NetStruct_Hierarchy

Version 1.0

User Manual

February 2019

Gili Greenbaum

gili.greenbaum@gmail.com

Amir Rubin

amirubin87@gmail.com

https://github.com/amirubin87/NetStruct_Hierarchy

Contents

Introduction

`NetStruct_Hierarchy` is a program implementing a network-based approach for inference and analysis of population structure using genetic data. Details can be found here: TODO. This is the first version of `NetStruct_Hierarchy`, and thus may have many issues and bugs, and only limited functionality. All these will be addressed in forthcoming versions. This version can be run using java.

All you need is to have Java JRE installed on your computer (and Python 3 for the first prepossessing step).

The pipeline has two steps: (1) Building genetic similarity matrix from genotype data; (2) Constructing the population structure tree.

In the first step genotype data is used to construct an inter-individual genetic similarity matrix. In the second step, `NetStruct_Hierarchy` constructs a network, and applies in several iterations, described below, a community detection algorithm, in order to detect communities (dense subgraphs in the network, i.e. groups of individuals that are strongly related within the group vs. between the groups). In the first iteration of this step, the community detection process is performed on the full, weighted genetic similarity matrix. In case the a single community was found, `NetStruct_Hierarchy` removes edges below a certain threshold which is increased by a user defined parameter (starting from zero, all the way up to one), until a partition community emerges (at least two communities). In the next iteration, `NetStruct_Hierarchy` performs the same process on each of the communities detected separately.

If you run into any trouble, feel free to email us at amirubin87@gmail.com or gili.greenbaum@gmail.com

1 Step 1: Building genetic similarity matrix

In the first step genotype data is used to construct an inter-individual genetic similarity matrix.

1.1 Building the matrix

Given genetic data, the `NetStruct_Hierarchy_BuildMatrix.py`[1] script generates a similarity matrix between individuals. Note that it can be used in parallel, for example on HPC clusters, running on different sections of the input. However, in this manual we describe running the code as a single computation thread.

The scripts requires Python 3 [2]

The data is assumed to have in line i the genotypes of individual i, or (pivoted) to have in line i the information of loci i of all individuals.

The format of each line is a space separated list holding in each entry two comma separated alleles, or (binary mode) the digit 0, 1 or 2, which represents 00, 10 and 11 respectively.

So, for example, having 2 individuals with 3 loci requires a 2 lines input with 3 entries in each.

An example for such an input is:

```
A,A A,C A,A
C,G C,C G,G
```

1.1.1 Parameters

Mandatory parameters, in the following order:

- **inputFile**

  Path to file with genetics data as described above.

- **outputFolder**

  Path to output folder.

- **totalLoci**

  The number of loci in the input file.

- **totalIndividuals**

  The number of individuals in the input file.

[1] https://github.com/amirubin87/NetStruct_Hierarchy/blob/master/BuildMatrix/NetStruct_Hierarchy_BuildMatrix.py

[2] https://www.python.org/downloads/

- **allelesString**

  Comma separated list of symbols. Each symbol is a symbol of an allele in the input data.

- **alleleMissingValueChar**

  The character representing a missing value.

Non mandatory parameters, in the following order:

- **binaryMode** If true, input is 0,1 or 2. 0 is mapped to 00, 1 is mapped to 10, and 2 is mapped to 11.
- **pivoted** If true each line represents a loci. Alleles in this loci of all individuals are listed in each line. If false, each line represents an individual.
- **windowSize** Size of window to be processed.
- **windowIndex** Index of the window to be processes - will be read from the shuffeledFile
- **shuffeledFile** A path to a file generated by buildShuffledArray(n) where n is the number of snps in the inputFile.

1.1.2 Sample commands

In the below we assume the repository is copied to your local machine and that commands are executed from the BuildMatrix directory.

