mandrake

Release v1.0.0

John Lees and Gerry Tonkin-Hill

Dec 15, 2022

CONTENTS:

mandrake is a tool for creating visualisations of pathogen populations from their genome data. The visualisation produces are optimised to produce clusters of similar sequences, represented in a two dimensional embedding.

You may wish to use this tool to:

- Get a quick look at the structure of your population, and identify possible clusters.
- See if these clusters match with known labels.
- Determine whether supervised learning is likely to work on this input data.
- Make pretty pictures and animations.

mandrake is primarily a visualisation tool. To determine clusters robustly, we would recommend a model-based method such as fastbaps or poppunk.

To understand local embeddings better, we would recommend the following excellent guide: https://distill.pub/2016/misread-tsne/.

It can take as input:

- Assembly or read data (using sketchlib).
- A multiple sequence alignment.
- A gene presence/absence matrix.

Runs the following steps:

1. Distance calculation, and sparsification to k nearest neighbours, or using a threshold.
2. Conversion of distances to conditional probabilities at the specified perplexity.
3. A modified version of stochastic cluster embedding.
4. HDBSCAN on the embedding, or labelling with provided categories.

5. Plots of the output.

Producing the following output:

- A numpy version of the sparse matrix, for reuse.
- A text version of the output embedding.
- An interactive HTML file with the embedding, and hover labels.
- A static version of this embedding.
- A hexbin plot to show density of the embedding (which is usually overplotted).
- (optionally) A video of the embedding process as the algorithm runs.

mandrake is very fast, and can be used on millions of input samples.

CHAPTER

ONE

INSTALLATION

These instructions install a python command line interface to mandrake. If you are not comfortable with using the terminal, or just want to have a quick look, try our web version (and see documenation on *Web app*).

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 mandrake
```

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 mandrake with `conda create -n mandrake_env mandrake`

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

1.2 Installing manually

You will need to install the dependencies, which are listed in `environment.yml`. We would still recommend using conda to do this:

```
conda create -n mandrake_env python
conda env update -n mandrake_env --file environment.yml
conda activate mandrake_env
```

You can then install by running:

```
python setup.py install
```

If you have the CUDA toolkit installed and `nvcc` on your path, this will also compile the GPU code:

```
-- CUDA found, compiling both GPU and CPU code
```

By default the code will be built for CUDA SM versions 70 (V100), 75 (20xx series), 80 (A100) and 86 (30xx series). If you need more help on getting the GPU code to work, please see the page in the pp-sketchlib docs, which uses the same build procedure.

1.2.1 Developer notes

Install the debug build (which can be stepped through with `gdb` and `cuda-gdb`) by running:

```
python setup.py install --debug
```

To run:

```
cuda-gdb python
set args mandrake-runner.py <args>
r
```

To run without installing, run:

```
python setup.py build
```

and add the following lines to the top of each python module file:

```
import os, sys
sys.path.insert(0, os.path.dirname(__file__) + '/../build/lib.macosx-10.9-x86_64-3.9')
sys.path.insert(0, os.path.dirname(__file__) + '/../build/lib.linux-x86_64-3.9')
```

To change the compiler used, edit the following part of `setup.py`:

```
cmake_args = ['-DCMAKE_LIBRARY_OUTPUT_DIRECTORY=' + extdir,
              '-DPYTHON_EXECUTABLE=' + sys.executable,
              '-DCMAKE_C_COMPILER=gcc-10',
              '-DCMAKE_CXX_COMPILER=g++-10',
              '-DCMAKE_VERBOSE_MAKEFILE:BOOL=ON']
```

To profile the GPU code, uncomment the lines under 'Set these to profile' in `CMakeLists.txt` (there are three of these, two at the top, one in the CUDA section) and reinstall. Run nsight-systems with:

```
nsys profile -o sce_<hash> -c cudaProfilerApi --trace cuda,osrt,openmp mandrake <args>
```

nsight-compute with:

```
ncu -c 10 -o sce_<hash> --set full --target-processes all mandrake <args>
```

CHAPTER

TWO

EXAMPLES

Running mandrake on a multiple sequence alignment:

```
mandrake --alignment sub5k_hiv_refs_prrt_trim.fas --kNN 500 --cpus 4 \
--maxIter 30000000
```

Re-running at a different perplexities using these distances:

```
mandrake --output perplexity50 --distances mandrake.npz \
--perplexity 50 --cpus 4 --maxIter 30000000
mandrake --output perplexity5 --distances mandrake.npz \
--perplexity 5 --cpus 4 --maxIter 30000000
```

Running mandrake on a sketch database, and producing an animation:

```
mandrake --sketches mass_sketchlib.h5 --kNN 500 --cpus 4 \
--maxIter 1000000 --animate
```

Running mandrake on gene presence/absence matrix from panaroo:

```
mandrake --accessory gene_presence_absence.Rtab --kNN 50 --cpus 16 \
--perplexity 15 --bInit 0 --maxIter 1000000000
```

Running mandrake on a very large multiple sequence alignment:

```
mandrake --cpus 60 \
--alignment SC2.fasta \
--output sc2million \
--kNN 50 --perplexity 15 --bInit 0 --maxIter 10000000000
```

Re-running mandrake using a GPU:

```
mandrake --cpus 60 \
--distances SC2.fasta \
--output sc2million.npz \
--perplexity 50 --no-clustering\
--n-workers 94976 --maxIter 1000000000000 --use-gpu \
--animate --labels sc2million_pangolin.txt --no-html-labels
```

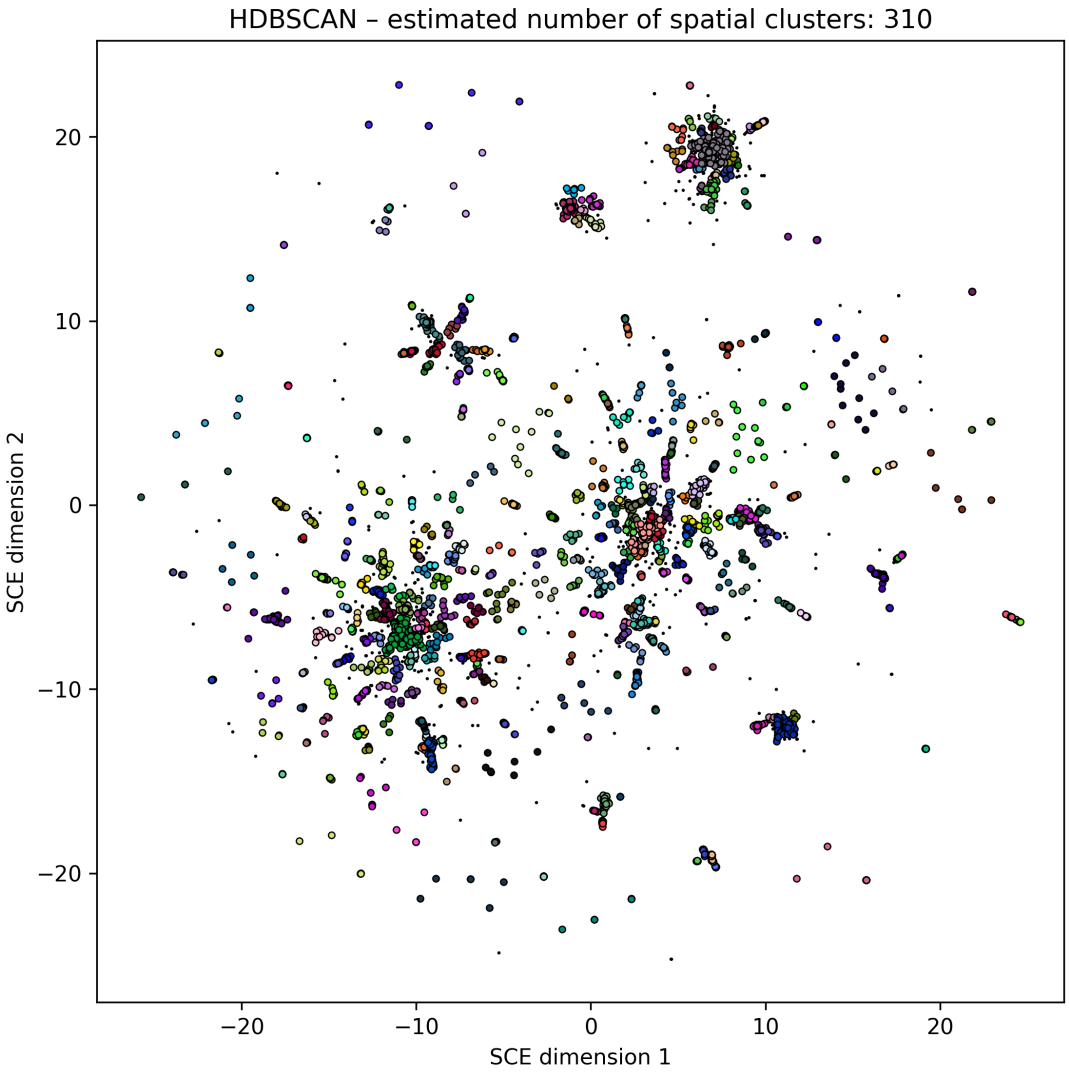


Fig. 1: Embedding of 5k HIV pol sequences (perplexity 15)

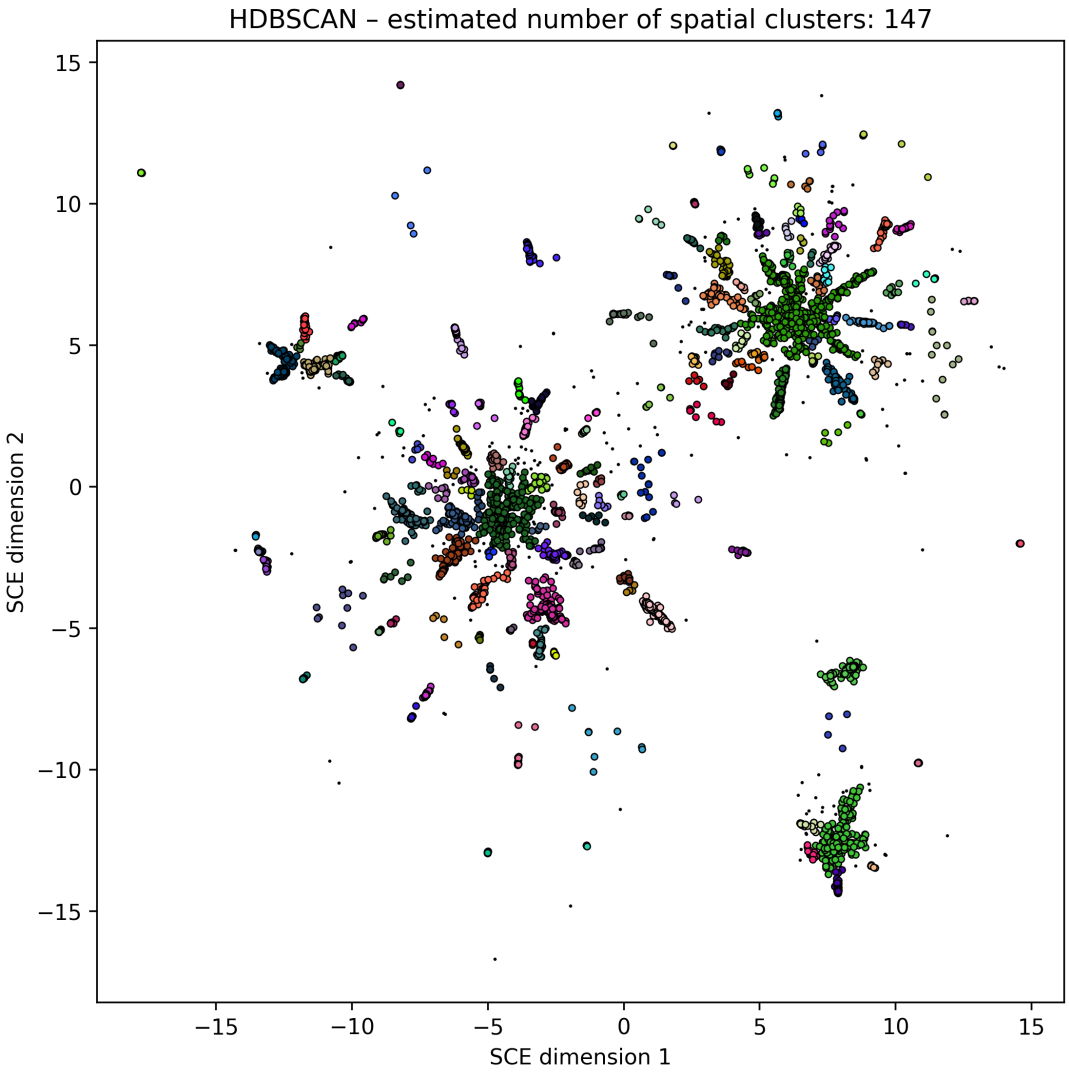


