PhyloHiC Documentation

Sylvain PULICANI

Sep 06, 2019

Contents

PhyloHiC is a set of scripts that allows the computation of distances between genomes based on *shared genomic regions* and *genomic contact data*. Currently, we use orthologous genes as shared regions and Hi-C data as contacts information. Thus, you must already have these data.

CHAPTER 1

License

The source code is available here and is licensed under the CeCILL V2.1 since it was written when I was working in a CNRS laboratory. The CeCILL license is close to the GNU GPL and compatible with it. The full text for the license is available in the LICENSE file in the repository.

CHAPTER 2

Citing

PhyloHiC has been written during my PhD thesis work, and is described in the third chapter of my thesis (in French). Moreover, the results presented there come from the pipeline documented here. Thus, if you use this pipeline, please cite my thesis:

```
Sylvain Pulicani.
Lien entre les réarrangements chromosomiques et la structure de la chromatine chez la␣
↪Drosophile.
Autre [cs.OH]. Université Montpellier, 2018. Français.
NNT: 2018MONTS105
HAL: tel-02161932
```

Please note that a scientific publication is coming.

This documentation is divided in three sections:

2.1 Introduction and General Process

2.1.1 Main Idea

We want to compare the shape of DNA between homologous genomic regions.

The shape of DNA cannot be directly observed on a genome-wide basis, thus we use the only experiment that offers an approaching information: genome-wide chromosome conformation capture, also called Hi-C [Lieberman2009].

The determination of homologous genomic regions is a complex problem. In order to simplify it and be able to work more easily with genomes of different size (around 150Mpb for fly to compare with the multiple Gpb of Human!), we chose to use orthologous genes. Basically, orthologous genes (or orthologs) are homologous genes found in different species. For more information on how to get them, please refer to the Quest for Orthologs consortium website.

Let *a1* and *b1* be two genes in the genome 1; *a2* and *b2* be two genes in the genome 2. *a1* and *a2* are orthologs, so are *b1* and *b2*. The idea is to look at the contacts between *a1* and *b1*, and to compare them with the contacts between *a2* and *b2*. For that we compute the ratio between their number of contacts.

To completely compare two genomes, we compute the ratios for all possible pairs of orthologs.

2.1.2 General Process

The basic process consists of the following steps:

1. **Prepare the data.** Basically, this means selecting the orthologous genes that will be used, checking the orthology relationships (only 1-1), getting Hi-C contacts, taking care of the Hi-C data normalization.
2. **Look at the contacts for all pairs of genes.** This is done inside a genome. Thus, this step has to be repeated for each analyzed genome.
3. **Join the contacts between two genomes.** This joining is done by using the orthology relationships: the genes *a1* and *b1* in the genome 1 are joined to their orthologs counterparts *a2* and *b2* in the genome 2. The term *joining* here is just a placeholder for the performed mathematical operation ; as said earlier, in our case we just compute a ratio. The process is applied to all pairs of orthologs in both genome 1 and 2. This step has to be repeated for each pair of analyzed genome.
4. **Make the distances matrix.** This is done by computing a distance for each pair of genome using the results of the previous step.
5. **Infer a phylogeny** using the distances matrix computed at the step 4. For this step, regular distance-based methods and tools are used. Specifically, we used the BioNJ algorithm [Gascuel1997] implemented in the FastME tool [Lefort2015].

Disconnecting steps 2 and 3 enables parallel computation. Since step 2 is computed on a per-species basis, all species can be run in parallel. Likewise, all pairs of species can be run in parallel for step 3. in this pipeline, the parallel computations are handled automagically with Snakemake [Köster2012] and will be made explicit in the next sections.

2.1.3 Parameters

There are three main parameters that affect the results.

1. Species and orthologs.
2. Hi-C resolution, also called binsize.
3. Hi-C threshold.

The effect of changing the species is obvious. Changing the orthologs means looking at different genomic regions. While this will obviously change the experiment results, we cannot predict in which way. Thus, the whole pipeline has not been designed to save computation time when changing species and/or orthologs.

