# NetStruct_Hierarchy

Version 1.0

User Manual

August 2019

Gili Greenbaum

gili.greenbaum@gmail.com

Amir Rubin

amirubin87@gmail.com

https://github.com/amirubin87/NetStruct_Hierarchy

# Contents

# 1   Introduction

`NetStruct_Hierarchy` is a program implementing a network-based approach for inference and analysis of population structure using genetic data. Details on the method can be found at https://doi.org/10.1101/518696.

Note that 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.

The `NetStruct_Hierarchy` pipeline involves two steps: (1) Construction on a between-individual genetic similarity matrix from genotype data, and (2) Construction of a population structure tree (PST) from the genetic-similarity network.

Running step 1 is covered in chapter 2, and requires Python 3 to be installed. Running step 2 is covered in chapter 3, and requires Java JRE to be installed. The geographic visualization tool is presented in chapter 4, and requires Mathematica to be installed.

## 1.1   Demo run

To get a better sense of `NetStruct_Hierarchy`, we encourage you to first run it in a demo mode. To get started quickly on a dummy data, simply run:

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

The above command uses the dummy data found in

```
.\NetStruct_Hierarchy\sample
```

and create the output in the same folder under

```
DUMMYStepSize_0.01_Beta_1.0_DynamicChoose_false_minSizeOfCommToBrake_3\
```

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

# 2 Step 1: Building genetic similarity matrix

In the first step genotype data is used to construct an inter-individual genetic similarity matrix. Here, we implement the genetic similarity measure presented in [2], which is a linearly frequency-weighted allele-sharing measure (see [1]).

It is possible to use NetStruct_Hierarchy with a different genetic similarity matrix. If you would like to compute your own genetic similarity matrix, you may skip this section and move to section 3.

## 2.1 Building the matrix

Given genotype data, the NetStruct_Hierarchy_BuildMatrix.py[1] script [2] can be used to generate a between-individual similarity matrix. The scripts requires Python 3 [3]

Note that this code can be used in parallel to decrease computation time, for example on HPC clusters, running on different sections of the input. This is particularly true for genomic datasets with many loci. However, in this manual we describe running the code as a single computation thread, and the user can wrap this code with the code needed to run on a cluster.

### 2.1.1 Input formats

The input data format of $n$ individuals with $l$ loci should have in line $i$ the $l$ genotypes of individual $i$ (i.e. $n$ rows with $l$ columns). This should be a text file (*.txt).

There are two possible formats for the input file:

- **Non-binary.** Each line is a space separated list holding in each entry two comma separated alphanumeric strings (without spaces or commas), where each string represents an allele.

- **Binary.** Each line is a space separated list holding in each entry a a single digit, 0, 1 or 2, with 0 and 2 representing homozygotes of the two alleles, and 1 representing the heterozygote. Note that the character '-' represent bad or missing value.

Non-binary format is applicable to a dataset with multiple alleles in each locus, while binary format is applicable only for datasets with bi-allelic loci.

For example, having 2 individuals with 3 bi-allelic loci, the input in non-binary format would be a text file with 2 lines and 3 columns, for example:

---

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

[2]In the supplements of [1] the file is named Supplemental_NetStruct_Hierarchy_BuildMatrix.py

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

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

*Example of input data in non-binary format*

In binary format, the same data would be represented as:

```
0 1 0
1 0 2
```

*Example of input data in binary format*

To convert VCF format to the binary format, see section below.

In order to run the code and construct the genetic similarity matrix, there are several mandatory parameters that need to be specified in the command line, as well as several optional parameters. See examples of command lines below.

### 2.1.2   Parameters

**Mandatory Parameters**
The parameters must be specified in the following order:

1. **inputFile** (string)

   Path to the input data file (genotypes file).

2. **outputFolder** (string)

   Path to output folder.

3. **totalLoci** (integer)

   The number of loci in the input file.

4. **totalIndividuals** (integer)

   The number of individuals in the input file.

5. **binaryMode** (True/False) True = input is in binary format. False = input is in non-binary format.

   **Mandatory Parameters only when binaryMode=False**

6. **allelesString** (comma-separated list of strings)

   Comma separated list of symbols. Each symbol is a symbol of an allele in the input data (e.g. A,T,G,C).

   In binary mode, this is set by default to 0,1,2 (no need to set it).

7. **alleleMissingValueChar** (character)

   A character(not a string) representing a missing value.

   In binary mode, this is set by default to '-' (no need to set it).

### 2.1.3    Running the command line

Before running the command line, you need to download the `NetStruct_Hierarchy` repository from GitHub. To use the code you need to run the `NetStruct_Hierarchy_BuildMatrix.py` file with python, and specify the parameters.

