

# TreeScaper Manual

**Version 1.0**

**May 3, 2020**

Jeremy Ash<sup>2,4</sup>,  
Jeremy M. Brown<sup>2</sup>,  
David Morris<sup>2</sup>,  
Guifang Zhou<sup>2</sup>,  
Wen Huang<sup>1</sup>,  
Melissa Marchand<sup>3</sup>,  
Paul Van Dooren<sup>1</sup>,  
Kyle A. Gallivan<sup>3</sup>,  
and Jim Wilgenbusch<sup>5</sup>

<sup>1</sup>Department of Mathematical Engineering, ICTEAM,  
Université catholique de Louvain, Belgium,  
wen.huang@uclouvain.be

<sup>2</sup> Department of Biological Sciences and Museum of Natural Science,  
Louisiana State University, Baton Rouge, LA, USA

<sup>3</sup> Department of Mathematics,  
Florida State University, Tallahassee, FL, USA

<sup>4</sup> Current Address: Bioinformatics Research Center,  
North Carolina State University, Raleigh, NC, USA

<sup>5</sup> Minnesota Supercomputing Institute,  
University of Minnesota, Minneapolis, MN, USA

# TABLE OF CONTENTS

<b>1</b>	<b>INTRODUCTION</b>	<b>2</b>
<b>2</b>	<b>QUICKSTART TUTORIAL</b>	<b>5</b>
2.1	Getting Started . . . . .	5
2.1.1	Installation . . . . .	5
2.1.2	Example Tree Set . . . . .	5
2.1.3	Output . . . . .	5
2.1.4	Saving Figures . . . . .	5
2.1.5	Formatting Conventions . . . . .	6
2.2	Loading the Tree Set and Basic Computations . . . . .	6
2.3	Visualizing Tree Space . . . . .	6
2.4	Community Detection . . . . .	7
2.4.1	Community Detection on a Topological Network . . . . .	8
2.4.2	Community Detection on a Bipartition-Covariance Network . . . . .	9
<b>3</b>	<b>INSTALLATION</b>	<b>11</b>
<b>4</b>	<b>TREESCAPER GUI MENUS</b>	<b>12</b>
4.1	Trees and Bipartition Computation . . . . .	12
4.1.1	Load Tree Data . . . . .	12
4.1.2	Compute Bipartition Matrix . . . . .	13
4.1.3	Load/Compute Bipartition Covariances . . . . .	14
4.1.4	Compute Consensus Tree . . . . .	15
4.1.5	Load Distance/Coordinate Data or Compute Tree-to-Tree Distances . . . . .	16
4.1.6	Convert Tree-to-Tree Distances to Affinities . . . . .	17
4.2	Nonlinear Dimensionality Reduction and Dimension Estimation . . . . .	18
4.2.1	Nonlinear Dimensionality Reduction (NLDR) . . . . .	18
4.2.2	Dimension Estimation . . . . .	21
4.3	Community Detection . . . . .	23
4.3.1	Community Detection in Bipartition-Covariance Networks . . . . .	23
4.3.2	Community Detection in Topological Affinity Networks . . . . .	26
<b>5</b>	<b>COMMAND-LINE TREESCAPER</b>	<b>27</b>
<b>A</b>	<b>COMMUNITY DETECTION MODELS</b>	<b>32</b>
A.1	No Null Model . . . . .	32
A.2	Erdős-Rényi Model . . . . .	32
A.3	Configuration Null Model . . . . .	33
A.4	Constant Potts Model . . . . .	33
	<b>References</b>	<b>35</b>

# 1 INTRODUCTION

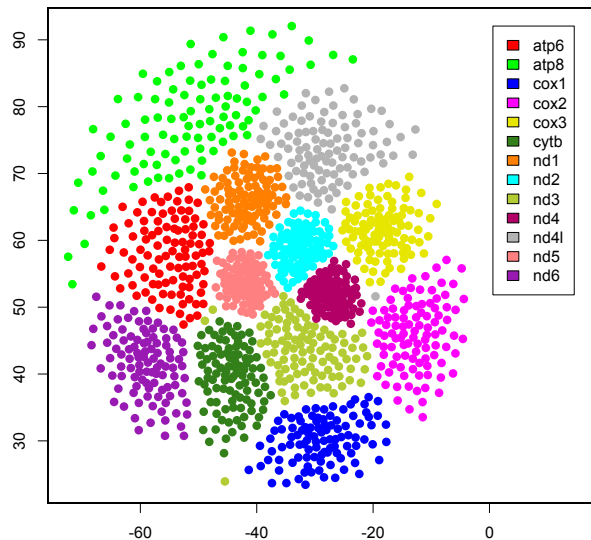
Phylogenetic trees are now routinely inferred from enormous genome-scale data sets, revealing extensive variation in phylogenetic signal both within and between individual genes. This variation may result from a wide range of biological phenomena, such as recombination, horizontal gene transfer, or hybridization. It may also indicate stochastic and/or systematic error. However, current approaches for summarizing the variation in a tree set typically condense it into point estimates, such as consensus trees, resulting in extensive loss of information.

We have written TreeScaper to provide a set of visual and quantitative tools for exploring and characterizing the full complement of phylogenetic information contained in a tree set. These tools can be broadly categorized into three types: (1) utilities for calculating basic information about topologies and bipartitions, (2) visualization of treespace in 2- or 3-dimensional space through non-linear dimensionality reduction (NLDR), and (3) detection and delineation of distinct communities of trees.

*Utilities* – Much of TreeScapers functionality requires calculating distances between trees, transforming distances into affinities, translating trees into their component bipartitions, and summarizing how these bipartitions are distributed across trees (i.e., their variances and covariances). However, this information can also be useful in its own right. Therefore, TreeScaper provides a set of built-in utilities to calculate a range of useful tree- and bipartition-related summaries. Once calculated, these values may be used for other tasks in TreeScaper or may be written to file for use in other applications.

