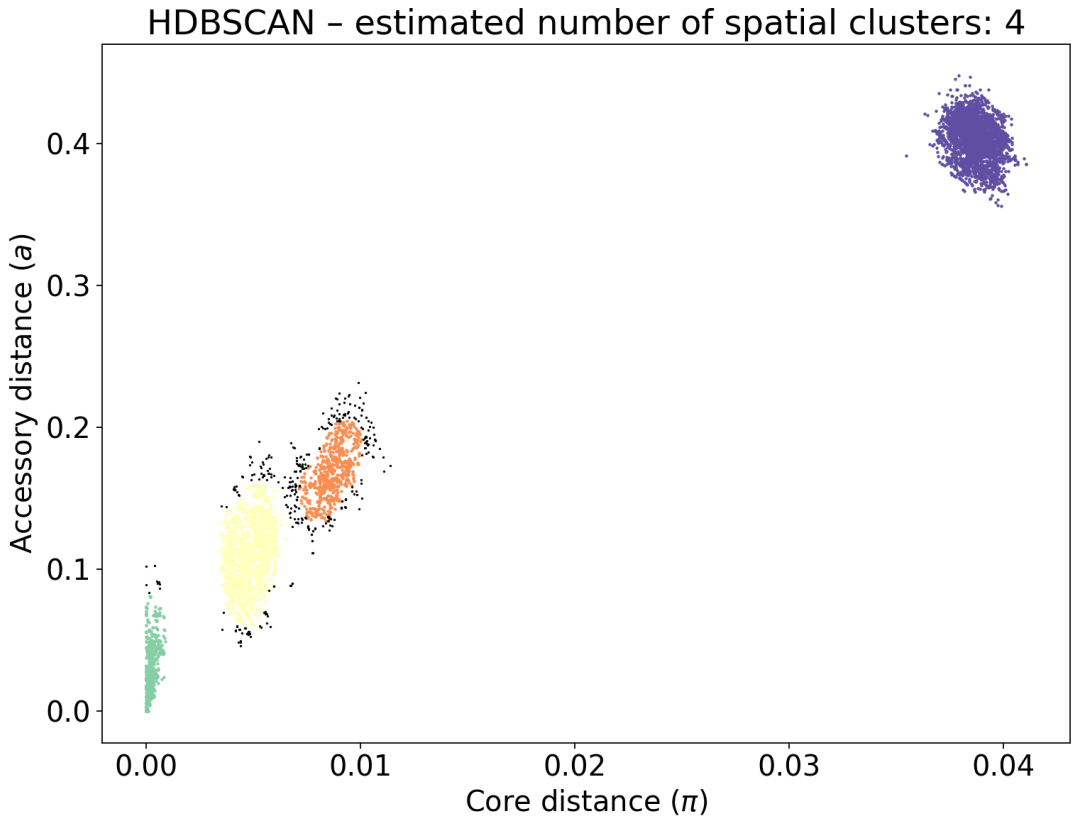
```
poppunk --fit-model dbscan --ref-db listeria --min-cluster-prop 0.01
PopPUNK (POPulation Partitioning Using Nucleotide Kmers)
    (with backend: sketchlib v1.6.0
    sketchlib: /Users/jlees/miniconda3/envs/pp-py38/lib/python3.8/site-packages/pp_
↪sketchlib.cpython-38-darwin.so)

Graph-tools OpenMP parallelisation enabled: with 1 threads
Mode: Fitting dbscan model to reference database
```

(continues on next page)

(continued from previous page)

```
Fit summary:
    Number of clusters      4
    Number of datapoints    8128
    Number of assignments   7805

Scaled component means
    [0.94155383 0.90322459]
    [0.00522549 0.06876396]
    [0.11515678 0.24488282]
    [0.21152104 0.37635505]

Network summary:
    Components      31
    Density 0.0886
    Transitivity    0.9953
    Score   0.9071
Removing 95 sequences

Done
```

But note that a few more noise points are generated, and fewer samples are removed when pruning cliques:



Setting either `--min-cluster-prop` or `--D` too low can cause the fit to fail:

```
poppunk --fit-model dbscan --ref-db listeria --min-cluster-prop 0.05
PopPUNK (POPulation Partitioning Using Nucleotide Kmers)
    (with backend: sketchlib v1.6.0
    sketchlib: /Users/jlees/miniconda3/envs/pp-py38/lib/python3.8/site-packages/pp_
↪sketchlib.cpython-38-darwin.so)

Graph-tools OpenMP parallelisation enabled: with 1 threads
Mode: Fitting dbscan model to reference database

Failed to find distinct clusters in this dataset
```

5.6 refine

Model refinement is slightly different: it takes a model already fitted by *bgmm* or *dbscan* and tries to improve it by optimising the network score. This starts with a parallelised global optimisation step, followed by a serial local optimisation step (which can be turned off with `--no-local`). Use of multiple `--cpus` is effective for these model fits.

Briefly:

- A line between the within- and between-strain means is constructed
- The point on this line where samples go from being assigned as within-strain to between-strain is used as the starting point
- A line normal to the first line, passing through this point is constructed. The triangle formed by this line and the x- and y-axes is now the decision boundary. Points within this line are within-strain.
- The starting point is shifted by a distance along the first line, and a new decision boundary formed in the same way. The network is reconstructed.
- The shift of the starting point is optimised, as judged by the network score. First globally by a grid search, then locally near the global optimum.

Applying this to the *Listeria* DBSCAN fit (noting that you may specify a separate directory to load the model from with `--model-dir`, if multiple model fits are available):

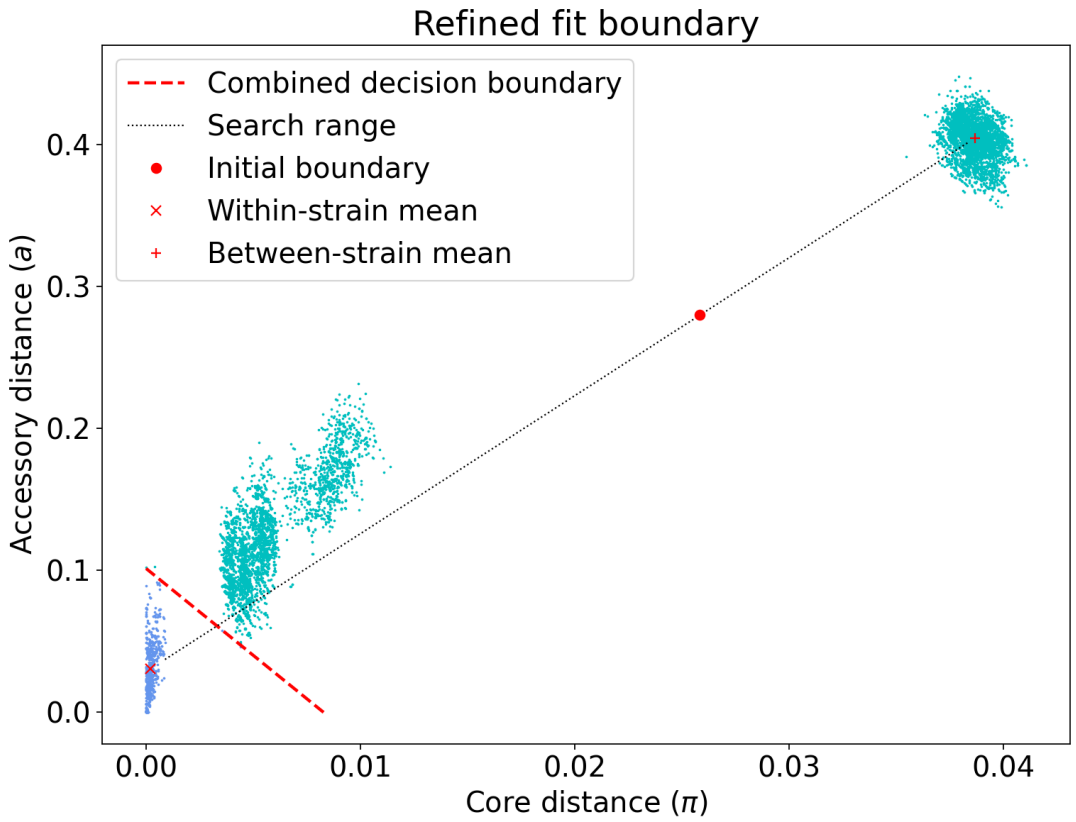
```
poppunk --fit-model refine --ref-db listeria --model-dir dbscan
PopPUNK (POPulation Partitioning Using Nucleotide Kmers)
    (with backend: sketchlib v1.6.0
    sketchlib: /Users/jlees/miniconda3/envs/pp-py38/lib/python3.8/site-packages/pp_
↪sketchlib.cpython-38-darwin.so)

Graph-tools OpenMP parallelisation enabled: with 1 threads
Mode: Fitting refine model to reference database

Loading DBSCAN model
Loaded previous model of type: dbscan
Initial model-based network construction based on DBSCAN fit
Initial boundary based network construction
Decision boundary starts at (0.63,0.62)
Trying to optimise score globally
Trying to optimise score locally

Optimization terminated successfully;
```

(continues on next page)

(continued from previous page)

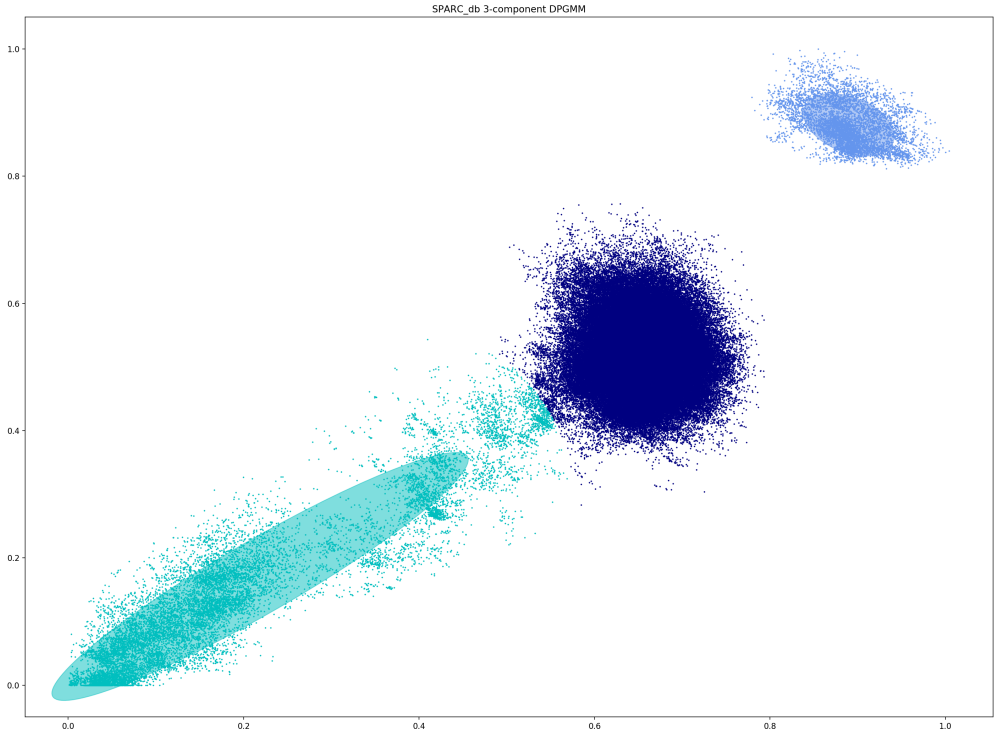
```
The returned value satisfies the termination criteria
(using xtol =  1e-05 )
Network summary:
    Components      29
    Density 0.0897
    Transitivity    0.9984
    Score   0.9088
Removing 93 sequences

Done
```

As this model was already well fitted, this doesn't change much, and finds very similar VLKC assignments (though noise points are eliminated):



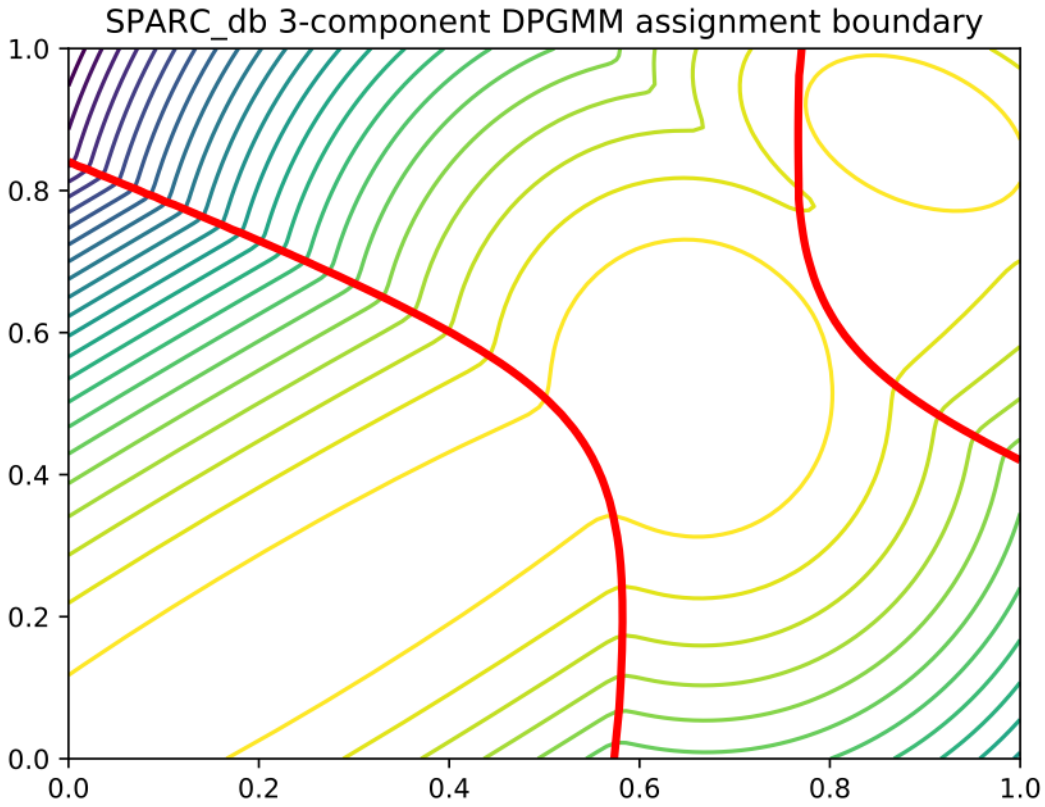
The default is to search along the entire range between the within- and between-strain clusters, but sometimes this can include undesired optima, particularly near the origin. To exclude these, use `--pos-shift` to alter the distance between the end of the search range and the origin and `--neg-shift` for the start of the search range.

This mode is more useful in species with a relatively high recombination rate the distinction between the within- and between-strain distributions may be blurred in core and accessory space. This does not give the mixture model enough information to draw a good boundary as the likelihood is very flat in this region:



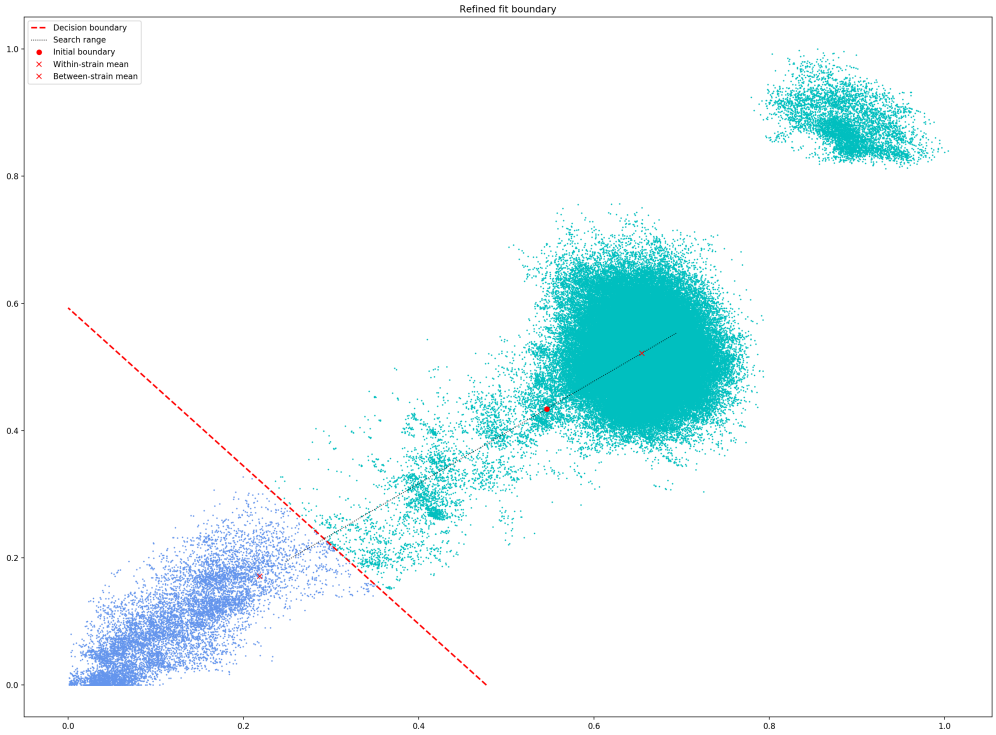
Although the score of this fit looks ok (0.904), inspection of the network and microreact reveals that it is too liberal and VLKCs/strains have been merged. This is because some of the blur between the origin and the central distribution has been included, and connected clusters together erroneously.

The likelihood of the model fit and the decision boundary looks like this:

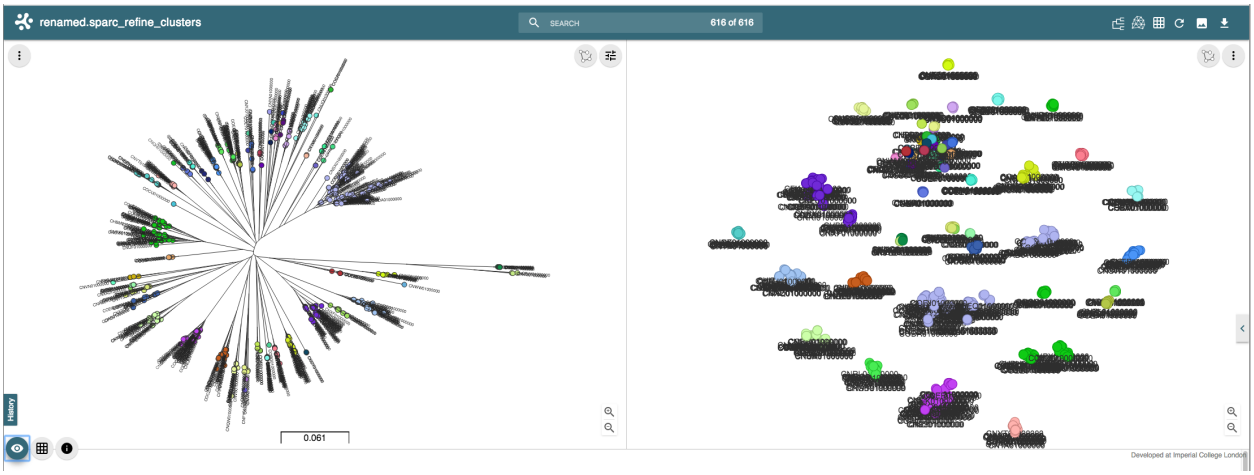


Using the core and accessory distributions alone does not give much information about exactly where to put the boundary, and the only way to fix this would be by specifying strong priors on the weights of the distributions. Fortunately the network properties give information in the region, and we can use `--refine-fit` to tweak the existing fit and pick a better boundary.

Here is the refined fit, which has a score of 0.939, and 62 rather than 32 components:



Which, looking at the microreact output, is much better:



5.6.1 Alternative network scores

Two additional network scores are now available using node betweenness. We have observed that in some refined fits to large datasets, some clusters are merged with a single high-stress edge at a relatively large distance. These scores aim to create a more conservative boundary that splits these clusters.

For these scores:

- The network is split into S connected components (the strains) each of size w_i
- For each component with at least four nodes, the betweenness of the nodes are calculated
- Each component is summarised by the maximum betweenness of any member node $b_i^{\max}$

$$\text{score}_1 = \text{score}_0 \cdot (1 - \frac{1}{S}\sum_{i=1}^{S} b_i^{\max})$$

$$\text{score}_2 = \text{score}_0 \cdot (1 - \frac{1}{S \cdot \Sigma w_i}\sum_{i=1}^{S} [b_i^{\max} \cdot w_i])$$

Score 1 is printed as score (w/ betweenness) and score 2 as score (w/ weighted-betweenness). Use `--score-idx` with 0 (default), 1 (betweenness) or 2 (weighted-betweenness) to choose which score to optimise in refine mode. The default is the original score 0. Note that scores 1 and 2 may take longer to compute due to the betweenness calculation, though this can take advantage of multiple `--threads`.

5.6.2 Unconstrained (two-dimensional) optimisation

In the default mode described above, the boundary gradient is set from the identified means in the input model, and the position of the intercept is optimised (one-dimensional optimisation).

In cases where the gradient of the boundary is not well set by the two means in the plot, you can optimise both the intercept and the gradient by adding the `--unconstrained` option (which is incompatible with `--indiv-refine`). This will perform a global search of 20 x 20 (400 total) x- and y-intercept positions, followed by a 1D local search to further optimise the intercept (unless `--no-local` is added).

As this calculates the boundary at ten times as many positions, it is generally expected to take ten times longer. However, you can effectively parallelise this with up to 20 `--threads`:

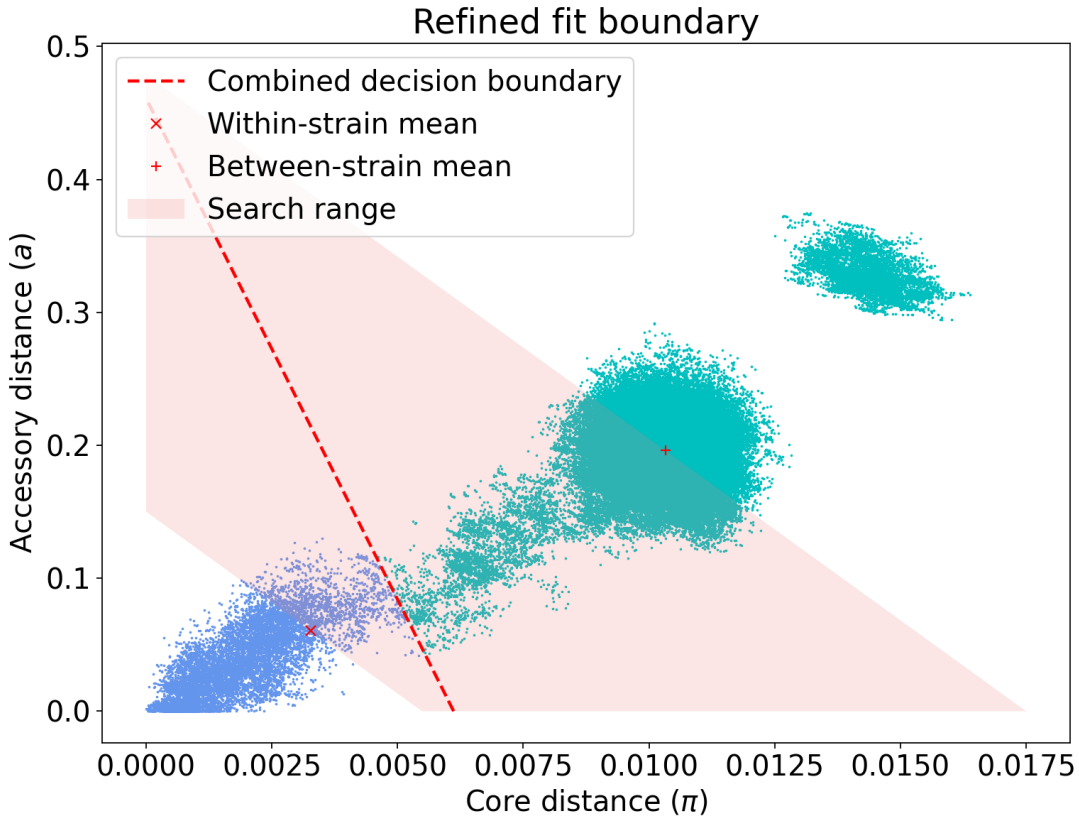
```
poppunk --fit-model refine --ref-db listeria --model-dir dbscan --unconstrained --
↪threads 4
PopPUNK (POPulation Partitioning Using Nucleotide Kmers)
    (with backend: sketchlib v1.6.2
    sketchlib: /Users/jlees/Documents/Imperial/pp-sketchlib/build/lib.macosx-10.9-x86_64-
↪3.8/pp_sketchlib.cpython-38-darwin.so)

Graph-tools OpenMP parallelisation enabled: with 4 threads
Mode: Fitting refine model to reference database

Loading BGMM 2D Gaussian model
Loaded previous model of type: bgmm
Initial model-based network construction based on Gaussian fit
Initial boundary based network construction
Decision boundary starts at (0.52,0.43)
Trying to optimise score globally
Trying to optimise score locally
```

(continues on next page)

(continued from previous page)

```
Optimization terminated successfully;
The returned value satisfies the termination criteria
(using xtol =  1e-05 )
Network summary:
    Components                          59
    Density                             0.0531
    Transitivity                        0.9966
    Mean betweenness                    0.0331
    Weighted-mean betweenness           0.0454
    Score                               0.9438
    Score (w/ betweenness)              0.9126
    Score (w/ weighted-betweenness)     0.9009
Removing 545 sequences

Done
```

Which gives a slightly higher network score, though overall similar clusters:



This is because the gradient from the 1D optimisation was well set. Unconstrained optimisation can be useful with clusters which aren't parallel to the line that connects them. This is an example in *E. coli*:

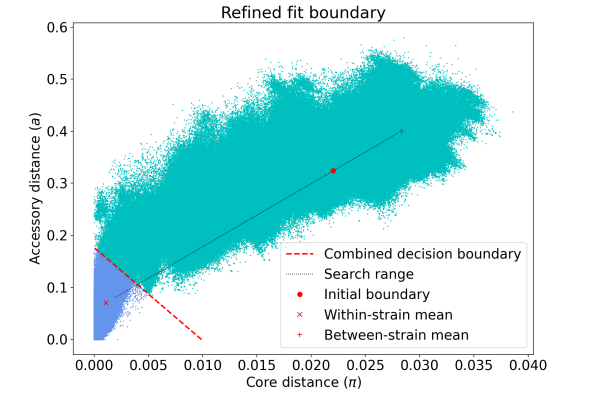


Fig. 1: 1D refine fit between DBSCAN cluster centroids

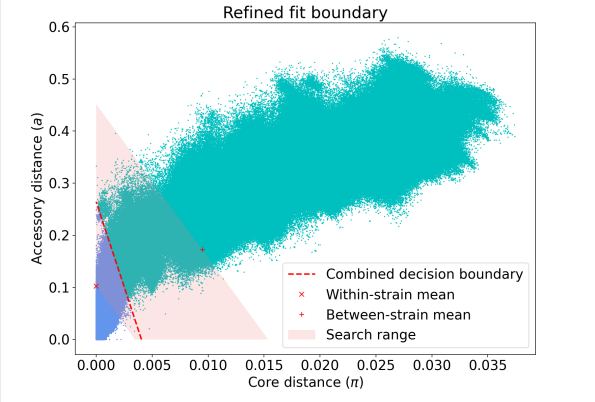


Fig. 2: Unconstrained 2D fit over a greater range

The search range will always be defined by a trapezium in light red – bounded by the two axes, and two lines passing through the means which are normal to the line which connects the means.

5.6.3 Using fit refinement when mixture model totally fails

If the mixture model does not give any sort of reasonable fit to the points, you can manually provide a file with `--manual-start` to give the starting parameters to `--refine-fit` mode. The format of this file is as follows:

```
start 0,0
end 0.5,0.6
scaled True
```

A key, followed by its value (space separated).

`start` and `end` define the points (x,y) to draw the line between. These define the two red points (and therefore the search range) in the output plot.

`scaled` defines whether these are on the [0,1] scale. If these have been set using means from the terminal output this should be `True`. Otherwise, if you have set them based on the plot (unscaled space), set to `False`.

