StripePy Documentation

Andrea Raffo

Dec 10, 2025

INSTALLATION

StripePy is a CLI application written in Python that recognizes architectural stripes found in the interaction matrix files generated by Chromosome Conformation Capture experiments, such as Hi-C and Micro-C.

CHAPTER

ONE

DOCUMENTATION FORMATS

You are reading the PDF version of the documentation.

The live HTML version of the documentation is available at https://stripepy.readthedocs.io/en/v1.3.0/.

Installation

StripePy can be installed using pip or conda with e.g., `pip install 'stripepy-hic[all]'`. Refer to *Installation* for more details.

How to cite this project?

Please use the following BibTeX template to cite StripePy in scientific discourse:

```
@article{stripepy,
  author = {Raffo, Andrea and Rossini, Roberto and Paulsen, Jonas},
  title = {{StripePy: fast and robust characterization of architectural stripes}},
  journal = {Bioinformatics},
  volume = {41},
  number = {6},
  pages = {btaf351},
  year = {2025},
  month = {06},
  issn = {1367-4811},
  doi = {10.1093/bioinformatics/btaf351},
  url = {https://doi.org/10.1093/bioinformatics/btaf351},
  eprint = {https://academic.oup.com/bioinformatics/article-pdf/41/6/btaf351/63484367/
↪btaf351.pdf},
}
```

CHAPTER

TWO

INSTALLATION

StripePy can be installed in various ways.

2.1 Installing with pip

```
pip install 'stripepy-hic[all]'
```

2.2 Installing with conda

```
conda create -n stripepy -c conda-forge -c bioconda stripepy-hic
```

2.3 Installing from source

Installing from source requires git to be available on the host.

2.3.1 Installing the latest version from the main branch

```
pip install 'stripepy-hic[all] @ git+https://github.com/paulsengroup/StripePy.git@main
↪'
```

2.3.2 Installing version corresponding to a git tag

```
pip install 'stripepy-hic[all] @ git+https://github.com/paulsengroup/StripePy.git@v1.
↪3.0'
```

2.3.3 Installing from a release archive

```
pip install 'stripepy-hic[all] @ https://pypi.python.org/packages/source/s/stripepy_
↪hic/stripepy_hic-1.3.0.tar.gz'
```

2.4 Containers (Docker or Singularity/Apptainer)

First, ensure you have followed the instructions on how to install Docker or Singularity/Apptainer on your OS.

The following instructions assume you have root/admin permissions.

- Linux
- macOS
- Windows

On some Linux distributions, simply installing Docker is not enough. You also need to start (and optionally enable) the appropriate service(s). This is usually done with one of the following:

```
sudo systemctl start docker
sudo systemctl start docker.service
```

Refer to Docker or your OS/distribution documentation for more details.

2.5 Pulling stripepy Docker image

stripepy Docker images are available on GHCR.io and DockerHub.

Downloading and running the latest stable release can be done as follows:

```
# Using Docker, may require sudo
user@dev:/tmp$ docker run ghcr.io/paulsengroup/stripepy:1.3.0 --help

# Using Singularity/Apptainer
user@dev:/tmp$ singularity run ghcr.io/paulsengroup/stripepy:1.3.0 --help

usage: stripepy [-h] [-v] {call,download,plot,view} ...

stripepy is designed to recognize linear patterns in contact maps (.hic, .mcool, .
↪cool) through the geometric reasoning, including topological persistence and quasi-
↪interpolation.

options:
  -h, --help            show this help message and exit
  -v, --version         show program's version number and exit

subcommands:
  {call,download,plot,view}
                        List of available subcommands:
    call                stripepy works in four consecutive steps:
                        • Step 1: Pre-processing
                        • Step 2: Recognition of loci of interest (also called 'seeds
↪')
                        • Step 3: Shape analysis (i.e., width and height estimation)
                        • Step 4: Signal analysis
    download            Helper command to simplify downloading datasets that can be␣
↪used to test StripePy.
    plot                Generate various static plots useful to visually inspect the␣
↪output produced by stripepy call.
    view                Fetch stripes from the HDF5 file produced by stripepy call.
```

The above will print stripepy's help message, and is equivalent to running `stripepy --help` from the command line (assuming stripepy is available on your machine).

CHAPTER

THREE

QUICKSTART

StripePy is organized into a few subcommands:

- *stripepy download*: download a minified sample dataset suitable to quickly test StripePy.
- *stripepy call*: run the stripe detection algorithm and store the identified stripes in a `.hdf5` file.
- *stripepy view*: take the `result.hdf5` file generated by *stripepy call* and extract stripes in BEDPE format.
- *stripepy plot*: generate various kinds of plots to inspect the stripes identified by *stripepy call*.

3.1 Walkthrough

The following is a synthetic example of a typical run of StripePy. The steps outlined in this section assume that StripePy is running on a UNIX system. Some commands may need a bit of tweaking to run on Windows.

3.1.1 1) Download a sample dataset (optional)

If you need to download the example matrix used here, you can do so by running:

```
user@dev:/tmp$ stripepy download --name 4DNFI9GMP2J8
```

Feel free to use your own interaction matrix instead of `4DNFI9GMP2J8.mcool`. Please make sure the matrix is in `.cool`, `.mcool`, or `.hic` format.

A more extended description of the subcommand *stripepy download* is found in *Downloading sample datasets*.

3.1.2 2) Detect architectural stripes

The *stripepy call* subcommand is the core of the analysis, designed to identify architectural stripes within contact maps. This process can be quite time-consuming, especially when working with large files.

The path to your contact map file and the desired resolution are required to run the analysis. For instance, to analyse the `4DNFI9GMP2J8.mcool` file at a 10,000 bp resolution, you would use:

```
user@dev:/tmp$ stripepy call 4DNFI9GMP2J8.mcool 10000
```

The command will output a single HDF5 file (e.g., `4DNFI9GMP2J8.10000.hdf5`).

Additional information is provided in *Detect architectural stripes*.

3.1.3 3) Fetch stripes in BEDPE format

Stripe coordinates can be fetched from the `.hdf5` file using *stripepy view*, as in

```
user@dev:/tmp$ stripepy view 4DNFI9GMP2J8.10000.hdf5 > stripes.bedpe
```

Further details can be found in *Fetch architectural stripes*.

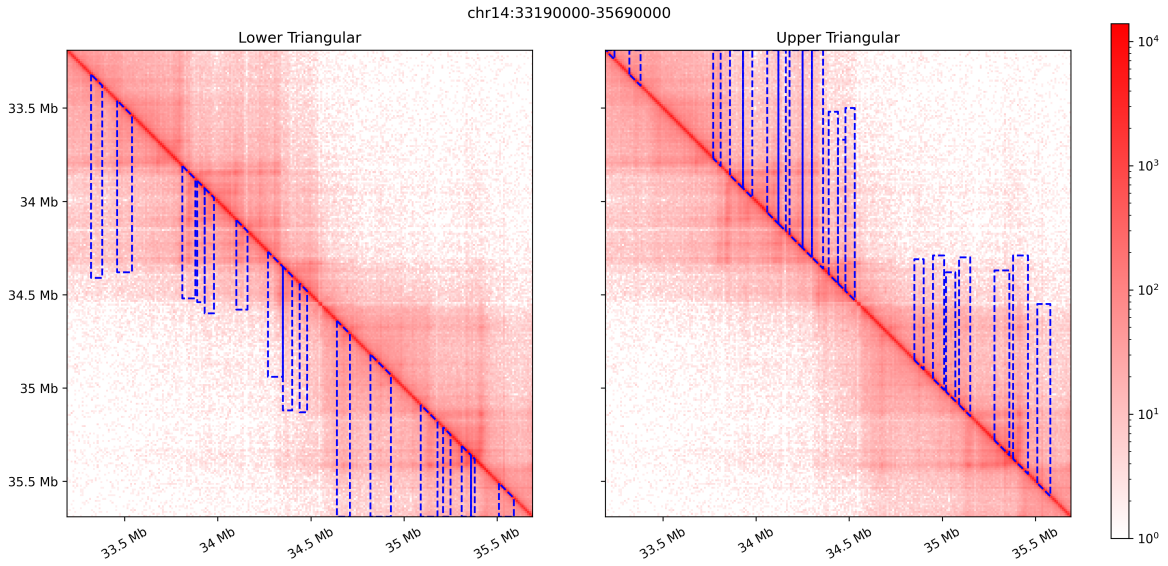
3.1.4 4) Generating plots

StripePy comes with a `plot` subcommand that can be used to visualize architectural stripes overlaid on top of the Hi-C matrix. *stripepy plot* can also generate several graphs showing the general properties of the called stripes, see *Generating plots* for a complete overview.

For instance, running

```
user@dev:/tmp$ stripepy plot cm 4DNFI9GMP2J8.mcool 10000 /tmp/matrix_with_stripes.png␣
↪--stripepy-hdf5 4DNFI9GMP2J8.10000.hdf5 --highlight-stripes
```

will generate the following plot



3.2 Accessing stripes and descriptors from Python

If you are working in Python, you might want to carry out analysis on the stripes and their biodescriptors. The `ResultFile` class helps load and process HDF5 files (e.g., `4DNFI9GMP2J8.10000.hdf5`) generated by StripePy.

