PGS: Population-based Genome Structure Modeling Documentation

Release 0.0.1

Hanjun Shin, Nan Hua, Harianto Tjong

Mar 14, 2017

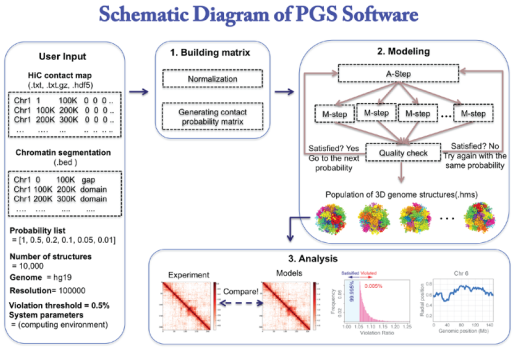
Contents

PGS is a population-based 3D genome-modeling package implemented in Python. The software takes Hi-C matrix and chromosome regions segmentation or topological associated domains(TADs) information, which then generates an ensemble of structure population. The software also automatically generates analysis reports, such as structure quality based on scoring parameters, plots of radial positions and contact frequency maps from the structures. The whole codes are wrapped in Python, and users can simply execute it one time.

Note: PGS software will run on high performance computing environment (HPC), such as sun grid engine and TORQUE (pbs script), as well as local machine. But, it is highly recommended to run on HPC because of computational resource and running time.

The code is open source, and available on github.

Here is the overview of PGS pipeline:



Contents:

CHAPTER 1

Quickstart

Installation

Requirements:

- Python 2.7
- Python packages `numpy`, `scipy`, `pandas`, `h5py`, `matplotlib` , `seaborn`
- IMP (Integrative Modeling Package.)

Conda package is recommended to install all the requirements. Either Anaconda or the minimal Miniconda are suitable for managing required packages including IMP. If you use Miniconda, then you can install as follows:

```
$ conda install numpy scipy pandas h5py matplotlib seaborn
```

Install IMP using conda:

```
$ conda config --add channels salilab
$ conda install imp
```

All other dependencies for imp and python packages will be automatically installed.

Then install PGS workflow packages:

```
$ python setup.py install
```

To test if the installation goes well, the following command can be executed:

```
$ cd test
$ ./runPgs_workflow_test.sh
```

PGS Helper GUI

PGS package includes a Graphical User Interface (GUI) based helper program for user to run PGS easily. With this, a user can generate a command script `runPgs.sh` and a configuration file `input_config.json`.

Tip: PGS Helper uses Java Runtime Envrionment, the latest Java SE 8 update is recommended.

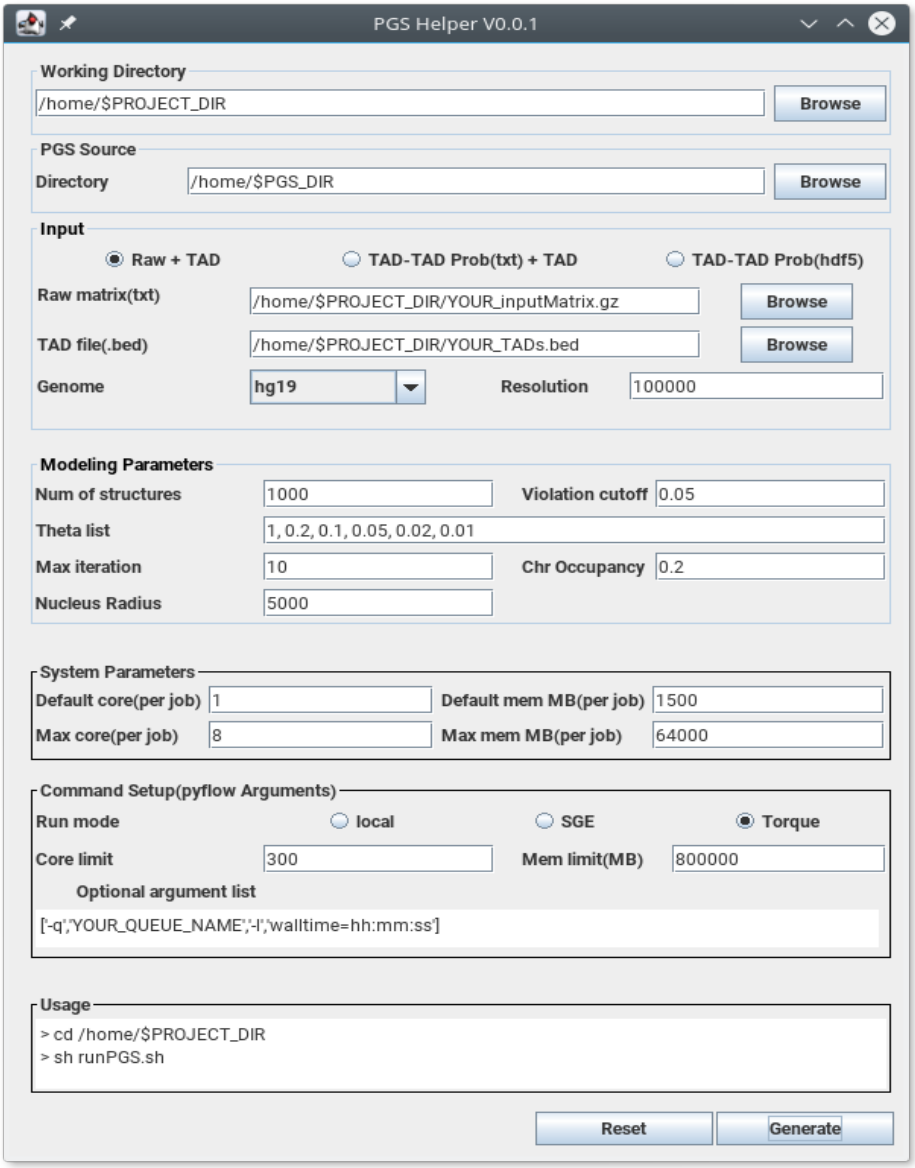
Note: PGS Helper is a convenient interface but not required to run PGS. We also provide the `runPgs.sh` and `input_config.json` examples that are easy to modify (*PGS Helper Output*) without using the GUI.