- **Run on small dummy (ACTG) input** Run on a file containing genetic data of 4 individuals using 3 snps.

```
python ./NetStruct_Hierarchy_BuildMatrix.py
./SampleInputGenes.txt ./sample/
3 4 A,B,C,D,T,G N
```

- **Run on small dummy (binary, pivoted) input** Run on a file containing binary pivoted genetic data of 3 individuals using 4 snps.

```
python ./NetStruct_Hierarchy_BuildMatrix.py
./SampleInputGenesBinary.txt ./sample/
4 3 notUsed NotUsed True True
```

- **Run on small dummy Arabidopsis data** Run on a small Arabidopsis data with 20 individuals and 10k snps in a binary format, pivoted.

```
python ./NetStruct_Hierarchy_BuildMatrix.py
./Sample_Arabidopsis_20_ind_10k_snps.tsv
./Sample_Arabidopsis/ 10000 20
notUsed NotUsed True True
```

1.1.3 Outputs

There are three outputs:

- The resulting matrix can be found under `./sample/Distances/`. It is an upper triangular matrix of weights between nodes. Line i corresponds to node i, where the first line is for node 0 (zero-based count). Each line is a space separated string of doubles, each double indicates the weight of the undirectional edge between node i (line) and j (entry in the line). The file should only contain the upper part of the "real" adjacency matrix, excluding the main diagonal.
- In case you had in the data any missing values, in the file prefixed with `"Counts"` you will have in line i and entry j the number of valid entries used to calculate the distance between individuals i and j. If all values are valid, this file will only include a single line indicating how many locis were used.
- Allele frequencies per locus are outputted to the `./sample/Frequencies/` folder.

2 Step 2: Constructing population structure tree

In this step we use the output from Step 1 to construct the population structure tree.

For a demo execution of `NetStruct_Hierarchy`, simply type

```
java -jar NetStruct_Hierarchy_v1.jar -demo
```

This will generate sample input and resulting outputs under "sample" directory.

Below is a description of the files and parameters required to execute `NetStruct_Hierarchy` on genomic data, and the outputs generated.

2.1 Data Preparation

`NetStruct_Hierarchy` requires JRE (Java runtime enviorment) 8 or higher. You can download it at http://www.oracle.com/technetwork/java/javase/downloads/jre8-downloads-2133155.html.

Several inputs are required:

- **Network file** - in one of two possible formats:
  - **Adjacency matrix** - the output of the process described in Chapter 1.

    Since genetic similarity is assumed to be symmetric, the file contains only the upper part (above the diagonal) of the matrix.

    Optionally, you can provide a genetic similarity matrix (provided it's in the same format, and with values between 0 and 1) from any other source.

    For instance, if we have three individuals with distances:
    0-1:0.5
    0-2:0.25
    1-2:0.3
    the input matrix file will be:

    ```
    0.5 0.25
    0.3
    ```

    Please see subsection 1.1 ("Building the matrix") for a supplementary tool to construct the matrix from raw genetic data
  - **List of edges** - a list of weighted edges (format is: node node weight).

    Please note that all numbers between 0 and $n-1$(number of nodes) must appear.

    For instance, if we have three individuals with distances:
    0-1:0.5
    0-2:0.25

1-2:0.3
the input list of edges file will be:

```
0 1 0.5
0 2 0.25
1 2 0.3
```

- **Nodes sample sites** - a txt/csv file containing in each line i the sample site of node i. The first line is for node 0 (zero-based count). Note that the number of lines here must match the number of nodes referenced in the network file: in case you are using a matrix, the nodes sample sites file should have one line more than the matrix file. If you supplied a list of edges - all nodes between 0 and $n-1$, where n is the number of lines in the nodes sample sites file.
  Example file for three individuals:

```
Africa
Africa
China
```

- Sample Sites - a txt/csv file containing in each line a list of comma separated sample sites (which match the values in the nodes sample sites file). In each line you can put several sample sites, which will be considered to be in the same "area" - so that in the output they will be listed sequentially for readability. This is non mandatory - you may put all sample sites in a single line.
  Examples for sample sites files:
  a. each in its own "area":

```
Africa
China
```

  b. both in the same area:

```
Africa,China
```

- (Non mandatory) list of individuals to exclude - a txt/csv file containing a comma separated list of nodes to exclude. Note that the first node index is 0.