The following code snippet can be used to load lower-triangular stripes over the whole genome:

```
In [1]: from stripepy.data_structures import ResultFile

In [2]:  with ResultFile("4DNFI9GMP2J8.10000.hdf5") as f:
   ...:      df = f.get(
   ...:          chrom="chr1",      # Pass None to fetch data for all chromosomes
   ...:          field="stripes",  # See API docs for a complete list of supported␣
↪fields
   ...:          location="LT",     # Use "UT" to fetch from the upper-triangle
   ...:      )
   ...:

In [3]: df
Out[3]:
       seed  top_persistence  left_bound  right_bound  top_bound  ...  outer_lmean ␣
↪outer_rmean  outer_mean  rel_change  cfx_of_variation
0        93         0.398490          91           96         93  ...     0.180769  ␣
↪ 0.240014    0.210392   19.138436          0.563444
1       102         0.053084          99          105        102  ...     0.250077  ␣
↪ 0.246783    0.248430    1.276074          0.605748
2       108         0.082636         106          111        108  ...     0.251255  ␣
```

(continues on next page)

(continued from previous page)

```
↪ 0.242434    0.246845    6.744239          0.629097
3       116         0.103803         114          119        116  ...     0.452872  ␣
↪ 0.395339    0.424105    3.394272          0.394917
4       130         0.073611         126          132        130  ...     0.235412  ␣
↪ 0.249025    0.242219    3.656868          0.608349
...     ...              ...         ...          ...        ...  ...          ...  ␣
↪       ...         ...         ...              ...
1743  24693         0.057216       24687        24695      24693  ...     0.274141  ␣
↪ 0.284040    0.279090    5.741370          0.382488
1744  24708         0.048084       24706        24710      24708  ...     0.280574  ␣
↪ 0.322965    0.301770    7.036960          0.354274
1745  24720         0.044175       24718        24723      24720  ...     0.162981  ␣
↪ 0.155803    0.159392    5.192390          0.833381
1746  24733         0.054484       24730        24737      24733  ...     0.181836  ␣
↪ 0.191120    0.186478    0.238297          0.791300
1747  24793         0.052317       24790        24796      24793  ...     0.168377  ␣
↪ 0.219650    0.194013    7.811918          0.518017

[1748 rows x 22 columns]
```

CHAPTER

FOUR

DOWNLOADING SAMPLE DATASETS

stripepy download is used to download sample datasets suitable to quickly test StripePy. It provides various options to specify which datasets to download, where to store them, and how to handle existing files.

4.1 Listing available datasets

Before downloading, you might want to see the available datasets. The `--list-only` options prints a nested JSON object without initiating any downloads.

The outermost JSON object has dataset names as keys. Each value is an object containing metadata like URL, assembly, and file format.

```
user@dev:/tmp$ stripepy download --list-only

{
  "4DNFI3RFZLZ5": {
    "url": "https://zenodo.org/records/15301784/files/4DNFI3RFZLZ5.stripepy.mcool?
↪download=1",
    "md5": "f6e060211c95dd5fbf6e708c637d1c1c",
    "assembly": "mm10",
    "format": "mcool",
    "size_mb": 83.85
  },
  ...
  "ENCFF993FGR": {
    "url": "https://zenodo.org/records/15301784/files/ENCFF993FGR.stripepy.hic?
↪download=1",
    "md5": "3bcb8c8c5aac237f26f994e0f5e983d7",
    "assembly": "hg38",
    "format": "hic",
    "size_mb": 185.29
  }
}
```

4.2 Basic download operations

stripepy download can be used to download a dataset by specifying its name.

For example, to download the dataset named `4DNFI9GMP2J8`:

```
# This may take a while on slow internet connections
user@dev:/tmp$ stripepy download --name 4DNFI9GMP2J8

2025-01-14 12:46:01.304277 [info     ] downloading dataset "4DNFI9GMP2J8"␣
↪(assembly=hg38)...
```

(continues on next page)

(continued from previous page)

```
2025-01-14 12:46:23.900411 [info     ] DONE! Downloading dataset "4DNFI9GMP2J8" took␣
↪22.596s.
2025-01-14 12:46:23.901141 [info     ] computing MD5 digest for file "/tmp/
↪4DNFI9GMP2J8.dvizz7v1"...
2025-01-14 12:46:24.050566 [info     ] MD5 checksum match!
2025-01-14 12:46:24.050695 [info     ] successfully downloaded dataset "https://
↪zenodo.org/records/14643417/files/4DNFI9GMP2J8.stripepy.mcool?download=1" to file
↪"4DNFI9GMP2J8.mcool"
2025-01-14 12:46:24.050752 [info     ] file size: 106.84MB. Elapsed time: 22.979s
```

Note that, if the dataset already exists, it will be overwritten if and only if the `--force` flag is specified; otherwise, the command will fail to prevent accidental data loss.

If no name is provided, the tool will default to selecting a random dataset. You can refine this random selection by specifying parameters like the maximum allowed size:

```
# This may take a while on slow internet connections
user@dev:/tmp$ stripepy download --max-size 100

2025-07-02 12:29:43.095292 [info     ] downloading dataset "4DNFI3RFZLZ5"␣
↪(assembly=mm10)...
2025-07-02 12:30:31.664521 [info     ] DONE! Downloading dataset "4DNFI3RFZLZ5" took␣
↪48.569s.
2025-07-02 12:30:31.665930 [info     ] computing MD5 digest for file "/tmp/
↪4DNFI3RFZLZ5._3a7e1bs"...
2025-07-02 12:30:31.850671 [info     ] MD5 checksum match!
2025-07-02 12:30:31.851358 [info     ] successfully downloaded dataset "https://
↪zenodo.org/records/15301784/files/4DNFI3RFZLZ5.stripepy.mcool?download=1" to file
↪"4DNFI3RFZLZ5.mcool"
2025-07-02 12:30:31.851449 [info     ] file size: 83.86 MiB. Elapsed time: 49.160s
```

The `--assembly` option provides a mechanism to restrict downloads to datasets specifically associated with a given reference genome assembly. Currently, the supported options for this parameter are `hg38` (human) and `mm10` (mouse).

Additionally, for internal testing or specific development purposes, the `--include-private` flag allows the inclusion of datasets typically for internal testing.

4.3 Downloading test datasets

The *stripepy download* command also provides dedicated functionalities for acquiring specific test datasets used in unit and end-to-end testing.

The `--unit-test` option allows the download of datasets specifically required by the unit tests. When this option is invoked, the downloaded files are automatically stored within the `test/data/` directory, and any existing files at that location will be overwritten. It is important to note that when `--unit-test` is specified, all other command-line options are ignored, streamlining the test data acquisition process.

Similarly, the `--end2end-test` option allows the download of datasets necessary for end-to-end tests. Consistent with the unit test behavior, these files are also stored in the `test/data/` directory, existing files are overwritten, and all other command-line options are disregarded when this flag is active.

CHAPTER

FIVE

DETECT ARCHITECTURAL STRIPES

The *stripepy call* command serves as the main component within StripePy, and may take several minutes to complete when processing large files.

5.1 Positional arguments

The command mandates two positional arguments:

- `contact_map`, which specifies the path to the input Hi-C file (in .cool, .mcool, or .hic format).
- `resolution`, which represents the resolution (in base pairs) at which the analysis should be run.

For instance, to run the command on a file named `4DNFI9GMP2J8.mcool` at a resolution of 10,000 bp, you would use the following command:

```
user@dev:/tmp$ stripepy call 4DNFI9GMP2J8.mcool 10000
```

Running the above command produces a similar output, here truncated for the sake of brevity:

```
2025-04-15 08:13:24.639742 [info     ] running StripePy v1.0.0
2025-04-15 08:13:24.637358 [info     ] [main      ] CONFIG:
{
  "constrain_heights": false,
  "contact_map": "4DNFI9GMP2J8.mcool",
  "force": false,
  "genomic_belt": 5000000,
  "glob_pers_min": 0.04,
  "loc_pers_min": 0.33,
  "loc_trend_min": 0.25,
  "log_file": null,
  "max_width": 100000,
  "min_chrom_size": 2000000,
  "normalization": null,
  "nproc": 1,
  "output_file": "/tmp/4DNFI9GMP2J8.10000.hdf5",
  "plot_dir": null,
  "resolution": 10000,
  "roi": null,
  "verbosity": "info"
}
2025-04-15 08:13:24.637440 [info     ] [main      ] validating file "4DNFI9GMP2J8.
↪mcool" (10000bp)
2025-04-15 08:13:24.650236 [info     ] [main      ] file "4DNFI9GMP2J8.mcool"␣
↪successfully validated
2025-04-15 08:13:24.650445 [info     ] [IO        ] initializing result file "/tmp/
↪4DNFI9GMP2J8.10000.hdf5"
```

(continues on next page)