Using the PGS Helper

To initialize PGS Helper:

```
$ java -jar PGSHelper.jar
```

The following GUI will appear:



A. Working Directory

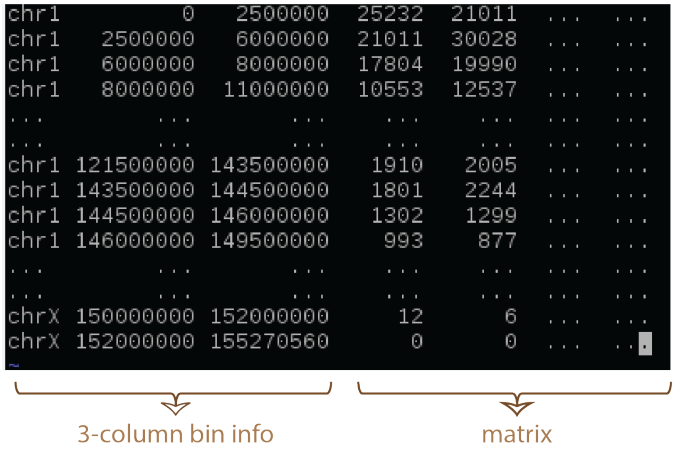
Specify the project/working directory using the `Browse` button on right side. PGS will run in the specified project directory and all files, such as running script(i.e. `runPGS.sh`), configuration file(i.e. `input_config.json`), log(i.e. `pyflow.data`), and output results, will be stored in the directory.

B. PGS Source – Directory

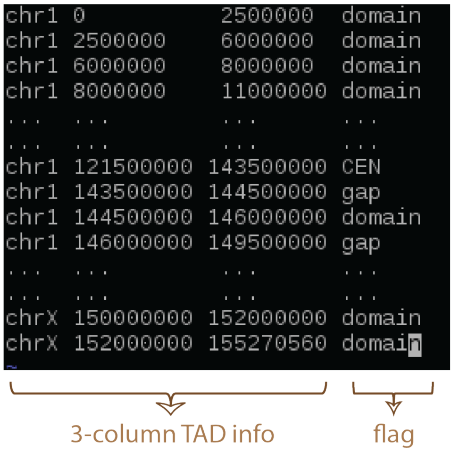
Specify the directory of PGS source code using the `Browse` button on right side.

C. Input

Contact frequency matrix file format



Chromosome segmentation (TADs)



Note: Currently our pipeline only support diploid genome, so chromosome Y is not supported and please don't include it in the input file. We will be updating the support for chromosome Y soon.

- Experiment data

  Option 1 : Raw + TAD

  - Raw contact matrix file (txt); the first three columns contain chromosome, start position (bp), and end position (bp) and followed by contact matrix (all numeric values must be **integers**; see figure above).
  - TAD_file (bed); a 4-column chromatin segmentation or TAD file, we adopt bed file format while the 4-th column must contain "domain", "CEN", and "gap" (see figure above).

  Option 2 : TAD-TAD Prob(txt) + TAD

  - Probability matrix file (txt); adopts the same format as the raw contact matrix file above, but the matrix contains probability values from 0 to 1.
  - TAD_file (bed); the format as figure above.

  Option 3 : TAD-TAD Prob(hdf5)

  - Probability matrix file (hdf5) : if a user have generated probability matrix using PGS (i.e. under old `$PROJECT_DIR/result/probMat/probMat.hdf5.hmat`), then the user can use the previous probability matrix. This process will skip the first workflow, buildTADMap task. This option is good for replica calculations (in a new working directory, of course).