5.6.4 Using core/accessory only

In some cases, such as analysis within a lineage, it may be desirable to use only core or accessory distances to classify further queries. This can be achieved by adding the `--indiv-refine both` option, which will allow these boundaries to be placed independently, allowing the best fit in each case:

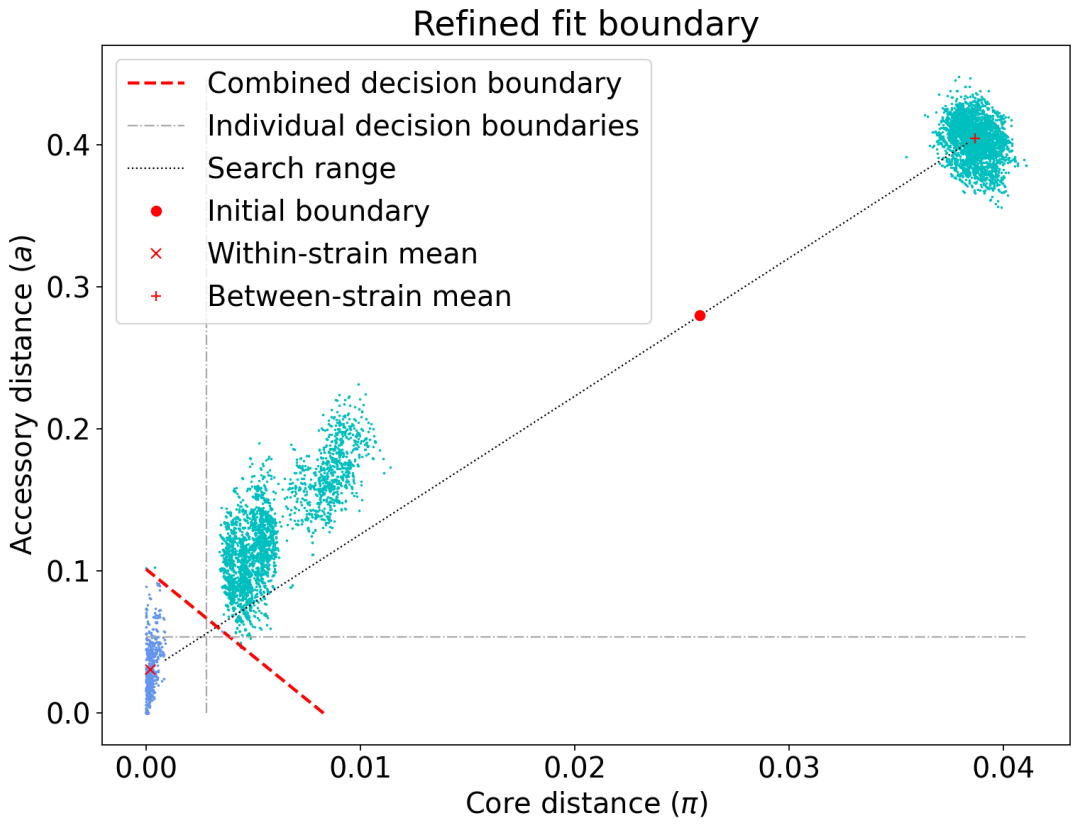
```
poppunk --fit-model refine --ref-db listeria --model-dir dbscan --indiv-refine both
PopPUNK (POPulation Partitioning Using Nucleotide Kmers)
    (with backend: sketchlib v1.6.0
    sketchlib: /Users/jlees/miniconda3/envs/pp-py38/lib/python3.8/site-packages/pp_
↪sketchlib.cpython-38-darwin.so)

Graph-tools OpenMP parallelisation enabled: with 1 threads
Mode: Fitting refine model to reference database
```

(continues on next page)

(continued from previous page)

```
Loading DBSCAN model
Loaded previous model of type: dbscan
Initial model-based network construction based on DBSCAN fit
Initial boundary based network construction
Decision boundary starts at (0.63,0.62)
Trying to optimise score globally
Trying to optimise score locally

Optimization terminated successfully;
The returned value satisfies the termination criteria
(using xtol =  1e-05 )
Refining core and accessory separately
Initial boundary based network construction
Decision boundary starts at (0.63,0.62)
Trying to optimise score globally
Trying to optimise score locally

Optimization terminated successfully;
The returned value satisfies the termination criteria
(using xtol =  1e-05 )
Initial boundary based network construction
Decision boundary starts at (0.63,0.62)
Trying to optimise score globally
Trying to optimise score locally

Optimization terminated successfully;
The returned value satisfies the termination criteria
(using xtol =  1e-05 )
Network summary:
    Components      29
    Density 0.0897
    Transitivity    0.9984
    Score   0.9088
Network summary:
    Components      31
    Density 0.0897
    Transitivity    1.0000
    Score   0.9103
Network summary:
    Components      31
    Density 0.0808
    Transitivity    0.9862
    Score   0.9064
Removing 93 sequences

Done
```

There are three different networks, and the core and accessory boundaries will also be shown on the _refined_fit.png plot as dashed gray lines:



To use one of these for your saved model, rerun, but instead setting `--indiv-refine core` or `--indiv-refine accessory`.

5.6.5 Running with multiple boundary positions

To create clusters at equally spaced positions across the refinement range, add the `--multi-boundary <n>` argument, with the number of positions specified by <n>. This will create up to <n> sets of clusters, with boundaries equally spaced between the origin and the refined boundary position.

Trivial cluster sets, where every sample is in its own cluster, will be excluded, so the final number of clusters may be less than <n>.

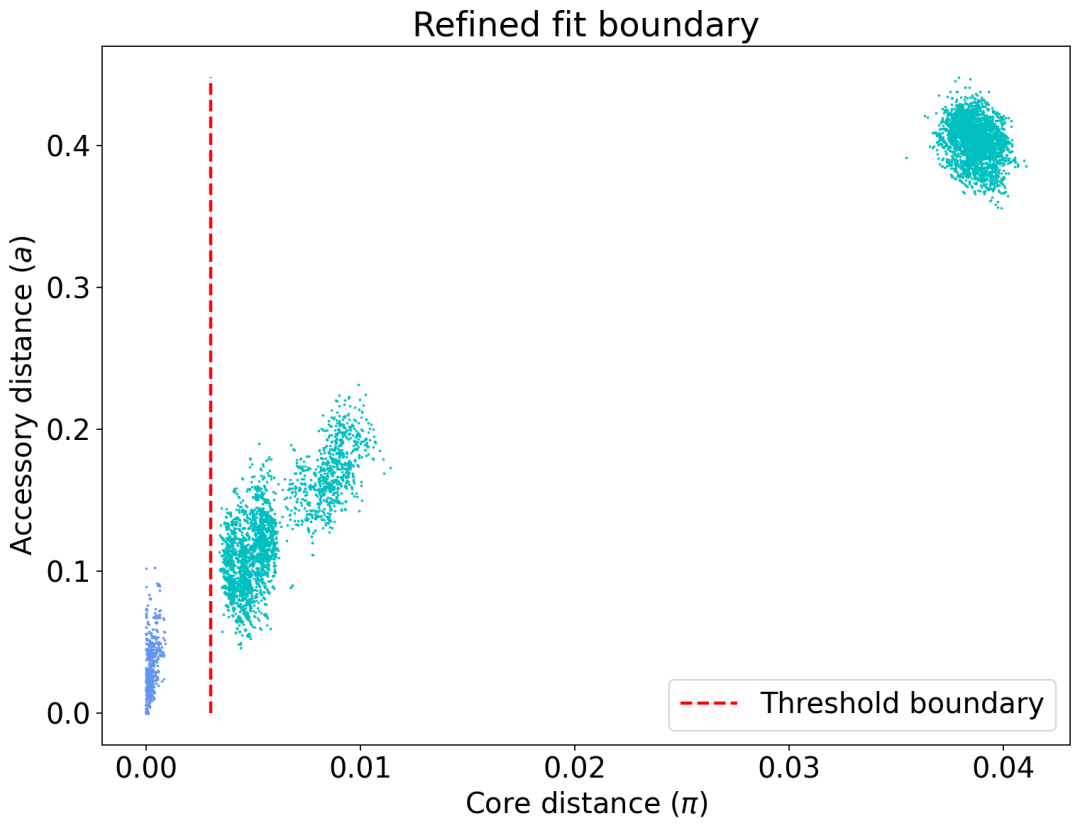
For a use of these cluster sets, see the *Iterative PopPUNK* section.

5.7 threshold

In this mode no model is fitted. You provide the threshold at which within- and between-strain distances is drawn. This can be useful if `refine` cannot find a boundary due to a poorly performing network score, but one can clearly be seen from the plot. It may also be useful to compare with other fits from related species where a boundary has been identified using one of the fitting procedures.

Currently only a core-distance boundary is supported (if you would like an accessory or combined mode available, please raise an issue). Provide the cutoff with `--threshold`:

```
poppunk --fit-model threshold --ref-db listeria --threshold 0.003
PopPUNK (POPulation Partitioning Using Nucleotide Kmers)
    (with backend: sketchlib v1.6.0
    sketchlib: /Users/jlees/miniconda3/envs/pp-py38/lib/python3.8/site-packages/pp_
↪sketchlib.cpython-38-darwin.so)

Graph-tools OpenMP parallelisation enabled: with 1 threads
Mode: Fitting threshold model to reference database

Network summary:
    Components      31
    Density 0.0897
    Transitivity    1.0000
    Score   0.9103
Removing 97 sequences

Done
```



5.8 lineage

This mode defines clusters by joining nearest neighbours. As this will typically define subclusters within strains, we refer to these as 'lineages'. This can be used to find subclusters in addition to one of the above models, or for species without strain-structure (e.g. some viruses, *Neisseria gonorrhoeae*, *Mycobacterium tuberculosis*). This is the highest resolution (most specific clusters) provided directly by PopPUNK. If it does not meet your needs, take a look at *Subclustering with PopPIPE* for other options.

A model is not fitted, and a simple data-driven heuristic is used. For each sample, the nearest k neighbours will be indentified, and joined in the network. Connected components of the network define lineages, as in the other models. Only core distances are used (add `--use-accessory` to modify this), and in the case of ties all distances are included. Note that these are not necessarily expected to be transitive, so network scores are not as informative of the optimum.

We refer to k as the 'rank' of the model. Typically you won't know which rank to use beforehand, so you can provide multiple integer values to the `--rank` option, comma separated. Clusters from all ranks will be output, and all used with *Query assignment (poppunk_assign)*. $k = 1$ is the most specific rank, and higher values will form looser clusters. With the *Listeria* example:

```
poppunk --fit-model lineage --ref-db listeria --ranks 1,2,3,5
PopPUNK (POPulation Partitioning Using Nucleotide Kmers)
    (with backend: sketchlib v1.6.0
    sketchlib: /Users/jlees/miniconda3/envs/pp-py38/lib/python3.8/site-packages/pp_
↪sketchlib.cpython-38-darwin.so)

Graph-tools OpenMP parallelisation enabled: with 1 threads
Mode: Fitting lineage model to reference database

Network for rank 1
Network summary:
    Components      26
    Density 0.0271
    Transitivity    0.1834
    Score   0.1785
Network for rank 2
Network summary:
    Components      12
    Density 0.0428
    Transitivity    0.3528
    Score   0.3377
Network for rank 3
Network summary:
    Components      6
    Density 0.0589
    Transitivity    0.4191
    Score   0.3944
Network for rank 5
Network summary:
    Components      2
    Density 0.0904
    Transitivity    0.5319
    Score   0.4838
Parsed data, now writing to CSV

Done
```

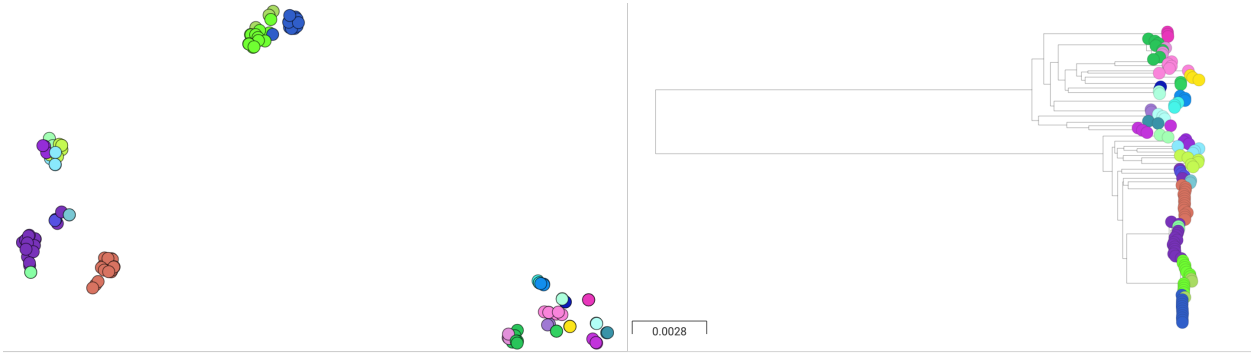
This has produced four fits, with ranks 1, 2, 3 and 5 (with fit information contained in the .pkl file, and a .npz file for each rank). The _clusters.csv will contain the clusters from the lowest rank. The _lineages.csv file contains all of the assignments, a column with all of the ranks hyphen-separated (which will give clusters indentical to the lowest rank):

```
id,Rank_1_Lineage,Rank_2_Lineage,Rank_3_Lineage,Rank_5_Lineage,overall_Lineage
12673_8#24,18,2,2,1,18-2-2-1
12673_8#26,4,2,2,1,4-2-2-1
12673_8#27,26,1,1,1,26-1-1-1
12673_8#28,1,1,1,1,1-1-1-1
12673_8#29,4,2,2,1,4-2-2-1
12673_8#31,18,2,2,1,18-2-2-1
12673_8#32,9,8,1,1,9-8-1-1
12673_8#34,7,7,1,1,7-7-1-1
12673_8#36,1,1,1,1,1-1-1-1
```

The best way to assess the ranks is by visualising them (*Creating visualisations*):

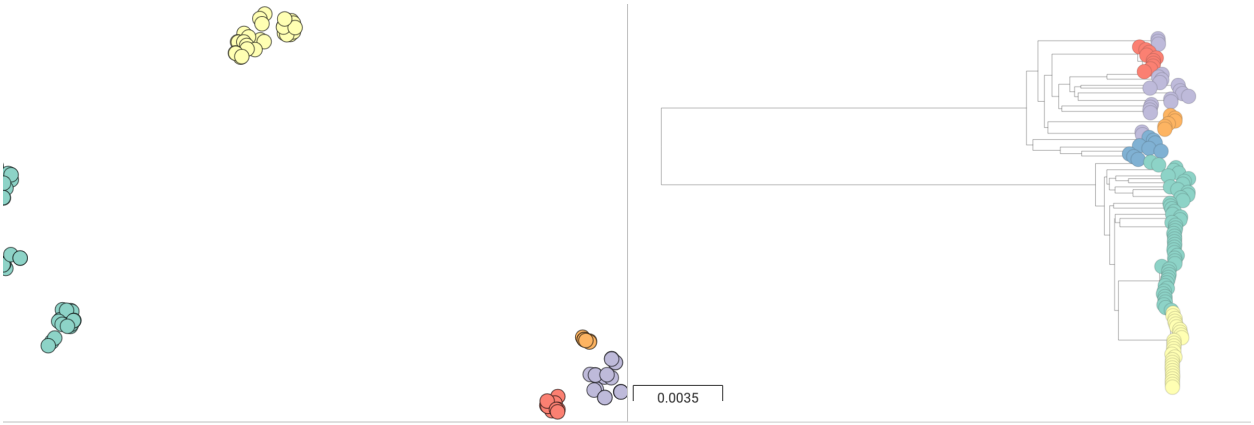
```
poppunk_visualise --distances listeria/listeria.dists --ref-db listeria --microreact

Graph-tools OpenMP parallelisation enabled: with 1 threads
PopPUNK: visualise
Loading previously lineage cluster model
Writing microreact output
Parsed data, now writing to CSV
Building phylogeny
Running t-SNE

Done
```

This can be loaded in microreact: https://microreact.org/project/dVNMftmK6VXRvDxBfrH2y. Rank 1 has the smallest clusters:



ID	Rank_1_Lineage_Cluster	Rank_2_Lineage_Cluster	Rank_3_Lineage_Cluster	Rank_5_Lineage_Cluster	overall_Lineage_Cluster
12673_8#24	18	2	2	1	18-2-2-1
12673_8#26	4	2	2	1	4-2-2-1
12673_8#27	26	1	1	1	26-1-1-1
12673_8#28	1	1	1	1	1-1-1-1
12673_8#29	4	2	2	1	4-2-2-1

Rank 3 has larger clusters. Some of these clusters are polyphyletic on the core neighbour-joining tree:



ID	Rank_1_Lineage_Cluster	Rank_2_Lineage_Cluster	Rank_3_Lineage_Cluster	Rank_5_Lineage_Cluster	overall_Lineage_Cluste
12673_8#24	18	2	2	1	18-2-2-1
12673_8#26	4	2	2	1	4-2-2-1
12673_8#27	26	1	1	1	26-1-1-1
12673_8#28	1	1	1	1	1-1-1-1
12673_8#29	4	2	2	1	4-2-2-1

At the model fit stage, you will also get histograms which show the distances included in the network, a useful comparison with the original distance distribution and between ranks:

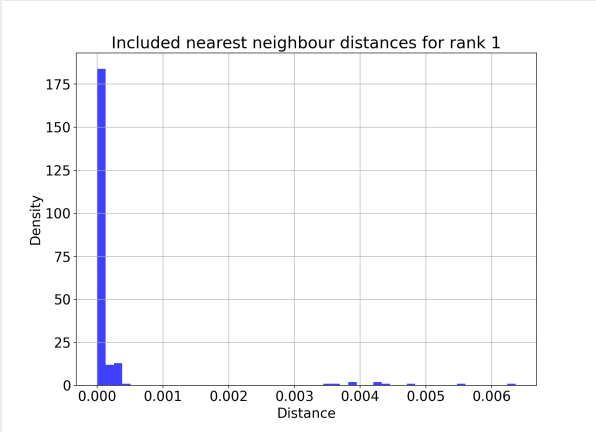


Fig. 3: Rank 1

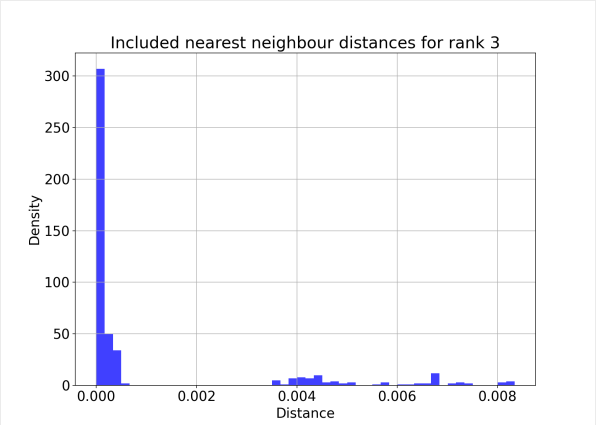


Fig. 4: Rank 3

5.9 Use an existing model with new data

There is also one further mode, `--use-model`, which may be useful in limited circumstances. This applies any of the above models to a new dataset without refitting it. This may be useful if a reference dataset has changed (been added to or removed from) and you do not wish to refit the model, for example because it is already in use. However, typically you would use *Query assignment (poppunk_assign)* with `--update-db` to add to a model:

```
poppunk --use-model --ref-db new_db --model-dir old_db
PopPUNK (POPulation Partitioning Using Nucleotide Kmers)
    (with backend: sketchlib v1.6.0
```

(continues on next page)

(continued from previous page)

```
    sketchlib: /Users/jlees/miniconda3/envs/pp-py38/lib/python3.8/site-packages/pp_
↪sketchlib.cpython-38-darwin.so)

Graph-tools OpenMP parallelisation enabled: with 1 threads
Mode: Using previous model with a reference database

Loading BGMM 2D Gaussian model
Loaded previous model of type: bgmm
Network summary:
    Components      31
    Density 0.0897
    Transitivity    1.0000
    Score   0.9103
Removing 97 sequences

Done
```

CHAPTER

SIX

DISTRIBUTING POPPUNK MODELS

If you have fitted a model yourself, you may be interested in distributing it so that others can use it for your species. This will give consistent cluster names across datasets, mean the high-quality tested fit can be reused, and speeds up future analysis.

Please contact us at poppunk@bacpop.org. We would be happy to add your sketches and fitted model to our other databases.

6.1 Database contents

A database requires the following files:

- `.h5`. The sketch database, a HDF5 file.
- `.dists.pkl` and `.dists.npy` files. Distances for all vs all samples in the sketch database.
- `_fit.npz` and `_fit.pkl` files. Python files which describe the model fit.
- `_graph.gt`. The network relating distances, fit and strain assignment for all samples in the sketch database.
- `_clusters.csv`. The strain assignment of all samples in the sketch database.

If you used a *lineage* you will also need:

- `rank_k_fit.npz`. Distances for each rank k fit.
- `_lineages.csv`. Combined lineage assignments for each rank.

You may also have `.refs` versions of these files, which are pruned to contain just the reference samples (see below). We would highly recommend including the `output/output.refs` file with any database, even though it is not strictly required, as it will speed up query assignment. Lineage models do not use references.

Note: If the database is very large, you may consider just distributing the `.refs` files. This will enable query assignment, but visualisation and subclustering within strains will no longer be possible, as full information within each strain will be missing.

These databases can be automatically generated using the `poppunk_distribute_fit.py` script after model fitting. See the *Scripts* page for more information.

6.2 Picking references

PopPUNK automatically prunes redundant sequence information from databases by removing samples from cliques (where every sample is in the same strain as every other sample). This algorithm has changed slightly from the originally published one:

1. Split the graph into connected components (strains), which are analysed in parallel.
2. Identify a clique. If no samples in the clique are already references, add one sample as a reference.
3. Prune the clique from the graph, creating a subgraph.
4. Recursively apply steps 2-3 until only two samples or fewer remain.
5. Add the remaining samples as references
6. Create the reference graph, and find connected components again.
7. For any samples which are no longer in the same connected component, find a minimum path between them in the full graph, and add all samples in this path as references.

This makes the algorithm scale better, and ensures clusters remain connected. You may find that more references are picked than before using this method, which is a small cost for the increase robustness.

This process occurs automatically after the model fit. In the *Listeria* example:

```
Removing 97 sequences
```

31 strains are represented by $128 - 97 = 31$ references, exactly one reference per cluster, which is the minimum. The refined fit removes 93 sequences with 29 strains, so some larger clusters need to be represented by multiple references. The names of the chosen references are written to the .refs file. In addition, the distances, sketch database and graph have the non-reference sequences pruned and saved with .refs suffixes. This gives a complete database suitable for assignment with references only, should the full database be prohibitively large.

Note: Previous fans (users) of PopPUNK may remember the `--full-db` option which switched off reference picking. This was useful, as reference-only databases always lost information. This option has now been removed, and reference picking will always be run. Both full and reference databases are always produced (apart from in lineage mode). The default assignment uses just references, but has the full database available for strain visualisation and subclustering.

If you interrupt the reference picking the output will still be valid. If you wish to run reference picking on a database where it is missing (due to being from an older version, or interrupted) you can do this with the `poppunk_references` script.

CHAPTER

SEVEN

QUERY ASSIGNMENT (POPPUNK_ASSIGN)

This is the recommended mode to use PopPUNK, as long as a database is available for your species. If there is no database available, you can fit your own (*Fitting new models (--fit-model)*).

Briefly, download your reference database and run:

```
poppunk_assign --db database --query qfile.txt \
--output poppunk_clusters --threads 8
```

7.1 Downloading a database

Current PopPUNK databases can be found here: https://www.bacpop.org/poppunk/

We refer to sequences in the database as references, and those being added as queries. The clusters assigned by PopPUNK are variable-length-k-mer clusters (VLKCs).

A database called `database` will contain the following files, in `database/`:

- `database.h5` – the sketches of the reference sequences generated by `pp-sketchlib`.
- `database.dists.npy` and `database.dists.pkl` – the core and accessory distances for all pairwise comparisons in the sketch database.
- `database_fit.npz` and `database_fit.pkl` – the model fit to the core and accessory distances.
- `database_graph.gt` – the network defining the fit (loadable with `graph_tool`).
- `database_clusters.csv` – the PopPUNK clusters for the reference sequences.
- `database.refs` – a minimal list of references needed to produce correct clusters.

If the `.refs` file is missing, all of the samples in the sketch database will be used in the distance calculations.

You can use the following arguments to individually target these items if necessary, for example when using an alternative fit, or if split across different directories. The examples below refer to the default database name:

- (required) `--db database` – the name of directory containing the .h5 file.
- `--distances database/database.dists` – prefix of the distances.
- `--model-dir database` – directory containing the model fit and network (dists + fit define the network).
- `--previous-clustering database` – directory containing the PopPUNK clusters for the references.

7.2 Clustering your genomes

Create a file which lists your sample names and paths to their sequence data. This file has no header, is tab separated, and contains the sample name in the first column. Subsequent columns may contain paths to either assembled or raw read data (the type will automatically be inferred by checking for the presence of quality scores). Data may be gzipped or uncompressed:

```
MS1 ms1_assembled.fa
MS2 ms2_assembled.fa
SM14        SM14_1.fq.gz SM14_2.fq.gz
```

Save this as `qfile.txt`. You're now ready to cluster them! Run the following command:

```
poppunk_assign --db database --query qfile.txt \
--output poppunk_clusters --threads 8
```

This will first of all sketch your input genomes, saving them in `poppunk_clusters/poppunk_clusters.h5`. If you need to rerun part of the analysis with different options this will automatically be picked up and loaded.

Note: *Data quality control (--qc-db)* does not apply to query sequences. A test for maximum accessory distance will be made, but the program will only emit warnings and will run with all genomes anyway. Most options for sketching will be taken from the reference database, but you can still specify error filtering options from read input (`--min-kmer-count` and `--exact-count`) and specify your input as `--strand-preserved`. See *Sketching (--create-db)* for more information on these options.

Next, core and accessory distances between your input sketches and those in the database will be computed. This has complexity $O(RQ)$ where R is the number of samples in `database_references.refs` and Q is the number in `qfile.txt`. These distances are then fed into the model and used to update the network, and therefore clusters.

The output will look something like this:

```
Graph-tools OpenMP parallelisation enabled: with 4 threads
PopPUNK (POPulation Partitioning Using Nucleotide Kmers)
        (with backend: sketchlib v1.5.1
         sketchlib: /Users/jlees/miniconda3/envs/pp-py38/lib/python3.8/site-packages/pp_
↪sketchlib.cpython-38-darwin.so)
Mode: Assigning clusters of query sequences

Sketching genomes using 8 thread(s)
Calculating distances using 8 thread(s)
Loading previously refined model
Network loaded: 2007 samples
Found novel query clusters. Calculating distances between them.
Could not find random match chances in database, calculating assuming equal base␣
↪frequencies
Calculating distances using 8 thread(s)
```

Your VLKCs will be written to `poppunk_clusters/poppunk_clusters_clusters.csv`:

```
Taxon,Cluster
21946_6_66,9
22695_3_148,9
22984_8_88,9
```

(continues on next page)

(continued from previous page)

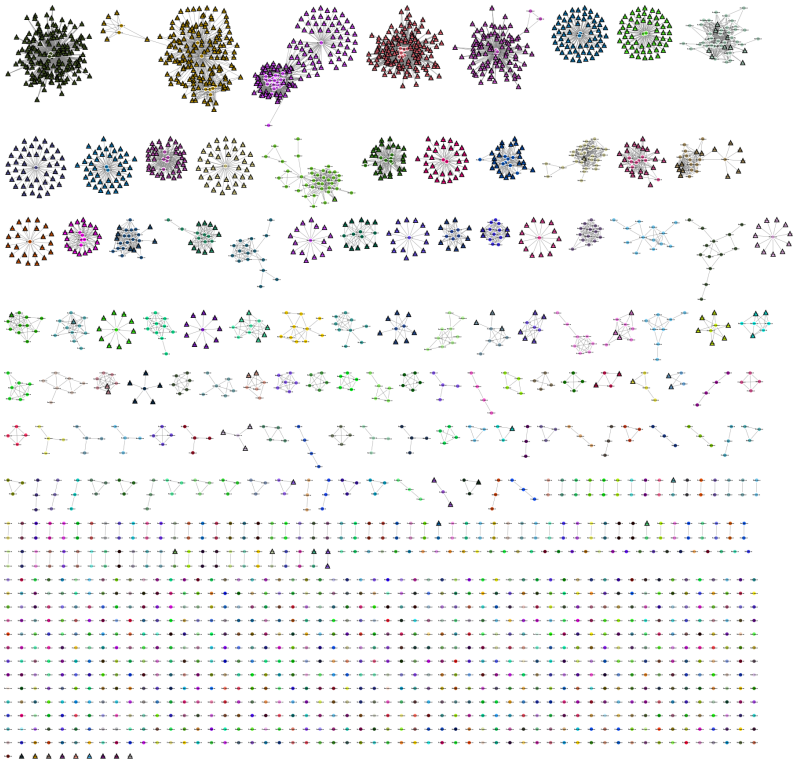
```
21946_6_245,116
21946_6_189,814
22695_3_73,814
21946_6_50,422
21903_8_95,148
21903_8_250,301
22984_8_47,70
```

These names are identical to those used in the reference database, so retain the same meaning between studies. If new clusters are found they will be numbered in ascending order from largest to smallest, beginning from the end of the reference clusters.