(continued from previous page)

```
2025-04-15 08:13:24.672613 [info     ] [chr1 ] [main      ] begin processing
2025-04-15 08:13:24.672729 [info     ] [chr1 ] [IO        ] fetching interactions␣
↪using normalization=NONE
2025-04-15 08:13:25.483686 [info     ] [chr1 ] [IO        ] fetched 6823257 pixels in␣
↪810.948ms
2025-04-15 08:13:25.483913 [info     ] [chr1 ] [step 1    ] data pre-processing
2025-04-15 08:13:25.483995 [info     ] [chr1 ] [step 1.1  ] focusing on a␣
↪neighborhood of the main diagonal
2025-04-15 08:13:25.535171 [info     ] [chr1 ] [step 1.1  ] removed 0.00% of the non-
↪zero entries (0/6823257)
2025-04-15 08:13:25.535378 [info     ] [chr1 ] [step 1.2  ] applying log-
↪transformation
2025-04-15 08:13:25.549232 [info     ] [chr1 ] [step 1.3  ] projecting interactions␣
↪onto [1, 0]
2025-04-15 08:13:25.553946 [info     ] [chr1 ] [step 1    ] preprocessing took 69.
↪937ms
2025-04-15 08:13:25.558918 [info     ] [chr1 ] [step 2    ] topological data analysis
2025-04-15 08:13:25.559059 [info     ] [chr1 ] [step 2.1.0] [LT] computing global 1D␣
↪pseudo-distribution
2025-04-15 08:13:25.583652 [info     ] [chr1 ] [step 2.2.0] [LT] detection of␣
↪persistent maxima and corresponding minima
2025-04-15 08:13:25.583770 [info     ] [chr1 ] [step 2.2.1] [LT] computing persistence
2025-04-15 08:13:25.625730 [info     ] [chr1 ] [step 2.2.2] [LT] filtering low␣
↪persistence values
2025-04-15 08:13:25.626417 [info     ] [chr1 ] [step 2.2.3] [LT] removing seeds␣
↪overlapping sparse regions
2025-04-15 08:13:25.686625 [info     ] [chr1 ] [step 2.2.3] [LT] number of seed sites␣
↪reduced from 1807 to 1748
2025-04-15 08:13:25.686795 [info     ] [chr1 ] [step 2.3.1] [LT] generating the list␣
↪of candidate stripes
2025-04-15 08:13:25.687662 [info     ] [chr1 ] [step 2.3.1] [LT] identified 1748␣
↪candidate stripes
2025-04-15 08:13:25.687864 [info     ] [chr1 ] [step 2.1.0] [UT] computing global 1D␣
↪pseudo-distribution
2025-04-15 08:13:25.713048 [info     ] [chr1 ] [step 2.2.0] [UT] detection of␣
↪persistent maxima and corresponding minima
2025-04-15 08:13:25.713154 [info     ] [chr1 ] [step 2.2.1] [UT] computing persistence
2025-04-15 08:13:25.753436 [info     ] [chr1 ] [step 2.2.2] [UT] filtering low␣
↪persistence values
2025-04-15 08:13:25.753932 [info     ] [chr1 ] [step 2.2.3] [UT] removing seeds␣
↪overlapping sparse regions
2025-04-15 08:13:25.813509 [info     ] [chr1 ] [step 2.2.3] [UT] number of seed sites␣
↪reduced from 1698 to 1647
2025-04-15 08:13:25.813687 [info     ] [chr1 ] [step 2.3.1] [UT] generating the list␣
↪of candidate stripes
...
2025-04-15 08:14:59.123408 [info     ] [IO        ] finalizing file "/tmp/
↪4DNFI9GMP2J8.10000.hdf5"
2025-04-15 08:14:59.127303 [info     ] [main      ] DONE!
2025-04-15 08:14:59.127399 [info     ] [main      ] processed 24 chromosomes in 1m:34.
↪490s
```

Upon successful completion, the above command will generate a single HDF5 file named `4DNFI9GMP2J8.10000.hdf5` in the current working directory. Note that, if the HDF5 already exists, it will be overwritten if and only if the `--force` flag is specified; otherwise, the command will fail to prevent accidental data loss.

5.2 Output and logging configuration

By default, the output HDF5 file is named after the input matrix file with the resolution appended, for example, `4DNFI9GMP2J8.10000.hdf5`. However, you have the flexibility to specify an alternative output path and filename for this HDF5 file using the `--output-file` option.

Furthermore, it is possible to save the complete log of a run to a file by specifying the path where to store the log file through the `--log-file` CLI option.

5.3 Stripe detection parameters

Beyond these arguments, *stripepy call* comes with a suite of optional parameters for fine-tuning the stripe detection process. For a full understanding of their meaning, the user is referred to our paper.

5.3.1 Step 1: pre-processing

You can apply a specific `--normalization` method when fetching the contact map data from the input file; by default, no normalization is applied. As found in our experiments, our algorithm performs optimally when no prior balancing is applied (see the Supplementary Information from our paper).

The `--genomic-belt` option defines a radial band around the main diagonal of the contact map, specified in base pairs, to which the stripe search is confined; its default value is 5 Mbp.

5.3.2 Step 2: line detection

The `--glob-pers-min` option sets a critical threshold value between 0 and 1 (defaulting to 0.04). This threshold is instrumental in filtering persistence maxima points for the global pseudo-distribution, which are crucial for identifying initial candidate stripe locations, frequently referred to as "seeds".

5.3.3 Step 3: shape analysis

The maximum permissible stripe width can be explicitly controlled using the `--max-width` option, which is specified in base pairs and defaults to 100,000 bp.

The height of a stripe is determined by studying a local pseudo-distribution. The algorithm applies topological persistence to the local pseudo-distribution to identify persistent peaks. The `--loc-pers-min` option acts as a threshold value between 0 and 1 (defaulting to 0.33) used to determine which peaks are persistent with respect to their topological persistence. The location of the furthest identified peak is then used as a boundary for the stripe. If no persistent maximum other than the global maximum is found, we threshold the local pseudo-distribution to a minimum value, specified via the option `--loc-trend-min`, which should be set to a value between 0 and 1 (defaulting to 0.25). A higher value for this parameter generally results in the detection of shorter stripes.

5.3.4 Step 4: signal analysis

The `--k-neighbour` option allows you to define 'k' for the k-neighbours: it represents the number of bins that are considered adjacent to the stripe boundaries on both sides, with a default value of 3. It is used to compute various signal descriptors, such as the relative change parameter.

5.4 Diagnostic plots generation

The command *stripepy call* can generate several diagnostic plots that can be of help to gain more insights into the decisions made by the tool.

To generate the diagnostic plots, pass `--roi=middle` and specify the path to a folder where to store the plots using `--plot-dir`. The `--roi` option requires you to specify a criterion (`start` or `middle`) to select a representative region from each chromosome for plot generation. Concurrently, the `--plot-dir` option designates the path to a directory where these output plots will be stored. It is important to note that the `--plot-dir` option is mandatory when `--roi` is specified and is otherwise ignored. If the specified directory does not exist at the time of execution, `stripepy` will automatically create it.

5.5 Performance options

When processing larger Hi-C matrix, StripePy can take advantage of multicore processors.

The maximum number of CPU cores use by StripePy can be changed through option `--nproc` (set to 1 core by default).

Whenever possible, we recommend using 4-8 CPU cores. Using more than 8 CPU cores is unlikely to result in significantly better computational performance (that is unless your Hi-C dataset is particularly dense).

CHAPTER

SIX

FETCH ARCHITECTURAL STRIPES

The `.hdf5` file produced by *stripepy call* contains various kinds of information, including stripe coordinates, various descriptive statistics, persistence vectors, and more.

While having access to all this information can be useful, usually we are mostly interested in the stripe coordinates, which can be fetched using *stripepy view*.

```
# Fetch the first 10 stripes in BEDPE format
user@dev:/tmp$ stripepy view 4DNFI9GMP2J8.10000.hdf5 | head

chr1  910000  960000  chr1    930000  3590000
chr1  1060000 1110000 chr1    1080000 3540000
chr1  1570000 1620000 chr1    1600000 2590000
chr1  1600000 1670000 chr1    880000  1620000
chr1  1670000 1700000 chr1    1680000 2610000
chr1  1730000 1780000 chr1    1750000 2570000
chr1  1780000 1840000 chr1    1780000 2580000
chr1  1890000 1940000 chr1    1920000 3540000
chr1  1940000 2020000 chr1    1960000 3590000
chr1  2020000 2060000 chr1    2020000 3550000

# Redirect stdout to a file
user@dev:/tmp$ stripepy view 4DNFI9GMP2J8.10000.hdf5 > stripes.bedpe

# Compress stripes on the fly before writing to a file
user@dev:/tmp$ stripepy view 4DNFI9GMP2J8.10000.hdf5 | gzip -9 > stripes.bedpe.gz
```

6.1 Output customization and filtering

When viewing the stripes, several optional parameters are available to customize the output.

The `--relative-change-threshold` option allows you to set a cutoff value (defaulting to 5.0) for filtering stripes based on their relative change. This relative change is calculated as the ratio between the average number of interactions found inside a stripe and the number of interactions in a neighborhood immediately outside of the stripe.

The `--coefficient-of-variation-threshold` option allows you to set a cutoff value for filtering stripes based on their coefficient of variation. The coefficient of variation is calculated as the ratio between the standard deviation and the mean of the interactions inside a stripe. Since the contact map values are normalized to the range [0, 1], this coefficient is always nonnegative. When not provided, no filtering is done. The options `--relative-change-threshold` and `--coefficient-of-variation-threshold` can be used in combination.

If you are interested in the biodescriptors associated with each individual stripe, you can pass `--with-header` and `--with-biodescriptors` when calling *stripepy view*.