- Genome : the genome version of 3D models being constructed (current PGS supports chromosomes 1-22 and X).
- Resolution : the resolution of raw input data (in bp).

D. Modeling Parameters

- Num of structures : the number of structures to generate. `default = 1,000`
- Violation cutoff : violation cutoff. `default = 0.05`

- Theta list : a probability list for step-wise optimizations; 1 < theta < 0. `default = 1, 0.2, 0.1, 0.05, 0.02, 0.01`
- Max iteration : the number of maximum iterations for each theta. `default = 10`
- Chromosome Occupancy : the expected ratio of chromosome volume in the nucleus. `default = 0.2`
- Nucleus Radius : assumed radius (nm) of cell nucleus. Changing this value will change the relative size of each domain sphere. `default = 5000.0`

E. System Parameters

In order to proceed efficiently, PGS submits both single-core and multi-thread jobs on HPC clusters (e.g. for the M-step and A-step jobs, respectively). Thus the following parameters need to be specified. - Default core : the number of cores to use for each regular job. - Default MemMB : the memory (Mb) to use for each regular job. - Max cores : the number of cores to use for each multi-thread job. - Max MemMB : the total memory (Mb) to use for each multi-thread job.

F. Command Setup

- Run mode : select a platform such as local computer, Sun Grid Engine (SGE) or Torque.
- Core limit : the maximum number of cores for PGS to use (limited to user's quota).
- Mem limit : the limit of memory for PGS to use.
- Optional argument list : additional options for each job to run/be assigned properly on the user's HPC, such as queue name, running time, etc. Example arguments for PBS jobs: `['-l','your_qname_here','-l','walltime=333:00:00']`. Note that the option list will be applied to each job.

G. Generate Scripts

Click the `Generate` button on the bottom to write a file (input_config.json) with the parameters on the working directory which has been specified by the user. There will be a confirmation window with `Yes` or `No` button, and at this point the user can see a simple instruction in the `Usage` box. If `Yes` is clicked, then the GUI will be closed.

PGS Helper Output

PGSHelper writes configuration in `input_config.json`, and a shell script (`runPGS.sh`) under the project directory. The following describes the contents of those 2 files.

1. `$PROJECT_DIR/input_config.json`

```
{   "source_dir" : "[Directory name where pgs source is]",
    "input" : {
    "raw_matrix_file " : "[raw matrix file]",
        "TAD_file" : "[ TAD file, .bed format]",
        "resolution" : "[Resolution of input contact_map_file, e.g. 100000]",
        "genome" : "[Genome version, e.g. hg19]"
    },
    "output_dir" : "[Output Directory to store the results, e.g. $PROJECT_DIR/result]
↪",

    "modeling_parameters" : {
```

```
        "theta_list" : [Theta list e.g, "1", "0.2", "0.1","0.05","0.02","0.01"],
        "num_of_structures" : [Number of structure to generate, e.g. 1000],
        "max_iter_per_theta" : [Max Iterations per job, e.g. 10],
        "violation_cutoff" : [Violation Cutoff, e.g. 0.05]
        "chr_occupancy" : [Chromosome Occupancy, e.g. 0.2]
        "nucleus_radius" : [Nucleus Radius (nm), e.g. 5000.0]
    },
    "system" : {
        "max_core" : [Maximum number of cores in a single node],
        "max_memMB" : [Maximum size of mem(MB) in a single node],
        "default_core" : [Default number of cores],
        "default_memMB" : [Default size of mem(MB)]
    }
}
```

2. `$PROJECT_DIR/runPGS.sh`

```
python $PGS_DIRECTORY/pgs.py
    --input_config $PROJECT_DIR/input_config.json
    --run_mode [running platform]
    --nCores 300
    --memMb 800000
    --pyflow_dir $PROJECT_DIR
    --schedulerArgList  ["-q","qname","-l","walltime=100:00:00"]
```

RUN PGS

User can execute PGS under the project/working directory with the following command.

```
$ sh runPgs.sh
```

CHAPTER 2

Expected Results

A successful PGS execution will generate a `result/` folder which composed of 4 subfolders:

- `probMat/` : contains the input hdf5 TAD-TAD probability matrix (if option 1 or 2 is used).
- `actDist/` : contains intermediate files generated by A-step (restraint assignment step). A file is generated at each A/M iteration, and will be used in the subsequent M-step (model optimization).
- `structure/` : contains structure information files. One `.hms` file has snapshot coordinates of a model with different optimization steps depending on theta parameters.
- `report/` : contains preliminary analysis with some plots.