Note: You may observe clusters merging (but never splitting). If your genomes do cause clusters to merge this will be noted in the output, and the new clusters will be named using the old ones. For example, if clusters 23 and 38 merged, the new cluster would be called 23_38.

By default, only the query genome clusters are included here. The reference genome clusters are considered unchanged from the input. If there are many merges and you wish to know their new cluster IDs, use `--update-db` (*Updating the database*).

You can use `poppunk_visualise` to look at your results. Here's an example output to cytoscape, showing the clusters as colours, reference genomes as circles and queries as triangles (open in a new tab to zoom on detail):



7.2.1 Adding external cluster labels (MLST, CC etc)

Add the `--external-clustering` argument to add a CSV file of cluster definitions which the output will be additionally labelled with, and output to `database/database_external_clusters.csv`. These can be any cluster definitions you wish, with as many columns as you like. A header row is required:

```
sample,GPSC,MLST
23430_1_186,1,22
17794_6_29,23,43
12291_4_13,1,2
```

For each PopPUNK cluster, all the samples found in said cluster will be accumulated. From these accumulated samples the external clusters will be collected, and assigned to all of these examples. This may give you a one-to-one mapping between PopPUNK clusters and your external cluster, or you may find multiple external clusters refer to the PopPUNK cluster giving output such as `227;811;763;824`.

7.2.2 Using a model fitted with `--indiv-refine`

If the database was fitted with the refine fit mode, and `indiv-refine` you may have a core distance boundary, accessory boundary and combined core-accessory boundary fit. The default is to use the combined boundary, to use the others add `--core-only` or `--accessory-only`.

7.3 Increasing speed

Query assignment is the most efficient mode in which to run PopPUNK, typically requiring Q sketches and RQ distances. If you are updating the database, this increases to $Q^2 + RQ$ distances. If you are assigning a very large number of queries you can run `poppunk_assign` with `--update-db` repeatedly for batches of query input, as the Q^2 term will be reduced by clique-pruning at each iteration.

Straightforward ways to increase speed include:

- Add `--gpu-dist`, if you have a GPU available.
- Add `--gpu-sketch`, if your input is all reads, and you have a GPU available. If your input is a mix of assemblies and reads, run in two separate batches, with the batch of reads using this option.
- Increase `--threads`.

7.4 Updating the database

If you want to add your query genomes into the reference database so that they can be used to inform future cluster assignment, this is as simple as adding the `--update-db` option to the command above. This is particularly useful when novel query clusters have been found – they will then be the consistent name for future assignments:

```
poppunk_assign --db database --query qfile.txt \
--output poppunk_clusters --threads 8 --update-db

Graph-tools OpenMP parallelisation enabled: with 4 threads
PopPUNK (POPulation Partitioning Using Nucleotide Kmers)
    (with backend: sketchlib v1.5.1
    sketchlib: /Users/jlees/miniconda3/envs/pp-py38/lib/python3.8/site-packages/pp_
```

(continues on next page)

(continued from previous page)

```
↪sketchlib.cpython-38-darwin.so)
Mode: Assigning clusters of query sequences

Sketching 28 genomes using 4 thread(s)
Writing sketches to file
Calculating distances using 4 thread(s)
Loading BGMM 2D Gaussian model
Network loaded: 18 samples
Calculating all query-query distances
Could not find random match chances in database, calculating assuming equal base␣
↪frequencies
Calculating distances using 4 thread(s)
Updating reference database to poppunk_clusters
Removing 27 sequences

Done
```

The new database contains all of the reference sequences, and all of your query sequences. The `poppunk_clusters` folder will now contain all of the files of a reference database listed above, except for the model. You can use `--model-dir` to target this for future assignment, or copy it over yourself. Alternatively, if you run with the same `--output` folder as `--ref-db`, adding `--overwrite`, the original input folder will contain the updated database containing everything needed.

Note: This mode can take longer to run with large numbers of input query genomes, as it will calculate all Q^2 query-query distances, rather than just those found in novel query clusters.

7.5 Visualising results

If you wish to produce visualisations from query assignment results the best way to do this is to run with `--update-db`, and then run `poppunk_visualise` on the output directory, as if visualising a full reference fit.

However, it is possible to run directly on the outputs by adding a `--ref-db` as used in the assign command, and a `--query-db` which points to the `--output` directory used in the assign command. In this mode isolates will be annotated depending on whether they were a query or reference input.

> **Warning:** Without `--update-db`, visualisation is required to recalculate all query-query distances each time it is called. If your query set is large and you want repeated visualisations, run `poppunk_assign` with `--update-db`.

See *Creating visualisations* for more details.

CHAPTER

EIGHT

CREATING VISUALISATIONS

We have moved visualisation tools into their own program `poppunk_visualise`, both to reinforce our commitment to UK spellings, and so that you can rerun visualisations with different outputs and settings without rerunning the other parts of the code.

Starting with either a full database where you have fitted a model (*Fitting new models (--fit-model)*), or output where you have assigned queries using an existing database (*Query assignment (poppunk_assign)*), you can create outputs for external interactive visualisation tools:

- Microreact – a genomic epidemiology visualisation tool, displaying clusters, phylogeny and accessory clustering.
- GrapeTree – a tool to visualise strains (designed for cgMLST).
- Phandango – visualisation linking genomes and phylogenies.
- Cytoscape – a network visualisation tool, which can be used to create, view and manipulate a layout of the graph.

At least one of these output types must be specified as a flag.

Important: If you run a visualisation on output from query assignment (*Query assignment (poppunk_assign)*) this will not contain all the necessary distances, and they will be calculated before the visualisation files are produced. You will see a message `Note: Distance will be extended to full all-vs-all distances`. If you are running multiple visualisations this calculation will be completed every time. To avoid this re-run your assignment with `--update-db`, which will add these distances in permanently.

8.1 Common options

Some typical commands for various input settings (with `--microreact`, but this can be altered to any output type) with a database `example`.

Visualisation of a full database:

```
poppunk_visualise --ref-db example_db --output example_viz --microreact
```

Visualisation after query assignment:

```
poppunk_visualise --ref-db example_db --query-db example_query --output example_viz --
↪microreact
```

Visualisation when sketches and models are in different folders:

```
poppunk_visualise --ref-db example_db --previous-clustering example_lineages/example_
↪lineages_lineages.csv \
 --model-dir example_lineages --output example_viz --microreact
```

Visualisation with a lineage model, which has been queried (query-query distances must be provided):

```
poppunk_visualise --distances example_query/example_query.dists --ref-db example_db \
 --model-dir example_lineages --query-db example_lineage_query \
 --output example_viz --microreact
```

Notable modifiers include:

- `--include-files` – give a file with a subset of names to be included in the visualisation.
- `--external-clustering` – other cluster names to map to strains (such as MLST, serotype etc), as described in model fitting and query assignment.
- `--info-csv` – similar to the above, but a CSV which is simply (inner-)joined to the output on sample name.
- `--rapidnj` – the location of a rapidnj binary, used to build the core NJ tree. We highly recommend using this for any tree-building (and is included with the conda install). This defaults to `rapidnj` on the PATH. Set blank to use dendropy instead (slower, especially for large datasets).
- `--core-only`/`--accessory-only` – use the core or accessory fit from an individually refined model (see *Using core/accessory only*).
- `--threads`, `--gpu-dist`, `--deviceid`, `--strand-preserved` – querying options used if extra distance calculations are needed. To avoid these, rerun your query with `--update-db`.

8.2 Microreact

Adding the `--microreact` flag will create the following files:

- _microreact_clusters.csv – the strain or lineage assignments with headers formatted for microreact, plus anything from `--info-csv`.
- _core_NJ.nwk – a neighbour-joining tree from the core distances.
- _perplexity20.0_accessory_mandrake.dot – a 2D embedding of the accessory distances, produced using mandrake (in this case with *Setting the perplexity parameter for mandrake* 20).

From version 2.5.0 this will also include:

- **.microreact – a Microreact compatible JSON containing all of the above, which** can be uploaded directly.

If you add `--api-key` and provide *your account's API key <https://docs.microreact.org/api/access-tokens>__* this will automatically create an instance, and the URL will be output to the terminal.

Otherwise, open https://microreact.org/upload in your browser, and drag and drop these three files to create your visualisation. Here is the result of running the visualisation on the *Listeria* BGMM model:

```
poppunk_visualise --ref-db listeria --microreact --threads 8

Graph-tools OpenMP parallelisation enabled: with 8 threads
PopPUNK: visualise
Loading BGMM 2D Gaussian model
Completed model loading
```

(continues on next page)

(continued from previous page)

```
Building phylogeny
Writing microreact output
Parsed data, now writing to CSV
Running mandrake
Running on CPU
Preprocessing 128 samples with perplexity = 20 took 0ms
Optimizing   Progress: 99.9%, eta=0.0010, Eq=0.2583546852, clashes=2.1%
Optimizing done in 30s
Provide --api-key to create microreact automatically

Done
```

This can be viewed at https://microreact.org/project/3JAZKqzJiaNyViWXindNLv-listeria-poppunk-perplexity-20:

Useful controls include the tree shape, accessed with the control slider in the top right of the phylogeny page, and the metadata labels, accessed with the 'eye' on the right of the page. When visualising lineages, changing the 'Colour by' is useful to compare results from different ranks.

8.2.1 Setting the perplexity parameter for mandrake

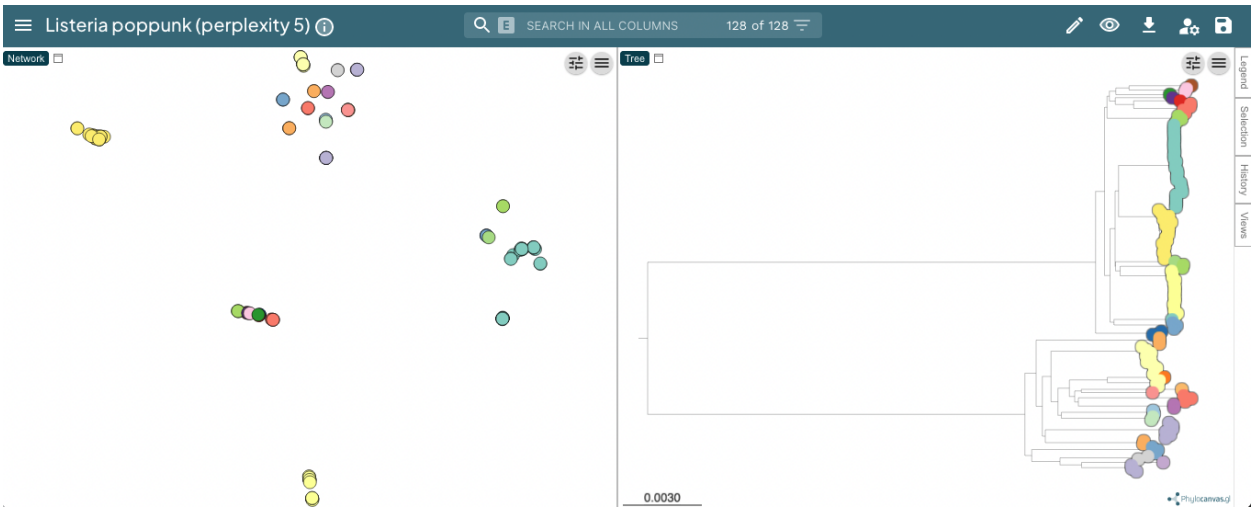
In mandrake an embedding of the accessory genome distances is found which represents local structure of the data. Isolates with similar accessory content will visually appear in clusters together.

The perplexity sets a guess about the number of close neighbours each point has, and is a trade-off between local and global structure. t-SNE (and by extension mandrake) is reasonably robust to changes in the perplexity parameter (set with `--perplexity` when creating microreact output with `--microreact`), however we would recommend trying a few values to get a good embedding for the accessory distances.

There is a good discussion of the effect of perplexity here and the sklearn documentation shows some examples of the effect of changing perplexity. In mandrake, points will usually appear 'tighter' than in t-SNE, and form more obvious clusters.

In the example with *Listeria monocytogenes* above, a perplexity of 20 gives clear clustering of the accessory genome content, condordant with the core genome structure (data):

With a lower perplexity of 5, the clustering is not as tight, but it still looks ok (data):



30 is a good default, but you may wish to try other values, particularly with larger or smaller datasets. You can re-run the mandrake using the `poppunk_mandrake` command, providing the distances from the previous run:

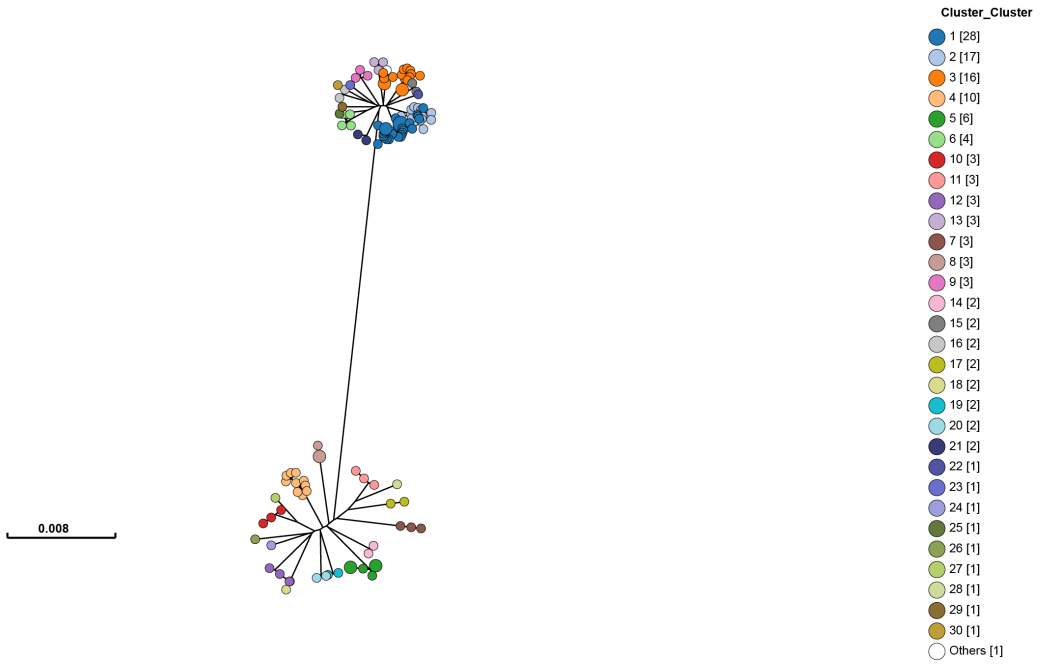
```
poppunk_mandrake --distances strain_db/strain_db.dists --output strain_db \
--perplexity 50
```

8.3 GrapeTree

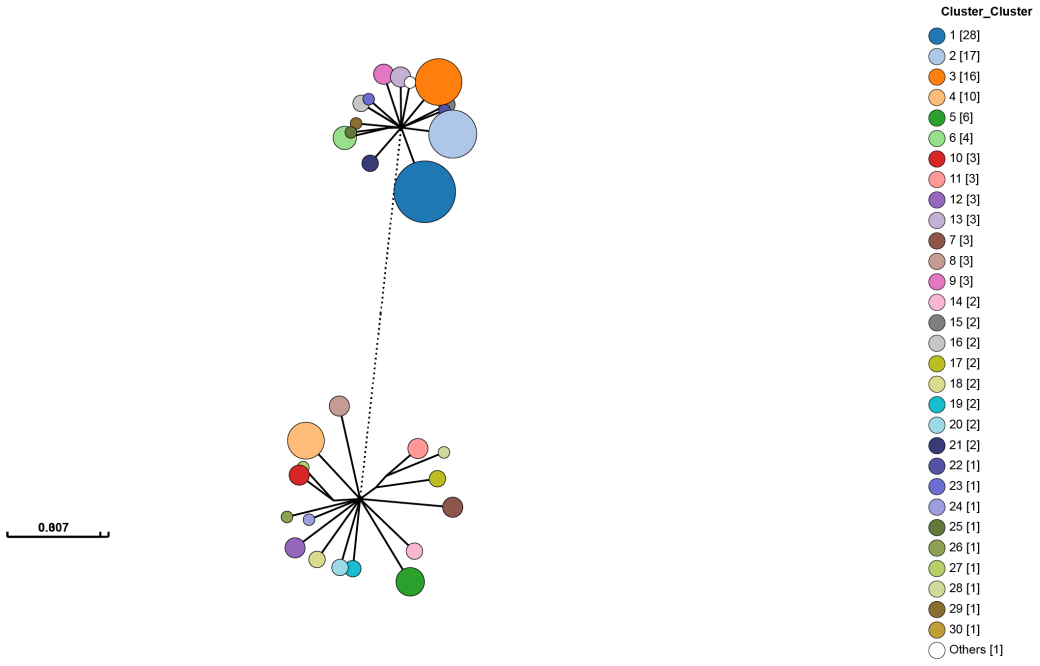
Adding the `--grapetree` flag will create:

- _microreact_clusters.csv – the strain or lineage assignments with headers formatted for grapetree, plus anything from `--info-csv`.
- _core_NJ.nwk – a neighbour-joining tree from the core distances.

Open https://achtman-lab.github.io/GrapeTree/MSTree_holder.html in your browser, and use the 'Load files' button once for each of the files to add the tree and strain assignments to GrapeTree. This will display an unrooted tree with your clusters:



One of GrapeTree's key features is the ability to collapse branches, and condense information into nodes. By going to Tree Layout -> Branch style -> Collapse branches, and setting the long branch to be shortened, one can obtain a view which shows strain prevalence and relationships:



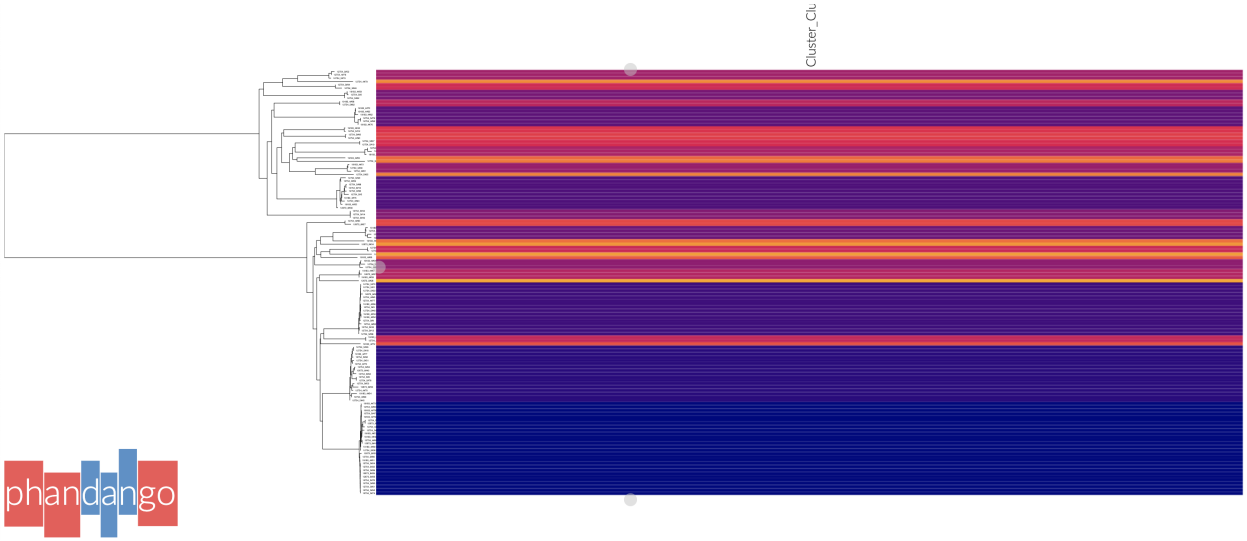
There is also a handy 'Export to Microreact' button in GrapeTree, though this will not include the accessory embedding, so you may wish to add the `--microreact` flag and generate the files yourself.

8.4 Phandango

Adding the `--phandango` flag will create:

- _phandango_clusters.csv – the strain or lineage assignments with headers formatted for phandango, plus anything from `--info-csv`.
- _core_NJ.tree – a neighbour-joining tree from the core distances.

Open https://www.phandango.net in your browser, and use the 'Load files' button once for each of the files to add the tree and strain assignments to GrapeTree. This will display the tree with your clusters:



Press 's' to access settings, and 'p' to create an .svg file. Phandango is most useful with a genome (.gff file), and either a plot of recombination, accessory genome analysis or GWAS results. See the documentation for more information.

8.5 Cytoscape

Cytoscape is different from the above modes as it creates a layout and visualisation of the graph used to create strains from distances. This can be useful for more detailed investigation of network scores, particularly in strains which have less than perfect transitivity.

Add the `--cytoscape` option, and also `--network-file` to point to the network you wish to visualise:

```
poppunk_visualise --ref-db listeria --cytoscape --network-file listeria/listeria_graph.gt

Graph-tools OpenMP parallelisation enabled: with 1 threads
PopPUNK: visualise
Loading BGMM 2D Gaussian model
Writing cytoscape output
Network loaded: 128 samples
Parsed data, now writing to CSV

Done
```

Which will create:

- _cytoscape.csv – the strain or lineage assignments with headers formatted for cytoscape, plus anything from `--info-csv`.
- _cytoscape.graphml – the network in graphml format.

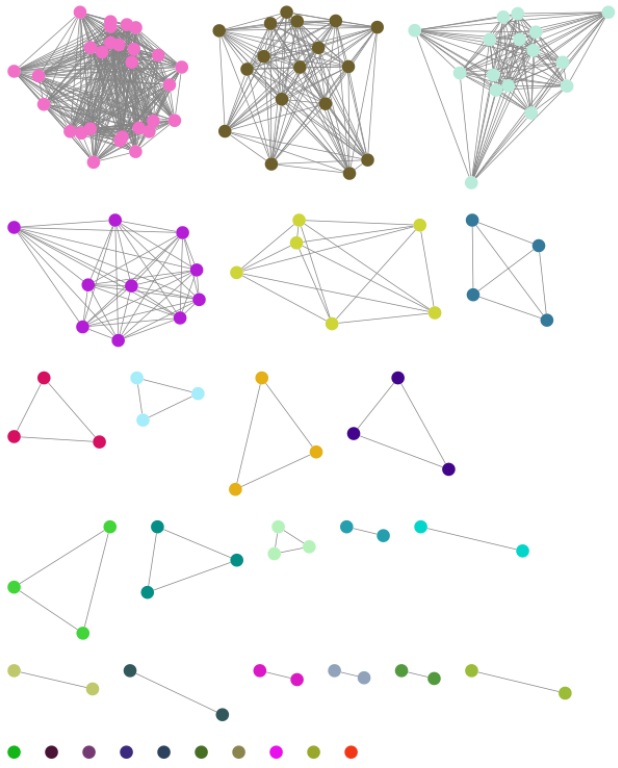
The .graphml file is an XML file which contains definitions of the nodes (samples) and edges (within-strain distances) connecting them. If you used `--graph-weights` when you fitted your model the edges will be annotated with their Euclidean distances in the 'weight' attribute (which you will need to tell cytoscape). These can be added with the `poppunk_add_weights` script if this flag was not used.

Open cytoscape and drag and drop the .graphml file onto the window to import the network. Import -> table -> file to load the CSV. Click 'Select None' then add the 'id' column as a key, and any required metadata columns (at least the 'Cluster' column) as attributes. Make sure 'Node Table Columns' is selected as the data type.

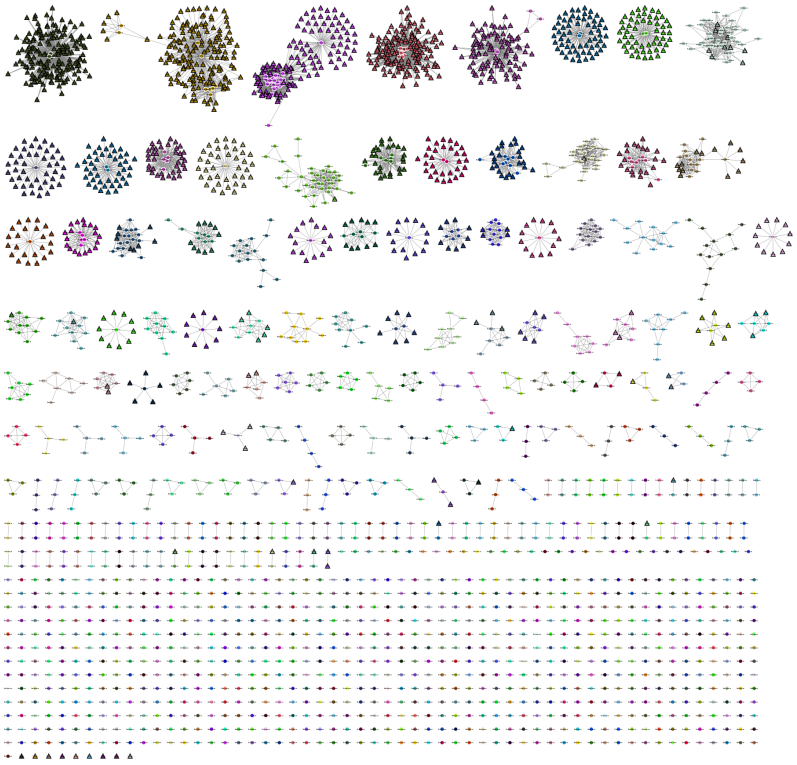
The graphml file does not contain a layout for the graph, that is, positions of nodes and edges are not specified for a visualisation. These will be calculated by cytoscape, automatically for small graphs, and with the 'Layout' menu for larger graphs. The 'Prefuse force directed layout' or 'yFiles Organic Layout' work well. Select the 'weight' dropdown to use the edge-lengths when drawing the network.

> **Warning:** We have found that making graphs with >10k nodes may exceed the memory on a typical laptop. To view larger graphs, first splitting into subgraphs of each connected component is very helpful. Older versions of cytoscape allowed you to split the graph into connected components, but newer versions have removed this feature. This can be done programmatically with `networkx` or `graph-tool` in python, or `igraph` in R.