Changing the Hi-C resolution enables more precise results. Indeed, when the binsize is smaller (and the Hi-C resolution is higher), one can better distinguish contact locations. This leads to a better map of which gene contacts which one. However, lowering binsize is a double-edge sword. Since binning is done in order to get more signal, lowering it dilutes that signal. This can potentially lead to a loss of information and eventually wrong results. Finding the good resolution depends on the data and a lot of tries. In this pipeline, the choice of a binsize is done at step 2. One can run in parallel multiple steps 2 for the same species, each one with a different binsize.

The comparison of the contacts for a pair of orthologs between two species is done using a ratio. We usually don't want to be sensitive to the order of magnitude of the number of contacts. That is why we chose a ratio. However, if the contacts from both species are really small, this can mean that they are not informative (not enough signal). In order to not being affected by this flaw, a threshold is applied on the contact data during step 3. Testing for different threshold values can be done by running in parallel different steps 3 with different threshold values.

2.2 A few concepts to define

Before to jump in the next section, let's agree on a few term and their associated concepts we'll use.

2.2.1 Genes Adjacency

Two genes are *adjacent* if and only if they are on the same chromosome (or DNA molecule) and next to each other on the DNA sequence. Obviously, this concept is only relevant for genes from the same genome.

Now, let $S1$ be a set of genes from a species 1 and $S2$ be a set of genes from a species 2. Then let

$$S1 \times S2 = \{((a1, b1), (a2, b2))\}$$

with

- $a1 \in S1$,
- $b1 \in S1$,
- $a2 \in S2$,
- $b2 \in S2$,
- $a1$ and $a2$ are orthologous,
- $b1$ and $b2$ are orthologous.

This corresponds to the step 3 of the *General Process*.

When collecting the pairs of orthologous from $S1 \times S2$, we can filter the pairs of pairs of genes based on the adjacency in $S1$ and $S2$. The following situations are possible:

- The genes are adjacent in both pairs ($a1$ and $b1$ are adjacent as are $a2$ and $b2$).
- The genes are *not* adjacent in both pairs ($a1$ and $b1$ are *not* adjacent as are $a2$ and $b2$).
- The genes are adjacent in $S1$ but not in $S2$ ($a1$ and $b1$ are adjacent while $a2$ and $b2$ are *not* adjacent).
- The genes are *not* adjacent in $S1$ but are in $S2$ ($a1$ and $b1$ are *not* adjacent while $a2$ and $b2$ are adjacent).

We define the following strategies of collecting such pairs of pairs of genes:

- All pairs are collected without taking the adjacency into account; we call this case `all`.
- The pairs are collected only when both genes pairs are *not* adjacent; we call this case `none`.
- The pairs are collected only when the genes of one genes pair are adjacent while the genes of the other genes pair are not; we call this case `xor`.
- The pairs are collected only when at least one of the genes pairs has its genes adjacent; we call this case `or`.
- The pairs are collected only when the genes are adjacent in both genes pairs; we call this case `and`.

2.2.2 Values selection mode

The result of the step 3 of the *General Process* is a list of values for each pairs of pairs of genes between each pairs of genomes. This list can contain the same pairs of orthologs among the different pairs of genomes or not. This depends on the orthologs chosen in the first place and on the availability of contacts data along each genomes.

The previous facts can have the following effect: different pairs of genomes raises (partially or totally) different pairs of orthologs. Since we cannot know *a priori* if this is an issue or not, we put names on the different situations in order to make it possible to work with each of them in step 4 of the *General Process*.

These situations are:

- All values are kept, whatever they are present in all or just a subset of the pairs of species; we call that situation `union`.
- Only the values that are present in all pairs of species are kept; we call that situation `intersection`.
- Only the values that are present in at least two pairs of species (so at least 3 species) are kept; we soberly call that situation `atLeastTwo`.

2.2.3 Randomization

An important part of this work is to test whether or not a phylogenetic signal is present in the contact data. In order to achieve that goal, we need to apply the method to the actual data, then to a randomized set of data, and finally to compare them.

Let's talk about what we call *a randomized set of data*.

Scrambling matrices

The contact data are represented as matrices, each axis being the coordinates along a chromosome. Thus, a box in such a matrix corresponds to the contacts between the two chromosomes at the given coordinates.

Now, let's take a particular box (that is a pair of genes) in a matrix. Scrambling that matrix multiple times will make the contacts in this box tend to the mean number of contacts of the whole matrix.

Consequently, by scrambling a matrix, we lose the structural information. Thus, comparing the results obtained using actual with the ones obtained from multiple scrambling allows us to look for a phylogenetic signal.