*NLDR* – One way to visually explore tree sets is to plot a 2- or 3-dimensional representation of treespace using non-linear dimensionality reduction (NLDR; Fig. 1). This approach was first suggested for the visualization of phylogenetic trees by Amenta and Klingner, 2002 and Hillis et al., 2005, and recently extended by Wilgenbusch et al., 2017. The general idea behind NLDR is to find a lower dimensional representation of the relationships among trees that best preserves the true distances between them. TreeScaper implements several different methods for calculating tree-to-tree distances [e.g., Robinson-Foulds (RF) distances, matching distances, and subtree-prune-regraft (SPR) distances], several stress functions to assess how the original tree-to-tree distances should be optimally represented in lower dimensional space [e.g., Normalized stress, Kruskal-1 stress, nonlinear mapping (NLM) stress, and Curvilinear Components Analysis (CCA) stress], and several optimization algorithms for finding the best low-dimensional representation given a chosen stress function (e.g., Gauss-Seidel-Newton, stochastic gradient descent, and simulated annealing).

*Community Detection* – When tree sets are summarized by condensing them into a single point estimate, one of the key pieces of lost information is whether distinct phylogenetic “signals” exist in the set. Distinct signals can be created by a variety of biological processes like coalescence within a species, incomplete lineage sorting between species, horizontal gene transfer, hybridization, and migration. Artificial signals can also be created by systematic error during the process of phylogenetic inference. The process of detecting distinct signals in a tree set and assigning trees to one or more groups can be formalized in many different ways (e.g., [Gori et al., 2016; Lewitus and Morlon, 2016]). TreeScaper uses a graph-theoretic approach known as community detection. Roughly speaking, communities are parts of a graph with dense, positive connections between nodes within a community and sparse or negative connections between nodes in different communities. By formalizing the problem of detecting distinct phylogenetic signals as a community detection problem,



**Figure 1:** A 2-dimensional representation of treespace generated by non-linear dimensionality reduction (NLDR) for a set of 1,300 trees sampled from individual Bayesian analyses of 13 mitochondrial protein-coding genes in squamates [Castoe et al., 2009]. Each point represents a tree sampled from the posterior distribution of one gene. One hundred trees were sampled from each posterior distribution. Points are colored by gene. Plot created using R.

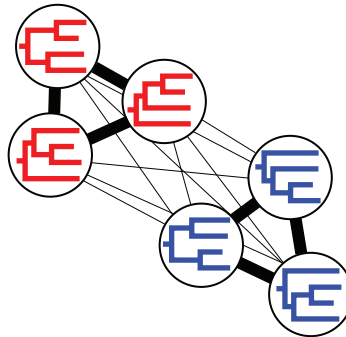
we can draw from a large body of existing work in graph theory.

TreeScaper can perform community detection on two distinct types of networks. In the first, nodes in the graph correspond to individual trees in the tree set and the edges between these nodes are weighted by the affinity between these trees (Fig. 2).

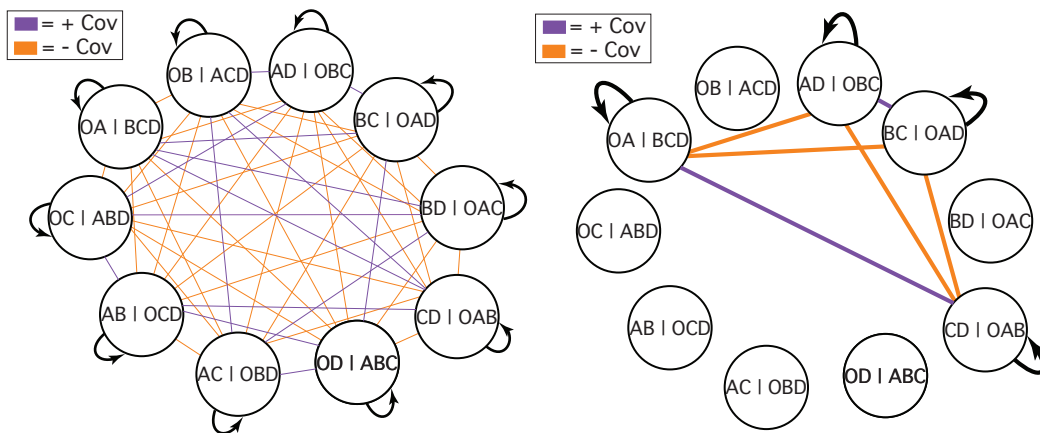
Affinity can be calculated in different ways, but it broadly corresponds to the converse of distance – a pair of trees separated by a small distance have high affinity, while a pair separated by a large distance have low affinity. Communities in these networks should intuitively correspond to sets of trees that are topologically similar to one another and topologically dissimilar to trees in other communities. Topological affinity networks have received some previous attention in attempts to define distinct phylogenetic signals [Stockham et al., 2002; Gori et al., 2016; Lewitus and Morlon, 2016].

The other type of network employed by TreeScaper uses nodes to represent individual bipartitions, with edge weights corresponding to the covariance in presence/absence of bipartition pairs across trees in the tree set (Fig. 3). When bipartitions are very common or very rare in the tree set, they tend to have weak covariances with all other bipartitions. However, if two bipartitions are present at intermediate frequencies and they are always found in the same trees or always found in different trees, they will have strong positive or strong negative covariances, respectively. Communities can be identified in bipartition covariance networks just like in topological affinity networks, with the distinction that bipartition covariance networks may contain negative edge weights. In this

case, communities should consist of sets of bipartitions that tend to have strong positive covariances, while bipartitions in separate communities should tend to have strong negative covariances (Fig. 3).



**Figure 2:** A cartoon topological affinity network. Circles are nodes in the network, each of which corresponds to one tree from a tree set. Edges represent the affinity (or similarity) between the trees, with thicker lines indicating greater affinity. Tree colors correspond to one intuitive definition of communities in this network.