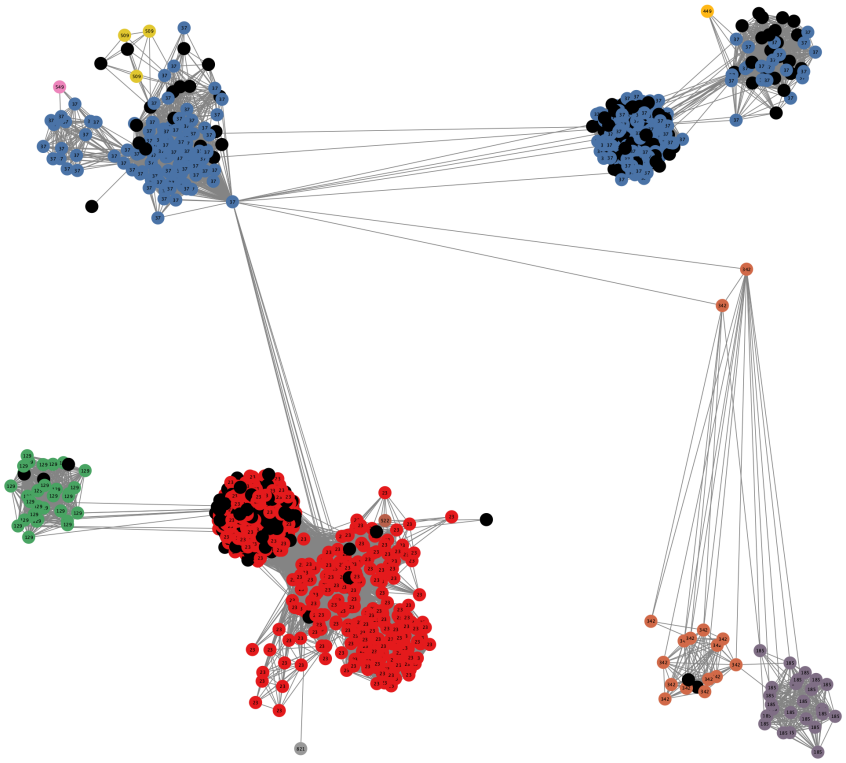
Click on 'Style' and change the node fill colour to be by cluster, the mapping type as discrete, then right click to autogenerate a colour scheme ('Random' is usually best). You can also modify the node size and shape here. Here is the *Listeria* example, using edge weights in the layout:



If you used assign query mode you will also have a column with 'Query' or 'Reference', which can be used to map to different shapes or colours:



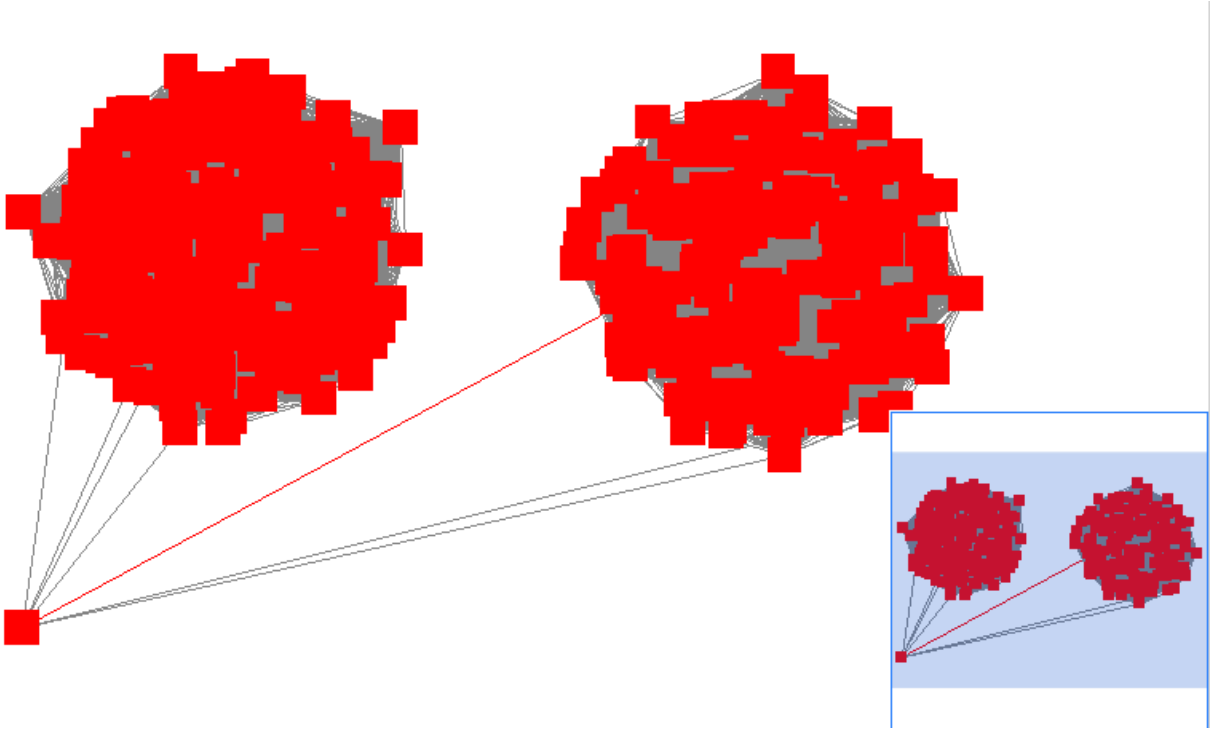
Adding an info CSV, or loading external tables directly into cytoscapes gives further options for investigating individual strains:



In some cases, edges which are between strain links may have been erroneously included in the network. This could be due to poor model fit, or a poor quality sequence. Use Tools -> NetworkAnalyzer -> Analyze Network to compute

information for each node and edge. It may help to analyze connected components separately. They can be split under Tools -> NetworkAnalyzer -> Subnetwork Creation.

Here is an example where an errant node is connecting two clusters into one large cluster, which should be split:



The incorrect node in question has a low CluteringCoefficient and high Stress. The EdgeBetweeness of its connections are also high. Sorting the node and edge tables by these columns can find individual problems such as this.

CHAPTER

NINE

MINIMUM SPANNING TREES

Using the distances and a network, you can generate a minimum spanning tree. This can be useful when a neigbour joining tree is difficult to produce, for example if the dataset is very large, and in some cases has uses in tracing spread (take care with this interpretation, direction is not usually obvious).

There are three different ways to make MSTs, depending on how much data you have. Roughly:

- 'Small': Up to $\sim 10^3$ samples.
- 'Medium': Up to $\sim 10^5$ samples.
- 'Large': Over 10^5 samples.

In each mode, you can get as output:

- A plot of the MST as a graph layout, optionally coloured by strain.
- A plot of the MST as a graph layout, highlighting edge betweenness and node degree.
- The graph as a graphml file, to view in *Cytoscape*.
- The MST formatted as a newick file, to view in a tree viewer of your choice.

9.1 With small data

For a small dataset it's feasible to find the MST from your (dense) distance matrix. In this case you can use *Creating visualisations* with the `--tree` option: use `--tree both` to make both a MST and NJ tree, or `--tree mst` to just make the MST:

```
poppunk_visualise --ref-db listeria --tree both --microreact --output dense_mst_viz

Graph-tools OpenMP parallelisation enabled: with 1 threads
PopPUNK: visualise
Loading BGMM 2D Gaussian model
Completed model loading
Generating MST from dense distances (may be slow)
Starting calculation of minimum-spanning tree
Completed calculation of minimum-spanning tree
Drawing MST
Building phylogeny
Writing microreact output
Parsed data, now writing to CSV
Running t-SNE

Done
```

Note the warning about using dense distances. If you are waiting a long time at this point, or running into memory issues, considering using one of the approaches below.

Note: The default in this modeis to use core distances, but you can use accessory or Euclidean core-accessory distances by modifying `--mst-distances`.

This will produce a file `listeria_MST.nwk` which can be loaded into Microreact, Grapetree or Phandango (or other tree viewing programs). If you run with `--cytoscape`, the .graphml file of the MST will be saved instead.

You will also get two visualisations of a force-directed graph layout:

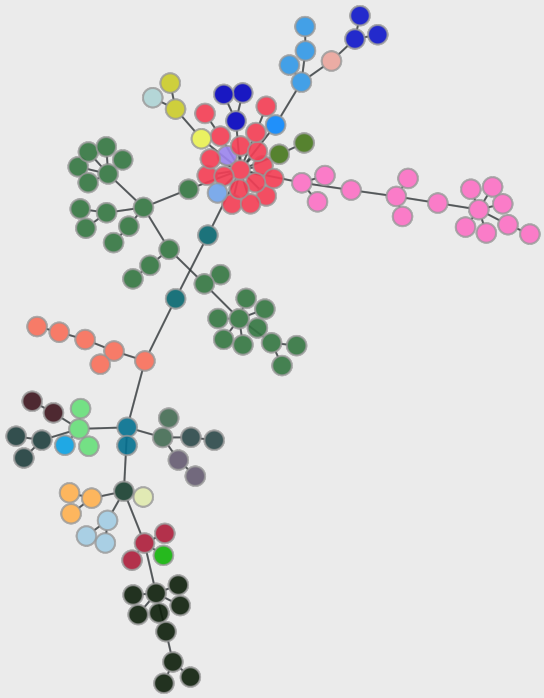


Fig. 1: MST coloured by strain

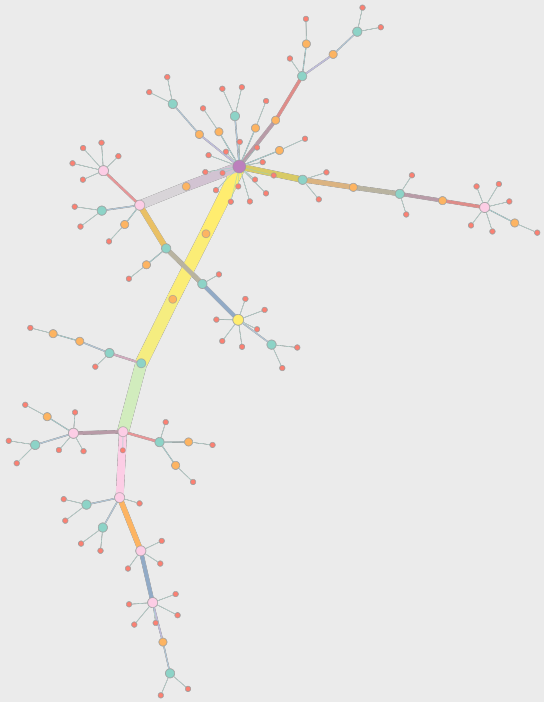


Fig. 2: MST coloured by betweenness

The left plot colours nodes (samples) by their strain, with the colour selected at random. The right plot colours and sizes nodes by degree, and edges colour and width is set by their betweenness.

9.2 With medium data

As the number of edges in a dense network will grow as $O(N^2)$, you will likely find that as sample numbers grow creating these visualisations becomes prohibitive in terms of both memory and CPU time required. To get around this, you can first sparisfy your distance matrix before computing the MST. The *lineage* mode does exactly this: keeping only a specified number of nearest neighbours for each sample.

Therefore, there are two steps to this process:

- Fit a lineage model to your data, using a high rank.

- Use `poppunk_mst` to make the MST from the sparse matrix saved by this mode.

As an example, two commands might be:

```
poppunk --fit-model lineage --ref-db listeria_all --ranks 50 --threads 4 --output sparse_
↪mst

poppunk_visualise --ref-db listeria --tree both --microreact  \
--rank-fit sparse_mst/sparse_mst_rank50_fit.npz --output sparse_mst_viz --threads 4
```

Ideally you should pick a rank which is large enough to join all of the components together. If you don't, components will be artificially connected by nodes with the largest degree, at the largest included distance. Look for components to be one:

```
Network for rank 100
Network summary:
        Components      1
        Density 0.3252
        Transitivity    0.5740
        Score   0.3873
```

This will produce a `<name>_rank100_fit.npz` file, which is the sparse matrix to load. You will also need your dense distances, but only the `.pkl` file is loaded to label the samples. `--previous-clustering` is optional, and points to any .csv output from PopPUNK. Note that the clusters produced from your high rank fit are likely to be meaningless, so use clusters from a fit you are happy with. These are combined to give samples coloured by strain in the first plot:

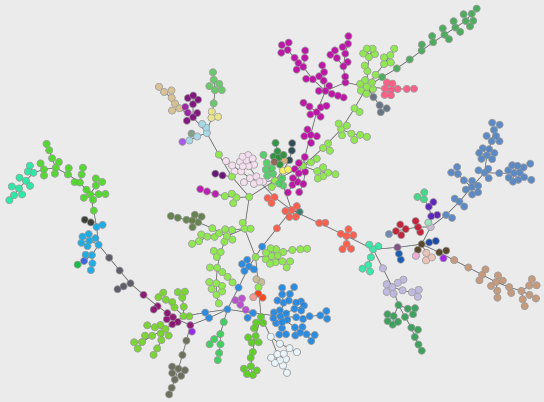


Fig. 3: MST from a sparse matrix, coloured by strain

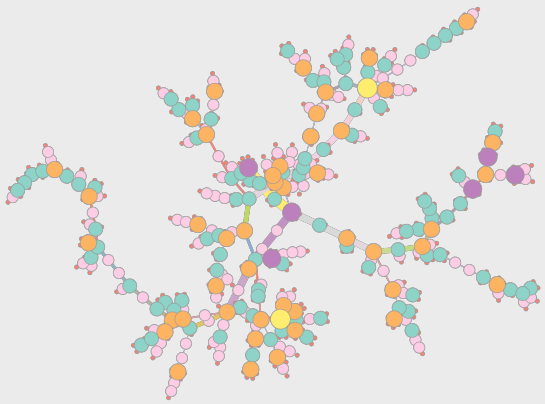


Fig. 4: MST from a sparse matrix, coloured by betweenness

9.3 With big data

For very large datasets, producing a dense distance matrix at all may become totally infeasible. Fortunately, it is possible to add to the sparse matrix iteratively by making a lineage fit to a subset of your data, and then repeatedly adding in blocks with `poppunk_assign` and `--update-db`:

```
poppunk --create-db --r-files qfile1.txt --output listeria_1
poppunk --fit-model lineage --ref-db listeria_1 --ranks 500 --threads 16
```

(continues on next page)

(continued from previous page)

```
poppunk_assign --ref-db listeria_1 --q-files qfile2.txt --output listeria_1 --threads 16␣
↪--update-db
poppunk_assign --ref-db listeria_1 --q-files qfile3.txt --output listeria_1 --threads 16␣
↪--update-db
```

This will calculate all vs. all distances, but many of them will be discarded at each stage, controlling the total memory required. The manner in which the sparse matrix grows is predictable: $Nk + 2NQ + Q^2 - Q$ distances are saved at each step, where N is the number of references, Q is the number of requires queries and k is the rank.

If you split the samples into roughly equally sized blocks of Q samples, the Q^2 terms dominate. So you can pick Q such that $\sim 3Q^2$ distances can be stored (each distance uses four bytes). The final distance matrix will contain Nk distances, so you can choose a rank such that this will fit in memory.

You may then follow the process described above to use `poppunk_visualise` to generate an MST from your `.npz` file after updating the database multiple times.

9.3.1 Using GPU acceleration for the graph

As an extra optimisation, you may add `--gpu-graph` to use cuGraph from the RAPIDS library to calculate the MST on a GPU:

```
poppunk_visualise --ref-db listeria --tree both --rank-fit sparse_mst/sparse_mst_rank50_
↪fit.npz\
--microreact --output sparse_mst_viz --threads 4 --gpu-graph

Graph-tools OpenMP parallelisation enabled: with 1 threads
Loading distances into graph
Calculating MST (GPU part)
Label prop iterations: 6
Label prop iterations: 5
Label prop iterations: 5
Label prop iterations: 4
Label prop iterations: 2
Iterations: 5
12453,65,126,13,283,660
Calculating MST (CPU part)
Completed calculation of minimum-spanning tree
Generating output
Drawing MST
```

This uses cuDF to load the sparse matrix (network edges) into the device, and cuGraph to do the MST calculation. At the end, this is converted back into graph-tool format for drawing and output. Note that this process incurs some overhead, so will likely only be faster for very large graphs where calculating the MST on a CPU is slow.

To turn off the graph layout and drawing for massive networks, you can use `--no-plot`.

Important: The RAPIDS packages are not included in the default PopPUNK installation, as they are in non-standard conda channels. To install these packages, see the guide.

CHAPTER

TEN

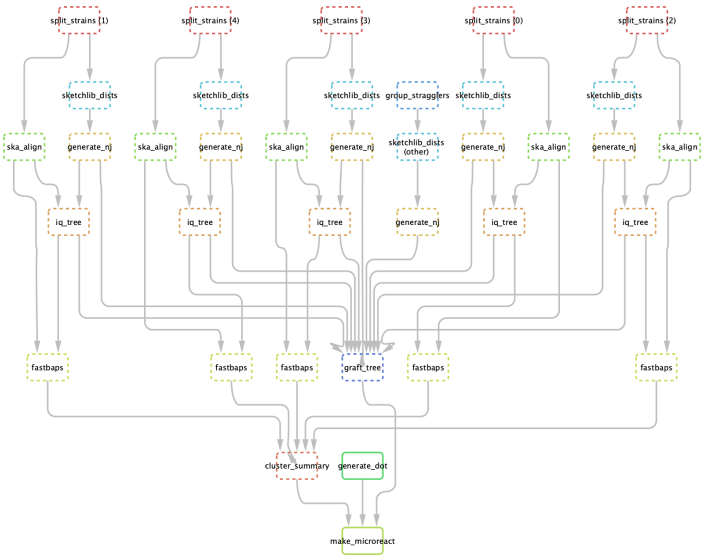
SUBCLUSTERING WITH POPPIPE

10.1 Overview

You can run PopPIPE on your PopPUNK output, which will run subclustering and visualisation within your strains. The pipeline consists of the following steps:

- Split files into their strains.
- Calculate core and accessory distances within each strain.
- Use the core distances to make a neighbour-joining tree.
- (lineage_clust mode) Generate clusters from core distances with lineage clustering in PopPUNK.
- Use ska to generate within-strain alignments.
- Use IQ-TREE to generate an ML phylogeny for each strain using this alignment, and the NJ tree as a starting point.
- Use fastbaps to generate subclusters which are partitions of the phylogeny.
- Create an overall visualisation with both core and accessory distances, as in PopPUNK. The final tree consists of refining the NJ tree by grafting the maximum likelihood trees for subclusters to their matching nodes.

An example DAG for the steps (excluding `ska index`, for which there is one per sample):



10.2 Installation

PopPIPE is a snakemake pipeline, which depends upon snakemake and pandas:

```
conda install snakemake pandas
```

Other dependencies will be automatically installed by conda the first time you run the pipeline. You can also install them yourself and omit the *-use-conda* directive to snakemake:

```
conda env create -n poppipe --file=environment.yml
```

Then clone the repository:

```
git clone git@github.com:johnlees/PopPIPE.git
```

10.3 Usage

1. Modify `config.yml` as appropriate.
2. Run `snakemake --cores <n_cores> --use-conda`.

On a cluster or the cloud, you can use snakemake's built-in `--cluster` argument:

```
snakemake --cluster qsub -j 16 --use-conda
```

See the snakemake docs for more information on your cluster/cloud provider.

10.3.1 Alternative runs

For quick and dirty clustering and phylogenies using core distances from pp-sketchlib alone, run:

```
snakemake --cores <n_cores> --use-conda lineage_clust
```

To create a visualisation on microreact:

snakemake –use-conda make_microreact

10.4 Config file

10.4.1 PopPIPE configuration

- `script_location`: The `scripts/` directory, if not running from the root of this repository
- `poppunk_db`: The PopPUNK HDF5 database file, without the `.h5` suffix.
- `poppunk_clusters`: The PopPUNK cluster CSV file, usually `poppunk_db/poppunk_db_clusters.csv`.
- `poppunk_rfile`: The `--rfile` used with PopPUNK, which lists sample names and files, one per line, tab separated.
- `min_cluster_size`: The minimum size of a cluster to run the analysis on (recommended at least 6).

10.4.2 IQ-TREE configuration

- `enabled`: Set to `false` to turn off ML tree generation, and use the NJ tree throughout.
- `mode`: Set to `full` to run with the specified model, set to `fast` to run using `--fast` (like fasttree).
- `model`: A string for the `-m` parameter describing the model. Adding `+ASC` is recommended.

10.4.3 fastbaps configuration

- `levels`: Number of levels of recursive subclustering.
- `script`: Location of the `run_fastbaps` script. Find by running `system.file("run_fastbaps", package = "fastbaps")` in R.

10.5 Updating a run

Running `snakemake` from the same directory will keep outputs where possible, so new additions will automatically be included.

TODO: How to do this when adding new isolates with `poppunk_assign`

CHAPTER

ELEVEN

USING GPUS

PopPUNK can use GPU acceleration of sketching (only when using sequence reads), distance calculation, network construction and some aspects of visualisation. Installing and configuring the required packages necessitates some extra steps, outlined below.

11.1 Installing GPU packages

To use GPU acceleration, PopPUNK uses `cupy`, `numba` and the `cudatoolkit` packages from RAPIDS. Both `cupy` and `numba` can be installed as standard packages using conda. The `cudatoolkit` packages need to be matched to your CUDA version. The command `nvidia-smi` can be used to find the supported CUDA version. Installation of the `cudatoolkit` with conda (or the faster conda alternative, mamba) should be guided by the RAPIDS guide. This information will enable the installation of PopPUNK into a clean conda environment with a command such as (modify the `CUDA_VERSION` variable as appropriate):

```
export CUDA_VERSION=11.3
conda create -n poppunk_gpu -c rapidsai -c nvidia -c conda-forge \
-c bioconda -c defaults rapids>=22.12 python=3.8 cudatoolkit=$CUDA_VERSION \
pp-sketchlib>=2.0.1 poppunk>=2.6.0 networkx cupy numba
conda activate poppunk_gpu
```

The version of `pp-sketchlib` on conda only supports some GPUs. A more general approach is to install from source. This requires the installation of extra packages needed for building packages from source. Additionally, it is sometimes necessary to install versions of the CUDA compiler (`cuda-nvcc`) and runtime API (`cuda-cudart`) that match the CUDA version. Although conda can be used, creating such a complex environment can be slow, and therefore we recommend mamba as a faster alternative:

```
export CUDA_VERSION=11.3
mamba create -n poppunk_gpu -c rapidsai -c nvidia -c conda-forge \
-c bioconda -c defaults rapids=22.12 python>=3.8 cudatoolkit=$CUDA_VERSION \
cuda-nvcc=$CUDA_VERSION cuda-cudart=$CUDA_VERSION networkx cupy numba cmake \
pybind11 highfive Eigen openblas libgomp libgfortran-ng poppunk>=2.6.0
```

Note: On OSX replace `libgomp libgfortan-ng` with `llvm-openmp gfortran_impl_osx-64`, and remove `libgomp` from `environment.yml`.

Clone the sketchlib repository:

```
git clone https://github.com/bacpop/pp-sketchlib.git
cd pp-sketchlib
```

To correctly build `pp-sketchlib`, the GPU architecture needs to be correctly specified. The `nvidia-smi` command can be used to display the GPUs available to you. This can be used to identify the corresponding compute version needed for compilation (typically of the form `sm_*`) using this guide or the more limited table below. Edit the `CMakeLists.txt` if necessary to change the compute version to that used by your GPU. See the CMAKE_CUDA_COMPILER_VERSION section.

Table 1: GPU compute versions

GPU	Compute version
20xx series	75
30xx series	86
40xx series	89
V100	70
A100	80
A5000	86
H100	90

The conda-installed version of `pp-sketchlib` can then be removed with the command:

```
conda remove --force pp-sketchlib
```

Then run:

```
python setup.py install
```

You should see a message that the CUDA compiler is found, in which case the compilation and installation of sketchlib will include GPU components:

```
-- Looking for a CUDA compiler
-- Looking for a CUDA compiler - /usr/local/cuda-11.1/bin/nvcc
-- CUDA found, compiling both GPU and CPU code
-- The CUDA compiler identification is NVIDIA 11.1.105
-- Detecting CUDA compiler ABI info
-- Detecting CUDA compiler ABI info - done
-- Check for working CUDA compiler: /usr/local/cuda-11.1/bin/nvcc - skipped
-- Detecting CUDA compile features
-- Detecting CUDA compile features - done
```

You can confirm that your custom installation of sketchlib is being used by checking the location of sketchlib library reported by `popppunk` points to your python site-packages, rather than the conda version.

11.2 Selecting a GPU

A single GPU will be selected on systems where multiple devices are available. For sketching and distance calculations, this can be specified by the `--deviceid` flag. Alternatively, all GPU-enabled functions will used device 0 by default. Any GPU can be set to device 0 using the system `CUDA_VISIBLE_DEVICES` variable, which can be set before running PopPUNK; e.g. to use GPU device 1:

```
export CUDA_VISIBLE_DEVICES=1
```

11.3 Using a GPU

By default, PopPUNK will use not use GPUs. To use them, you will need to add the flag `--gpu-sketch` (when constructing or querying a database using reads), `--gpu-dist` (when constructing or querying a database from assemblies or reads), or `--gpu-graph` (when querying or visualising a database, or fitting a model).

CHAPTER

TWELVE

TROUBLESHOOTING

This page deals with common issues in running the analysis. For issues with installing or running the software please raise an issue on github.

12.1 Most/all of my samples merge when I run a query

If you see a gigantic merge when running `poppunk_assign`, for example:

```
Clusters 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 have merged into 1_2_3_4_5_6_7_8_9_10_11_
↪12_13_14_15_16
```

This might be caused by:

- A single/a few 'bad' genome(s) with artificial zero/close-to-zero distances to many other samples, which links much of the network together.
- A boundary which is actually a bit lax, and when more isolates are added, they have distances just below the boundary between a few clusters, and there are enough of these to link more and more clusters together.

The first issue should be easy to fix with one of the following methods:

- Use `--run-qc` (see *Data quality control (--qc-db)*) with one or more of:
  - `--max-pi-dist` and/or `--max-a-dist` to remove outlier distances.
  - `--max-zero-dist` which sets the maximum proportion of zeros to other samples. Poor quality genomes often end up with zero distances to many samples, linking them together.
  - `--max-merge` which sets the maximum number of clusters a single query can cause to merge.
  - `--betweenness` which will print all queries in order which have high stress in the network, and are likely causing merges.
- Use `--serial` to run samples through one-by-one. This is a little less efficient than fully batched querying, but much faster than running independent jobs. Note, lineage models and updating the database are not supported with this mode.

The second issue above is potentially more insidiuous, and may require a refit to all the data to obtain a tighter boundary. You can (mostly) keep old cluster IDs via the use of `--external-clustering` if you do this. Alternatively, you can add the `--serial` command to type samples one at a time as above.

See issue 194 for more discussion.

12.2 Memory/run-time issues

Here are some tips based on experiences analysing larger datasets:

- Add `--threads` – they are used fairly efficiently throughout.
- Consider the `--gpu-sketch` and `--gpu-dists` options is applicable, and a GPU is available.
- In `--refine-model` set `--pos-shift 0` to avoid creating huge networks with close to N^2 edges. Mixture models normally need to be pruned.
- In `--refine-model` you may add the `--no-local` option to skip that step and decrease run-time, though gains are likely marginal.
- Use `--rapid-nj`, if producing microreact output.

Another option for scaling is to run `--create-db` with a smaller initial set (not using the `--full-db` command), then use `--assign-query` to add to this.

12.3 Known bugs

12.3.1 Calculating query-query distances when unnecessary

A bug in v2.4.0 only, (fixed in v2.5.0 and not in previous versions).

You will always see `Found novel query clusters. Calculating distances between them.` when running `poppunk_assign` with more than one input sample. This should only happen when unassigned isolates/novel clusters are found. Our check on this condition became invalid.

Additionally, this *may* have affected typing when query-query links were present, this appeared as invalid merges in some tests. If you used this, you may wish to re-run with v2.5.0 or higher.

12.3.2 Older HDBSCAN models fail to load

tl;dr if you see an error `ModuleNotFoundError: No module named 'sklearn.neighbors._dist_metrics'` you probably need to downgrade `sklearn` to v0.24.

The change in scikit-learn's API in v1.0.0 and above mean that HDBSCAN models fitted with `` `sklearn <=v0.24` `` will give an error when loaded. If you run into this, the solution is one of: - Downgrade sklearn to v0.24. - Run model refinement to turn your model into a boundary model instead (this will change clusters). - Refit your model in an environment with `` `sklearn >=v1.0` ``.

If this is a common problem let us know, as we could write a script to 'upgrade' HDBSCAN models. See issue [#213](https://github.com/bacpop/PopPUNK/issues/213) for more details.

12.3.3 When I look at my clusters on a tree, they make no sense

This is a bug caused by alphabetic sorting of labels in `PopPUNK >=v2.0.0` with `pp-sketchlib <v1.5.1`. There are three ways to fix this:

- Upgrade to `PopPUNK >=v2.2` and `pp-sketchlib >=v1.5.1` (preferred).
- Run `scripts/poppunk_pickle_fix.py` on your `.dists.pkl` file and re-run model fits.
- Create the database with `poppunk_sketch --sketch` and `poppunk_sketch --query` directly, rather than `PopPUNK --create-db`.

12.3.4 I have updated PopPUNK, and my clusters still seemed scrambled

This is possible using query assignment with `--update-db`, or in some cases with `--gpu-dists`. Please update to `PopPUNK >=v2.4.0` with `pp-sketchlib >=v1.7.0`

12.3.5 Calculating distances using 0 thread(s)

This will lead to an error later on in execution. This is due to a version mismatch between PopPUNK and `pp-sketchlib`. Installation of both packages via conda should keep the versions compatible, but there are ways they can get out of sync.

The solution is as above: upgrade to `PopPUNK >=v2.2` and `pp-sketchlib >=v1.5.1`.

12.4 Error/warning messages

12.4.1 Row name mismatch

PopPUNK may throw:

```
RuntimeError: Row name mismatch. Old: 6999_2#17.fa,6259_5#6.fa
New: 6952_7#16.fa,6259_5#6.fa
```

This is an error where the mash output order does not match the order in stored databases (`.pkl`). Most likely, the input files are from different runs, possibly due to using `--overwrite`. Run again, giving each step its own output directory.