The scrambling is done at step 2 of the *General Process*. Technically, we use the Fisher–Yates shuffle [Durstenfeld1964] to scramble the matrices.

Bootstrap

At some point during the scientific process, we performed a step of bootstrap. The bootstrap has been introduced by Efron [Efron1979] and applied to phylogenies by Felsenstein [Felsenstein1985].

We used it on the pairs of genes after joining (step 3 of *General Process*). The idea was to make multiple distances matrices by bootstraping, then to compute the actual distance matrix. After that, we inferred all phylogenetic trees, and compared them.

However, after discussions it appeared we couldn't interpret our results. The code that produced those results is still in the repo, in the hope that the methodology could be fixed.

2.3 Preparing Data and Running an Analysis

Here are the steps needed to run one analysis (you can also read one experiment). By this, we mean measuring the distance between all species for a fixed set of species, a fixed set of orthologous genes and a fixed set of "ready-to-use" Hi-C datasets (understand corrected for experimental biases and normalized). However, the pipeline does allow you to compute the distances for several Hi-C resolutions/binsizes.

2.3.1 Prepare the data

1. Prepare the gene locations files for each species, in usual BED format.
2. Prepare the orthologs file in the *appropriate format*.
3. Prepare Hi-C data in the *appropriate format*.
4. Prepare the configuration file, in YAML, using the *commented sample*. This file is used by SnakeMake, so keep it safely. Using the same config file with the same datasets guarantees to re-compute the very same results.

2.3.2 Running the pipeline

You just need to launch SnakeMake (with a configuration file named `config.yaml`):

```
snakemake --configfile config.yaml
```

You can ask SnakeMake to perform multiple steps at once, if possible. For example, to use 6 jobs at the same time:

```
snakemake --configfile config.yaml -j6
```

SnakeMake can also automagically spans its jobs on a clustering system. However, be aware that this functionality is system-dependant. Here is a basic example with an SGE scheduler:

```
snakemake --configfile config.yaml -j6 --cluster 'qsub -o outfile -e errfile'
```

2.3.3 Results

The pipeline outputs a distance matrix in PHYLIP format called `all_replicates.phylip`. It also creates a number of intermediate files that are kept in case other analysis should be performed. This files are :

- the *pairs files* which are described *here*,
- the *values files* which are described *here*,
- the *stats file* which is described *here*.

2.4 Scripts and API documentation

2.4.1 Scripts

About the pipeline:

1. *make_pairs.py* which computes the contacts of pairs of genes.
2. *join_pairs.py* which extracts the intersection of pairs from two tables of contacts. The intersection is computed using the orthologs.

3. From there, we have two possibilities:
    - *dist_all_pairs.py* if we want to look at all pairs,
    - *dist_pairs_indep.py* if we want to look at each pair independently.

Tools (not exhaustive):

- *statshic.py* computes basic statistics over a dataset and output the results directly or write it as JSON.
- *norm_center.py* normalizes a buch of dataset in order to make them comparable with each other.
- *informative_traits.py* reports information about the traits that are informative, *i.e.* the traits that don't have the same value in all species.

Scripts usage:

make_pairs.py

Make the pairs of genes for a species.

For each pair, fetch the Hi-C values for two bins where the TSS are located. Also check the adjacency of the genes. If wanted, the Hi-C matrices can be scrambled in-memory before use. This feature uses two functions written by Krister SWENSON for the **locality** program.

The result is a TSV file, optionally Gzipped (if the file name has the extension .gz). The file has no header and the following columns:

- gene 1 name (string)
- gene 2 name (string)
- Hi-C value (64 bits float)
- Adjacency status (either True or False, case-insensitive)

  created May 2018

  last modified July 2018

Usage

Make the pairs of genes for a species.

```
usage: make_pairs.py [-h] [-N] [-i] [-s] [-v] [--debug] genes hic output
```

Positional Arguments

genes	the genes, in BED
hic	the directory with the Hi-C sparses matrices
output	explicit enough; can be gzipped

Named Arguments

-N, --no-nan	skip the pair if the Hi-C value is Not-a-Number Default: False
-i, --intra	only look at genes on the same chromosomes; if not set and there is no interchromosomal values, Not-a-Number is used Default: False
-s, --scramble	Scramble in-memory the Hi-C matrices before use Default: False
-v, --verbose	be verbose Default: False
--debug	print debug information Default: False

join_pairs.py

This script reads two pairs files (the left one and the right one) made with *make_pairs.py* and join them based on a list of orthologs.