This is the output generated by running *stripepy view* on the `.hdf5` generated using *stripepy call*. Files generated by older versions of StripePy may have different columns.

```
user@dev:/tmp$ stripepy view 4DNFI9GMP2J8.10000.hdf5 --with-biodescriptors --with-
↪header | head

chrom1        start1  end1    chrom2  start2  end2    top_persistence inner_mean
↪inner_std       outer_lsum      outer_lsize     outer_rsum      outer_rsize     min
↪    q1      q2      q3      max     outer_lmean     outer_rmean     outer_mean
↪rel_change      cfx_of_variation
chr1  910000  960000  chr1    930000  3590000 0.3984904019    0.2506571890861574
↪0.14123131812515843     144.79589039186396      801     192.25135582429806      801
↪    0.0     0.17139833204774585     0.22938081658911763     0.28656944403925566
↪0.9741568863537948      0.18076890186250183     0.24001417705904876     0.
↪2103915394607753      19.138435760573497      0.5634441152079366
chr1  1060000 1110000 chr1    1080000 3540000 0.0826359687    0.23019685453871336
↪0.14481608064533394     186.18030631678906      741     179.64345985134207      741
↪    0.0     0.1539575922232785      0.21018481227951455     0.2710230083036015
↪0.9903418421799679      0.2512554741117261      0.24243381896267485     0.
↪24684464653720048     6.744238626207448       0.6290966961105
chr1  1570000 1620000 chr1    1600000 2590000 0.04103011280000002     0.
↪33195798369580404     0.10697974882795283     99.02697827900961       300     85.
↪58022773213244       300     0.10509240613975727     0.2710230083036015      0.
↪3152772184192718      0.3662448898065007      0.9887477925105556      0.
↪3300899275966987      0.2852674257737748      0.3076786766852368      7.
↪891124361343245       0.3222689439094369
chr1  1600000 1670000 chr1    880000  1620000 0.10798038449999997     0.
↪34673478460468343     0.12547401272240433     79.95811315769556       225     63.
↪18147668278408       225     0.0     0.25904999836303577     0.33447322272887486
↪ 0.4155250840484962      0.9887477925105556      0.3553693918119803      0.
↪2808065630345959      0.3180879774232881      9.0059383612837 0.36187316154466237
chr1  1670000 1700000 chr1    1680000 2610000 0.08521339110000004     0.
↪30510000180174507     0.11602295320194354     84.13794539599031       282     71.
↪90225464650885       282     0.0     0.22938081658911763     0.304010183863723
↪ 0.37277167877770423     0.8753282776351561      0.29836150849641957     0.
↪2549725342074782      0.2766670213519489      10.276967710447305      0.
↪3802784415495862
chr1  1730000 1780000 chr1    1750000 2570000 0.09549401749999997     0.
↪34157106048803376     0.12939228310023276     66.96694495052422       249     77.
↪44100032822071       249     0.06630592590798857     0.25245019336736707     0.
↪32535592427102433     0.41427461878487365     0.9374989352738993      0.
↪26894355401816955     0.3110080334466695      0.28997579373241955     17.
↪792956471126924      0.37881512243852916
chr1  1780000 1840000 chr1    1780000 2580000 0.14961356020000005     0.
↪31446872398046843     0.14174768874612398     89.65252960337472       243     73.
↪53776985594494       243     0.0     0.2202635181312671      0.28656944403925566
↪ 0.3761154144433587      0.9150948504497306      0.3689404510426943      0.
↪3026245673084154      0.33578250917555486     6.347497148501883       0.
↪4507528982593763
chr1  1890000 1940000 chr1    1920000 3540000 0.13643510830000005     0.
↪27087952940479454     0.15589512088714813     98.34422915113818       489     137.
↪9512119037385       489     0.0     0.17139833204774585     0.2453610817780414
↪0.3592307814635864      0.989227567682685       0.20111294304936234     0.
↪2821088177990563      0.24161088042420928     12.113961477726793      0.
↪5755145884581886
chr1  1940000 2020000 chr1    1960000 3590000 0.05824488140000006     0.
↪267059000791004       0.1518633129658817      138.54936114286124      492     138.
↪81994263073136       492     0.0     0.17139833204774585     0.2453610817780414
↪0.34858989163711346     0.9751278353396942      0.28160439256679115     0.
↪2821543549405109      0.281879373753651       5.257700400455457       0.
```

(continues on next page)

(continued from previous page)

```
↪5686507944539472
```

If you are working in Python, you might want to take a look at the classes `Result` and `ResultFile`.

6.2 Coordinate transformation

The `--transform` option provides control over how stripe coordinates are presented in the output. By default, no transformation is applied. However, you can specify `transpose_to_ut` to transpose coordinates to the upper triangular part of the contact map, or `transpose_to_lt` to transpose them to the lower triangular part, which can be useful for specific downstream analyses or visualization preferences.

CHAPTER

SEVEN

GENERATING PLOTS

It is often a good idea to visually inspect at least some of the stripes to make sure that the used parameters are suitable for the dataset that was given to *stripepy call*. StripePy comes with a `plot` subcommand that can be used to generate various kinds of plots.

stripepy plot supports the following subcommands:

- `contact-map` (`cm`): plot stripes and other features over a region of the Hi-C matrix
- `pseudodistribution` (`pd`): plot the pseudo-distribution over a genomic interval
- `stripe-hist` (`hist`): generate and plot the histograms showing the distribution of the stripe heights and widths

All three subcommands support specifying a region of interest through the `--region` option. When the commands are run without specifying the region of interest, `stripepy plot cm` and `stripepy plot pd` will generate plots for a random 2.5 Mbp region, while `stripepy hist` will generate histograms using data from the entire genome.

7.1 Plots of contact maps

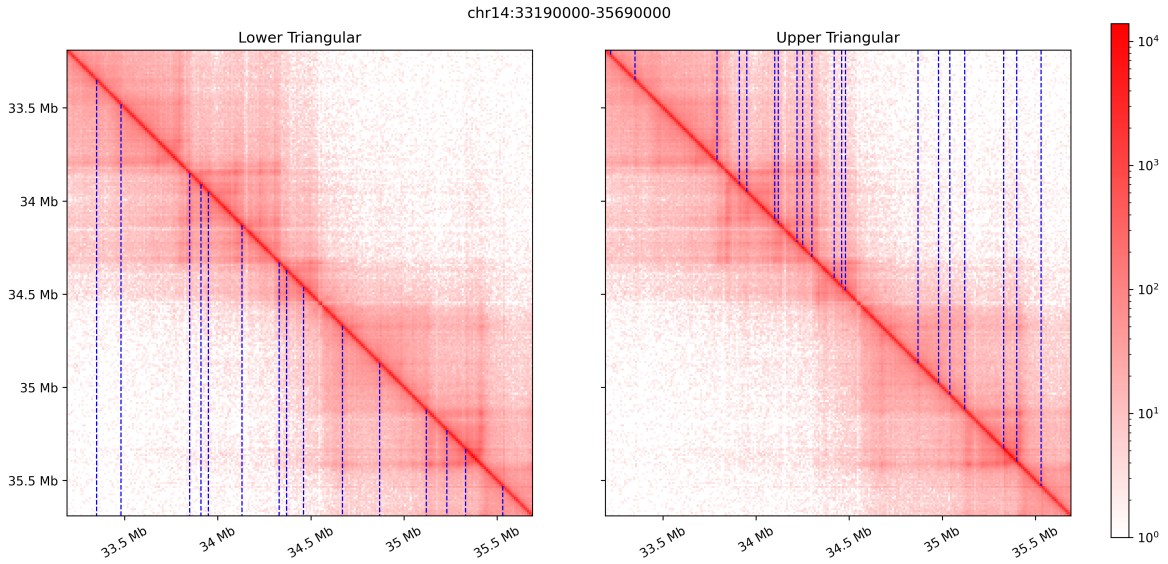
The `stripepy plot cm` command can be used to visualize Hi-C contact maps with different levels of annotation. `stripepy plot cm` takes as input a Hi-C matrix in `.cool`, `.mcool`, or `.hic` format, and optionally the `.hdf5` file generated by *stripepy call* (this parameter is mandatory when highlighting stripes or stripe seeds). We here provide a few examples.

The simplest plot displays the contact map at a given resolution:

```
# Plot the Hi-C matrix
user@dev:/tmp$ stripepy plot cm 4DNFI9GMP2J8.mcool 10000 /tmp/matrix.png
```

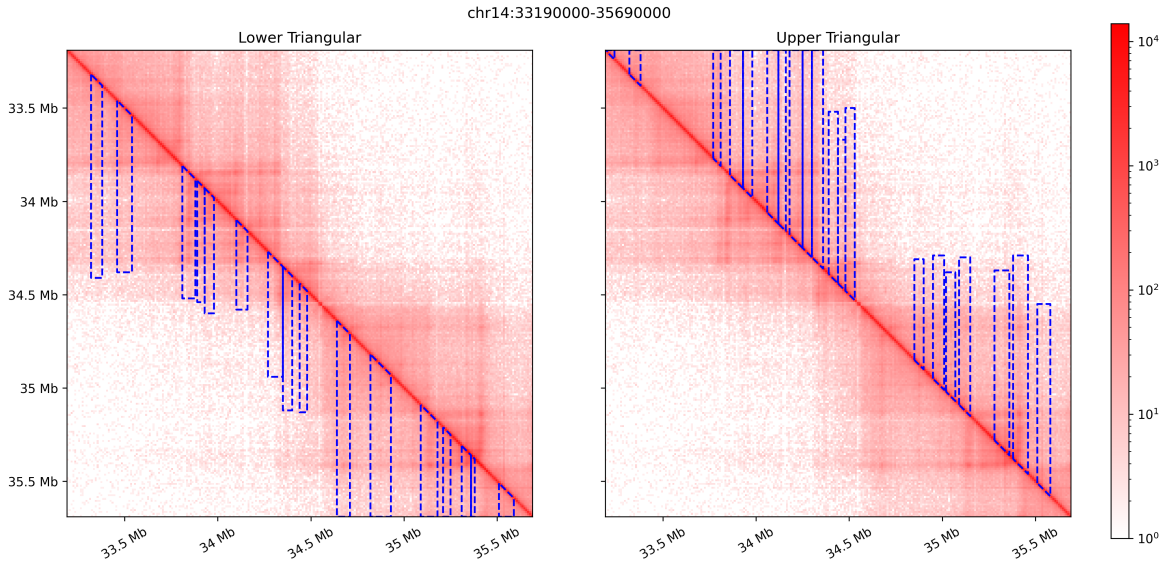
Stripe seeds can be visualized with the option `--highlight-seeds`. Note that the HDF5 output is required:

```
# Plot the Hi-C matrix highlighting the stripe seeds
user@dev:/tmp$ stripepy plot cm 4DNFI9GMP2J8.mcool 10000 /tmp/matrix_with_seeds.png --
↪stripepy-hdf5 4DNFI9GMP2J8.10000.hdf5 --highlight-seeds
```