**Figure 3:** Two example bipartition covariance networks for sets of unrooted, 5-taxon trees. The network on the left corresponds to a tree set with a uniform distribution of frequencies across all possible tree topologies. Some weak covariances exist in this network, because some pairs of bipartitions are mutually exclusive and are therefore found together less often than would be expected based on their frequencies alone. The network on the right corresponds to a tree set with only two topologies present at equal frequencies.

## 2 QUICKSTART TUTORIAL

### 2.1 Getting Started

#### 2.1.1 Installation

The Quickstart Tutorial uses a pre-compiled version of TreeScaper with a graphical user interface (GUI), which requires no special installation steps. Just download the appropriate version of the software from the TreeScaper website, open it, and get started. Pre-compiled versions of TreeScaper for Mac and Linux, as well as the files needed for this tutorial, are available here:

<https://github.com/whuang08/TreeScaper/releases>

#### Important Notes for MAC Version:

1. Open `TreeScaper.dmg` and move TreeScaper application into `MAC_TreeScaper_v1.0.0_Binary` folder.
2. The default security settings on current versions of Mac OS X do not allow users to open applications that have been downloaded outside the App Store with a simple double-click. Instead, you will need to hold down control and single-click on the TreeScaper app, then select “Open”. When asked if you are sure, select “Open” again.

#### 2.1.2 Example Tree Set

The tree set that will be used throughout this tutorial is titled “1000bp1L.nex”. The alignment used to generate this set of bootstrap trees was simulated such that half the sites were generated using one topology, while the other half were generated using another. “guide\_tree1.pdf” and “guide\_tree2.pdf” show the two topologies, corresponding to the first and second halves of the alignment, respectively. Bipartitions that conflict between these topologies are in color. A bootstrap analysis was then performed in Garli, to produce the 100 trees in this tree set. In this tutorial, we will analyze this tree set in TreeScaper to explore the two conflicting signals present in the data.

#### 2.1.3 Output

At any point, you can use the data menu in the bottom left pane to see what data structures you have saved in memory. Select any data structure and double click it to output or delete that data. Deleting data structures will allow you to control TreeScapers memory footprint. When outputting data structures, the name of the output file will be printed to the log.

#### 2.1.4 Saving Figures

At this time, TreeScaper cannot natively save figures to file, so graphs and plots are most easily preserved through screen capture. On Macs, the Grab utility makes capturing and saving individual windows easy. For more customized versions of non-linear dimensionality reduction (NLDR) projections, users can read NLDR coordinates automatically written by TreeScaper into other software (e.g., R) and adjust settings as they see fit.

### 2.1.5 Formatting Conventions

Throughout this manual, we format the text differently when referring to different TreeScaper components for clarity. *Tabs and labels* are italicized, **buttons** are bolded, “options” are in quotes, and dropdown menu options are underlined.

## 2.2 Loading the Tree Set and Basic Computations

Starting off in *Trees and Bipartition Computation*, Load Tree Data should already be selected. **Browse** and select the tree file. Select “weighted trees”, but do not select “rooted”. Click **Load all Trees** to load the tree set into memory.

From the dropdown menu, now select Compute Bipartition Matrix, then click the **Compute Bipartition-Tree Matrix** button. The log window will show a list of all the bipartitions in the tree set, along with their frequency. The data structure called “Bipartition Matrix now contains all trees and the bipartitions they contain, along with all the bipartition weights (i.e., branch lengths) for each tree. These data can be printed to a file in two different forms: as a list or as a matrix. In the list format, the first column indexes the unique bipartitions (which may be present in multiple trees), the second column indexes the trees in which those bipartitions were found, and the third column gives the bipartition weights.

Next, we will use the information in the bipartition-tree matrix to compute a consensus tree. To do so, select Compute Consensus Tree from the dropdown menu. Leaving the default parameters alone, click **Consensus Tree** and then **Plot**. While examining the plot, you can zoom in and out by scrolling up or down. Holding down the option key while clicking and grabbing the plot will allow you to move the tree. You can see in the consensus tree that there is low bootstrap support ( $< 0.7$ ) for several bipartitions in the consensus tree. It is impossible to determine, from this consensus tree alone, whether the low support is due to a lack of phylogenetic signal or strong support for a few conflicting topologies.

## 2.3 Visualizing Tree Space

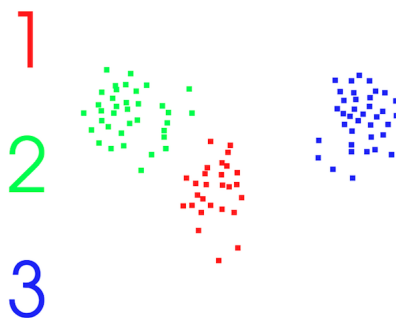
First, to get an idea of what tree space looks like for this tree set, we will visualize it in 3D using a non-linear dimensionality reduction (NLDR) projection method. In these projections, each point represents a tree in the tree set, and the distance between two points is an approximation of the actual tree-to-tree distance. From the main dropdown menu, select Load Distance/Coordinate Data or Compute Tree-to-Tree Distances. In this case, we will calculate the tree-to-tree distances from scratch, but you could also load distances from file if they were calculated previously. If you plan on conducting many analyses with the same large tree set, storing and loading the tree distances will save considerable time. From the Method dropdown select Weighted Robinson Foulds, then click **Distance** to compute a matrix of pairwise distances between trees in your tree set. This matrix should appear as a new data structure in the bottom left pane. Also select Unweighted Robinson Foulds and compute the distance. You will need to use this distance matrix later. We have now conducted all of the necessary preliminary calculations to conduct NLDR.

In the topmost tab menu, now select *NLDR and Dimension Estimation*. In the main dropdown menu under this tab, Nonlinear Dimension Reduction should already be selected. Make sure that

Weighted RF-distance is selected in the *Distance* dropdown, with CCA and Stochastic selected in *Method* and *Algorithm* drop downs, respectively. Change *Dimension* to 3. Make sure that Random is selected in the *Initial Projection* dropdown.