The result is a data set with 5 columns:

- (str) gene name 1 in species left
- (str) gene name 2 in species left
- (str) gene name 1 in species right
- (str) gene name 2 in species right
- (float) Hi-C value for gene 1 and 2 in species left
- (float) Hi-C value for gene 1 and 2 in species right

This table is written as TSV with no header row, in a Gzipped file.

created May 2018

last modified July 2018

Usage

Reads two pairs files (the left one and the right one) and join them based on a list of orthologs.

```
usage: join_pairs.py [-h] [-t THRESHOLD] [-x] [-a {all,none,and,or,xor}] [-v]
                     orthos left right outfile
```

Positional Arguments

orthos	the orthologs file, can be a pair of species or not (see doc)
left	the left pairs file, optionally Gzipped

right	the right pairs file, optionally Gzipped
outfile	the output file, optionally Gzipped

Named Arguments

-t, --threshold	the threshold to apply
-x, --exclude	exclude the pairs with both values lesser than or equal to the threshold
	Default: False
-a, --adjacencies	Possible choices: all, none, and, or, xor
	the adjacencies status to keep
	Default: "all"
-v, --verbose	be verbose
	Default: False

bootstrap.py

Randomly sample the pairs of orthologs and compute the scaled distance using that sample. The sample as the same size as the original set of orthologs (the sampling is done with replacement). This is the first step of a bootstrap.

The mode argument define what kind of value we want to keep while computing the distance:

- *intersection*: only the values present in all species are kept.
- *atLeastTwo*: keep the values present in at least two species.
- *union*: keep all values.

Note: It is intended to be used *instead of* *dist_all_pairs.py* and *dist_pairs_indep.py*.

---

created May 2018

last modified August 2019

Usage

Perform the first step of a bootstrap by sampling at random the orthologs and computing a distances matrix with that sample.

```
usage: bootstrap.py [-h] [-o] [-m {intersection,atLeastTwo,union}] [-p] [-v]
                    orthos outdir n values [values ...]
```

Positional Arguments

orthos	all the orthos, in TSV
outdir	the dir for the resulting samples
n	the number of replicates

values the values files, in (Gzipped) TSV

Named Arguments

-o, --one-file put all matrices in one file called all_replicates.phylip

Default: False

-m, --mode Possible choices: intersection, atLeastTwo, union

the mode, that is, the kind of values we want to keep while computing the distance

Default: "intersection"

-p, --progress print a progress bar; need tqdm to be installed

Default: False

-v, --verbose be verbose

Default: False

dist_all_pairs.py

Compute the scaled distance using all pairs of orthologs (the "standard way"). The result is **one** distance matrix for a full dataset.

The mode argument define what kind of value we want to keep while computing the distance:

- *intersection*: only the values present in all species are kept.
- *atLeastTwo*: keep the values present in at least two species.
- *union*: keep all values.

Note: It is intended to be used *instead of* *dist_pairs_indep.py* and *bootstrap.py*.

---

created August 2019

last modified August 2019

Usage

Compute the distance matrix using all pairs of genes.

```
usage: dist_all_pairs [-h] [-m {intersection,atLeastTwo,union}] [-p] [-v]
                      orthos outfile values [values ...]
```

Positional Arguments

orthos all the orthos, in TSV

outfile the matrix filename

values the values files, in (Gzipped) TSV

Named Arguments

-m, --mode	Possible choices: intersection, atLeastTwo, union
	the mode, that is, the kind of values we want to keep while computing the distance
	Default: "intersection"
-p, --progress	print a progress bar; need tqdm to be installed
	Default: False
-v, --verbose	be verbose
	Default: False

dist_pairs_indep.py

This script computes one distance matrix by pair of orthologs. Due to the fact that we want one matrix per pair of genes, this script runs in intersection mode (*i.e.* only the pairs present in all species are used).