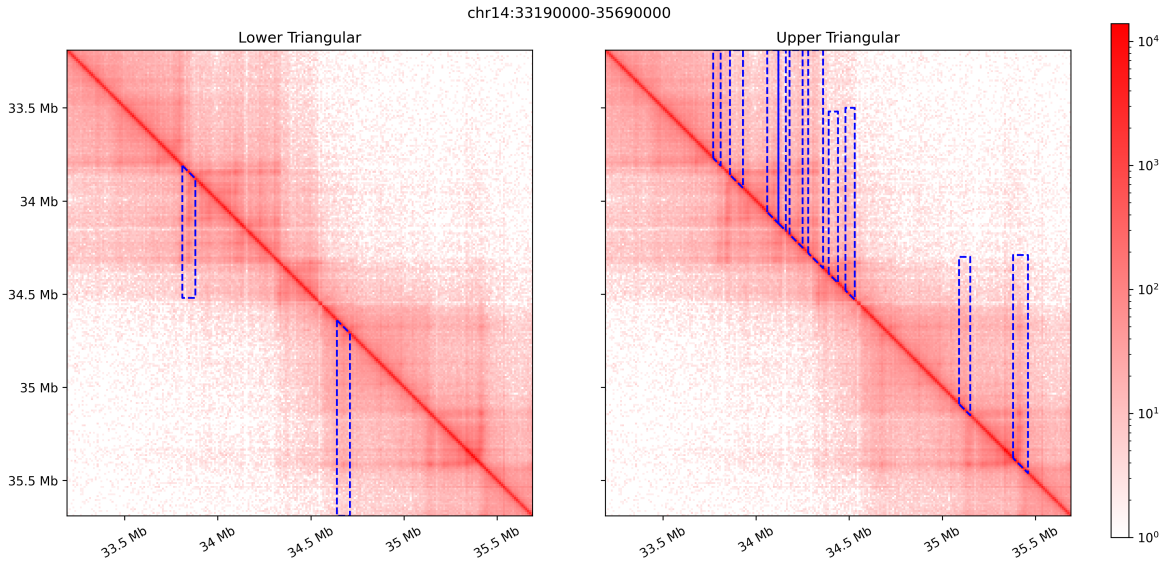
To visualize the detected architectural stripes, use the `--highlight-stripes` flag. It is possible to visualize the stripes **without** any thresholding

```
# Plot the Hi-C matrix highlighting the architectural stripes
user@dev:/tmp$ stripepy plot cm 4DNFI9GMP2J8.mcool 10000 /tmp/matrix_with_stripes.png␣
↪--stripepy-hdf5 4DNFI9GMP2J8.10000.hdf5 --highlight-stripes
```



or **with** thresholding (using either `--coefficient-of-variation-threshold`, `--relative-change-threshold`, or both options):

```
# Plot the Hi-C matrix highlighting the architectural stripes -- thresholding both␣
↪relative change and coefficient of variation
user@dev:/tmp$ stripepy plot cm 4DNFI9GMP2J8.mcool 10000 /tmp/matrix_with_stripes.png␣
↪--stripepy-hdf5 4DNFI9GMP2J8.10000.hdf5 --highlight-stripes --coefficient-of-
↪variation-threshold 1 --relative-change-threshold 5
```

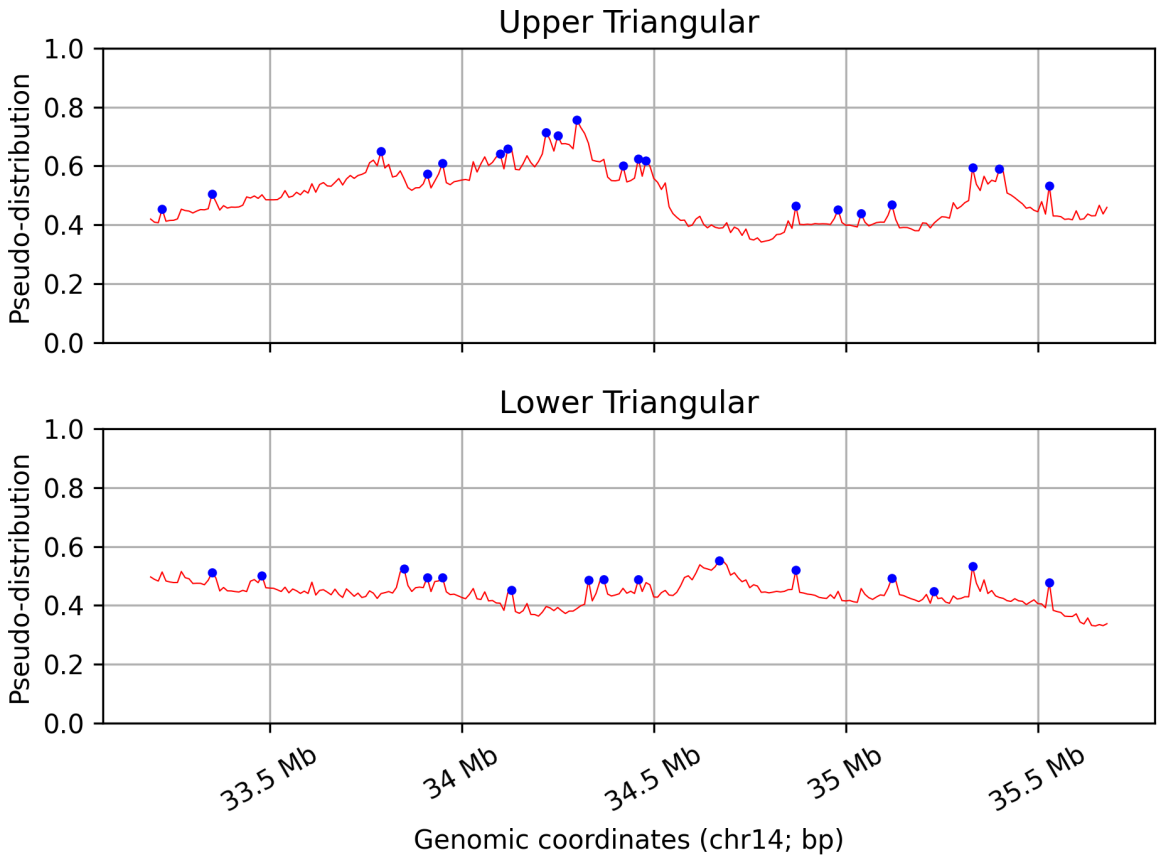


7.2 Plots of pseudo-distributions

`stripepy plot pd` (and `stripepy plot hist`) does not require the Hi-C matrix file, and require the `.hdf5` file generated by *stripepy call* instead.

Example usage:

```
# Plot the pseudo-distribution
user@dev:/tmp$ stripepy plot pd 4DNFI9GMP2J8.10000.hdf5 /tmp/pseudodistribution.png
```