The following we describe results under `report/` folder.

heatmap

Contains genome-wide heat map plots of contact probabilities, calculated from both the model population and experiment data (input matrix). A figure showing density scatter plots (log-log scale) as a comparison between them is also provided.

intraMatrix

Contains heat map plots of contact probabilities for every chromosome, calculated from both the model population and experiment data (input matrix).

radialPlot

Contains scater plots of mean radial position of all TADs, organized by chromosomes in pdf files. The mean radial positions are also recorded in a text file `radialPlot_summary.txt`.

violations

Contains a text file that tells the average portion of violated restraints per model.

CHAPTER 3

Tools

Analysis tools to get results shown under the `report/` directory

PGS package provides tools for analyzing the final structure population (see *Alberlab API Reference*). The following are some examples of how to extract information from the structure-population.

First, identify which model you would like to see, e.g. result/structure/copy0.hms. You can check group names in the file using a simple bash command:

```
$ h5ls result/structure/copy0.hms
```

And something like this will appear:

```
0.01b                    Group
0.01a                    Group
0.01                     Group
0.05                     Group
0.1                      Group
0.2                      Group
1                        Group
genome                   Dataset {SCALAR}
idx                      Dataset {2320}
```

In that example, the "idx" has information of 2320 TADs and it saves iteration snapshots at theta = {1, .., 0.01, 0.01a, 0.01b}. Thus the final structure is in group "0.01b" (at theta level p=0.01 there are 3 A/M iteration cycles).

- **Getting the 3D coordinates of the genome**

```
import alab
hmsfile = 'result/structure/copy0.hms
problvl = '0.01a'
hms = alab.modelstructures(hmsfile, [problvl])
TADidx = hms.idx  #TADs information
xyz = hms[0].xyz #diploid set of coordinates
```

Now the user can use the coordinates, stored in `xyz`, to do any analysis. The TAD information with genomic location is stored in `` `TADidx` `` variable. In the following we provide some other usage of coordinates.

- **Getting the contact probability map** We show example lines on how to get contact probability maps under `result/report/heatmap` and `result/report/intraMatrix``

```
import alab

hmsfiledir = 'result/structure'
problvl = '0.01a'
nstruct = 1000
summary = alab.structuresummary(hmsfiledir, problvl, nstruct)
m = summary.getContactMap()
m.plot('heatmap.png',format='png',clip_max=1)
m.makeIntraMatrix('chr1').plot('chr1_heatmap.pdf',format='pdf',clip_
↪max=1)
```

- **Getting the radial position** Radial position of a TAD is calculated by the average radial positions across the structure population of that TAD. 0 marks the center of nucleus, and 1 marks nuclear envelope.

```
rp = summary.getBeadRadialPosition(beads=range(len(summary.idx)*2))
rp_mean = rp.mean(axis=1)
rp_hapmean = (rp_mean[:len(summary.idx)]+rp_mean[len(summary.idx):])/2
```

- **Getting PDB** Some users might wish to get the coordinates and radii in a PDB format, maybe for visualization purpose. Hence we provide some nice scripts under `tool/` directory. Simply execute the following shell command under `$PROJECT_DIR/`:

```
$ tools/hms_export.py result/structure/copy0.hms 0.01b copy0.pdb
```

The script takes 3 arguments (hmsfile, theta_group, and output_name), then a pdb file will be saved and it will look something like this:

```
ATOM      1  PAM A1  a   1     2899.1    58.6   855.0   218
ATOM      2  PAM A1  a   2     3029.7   286.1  1257.0   244
ATOM      3  PAM A1  a   3     2575.2   106.8  1117.7   202
....      .  ... ..  .   .     ......  ......  ......   ...
....      .  ... ..  .   .     ......  ......  ......   ...
ATOM   1214  QAM BX  w  65    -2206.8   183.8  2465.6   202
ATOM   1215  QAM BX  w  66    -2452.5   434.7  3049.5   238
```

Note:

- The 2nd column marks the TADs ids.
- PAM and QAM marks the short and long arms of a chromosome, respectively.
- CEN marks the centromere representative TAD.
- Chromosome homologue (4th column) is labeled as the chromosome name preceeded with A or B, e.g. A1 = the first homolog of chr1, BX = the 2nd homolog of chrX.
- The first half of coordinates belong to the first diploid copy, the second half contains the homologues.
- Chain name (5th column) is unique for each chromosome (e.g. chains "a" to "w" are for chr1 to chrX, respectively).
- The 6th column contains TADs order in a chromosome.

- Columns 6-8 record the 3D coordinates.
- Column 9 stores the TADs radii.
- In human genome 3D models, the nuclear radius is set to 5000 nm.

- **Getting the structure-population summary file** Instead of analyzing structures from many files, user can also aggregate the final structure population into a file and ignore the intermediate snapshots. Let the final population be in "001b" group, thus the following shell command should be run:

```
$ tools/hms_to_hss.py result/strcuture/ 1000 0.01b structures_001b.hss
```

This command outputs a summary file called "structures_001b.hss" which contains all coordinates of the last optimized structure population, their radii, TAD information, optimization scores, etc. At this point, if the user is not interested in the structures at intermediate steps, all `structure/copy*.hms` files can be deleted to release some disk space.