Note: It is intended to be run *instead of* *dist_all_pairs.py* and *bootstrap.py*.

created June 2018

last modified August 2019

Usage

Compute the distance matrix on each pair of genes independently.

```
usage: dist_pairs_indep [-h] [-o] [-p] [-v] orthos outdir values [values ...]
```

Positional Arguments

orthos	all the orthos, in TSV
outdir	the directory in which put the results
values	the values files, in (Gzipped) TSV

Named Arguments

-o, --one-file	put all matrices in one file; outdir is this file name
	Default: False
-p, --progress	print a progress bar; need tqdm to be installed
	Default: False
-v, --verbose	be verbose
	Default: False

statshic.py

Compute basic statistics about the given Hi-C dataset.

The statistics are the following:

- Mean
- Standard deviation
- Median
- Percentiles at 10, 25, 75 and 95%

Also report the dimensions of the matrices and their size (number of elements). For the whole datasets, only the size is given.

created August 2018

last modified August 2018

Usage

Compute basic statistics about the given Hi-C dataset. The results are printed to standard output or written as JSON.

```
usage: statshic [-h] [-w] [-o OUTPUT] hic
```

Positional Arguments

hic	the Hi-C directory with the metadata.json file

Named Arguments

-w, --whole-dataset	compute also the stats on the whole dataset; may use a lot of memory Default: False
-o, --output	the output file, in JSON

norm_center.py

This script normalizes multiple Hi-C datasets so they can be compared together. This is done in two steps, each one being done independently on each dataset.

The first step consists of the following process:

1. All matrices are read into memory.
2. The mean and standard deviation are computed.
3. For each box of the matrices, the mean is subtracted from the box, then the box is divided by the standard deviation.
4. The normalized matrices are written.

The second step is simple: we just add to each box in each matrix from each dataset the minimum value over all dataset.

created August 2018

last modified August 2018

Usage

This script normalizes multiple Hi-C datasets so they can be compared together. This is done by first subtracting the mean from each box, then by dividing each one by the std. dev. and finally by adding the last value over all dataset. For more details, see the head of that file. The normalized datasets are named after the original ones suffixed with _centernorm

```
usage: norm_center [-h] [-t THREADS] [-v] [--debug] datasets [datasets ...]
```

Positional Arguments

datasets	the original datasets

Named Arguments

-t, --threads	the number of threads to use; high values use more memory
	Default: 1
-v, --verbose	be verbose
	Default: False
--debug	print debug information
	Default: False

informative_traits.py

Report information about the traits that are informative, *i.e.* the traits that don't have the same value in all species.

The result is a key-value association, either displayed in the console or written to a file as TSV or JSON.

The information reported is:

- The total number of values (keyed `TotalSize`).
- The number of informative values (keyed `InformativesSize`).
- The percentage of informative values (keyed `PercentageInformative`).
- The first non informative value seen during the computation (keyed `NonInformativeValue`); this was used mainly for debugging.

created September 2018

last modified August 2019

Usage

Report information about the traits that are informative

```
usage: informative_traits.py [-h] [-o OUTPUT]
                             [-m {intersection,atLeastTwo,union}] [-p] [-v]
                             orthos values [values ...]
```

Positional Arguments

orthos	all the orthos, in TSV
values	the values files, in (Gzipped) TSV

Named Arguments

-o, --output	write the output there; can be JSON or TSV
-m, --mode	Possible choices: intersection, atLeastTwo, union
	the mode, that is, the kind of values we want to keep while computing the distance
	Default: "intersection"
-p, --progress	print a progress bar; need tqdm to be installed
	Default: False
-v, --verbose	be verbose
	Default: False

2.4.2 API

The scripts make use of the following modules:

distlib

This module contains functions that compute distances and helpers.

created August 2019

last modified August 2019

distlib.**filter_values**(*constraint*, *species*, *values*)

Filter the *values* based upon the *species* and the wanted *constraint*:

- *intersection*: only the values present in all species are kept.
- *atLeastTwo*: keep the values present in at least two species.
- *union*: keep all values.

Note: This function is a generator.

> **Warning:** If *constraint* is something else than *intersection*, *atLeastTwo* or *union*, then a ValueError exception is raised.

`distlib.`**`scaled_L2norm`**`(`*species*, *values*`)`
Compute the distances using the L2-norm scaled by the size with the *values* for all pairs of *species*.

genes

This module contains functions that parses BED files containing genes and compute the adjacency of genes.