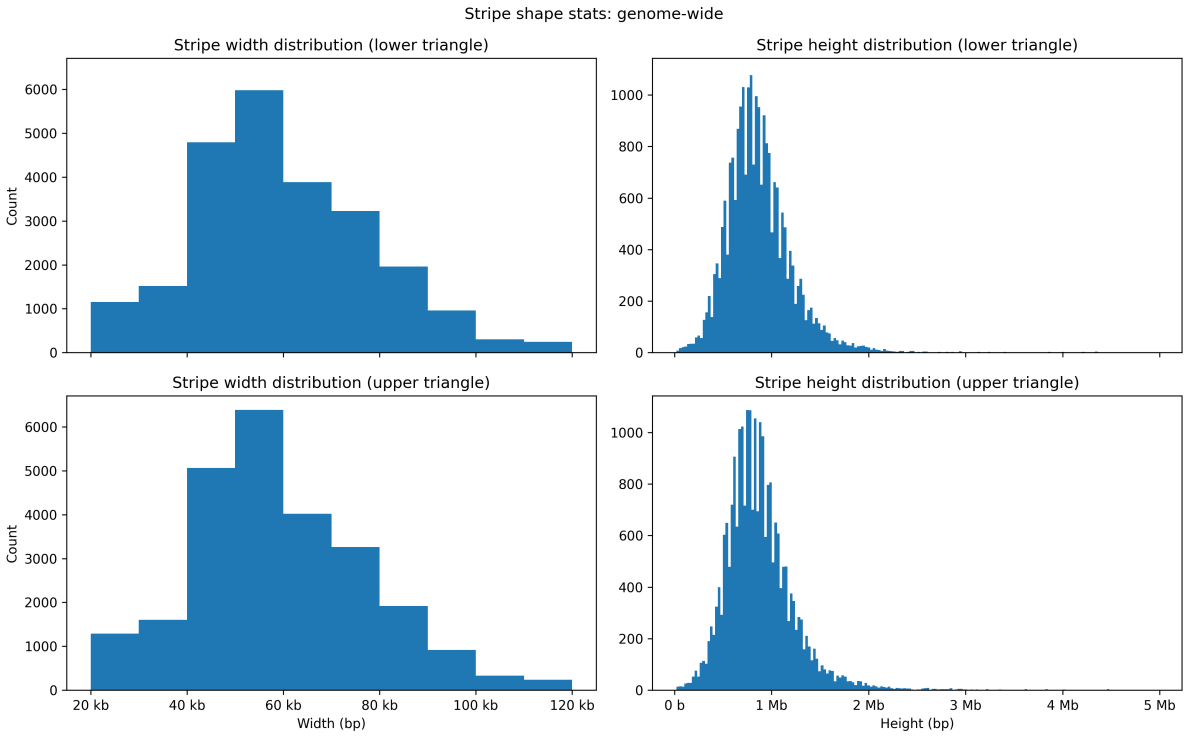


7.3 Plots of histograms

`stripepy hist` uses the `.hdf5` file generated by *stripepy call* to create histograms of .

Example usage:

```
# Plot the histograms using genome-wide data
user@dev:/tmp$ stripepy plot hist 4DNFI9GMP2J8.10000.hdf5 /tmp/stripe_hist_gw.png
```



7.3.1 Visualize architectural stripes in HiGlass

We provide a Jupyter notebook `visualize_stripes_with_highlass.ipynb` to facilitate this visual inspection with HiGlass. The notebook expects the input file to be in `.mcool` format.

More info available at Visualize architectural stripes with HiGlass.

CHAPTER

EIGHT

CLI REFERENCE

For an up-to-date list of subcommands and CLI options refer to `stripepy --help`.

8.1 Subcommands

```
usage: stripepy {call,download,plot,view} ...
stripepy is designed to recognize linear patterns in contact maps (.hic, .mcool, .
↪cool) through the geometric reasoning, including topological persistence and quasi-
↪interpolation.
options:
  -h, --help            show this help message and exit
  --license             Print StripePy's license and return.
  --cite                Print StripePy's reference and return.
  -v, --version         show program's version number and exit
subcommands:
  {call,download,plot,view}
                        List of available subcommands:
    call                stripepy works in four consecutive steps:
                        • Step 1: Pre-processing
                        • Step 2: Recognition of loci of interest (also called 'seeds
↪')
                        • Step 3: Shape analysis (i.e., width and height estimation)
                        • Step 4: Signal analysis
    download            Helper command to simplify downloading datasets that can be␣
↪used to test StripePy.
    plot                Generate various static plots useful to visually inspect the␣
↪output produced by stripepy call.
    view                Fetch stripes from the HDF5 file produced by stripepy call.
```

8.2 stripepy call

```
usage: stripepy call [-h] [-n NORMALIZATION] [-b GENOMIC_BELT]
                     [--roi {middle,start}] [-o OUTPUT_FILE]
                     [--log-file LOG_FILE] [--plot-dir PLOT_DIR]
                     [--max-width MAX_WIDTH] [--glob-pers-min GLOB_PERS_MIN]
                     [-k K] [--loc-pers-min LOC_PERS_MIN]
                     [--loc-trend-min LOC_TREND_MIN] [-f]
                     [--verbosity {debug,info,warning,error,critical}]
                     [-p NPROC] [--low-memory]
                     [--min-chrom-size MIN_CHROM_SIZE]
                     contact-map resolution
positional arguments:
  contact-map           Path to a .cool, .mcool, or .hic file for input.
```

(continues on next page)

(continued from previous page)

```
  resolution            Resolution (in bp).
options:
  -h, --help            show this help message and exit
  -n, --normalization NORMALIZATION
                        Normalization to fetch (default: None).
  -b, --genomic-belt GENOMIC_BELT
                        Radius of the band, centred around the diagonal, where the␣
↪search is restricted to (in bp, default: 5000000).
  --roi {middle,start}  Criterion used to select a region from each chromosome used␣
↪to generate diagnostic plots (default: None).
                        Requires --plot-dir.
  -o, --output-file OUTPUT_FILE
                        Path where to store the output HDF5 file.
                        When not specified, the output file will be saved in the␣
↪current working directory with a named based on the name of input matrix file.
  --log-file LOG_FILE   Path where to store the log file.
  --plot-dir PLOT_DIR   Path where to store the output plots.
                        Required when --roi is specified and ignored otherwise.
                        If the specified folder does not already exist, it will be␣
↪created.
  --max-width MAX_WIDTH
                        Maximum stripe width, in bp (default: 100000).
  --glob-pers-min GLOB_PERS_MIN
                        Threshold value between 0 and 1 to filter persistence maxima␣
↪points and identify loci of interest, aka seeds (default: 0.04).
  -k, --k-neighbour K   k for the k-neighbour, i.e., number of bins adjacent to the␣
↪stripe boundaries on both sides (default: 3).
  --loc-pers-min LOC_PERS_MIN
                        Threshold value between 0 and 1 to find peaks in signal in a␣
↪horizontal domain while estimating the height of a stripe (default: 0.33).
  --loc-trend-min LOC_TREND_MIN
                        Threshold value between 0 and 1 to estimate the height of a␣
↪stripe (default: 0.25); the higher this value, the shorter the stripe; it is used␣
↪to avoid overly long stripes when no persistent maximum besides the global one is␣
↪found.
  -f, --force           Overwrite existing file(s) (default: False).
  --verbosity {debug,info,warning,error,critical}
                        Set verbosity of output to the console (default: info).
  -p, --nproc NPROC     Maximum number of parallel processes to use (default: 1).
  --low-memory          Minimize memory usage when fetching interactions from files␣
↪(default: False).
  --min-chrom-size MIN_CHROM_SIZE
                        Minimum size, in bp, for a chromosome to be analysed␣
↪(default: 2000000).
```

8.3 stripepy download

```
usage: stripepy download [-h] [--assembly {hg38,mm10} | --name NAME |
                          --list-only] [--unit-test | --end2end-test]
                          [--include-private] [--max-size MAX_SIZE]
                          [-o OUTPUT_PATH] [-f]
                          [--verbosity {debug,info,warning,error,critical}]
options:
  -h, --help            show this help message and exit
  --assembly {hg38,mm10}
```

(continues on next page)

(continued from previous page)

```
                        Restrict downloads to the given reference genome assembly.
  --name NAME           Name of the dataset to be downloaded.
                        When not provided, randomly select and download a dataset␣
↪based on the provided CLI options (if any).
  --list-only           Print the list of available datasets and return (default:␣
↪False).
  --unit-test           Download the test datasets required by the unit tests.
                        Files will be stored under folder test/data/
                        When specified, all other options are ignored.
                        Existing files will be overwritten.
  --end2end-test        Download the test datasets required by the end2end tests.
                        Files will be stored under folder test/data/
                        When specified, all other options are ignored.
                        Existing files will be overwritten.
  --include-private     Include datasets used for internal testing (default: False).
  --max-size MAX_SIZE   Upper bound for the size of the files to be considered when --
↪name is not provided (default: 512.0).
  -o, --output OUTPUT_PATH
                        Path where to store the downloaded file (default: None).
  -f, --force           Overwrite existing file(s) (default: False).
  --verbosity {debug,info,warning,error,critical}
                        Set verbosity of output to the console (default: info).
```

8.4 stripepy plot

```
usage: stripepy plot [-h]
                     {contact-map,cm,pseudodistribution,pd,stripe-hist,hist} ...
options:
  -h, --help            show this help message and exit
plot_subcommands:
  {contact-map,cm,pseudodistribution,pd,stripe-hist,hist}
                        List of available subcommands:
    contact-map (cm)    Plot stripes and other features over the Hi-C matrix.
    pseudodistribution (pd)
                        Plot the pseudo-distribution over the given region of␣
↪interest.
    stripe-hist (hist)  Generate and plot the histograms showing the distribution of␣
↪the stripe heights and widths.
```

8.5 stripepy plot contact-map

```
usage: stripepy plot contact-map [-h] [--stripepy-hdf5 STRIPEPY_HDF5]
                                 [--relative-change-threshold RELATIVE_CHANGE_
↪THRESHOLD]
                                 [--coefficient-of-variation-threshold COEFFICIENT_OF_
↪VARIATION_THRESHOLD]
                                 [--highlight-seeds | --highlight-stripes]
                                 [--ignore-stripe-heights] [--cmap CMAP]
                                 [--linear-scale | --log-scale]
                                 [--region REGION] [--dpi DPI] [--seed SEED]
                                 [-n NORMALIZATION] [-f]
                                 [--verbosity {debug,info,warning,error,critical}]
                                 contact-map resolution output-name
positional arguments:
```

(continues on next page)