12.4.2 Samples are missing from the final network

When running `--assign-query` an error such as:

```
WARNING: Samples 7553_5#54.fa,6999_5#1.fa are missing from the final network
```

Means that samples present in `--distances` and or `--ref-db` are not present in the loaded network. This should be considered an error as it will likely lead to other errors and warnings. Make sure the provided network is the one created by applying the `--model-dir` to `--distances`, and that the same output directory has not been used and overwriten by different steps or inputs.

12.4.3 Old cluster split across multiple new clusters

When running `--assign-query`, after distances have been calculated and queries are being assigned warnings such as:

```
WARNING: Old cluster 1 split across multiple new clusters
```

Mean that a single cluster in the original clustering is now split into more than one cluster. This means something has gone wrong, as the addition of new queries should only be able to merge existing clusters, not cause them to split.

Most likely, the `--previous-clustering` directory is inconsistent with the `--ref-db` and/or `--model-dir`. Make sure the clusters are those created from the network being used to assign new queries.

If you want to change cluster names or assign queries to your own cluster definitions you can use the `--external-clustering` argument instead.

CHAPTER

THIRTEEN

SCRIPTS

Brief documentation on the helper scripts included in the package in the `/scripts` directory. To use these scripts you will need to have a clone of the git repository, or they should also be installed with the prefix 'poppunk' (e.g to run `extract_distances.py`, run the command `poppunk_extract_distances.py`).

13.1 Easy run mode

Previous versions of the software had an `--easy-run` mode which would run a pipeline of:

- `--create-db` to sketch genomes
- `--fit-model --dbscan` to fit a flexible model
- `--refine-model` to improve this model

This is now available as `poppunk_easy_run.py` which will chain calls to `poppunk` and `poppunk_visualise` to replicate this functionality.

13.2 Iterative PopPUNK

You can combine the output from multiple to produce further analysis. For an easy way to create multiple clusters, try the `--multi-boundary` option (*Running with multiple boundary positions*).

The script to analyse these is `poppunk_iterate.py`. Basic use is to provide the output directory as `--db`, but run `--help` for other common options. This relies on finding files named <db>/<db>_boundary<n>_clusters.csv, where <n> is the boundary iteration number (continuous integers increasing from zero). Clusters must contain at least two samples.

This script will do the following:

1. Starting from the most specific clusters (nearest the origin), it will iteratively add new clusters which are either:
    a) totally new clusters
    b) subsets of existing clusters
    c) existing clusters are subsets of the new cluster.
2. Remove duplicate clusters.
3. Calculate average core distance within this cluster set.
4. Create a tree by nesting smaller clusters within larger clusters they are subsets of.
5. Output the combined clusters, average core distances, and tree.

6. Cut this tree to pick a set of clusters under a similarity given by `--cutoff`.

13.3 Adding weights to the network

Converts binary within-cluster edge weights to the Euclidean core-accessory distance. This is equivalent to running with `--graph-weights`:

```
poppunk_add_weights <name>_graph.gt <name>.dists <output>
```

Default output is a graph-tool file. Add `--graphml` to save as .graphml instead.

13.4 Writing the pairwise distances to an output file

By default PopPUNK does not write the calculated π_n and a distances out, as this contains $\frac{1}{2}n * (n-1)$ rows, which gives a multi Gb file for large datasets.

However, if needed, there is a script available to extract these distances as a text file:

```
poppunk_extract_distances.py --distances strain_db.dists --output strain_db.dists.out
```

13.5 Writing network components to an output file

Visualisation of large networks with cytoscape may become challenging. It is possible to extract individual components/clusters for visualisation as follows:

```
poppunk_extract_components.py strain_db_graph.gpickle strain_db
```

13.6 Calculating Rand indices

This script allows the clusters formed by different runs/fits/modes of PopPUNK to be compared to each other. 0 indicates the clusterings are totally discordant, and 1 indicates they are identical.

Run:

```
poppunk_calculate_rand_indices.py --input poppunk_gmm_clusters.csv,poppunk_dbscan_
↪cluster.csv
```

The script will calculate the Rand index and the adjusted Rand index between all pairs of files provided (comma separated) to the `--input` argument. These will be written to the file `rand.out`, which can be changed using `--output`.

The `--subset` argument can be used to restrict comparisons to include only specific samples listed in the provided file.

13.7 Calculating silhouette indices

This script can be used to find how well the clusters project into core-accessory space by calculating the silhoutte index, which measures how close samples are to others in their own cluster compared to samples from other clusters. The silhoutte index is calculated for every sample and takes a value between -1 (poorly matched) to +1 (well matched). The script reports the average of these indices across all samples, using Euclidean distances between the (normalised) core and accessory divergences calculated by PopPUNK.

To run:

```
poppunk_calculate_silhouette.py --distances strain_db.dists --cluster-csv strain_db_
↪clusters.csv
```

The following additional options are available for use with external clusterings (e.g. from hierBAPS):

- `--cluster-col` the (1-indexed) column index containing the cluster assignment
- `--id-col` the (1-indexed) column index containing the sample names
- `--sub` a string to remove from sample names to match them to those in `--distances`

13.8 Distributing PopPUNK models

This script automatically generates compressed and uncompressed directories containing all files required for distribution and reuse of PopPUNK model fits.

To run:

```
python poppunk_distribute_fit.py --dbdir database_directory --fitdir model_fit_directory␣
↪--outpref output_prefix
```

The following additional arguments are available:

- `--lineage` specify only if lineage fit was used.
- `--no-compress` will not generate tar.bz2 archives

`--dbdir` and `--fitdir` can be the same directory, however both must still be specified. The output of this script is a directory and a compressed tar.bz2 archive for each of the full dataset and representative genomes dataset.

CHAPTER

FOURTEEN

ITERATIVE POPPUNK

14.1 Running with multiple boundary positions

To create clusters at equally spaced positions across the refinement range, add the `--multi-boundary <n>` argument, with the number of positions specified by `<n>`. This will create up to `<n>` sets of clusters, with boundaries equally spaced between the origin and the refined boundary position.

Trivial cluster sets, where every sample is in its own cluster, will be excluded, so the final number of clusters may be less than `<n>`. The script to analyse these is `poppunk_iterate.py`. Basic usage is to provide the output directory as `--db`, but run `--help` for other common options. This replies on finding files named `<db>/<db>_boundary<n>_clusters.csv`, where `<n>` is the boundary iteration number (continuous integers increasing from zero). Clusters must contain at least two samples.

The `poppunk_iterate.py` script performs the following steps:

1. Starting from the most specific clusters (nearest the origin), it will iteratively add new clusters which are either:
    a. Total new clusters
    b. Subsets of existing clusters
    c. Existing clusters are subset of the new cluster.
2. Remove duplicate clusters.
3. Calculate average core distance within this cluster set.
4. Create a tree by nesting smaller clusters within larger clusters whey are subsets of.
5. Output the combined clusters, average core distances, and tree.
6. Cut this tree to pick a set of clusters under a similarity given by a `--cutoff`.

14.2 Step-by-Step Tutorial

Step 1: Sketching (–create-db)

First, use `poppunk --create-db` to sketch input data and calculate distances between samples. For the details, refer to *Sketching (--create-db)*. To run this, using the following command:

```
poppunk --create-db --r-files rlist.txt --output <database> --threads 8
```

Step 2: Initial model fitting (–fit-model)

In iterative-PopPUNK, a universal model recommended for fitting the initial model is GMM with 2 or 3 components (K=2 or K=3), because more datasets can be analysed using this setting (DBSCAN fits sometimes fail to coverage). The details for model fitting can be found in *Fitting new models (--fit-model)*. To run this, using the following command:

```
poppunk --fit-model bgmm --K 2 --ref-db <database> --output <database> --threads 16
```

Step 3: Multi-level clustering by moving decision boundary iteratively (–fit-model refine –multi-boundary)

After fitting the initial model with GMM (with 2 or 3 components), refine it by moving the decision boundary to multiple positions between the origin and the combined decision boundary using the `--multi-boundary` option. To expand within-strain component, use `--neg-shift`. Details can be found in *Fitting new models (--fit-model)* (the **refine** section). To run this, use the following command:

```
poppunk --fit-model refine --multi-boundary 30 --ref-db <database> --output <database> --
↪threads 16
```

Step 4: Choosing clusters under given similarity cutoffs (poppunk_iterate.py)

With <n> sets of clusters created in Step 3, `poppunk_iterate.py` is used to assemble a hierarchical tree of iterative-PopPUNK clusters, while also calculating the average core distance within each cluster set. To run this step, use the following command:

```
poppunk_iterate.py --db <database> --cutoff 0.2  --output <prefix> --cpus 16
```

Outputs:

- <prefix>_iterate.tree.nwk: this is the hierarchical tree of iterative-PopPUNK clusters.
- <prefix>_iterate.clusters.csv: this file contains all cluster sets from <n> positions, along with their corresponding average core distances.
- <prefix>_iterative.cutoff_clusters.csv: this file contains a single set of clusters that meets a specified similarity cutoff.

Note: We recommend using same name for <database> as in steps 1-3 (`--ref-db` and `--output`) to save the outputs from each step in a same folder

---

14.3 Examples

The following example demonstrate how to use iterative-PopPUNK with 500 *E.coli* representative genomes from Horesh et al. 2021. You can download the genomes at https://doi.org/10.6084/m9.figshare.13270073.

Step 1:

```
paste <(ls *fa) <(ls *fa) > rlist.txt
poppunk --create-db --r-files rlist.txt --output ecoli --threads 16
```

This will create a PopPUNK database named “ecoli” by sketching the input genomes using 16 threads. The program will calculate random match chances using Monte Carlo and calculate distances using 16 threads. Once complete, the sketches will be written to a local file.

Step 2

Next, fit a Bayesian Gaussian Mixture Model (bgmm) to reference database using the following commands:

```
poppunk --fit-model bgmm --K 2 --ref-db ecoli --output ecoli --threads 16
```

This will fit the bgmm model to the `ecoli` database, assign distances with the model, and summarize the fit. The output will include the scaled component means and network summary.

Step 3

After fitting the bgmm model, refine the model using the following commands:

```
poppunk --fit-model refine --ref-db ecoli --output ecoli --multi-boundary 30 --threads 16
```

This will load the previous bgmm model, construct an initial model-based network, and optimize the score globally. The program will then create multiple boundary fits and summarize the network. The output will include the components, density, transitivity, mean betweenness, weighted-mean betweenness, score, score with betweenness and score with weighted-betweenness.

Step 4

Finally, run `poppunk_iterate.py` with the following command to iterate over the PopPUNK analysis:

```
poppunk_iterate.py --db ecoli --cpus 16 --cutoff 0.2 --output 0.2_ecoli
```

This will run iteraive-PopPUNK with a cutoff 0.2 and output the results to files with prefix `0.2_ecoli`.

14.4 Common questions

1. How can I use Iterative-PopPUNK to achieve sublineage clustering from a large outbreak dataset?

To demonstrate how to achieve multi-level clustering from a large outbreak setting using Iterative-PopPUNK, we will use another example dataset consisting of 2,640 pathogenic *Vibrio parahaemolyticus* genomes from Yang et al. 2022.

Step 1: Creating a PopPUNK database

First, we need to create a reference database and fit a PopPUNK model using the following commands:

```
paste <(ls *fa) <(ls *fa) > rlist.txt
poppunk --create-db --r-files rlist.txt --output VP --threads 16
poppunk --fit-model bgmm --K 2 --ref-db VP --output VP --threads 16
poppunk --fit-model refine --ref-db VP --output VP --multi-boundary 30 --threads 16 --
↪neg-shift -1
poppunk_iterate.py --db VP --cpus 16 --cutoff 0.5 --output 0.5_VP
```

After running Step 1, iterative-PopPUNK produces a result file `0.5_iterative.cutoff_clusters.csv`, which gives 9 PopPUNK clusters exactly corresponding to 9 clonal groups described in Yang et al. 2022.

Note: For closely related genomes (e.g. genomes within clonal group), we need to increase the sketch size for more precise distance calculation. In this example, we increase the sketching size to 100000. For more information, please refer to sketching section for details

---

Step 2: Sublineage clustering for clonal groups

Next, we extract the clonal groups estimated by Iterative-PoPUNK in Step 1 and run Iterative-PopPUNK again for sublineage clustering. For example, we use the two largest clonal groups estimated in Step 1 for downstream analysis.

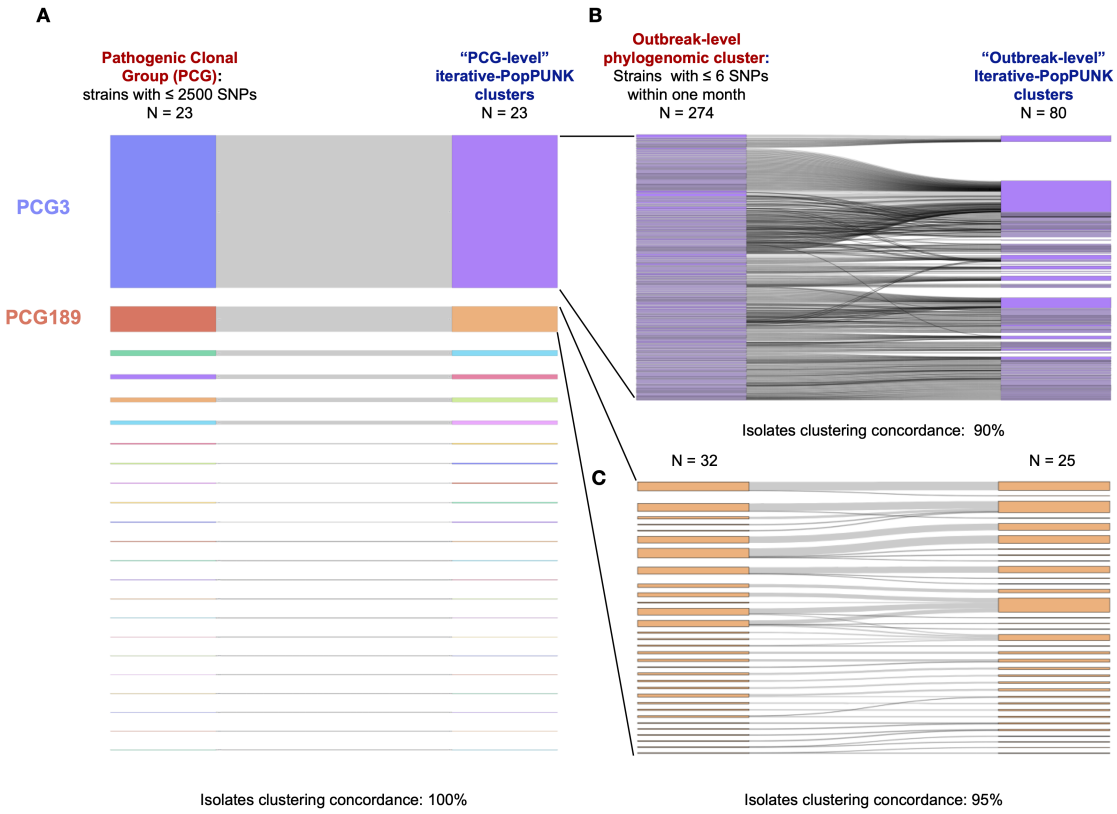
Sublineage clustering for clonal group CG3:

```
poppunk --create-db --r-files cg3_rlist.txt --output VP_cg3 --threads 16 --sketch-size␣
↪100000
poppunk --fit-model bgmm --K 2 --ref-db VP_cg3 --output VP_cg3 --threads 16
poppunk --fit-model refine --ref-db VP_cg3 --output VP_cg3 --multi-boundary 30 --threads␣
↪16
poppunk_iterate.py --db VP --cpus 16 --cutoff 0.4 --output 0.4_VP_cg3
```

Sublineage clustering for clonal group CG189:

```
poppunk --create-db --r-files cg189_rlist.txt --output VP_cg189 --threads 16 --sketch-
↪size 100000
poppunk --fit-model bgmm --K 2 --ref-db VP_cg189 --output VP_cg189 --threads 16
poppunk --fit-model refine --ref-db VP_cg189 --output VP_cg189 --multi-boundary 30 --
↪threads 16
poppunk_iterate.py --db VP --cpus 16 --cutoff 0.2 --output 0.2_VP_cg189
```

Results:



The results demonstrate the effectiveness of using iterative-PopPUNK for sublineage clustering in large outbreak datasets. For example, in clonal group CG3, we extracted 274 outbreak groups (after removal of non-pathogenic isolates), while iterative-PopPUNK identified 80 clusters with a 40% MACD cutoff. The concordance of iterative-PopPUNK clustering with the identified outbreak groups was 60% (48/80), indicating that iterative-PopPUNK is able to achieve a finer resolution than clonal level. Moreover, the isolate clustering concordance (i.e. isolates assigned to a same group by both methods) was 90% (1596/1768), indicating high agreement between iterative-PopPUNK and outbreak groups at the isolate level. For clonal group CG189, iterative-PopPUNK identified 25 clusters with a 20% MACD

cutoff, which showed a cluster concordance of 80% (20/25) with outbreak groups, and an isolate clustering concordance of 95% (259/273). It is worth noting that different genetic distance measurements can account for inconsistencies between iterative-PopPUNK clusters and outbreak groups.

2. How can I determine when I should add "–neg-shift" or "–pos-shift" ?

Iterative-PopPUNK allows for the use of "–neg-shift" and "–pos-shift" parameters to refine the initial model fit and estimate more clusters in either a higher or lower genetic level. These parameters adjust the decision boundary ranges for cluster estimation.

By default, adding "–neg-shift" with a value below zero moves the initial decision boundary line towards a higher genetic level, while a positive value for "–pos-shift" moves the line towards the origin. The value chosen for these parameters should align with the research interests of the user.

For instance, consider the E. coli dataset. Without adding "–neg-shift", iterative-PopPUNK may fail to find phylogroups. Figure 1 shows the distribution plot for this dataset with default settings, while Figure 2 shows the same plot after adding a value of -0.25 for "–neg-shift":

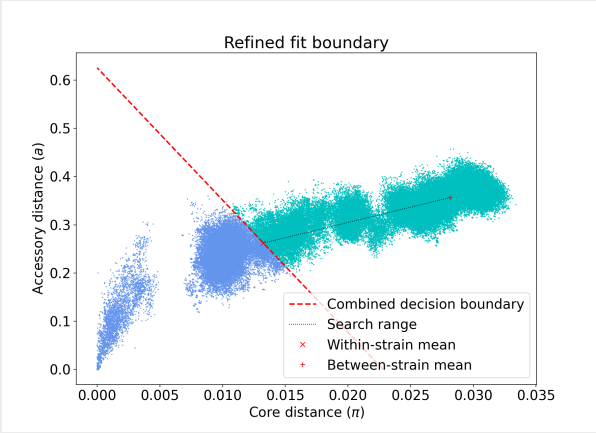


Fig. 1: Figure 1: refine fit with default settings

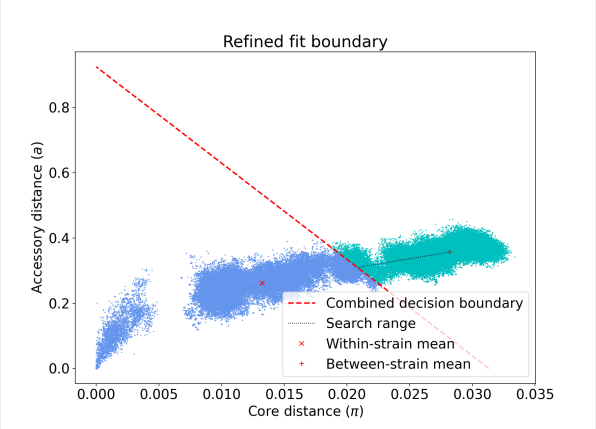


Fig. 2: Figure 2: refine fit with default settings after adding -0.25 neg-shift

In summary, users should determine whether to add "–neg-shift" or "–pos-shift" based on their research interests, and adjust the value accordingly.

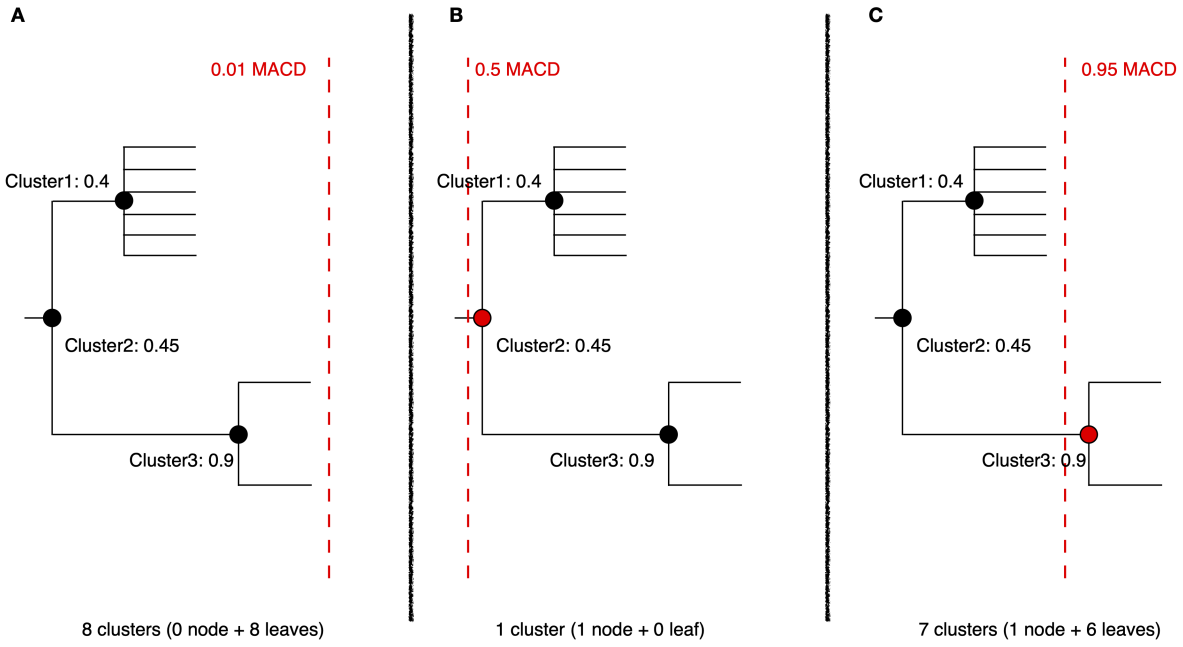
3. why does the number of clusters increase with larger cutoff values?

In iterative PopPUNK, the choice of cutoff value has a significant impact on the resulting number and composition of clusters. In general, increasing the cutoff value leads to fewer and larger clusters, while decreasing the cutoff value leads to more and smaller clusters. However, in some cases, increasing the cutoff value can paradoxically lead to an increase in the number of clusters, which may seem counterintuitive.

To understand why this happens, we need to look at the role of average core distance (ACD) in cluster formation. ACD is a measure of the average genetic distance between all pairs of isolates within a cluster. When the ACD is low, it means that the isolates within the cluster are genetically similar to each other, while a high ACD indicates that there are significant genetic differences between the isolates.

In a large dataset, the ACD of one cluster's parent cluster is greatly affected by small clusters or singleton isolates. This is because the ACD of a parent cluster is calculated as the average ACD of all the clusters it contains. As a result, if a parent cluster contains one or more small clusters with extreme high ACD values, its own ACD value will be inflated, which may cause it to be split into multiple singletons at higher cutoff values.

Consider the following simplified iterative-PopPUNK tree:



In Figure A, a very small cutoff value results in no nodes being selected, leaving a set of 8 clusters or singletons. When the cutoff is increased to 0.5 in Figure B, only one cluster, Cluster2, is selected. However, when a higher cutoff value of 0.95 is adopted, Cluster3 is selected, and the remaining six isolates are left as singletons, resulting in a total of 7 clusters.

To address this issue, you can check the "<prefix>.clusters.csv" file to identify small clusters with extreme high ACD values. If present, check the PopPUNK distribution plot ("<prefix>_distanceDistribution.png") to determine if there are any distanced components. You can remove low-quality or distanced samples from your dataset using the "–qc-db" option (please refer to *Data quality control (--qc-db)*).

In the examples given above, removing the two isolates from Cluster3 would help to solve the problem and lead to a more accurate clustering result.

By carefully selecting the cutoff value and performing quality control on the input data, you can obtain robust and biologically meaningful clusterings with iterative PopPUNK.

CHAPTER

FIFTEEN

CITING POPPUNK

If you use PopPUNK, PopPIPE or pp-sketchlib in a scientific paper, we would appreciate a citation. As a minimum, please cite the following paper(s):

Lees JA, Harris SR, Tonkin-Hill G, Gladstone RA, Lo SW, Weiser JN, Corander J, Bentley SD, Croucher NJ. Fast and flexible bacterial genomic epidemiology with PopPUNK. *Genome Research* **29**:1-13 (2019). doi:10.1101/gr.241455.118

15.1 Generating citations and methods

You can add `--citation` to your PopPUNK command to generate a full list of papers to cite. This will also produce a basic methods paragraph for you to edit and include. You can do this after running `poppunk_assign` or `poppunk --fit-model`:

```
poppunk --citation --fit-model bgmm --ref-db example_db --K 4
```

gives:

> We built a database of 28 isolates using pp-sketchlib version 1.7.0 (doi:10.5281/zenodo.4531418) with sketch version 88ee3ff83ba294c928505f991e20078691ed090e, k-mer lengths 13-28, a sketch size of 9984 and dense seeds [6-8]. We assigned variable-length-k-mer clusters (VLKCs) using PopPUNK version 2.4.0 (doi:10.1101/gr.241455.118) by fitting a BGMM with 4 components [1-5].

or:

```
poppunk_assign --citation --query some_queries.txt --db example_db
```

gives:

> We queried a database of 28 isolates and their pre-assigned variable-length-k-mer clusters (VLKCs) using pp-sketchlib version 1.7.0 (doi:10.5281/zenodo.4531418) with sketch version 88ee3ff83ba294c928505f991e20078691ed090e, k-mer lengths 13-28, a sketch size of 9984 and dense seeds [6-8]. We assigned the VLKCs using PopPUNK version 2.4.0 (doi:10.1101/gr.241455.118) [1-5].

If your journal requires versions for all software packages, you may find running `conda list` helpful. Running `poppunk_info` (*Viewing information about a database*) on your `.h5` files will give useful information too.

CHAPTER

SIXTEEN

REFERENCE DOCUMENTATION

Documentation for module functions (for developers)

16.1 assign.py

poppunk_assign main function

PopPUNK.assign.**assign_query**(*dbFuncs*, *ref_db*, *q_files*, *output*, *qc_dict*, *update_db*, *write_references*, *distances*, *serial*, *threads*, *overwrite*, *plot_fit*, *graph_weights*, *model_dir*, *strand_preserved*, *previous_clustering*, *external_clustering*, *core*, *accessory*, *gpu_sketch*, *gpu_dist*, *gpu_graph*, *deviceid*, *save_partial_query_graph*)

Code for assign query mode for CLI

PopPUNK.assign.**assign_query_hdf5**(*dbFuncs*, *ref_db*, *qNames*, *output*, *qc_dict*, *update_db*, *write_references*, *distances*, *serial*, *threads*, *overwrite*, *plot_fit*, *graph_weights*, *model_dir*, *strand_preserved*, *previous_clustering*, *external_clustering*, *core*, *accessory*, *gpu_dist*, *gpu_graph*, *save_partial_query_graph*)

Code for assign query mode taking hdf5 as input. Written as a separate function so it can be called by web APIs

PopPUNK.assign.**main**()

Main function. Parses cmd line args and runs in the specified mode.

16.2 bgmm.py

Functions used to fit the mixture model to a database. Access using *BGMMFit*. BGMM using sklearn

PopPUNK.bgmm.**findWithinLabel**(*means*, *assignments*, *rank=0*)

Identify within-strain links

Finds the component with mean closest to the origin and also akes sure some samples are assigned to it (in the case of small weighted components with a Dirichlet prior some components are unused)

Args:

means (numpy.array)
K x 2 array of mixture component means

assignments (numpy.array)
Sample cluster assignments

rank (int)
Which label to find, ordered by distance from origin. 0-indexed.

(default = 0)

Returns:

within_label (int)
The cluster label for the within-strain assignments

PopPUNK.bgmm.**fit2dMultiGaussian**(*X*, *dpgmm_max_K=2*)

Main function to fit BGMM model, called from *fit()*

Fits the mixture model specified, saves model parameters to a file, and assigns the samples to a component. Write fit summary stats to STDERR.