Fig. 2: Embedding of 5k HIV pol sequences (perplexity 50)

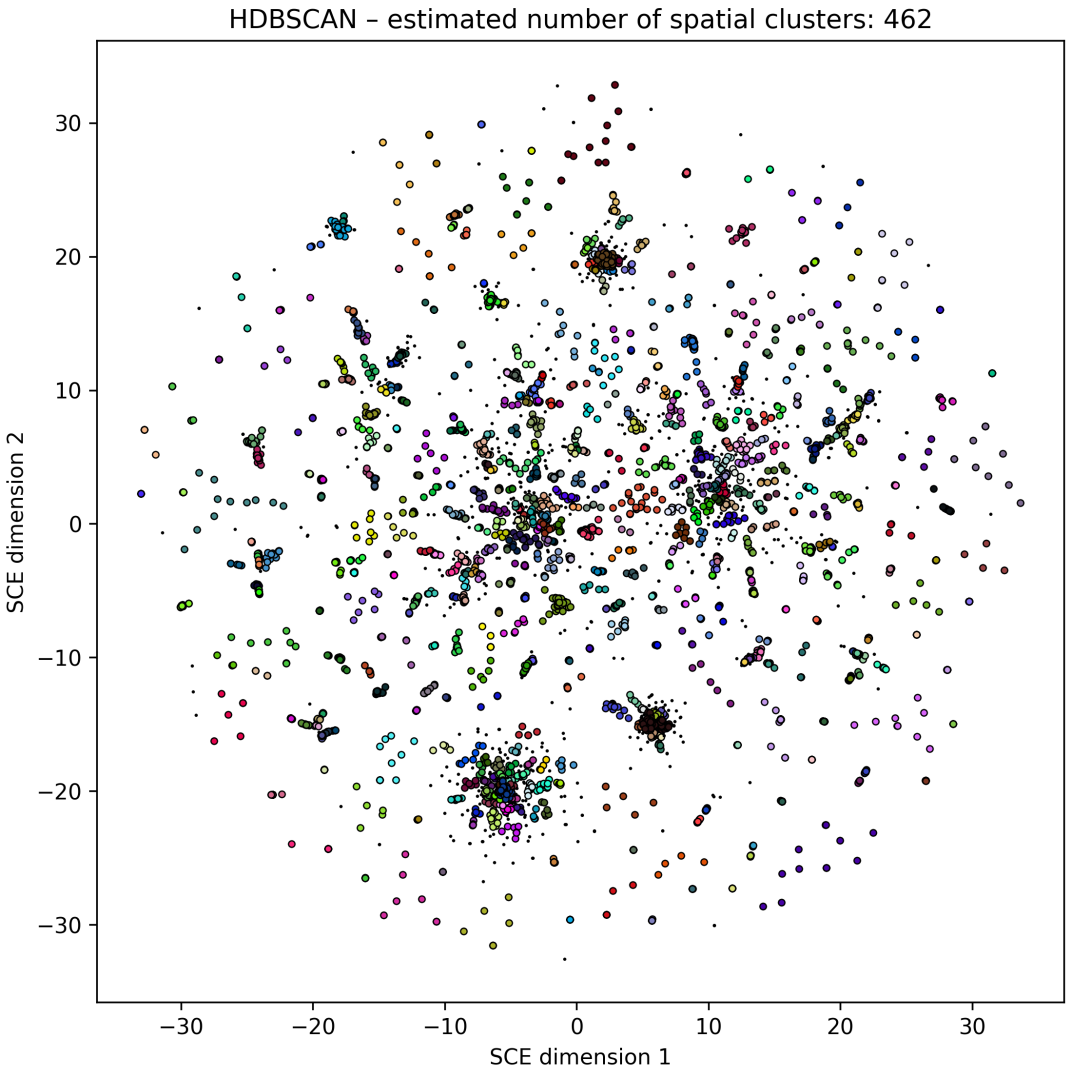


Fig. 3: Embedding of 5k HIV pol sequences (perplexity 5)

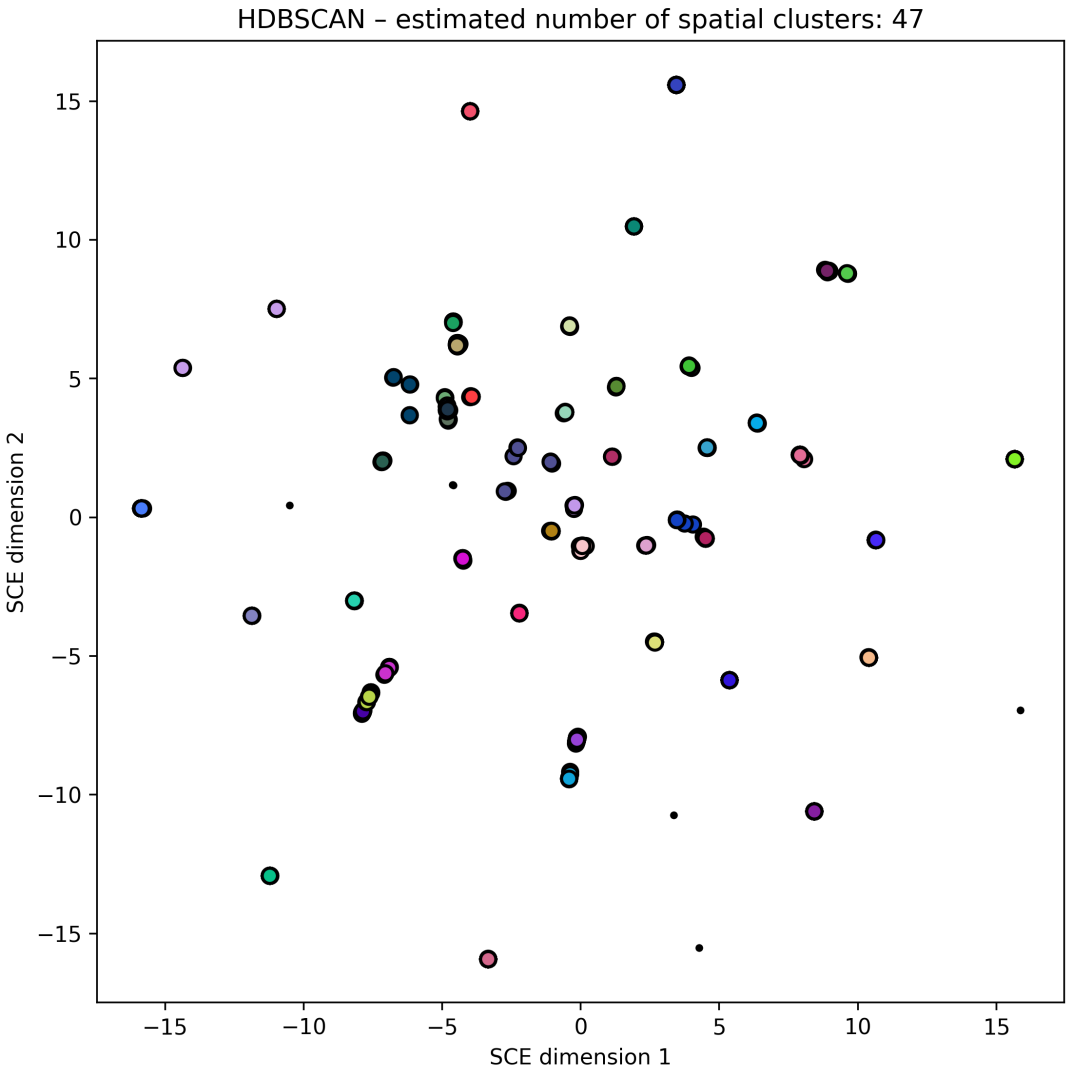


Fig. 4: Embedding of 616 *S. pneumoniae* genome core distances

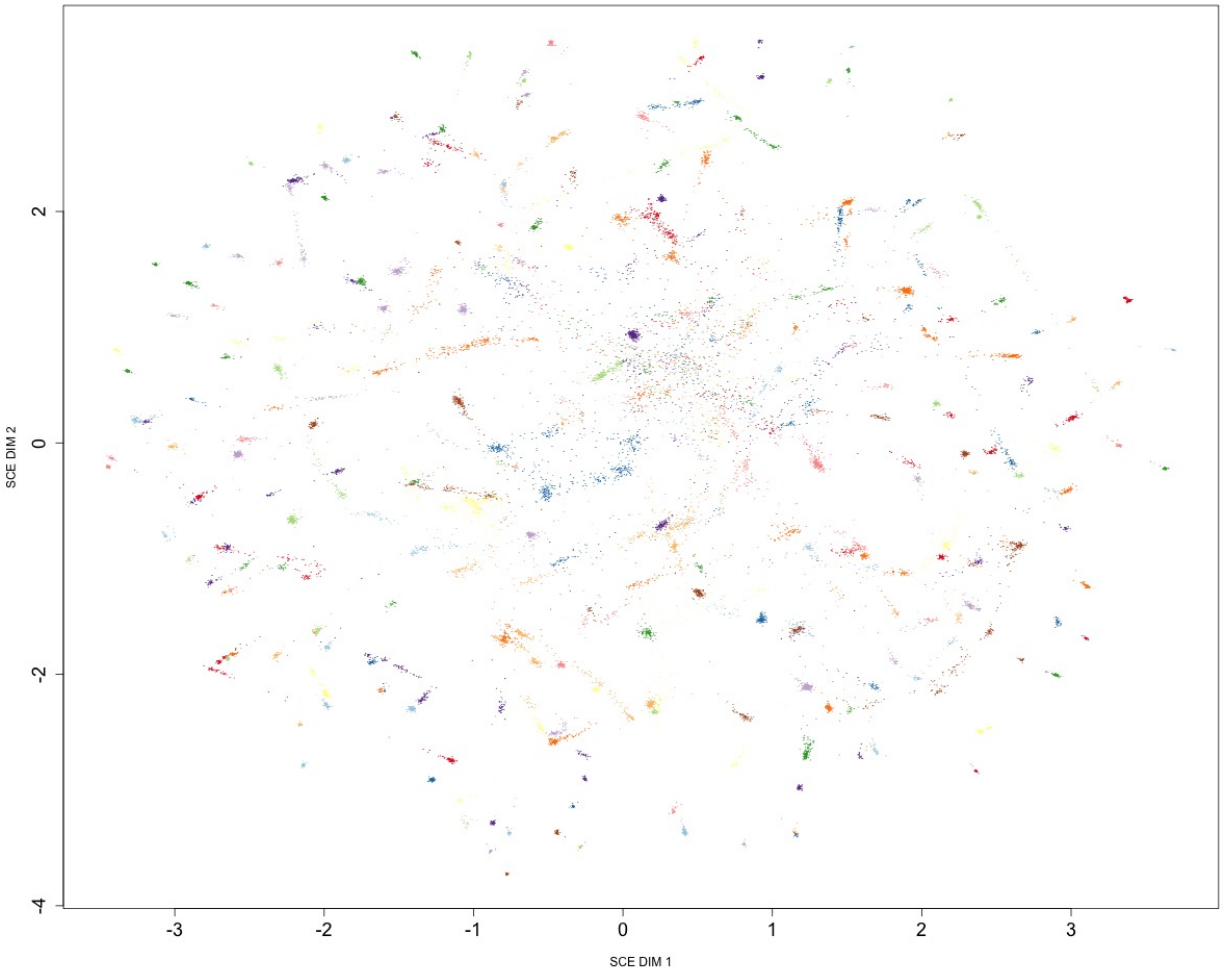


Fig. 5: Embedding of ~20k *S. pneumoniae* accessory genome distances