(continued from previous page)

```
  contact-map           Path to the .cool, .mcool, or .hic file used to call stripes.
  resolution            Resolution (in bp).
  output-name           Path where to store the generated plot.
options:
  -h, --help            show this help message and exit
  --stripepy-hdf5 STRIPEPY_HDF5
                        Path to the .hdf5 generated by stripepy call.
                        Required when highlighting stripes or seeds.
  --relative-change-threshold RELATIVE_CHANGE_THRESHOLD
                        Cutoff for the relative change (default: None).
                        Only used when highlighting architectural stripes.
                        The relative change is computed as the ratio between the␣
↪average number of interactions found inside a stripe and the number of interactions␣
↪in a neighborhood outside of the stripe.
  --coefficient-of-variation-threshold COEFFICIENT_OF_VARIATION_THRESHOLD
                        Cutoff for the coefficient of variation (default: None).
                        Only used when highlighting architectural stripes.
                        The coefficient of variation is computed as the ratio between␣
↪the standard deviation and the mean
                        of the values inside a stripe. In our case, it is always␣
↪nonnegative because of the preprocessing step.
  --highlight-seeds     Highlight the stripe seeds (default: False).
  --highlight-stripes   Highlight the architectural stripes (default: False).
  --ignore-stripe-heights
                        Ignore the stripes height (default: False).
                        Has no effect when --highlight-stripes is not specified.
  --cmap CMAP           Color map used to plot Hi-C interactions (default: fruit_
↪punch).
                        Can be any of the color maps supported by matplotlib as well␣
↪as: fall, fruit_punch, blues, acidblues, and nmeth.
  --linear-scale        Plot interactions in linear scale (default: False).
  --log-scale           Plot interactions in log scale (default: True).
  --region REGION       Genomic region to be plotted (UCSC format). When not␣
↪specified, a random 2.5Mb region is plotted.
  --dpi DPI             DPI of the generated plot (default: 300; ignored when the␣
↪output format is a vector graphic).
  --seed SEED           Seed for random number generation (default:␣
↪7606490399616306585).
  -n, --normalization NORMALIZATION
                        Normalization to fetch (default: None).
  -f, --force           Overwrite existing file(s) (default: False).
  --verbosity {debug,info,warning,error,critical}
                        Set verbosity of output to the console (default: info).
```

8.6 stripepy plot pseudodistribution

```
usage: stripepy plot pseudodistribution [-h] [--region REGION] [--dpi DPI]
                                        [--seed SEED] [-n NORMALIZATION] [-f]
                                        [--verbosity {debug,info,warning,error,
↪critical}]
                                        stripepy-hdf5 output-name

positional arguments:
  stripepy-hdf5         Path to the .hdf5 generated by stripepy call.
  output-name           Path where to store the generated plot.
options:
```

(continues on next page)

(continued from previous page)

```
  -h, --help            show this help message and exit
  --region REGION       Genomic region to be plotted (UCSC format). When not␣
↪specified, a random 2.5Mb region is plotted.
  --dpi DPI             DPI of the generated plot (default: 300; ignored when the␣
↪output format is a vector graphic).
  --seed SEED           Seed for random number generation (default:␣
↪76064903996163065 85).
  -n, --normalization NORMALIZATION
                        Normalization to fetch (default: None).
  -f, --force           Overwrite existing file(s) (default: False).
  --verbosity {debug,info,warning,error,critical}
                        Set verbosity of output to the console (default: info).
```

8.7 stripepy plot stripe-hist

```
usage: stripepy plot stripe-hist [-h] [--region REGION] [--dpi DPI]
                                 [--seed SEED] [-n NORMALIZATION] [-f]
                                 [--verbosity {debug,info,warning,error,critical}]
                                 stripepy-hdf5 output-name
positional arguments:
  stripepy-hdf5         Path to the .hdf5 generated by stripepy call.
  output-name           Path where to store the generated plot.
options:
  -h, --help            show this help message and exit
  --region REGION       Genomic region to be plotted (UCSC format). When not␣
↪specified, data for the entire genome is plotted.
  --dpi DPI             DPI of the generated plot (default: 300; ignored when the␣
↪output format is a vector graphic).
  --seed SEED           Seed for random number generation (default:␣
↪76064903996163065 85).
  -n, --normalization NORMALIZATION
                        Normalization to fetch (default: None).
  -f, --force           Overwrite existing file(s) (default: False).
  --verbosity {debug,info,warning,error,critical}
                        Set verbosity of output to the console (default: info).
```

8.8 stripepy view

```
usage: stripepy view [-h]
                     [--relative-change-threshold RELATIVE_CHANGE_THRESHOLD]
                     [--coefficient-of-variation-threshold COEFFICIENT_OF_VARIATION_
↪THRESHOLD]
                     [--with-biodescriptors] [--with-header]
                     [--transform {None,transpose_to_lt,transpose_to_ut}]
                     [--verbosity {debug,info,warning,error,critical}]
                     h5-file
positional arguments:
  h5-file               Path to the HDF5 file generated by stripepy call.
options:
  -h, --help            show this help message and exit
  --relative-change-threshold RELATIVE_CHANGE_THRESHOLD
                        Cutoff for the relative change (default: 5.0).
                        The relative change is computed as the ratio between the␣
↪average number of interactions
```

(continues on next page)

(continued from previous page)

```
                        found inside a stripe and the number of interactions in a␣
↪neighborhood outside of the stripe.
  --coefficient-of-variation-threshold COEFFICIENT_OF_VARIATION_THRESHOLD
                        Cutoff for the coefficient of variation (default: None).
                        The coefficient of variation is computed as the ratio between␣
↪the standard deviation and the mean
                        of the values inside a stripe. In our case, it is always␣
↪nonnegative because of the preprocessing step.
  --with-biodescriptors
                        Include the stripe biodescriptors in the output.
  --with-header         Include column names in the output.
  --transform {None,transpose_to_lt,transpose_to_ut}
                        Control if and how stripe coordinates should be transformed␣
↪(default: None).
  --verbosity {debug,info,warning,error,critical}
                        Set verbosity of output to the console (default: info).
```

CHAPTER

NINE

PYTHON API REFERENCE

stripepy.data_structures.**SparseMatrix**

alias of csr_matrix | csc_matrix

class stripepy.data_structures.**Stripe**(
seed: int,
top_pers: float | None,
horizontal_bounds: Tuple[int, int] | None = None,
vertical_bounds: Tuple[int, int] | None = None,
where: str | None = None,
)

A class used to represent architectural stripes. This class takes care of validating stripe coordinates and computing several descriptive statistics.

This is how this class should be used:

- Initialize the class by providing at least the seed position
- At a later time, set the vertical and horizontal boundaries by calling *set_horizontal_bounds* and *set_vertical_bounds*
- Finally, call *compute_biodescriptors* to compute and store the descriptive statistics

The stripe properties and statistics can now be accessed through the attributes listed below.

Attributes representing the descriptive statistics return negative values to signal that it was not possible to compute the statistics for the current Stripe instance.

__init__(
seed: int,
top_pers: float | None,
horizontal_bounds: Tuple[int, int] | None = None,
vertical_bounds: Tuple[int, int] | None = None,
where: str | None = None,
)

Parameters

- **seed** – the stripe seed position
- **top_pers** – the topological persistence of the seed
- **horizontal_bounds** – the horizontal bounds of the stripe
- **vertical_bounds** – the_vertical bounds of the stripe
- **where** – the location of the stripe: should be “upper_triangular” or “lower_triangular”. When provided, this is used validate the coordinates set when calling *set_horizontal_bounds()* and *set_vertical_bounds()*.

property seed: int

The stripe seed

property top_persistence: float | None

The topological persistence

property lower_triangular: bool

True when the stripe extends in the lower-triangular portion of the matrix

property upper_triangular: bool

True when the stripe extends in the upper-triangular portion of the matrix

property triangular_undetermined: bool

True when the stripe has height 0

property left_bound: int

The left bound of the stripe

property right_bound: int

The right bound of the stripe

property top_bound: int

The top bound of the stripe

property bottom_bound: int

The bottom bound of the stripe

property inner_mean: float

The average number of interactions within the stripe

property inner_std: float

The standard deviation of the number of interactions within the stripe

property five_number: ndarray[tuple[Any, ...], dtype[float]]

A vector of five numbers corresponding to the 0, 25, 50, 75, and 100 percentiles of the number of within-stripe interactions

property outer_lsum: float

The sum of interactions in the band to the left of the stripe

property outer_rsum: float

The sum of interactions in the band to the right of the stripe

property outer_lsize: float

The number of entries in the band to the left of the stripe

property outer_rsize: float

The number of entries in the band to the right of the stripe

property outer_lmean: float

The average number of interactions in the band to the left of the stripe

property outer_rmean: float

The average number of interactions in the band to the right of the stripe

property outer_mean: float

The average number of interactions in the bands to the left and right of the stripe

property rel_change: float

The ratio of the average number of interactions within the stripe and in the neighborhood outside of the stripe

set_horizontal_bounds(*left_bound: int*, *right_bound: int*)

Set the horizontal bounds for the stripe. This function raises an exception when the coordinates have already been set or when the given coordinates are incompatible with the seed position.

Parameters

- **left_bound**
- **right_bound**

set_vertical_bounds(*top_bound: int*, *bottom_bound: int*)

Set the vertical bounds for the stripe. This function raises an exception when the coordinates have already been set or when the given coordinates are incompatible with the seed position and/or the where location.

Parameters

- **top_bound**
- **bottom_bound**

compute_biodescriptors(
matrix: csr_matrix | csc_matrix,
window: int = 3,
)

Use the sparse matrix to compute various descriptive statistics. Statistics are stored in the current Stripe instance. This function raises an exception when it is called before the stripe bounds have been set.