Warning: Check the content of the summary file (hss) first before deleting the *.hms files!

CHAPTER 4

Frequently Asked Questions

FAQ

We are happy to update this list with your questions, please write them on the GitHub page.

1. **What is PGS for?** It is a user-friendly software package to compute 3D genome structures from contact frequency data (e.g. Hi-C matrix). Since it generates a lot of structures (population), it is better to run it on HPC clusters. Be ready to give a generous amount of disk space (a typical intermediate structure file can be ~4MB. But once a structure summary file is created (~1.2 GB for 10,000 structures), the intermediate files can be deleted).
2. **Should I edit the text under "Optional argument list" of PGS helper?** Yes, you should. Replace `"qname"` with your queue on HPC, but please do not delete the quote marks there (you may also delete this option and its value if you usually do not need to specify it when you submit jobs). Replace `hh:mm:ss` with number of hours, minutes, and seconds you wish to limit the time for a job to run. You can also add additional options with similar syntax (place a pair of quotes for each new option and its value, separated by a comma, and keep the brackets as it is).
3. **How long should I expect the PGS to complete 1,000 structures?** It depends on the computing power you assign it to. A typical M-step for 2 x 2320 TADs can take around 45 minutes on our HPC cluster (2.6 GHz speed). It will also depend on the theta list you set (correspond to A/M iteration cycles). The lower theta value will give more restraints to optimize, thus the longer is an A/M cycle. If you have 1,000 cpus running for PGS, and there will be 10 A/M cycles, you might get the final population in ~8 hours.
4. **Some nodes of my computing clusters crashed and some of PGS jobs were terminated, what should I do?** No worries, PGS can resubmit the fail jobs for you automatically and continues without problems. If PGS is still running, you do not need to do anything, just wait.

Warning: Do not alter `pyflow.data/` during PGS run. It contains logs and workflow state information. Deleting this folder will cause PGS to run from the beggining of the workflow again.

5. **I accidentally killed the terminal where PGS is running, how should I proceed PGS?** No worries, just go

to the working directory and execute the PGS again using the exact same command (`sh runPgs.sh`). PGS is capable of tracking the last interruption and restarting the workflows from there without hasle (as long as the last workflow state recorded in `pyflow.data/` remains valid).

6. **PGS was terminated because of errors before creating any results, what should I do?** You can first check the log files created under `pyflow.data/logs/` and try to fix that problems. However, in most cases the failures come from the input files, e.g. the matrix file or TAD file. Here are some points to check while fixing the error(s):
   - Make sure all formating rules are met.
   - The Hi-C matrix should represent a complete genome (include gaps so it is continues) in uniformly-sized bins.
   - TADs representation must be continues (include gaps and centromeres and label them as "gap" and "CEN", respectively. Every chromosome has one "CEN"). Of course, the size of TADs can differ.
   - All numeric must be integers (Hi-C counts, TAD genomic loci, etc.) except probability values (floating number from 0 to 1).
7. **What is TAD and how to get it for PGS run?** TAD (Topologically Associating Domain) is a continuous genomic region within which interact relatively frequently, whereas interactions across a TAD boundary occur relatively infrequently. Depending on the genome, their size can vary from tens of kb to a few Mb. We think this is a good chromosomal unit for 3D models. There are many TAD calling algorithm out there you can use. Ours is pretty much simple and quick, it's called TopDom and can be downloaded here.
8. **How many structures do I need to generate?** We think it will depend on your analysis, but in general the more the better. The radial position of chromosomes or TADs, for example, are relatively stable so that hundreds to a thousand of structures are good enough. To reproduce the Hi-C map that comes from million of cells, we need around 10,000 structures. Beyond that we may gain marginal increase of correlations but computational costly. If the object of study, e.g. higher order interactions, are rare events like ~1%, consider getting at least 10,000 structures.
9. **What is `probMat.hdf5.hmat` under `result/probMat/` folder?** It is a TAD-TAD probability matrix obtained from your raw matrix or converted from your TAD-TAD probability matrix text file. This matrix can be used to get a replica population: copy it to a new directory, run the PGS-helper and choose option 3.
10. **Are the messages on screen while PGS is running saved somewhere?** Yes, the stdout log messages are accumulated in your working directory under `pyflow.data/logs/pyflow_log.txt`.
11. **What is the `pyflow.data/logs/pyflow_tasks_stdout_log.txt` for?** It contains detail information of all specific running jobs, i.e. processing matrix and modeling report (timing and scoring for all structures at all A/M cycles). For instance, you can search for "copy0.hms" in the log file and see how it performs from initial to final stages.
12. **Any reference for the PGS?**
    - Kalhor *et al.* Genome architectures revealed by tethered chromosome conformation capture and population-based modeling. *Nat. Biotechnol.* **30**, 90-98 (2012).
    - Tjong *et al.* Population-based 3D genome structure analysis reveals driving forces in spatial genome organizations. *PNAS* **113**, E1663-E1672 (2016).
    - Coming soon....