Examples:

- **Running on an input file in non-binary format (ACTG)**. The input data (in the file `SampleInputGenes.txt`) has 4 individuals and 3 loci (SNPs). Missing data is represented by the character "N". Note that the input data file and the `NetStruct_Hierarchy_BuildMatrix.py` file are in the same folder in this example.

  ```
  python ./NetStruct_Hierarchy_BuildMatrix.py
  ./SampleInputGenes.txt ./OutputNonBinary/
  3 4 False
  A,C,T,G,XYZ N
  ```

- **Running on an input file in binary format**. The input data (in the file `SampleInputGenesBinary.txt`) has 4 individuals and 3 loci (SNPs). Missing data is represented by the character "N". Note that no specification of characters is needed, "notUsed" is placed instead.

  ```
  python ./NetStruct_Hierarchy_BuildMatrix.py
  ./SampleInputGenesBinary.txt ./OutputBinary/
  3 4 True
  ```

### 2.1.4    Outputs

Running the command line generates three output files in the output folder:

1. **Genetic similarity matrix** (space separated text file ).

   The file's path is ./Distances/Matrix*.csv.

   The matrix is an upper triangular matrix of weights between individuals. Line $n$ corresponds to individual $n$, where the first line is for individual 0 (zero-based count). Each line is a space separated list of doubles. Each double indicates the weight of the edge between individual $i$ (line) and

individual $j$ (entry in the line). Since the genetic similarity matrix is symmetric, this file contains only the upper part of the matrix, excluding the main diagonal.

2. **Missing loci file**.

   The file's path is ./Distances/Counts*.csv.

   This file represents a matrix where in line $i$ and entry $j$ the value (integer) represents the number of non-missing loci used to compute the edge between the two individuals. If there are missing data in the input file, the file will have in line $i$ and entry $j$ the number of valid (non-missing) loci used to calculate the distance between individuals $i$ and $j$. If there are no missing data in the input file, this file will only include a single line indicating how many loci were used. This is an auxiliary file, used to calculate the distances.

3. **Allele frequencies file**

   Allele frequencies per locus are outputted to a file in sub-folder called `Frequencies` (a single file is found there). This is an auxiliary file, used to calculate the distances.

### 2.1.5 Converting VCF format to binary input format

Converting from VCF format ot the binary format of the input is possible with vcftools. From VCF format to a 012 matrix:

```
vcftools --gzvcf filename.vcf.gz --out filename --012
```

and then replacing the "-1" with "-":

```
sed 's/\-1/\-/g' filename.txt > filename.txt
```

# 3 Step 2: Constructing population structure tree (PST)

In this step we use the output from Step 1 (a between-individual genetic similarity matrix) to construct the population structure tree (PST).

## 3.1 Demo

For a demo execution of NetStruct_Hierarchy, simply type:

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

in a folder that contains the NetStruct_Hierarchy_v1.jar file[4]. This will generate a sample input, and the resulting outputs, under a folder called "sample".

## 3.2 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.

There are several mandatory and optional inputs required for running step 2:

- **Genetic similarity matrix file** (Mandatory). There are two possible formats for the network file.:

  1. **Adjacency matrix format** (space separated text file). This is the format of one of the output that generated by step 1, described in Chapter 2.

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

     The genetic similarity computed by the process in Chapter 2 follows equation 1 in [1]. Optionally, you can provide a genetic similarity matrix (provided it's in the same format: values between 0 and 1, and only upper part of the matrix) computed 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 a space separated text file with 2 lines:

     ```
     0.5 0.25
     0.3
     ```

     *Example of input file - adjacency matrix*

---

[4]In the supplements of [1] the file is named Supplemental_NetStruct_Hierarchy_v1.jar

2. **List of edges** (space separated text file). This format is a list of weighted edges, with 3 entries in each line: (1)node A (2) node B (3) weight of edge connecting nodes A and B.

   Please note that all numbers between 0 and $n - 1$(number of nodes) must appear (i.e., no isolated nodes).

   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
   ```
   *Example of input file - list of weighted edges(U V W)*

- **Individual's sample-site labels** (mandatory; txt/csv file). Since in many studies an individual sample is associated with a label indicating sampling location (or other categorical label), the outputs of `NetStruct_Hierarchy` also present results associated with these labels to facilitate with interpretation and post-analyses.

  This file should contain 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: for an Adjacency matrix, the nodes sample sites file should have one line more than the matrix file; for a list of edges with $n$ nodes, the sample-sites file should contain $n$ lines.
  Example file for four individuals (2 from Africa, 1 from China, and 1 from India):

  ```
  Africa
  Africa
  China
  India
  ```
  *Example of individuals sample-site labels file*

- **Sample sites** (mandatory; txt file). This file should contain a list of comma separated sample sites, matching the sites in the Individual's sample-site labels file.

  For the example above, the sample-sites file should be

```
Africa,China,India
```

*Example of a sample-sites file with a single region*

NetStruct_Hierarchy also supports labels assigned to "regions", which are groups of sample-site labels. In this case, each line will contain the sample-site labels of a single region.
For example (2 regions, "Africa" and "Asia"):

```
Africa
China,India
```

*Example of a sample-sites file with two regions*

**Important note** — the two files with sample-site information are mandatory, but need not contain any real information. If there are no sample-site labels associated with individuals, the individual's sample-site labels file should contain $n$ lines with some uninformative label (e.g., all lines with "A"):

```
A
A
A
A
```

*Example of a dummy individuals sample-site labels file*

and the sample sites file should contain a single entry with that label:

```
A
```

*Example of a dummy sample-sites file*

## 3.3 Execution

Construction of the PST requires 3 input files (genetic similarity matrix and 2 sample site files). It also requires 2 types of user defined parameters that are used to adjust the construction of the PST (see *Edge pruning* section in [1]): (1) The resolution of edge-pruning thresholds used to explore population structure at different hierarchical levels, and (2) The minimal size of a cluster. There are also several optional parameters. We first describe the parameters (section 3.3.1), then provide examples of command lines (section 3.3.2), and then explain how the user-defined parameters should be used (section 3.3.3).

### 3.3.1   Command line and parameters

The command line starts with

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

and continues with listing of parameter names and their values. The parameters can be entered in any order. The command line should be run from a folder that contains the `NetStruct_Hierarchy_v1.jar` file.
First we describe the mandatory parameters.

#### Mandatory Parameters

1. -pro (string).

   The 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 tractability.

2. -pmn (string).

   The path to the Individual's sample-site labels file.

3. -pss (string).

   The path to the sample sites file.

4. Also, one of the following two is mandatory:

   - -pm (string).
     The path to a file containing a genetic similarity matrix in the adjacency-matrix format (e.g. from the output of step 1, described in section 2)
   - -pe (string).
     The path to a file containing a genetic similarity matrix in the list-of-edge-weights format

#### Non-Mandatory Parameters

1. -minb (integer)

   The minimum amount of nodes in a community which will be divided to sub-communities. Must be equal or larger than 3. Default = 3.

2. -mino (integer)

   The minimum amount of nodes in a community which will be outputted. Must be equal or larger than 3. Default = 3.

3. -ss (double; real number between 0 and 1)

   The threshold increment used to select which edges to include in the community detection process. Default = 0.0001.

### 3.3.2 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**

  Running this command line will construct the PST using the default parameters (run as one line).

  ```
  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
  ```

- **Non-default values**

  Running this command line will construct the PST with a minimum cluster size of 5, and resolution of thresholds of 0.001 for edge pruning (run as one line).

  ```
  %   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
      -ss 0.001 -minb 5 -mino 5
  ```

### 3.3.3 How to use the user-defined parameters

In step 2 (PST construction), the 4 mandatory parameters are merely paths to files. The 3 non-mandatory parameters are user-defined, but conceptually they control two features: the resolution of thresholds for edge pruning (-ss parameter) and the minimum cluster size (-mino and -minb parameters).

The -ss parameter can be used to adjust the scale in which hierarchical levels are detected in the data. If the -ss parameter is low (i.e. high resolution), community detection is performed after removal of few edges, and therefore potentially more hierarchical levels may be detected. However, these resolution may be non-informative, as it may generate several uninterpretable splits until between two meaningful splits. If -ss value is two high, meaningful hierarchical

levels may not be detected. Therefore, we suggest testing several -ss values, over several orders of magnitudes (e.g. $10^{-6}$,$10^{-5}$,$10^{-4}$, and $10^{-3}$), and adjusting the -ss to reflect the desired resolution of hierarchy in the data. Note that the method is not very sensitive to the -ss parameter, unless extremely low or high values are used, and therefore the general topology and interpretation of the PST should not change when different -ss values are used, and changing -ss is essentially a fine-tuning of the outputs with respect ot the meta-data available or question of interest.

The -mino and -minb parameters control the minimal cluster size to include in the PST and the minimal cluster size for which to generate a split in the PST, respectively. For all reasonable uses of the method, -mino and -minb should have equal values, but we provide the user with the ability to define them independently. We therefore treat these two parameters as a single parameter. The minimal cluster size adjusts the scale in which clusters are included in the PST, with low minimal sizes resulting in PSTs with more clusters and more leaves. In general, the minimal cluster size should correspond to the smallest interpretable clustering of the data. For example, in [1], we analyzed the HGDP dataset, which has labeling for groups, the smallest being of size 5 (San). Therefore, it would make sense for the minimal cluster size to be of size 5, or close to it. The minimal cluster size has to be 3 or larger, since this is the minimal cluster size detectable by the Louvaine community-detection algorithm. As with the -ss parameter, it would be worth adjusting the minimal cluster size with respect to the meta-data and question of interest.

## 3.4   Output files

This section describes the format of the output files from construction of the PST.

- **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.

# 4   Step 3: Visualizing PSTs on geographic maps

The outputs of Step 2 can be visualized onto geographic maps using a Mathematica notebook (`Map_plotting_tool.nb`)[5] that draws maps from Wolfram Alpha. Instruction can be found in the notebook, which can be downloaded at https://github.com/amirubin87/NetStruct_Hierarchy

# References

[1] Gili Greenbaum, Amir Rubin, Alan R. Templeton, and Noah A. Rosenberg. Network-based hierarchical population structure analysis for large genomic datasets. *Genome Research*, 2019.

[2] Gili Greenbaum, Alan R Templeton, and Shirli Bar-David. Inference and analysis of population structure using genetic data and network theory. *Genetics*, 202(4):1299–1312, 2016.

---

[5]In the supplements of [1] the file is named `Supplemental_Map_plotting_tool.nb`