A `gene` is a `dict` with the following keys:

- `chrom`, a `str`
- `start`, an `int`
- `end`, an `int`
- `name`, a `str`
- `strand`, a `str`, but can only be + or -

In this file, the `genes` argument refers to a list of such dicts.

created May 2018

last modified September 2018

`genes.`**`compute_adjacent`**`(`*genes*`)`
Look for the adjacent genes of all genes and return a mapping of a gene name to its left and right neighbours. The strand is not taken into account for this.

> **Warning:** *genes* is expected to be sorted.

`genes.`**`make_bed`**`(`*genes*`)`
Convert the *genes* to BED. Return the resulting string.

> **Note:** The *score* column is set to 0.

`genes.`**`read_bed`**`(`*name*`)`
Read the BED file *name*, and return a list of dict. Each dict has the following keys: chrom, start, end, name, strand. The list is sorted.

> **Note:** The BED file is loaded into memory for faster processing.

`genes.`**`write_bed`**`(`*genes*, *name*`)`
Write the *genes* to the BED file *name*.

hic

This module contains classes used to work with Hi-C data.

class hic.**HiC**(*dirname*)

Bases: object

HiC represents an Hi-C experiment, handling the matrices reading, and position fetching.

The experiment is encapsulated into a directory, with one file per matrix. The matrices are written as **gzipped TSV** files with 3 columns: *row* position, *column* position and *value*. The matrices are **sparse**.

There is also a JSON file called *metadata.json* of the following form:

```
{
  "Binsize": 5000,
  "Assembly": "droYak2",
  "Species": "Drosophila yakuba",
  "Comment": "nm_none - No NaN",
  "Date": "",
  "Dataset": "Dyak_c",
  "Dims": {
    "2R|2R": [1234, 1234],
    "4|X": [345, 2345],
    ...
  }
  "MapFiles": {
    "2R|2R": "2R_2R.tsv.gz",
    "4|X": "4_X.tsv.gz",
    ...
  }
}
```

Please note that the format of the file names is free, but the one of the keys is not. This is of the form *chromosome|chromosome*.

Created May 2018

Last modified August 2018

binsize = None
The binsize of the dataset.

chromosomes = None
The list of chromosomes for which data are available.

current = None
The currently loaded Hi-C map.

get_contact(*g1*, *g2*)
Fetch the contact between the genes *g1* and *g2*. They are dict of the same kind as returned in the *genes*. If the genes are not on the same chromosomes as the currently loaded heatmap, an exception is raised. If at least one of them is outbound, Not-a-Number (NaN) is returned.

inter = None
True iff the dataset contains inter-chromosomal contacts.

load_all_maps()
Load all the matrices. The result is an array with all the values.

> **Warning:** This function can need a lot of memory.

> **Warning:** This function's result may change in the future.

load_map(*rowChrom*, *colChrom*, *scramble=False*)
Load the wanted matrix. If needed, the row and column chromosomes will be swapped. If there is no matrix with this pair of chromosomes, a *NoSuchHeatmap* error is raised. if *scramble* is *True*, then the matrix is scramble in-place after being loaded.

exception hic.**NoSuchHeatmap**
Bases: Exception

iolib

This module contains functions that parses tabular files used to store orthologs or pair values, and to format output such as making PHYLIP matrices.

created June 2018

last modified August 2019

iolib.**phylip**(*species*, *distances*)
Make the distance matrix for the wanted *species* in PHYLIP format. The resulting matrix is returned as a single string object.

> **Warning:** If not all species are present in distances, a KeyError exception is raised.

iolib.**read_orthos**(*name*)
Read the orthologs from a TSV file and return a set of the orthologs names (from all species) and a mapping of the ortholog names to their orthology group.

The orthologs file is described *in the documentation*.

iolib.**read_values**(*orthos*, *groups*, *sp_left*, *sp_right*, *name*)
Read the value file at *name* and return a dict of dict of group ids to species name to Hi-C value.

The values file is described *in the documentation*.

> **Note:** Extract only the values where all four genes are present in *orthos*.

2.5 File Formats

2.5.1 Experiment Configuration file

This file contains all the needed configuration to run an experiment. It's used by SnakeMake as well as the scripts. It's a standard YAML formatted file.