CHAPTER 5

Alberlab API Reference

alab

class alab.**contactmatrix**(*filename, genome=None, resolution=None, usechr=['#', 'X']*)
A flexible matrix instant that supports various methods for processing HiC contacts

Parameters

- **filename** (*str*) – matrix file stored in hdf5 format or an integer for the matrix size to initialize an empty matrix instance
- **genome** (*str*) – genome e.g.'hg19','mm9'
- **resolution** (*int*) – the resolution for the hic matrix e.g. 100000
- **usechr** (*list*) – containing the chromosomes used for generating the matrix

matrix
numpy 2d array – storing all infor for the hic contact matrix

idx
numpy structure array – matrix index

genome
str – the genome

resolution
int – resolution for the contact matrix

applyed(*method*)

assignDomain(*domain*, *pattern=''*)
Load Domain information

Parameters

- **domain** (*alab.files.bedgraph instance*) – bedgraph for domain definition
- **pattern** (*str*) – a string use to filter the flags in the bedgraph

buildindex(***kwargs*)

columnsum()

fmaxScaling(*fmax*, *force=False*)
use fmax to generate probability matrix for uniform fmax, simply divide the matrix by fmax and clip to 1 for neighbouring contact fmax P[i,j] = F[i,j]/min(fmax[i],fmax[j])

fmaximization(***kwargs*)

getDomainMatrix(*domainChrom*, *domainStartPos*, *domainEndPos*, *rowmask*, *minSize=1*, *maxSize=None*)
Return a submatrix defined by domainChrom, domainStartPos, domainEndPos

Parameters

- **domainChrom** (*str*) – domain chromosome e.g. 'chr1'
- **domainStartPos** (*int*) – start position e.g. 0
- **domainEndPos** (*int*) – end position e.g. 700000
- **minSize** (*int, > 0*) – min domain size
- **maxSize** (*int, optional*) – max domain size, in bins if the domain is larger than a given number of bins, this function will return None

getICP(*index*)
return inter-chromosomal proportion of a given bin index

getfmax(*method='UF'*, *minSize=1*, *maxSize=2000*, *removeZero=False*, *boxplotTrim=False*, *offdiag=1*, *target='median'*)
calculate fmax based on different methods

Parameters

- **method** (*str*) – NM #neighbouring max UF #uniform fmax
- **target** (*str*) – 'mean'/'median'

identifyInterOutliersCutoff(*N=100*)
Identify interchromosome outliers' cutoff Do an N round random choice as the original contact freq distribution and estimate the sample std for every contact freq If the sample std is larger than half of the frequency (contact #), lable this contact frequency as spourious cutoff is set to the number that first consecutive 2 non-spourious frequency from the right side (scan from high frequency to low)

iterativeFmaxScaling(*domainAverageContacts=23.2*, *tol=0.01*)
Automatic fmax scaling to get domain level matrix and match the rowsum average domain level matrix to domainAverageContacts

krnorm(*mask=None*, *force=False*, ***kwargs*)
using krnorm balacing the matrix (overwriting the matrix!)

Parameters

- **mask** (*list/array*) – mask is a 1-D vector with the same length as the matrix where 1s specify the row/column to be ignored or a 1-D vector specifing the indexes of row/column to be ignored if no mask is given, row/column with rowsum==0 will be automatically detected and ignored
- **large_mem** (*bool*) – when large_mem is set to 1, matrix product is calculated using small chunks, but this will slowdown the process a little bit.

makeDomainLevelMatrix(*method='topmean'*, *top=10*, *removeOutlier=True*)
Use domain INFO to generate Domain level matrix

Parameters

- **method** (*str*) – "topmean" or "median"
- **top** (*int 0<top<100*) – the top percentage to calculate the mean, top=10 means top 10% of the subdomain matrix
- **removeOutlier** (*bool*) – option to remove outlier using 1.5IQR

makeIntraMatrix(*chrom*)
substract a chromsome matrix given a chromsome name

Parameters **chrom** (*str, chromosome name e.g 'chr1'*) –

plot(*figurename*, *log=False*, ***kwargs*)
plot the matrix heat map

Parameters

- **figurename** (*str*) –
- **log** (*bool*) – if True, plot the log scale of the matrix if False, plot the original matrix
- **clip_max** (*float*) –
- **clip_min** (*float*) – 2 options that will clip the matrix to certain value
- **cmap** (*matplotlib.cm instance*) – color map of the matrix
- **label** (*str*) – label of the figure

plotSum(*figurename*, *outlier=False*, *line=None*, ***kwargs*)
Print the rowsum frequency histogram