Next, click the **Plot Parameters** button. A new dialogue box should appear. Points should be selected in the dialogues dropdown menu. Set the “Number of Clusters” to 1, then click **OK**. 1 cluster should appear in the “Cluster Index” section. Check the “Step Size” box. Set the step size to 100. The single cluster should now include all 100 trees in the tree set (with indices 0–99). Under “Other parameters” change “point size” to 10, then click **Apply**, followed by **Close**.

Now, click the **Run NLDR** button. A dialogue box will open, prompting you to open a folder. This is the folder where output files from the NLDR analysis will be saved. Once the NLDR method is finished running, click **Plot Result**. A dialogue box with the plot of tree space should appear. Click and drag to rotate the 3D plot. Scroll up and down to zoom in and out. Hold down Shift+R, select a small number of points (1 – 5), then click the **Plot trees** button to examine each of these trees individually. By rotating the plot, you should see what looks like three distinct groups, or communities, of points. The NLDR projection suggests that these communities have lower tree-to-tree distances between trees in the same community and larger distances between trees in different communities. We will later confirm that the original tree-to-tree distances support these community designations by performing a community detection analysis on the tree sets topological affinity network. After rotation, your NLDR plot should look something like Figure 4. In this diagram, trees are colored by affinity community, which we will find in the next step of the tutorial. TreeScapers NLDR plots (e.g., Fig. 4) can be saved using screen capture (or the Grab utility on Mac). Customized versions of such plots can be created by reading raw NLDR coordinates into software such as R, which have more advanced plotting capabilities. See Section 4.1.2 for more details on NLDR plotting options and output files.



**Figure 4:** One rotation of the 3D NLDR projection outlined in the quickstart tutorial. Points are colored according to the output of topological affinity community detection analysis, which is described below.

## 2.4 Community Detection

Before using community detection to identify distinct phylogenetic signals, we need to construct networks using our tree set. We will then perform community detection on these networks to identify distinct topological signals and the sets of bipartitions that strongly conflict between them.



As mentioned in the Introduction, TreeScaper uses two types of networks: topological affinity and bipartition covariance networks. See above for more detail.

### 2.4.1 Community Detection on a Topological Network

First, we will construct a topological affinity network, where nodes represent individual trees in the tree set and edges between these nodes are weighted by the affinity between the trees. Since affinities are the converse of distances, we need to convert our previously calculated distances to affinities. In the topmost tab menu, go back to the *Trees and Bipartition Computation* tab. Select Load/Convert Tree-to-Tree Distances to Affinities from the main dropdown menu in this tab. Select Unweighted RF-distance from the *Distance in Memory* dropdown. Make sure the *Affinity Type* is set to Reciprocal then click **Affinity**. This will calculate the reciprocal of each distance in the distance matrix and use this as a measure of affinity, or similarity, between the trees.

Now select the *Community Detection* tab at the top and Community Detection on Affinities from main dropdown menu. Affinity-URF should be selected in the “Affinity in memory” dropdown. In the “Model Type” dropdown select Constant Potts Model, and in “Find Plateaus” select “Automatically”. Click **Community**. A dialogue box will open that asks you to open a directory. Output files from community detection will be written to this directory. A variety of specific information about the community detection analysis is also printed to the log. For now, we will ignore most of this information. The basic idea is to vary a tuning parameter ( $\lambda$ ) and see how the inferred community structure changes. The preferred community structure is the one that is most robust to changes in the tuning parameter. These naturally robust community definitions are referred to as plateaus (which will make more sense in a moment). We prefer the plateau with the largest “lower bound length” value. This plateau should represent some intrinsic community structure of the affinity matrix.

When community detection is finished, click **Plot** to generate a figure showing how various summaries of the inferred community structure change with varying values of  $\lambda$ . For now, we will focus on the “Labels for Communities” line and look for the largest flat region (a plateau!) where the community labels remain stable as  $\lambda$  varies. Note that plateaus frequently occur on the left and right ends of these plots, but they represent trivial community structures where all nodes are placed in a single community (the left end) or every node is assigned to its own unique community (the right end). In this example, the plateau of interest occurs when  $\lambda$  is between 0.6 and 0.9. See Section 4.3 of this manual and the Appendix A for more details on why TreeScaper uses this definition of intrinsic community structure. Pick a  $\lambda+$  value in the range of  $\lambda+$  values corresponding to the plateau (0.7, for example). Returning to the main window, select “Manually” for the “Find Plateaus” option. Set both the “From” and “To” values to the chosen  $\lambda+$  value. Click **Community**. The plateau community structure will be printed to the log pane. For now, we are primarily interested in which trees (i.e., nodes in the network) are assigned to each community.

Using the indices of trees assigned to different communities, you can create community-specific consensus trees or color points by community assignment in NLDR plots. Copy the node indices corresponding to a community (e.g., those numbers that follow “Community 1 includes nodes:” in the log) and create a separate index file for each community that has each number on its own line (see the format of “example\_index\_aff1.txt” for each affinity community or use the pre-made files, “example\_index\_aff1-3.txt”). Go back to Compute Consensus Tree under *Trees and*

*Bipartition Computation*, select index file in “Considered Trees”. Then, load an index file for the community of your choice. Click **Consensus Tree**, and then **Plot**. The consensus tree for each affinity community should provide a sense for its dominant phylogenetic signal. The files, “AffinityConsensusCommunity1-3.pdf”, show the affinity community consensus trees, although the ordering of the three trees is arbitrary. Conflicting bipartitions from the two topologies used to simulate the data are in color.