2.2 Execution

Execution of `NetStruct_Hierarchy` requires the input files described above and is control by several user defined parameters - most of them have a default value which in most cases shouldn't be changed. Below we describe these parameters, and supply some sample commands for `NetStruct_Hierarchy` execution.

2.2.1 Parameters

Each parameter has a flag which is stated in brackets, used to set a value for it via the command line. For instance, executing the below will execute `NetStruct_Hierarchy` with minSizeOfCommToBrake = 3.

```
java -jar NetStruct_Hierarchy_v1.jar -minb 3
```

First we describe the mandatory parameters.

2.2.2 Mandatory Parameters

- **pathToRootOutputDir** (-pro)

  String. Path to the directory where the output will be written.

  This must be a path to a folder which does not exist, to make sure no older files are being overwritten. The output path will include some of the parameters used for easy tracking.

- **pathToMapNode2SampleSite** (-pmn)

  String. Path to the file containing in each line i the sample site of node i.

- **pathToSampleSites** (-pss)

  String. Path to the file containing in each line a list of comma separated sample sites (which match the values in @pathToMapNode2SampleSite). In each line you can put several sample sites, they will be considered to be in the same "area" - meaning that in the output they will be listed one after the other, when "areas" are separated by a '|'. (This option is non mandatory - you may put all sample sites in a single line.)

Also, one of the following two is mandatory:

- **pathToMatrixFile** (-pm)

  String. Path to the file containing a matrix of weights between nodes, with respect to the list of nodes in @pathToMapNode2SampleSite.

- **pathToEdgesFile** (-pe)

  String. Path to the the file containing a list of weighted edges (format is node node weight), with respect to the list of nodes in @pathToMapNode2SampleSite. ALL NUMBERS BETWEEN 0 AND n-1 SHOULD APPEAR! If the parameter given in @pathToMatrixFile is not a valid file, the @pathToEdgesFile will be used.

2.2.3 Non-Mandatory Parameters

As for non mandatory parameters, we also specify the default value for each.

- **pathToIndividulasToExclude** (-indtoex. Default - null.)

  String. Path to a file containing a comma separated list of nodes. Note that if a matrix of edges is given, the first node index is 0.

- **minSizeOfCommToBrake** (-minb. Default - 3.)

  Integer. The minimum amount of nodes in a community which will be divided to sub-communities.

- **minSizeOfCommToOutput** (-mino. Default - 3.)

  Integer. The minimum amount of nodes in a community which will be outputted. In most cases it would make sense that minSizeOfCommToBrake=minSizeOfCommToOutput

- **weighted** (-w. Default - true.)

  Boolean. When true - using the edges weights. Otherwise assign weight of 1 to any edge which passes the used threshold.

- **stepSize** (-ss. Default - 0.0001.)

  Double. The threshold used to select which edges to include in the community detection process is raised by this value until we divide the given input community.

2.2.4 Sample commands

The below commands assumes a successful run of `NetStruct_Hierarchy` using the `-demo` parameter, meaning that files are located under "./sample/". So, before using any of the sample commands, simply run:

```
java -jar NetStruct_Hierarchy_v1.jar -demo
```

In order to process your own data, change the appropriate parameters.

- **Basic execution**

```
java -jar NetStruct_Hierarchy_v1.jar
-pro ./sample/
-pm ./sample/DUMMY_All_chrome_M.txt
-pmn ./sample/DUMMY_indlist.txt
-pss ./sample/DUMMY_SampleSites.txt}
```

- **Default values**

```
java -jar NetStruct_Hierarchy_v1.jar
-pro ./sample/
-pm ./sample/DUMMY_All_chrome_M.txt
-pmn ./sample/DUMMY_indlist.txt
-pss ./sample/DUMMY_SampleSites.txt
-w true -ss 0.0001 -minb 3 -mino 3 -b 1.0
```