Parameters

- **figurename** (*string*) – Name of the plot
- **outlier** (*bool*) – option to select plotting the outlier line, only functioning if 'line' parameter is set to None
- **line** (*float/array/list*) – draw vertical lines at a list of positions

plotZeroCount(*figurename*, ***kwargs*)

range(*chrom*)
return the index range for a give chromsome

removeDiagonal(*force=False*)

removePoorRegions(*cutoff=1*, *usepvalue=0.1*, *force=False*)
Removes "cutoff" percent of bins with least counts

Parameters

- **cutoff** (*int, 0<cutoff<100*) – Percent of lowest-counts bins to be removed
- **usepvalue** (*float, 0<usepvalue<1*) – use this pvalue as correlation cutoff to remove bins bins whose pvalue greater than this cutoff will be removed

rowsum()

save(*filename*)
Save the matrix along with information in hdf5 file

scale(*cellaverage=1*)
Scale matrix so that average of cells is the given value. By default, the rowsum will be the number of rows/columns

smoothGenomeWideHighValue(*w=3*, *s=3*, *p=3*, *z=5*, *force=False*)
Use power law smoothing function to smooth high spikes in chromosomes blocks

Parameters

- **w** (*int*) – the window size, the smoothing is computed using target +/- w
- **s** (*int*) – weight of the location deviation
- **p** (*int*) – power of the location deviation
- **z** (*int*) – range of standard deviation to set cutoff

smoothInterContactByCutoff(*cutoff*, *w=3*, *s=3*, *p=3*, *force=False*)
given the cutoff, run a power law smoothing for the interchromosome matrix for contacts > cutoff

vcnorm(*iterations=1*, *mask=None*, *force=False*)

class alab.**tadmodel**(*probfile*, *nucleusRadius=5000.0*, *chromosomeOccupancy=0.2*, *contactRange=1*, *level=None*)
A wrapper to do IMP modeling in TAD level beads

Parameters

- **probfile** (*alab.matrix.contactmatrix instant*) – probability matrix at TAD level, alab.matrix.contactmatrix hdf5 format is required
- **nucleusRadius** (*float*) – radius of nucleus, default 5000(nm)
- **contactRange** (*int*) – folds for surface-surface contact coefficient
- **level** (*loglevel,*) – default:None will record everything during caculation debug, info, warning, error, critical are supported

CondenseChromosome(*rrange=0.5*)
Collapse chains around centromere beads

Parameters **rrange** (*float*) – scale parameter in [0,1] for the radius limit

SimulatedAnnealing(*hot*, *cold*, *nc=10*, *nstep=500*)
perform a cycle of simulated annealing from hot to cold

SimulatedAnnealing_Scored(*hot*, *cold*, *nc=10*, *nstep=500*, *lowscore=10*)
perform a cycle of simulated annealing but stop if reach low score

cache_coordinates()

cgstep(*step*, *silent=False*)
perform conjugate gradient on model using scoring function sf

evaluateRestraints(*restraintset*, *tolerance=0.05*)

mdstep(*t*, *step*, *gamma=0.1*, *silent=False*)

mdstep_withChromosomeTerritory(*t*, *step*)
perform an mdstep with chromosome terriory restraint

Parameters

- **t** (*int*) – temperature
- **step** (*int*) – optimization steps

saveCoordinates(*filename*, *prefix*)

savepym(*filename*)

savepym_withChromosome(*filename*, *s=1*, *v=1*)

set_consecutiveBeads (*lowprob=0.1*)

set_contactRestraints (*actdist*, *kspring=1*)

set_coordinates (*coordinates=None*)

set_excludedVolume (*ksping=1*, *slack=10*)

set_fmaxRestraints (*kspring=1*)

set_nucleusEnvelope (*kspring*)

shrinkingOptimization (*drange*, *shrinkScore*, *minscore*, *interScale*)

updateScoringFunction (*restraintset=None*)

class alab.**modelstructures** (*filename*, *usegrp*)
Instance manipulating hdf5 packed model structures take .hms file generated by tad modeling

Parameters

- **filename** (*str*) – model result file *.hms
- **usegrp** (*str*) – group label array, usually assigned with probability prefix like ['1','0.2']

class alab.**structuresummary** (*target*, *usegrp=None*, *nstruct=10000*, *pid=10*, ***kwargs*)
This class offers a series of methods to study the model structure populations.