Args:

X (np.array)
n x 2 array of core and accessory distances for n samples. This should be subsampled to 100000 samples.

dpgmm_max_K (int)
Maximum number of components to use with the EM fit. (default = 2)

Returns:

dpgmm (sklearn.mixture.BayesianGaussianMixture)
Fitted bgmm model

PopPUNK.bgmm.**log_likelihood**(*X*, *weights*, *means*, *covars*, *scale*)

modified sklearn GMM function predicting distribution membership

Returns the mixture LL for points X. Used by `assign_samples()` and *plot_contours()*

Args:

X (numpy.array)
n x 2 array of core and accessory distances for n samples

weights (numpy.array)
Component weights from *fit2dMultiGaussian()*

means (numpy.array)
Component means from *fit2dMultiGaussian()*

covars (numpy.array)
Component covariances from *fit2dMultiGaussian()*

scale (numpy.array)
Scaling of core and accessory distances from *fit2dMultiGaussian()*

Returns:

logprob (numpy.array)
The log of the probabilities under the mixture model

lpr (numpy.array)
The components of the log probability from each mixture component

PopPUNK.bgmm.**log_multivariate_normal_density**(*X*, *means*, *covars*, *min_covar=1e-07*)

Log likelihood of multivariate normal density distribution

Used to calculate per component Gaussian likelihood in `assign_samples()`

Args:

X (numpy.array)
n x 2 array of core and accessory distances for n samples

means (numpy.array)
Component means from *fit2dMultiGaussian()*

covars (numpy.array)
Component covariances from *fit2dMultiGaussian()*

min_covar (float)
Minimum covariance, added when Choleksy decomposition fails due to too few observations (default = 1.e-7)

Returns:

log_prob (numpy.array)
An n-vector with the log-likelihoods for each sample being in this component

16.3 dbscan.py

Functions used to fit DBSCAN to a database. Access using *DBSCANFit*. DBSCAN using hdbscan

PopPUNK.dbscan.**evaluate_dbscan_clusters**(*model*)

Evaluate whether fitted dbscan model contains non-overlapping clusters

Args:

model (DBSCANFit)
Fitted model from *fit()*

Returns:

indistinct (bool)
Boolean indicating whether putative within- and between-strain clusters of points overlap

PopPUNK.dbscan.**findBetweenLabel**(*assignments*, *within_cluster*)

Identify between-strain links from a DBSCAN model

Finds the component containing the largest number of between-strain links, excluding the cluster identified as containing within-strain links.

Args:

assignments (numpy.array)
Sample cluster assignments

within_cluster (int)
Cluster ID assigned to within-strain assignments, from *findWithinLabel()*

Returns:

between_cluster (int)
The cluster label for the between-strain assignments

PopPUNK.dbscan.**fitDbScan**(*X*, *min_samples*, *min_cluster_size*, *cache_out*)

Function to fit DBSCAN model as an alternative to the Gaussian

Fits the DBSCAN model to the distances using hdbscan

Args:

X (np.array)
n x 2 array of core and accessory distances for n samples

min_samples (int)
Parameter for DBSCAN clustering 'conservativeness'

min_cluster_size (int)
Minimum number of points in a cluster for HDBSCAN

cache_out (str)
Prefix for DBSCAN cache used for refitting

Returns:

hdb (hdbscan.HDBSCAN)
Fitted HDBSCAN to subsampled data

labels (list)
Cluster assignments of each sample

n_clusters (int)
Number of clusters used

16.4 mandrake.py

PopPUNK.mandrake.**generate_embedding**(*seqLabels*, *accMat*, *perplexity*, *outPrefix*, *overwrite*, *kNN=50*, *maxIter=10000000*, *n_threads=1*, *use_gpu=False*, *device_id=0*)

Generate t-SNE projection using accessory distances

Writes a plot of t-SNE clustering of accessory distances (.dot)

Args:

seqLabels (list)
Processed names of sequences being analysed.

accMat (numpy.array)
n x n array of accessory distances for n samples.

perplexity (int)
Perplexity parameter passed to t-SNE

outPrefix (str)
Prefix for all generated output files, which will be placed in *outPrefix* subdirectory

overwrite (bool)
Overwrite existing output if present (default = False)

kNN (int)
Number of neigbours to use with SCE (cannot be > n_samples) (default = 50)

maxIter (int)
Number of iterations to run (default = 1000000)

n_threads (int)
Number of CPU threads to use (default = 1)

use_gpu (bool)
Whether to use GPU libraries

device_id (int)
Device ID of GPU to be used (default = 0)

Returns:

mandrake_filename (str)
Filename with .dot of embedding

16.5 models.py

Classes used for model fits

`class` PopPUNK.models.**BGMMFit**(*outPrefix*, *max_samples=50000*)

Class for fits using the Gaussian mixture model. Inherits from *ClusterFit*.

Must first run either *fit()* or *load()* before calling other functions

Args:

outPrefix (str)
The output prefix used for reading/writing

max_samples (int)
The number of subsamples to fit the model to

(default = 100000)

assign(*X*, *values=False*, *progress=True*)

Assign the clustering of new samples using `assign_samples()`

Args:

X (numpy.array)
Core and accessory distances

values (bool)
Return the responsibilities of assignment rather than most likely cluster

num_processes (int)
Number of threads to use

progress (bool)
Show progress bar

[default = True]

Returns:

y (numpy.array)
Cluster assignments or values by samples

fit(*X*, *max_components*)

Extends *fit()*

Fits the BGMM and returns assignments by calling *fit2dMultiGaussian()*.

Fitted parameters are stored in the object.

Args:

X (numpy.array)
The core and accessory distances to cluster. Must be set if preprocess is set.

max_components (int)
Maximum number of mixture components to use.

Returns:

y (numpy.array)
Cluster assignments of samples in X

`load`(*fit_npz*, *fit_obj*)

Load the model from disk. Called from *loadClusterFit()*

Args:

fit_npz (dict)
Fit npz opened with `numpy.load()`

fit_obj (sklearn.mixture.BayesianGaussianMixture)
The saved fit object

`plot`(*X*, *y*)

Extends *plot()*

Write a summary of the fit, and plot the results using *PopPUNK.plot.plot_results()* and *PopPUNK.plot.plot_contours()*

Args:

X (numpy.array)
Core and accessory distances

y (numpy.array)
Cluster assignments from *assign()*

`save`()

Save the model to disk, as an npz and pkl (using outPrefix).

`class` PopPUNK.models.`ClusterFit`(*outPrefix*, *default_dtype=numpy.float32*)

Parent class for all models used to cluster distances

Args:

outPrefix (str)
The output prefix used for reading/writing

`copy`(*prefix*)

Copy the model to a new directory

`fit`(*X=None*)

Initial steps for all fit functions.

Creates output directory. If preprocess is set then subsamples passed X

Args:

X (numpy.array)
The core and accessory distances to cluster. Must be set if preprocess is set.

(default = None)

default_dtype (numpy dtype)
Type to use if no X provided

`no_scale()`

Turn off scaling (useful for refine, where optimization is done in the scaled space).

`plot(X=None)`

Initial steps for all plot functions.

Ensures model has been fitted.

Args:

X (numpy.array)
The core and accessory distances to subsample.

(default = None)

`class PopPUNK.models.DBSCANFit(outPrefix, max_samples=100000)`

Class for fits using HDBSCAN. Inherits from *ClusterFit*.

Must first run either *fit()* or *load()* before calling other functions

Args:

outPrefix (str)
The output prefix used for reading/writing

max_samples (int)
The number of subsamples to fit the model to

(default = 100000)

`assign(X, no_scale=False, progress=True)`

Assign the clustering of new samples using `assign_samples_dbscan()`

Args:

X (numpy.array)
Core and accessory distances

no_scale (bool)
Do not scale X

[default = False]

progress (bool)
Show progress bar

[default = True]

Returns:

y (numpy.array)
Cluster assignments by samples

`fit(X, max_num_clusters, min_cluster_prop)`

Extends *fit()*

Fits the distances with HDBSCAN and returns assignments by calling *fitDbScan()*.

Fitted parameters are stored in the object.

Args:

X (numpy.array)
The core and accessory distances to cluster. Must be set if preprocess is set.

max_num_clusters (int)
Maximum number of clusters in DBSCAN fitting

min_cluster_prop (float)
Minimum proportion of points in a cluster in DBSCAN fitting

Returns:

y (numpy.array)
Cluster assignments of samples in X

`load`(*fit_npz*, *fit_obj*)

Load the model from disk. Called from *loadClusterFit()*

Args:

fit_npz (dict)
Fit npz opened with `numpy.load()`

fit_obj (hdbscan.HDBSCAN)
The saved fit object

`plot`(*X=None*, *y=None*)

Extends *plot()*

Write a summary of the fit, and plot the results using *PopPUNK.plot.plot_dbscan_results()*

Args:

X (numpy.array)
Core and accessory distances

y (numpy.array)
Cluster assignments from *assign()*

`save`()

Save the model to disk, as an npz and pkl (using outPrefix).

`class PopPUNK.models.LineageFit`(*outPrefix*, *ranks*, *max_search_depth*, *reciprocal_only*, *count_unique_distances*, *dist_col=None*, *use_gpu=False*)

Class for fits using the lineage assignment model. Inherits from *ClusterFit*.

Must first run either *fit()* or *load()* before calling other functions

Args:

outPrefix (str)
The output prefix used for reading/writing

ranks (list)
The ranks used in the fit

`assign`(*rank*)

Get the edges for the network. A little different from other methods, as it doesn't go through the long form distance vector (as coo_matrix is basically already in the correct gt format)

Args:

rank (int)
Rank to assign at

Returns:

y (list of tuples)
Edges to include in network

`edge_weights`(*rank*)

Get the distances for each edge returned by assign

Args:

rank (int)
Rank assigned at

Returns:

weights (list)
Distance for each assignment

`extend`(*qqDists*, *qrDists*)

Update the sparse distance matrix of nearest neighbours after querying

Args:

qqDists (numpy or cupy ndarray)
Two column array of query-query distances

qqDists (numpy or cupy ndarray)
Two column array of reference-query distances

Returns:

y (list of tuples)
Edges to include in network

`fit`(*X*, *accessory*)

Extends *fit()*

Gets assignments by using nearest neigbours.

Args:

X (numpy.array)
The core and accessory distances to cluster. Must be set if preprocess is set.

accessory (bool)
Use accessory rather than core distances

Returns:

y (numpy.array)
Cluster assignments of samples in X

`load`(*fit_npz*, *fit_obj*)

Load the model from disk. Called from *loadClusterFit()*

Args:

fit_npz (dict)
Fit npz opened with `numpy.load()`

fit_obj (sklearn.mixture.BayesianGaussianMixture)
The saved fit object

plot(*X*, *y=None*)

Extends *plot()*

Write a summary of the fit, and plot the results using *PopPUNK.plot.plot_results()* and *PopPUNK.plot.plot_contours()*

Args:

X (numpy.array)
Core and accessory distances

y (any)
Unused variable for compatibility with other plotting functions

save()

Save the model to disk, as an npz and pkl (using outPrefix).

class PopPUNK.models.**NumpyShared**(*name*, *shape*, *dtype*)

dtype

Alias for field number 2

name

Alias for field number 0

shape

Alias for field number 1

class PopPUNK.models.**RefineFit**(*outPrefix*)

Class for fits using a triangular boundary and network properties. Inherits from *ClusterFit*.

Must first run either *fit()* or *load()* before calling other functions

Args:

outPrefix (str)
The output prefix used for reading/writing

apply_threshold(*X*, *threshold*)

Applies a boundary threshold, given by user. Does not run optimisation.

Args:

X (numpy.array)
The core and accessory distances to cluster. Must be set if preprocess is set.

threshold (float)
The value along the x-axis (core distance) at which to draw the assignment boundary

Returns:

y (numpy.array)
Cluster assignments of samples in X

assign(*X*, *slope=None*)

Assign the clustering of new samples

Args:

X (numpy.array)
Core and accessory distances

slope (int)
Override self.slope. Default - use self.slope Set to 0 for a vertical line, 1 for a horizontal line, or 2 to use a slope

Returns:

y (numpy.array)
Cluster assignments by samples

`fit`(*X*, *sample_names*, *model*, *max_move*, *min_move*, *startFile=None*, *indiv_refine=False*, *unconstrained=False*, *multi_boundary=0*, *score_idx=0*, *no_local=False*, *betweenness_sample=100*, *use_gpu=False*)

Extends *fit()*

Fits the distances by optimising network score, by calling `refineFit2D()`.

Fitted parameters are stored in the object.

Args:

X (numpy.array)
The core and accessory distances to cluster. Must be set if preprocess is set.

sample_names (list)
Sample names in X (accessed by *iterDistRows()*)

model (ClusterFit)
The model fit to refine

max_move (float)
Maximum distance to move away from start point

min_move (float)
Minimum distance to move away from start point

startFile (str)
A file defining an initial fit, rather than one from `--fit-model`. See documentation for format. (default = None).

indiv_refine (str)
Run refinement for core or accessory distances separately (default = None).

multi_boundary (int)
Produce cluster output at multiple boundary positions downward from the optimum. (default = 0).

unconstrained (bool)
If True, search in 2D and change the slope of the boundary

score_idx (int)
Index of score from *networkSummary()* to use [default = 0]

no_local (bool)
Turn off the local optimisation step. Quicker, but may be less well refined.

betweenness_sample (int)
Number of sequences per component used to estimate betweenness using a GPU. Smaller numbers are faster but less precise [default = 100]

use_gpu (bool)
Whether to use cugraph for graph analyses

Returns:

y (numpy.array)
Cluster assignments of samples in X

`load`(*fit_npz*, *fit_obj*)

Load the model from disk. Called from *loadClusterFit()*

Args:

fit_npz (dict)
Fit npz opened with `numpy.load()`

fit_obj (None)
The saved fit object (not used)

`plot`(*X*, *y=None*)

Extends *plot()*

Write a summary of the fit, and plot the results using *PopPUNK.plot.plot_refined_results()*

Args:

X (numpy.array)
Core and accessory distances

y (numpy.array)
Assignments (unused)

`save`()

Save the model to disk, as an npz and pkl (using outPrefix).

PopPUNK.models.`assign_samples`(*chunk*, *X*, *y*, *model*, *scale*, *chunk_size*, *values=False*)

Runs a models assignment on a chunk of input

Args:

chunk (int)
Index of chunk to process

X (NumpyShared)
n x 2 array of core and accessory distances for n samples

y (NumpyShared)
An n-vector to store results, with the most likely cluster memberships or an n by k matrix with the component responsibilities for each sample.

weights (numpy.array)
Component weights from *BGMMFit*

means (numpy.array)
Component means from *BGMMFit*

covars (numpy.array)
Component covariances from *BGMMFit*

scale (numpy.array)
Scaling of core and accessory distances from *BGMMFit*

chunk_size (int)
Size of each chunk in X

values (bool)
Whether to return the responsibilities, rather than the most likely assignment (used for entropy calculation).

Default is False

PopPUNK.models.**loadClusterFit**(*pkl_file*, *npz_file*, *outPrefix=''*, *max_samples=100000*, *use_gpu=False*)

Call this to load a fitted model

Args:

pkl_file (str)
Location of saved .pkl file on disk

npz_file (str)
Location of saved .npz file on disk

outPrefix (str)
Output prefix for model to save to (e.g. plots)

max_samples (int)
Maximum samples if subsampling X [default = 100000]

use_gpu (bool)
Whether to load npz file with GPU libraries for lineage models

Returns:

load_obj (model)
Loaded model

16.6 network.py

Functions used to construct the network, and update with new queries. Main entry point is `constructNetwork()` for new reference databases, and `findQueryLinksToNetwork()` for querying databases. Network functions

PopPUNK.network.**addQueryToNetwork**(*dbFuncs*, *rList*, *qList*, *G*, *assignments*, *model*, *queryDB*, *kmers=None*, *distance_type='euclidean'*, *queryQuery=False*, *strand_preserved=False*, *weights=None*, *threads=1*, *use_gpu=False*)

Finds edges between queries and items in the reference database, and modifies the network to include them.

Args:

dbFuncs (list)
List of backend functions from *setupDBFuncs()*

rList (list)
List of reference names

qList (list)
List of query names

G (graph)
Network to add to (mutated)

assignments (numpy.array)
Cluster assignment of items in qlist

model (ClusterModel)
Model fitted to reference database

queryDB (str)
Query database location

distances (str)
Prefix of distance files for extending network

kmers (list)
List of k-mer sizes

distance_type (str)
Distance type to use as weights in network

queryQuery (bool)
Add in all query-query distances (default = False)

strand_preserved (bool)
Whether to treat strand as known (i.e. ignore rc k-mers) when adding random distances. Only used if queryQuery = True [default = False]

weights (numpy.array)
If passed, the core,accessory distances for each assignment, which will be annotated as an edge attribute

threads (int)
Number of threads to use if new db created

use_gpu (bool)
Whether to use cugraph for analysis

(default = 1)

Returns:

distMat (numpy.array)
Query-query distances

PopPUNK.network.**add_self_loop**(*G_df*, *seq_num*, *weights=False*, *renumber=True*)

Adds self-loop to cugraph graph to ensure all nodes are included in the graph, even if singletons.

Args:

G_df (cudf)
cudf data frame containing edge list

seq_num (int)
The expected number of nodes in the graph

renumber (bool)
Whether to renumber the vertices when added to the graph

Returns:

G_new (graph)
Dictionary of cluster assignments (keys are sequence names)

PopPUNK.network.**checkNetworkVertexCount**(*seq_list*, *G*, *use_gpu*)

Checks the number of network vertices matches the number of sequence names.

Args:

seq_list (list)
The list of sequence names

G (graph)
The network of sequences

use_gpu (bool)
Whether to use cugraph for graph analyses

PopPUNK.network.**cliquePrune**(*component*, *graph*, *reference_indices*, *components_list*)

Wrapper function around *getCliqueRefs()* so it can be called by a multiprocessing pool

PopPUNK.network.**construct_dense_weighted_network**(*rlist*, *distMat*, *weights_type=None*, *use_gpu=False*)

Construct an undirected network using sequence lists, assignments of pairwise distances to clusters, and the identifier of the cluster assigned to within-strain distances. Nodes are samples and edges where samples are within the same cluster

Will print summary statistics about the network to STDERR

Args:

rlist (list)
List of reference sequence labels

distMat (2 column ndarray)
Numpy array of pairwise distances

weights_type (str)
Type of weight to use for network

use_gpu (bool)
Whether to use GPUs for network construction

Returns:

G (graph)
The resulting network

PopPUNK.network.**construct_network_from_assignments**(*rlist*, *qlist*, *assignments*, *within_label=1*, *int_offset=0*, *weights=None*, *distMat=None*, *weights_type=None*, *previous_network=None*, *old_ids=None*, *adding_qq_dists=False*, *previous_pkl=None*, *betweenness_sample=100*, *summarise=True*, *use_gpu=False*)

Construct an undirected network using sequence lists, assignments of pairwise distances to clusters, and the identifier of the cluster assigned to within-strain distances. Nodes are samples and edges where samples are within the same cluster

Will print summary statistics about the network to STDERR

Args:

rlist (list)
List of reference sequence labels

qlist (list)
List of query sequence labels

assignments (numpy.array or int)
Labels of most likely cluster assignment

within_label (int)
The label for the cluster representing within-strain distances

int_offset (int)
Constant integer to add to each node index

weights (list)
List of weights for each edge in the network

distMat (2 column ndarray)
Numpy array of pairwise distances

weights_type (str)
Measure to calculate from the distMat to use as edge weights in network - options are core, accessory or euclidean distance

previous_network (str)
Name of file containing a previous network to be integrated into this new network

old_ids (list)
Ordered list of vertex names in previous network

adding_qq_dists (bool)
Boolean specifying whether query-query edges are being added to an existing network, such that not all the sequence IDs will be found in the old IDs, which should already be correctly ordered

previous_pkl (str)
Name of file containing the names of the sequences in the previous_network

betweenness_sample (int)
Number of sequences per component used to estimate betweenness using a GPU. Smaller numbers are faster but less precise [default = 100]

summarise (bool)
Whether to calculate and print network summaries with *networkSummary()* (default = True)

use_gpu (bool)
Whether to use GPUs for network construction

Returns:

G (graph)
The resulting network

PopPUNK.network.**construct_network_from_df**(*rlist*, *qlist*, *G_df*, *weights=False*, *distMat=None*, *previous_network=None*, *adding_qq_dists=False*, *old_ids=None*, *previous_pkl=None*, *betweenness_sample=100*, *summarise=True*, *use_gpu=False*)

Construct an undirected network using a data frame of edges. Nodes are samples and edges where samples are within the same cluster

Will print summary statistics about the network to STDERR

Args:

rlist (list)
List of reference sequence labels

qlist (list)
List of query sequence labels

G_df (cudf or pandas data frame)
Data frame in which the first two columns are the nodes linked by edges

weights (bool)
Whether weights in the G_df data frame should be included in the network

distMat (2 column ndarray)
Numpy array of pairwise distances

previous_network (str or graph object)
Name of file containing a previous network to be integrated into this new network, or the already-loaded graph object

adding_qq_dists (bool)
Boolean specifying whether query-query edges are being added to an existing network, such that not all the sequence IDs will be found in the old IDs, which should already be correctly ordered

old_ids (list)
Ordered list of vertex names in previous network

previous_pkl (str)
Name of file containing the names of the sequences in the previous_network

betweenness_sample (int)
Number of sequences per component used to estimate betweenness using a GPU. Smaller numbers are faster but less precise [default = 100]

summarise (bool)
Whether to calculate and print network summaries with *networkSummary()* (default = True)

use_gpu (bool)
Whether to use GPUs for network construction

Returns:

G (graph)
The resulting network

PopPUNK.network.**construct_network_from_edge_list**(*rlist*, *qlist*, *edge_list*, *weights=None*, *distMat=None*, *previous_network=None*, *adding_qq_dists=False*, *old_ids=None*, *previous_pkl=None*, *betweenness_sample=100*, *summarise=True*, *use_gpu=False*)

Construct an undirected network using a data frame of edges. Nodes are samples and edges where samples are within the same cluster

Will print summary statistics about the network to STDERR

Args:

rlist (list)
List of reference sequence labels

qlist (list)
List of query sequence labels

G_df (cudf or pandas data frame)
Data frame in which the first two columns are the nodes linked by edges

weights (list)
List of edge weights

distMat (2 column ndarray)
Numpy array of pairwise distances

previous_network (str or graph object)
Name of file containing a previous network to be integrated into this new network, or the already-loaded graph object

adding_qq_dists (bool)
Boolean specifying whether query-query edges are being added to an existing network, such that not all the sequence IDs will be found in the old IDs, which should already be correctly ordered

old_ids (list)
Ordered list of vertex names in previous network

previous_pkl (str)
Name of file containing the names of the sequences in the previous_network

betweenness_sample (int)
Number of sequences per component used to estimate betweenness using a GPU. Smaller numbers are faster but less precise [default = 100]

summarise (bool)
Whether to calculate and print network summaries with *networkSummary()* (default = True)

use_gpu (bool)
Whether to use GPUs for network construction

Returns:

G (graph)
The resulting network

PopPUNK.network.**construct_network_from_sparse_matrix**(*rlist*, *qlist*, *sparse_input*, *weights=None*, *previous_network=None*, *previous_pkl=None*, *betweenness_sample=100*, *summarise=True*, *use_gpu=False*)

Construct an undirected network using a sparse matrix. Nodes are samples and edges where samples are within the same cluster

Will print summary statistics about the network to STDERR

Args:

rlist (list)
List of reference sequence labels

qlist (list)
List of query sequence labels

sparse_input (numpy.array)
Sparse distance matrix from lineage fit

weights (list)
List of weights for each edge in the network

distMat (2 column ndarray)
Numpy array of pairwise distances

previous_network (str)
Name of file containing a previous network to be integrated into this new network

previous_pkl (str)
Name of file containing the names of the sequences in the previous_network

betweenness_sample (int)
Number of sequences per component used to estimate betweenness using a GPU. Smaller numbers are faster but less precise [default = 100]

summarise (bool)
Whether to calculate and print network summaries with *networkSummary()* (default = True)

use_gpu (bool)
Whether to use GPUs for network construction

Returns:

G (graph)
The resulting network

PopPUNK.network.**cugraph_to_graph_tool**(*G*, *rlist*)

Save a network to disk

Args:

G (cugraph network)
Cugraph network

rlist (list)
List of sequence names

Returns:

G (graph-tool network)
Graph tool network

PopPUNK.network.**extractReferences**(*G*, *dbOrder*, *outPrefix*, *outSuffix=''*, *type_isolate=None*, *existingRefs=None*, *threads=1*, *use_gpu=False*)

Extract references for each cluster based on cliques

Writes chosen references to file by calling *writeReferences()*

Args:

G (graph)
A network used to define clusters

dbOrder (list)
The order of files in the sketches, so returned references are in the same order

outPrefix (str)
Prefix for output file

outSuffix (str)
Suffix for output file (.refs will be appended)

type_isolate (str)
Isolate to be included in set of references

existingRefs (list)
References that should be used for each clique

use_gpu (bool)
Use cugraph for graph analysis (default = False)

Returns:

refFileName (str)
The name of the file references were written to

references (list)
An updated list of the reference names

PopPUNK.network.**fetchNetwork**(*network_dir*, *model*, *refList*, *ref_graph=False*, *core_only=False*, *accessory_only=False*, *use_gpu=False*)

Load the network based on input options

Returns the network as a graph-tool format graph, and sets the slope parameter of the passed model object.

Args:

network_dir (str)
A network used to define clusters

model (ClusterFit)
A fitted model object

refList (list)
Names of references that should be in the network

ref_graph (bool)
Use ref only graph, if available [default = False]

core_only (bool)
Return the network created using only core distances [default = False]

accessory_only (bool)
Return the network created using only accessory distances [default = False]

use_gpu (bool)
Use cugraph library to load graph

Returns:

genomeNetwork (graph)
The loaded network

cluster_file (str)
The CSV of cluster assignments corresponding to this network

PopPUNK.network.**generate_minimum_spanning_tree**(*G*, *from_cugraph=False*)

Generate a minimum spanning tree from a network

Args:

G (network)
Graph tool network

from_cugraph (bool)
If a pre-calculated MST from cugraph [default = False]

Returns:

mst_network (str)
Minimum spanning tree (as graph-tool graph)

PopPUNK.network.**getCliqueRefs**(*G*, *reference_indices={}*)

Recursively prune a network of its cliques. Returns one vertex from a clique at each stage

Args:

G (graph)
The graph to get clique representatives from

reference_indices (set)
The unique list of vertices being kept, to add to

PopPUNK.network.**get_cugraph_triangles**(*G*)

Counts the number of triangles in a cugraph network. Can be removed when the cugraph issue https://github.com/rapidsai/cugraph/issues/1043 is fixed.

Args:

G (cugraph network)
Network to be analysed

Returns:

triangle_count (int)
Count of triangles in graph

PopPUNK.network.**get_vertex_list**(*G*, *use_gpu=False*)

Generate a list of node indices

Args:

G (network)
Graph tool network

use_gpu (bool)
Whether graph is a cugraph or not [default = False]

Returns:

vlist (list)
List of integers corresponding to nodes

PopPUNK.network.**load_network_file**(*fn*, *use_gpu=False*)

Load the network based on input options

Returns the network as a graph-tool format graph, and sets the slope parameter of the passed model object.

Args:

fn (str)
Network file name

use_gpu (bool)
Use cugraph library to load graph

Returns:

genomeNetwork (graph)
The loaded network

PopPUNK.network.**networkSummary**(*G*, *calc_betweenness=True*, *betweenness_sample=100*, *use_gpu=False*)

Provides summary values about the network

Args:

G (graph)
The network of strains

calc_betweenness (bool)
Whether to calculate betweenness stats

use_gpu (bool)
Whether to use cugraph for graph analysis

Returns:

metrics (list)
List with # components, density, transitivity, mean betweenness and weighted mean betweenness

scores (list)
List of scores

PopPUNK.network.**network_to_edges**(*prev_G_fn*, *rlist*, *adding_qq_dists=False*, *old_ids=None*, *previous_pkl=None*, *weights=False*, *use_gpu=False*)

Load previous network, extract the edges to match the vertex order specified in rlist, and also return weights if specified.