> **Warning:** All path shall end with a /

Here is the content of `config/sample.yaml` that shows all available configuration keys with a lot of comments. In this sample, we have three species (suitably named *Species1*, *Species2* and *Species3*), and two Hi-C resolutions (10kb and 20kb).

```
# sample.yaml

# Sample config file.

#-------------------------------------------------------------------
# WARNING: all path shall end with a /
#-------------------------------------------------------------------

# Where are the scripts?
bin: "/path/to/python/scripts"

# Activate a conda environment. If not needed, leave empty
condacmd: "source /path/to/conda/bin/activate myenv"

# The path to the root directory for the results (holds multiple experiments)
rootdir: "/path/to/rootdir"

# The experiment name
experiment: "myexpe"

# The path to the TSV with all the orthologs
orthologs: "/path/to/the/orthologs.tsv"

# The method for joining pairs, exactly one of "union", "intersection" or
# "atLeastTwo. See the documentation for more information.
method: "union"

# Will we exclude pairs with both values lesser than or equal to the threshold?
exclude: false

# The Hi-C resolutions
resolutions: ["10kb", "20kb"]

# Let's talk about data...
datasets:

  # The first dataset, "Species1" will be used as dataset name in the results
  Species1:
    # Its genes, in BED format
    genes: "/path/to/sp1/genes.bed"
    # The different directories for the Hi-C, one directory per resolutions
    hic:
      # The keys are the resolutions, and the values the paths
      10kb: "/path/to/hic/at/10kb/"
      20kb: "/path/to/hic/at/20kb/"

  # Idem for the other datasets

  Species2:
    genes: "/path/to/sp2/genes.bed"
    hic:
      10kb: "/path/to/sp2/hic/at/10kb/"
      20kb: "/path/to/sp2/hic/at/20kb/"
```

(continues on next page)

(continued from previous page)

```
  Species3:
    genes: "/path/to/sp3/genes.bed"
    hic:
      10kb: "/path/to/sp3/hic/at/10kb/"
      20kb: "/path/to/sp3/hic/at/20kb/"
```

2.5.2 Hi-C dataset format

An Hi-C dataset is a folder containing the heatmaps and a `metadata.json` file.

Metadata file

As its name suggests, the JSON file holds basic metadata about the dataset. Its format is the following:

```
{
  "Binsize": 5000,
  "Assembly": "droYak2",
  "Species": "Drosophila yakuba",
  "Comment": "nm_none - No NaN",
  "Date": "",
  "Dataset": "Dyak_c",
  "Dims": {
    "2R|2R": [1234, 1234],
    "4|X": [345, 2345],
    ...
  }
  "MapFiles": {
    "2R|2R": "2R_2R.tsv.gz",
    "4|X": "4_X.tsv.gz",
    ...
  }
}
```

Note: The *MapFiles* maps to the key-value store of the heatmaps files. The files can be named in any way, but not the key. They must be of the form *X|Y*.

Heatmap files

There are one file per matrix. The matrices are written as tabulated-separated files, optionally gzipped. There is no header row, and 3 columns:

- row position (an integer),
- column position (an integer),
- value (a floating-point number).

In order to save space, only the non-empty boxes from the matrices are written (the matrices are sparse).

2.5.3 Orthologous Genes File

This file contains the orthology relationship between the different species. It's a tabulated-separated values file, *not* gzipped. The header row contains the species names or ID (or taxonomic IDs or whatever names you used for the species in the *configuration file*). The other rows contain the gene identifiers used in the genes locations files (those files are in BED format, so this identifiers are likely the gene IDs from the database used to get the genes).

On a row, all genes are orthologous together.

Here's an example with six species:

```
10090               46245       7227        7240        7245        9606
ENSMUSG00000035948 FBgn0079584 FBgn0039184 FBgn0192545 FBgn0240653 ENSG00000111058
ENSMUSG00000039632 FBgn0072823 FBgn0036219 FBgn0186022 FBgn0239103 ENSG00000198003
ENSMUSG00000031578 FBgn0070523 FBgn0030067 FBgn0196092 FBgn0233372 ENSG00000198042
ENSMUSG00000014856 FBgn0080973 FBgn0034059 FBgn0182903 FBgn0229535 ENSG00000168701
ENSMUSG00000026869 FBgn0245366 FBgn0030457 FBgn0188676 FBgn0234534 ENSG00000095261
ENSMUSG00000018845 FBgn0075456 FBgn0010812 FBgn0191449 FBgn0242058 ENSG00000141161
ENSMUSG00000033629 FBgn0081643 FBgn0032524 FBgn0193470 FBgn0229380 ENSG00000074696
ENSMUSG00000004264 FBgn0073511 FBgn0010551 FBgn0187108 FBgn0229780 ENSG00000215021
ENSMUSG00000025939 FBgn0080184 FBgn0033544 FBgn0182517 FBgn0230555 ENSG00000104343
```