To color points in an NLDR plot by community, create a comma-separated list of the indices for that community. Return to the *NLDR and Dimension Estimation* tab and click the **Plot Parameters** button. Change the number of clusters to 3. In the box for each cluster under *Cluster Index* paste the comma-separated list of indices for each community. Press Tab after pasting each set of values, then click **Apply** and **Close**. Once back at the NLDR menu, click **Plot result**. The new plot should have the same structure as the previous NLDR plot, but with points colored as in Fig. 4.

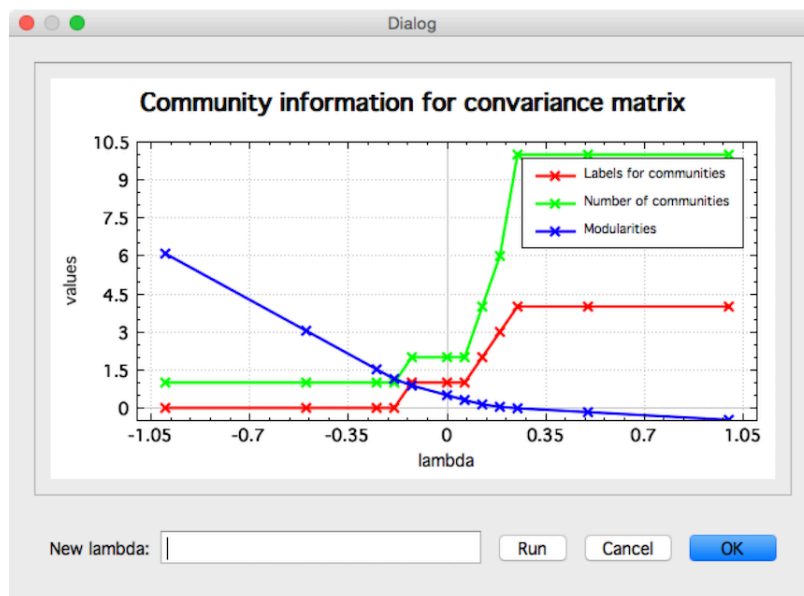
For extra practice, try redoing community detection with affinities using weighted RF distances. Does the preferred number of communities differ from the analysis using unweighted RF distances? How many large plateaus are found? After coloring communities in the NLDR plot also based on weighted RFs, do these results make sense?

## 2.4.2 Community Detection on a Bipartition-Covariance Network

Next, we will construct a bipartition covariance network. Go back through the steps of Section 2.2 to reload the tree set, with one important difference: *do not select the “weighted trees” option*. You should never use weighted trees to compute a covariance matrix, as the computed values will correspond to covariances in branch lengths rather than bipartition presence/absence. When you read the tree set back in without weights, all of the previous data structures will automatically be erased. As before, select Compute Bipartition Matrix from the main dropdown menu and click **Compute Bipartition-Tree Matrix**.

Now, select Load/Compute Bipartition Covariances from the main dropdown, then click **Covariance using Bipartition-Tree Matrix**. Alternatively, you have the option to load a previously computed covariance matrix at this stage. Once the covariance matrix has been created, select the *Community Detection* tab at the top and choose Community Detection on Covariances. The Covariance Matrix option should now appear in the “Covariance in Memory” dropdown. Set the “High Freq” to 0.95 and the “Low Freq” to 0.05. These settings will filter out high and low frequency bipartitions, because they are very difficult to assign to particular communities. In the “Model Type” dropdown select Constant Potts Model, and for “Find Plateaus” select “Automatically”. Click **Community**. The lambda values used and the largest plateaus are output to the log. The largest plateau is the plateau with the largest “lower bound length” values. This plateau indicates the intrinsic community structure of the covariance matrix.

When community detection is finished, click **Plot** (Fig. 5). As with community detection for affinity networks, this plot will give you an intuitive sense for the most stable community structure – the largest plateau on the “Labels for communities” line. Also as with affinity networks, plateaus frequently occur on the left and right ends of these plots, but they represent trivial community structures where all nodes are placed in a single community (the left end) or every node is assigned to its own unique community (the right end). For this example, there is 1 community in the left trivial plateau and 10 communities in the right trivial plateau.



**Figure 5:** Example output from automatic community detection on the bipartition covariance network discussed in the tutorial. Note the trivial community structures represented by the plateaus at 1 and 10 communities.

Pick a  $\lambda^+$  value in the range of  $\lambda^+$  values for the largest non-trivial plateau (0, for instance). In *Find Plateaus* select “Manually”. In  *$\lambda$  Fixed*, select  $\lambda^-$  and set the value to 0. Set both the *From* and *To* values to be the chosen  $\lambda^+$  value. Click **Community**. The plateau community structure will be printed to the log. While TreeScaper does not have built-in functionality to visualize bipartition networks and communities, you can examine the bipartition matrix itself to see which bipartitions are assigned to each community. See the “Compute Bipartition-Tree Matrix” menu in Section 4.3.1 for an explanation of how to read the bipartition matrix output.

The file “1000bp1L.comKey.out” shows the bipartitions assigned to each community, along with their frequencies. Recall that the dataset used to generate this tree set was simulated such that half the sites evolved on one topology and the other half evolved along another (see “guide\_tree1.pdf” and “guide\_tree2.pdf”, respectively, where conflicting bipartitions are shown in color). The file “1000bp1LcovarianceNetwork.pdf” contains a schematic of the bipartition-covariance network. Node numbers correspond to the numbered bipartitions in the guide trees. Blue and green edges indicate positive and negative covariances between bipartitions, respectively. Line weights correspond to the magnitude of the covariances. Colored circles circumscribe the nodes assigned to each of the detected communities. Each community contains the conflicting bipartitions from one of the two topologies used in the simulation. Thus, we have identified the conflicting bipartitions that define the two distinct topological signals underlying the sites in this alignment. In the affinity network, we also identified both of the original tree topologies and a third topology that seems to arise when bootstrapped alignments contain balanced mixtures of sites of the two types.