Args:

prev_G_fn (str or graph object)
Path of file containing existing network, or already-loaded graph object

adding_qq_dists (bool)
Boolean specifying whether query-query edges are being added to an existing network, such that not all the sequence IDs will be found in the old IDs, which should already be correctly ordered

rlist (list)
List of reference sequence labels in new network

old_ids (list)
List of IDs of vertices in existing network

previous_pkl (str)
Path of pkl file containing names of sequences in previous network

weights (bool)
Whether to return edge weights (default = False)

use_gpu (bool)
Whether to use cugraph for graph analyses

Returns:

source_ids (list)
Source nodes for each edge

target_ids (list)
Target nodes for each edge

edge_weights (list)
Weights for each new edge

PopPUNK.network.**printClusters**(*G*, *rlist*, *outPrefix=None*, *oldClusterFile=None*, *externalClusterCSV=None*, *printRef=True*, *printCSV=True*, *clustering_type='combined'*, *write_unwords=True*, *use_gpu=False*)

Get cluster assignments

Also writes assignments to a CSV file

Args:

G (graph)
Network used to define clusters

rlist (list)
Names of samples

outPrefix (str)
Prefix for output CSV Default = None

oldClusterFile (str)
CSV with previous cluster assignments. Pass to ensure consistency in cluster assignment name. Default = None

externalClusterCSV (str)
CSV with cluster assignments from any source. Will print a file relating these to new cluster assignments Default = None

printRef (bool)
If false, print only query sequences in the output Default = True

printCSV (bool)
Print results to file Default = True

clustering_type (str)
Type of clustering network, used for comparison with old clusters Default = 'combined'

write_unwords (bool)
Write clusters with a pronouncable name rather than numerical index Default = True

use_gpu (bool)
Whether to use cugraph for network analysis

Returns:

clustering (dict)
Dictionary of cluster assignments (keys are sequence names)

PopPUNK.network.**printExternalClusters**(*newClusters*, *extClusterFile*, *outPrefix*, *oldNames*, *printRef=True*)

Prints cluster assignments with respect to previously defined clusters or labels.

Args:

newClusters (set iterable)
The components from the graph G, defining the PopPUNK clusters

extClusterFile (str)
A CSV file containing definitions of the external clusters for each sample (does not need to contain all samples)

outPrefix (str)
Prefix for output CSV (_external_clusters.csv)

oldNames (list)
A list of the reference sequences

printRef (bool)
If false, print only query sequences in the output

Default = True

PopPUNK.network.**print_network_summary**(*G*, *betweenness_sample=100*, *use_gpu=False*)

Wrapper function for printing network information

Args:

G (graph)
List of reference sequence labels

betweenness_sample (int)
Number of sequences per component used to estimate betweenness using a GPU. Smaller numbers are faster but less precise [default = 100]

use_gpu (bool)
Whether to use GPUs for network construction

PopPUNK.network.**process_previous_network**(*previous_network=None*, *adding_qq_dists=False*, *old_ids=None*, *previous_pkl=None*, *vertex_labels=None*, *weights=False*, *use_gpu=False*)

Extract edge types from an existing network

Args:

previous_network (str or graph object)
Name of file containing a previous network to be integrated into this new network, or already-loaded graph object

adding_qq_dists (bool)
Boolean specifying whether query-query edges are being added to an existing network, such that not all the sequence IDs will be found in the old IDs, which should already be correctly ordered

old_ids (list)
Ordered list of vertex names in previous network

previous_pkl (str)
Name of file containing the names of the sequences in the previous_network ordered based on the original network construction

vertex_labels (list)
Ordered list of sequence labels

weights (bool)
Whether weights should be extracted from the previous network

use_gpu (bool)
Whether to use GPUs for network construction

Returns:

extra_sources (list)
List of source node identifiers

extra_targets (list)
List of destination node identifiers

extra_weights (list or None)
List of edge weights

PopPUNK.network.**process_weights**(*distMat*, *weights_type*)

Calculate edge weights from the distance matrix

Args:

distMat (2 column ndarray)
Numpy array of pairwise distances

weights_type (str)
Measure to calculate from the distMat to use as edge weights in network - options are core, accessory or euclidean distance

Returns:

processed_weights (list)
Edge weights

PopPUNK.network.**save_network**(*G*, *prefix=None*, *suffix=None*, *use_graphml=False*, *use_gpu=False*)

Save a network to disk

Args:

G (network)
Graph tool network

prefix (str)
Prefix for output file

use_graphml (bool)
Whether to output a graph-tool file in graphml format

use_gpu (bool)
Whether graph is a cugraph or not [default = False]

PopPUNK.network.**sparse_mat_to_network**(*sparse_mat*, *rlist*, *use_gpu=False*)

Generate a network from a lineage rank fit

Args:

sparse_mat (scipy or cupyx sparse matrix)
Sparse matrix of kNN from lineage fit

rlist (list)
List of sequence names

use_gpu (bool)
Whether GPU libraries should be used

Returns:

G (network)
Graph tool or cugraph network

PopPUNK.network.**translate_network_indices**(*G_ref_df*, *reference_indices*)

Function for ensuring an updated reference network retains numbering consistent with sample names

Args:

G_ref_df (cudf data frame)
List of edges in reference network

reference_indices (list)
The ordered list of reference indices in the original network

Returns:

G_ref (cugraph network)
Network of reference sequences

PopPUNK.network.**vertex_betweenness**(*graph*, *norm=True*)

Returns betweenness for nodes in the graph

PopPUNK.network.**writeReferences**(*refList*, *outPrefix*, *outSuffix=''*)

Writes chosen references to file

Args:

refList (list)
Reference names to write

outPrefix (str)
Prefix for output file

outSuffix (str)
Suffix for output file (.refs will be appended)

Returns:

refFileName (str)
The name of the file references were written to

16.7 refine.py

Functions used to refine an existing model. Access using *RefineFit*. Refine mixture model using network properties

class PopPUNK.refine.**NumpyShared**(*name*, *shape*, *dtype*)

dtype

Alias for field number 2

name

Alias for field number 0

shape

Alias for field number 1

PopPUNK.refine.**check_search_range**(*scale*, *mean0*, *mean1*, *lower_s*, *upper_s*)

Checks a search range is within a valid range

Args:

scale (np.array)
Rescaling factor to [0, 1] for each axis

mean0 (np.array)
(x, y) of starting point defining line

mean1 (np.array)
(x, y) of end point defining line

lower_s (float)
distance along line to start search

upper_s (float)
distance along line to end search

Returns:

min_x, max_x
minimum and maximum x-intercepts of the search range

min_y, max_y
minimum and maximum x-intercepts of the search range

PopPUNK.refine.**expand_cugraph_network**(*G*, *G_extra_df*)

Reconstruct a cugraph network with additional edges.

Args:

G (cugraph network)
Original cugraph network

extra_edges (cudf dataframe)
Data frame of edges to add

Returns:

G (cugraph network)
Expanded cugraph network

PopPUNK.refine.**growNetwork**(*sample_names*, *i_vec*, *j_vec*, *idx_vec*, *s_range*, *score_idx=0*, *thread_idx=0*, *betweenness_sample=100*, *write_clusters=None*, *use_gpu=False*)

Construct a network, then add edges to it iteratively. Input is from `pp_sketchlib.iterateBoundary1D` or``pp_sketchlib.iterateBoundary2D``

Args:

sample_names (list)
Sample names corresponding to distMat (accessed by iterator)

i_vec (list)
Ordered ref vertex index to add

j_vec (list)
Ordered query (==ref) vertex index to add

idx_vec (list)
For each i, j tuple, the index of the intercept at which these enter the network. These are sorted and increasing

s_range (list)
Offsets which correspond to idx_vec entries

score_idx (int)
Index of score from *networkSummary()* to use [default = 0]

thread_idx (int)
Optional thread idx (if multithreaded) to offset progress bar by

betweenness_sample (int)
Number of sequences per component used to estimate betweenness using a GPU. Smaller numbers are faster but less precise [default = 100]

write_clusters (str)
Set to a prefix to write the clusters from each position to files [default = None]

use_gpu (bool)
Whether to use cugraph for graph analyses

Returns:

scores (list)
-1 * network score for each of x_range. Where network score is from *networkSummary()*

PopPUNK.refine.**likelihoodBoundary**(*s*, *model*, *start*, *end*, *within*, *between*)

Wrapper function around *fit2dMultiGaussian()* so that it can go into a root-finding function for probabilities between components

Args:

s (float)
Distance along line from mean0

model (BGMMFit)
Fitted mixture model

start (numpy.array)
The co-ordinates of the centre of the within-strain distribution

end (numpy.array)
The co-ordinates of the centre of the between-strain distribution

within (int)
Label of the within-strain distribution

between (int)
Label of the between-strain distribution

Returns:

responsibility (float)
The difference between responsibilities of assignment to the within component and the between assignment

PopPUNK.refine.**multi_refine**(*distMat*, *sample_names*, *mean0*, *mean1*, *scale*, *s_max*, *n_boundary_points*, *output_prefix*, *num_processes=1*, *use_gpu=False*)

Move the refinement boundary between the optimum and where it meets an axis. Discrete steps, output the clusers at each step

Args:

distMat (numpy.array)
n x 2 array of core and accessory distances for n samples

sample_names (list)
List of query sequence labels

mean0 (numpy.array)
Start point to define search line

mean1 (numpy.array)
End point to define search line

scale (numpy.array)
Scaling factor of distMat

s_max (float)
The optimal s position from refinement (*refineFit()*)

n_boundary_points (int)
Number of positions to try drawing the boundary at

num_processes (int)
Number of threads to use in the global optimisation step. (default = 1)

use_gpu (bool)
Whether to use cugraph for graph analyses

PopPUNK.refine.**newNetwork**(*s*, *sample_names*, *distMat*, *mean0*, *mean1*, *gradient*, *slope=2*, *score_idx=0*, *cpus=1*, *betweenness_sample=100*, *use_gpu=False*)

Wrapper function for *construct_network_from_edge_list()* which is called by optimisation functions moving a triangular decision boundary.

Given the boundary parameterisation, constructs the network and returns its score, to be minimised.

Args:

s (float)
Distance along line between start_point and mean1 from start_point

sample_names (list)
Sample names corresponding to distMat (accessed by iterator)

distMat (numpy.array or NumpyShared)
Core and accessory distances or NumpyShared describing these in sharedmem

mean0 (numpy.array)
Start point

mean1 (numpy.array)
End point

gradient (float)
Gradient of line to move along

slope (int)
Set to 0 for a vertical line, 1 for a horizontal line, or 2 to use a slope [default = 2]

score_idx (int)
Index of score from *networkSummary()* to use [default = 0]

cpus (int)
Number of CPUs to use for calculating assignment

betweenness_sample (int)
Number of sequences per component used to estimate betweenness using a GPU. Smaller numbers are faster but less precise [default = 100]

use_gpu (bool)
Whether to use cugraph for graph analysis

Returns:

score (float)
-1 * network score. Where network score is from *networkSummary()*

PopPUNK.refine.**newNetwork2D**(*y_idx*, *sample_names*, *distMat*, *x_range*, *y_range*, *score_idx=0*, *betweenness_sample=100*, *use_gpu=False*)

Wrapper function for thresholdIterate2D and *growNetwork()*.

For a given y_max, constructs networks across x_range and returns a list of scores

Args:

y_idx (float)
Maximum y-intercept of boundary, as index into y_range

sample_names (list)
Sample names corresponding to distMat (accessed by iterator)

distMat (numpy.array or NumpyShared)
Core and accessory distances or NumpyShared describing these in sharedmem

x_range (list)
Sorted list of x-intercepts to search

y_range (list)
Sorted list of y-intercepts to search

score_idx (int)
Index of score from *networkSummary()* to use [default = 0]

betweenness_sample (int)
Number of sequences per component used to estimate betweenness using a GPU. Smaller numbers are faster but less precise [default = 100]

use_gpu (bool)
Whether to use cugraph for graph analysis

Returns:

scores (list)
-1 * network score for each of x_range. Where network score is from *networkSummary()*

PopPUNK.refine.**readManualStart**(*startFile*)

Reads a file to define a manual start point, rather than using `--fit-model`

Throws and exits if incorrectly formatted.

Args:

startFile (str)
Name of file with values to read

Returns:

mean0 (numpy.array)
Centre of within-strain distribution

mean1 (numpy.array)
Centre of between-strain distribution

scaled (bool)
True if means are scaled between [0,1]

PopPUNK.refine.**refineFit**(*distMat*, *sample_names*, *mean0*, *mean1*, *scale*, *max_move*, *min_move*, *slope=2*, *score_idx=0*, *unconstrained=False*, *no_local=False*, *num_processes=1*, *betweenness_sample=100*, *use_gpu=False*)

Try to refine a fit by maximising a network score based on transitivity and density.

Iteratively move the decision boundary to do this, using starting point from existing model.

Args:

distMat (numpy.array)
n x 2 array of core and accessory distances for n samples

sample_names (list)
List of query sequence labels

mean0 (numpy.array)
Start point to define search line

mean1 (numpy.array)
End point to define search line

scale (numpy.array)
Scaling factor of distMat

max_move (float)
Maximum distance to move away from start point

min_move (float)
Minimum distance to move away from start point

slope (int)
Set to 0 for a vertical line, 1 for a horizontal line, or 2 to use a slope

score_idx (int)
Index of score from *networkSummary()* to use [default = 0]

unconstrained (bool)
If True, search in 2D and change the slope of the boundary

no_local (bool)
Turn off the local optimisation step. Quicker, but may be less well refined.

num_processes (int)
Number of threads to use in the global optimisation step. (default = 1)

betweenness_sample (int)
Number of sequences per component used to estimate betweenness using a GPU. Smaller numbers are faster but less precise [default = 100]

use_gpu (bool)
Whether to use cugraph for graph analyses

Returns:

optimal_x (float)
x-coordinate of refined fit

optimal_y (float)
y-coordinate of refined fit

optimised_s (float)
Position along search range of refined fit

16.8 plot.py

Plots of GMM results, k-mer fits, and microreact output

`PopPUNK.plot.createMicroreact(`*prefix*, *microreact_files*, *api_key=None*`)`

Creates a .microreact file, and instance via the API

Args:

prefix (str)
Prefix for output file

microreact_files (str)
List of Microreact files [clusters, dot, tree, mst_tree]

api_key (str)
API key for your account

`PopPUNK.plot.distHistogram(`*dists*, *rank*, *outPrefix*`)`

Plot a histogram of distances (1D)

Args:

dists (np.array)
Distance vector

rank (int)
Rank (used for name and title)

outPrefix (int)
Full path prefix for plot file

`PopPUNK.plot.drawMST(`*mst*, *outPrefix*, *isolate_clustering*, *clustering_name*, *overwrite*`)`

Plot a layout of the minimum spanning tree

Args:

mst (graph_tool.Graph)
A minimum spanning tree

outPrefix (str)
Output prefix for save files

isolate_clustering (dict)
Dictionary of ID: cluster, used for colouring vertices

clustering_name (str)
Name of clustering scheme to be used for colouring

overwrite (bool)
Overwrite existing output files

PopPUNK.plot.**get_grid**(*minimum*, *maximum*, *resolution*)

Get a square grid of points to evaluate a function across

Used for *plot_scatter()* and *plot_contours()*

Args:

minimum (float)
Minimum value for grid

maximum (float)
Maximum value for grid

resolution (int)
Number of points along each axis

Returns:

xx (numpy.array)
x values across n x n grid

yy (numpy.array)
y values across n x n grid

xy (numpy.array)
n x 2 pairs of x, y values grid is over

PopPUNK.plot.**outputsForCytoscape**(*G*, *G_mst*, *isolate_names*, *clustering*, *outPrefix*, *epiCsv*, *queryList=None*, *suffix=None*, *writeCsv=True*)

Write outputs for cytoscape. A graphml of the network, and CSV with metadata

Args:

G (graph)
The network to write

G_mst (graph)
The minimum spanning tree of G

isolate_names (list)
Ordered list of sequence names

clustering (dict)
Dictionary of cluster assignments (keys are nodeNames).

outPrefix (str)
Prefix for files to be written

epiCsv (str)
Optional CSV of epi data to paste in the output in addition to the clusters.

queryList (list)
Optional list of isolates that have been added as a query. (default = None)

suffix (string)
String to append to network file name. (default = None)

writeCsv (bool)
Whether to print CSV file to accompany network

PopPUNK.plot.**outputsForGrapetree**(*combined_list*, *clustering*, *nj_tree*, *mst_tree*, *outPrefix*, *epiCsv*, *queryList=None*, *overwrite=False*)

Generate files for Grapetree

Write a neighbour joining tree (.nwk) from core distances and cluster assignment (.csv)

Args:

combined_list (list)
Name of sequences being analysed. The part of the name before the first '.' will be shown in the output

clustering (dict or dict of dicts)
List of cluster assignments from *printClusters()*. Further clusterings (e.g. 1D core only) can be included by passing these as a dict.

nj_tree (str or None)
String representation of a Newick-formatted NJ tree

mst_tree (str or None)
String representation of a Newick-formatted minimum-spanning tree

outPrefix (str)
Prefix for all generated output files, which will be placed in *outPrefix* subdirectory.

epiCsv (str)
A CSV containing other information, to include with the CSV of clusters

queryList (list)
Optional list of isolates that have been added as a query for colouring in the CSV. (default = None)

overwrite (bool)
Overwrite existing output if present (default = False).

PopPUNK.plot.**outputsForMicroreact**(*combined_list*, *clustering*, *nj_tree*, *mst_tree*, *accMat*, *perplexity*, *maxIter*, *outPrefix*, *epiCsv*, *queryList=None*, *overwrite=False*, *n_threads=1*, *use_gpu=False*, *device_id=0*)

Generate files for microreact

Output a neighbour joining tree (.nwk) from core distances, a plot of t-SNE clustering of accessory distances (.dot) and cluster assignment (.csv)

Args:

combined_list (list)
Name of sequences being analysed. The part of the name before the first '.' will be shown in the output

clustering (dict or dict of dicts)
List of cluster assignments from *printClusters()*. Further clusterings (e.g. 1D core only) can be included by passing these as a dict.

nj_tree (str or None)
String representation of a Newick-formatted NJ tree

mst_tree (str or None)
String representation of a Newick-formatted minimum-spanning tree

accMat (numpy.array)
n x n array of accessory distances for n samples.

perplexity (int)
Perplexity parameter passed to mandrake

maxIter (int)
Maximum iterations for mandrake

outPrefix (str)
Prefix for all generated output files, which will be placed in *outPrefix* subdirectory

epiCsv (str)
A CSV containing other information, to include with the CSV of clusters

queryList (list)
Optional list of isolates that have been added as a query for colouring in the CSV. (default = None)

overwrite (bool)
Overwrite existing output if present (default = False)

n_threads (int)
Number of CPU threads to use (default = 1)

use_gpu (bool)
Whether to use a GPU for t-SNE generation

device_id (int)
Device ID of GPU to be used (default = 0)

Returns:

outfiles (list)
List of output files create

PopPUNK.plot.**outputsForPhandango**(*combined_list*, *clustering*, *nj_tree*, *mst_tree*, *outPrefix*, *epiCsv*, *queryList=None*, *overwrite=False*)

Generate files for Phandango

Write a neighbour joining tree (.tree) from core distances and cluster assignment (.csv)

Args:

combined_list (list)
Name of sequences being analysed. The part of the name before the first '.' will be shown in the output

clustering (dict or dict of dicts)
List of cluster assignments from *printClusters()*. Further clusterings (e.g. 1D core only) can be included by passing these as a dict.

nj_tree (str or None)
String representation of a Newick-formatted NJ tree

mst_tree (str or None)
String representation of a Newick-formatted minimum-spanning tree

outPrefix (str)
Prefix for all generated output files, which will be placed in *outPrefix* subdirectory

epiCsv (str)
A CSV containing other information, to include with the CSV of clusters

queryList (list)
Optional list of isolates that have been added as a query for colouring in the CSV. (default = None)

overwrite (bool)
Overwrite existing output if present (default = False)

threads (int)
Number of threads to use with rapidnj

PopPUNK.plot.**plot_contours**(*model*, *assignments*, *title*, *out_prefix*)

Draw contours of mixture model assignments

Will draw the decision boundary for between/within in red

Args:

model (BGMMFit)
Model we are plotting from

assignments (numpy.array)
n-vectors of cluster assignments for model

title (str)
The title to display above the plot

out_prefix (str)
Prefix for output plot file (.pdf will be appended)

PopPUNK.plot.**plot_dbscan_results**(*X*, *y*, *n_clusters*, *out_prefix*)

Draw a scatter plot (png) to show the DBSCAN model fit

A scatter plot of core and accessory distances, coloured by component membership. Black is noise

Args:

X (numpy.array)
n x 2 array of core and accessory distances for n samples.

Y (numpy.array)
n x 1 array of cluster assignments for n samples.

n_clusters (int)
Number of clusters used (excluding noise)

out_prefix (str)
Prefix for output file (.png will be appended)

PopPUNK.plot.**plot_fit**(*klist*, *raw_matching*, *raw_fit*, *corrected_matching*, *corrected_fit*, *out_prefix*, *title*)

Draw a scatter plot (pdf) of k-mer sizes vs match probability, and the fit used to assign core and accessory distance

K-mer sizes on x-axis, log(pr(match)) on y - expect a straight line fit with intercept representing accessory distance and slope core distance

Args:

klist (list)
List of k-mer sizes

raw_matching (list)
Proportion of matching k-mers at each klist value

raw_fit (numpy.array)
Fit to klist and raw_matching from *fitKmerCurve()*

corrected_matching (list)
Corrected proportion of matching k-mers at each klist value

corrected_fit (numpy.array)
Fit to klist and corrected_matching from *fitKmerCurve()*

out_prefix (str)
Prefix for output plot file (.pdf will be appended)

title (str)
The title to display above the plot

PopPUNK.plot.**plot_refined_results**(*X*, *Y*, *x_boundary*, *y_boundary*, *core_boundary*, *accessory_boundary*, *mean0*, *mean1*, *min_move*, *max_move*, *scale*, *threshold*, *indiv_boundaries*, *unconstrained*, *title*, *out_prefix*)

Draw a scatter plot (png) to show the refined model fit

A scatter plot of core and accessory distances, coloured by component membership. The triangular decision boundary is also shown

Args:

X (numpy.array)
n x 2 array of core and accessory distances for n samples.

Y (numpy.array)
n x 1 array of cluster assignments for n samples.

x_boundary (float)
Intercept of boundary with x-axis, from *RefineFit*

y_boundary (float)
Intercept of boundary with y-axis, from *RefineFit*

core_boundary (float)
Intercept of 1D (core) boundary with x-axis, from *RefineFit*

accessory_boundary (float)
Intercept of 1D (core) boundary with y-axis, from *RefineFit*

mean0 (numpy.array)
Centre of within-strain distribution

mean1 (numpy.array)
Centre of between-strain distribution

min_move (float)
Minimum s range

max_move (float)
Maximum s range

scale (numpy.array)
Scaling factor from *RefineFit*

threshold (bool)
If fit was just from a simple thresholding

indiv_boundaries (bool)
Whether to draw lines for core and accessory refinement

title (str)
The title to display above the plot

out_prefix (str)
Prefix for output plot file (.png will be appended)

PopPUNK.plot.**plot_results**(*X*, *Y*, *means*, *covariances*, *scale*, *title*, *out_prefix*)

Draw a scatter plot (png) to show the BGMM model fit

A scatter plot of core and accessory distances, coloured by component membership. Also shown are ellipses for each component (centre: means axes: covariances).

This is based on the example in the sklearn documentation.

Args:

X (numpy.array)
n x 2 array of core and accessory distances for n samples.

Y (numpy.array)
n x 1 array of cluster assignments for n samples.

means (numpy.array)
Component means from *BGMMFit*

covars (numpy.array)
Component covariances from *BGMMFit*

scale (numpy.array)
Scaling factor from *BGMMFit*

out_prefix (str)
Prefix for output plot file (.png will be appended)

title (str)
The title to display above the plot

PopPUNK.plot.**plot_scatter**(*X*, *out_prefix*, *title*, *kde=True*)

Draws a 2D scatter plot (png) of the core and accessory distances

Also draws contours of kernel density estimare

Args:

X (numpy.array)
n x 2 array of core and accessory distances for n samples.

out_prefix (str)
Prefix for output plot file (.png will be appended)

title (str)
The title to display above the plot

kde (bool)
Whether to draw kernel density estimate contours

(default = True)

PopPUNK.plot.**writeClusterCsv**(*outfile*, *nodeNames*, *nodeLabels*, *clustering*, *output_format='microreact'*, *epiCsv=None*, *queryNames=None*, *suffix='_Cluster'*)

Print CSV file of clustering and optionally epi data

Writes CSV output of clusters which can be used as input to microreact and cytoscape. Uses pandas to deal with CSV reading and writing nicely.

The epiCsv, if provided, should have the node labels in the first column.

Args:

outfile (str)
File to write the CSV to.

nodeNames (list)
Names of sequences in clustering (includes path).

nodeLabels (list)
Names of sequences to write in CSV (usually has path removed).

clustering (dict or dict of dicts)
Dictionary of cluster assignments (keys are nodeNames). Pass a dict with depth two to include multiple possible clusterings.

output_format (str)
Software for which CSV should be formatted (microreact, phandango, grapetree and cytoscape are accepted)

epiCsv (str)
Optional CSV of epi data to paste in the output in addition to the clusters (default = None).

queryNames (list)
Optional list of isolates that have been added as a query.

(default = None)

16.9 sparse_mst.py

16.10 sketchlib.py

Sketchlib functions for database construction

PopPUNK.sketchlib.**addRandom**(*oPrefix*, *sequence_names*, *klist*, *strand_preserved=False*, *overwrite=False*, *threads=1*)

Add chance of random match to a HDF5 sketch DB

Args:

oPrefix (str)
Sketch database prefix

sequence_names (list)
Names of sequences to include in calculation

klist (list)
List of k-mer sizes to sketch

strand_preserved (bool)
Set true to ignore rc k-mers

overwrite (str)
Set true to overwrite existing random match chances

threads (int)
Number of threads to use (default = 1)

PopPUNK.sketchlib.**checkSketchlibLibrary**()

Gets the location of the sketchlib library

Returns:

lib (str)
Location of sketchlib .so/.dyld

PopPUNK.sketchlib.**checkSketchlibVersion**()

Checks that sketchlib can be run, and returns version

Returns:

version (str)
Version string

PopPUNK.sketchlib.**constructDatabase**(*assemblyList*, *klist*, *sketch_size*, *oPrefix*, *threads*, *overwrite*, *strand_preserved*, *min_count*, *use_exact*, *calc_random=True*, *codon_phased=False*, *use_gpu=False*, *deviceid=0*)

Sketch the input assemblies at the requested k-mer lengths

A multithread wrapper around `runSketch()`. Threads are used to either run multiple sketch processes for each klist value.

Also calculates random match probability based on length of first genome in assemblyList.

Args:

assemblyList (str)
File with locations of assembly files to be sketched

klist (list)
List of k-mer sizes to sketch

sketch_size (int)
Size of sketch (`-s` option)

oPrefix (str)
Output prefix for resulting sketch files

threads (int)
Number of threads to use (default = 1)

overwrite (bool)
Whether to overwrite sketch DBs, if they already exist. (default = False)

strand_preserved (bool)
Ignore reverse complement k-mers (default = False)

min_count (int)
Minimum count of k-mer in reads to include (default = 0)

use_exact (bool)
Use exact count of k-mer appearance in reads (default = False)

calc_random (bool)
Add random match chances to DB (turn off for queries)

codon_phased (bool)
Use codon phased seeds (default = False)

use_gpu (bool)
Use GPU for read sketching (default = False)

deviceid (int)
GPU device id (default = 0)

Returns:

names (list)
List of names included in the database (from rfile)

PopPUNK.sketchlib.**createDatabaseDir**(*outPrefix*, *kmers*)

Creates the directory to write sketches to, removing old files if unnecessary

Args:

outPrefix (str)
output db prefix

kmers (list)
k-mer sizes in db

PopPUNK.sketchlib.**fitKmerCurve**(*pairwise*, *klist*, *jacobian*)

Fit the function $pr = (1-a)(1-c)^k$

Supply `jacobian = -np.hstack((np.ones((klist.shape[0], 1)), klist.reshape(-1, 1)))`

Args:

pairwise (numpy.array)
Proportion of shared k-mers at k-mer values in klist

klist (list)
k-mer sizes used

jacobian (numpy.array)
Should be set as above (set once to try and save memory)

Returns:

transformed_params (numpy.array)
Column with core and accessory distance

PopPUNK.sketchlib.**getKmersFromReferenceDatabase**(*dbPrefix*)

Get kmers lengths from existing database

Args:

dbPrefix (str)
Prefix for sketch DB files

Returns:

kmers (list)
List of k-mer lengths used in database

PopPUNK.sketchlib.**getSeqsInDb**(*dbname*)

Return an array with the sequences in the passed database

Args:

dbname (str)
Sketches database filename

Returns:

seqs (list)
List of sequence names in sketch DB

PopPUNK.sketchlib.**getSketchSize**(*dbPrefix*)

Determine sketch size, and ensures consistent in whole database

`sys.exit(1)` is called if DBs have different sketch sizes

Args:

dbprefix (str)
Prefix for databases

Returns:

sketchSize (int)
sketch size (64x C++ definition)

codonPhased (bool)
whether the DB used codon phased seeds

PopPUNK.sketchlib.**joinDBs**(*db1*, *db2*, *output*, *update_random=None*)

Join two sketch databases with the low-level HDF5 copy interface

Args:

db1 (str)
Prefix for db1

db2 (str)
Prefix for db2

output (str)
Prefix for joined output

update_random (dict)
Whether to re-calculate the random object. May contain control arguments strand_preserved and threads (see *addRandom()*)

PopPUNK.sketchlib.**queryDatabase**(*rNames*, *qNames*, *dbPrefix*, *queryPrefix*, *klist*, *self=True*, *number_plot_fits=0*, *threads=1*, *use_gpu=False*, *deviceid=0*)

Calculate core and accessory distances between query sequences and a sketched database

For a reference database, runs the query against itself to find all pairwise core and accessory distances.