- **Big communities**

  Note that as we have no big communites in our data, most outputs are empty..

```
java -jar NetStruct_Hierarchy_v1.jar
-pro ./sample/
-pm ./sample/DUMMY_All_chrome_M.txt
-pmn ./sample/DUMMY_indlist.txt
-pss ./sample/DUMMY_SampleSites.txt
-w true -ss 0.0001 -minb 10 -mino 10 -b 1.0
```

- **Triplets**

  Note that we break communities of size 4 (`-minb 4`) and output communities of size 3 (`-mino 3`)

```
java -jar NetStruct_Hierarchy_v1.jar
-pro ./sample/ -pm ./sample/DUMMY_All_chrome_M.txt
-pmn ./sample/DUMMY_indlist.txt
-pss ./sample/DUMMY_SampleSites.txt
-w true -ss 0.0001 -minb 4 -mino 3 -b 1.0
```

- **No granularity**

  Changing the step size (`-ss 1.0`) will results in a single level.

```
java -jar NetStruct_Hierarchy_v1.jar
-pro ./sample/
-pm ./sample/DUMMY_All_chrome_M.txt
-pmn ./sample/DUMMY_indlist.txt
-pss ./sample/DUMMY_SampleSites.txt
-w true -ss 1.0 -minb 4 -mino 3 -b 1.0
```

- **Process Arabidopsis data**

  Assuming you executed NetStruct_Hierarchy_BuildMatrix.py on the Arabidopsis data as listed above, you can run:

```
java -jar NetStruct_Hierarchy_v1.jar
-pro ./BuildMatrix/Sample_Arabidopsis/
-pm ./BuildMatrix/Sample_Arabidopsis/Distances/Matrix10000_0.csv
-pmn ./BuildMatrix/Sample_Arabidopsis/ind2SampleSite.txt
-pss ./BuildMatrix/Sample_Arabidopsis/SampleSites.txt
-minb 9 -mino 9
-ss 0.001 1.0
```

2.3 Outputs

- **1_CommAnalysis_...txt**

  All communities found. Each line represents a community. In each community you can find information about the community such as its size and a breakdown of its members according to the sample sites.

  For example, in the below line we have a community of size 3, all of its members were sampled in Africa, and it was split from the community in level 0, at entry 0, at line 0 (according to the "ParentLevel", "ParentEntry" and "ParentLine".

  `Size_03_Level_1_Entry_0_Line_0_ParentLevel_0_ParentEntry_0_ ParentLine_0_TH_0.0001_Modularity |Africa:3 America:0`

  Note that if the sample sites are listed in different lines, we get:

  `|Africa:3 | America:0`

- **2_Leafs_NoOverlap.txt**

  Leafs found in the full tree, based on the previous output. Dropped individuals are assigned to a singleton leaf.

- **2_Leafs_WithOverlap.txt**

  Leafs found in the full tree. Dropped individuals are assigned to a all leafs under the node in which they appear.

- **log_⟨matrixFileName⟩_HH-mm_d–MM-YYYY.log**

  A high-level log of the process - what step is performed.

The below will be outputted several times, once for each community processed by the algorithm.

- **Le_#_En_#_PaLe_#_PaEn_#_PaLi_#_TH_# _C.txt**

  A list of communities (lists of nodes) found by the algorithm in the given level('Le') and entry('En'), using the given threshold (as summarized in the 1_CommAnalysis_...txt file). In the first level (0) all nodes are in a single community.

- **Le_#_En_#_PaLe_#_PaEn_#_PaLi_#_TH_# _E.txt**

  A list of edges in the given level('Le') and entry('En'), passing the threshold (as summarized in the 1_CommAnalysis_...txt file). These are the edges included in the network.

3 Step 3: Visualizing PSTs on geographic maps

The outputs of Step 2 can be visualized onto geographic maps using a Mathematica notebook that draws maps from Wolfram Alpha. Instruction can be found in the notebook, which can be downloaded at https://github.com/amirubin87/NetStruct_Hierarchy (Map_plotting_tool.nb).

4 Referencing `NetStruct_Hierarchy`

Please use this reference for `NetStruct_Hierarchy`:

TODO