### 3 INSTALLATION

Executable versions of TreeScaper for Mac or Linux can be downloaded from

<https://github.com/whuang08/TreeScaper/releases>

Also available as separate downloads are files needed for the Quickstart tutorial at the beginning of this manual and the source code corresponding to the most recent release of TreeScaper. The most current (although not officially released) code is available from the corresponding GitHub repository:

<https://github.com/whuang08/TreeScaper>

The graphical, pre-compiled version of TreeScaper can be started by simply doubleclicking (the name will either be “TreeScaper” or “TreeScaper.app”).

**One Important Note for Macs:** The default security settings on current versions of Mac OSX do not allow users to open applications that have been downloaded outside the App Store with a simple double-click. Instead, you will need to hold down control and single-click on the TreeScaper app, then select “Open”. When asked if you are sure, select “Open” again.

Command-line versions of TreeScaper can be generated by following these instructions:

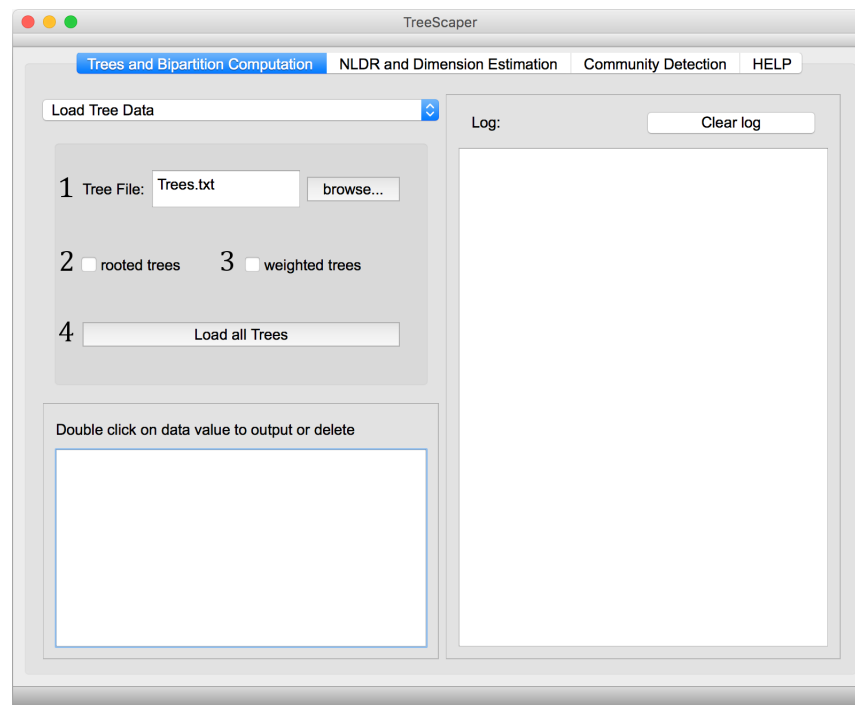
- (1) Download “clapack-3.2.1-CMAKE.tgz” from <http://www.netlib.org/clapack/> and unpack the archive.
- (2) If using a Mac, open the “makefile.inc.example” file in the clapack folder and change the platform from “\_LINUX” to “\_MAC”. If using Linux, leave the platform as is. Next, rename the file to “makefile.inc”.
- (3) Open a terminal, change directories (cd) into the clapack folder, then type “make” to compile cmake.
- (4) Open the Makefile in the folder containing TreeScaper code and simply replace “<PATH\_TO\_CLAPACK>” with the path to your clapack cmake installation.
- (5) Finally, open the wdef.h file in the TreeScaper folder. On the 2<sup>nd</sup> #define line, replace \_MAC with COMMAND\_LINE\_VERSION
- (6) Now, in Terminal type “make” to compile Treescaper.

## 4 TREESCAPER GUI MENUS

TreeScapers windows are arranged in a series of tabs, organized by the different types of analyses that TreeScaper can perform. Each tab contains a header at the top of its window that specifies its features. Every tab also contains a data window at the bottom left which shows the data structures currently available in memory. Each data structure may be deleted or output to file by double clicking and selecting the relevant option. The TreeScaper log is visible on the right side no matter which tab is selected. The log can be reset by clicking **Clear log**. All the example files mentioned below are included in the tutorial files available for download from the TreeScaper website.

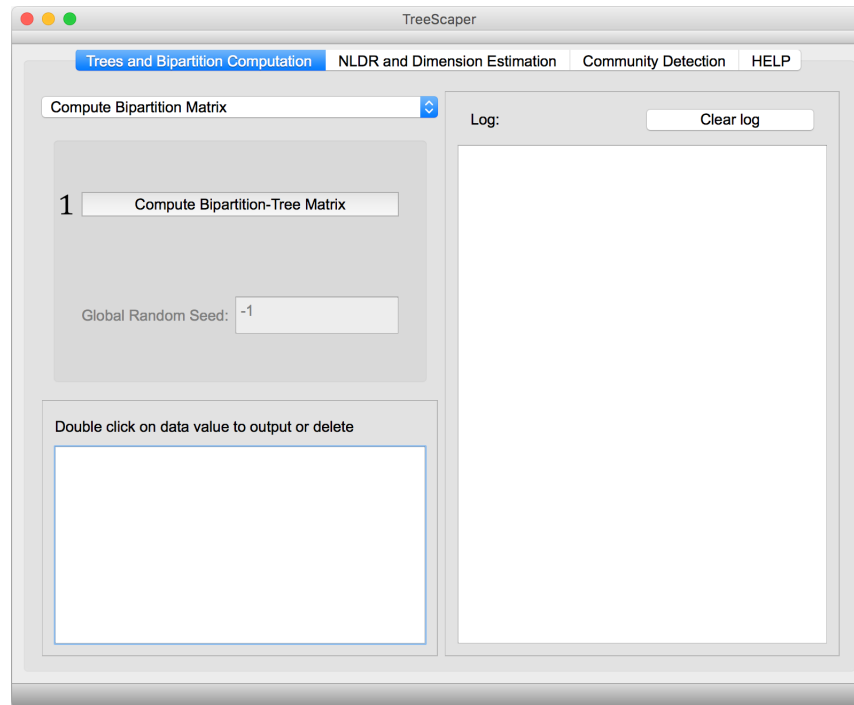
### 4.1 Trees and Bipartition Computation