Parameters

- **target** (*str*) – the output directory for population structures, containing copy*.hms files or can be seen as summary file *.hss
- **usegrp** (*str*) – the probablility key used in modeling, e.g. p005j
- **nstruct** (*int*) – number of structures to read

findBinIndex (*chrom*, *start*, *end*)
To find bead indexes given a chromosome region

Parameters

- **chrom** (*str*) – chromosome, should match the .idx representation
- **start,end** (*int*) – location range

Returns

Return type Bin indexes array, or None if there is no valid ones

getABCopyMeanBeadRadialPosition (*nucleusRadius=5000.0*)
Calculate mean radial position for every bead in structures, and differentiate diploid copy into A/B by inner or outer radial position

Parameters **nucleusRadius** (*float*) – radius of nucleus, default 5000(nm)

Returns **Two 1-D array**

Return type mean radian position for all beads, and first array contains the inner bead in the diploid genome.

getAveragePairwiseDistance (*form='list'*)
Calculate pairwise distance mean for each pair of beads in the structure population

Parameters **form** (*str*) – the return form of the function 'list' return the list form 'matrix' return the matrix form

getBeadRadialPosition (*beads*, *nucleusRadius=5000.0*)
Calculate radial position for every bead in the input list beads

Parameters

- **beads** (*array-like*) – list of all beads to calculate
- **nucleusRadius** (*float*) – radius of nucleus, default 5000(nm)

Returns **M*N matrix** – M = len(beads) N = number of structures in population

Return type radial position for all beads in the input and all structures in population

getChromosomeRadialPosition (*chrom*, *nucleusRadius=5000.0*)
Calculate radial position for the chrom

Parameters

- **chrom** (*str*) – the chromosome to calculate
- **nucleusRadius** (*float*) – radius of nucleus, default 5000(nm)

Returns **2N*1 vector**

Return type radial position for the chromosome in all structures in population

getContactMap (*contactRange=1*)
Return contact matrix format contact heatmap

getPairDistance (*bead1*, *bead2*)
Calculate pairwise distance across all structures

Parameters **bead1,bead2** (*int*) – two bead in a pair, input to calculate distance

Returns

Return type numpy array that has distance for the bead pair.

plotRadialPosition (*figurename*, *chrom*, *format='pdf'*, *color='dodgerblue'*, *nucleusRadius=5000.0*)
Plot Radial Position of beads for given chromosome

Parameters

- **figurename** (*str*) – name of the figure
- **chrom** (*str*) – given chromosome name
- **color** (*str*) – given the color for ploting

save (*filename*)
save all info into disk

totalRestraints
returns: **numpy array** :rtype: all restraints for each structure

totalViolations
returns: **numpy array** :rtype: all violations for each structure

violationPercentage
returns: **numpy array** :rtype: violation percentage for each structure

alab.files

class alab.files.**modelgroup** (*grouphandler*, *genome*, *idx*)

CHAPTER 6

Miscellaneous

Organization of the .hmat file

PGS saves a probability matrix in a binary hdf5 file with extension hmat. In the following we describe the file format. The file can be operated using alab api (see *Alberlab API Reference*)

- matrix (n * n numpy 2D matrix storing the hic contacts)
- idx (n * 4 numpy struct array consist of [chrom, start, end, flag] fields)
- genome (cPickle serialized string)
- resolution (cPickle serialized string)
- applyedMethods (cPickle serialized string, storing all process done after creating the matrix)

Organization of the .hms file

The output of each modeling step is dumped in a file with .hms extension. One .hms file contains information of a structure that evolve from the highest to lowest probability levels in theta list. Usually, only the final model extracted from this file is used for further analysis. In the following we describe the file format. The file can be operated using alab api (see *Alberlab API Reference*)

- genome (cPickle serialized string)
- resolution (cPickle serialized string)
- idx (n * 4 numpy struct array consist of [chrom, start, end, flag] fields)
- <group name 1>
- <group name 2>
- ...

Group names are usually threshold values, each group consists the following components:

- xyz (n * 3 array, coordinates)
- r (n * 1 array, radii)
- log (cPickled serialized string, all processing logs)
- pym (serialized pym file content)

Organization of the .hss file

A structure summary file, .hss can be generated once all final structures are successfully generated. In the following we describe the file format. The file can be operated using alab api (see *Alberlab API Reference*)

- genome (cPickle serialized string)
- resolution (cPickle serialized string)
- idx (n * 4 numpy struct array consist of [chrom, start, end, flag] fields)
- usegrp (cPickle serialized string of threshold)
- nbead (cPickle serialized int, number of representative beads in haploid)
- nstruct (cPickle serialized int, number of total structures)
- radius (n * 1 array of bead radii)
- coordinates (nstruct * n * 3 array, coordinates for all structures)
- score (1 * nstruct arry of optimization score)
- consecutiveViolations
- contactViolations
- intraRestraints
- interRestrains

Activation distance file

We start the modeling from random configurations for each structure and assign contacts between TADs that occur in 100% of the population (see Methods). Since we increase the number of restraints gradually, we need to assign contacts to a set of structures in the population at every A/M cycles. To do so, we compute the cummulative histogram and determine the distance cutoff according to the probability of a TAD pair in contact. The files are saved under `result/actDist` (the first 4 columns are TAD1, TAD2, contact probability, distance_cutoff_to_activate_contact).

- genindex

Index

T

U

V