Uses the relation $pr(a, b) = (1 - a)(1 - c)^k$

To get the ref and query name for each row of the returned distances, call to the iterator *iterDistRows()* with the returned refList and queryList

Args:

rNames (list)
Names of references to query

qNames (list)
Names of queries

dbPrefix (str)
Prefix for reference sketch database created by *constructDatabase()*

queryPrefix (str)
Prefix for query sketch database created by *constructDatabase()*

klist (list)
K-mer sizes to use in the calculation

self (bool)
Set true if query = ref (default = True)

number_plot_fits (int)
If > 0, the number of k-mer length fits to plot (saved as pdfs). Takes random pairs of comparisons and calls *plot_fit()* (default = 0)

threads (int)
Number of threads to use in the process (default = 1)

use_gpu (bool)
Use a GPU for querying (default = False)

deviceid (int)
Index of the CUDA GPU device to use (default = 0)

Returns:

distMat (numpy.array)
Core distances (column 0) and accessory distances (column 1) between refList and queryList

PopPUNK.sketchlib.**readDBParams**(*dbPrefix*)

Get kmers lengths and sketch sizes from existing database

Calls *getKmersFromReferenceDatabase()* and *getSketchSize()* Uses passed values if db missing

Args:

dbPrefix (str)
Prefix for sketch DB files

Returns:

kmers (list)
List of k-mer lengths used in database

sketch_sizes (list)
List of sketch sizes used in database

codonPhased (bool)
whether the DB used codon phased seeds

PopPUNK.sketchlib.**removeFromDB**(*db_name*, *out_name*, *removeSeqs*, *full_names=False*)

Remove sketches from the DB the low-level HDF5 copy interface

Args:

db_name (str)
Prefix for hdf database

out_name (str)
Prefix for output (pruned) database

removeSeqs (list)
Names of sequences to remove from database

full_names (bool)
If True, db_name and out_name are the full paths to h5 files

16.11 utils.py

General utility functions for data read/writing/manipulation in PopPUNK

PopPUNK.utils.**check_and_set_gpu**(*use_gpu*, *gpu_lib*, *quit_on_fail=False*)

Check GPU libraries can be loaded and set managed memory.

Args:

use_gpu (bool)
Whether GPU packages have been requested

gpu_lib (bool)
Whether GPU packages are available

Returns:

use_gpu (bool)
Whether GPU packages can be used

PopPUNK.utils.**decisionBoundary**(*intercept*, *gradient*)

Returns the co-ordinates where the triangle the decision boundary forms meets the x- and y-axes.

Args:

intercept (numpy.array)
Cartesian co-ordinates of point along line (*transformLine()*) which intercepts the boundary

gradient (float)
Gradient of the line

Returns:

x (float)
The x-axis intercept

y (float)
The y-axis intercept

PopPUNK.utils.**get_match_search_depth**(*rlist*, *rank_list*)

Return a default search depth for lineage model fitting.

Args:

rlist (list)
List of sequences in database

rank_list (list)
List of ranks to be used to fit lineage models

Returns:

max_search_depth (int)
Maximum kNN used for lineage model fitting

PopPUNK.utils.**isolateNameToLabel**(*names*)

Function to process isolate names to labels appropriate for visualisation.

Args:

names (list)
List of isolate names.

Returns:

labels (list)
List of isolate labels.

PopPUNK.utils.**iterDistRows**(*refSeqs*, *querySeqs*, *self=True*)

Gets the ref and query ID for each row of the distance matrix

Returns an iterable with ref and query ID pairs by row.

Args:

refSeqs (list)
List of reference sequence names.

querySeqs (list)
List of query sequence names.

self (bool)
Whether a self-comparison, used when constructing a database. Requires refSeqs == querySeqs Default is True

Returns:

ref, query (str, str)
Iterable of tuples with ref and query names for each distMat row.

PopPUNK.utils.**joinClusterDicts**(*d1*, *d2*)

Join two dictionaries returned by *readIsolateTypeFromCsv()* with return_dict = True. Useful for concatenating ref and query assignments

Args:

d1 (dict of dicts)
First dictionary to concat

d2 (dict of dicts)
Second dictionary to concat

Returns:

d1 (dict of dicts)
d1 with d2 appended

PopPUNK.utils.**listDistInts**(*refSeqs*, *querySeqs*, *self=True*)

Gets the ref and query ID for each row of the distance matrix

Returns an iterable with ref and query ID pairs by row.

Args:

refSeqs (list)
List of reference sequence names.

querySeqs (list)
List of query sequence names.

self (bool)
Whether a self-comparison, used when constructing a database. Requires refSeqs == querySeqs Default is True

Returns:

ref, query (str, str)
Iterable of tuples with ref and query names for each distMat row.

PopPUNK.utils.**readIsolateTypeFromCsv**(*clustCSV*, *mode='clusters'*, *return_dict=False*)

Read cluster definitions from CSV file.

Args:

clustCSV (str)
File name of CSV with isolate assignments

mode (str)
Type of file to read 'clusters', 'lineages', or 'external'

return_type (str)
If True, return a dict with sample->cluster instead of sets [default = False]

Returns:

clusters (dict)
Dictionary of cluster assignments (keys are cluster names, values are sets containing samples in the cluster). Or if return_dict is set keys are sample names, values are cluster assignments.

PopPUNK.utils.**readPickle**(*pklName*, *enforce_self=False*, *distances=True*)

Loads core and accessory distances saved by *storePickle()*

Called during `--fit-model`

Args:

pklName (str)
Prefix for saved files

enforce_self (bool)
Error if self == False

[default = True]

distances (bool)
Read the distance matrix

[default = True]

Returns:

rlist (list)
List of reference sequence names (for *iterDistRows()*)

qlist (list)
List of query sequence names (for *iterDistRows()*)

self (bool)
Whether an all-vs-all self DB (for *iterDistRows()*)

X (numpy.array)
n x 2 array of core and accessory distances

PopPUNK.utils.**readRfile**(*rFile*, *oneSeq=False*)

Reads in files for sketching. Names and sequence, tab separated

Args:

rFile (str)
File with locations of assembly files to be sketched

oneSeq (bool)
Return only the first sequence listed, rather than a list (used with mash)

Returns:

names (list)
Array of sequence names

sequences (list of lists)
Array of sequence files

PopPUNK.utils.**read_rlist_from_distance_pickle**(*fn*, *allow_non_self=True*)

Return the list of reference sequences from a distance pickle.

Args:

fn (str)
Name of distance pickle

allow_non_self (bool)
Whether non-self distance datasets are permissible

Returns:

rlist (list)
List of reference sequence names

PopPUNK.utils.**set_env**(***environ*)

Temporarily set the process environment variables. >>> with set_env(PLUGINS_DIR=u'test/plugins'): ... "PLUGINS_DIR" in os.environ True >>> "PLUGINS_DIR" in os.environ False

PopPUNK.utils.**setupDBFuncs**(*args*)

Wraps common database access functions from sketchlib and mash, to try and make their API more similar

Args:

args (argparse.opts)
Parsed command lines options

qc_dict (dict)
Table of parameters for QC function

Returns:

dbFuncs (dict)
Functions with consistent arguments to use as the database API

PopPUNK.utils.**stderr_redirected**(*to='/dev/null'*)

import os

with stdout_redirected(to=filename):
print("from Python") os.system("echo non-Python applications are also supported")

PopPUNK.utils.**storePickle**(*rlist*, *qlist*, *self*, *X*, *pklName*)

Saves core and accessory distances in a .npy file, names in a .pkl

Called during `--create-db`

Args:

rlist (list)
List of reference sequence names (for *iterDistRows()*)

qlist (list)
List of query sequence names (for *iterDistRows()*)

self (bool)
Whether an all-vs-all self DB (for *iterDistRows()*)

X (numpy.array)
n x 2 array of core and accessory distances

pklName (str)
Prefix for output files

PopPUNK.utils.**transformLine**(*s*, *mean0*, *mean1*)

Return x and y co-ordinates for traversing along a line between mean0 and mean1, parameterised by a single scalar distance s from the start point mean0.

Args:

s (float)
Distance along line from mean0

mean0 (numpy.array)
Start position of line (x0, y0)

mean1 (numpy.array)
End position of line (x1, y1)

Returns:

x (float)
The Cartesian x-coordinate

y (float)
The Cartesian y-coordinate

PopPUNK.utils.**update_distance_matrices**(*refList*, *distMat*, *queryList=None*, *query_ref_distMat=None*, *query_query_distMat=None*, *threads=1*)

Convert distances from long form (1 matrix with n_comparisons rows and 2 columns) to a square form (2 NxN matrices), with merging of query distances if necessary.

Args:

refList (list)
List of references

distMat (numpy.array)
Two column long form list of core and accessory distances for pairwise comparisons between reference db sequences

queryList (list)
List of queries

query_ref_distMat (numpy.array)
Two column long form list of core and accessory distances for pairwise comparisons between queries and reference db sequences

query_query_distMat (numpy.array)
Two column long form list of core and accessory distances for pairwise comparisons between query sequences

threads (int)
Number of threads to use

Returns:

seqLabels (list)
Combined list of reference and query sequences

coreMat (numpy.array)
NxN array of core distances for N sequences

accMat (numpy.array)
NxN array of accessory distances for N sequences

16.12 visualise.py

`poppunk_visualise` main function

`PopPUNK.visualise.`**`main`**`()`

Main function. Parses cmd line args and runs in the specified mode.

16.13 web.py

Functions used by the web API to convert a sketch to an h5 database, then generate visualisations and post results to PopPUNK-web.

`PopPUNK.web.`**`api`**`(`*query*, *ref_db*`)`

Post cluster and tree information to microreact

`PopPUNK.web.`**`calc_prevalence`**`(`*cluster*, *cluster_list*, *num_samples*`)`

Cluster prevalences for Plotly.js

`PopPUNK.web.`**`graphml_to_json`**`(`*network_dir*`)`

Converts full GraphML file to JSON subgraph

`PopPUNK.web.`**`highlight_cluster`**`(`*query*, *cluster*`)`

Colour assigned cluster in Microreact output

`PopPUNK.web.`**`sketch_to_hdf5`**`(`*sketches_dict*, *output*`)`

Convert dict of JSON sketches to query hdf5 database

`PopPUNK.web.`**`summarise_clusters`**`(`*output*, *species*, *species_db*, *qNames*`)`

Retreieve assigned query and all cluster prevalences. Write list of all isolates in cluster for tree subsetting

CHAPTER

SEVENTEEN

ROADMAP

This document describes our future plans for additions to PopPUNK, pp-sketchlib and BeeBOP and BeeBOP. Tasks are in order of priority.

17.1 PopPUNK

1. Containerise the workflow. See #193, #277, #278.
2. Add full worked tutorials back to the documentation #275.
3. Make the update pipeline more robust. See #273.
4. **Codebase optimsation and refactoring**
    - Modularisation of the network code #249.
    - Removing old functions #103
5. **Add more species databases:**
    - N. meningitidis #267.
    - H. influenzae #276.
6. Stable names for lineage/subclustering modes.

Other enhancements listed on the issue page are currently not planned.

17.2 pp-sketchlib

1. **Update installation in package managers**
    - Update for new macOS #92
    - Rebuild conda recipe for CUDA12 and newer HDF5 #46
2. Allow amino-acids as input #89.

Other enhancements listed on the issue page are currently not planned.

17.3 BeeBOP

1. Update backend database to v8 #42.
2. Update CI images.
3. **Add more info on landing page.**
    - News page.
    - About page.
    - Methods description.
4. Add custom cluster names (e.g. GPSCs)
5. Integration tests for error logging.
6. **Persist user data.**
    - Persist microreact tokens #41.
    - Allow user to change or delete tokens
7. Add linage/subclusters to results page #23.
8. Report sample quality to user.
9. Front-end update for large numbers of input files.
10. Add serotype prediction for *S. pneumoniae*.
11. Add multiple species databases.

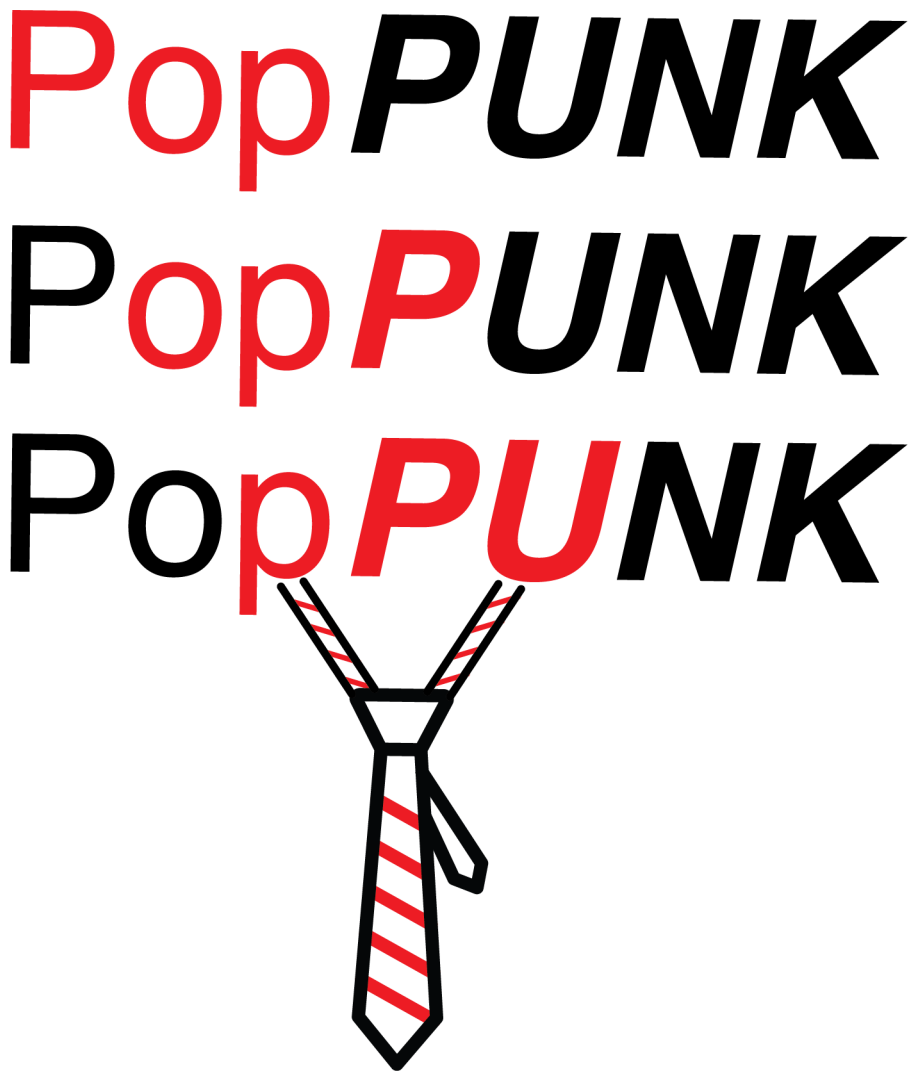
CHAPTER

EIGHTEEN

MISCELLANEOUS

18.1 Rejected/alternative logos


GOD Save THE QUEEN
Sex PISTOLS
poppunk


PopPUNK
PopPUNK
PopPUNK

CHAPTER

NINETEEN

WHY USE POPPUNK?

The advantages of PopPUNK are broadly that:

- It is fast, and scalable to over 10^5 genomes in a single run.
- Assigning new query sequences to a cluster using an existing database is scalable even beyond this.
- Cluster names remain consistent between studies, and other cluster labels such as MLST can be appended.
- Databases can be updated online (as sequences arrive).
- Online updating is equivalent to building databases from scratch.
- Databases can be kept small and managable by only keeping representative isolates.
- Databases naturally allow in-depth analysis of single clusters, but keeping the full context of the whole database.
- There is no bin cluster. Outlier isolates will be in their own cluster.
- Pre-processing, such as generation of an alignment, is not required.
- Raw sequence reads can be used as input, while being filtered for sequencing errors.
- The definition of clusters are biologically relevant.
- Many quantitative and graphical outputs are provided.
- A direct import into microreact is available, as well as cytoscape, grapetree and phandango.
- Everything is available within a single python executable.

CHAPTER

TWENTY

CITATION

If you find PopPUNK useful, please cite as:

Lees JA, Harris SR, Tonkin-Hill G, Gladstone RA, Lo SW, Weiser JN, Corander J, Bentley SD, Croucher NJ. Fast and flexible bacterial genomic epidemiology with PopPUNK. *Genome Research* **29**:1-13 (2019). doi:10.1101/gr.241455.118

See *Citing PopPUNK* for more details.

CHAPTER

TWENTYONE

INDEX

- genindex
- search

PYTHON MODULE INDEX

p

PopPUNK.assign, 91
PopPUNK.bgmm, 91
PopPUNK.dbscan, 93
PopPUNK.mandrake, 94
PopPUNK.models, 95
PopPUNK.network, 103
PopPUNK.plot, 121
PopPUNK.refine, 116
PopPUNK.sketchlib, 128
PopPUNK.sparse_mst, 128
PopPUNK.utils, 133
PopPUNK.visualise, 138
PopPUNK.web, 138

INDEX

A

add_self_loop() (*in module PopPUNK.network*), 104
addQueryToNetwork() (*in module PopPUNK.network*), 103
addRandom() (*in module PopPUNK.sketchlib*), 128
api() (*in module PopPUNK.web*), 138
apply_threshold() (*PopPUNK.models.RefineFit method*), 100
assign() (*PopPUNK.models.BGMMFit method*), 95
assign() (*PopPUNK.models.DBSCANFit method*), 97
assign() (*PopPUNK.models.LineageFit method*), 98
assign() (*PopPUNK.models.RefineFit method*), 100
assign_query() (*in module PopPUNK.assign*), 91
assign_query_hdf5() (*in module PopPUNK.assign*), 91
assign_samples() (*in module PopPUNK.models*), 102

B

BGMMFit (*class in PopPUNK.models*), 95

C

calc_prevalence() (*in module PopPUNK.web*), 138
check_and_set_gpu() (*in module PopPUNK.utils*), 133
check_search_range() (*in module PopPUNK.refine*), 116
checkNetworkVertexCount() (*in module PopPUNK.network*), 104
checkSketchlibLibrary() (*in module PopPUNK.sketchlib*), 128
checkSketchlibVersion() (*in module PopPUNK.sketchlib*), 129
cliquePrune() (*in module PopPUNK.network*), 104
ClusterFit (*class in PopPUNK.models*), 96
construct_dense_weighted_network() (*in module PopPUNK.network*), 105
construct_network_from_assignments() (*in module PopPUNK.network*), 105
construct_network_from_df() (*in module PopPUNK.network*), 106
construct_network_from_edge_list() (*in module PopPUNK.network*), 107
construct_network_from_sparse_matrix() (*in module PopPUNK.network*), 108
constructDatabase() (*in module PopPUNK.sketchlib*), 129
copy() (*PopPUNK.models.ClusterFit method*), 96
createDatabaseDir() (*in module PopPUNK.sketchlib*), 130
createMicroreact() (*in module PopPUNK.plot*), 121
cugraph_to_graph_tool() (*in module PopPUNK.network*), 109

D

DBSCANFit (*class in PopPUNK.models*), 97
decisionBoundary() (*in module PopPUNK.utils*), 133
distHistogram() (*in module PopPUNK.plot*), 121
drawMST() (*in module PopPUNK.plot*), 121
dtype (*PopPUNK.models.NumpyShared attribute*), 100
dtype (*PopPUNK.refine.NumpyShared attribute*), 116

E

edge_weights() (*PopPUNK.models.LineageFit method*), 99
evaluate_dbscan_clusters() (*in module PopPUNK.dbscan*), 93
expand_cugraph_network() (*in module PopPUNK.refine*), 116
extend() (*PopPUNK.models.LineageFit method*), 99
extractReferences() (*in module PopPUNK.network*), 109

F

fetchNetwork() (*in module PopPUNK.network*), 109
findBetweenLabel() (*in module PopPUNK.dbscan*), 93
findWithinLabel() (*in module PopPUNK.bgmm*), 91
fit() (*PopPUNK.models.BGMMFit method*), 95
fit() (*PopPUNK.models.ClusterFit method*), 96
fit() (*PopPUNK.models.DBSCANFit method*), 97
fit() (*PopPUNK.models.LineageFit method*), 99
fit() (*PopPUNK.models.RefineFit method*), 101
fit2dMultiGaussian() (*in module PopPUNK.bgmm*), 92

fitDbScan() (*in module PopPUNK.dbscan*), 93
fitKmerCurve() (*in module PopPUNK.sketchlib*), 130

G

generate_embedding() (*in module PopPUNK.mandrake*), 94
generate_minimum_spanning_tree() (*in module PopPUNK.network*), 110
get_cugraph_triangles() (*in module PopPUNK.network*), 110
get_grid() (*in module PopPUNK.plot*), 122
get_match_search_depth() (*in module PopPUNK.utils*), 133
get_vertex_list() (*in module PopPUNK.network*), 111
getCliqueRefs() (*in module PopPUNK.network*), 110
getKmersFromReferenceDatabase() (*in module PopPUNK.sketchlib*), 130
getSeqsInDb() (*in module PopPUNK.sketchlib*), 130
getSketchSize() (*in module PopPUNK.sketchlib*), 130
graphml_to_json() (*in module PopPUNK.web*), 138
growNetwork() (*in module PopPUNK.refine*), 116

H

highlight_cluster() (*in module PopPUNK.web*), 138

I

isolateNameToLabel() (*in module PopPUNK.utils*), 133
iterDistRows() (*in module PopPUNK.utils*), 134

J

joinClusterDicts() (*in module PopPUNK.utils*), 134
joinDBs() (*in module PopPUNK.sketchlib*), 131

L

likelihoodBoundary() (*in module PopPUNK.refine*), 117
LineageFit (*class in PopPUNK.models*), 98
listDistInts() (*in module PopPUNK.utils*), 134
load() (*PopPUNK.models.BGMMFit method*), 96
load() (*PopPUNK.models.DBSCANFit method*), 98
load() (*PopPUNK.models.LineageFit method*), 99
load() (*PopPUNK.models.RefineFit method*), 102
load_network_file() (*in module PopPUNK.network*), 111
loadClusterFit() (*in module PopPUNK.models*), 103
log_likelihood() (*in module PopPUNK.bgmm*), 92
log_multivariate_normal_density() (*in module PopPUNK.bgmm*), 92

M

main() (*in module PopPUNK.assign*), 91
main() (*in module PopPUNK.visualise*), 138
module
    PopPUNK.assign, 91
    PopPUNK.bgmm, 91
    PopPUNK.dbscan, 93
    PopPUNK.mandrake, 94
    PopPUNK.models, 95
    PopPUNK.network, 103
    PopPUNK.plot, 121
    PopPUNK.refine, 116
    PopPUNK.sketchlib, 128
    PopPUNK.sparse_mst, 128
    PopPUNK.utils, 133
    PopPUNK.visualise, 138
    PopPUNK.web, 138
multi_refine() (*in module PopPUNK.refine*), 118

N

name (*PopPUNK.models.NumpyShared attribute*), 100
name (*PopPUNK.refine.NumpyShared attribute*), 116
network_to_edges() (*in module PopPUNK.network*), 111
networkSummary() (*in module PopPUNK.network*), 111
newNetwork() (*in module PopPUNK.refine*), 118
newNetwork2D() (*in module PopPUNK.refine*), 119
no_scale() (*PopPUNK.models.ClusterFit method*), 96
NumpyShared (*class in PopPUNK.models*), 100
NumpyShared (*class in PopPUNK.refine*), 116

O

outputsForCytoscape() (*in module PopPUNK.plot*), 122
outputsForGrapetree() (*in module PopPUNK.plot*), 123
outputsForMicroreact() (*in module PopPUNK.plot*), 123
outputsForPhandango() (*in module PopPUNK.plot*), 124

P

plot() (*PopPUNK.models.BGMMFit method*), 96
plot() (*PopPUNK.models.ClusterFit method*), 97
plot() (*PopPUNK.models.DBSCANFit method*), 98
plot() (*PopPUNK.models.LineageFit method*), 99
plot() (*PopPUNK.models.RefineFit method*), 102
plot_contours() (*in module PopPUNK.plot*), 125
plot_dbscan_results() (*in module PopPUNK.plot*), 125
plot_fit() (*in module PopPUNK.plot*), 125
plot_refined_results() (*in module PopPUNK.plot*), 126
plot_results() (*in module PopPUNK.plot*), 127
plot_scatter() (*in module PopPUNK.plot*), 127
PopPUNK.assign

module, 91
PopPUNK.bgmm
module, 91
PopPUNK.dbscan
module, 93
PopPUNK.mandrake
module, 94
PopPUNK.models
module, 95
PopPUNK.network
module, 103
PopPUNK.plot
module, 121
PopPUNK.refine
module, 116
PopPUNK.sketchlib
module, 128
PopPUNK.sparse_mst
module, 128
PopPUNK.utils
module, 133
PopPUNK.visualise
module, 138
PopPUNK.web
module, 138
print_network_summary() (*in module PopPUNK.network*), 113
printClusters() (*in module PopPUNK.network*), 112
printExternalClusters() (*in module PopPUNK.network*), 113
process_previous_network() (*in module PopPUNK.network*), 113
process_weights() (*in module PopPUNK.network*), 114

Q

queryDatabase() (*in module PopPUNK.sketchlib*), 131

R

read_rlist_from_distance_pickle() (*in module PopPUNK.utils*), 136
readDBParams() (*in module PopPUNK.sketchlib*), 132
readIsolateTypeFromCsv() (*in module PopPUNK.utils*), 134
readManualStart() (*in module PopPUNK.refine*), 120
readPickle() (*in module PopPUNK.utils*), 135
readRfile() (*in module PopPUNK.utils*), 135
RefineFit (*class in PopPUNK.models*), 100
refineFit() (*in module PopPUNK.refine*), 120
removeFromDB() (*in module PopPUNK.sketchlib*), 132

S

save() (*PopPUNK.models.BGMMFit method*), 96
save() (*PopPUNK.models.DBSCANFit method*), 98
save() (*PopPUNK.models.LineageFit method*), 100
save() (*PopPUNK.models.RefineFit method*), 102
save_network() (*in module PopPUNK.network*), 114
set_env() (*in module PopPUNK.utils*), 136
setupDBFuncs() (*in module PopPUNK.utils*), 136
shape (*PopPUNK.models.NumpyShared attribute*), 100
shape (*PopPUNK.refine.NumpyShared attribute*), 116
sketch_to_hdf5() (*in module PopPUNK.web*), 138
sparse_mat_to_network() (*in module PopPUNK.network*), 115
stderr_redirected() (*in module PopPUNK.utils*), 136
storePickle() (*in module PopPUNK.utils*), 136
summarise_clusters() (*in module PopPUNK.web*), 138

T

transformLine() (*in module PopPUNK.utils*), 137
translate_network_indices() (*in module PopPUNK.network*), 115

U

update_distance_matrices() (*in module PopPUNK.utils*), 137

V

vertex_betweenness() (*in module PopPUNK.network*), 115

W

writeClusterCsv() (*in module PopPUNK.plot*), 127
writeReferences() (*in module PopPUNK.network*), 115