#### 4.1.1 Load Tree Data



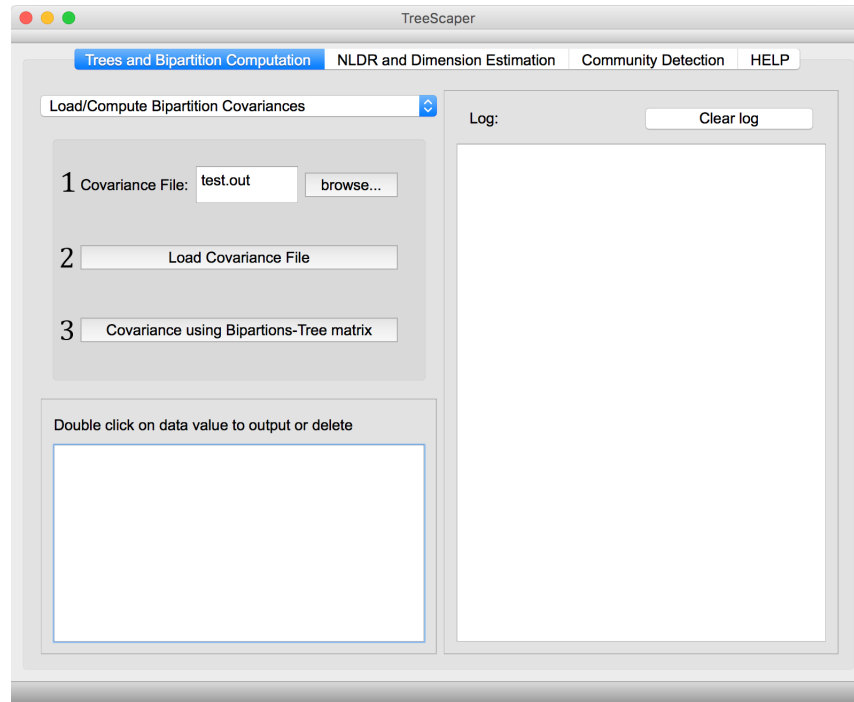
- 1- [**Tree File**] – Select the location of the tree set. The tree set file needs to be formatted as closely to “1000bp1L.nex” as possible. Make sure that there is a “translate” section in the header of the file. Comments in brackets are fine.
- 2- [**rooted trees**] – A checked box means the tree set contains rooted trees. An unchecked box means the tree set contains unrooted trees.
- 3- [**weighted trees**] – A checked box means the tree set contains trees with branch lengths. An unchecked box means the tree set contains trees without branch lengths.
- 4- [**Load all Trees**] – Load the tree set from the file specified in “Tree File” into memory.

### 4.1.2 Compute Bipartition Matrix



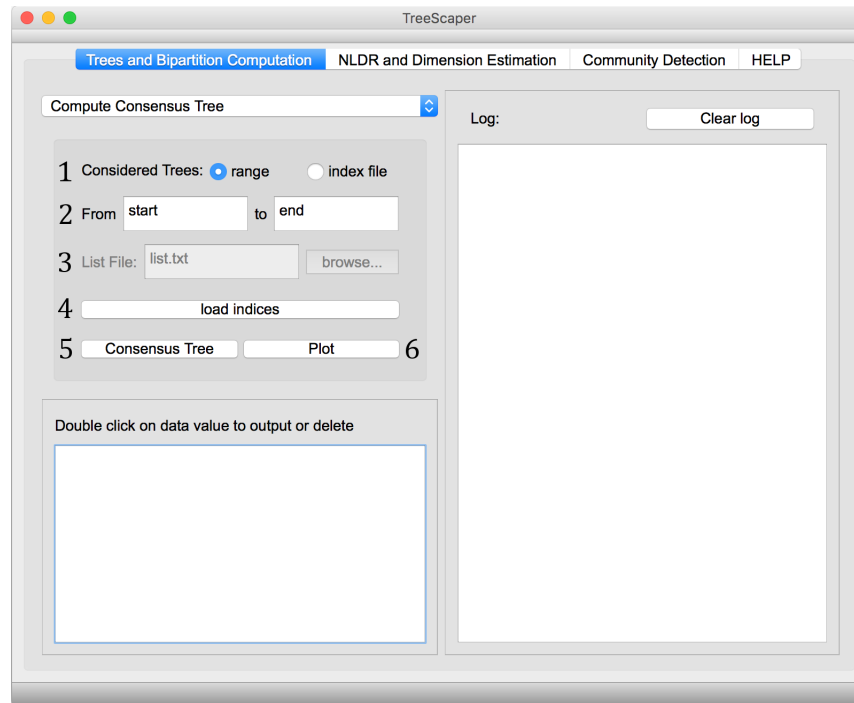
- 1- **[Computed Bipartition-Tree Matrix]** – Creates a list of all the bipartitions in the tree set, along with their number of occurrences, as well as a matrix of bipartition presence/absence across trees. The bipartitions are specified using a binary notation, where all taxa on each side of the bipartition have the same digit (0 or 1). The order of binary digits corresponds to the numbering of the taxa in the output taxa list. For example, the fifth digit in a bipartitions notation would correspond to the fifth taxon. For a six-taxon case (taxa A-F), 110000 would indicate a bipartition separating taxa A and B from taxa C-F.