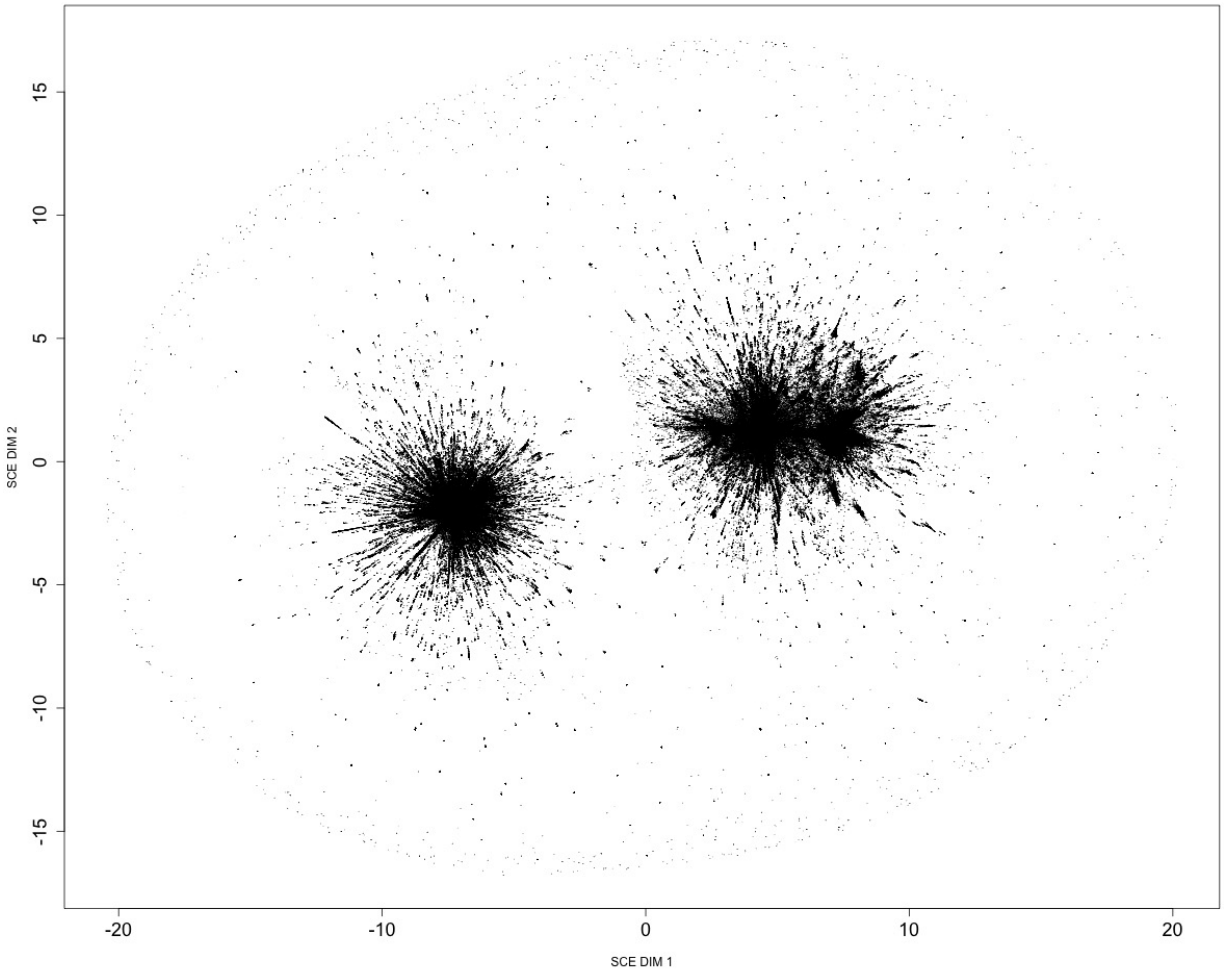


Fig. 6: Embedding of ~1M SARS-CoV-2 genomes

CHAPTER

THREE

SCE PARAMETERS

The options for the SCE algorithm are explained below, other options are explained in *Input options*, *Creating animations* and *Parallelisation*.

The progress bar shows the percentage complete, the current learning rate η, the divergence Eq (and clash rate, see *Parallelisation*).:

```
Optimizing   Progress: 99.9%, eta=0.0010, Eq=0.0547636151, clashes=0.1%
Optimizing done in 20s
```

The algorithm should be run until Eq stabilises.

SCE options:

```
--perplexity PERPLEXITY
                      Perplexity for distance to similarity conversion [default = 15]
--no-preprocessing    Turn off entropy pre-processing of distances
--weight-file WEIGHT_FILE
                      Weights for samples
--maxIter MAXITER     Maximum SCE iterations [default = 100000]
--nRepuSamp NREPUSAMP
                      Number of neighbours for calculating repulsion (1 or 5) [default =␣
↪5]
--eta0 ETA0           Learning rate [default = 1]
--bInit BINIT         1 for over-exaggeration in early stage [default = 0]
--no-clustering       Turn off HDBSCAN clustering after SCE
```

- `--perplexity` roughly sets the balance between global and local structure, smaller values making fewer neighbours matter, and emphasising local structure. Typical values are between 5 and 50, but making plots at multiple values is often useful.
- `--no-preprocessing` uses raw similarities as input, rather than a probability distribtion using a desired perplexity. Do not use this option unless you are having issues setting the perplexity.
- `--weight-file` allows you to give different samples different weights in being picked to attract or repel. The default is to equally weight samples, but you could for example weight by cluster size or its inverse. This requires a tab-separated file with no header, the first column with sample names, the second column with their weights.
- `--maxIter` sets how long the algorithm will run for. Larger datasets will need more iterations. We recommend running until Eq stabilises.
- `--nRepuSamp` is the number of neighbours used for calculating the repulsion at each iteration. This can be one or five.
- `--eta0` sets the scale of the learning rate. A higher value will move points around more at each iteration.

- `--bInit` turns on over-exaggeration, which sets the stength of attraction four times higher in the first 10% of the iterations.
- `--no-clustering` turns off the HDBSCAN clustering if you don't want spatial clustering run on the embedding (which may stall when no structure has been found). This is implied if `--labels` have been provided.

CHAPTER

FOUR

INPUT OPTIONS

Any of the following modes also accept `--labels`, which give the categories to colours points by in the output plots. This should be a tab-separated file with no header, the first column containing sample names, and the second column containing the labels for each sample.

There are two additional parameters for the distances:

```
--threshold THRESHOLD
                      Maximum distance to consider [default = None]
--kNN KNN             Number of k nearest neighbours to keep when sparsifying the␣
↪distance
                      matrix.
```

Rather than using a full (dense) matrix of all pairwise distances, mandrake uses a sparse matrix, ignoring large distances. This uses significantly less memory without affecting results.

`--kNN` sets the number of distances to keep for each sample, which will be the k closest. Set k to a number smaller than the number of samples. Memory use grows linearly with k. Setting k too small will miss global structure in the data.

`--threshold` instead picks a maximum distance that is considered meaningful, and larger distances will be removed from the input.

4.1 Multiple sequence alignment

Provide a multi-fasta alignment with `--alignment`. Distances will be calculated using a modified form of the pairsnp algorithm, and sparsified based on `-kNN` or `--threshold`.

If you are trying to align large numbers of sequences (e.g. SARS-CoV-2), the reference-guided mode of MAFFT may be helpful:

```
mafft --6merpair --thread -1 --keeplength --addfragments filtered_SC2.fasta \
nCoV-2019.reference.fasta > MA_filt_SC2.fasta
```

4.2 Sketch database (assemblies or reads)

Provide a pp-sketchlib database with `--sketches`, to calculate core and accessory distances between the sketches. Core distances are used by default, but add `--use-accessory` to alter this behaviour.

This should be a HDF5 file with suffix `.h5` produced by sketchlib, for example by a command such as:

```
sketchlib sketch -l sample_rfile.txt -o sketch_db --cpus 16
```

4.3 Gene presence/absence

Pan-genome programs such as roary and panaroo output a `gene_presence_absence.Rtab` file, which can be used with `--accessory` to calculate accessory distances (Hamming distances).

unitig counting programs such as unitig-caller also output this file format, though the interpretation of the distances is slightly different it can also be used as input.

4.4 Precalculated distances

After calculating distances, mandrake will save these as `<output_prefix>.npz`. These can be used as input without the need to compute them again with `--distances`, which is useful when you wish to run the embedding algorithm on the same data with different parameters.

CHAPTER

FIVE

PLOTS

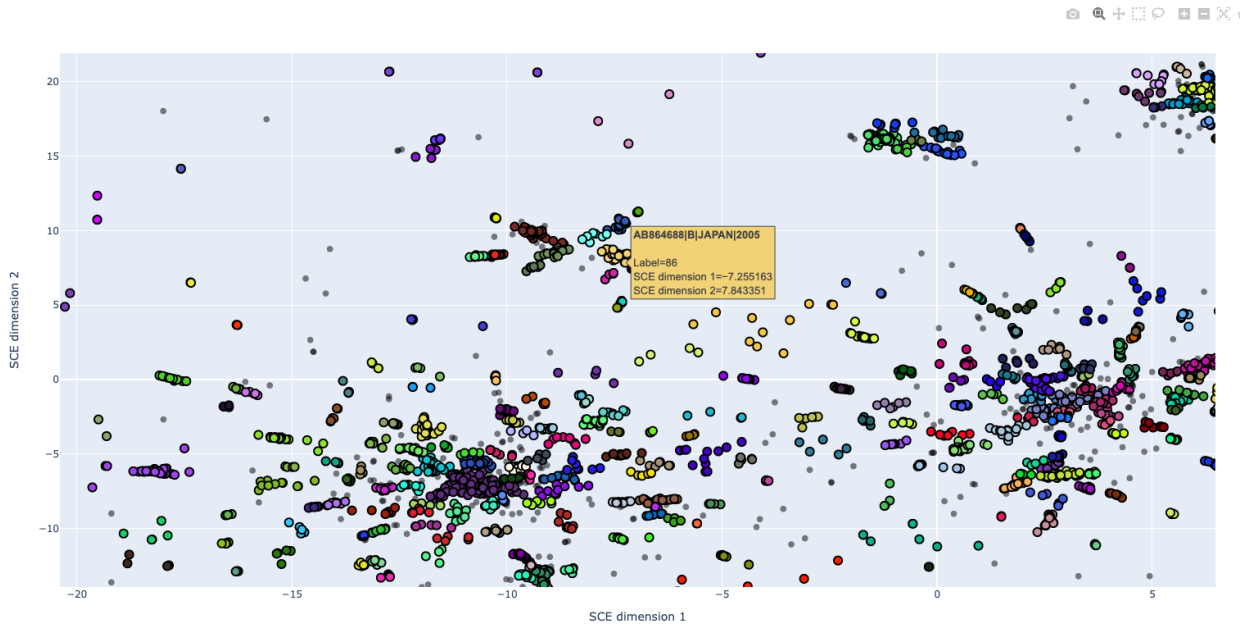
This page describes the outputs from mandrake. Along with the plots, mandrake also outputs the embedding as:

- `mandrake.embedding.txt` the X-Y coordinates of the embedding as text.
- `mandrake.names.txt` the sample names of each row in the embedding file.

Should you wish to create your own plots.

5.1 Interactive plot

An interactive plot of the embedding is written to `mandrake.embedding.html`, which can be viewing in your web browser. This plot can be zoomed and panned, and sample labels will appear when you hover over them. Use the controls in the top-right to save your view as a static image. Double-click to zoom out.



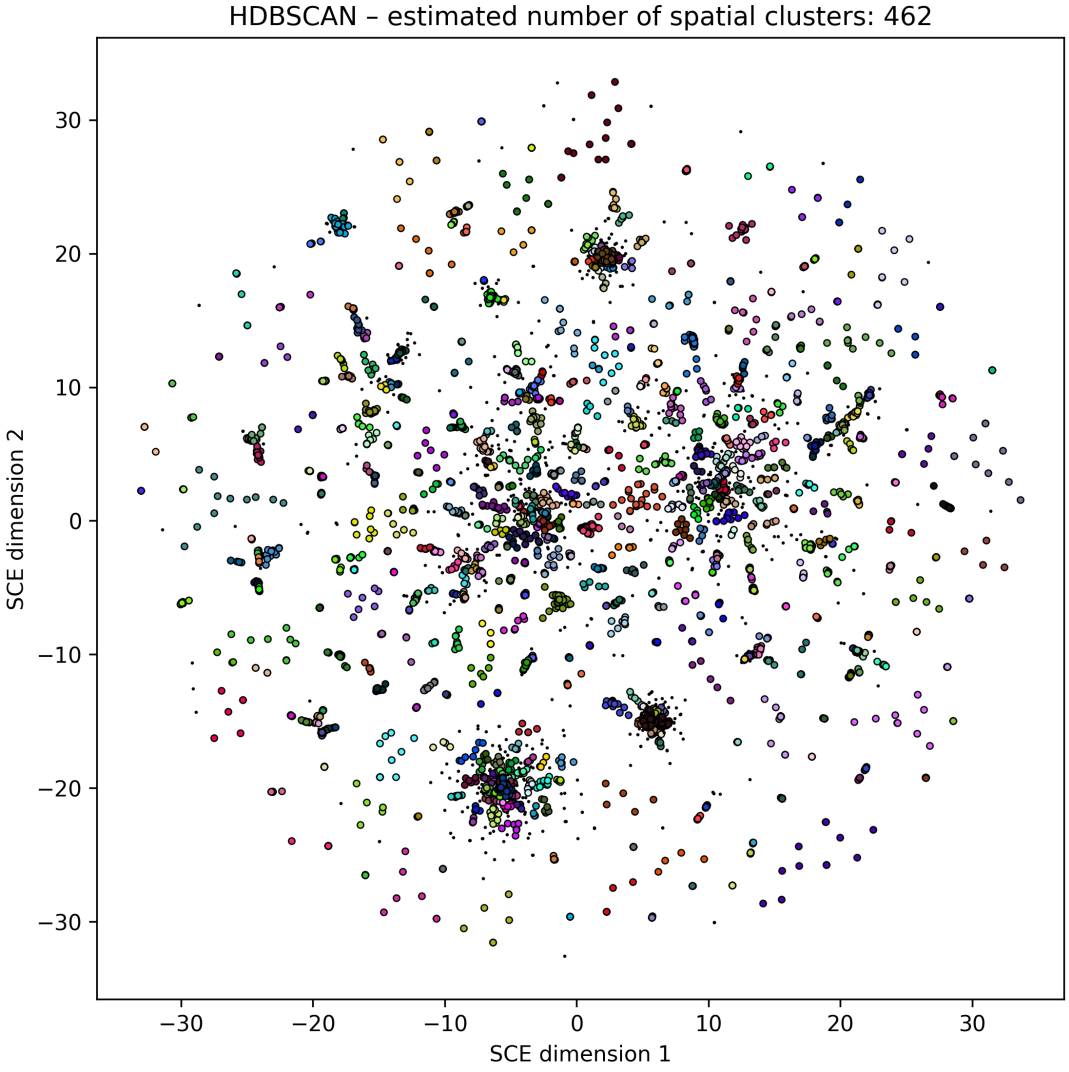
For large datasets this file may become very large due to the labels, in which case the static images might be preferred.

5.2 Static image

This is a non-interactive version of the embedding written to `mandrake.embedding_static.png`. Point size will vary depending on number of samples N:

- Small dataset, large points $N < 1000$.
- Medium dataset, medium points $1000 < N < 10000$.
- Large dataset, small points $N > 10000$.

Points will be coloured by their labels if provided, or otherwise their HDBSCAN cluster (total number noted in the title). Colours are chosen at random, but consistently between runs:

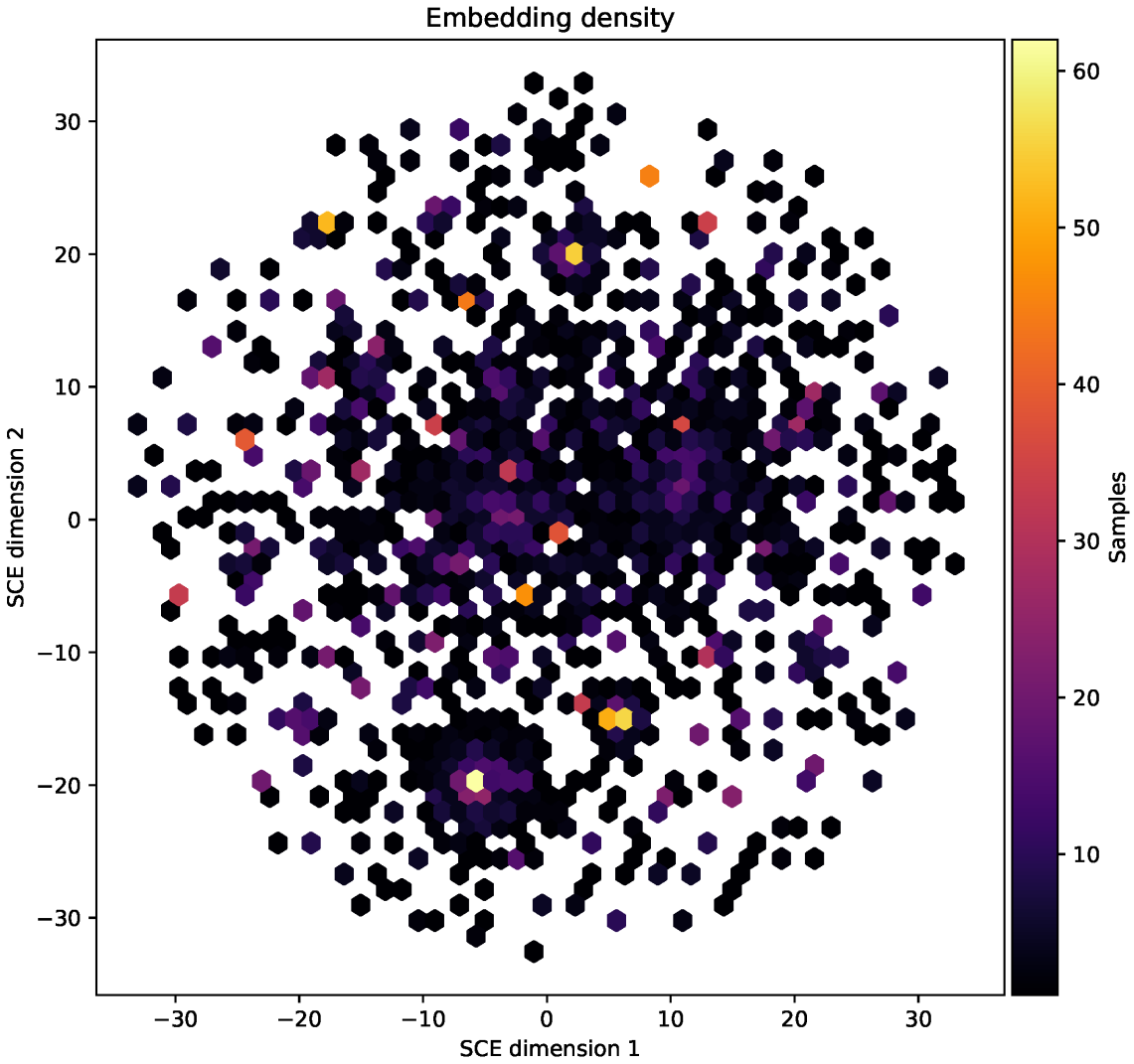


5.2.1 Changing colours in the plots

The colours for each label are chosen at (pseudo)random, so while they will be consistent between runs, they may also be fairly putrescent. To try a new set of colours, give a different `--seed`. This should be an integer, and setting `--seed` 1 will be the default colours.

5.3 Density plot

Due to the nature of the SCE algorithm, many points are likely to be overploted. This can be investiaged in the HTML plot by zooming in. For larger datasets, a useful companion to the static image is the hexbin plot `mandrake.embedding_density.pdf`, which shows the number of points in each region of the plot, divided into small tesselating hexagons:



5.4 Microreact output

You can also view your clusters in microreact, which will also let you combine the embedding visualisation with a tree, interactively.

Upload the `mandrake.embedding.dot` file. If you ran HDBSCAN clustering there will also be `mandrake.hdbscan_clusters.csv` file you can include to colour the points with.

If you used your own `--labels`, these can be used, as long as they are formatted as a Microreact-compliant .csv file:

- comma separated
- with a header
- sample names have column header 'id'
- append *__autocolour* to column headers you wish to colour points by

CHAPTER

SIX

WEB APP

You can find a version of mandrake at https://gtonkinhill.github.io/mandrake-web/

This is a static web app which uses a WebAssembly version of mandrake to run in your browser. This means no sequence data is transmitted across the network and the analysis is run entirely locally in your browser, which:

- Means data does not have to be uploaded, which can take a while.
- We see and store no sequence, so your data never leaves your machine, or is ever even read by us at all.
- We don't have to maintain a server (which is good for us), which is typically more reliable (which good for you).

The use of this is hopefully fairly self-explanatory:

1. Choose an input sequence file.
2. Select the type of distance based on the type of the input file (SNP distances, accessory or sketch; see *Input options*).
3. (optionally) set labels to colour the output with.
4. (optionally) change the SCE algorithm parameters.

6.1 Performance

The web app is suitable for up to a few thousand sequences. If you are analysing more sequences than that you should use the command line tool.

The web app can only use a single CPU core. It is usually faster in 'native' browers, Safari on OS X and Edge on Windows.

CHAPTER

SEVEN

CREATING ANIMATIONS

You can also create an animation of the embedding process as it runs simply by adding the `--animate` flag. This works with both the CPU and GPU versions of the algorithm, and will output a video file `mandrake.embedding_animation.mp4`:

The top panel shows the embedding, the bottom panels shows the iteration and Eq at the iteration. The learning rate always decreases linearly, so it is not plotted.

You will see two further progress bars after plotting:

```
Creating animation
100%|| 400/400 [00:11<00:00, 35.73frames/s]
Saving frame 400 of 400
```

The first is saving static images of each frame, the second is encoding these into a video using `ffmpeg`.

You can also add sound by additing the `--animate-sound` flag. This will add a soundtrack to the video which is based on how the animation changes at each frame (see below).

7.1 Details

- The colours are the 'final' colours of HDBSCAN run on the embedding result, or the provided labels. Black points are noise points.
- The dimensions are rescaled to have unit standard deviation in each direction at every frame.
- Animations have 400 frames played at 20fps, resulting in a 20s animation.
- Resolution is 1920x2560px.
- Samples are taken more regularly at the start, when learning is happening at a greater rate and points move more, and less frequently at the end, when learning is slow and points move less. Specifically, samples are taken at a rate such that the total amount of learning is divided equally, which is proportional to $1 - \sqrt{1 - x}$.
- Sound is created by adding a note with frequency proportional to the maximum change of any point between each frame. The x-axis gives the left channel, the y-axis gives the right channel.

CHAPTER

EIGHT

PARALLELISATION

To increase the speed of mandrake, you can alter the following options:

```
--n-workers N_WORKERS
                      Number of workers to use, sets max parallelism [default = 1]
--cpus CPUS           Number of CPUs to use [default = 1]
--use-gpu             Run SCE on the GPU. If this fails, the CPU will be used [default =
                      False]
--device-id DEVICE_ID
                      GPU ID to use
--blockSize BLOCKSIZE
                      CUDA blockSize [default = 128]
```

Set the number of cores to use by specifying `--cpus`. This will set the parallelism for the SCE (embedding), but also carries through to the distance calculation.

To use a GPU you will need: - A CUDA enabled GPU available with appropriate drivers. - A version of the code compiled with CUDA. - To add the `--use-gpu` flag.

A GPU will also be used for sketch distances, if `pp-sketchlib` is also set up to use GPUs. You can choose a GPU with the `--device-id` flag, or the `CUDA_VISIBLE_DEVICES` environment variable (in which case keep device ID as the default; this is how some HPCs set the GPU).

The float precision on GPUs can be set with `--fp`, 32 for single precision, 64 for double precision. On 'consumer' GPUs, using `--fp 32` will likely be faster. You can also change the block size for the SCE process with `--blockSize` (this should be a multiple of 32, and likely one of 32, 64, 128 or 256). The default is 128, and you probably don't need to change this unless you are interested in CUDA.

8.1 Setting the number of workers

The number of workers `--n-workers` sets the maximum amount of parallelism possible. Each worker will randomly pick a sample pair to update the attraction between, and then `--nRepuSamp` pairs to update the repulsion between. These can be run in parallel at each iteration.

If you are running with CPU cores, you probably want to set the number of workers equal to the number of available cores (this will be done automatically). If you are running on a GPU you should set at least as many workers as the block size. However, GPUs become more efficient with more threads, up to about 100k threads. So you want to set as many workers as possible, ideally up to 100k (and probably an integer multiple of the block size).

However, the more workers running in parallel there are relative to the number of samples, the more likely two or more workers will try and update the same sample at the same time. The frequency of these attempted overwrites is shown by the percentage of clashes in the progress bar.

The CPU code largely avoids multiple synchronous worker updates, but you may find that speedup decreases as clash rate increases. The GPU code does not correct for clashes (but does maintain memory validity), instead relying on the stochastic nature of the algorithm to give a correct result. Speed is likely to decreases with clashes, but at large values you may find issues with convergence. If the clash rate is approaching 100%, you should probably decrease the number of workers, even if this takes you below the optimum for a GPU.

CHAPTER

NINE

CITATIONS

Mandrake paper:

Lees JA, Tonkin-Hill G, Yang Z, Corander J. Mandrake: visualizing microbial population structure by embedding millions of genomes into a low-dimensional representation. *Philosophical Transactions of The Royal Society B*. 2022;377: 20210237.

https://doi.org/10.1098/rstb.2021.0237

Stochastic Cluster Embedding paper:

Yang Z, Chen Y, Sedov D, Kaski S, Corander J. Stochastic cluster embedding. Stat Comput 33, 12 (2023).

https://doi.org/10.1007/s11222-022-10186-z

Original SCE code:

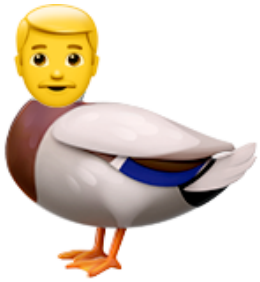
https://github.com/rozyangno/sce

CHAPTER

TEN

MISCELLANEOUS

Would a man wear his drake like this?



or like this?

10.1 Is the name a *Harry Potter* reference?

Not intentionally

10.2 Why mandrake then?

Initially a Final Fantasy enemy mandragora.

mandrakes are a real plant with a name we thought sounded a bit more memorable. There are a few myths about them which the Harry Potter ones are based on.

10.3 What's the bacronym?

A recursive acronym:

Mandrake **A**nalyses **N**ucleotides, **D**imension **R**educes, **A**nimates **K**-neighbour **E**mbeddings

(not really)

- genindex
- search