Parameters

- **matrix** – the sparse matrix from which the stripe originated
- **window** – window size used to compute statistics to the left and right of the stripe

class stripepy.data_structures.**ResultFile**(
path: Path,
mode: str = 'r',
_create_key: object | None = None,
)

A class used to read and write StripePy results to a HDF5 file.

There are 3 main use cases:

- Open the file in read mode:

```
with ResultFile("results.hdf5") as h5:
...
```

- Open file in write mode:
  - If all data will be written to the file before the file is closed:

```
with ResultFile.create("results.hdf5", mode="w", ...) as h5:
    h5.write_descriptors(res1)
    h5.write_descriptors(res2)
    ...
```

  - If the data will be added progressively:

```
with ResultFile.create("results.hdf5", mode="a", ...) as h5:
    h5.write_descriptors(res1)  # not mandatory, it is also possible to␣
↪create the
                                # file and close it immediately
...
with ResultFile.append("results.hdf5") as h5:
    h5.write_descriptors(res2)
    h5.write_descriptors(res3)
...
```

(continues on next page)

(continued from previous page)

```
with ResultFile.append("results.hdf5") as h5:
    h5.write_descriptors(res4)
    h5.finalize()  # IMPORTANT!
                   # Without the above line you'll get an error when␣
↪trying to open
                   # the file in read mode
```

When opening or creating a *ResultFile* write or append mode, a context manager (e.g. with:) must be used

__init__(
path: Path,
mode: str = 'r',
_create_key: object | None = None,
)

static create(
path: Path,
mode: str,
chroms: Dict[str, int],
resolution: int,
normalization: str | None = None,
assembly: str = 'unknown',
metadata: Dict[str, Any] | None = None,
compression_lvl: int = 9,
)

Create a *ResultFile* using the provided information.

static create_from_file(
path: Path,
mode: str,
matrix_file: File,
normalization: str | None = None,
metadata: Dict[str, Any] | None = None,
compression_lvl: int = 9,
)

Create a *ResultFile* using information from the given matrix file.

static append(*path: Path*)

Append to an existing *ResultFile*.

IMPORTANT: the file must have been created with *create* or *create_from_file* with `mode="a"`

property assembly: `str`

The name of the reference genome assembly used to generate the file

property chromosomes: `Dict[str, int]`

The chromosomes associated with the opened file

property creation_date: `datetime`

The file creation date

property format: `str`

The file format string

property format_url: `str`

The URL where the file format is documented

property format_version: int

The format version of the file currently opened

property generated_by: str

The name of the tool used to generate the opened file

property metadata: Dict[str, Any]

The metadata associated with the file

property normalization: str | None

The name of the normalization used to generate the data stored in the given file

property path: Path

The path to the opened file

property resolution: int

The resolution of the Hi-C matrix used to generate the file

finalize()

Finalize a file opened in append mode

__getitem__(*chrom: str*) → *Result*

get_min_persistence(*chrom: str*) → float

Get the minimum persistence associated with the given chromosome.

Parameters
chrom – chromosome name

Returns
the minimum persistence

get(
chrom: str | None,
field: str,
location: str,
) → DataFrame

Get the data associated with the given chromosome, field, and location.

Parameters

- **chrom** – chromosome name. when not provided, return data for the entire genome.
- **field** – name of the field to be fetched. Supported names:
  - pseudodistribution
  - all_minimum_points
  - persistence_of_all_minimum_points
  - all_maximum_points
  - persistence_of_all_maximum_points
  - geo_descriptors
  - bio_descriptors
  - stripes
- **location** – location of the attribute to be registered. Should be “LT” or “UT”

Returns
the data associated with the given chromosome, field, and location

write_descriptors(*result:* Result)

Read the descriptors from the given Result object and write them to the opened file.

Parameters
result – results to be added to the opened file

class stripepy.data_structures.**Result**(*chrom_name: str*, *chrom_size: int*)

A class used to represent the results generated by stripepy call.

__init__(*chrom_name: str*, *chrom_size: int*)

Parameters

- **chrom_name** (*str*) – chromosome name
- **chrom_size** (*int*) – chromosome size

property chrom: Tuple[str, int]

The name and length of the chromosomes to which the *Result* instance belongs to

property empty: bool

Check whether any stripe has been registered with the *Result* instance

property min_persistence: float

The minimum persistence used during computation

property roi: Dict[str, List[int]] | None

The region of interest associated with the *Result* instance

get(
name: str,
location: str,
) → List[*Stripe*] | ndarray[int] | ndarray[float]

Get the value associated with the given attribute name and location.

Parameters

- **name** – name of the attribute to be fetched
- **location** – location of the attribute to be fetched. Should be "LT" or "UT"

Returns
the value associated with the given name and location.

get_stripes_descriptor(
descriptor: str,
location: str,
) → ndarray[int] | ndarray[float]

Get the stripe descriptor for the given location.

Parameters

- **descriptor** – name of the descriptor to be fetched
- **location** – location of the attribute to be fetched. Should be "LT" or "UT"

Returns
the value associated with the given descriptor and location.

get_stripe_bio_descriptors(*location: str*) → DataFrame

Fetch all biological descriptors at once.

Parameters
location – location of the attribute to be fetched. Should be "LT" or "UT"

Returns
the table with the biological descriptors associated with the *Result* instance

`get_stripe_geo_descriptors`(*location: str*) → DataFrame

Fetch all geometric descriptors at once.

Parameters
`location` – location of the attribute to be fetched. Should be “LT” or “UT”

Returns
the table with the geometric descriptors associated with the `Result` instance

CHAPTER

TEN

TELEMETRY

Starting with version v1.1.1 of StripePy, we introduced support for telemetry collection.

This page outlines what information we are collecting and why. Furthermore, we provide instructions on how telemetry collection can be disabled.

10.1 What information is being collected

`stripepy` is instrumented to collect general information about `stripepy` itself and the system where it is being run.

We do not collect any sensitive information that could be used to identify our users, the machine where `stripepy` is being run, or the datasets processed by `stripepy`.

This is the data we are collecting:

- Information on how `stripepy` was installed (i.e., the package version and third-party dependency versions).
- Information on the system where `stripepy` is being run (i.e., operating system, processor architecture, and Python version).
- How `stripepy` is being invoked (i.e., the subcommand, input file format and parameters).
- Information about `stripepy` execution (i.e., when it was launched, how long the command took to finish, and whether the command terminated with an error).

The following table shows an example of the telemetry collected when running `stripepy call ENCFF993FGR.hic 10000 -p 8`:

Table 1: Telemetry information collected when running `stripepy call`

Field	Value
dependency.h5py.version	3.14.0
dependency.hictkpy.version	1.3.0
dependency.numpy.version	2.3.1
dependency.packaging.version	25.0
dependency.pandas.version	2.3.1
dependency.scipy.version	1.16.0
dependency.structlog.version	25.4.0
duration_ms	103074.812875
host.arch	x86_64
library.name	stripepy
meta.signal_type	trace
name	call
os.type	linux
os.version	6.11.0-1015-azure
params.constrain_heights	false
params.contact_map_format	mcool

continues on next page

Table 1 – continued from previous page

Field	Value
params.contact_map_raw_interactions	true
params.contact_map_resolution	20000
params.genomic_belt	5000000
params.glob_pers_min	0.04
params.k	3
params.loc_pers_min	0.33
params.loc_trend_min	0.25
params.max_width	100000
params.nproc	4
process.runtime.description	GCC 12.2.0
process.runtime.name	CPython
process.runtime.version	3.13.5
Sample Rate	1
service.name	stripepy
service.version	1.1.1.dev73+g2e13fec
span.kind	internal
span.num_events	0
span.num_links	0
status_code	1
telemetry.sdk.language	python
telemetry.sdk.name	opentelemetry
telemetry.sdk.version	1.34.1
trace.span_id	4b6f5534c8aec420
trace.trace_id	30b4804d781557f552d15dec8270fca8
type	internal

10.2 Why are we collecting this information?

There are two main motivations behind our decision to start collecting telemetry data:

1. To get an idea of how big our user base is: this will help us, among other things, to secure funding to maintain `stripepy` in the future.
2. To better understand which of the functionalities offered by `stripepy` are most used by our users: we intend to use this information to help us decide which features we should focus our development efforts on.

10.3 How is telemetry information processed and stored

Telemetry is sent to an OpenTelemetry collector running on a virtual server hosted on the Norwegian Research and Education Cloud (NREC).

The virtual server and collector are managed by us, and traffic between `stripepy` and the collector is encrypted.

The collector processes incoming data continuously and forwards it to a dashboard for data analytics and a backup solution (both services are hosted in Europe). Communication between the collector, dashboard, and backup site is also encrypted. Data stored by the dashboard and backup site is encrypted at rest.

The analytics dashboard keeps telemetry data for up to 60 days, while the backup site is currently set up to store telemetry data indefinitely (although this may change in the future).

10.4 How to disable telemetry collection

To disable telemetry collection, simply define the `STRIPEPY_NO_TELEMETRY` environment variable before launching `stripepy` (e.g., `STRIPEPY_NO_TELEMETRY=1 stripepy download`)

10.5 Where can I find the code used for telemetry collection?

All code concerning telemetry collection is defined in file src/stripepy/cli/telemetry.py.

PYTHON MODULE INDEX

S

INDEX

Symbols

A

B

C

E

F

G

I

L

M

N

O

P

R

S

T

U

W