2.5.4 Pairs File

An intermediate file created by the pipeline when looking at the contacts between each pairs of genes. It is created by the script *make_pairs.py*.

It is a tabulated-separated values file, usually gzipped. There is no header row. Each rows corresponds to a pair of genes. Then, the columns are the following:

- the name for the first gene of the pair, a string,
- the name for the second gene of the pair, a string,
- the Hi-C value (*i.e.* the corrected and normalized number of contacts) for these genes, a floating-point number,
- the adjacency status, *i.e.* if these genes are next to each other along the chromosome, a boolean value (either True or False, case-insensitive).

2.5.5 Basic Statistics Intermediate File

This file is created by the script `pairsStats.go` while reading the *values files*. It is a tabulated-separated values file, *not* gzipped. The header row contains the column names. The other rows contains the basic statistics computed for each datasets (which are the script arguments). The last row contains the mean of all datasets.

2.5.6 Values File

An intermediate file created by the pipeline when "joining" the contacts between different species based on the orthology relationships. It is created by the script *join_pairs.py*.

It is a tabulated-separated values file, usually gzipped. There is no header row. For each row, we have two species (named *left* and *right*), and four genes, two by species. The gene 1 in species left and the gene 1 in species right are orthologs, as are the gene 2 in species left and the gene 2 in species right. Then, the columns are the following:

- the name of the gene 1 in species left, a string,
- the name of the gene 2 in species left, a string,

- the name of the gene 1 in species right, a string,
- the name of the gene 2 in species right, a string,
- the Hi-C value (*i.e.* the corrected and normalized number of contacts) for the genes 1 and 2 in species left, a floating-point number,
- the Hi-C value for the genes 1 and 2 in species right.

Bibliography

[Gascuel1997] Gascuel O., “BIONJ: an improved version of the NJ algorithm based on a simple model of sequence data.”, Molecular Biology and Evolution. 1997 14:685-695.

[Lieberman2009] Lieberman-Aiden E., van Berkum N. L., Williams L., Imakaev M., Ragoczy T., Telling A., Amit I., Lajoie B. R., Sabo P. J., Dorschner M. O., Sandstrom R., Bernstein B., Bender M. A., Groudine M., Gnirke A., Stamatoyannopoulos J., Mirny L. A., Lander E. S., Dekker J. “Comprehensive Mapping of Long-Range Interactions Reveals Folding Principles of the Human Genome” Science 09 Oct 2009, Vol. 326, Issue 5950, pp. 289-293 doi: 10.1126/science.1181369

[Köster2012] Köster J. and Rahmann S., “Snakemake — a scalable bioinformatics workflow engine”, Bioinformatics, Volume 28, Issue 19, 1 October 2012, Pages 2520–2522, doi: 10.1093/bioinformatics/bts480

[Lefort2015] Lefort V., Desper R. and Gascuel O., “FastME 2.0: A Comprehensive, Accurate, and Fast Distance-Based Phylogeny Inference Program”, Molecular Biology and Evolution, Volume 32, Issue 10, 1 October 2015, Pages 2798–2800, doi: 10.1093/molbev/msv150

[Durstenfeld1964] Durstenfeld R., “Algorithm 235: Random permutation”, Communications of the ACM, Volume 7 Issue 7, July 1964, Page 420, doi: 10.1145/364520.364540

[Efron1979] Efron B., “Bootstrap Methods: Another Look at the Jackknife”, The Annals of Statistics, Volume 7, Number 1 (1979), Pages 1-26, doi: 10.1214/aos/1176344552

[Felsenstein1985] Felsenstein J., “Confidence Limits on Phylogenies: an Approach Using the Bootstrap”, Evolution, 1985, 39, 783-791, doi: 10.1111/j.1558-5646.1985.tb00420.x

Python Module Index

Index