### 4.1.3 Load/Compute Bipartition Covariances



- 1- **[Covariance File]** – Select the location of a covariance matrix (if previously computed). The matrix needs to be similar to the format of “1000bp1L\_Covariance Matrix.out” and will usually correspond to the output of a prior TreeScaper analysis. This is a lower triangle matrix where the rows and columns are labeled according to the bipartitions in the tree set, and cells give covariances in bipartition presence/absence across trees.
- 2- **[Load Covariance File]** – Load covariance matrix into memory (if selected).
- 3- **[Covariance using Bipartition-Tree matrix]** – Use the bipartition-tree matrix stored in memory to compute a covariance matrix. In general, DO NOT use weighted trees to compute a covariance matrix, because the computed values will correspond to covariances in branch lengths rather than bipartition presence/absence.

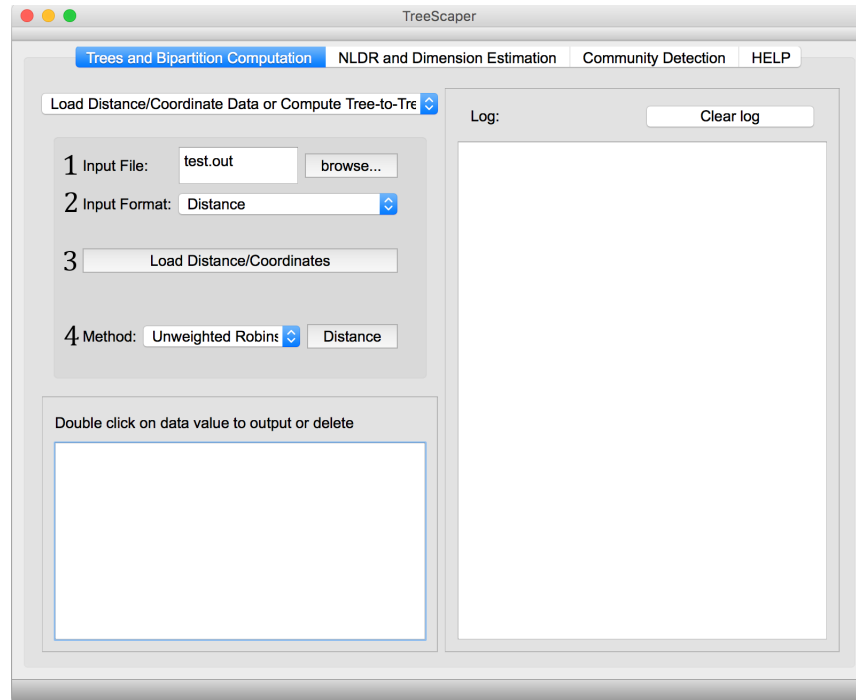
#### 4.1.4 Compute Consensus Tree



- 1- **[Considered Trees]** – Select the trees to be considered when computing a consensus tree. A manually defined range of tree indices can be provided by selecting “range”, or a series of indices can be specified in a list contained in a file by selecting “index file”.
- 2- **[range]** – The indices of a range of trees to consider when calculating a consensus tree. These values are needed if “range” is specified for the “Considered Trees”. By default, the range is the whole tree set, from “start” to “end”.
- 3- **[List file]** – Considers trees listed in an index file. Specify the location of the file in “List File”. The index file must be in the format of “indices.out”, with one tree index per line.
- 4- **[load indices]** – Loads the index (i.e., list) file into memory, if “index file” is selected for “Considered Trees”.
- 5- **[Consensus Tree]** – Compute the consensus tree of selected trees.
- 6- **[Plot]** – Plot the consensus tree in a separate window. Use the mouse scroll bar to zoom. COMMAND+click+drag to move the tree and zoom in at the desired location.



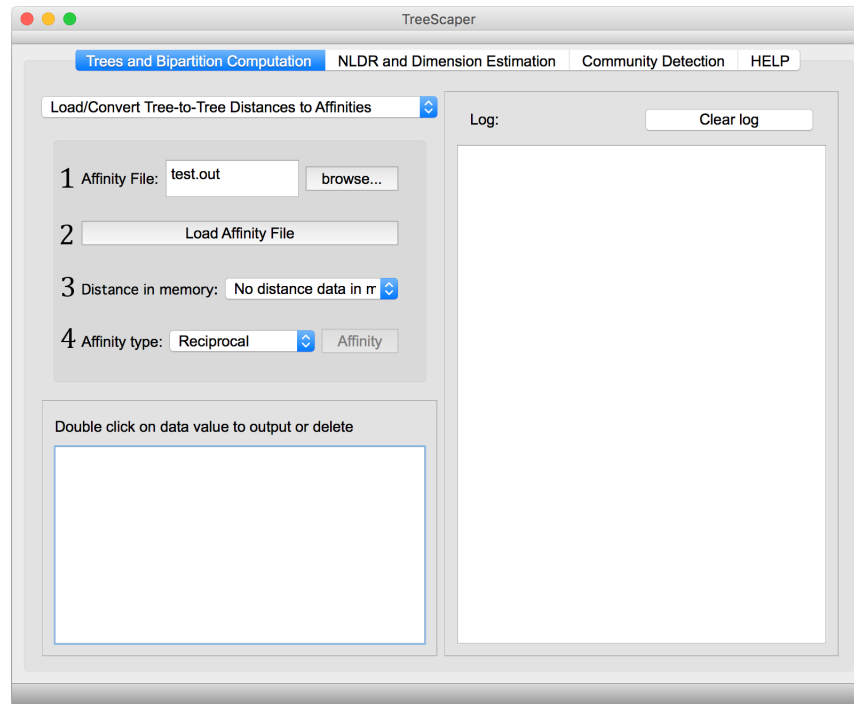
#### 4.1.5 Load Distance/Coordinate Data or Compute Tree-to-Tree Distances



The options in this pane allow tree-to-tree distances to either be computed from scratch (see 4 below) or to be loaded from file if already computed (see 1 - 3 below). If loaded from file, any distances may be used. If computed in TreeScaper, see the “Method” dropdown for a list of available distances.

- 1- **[Input File]** – Select the location of a file containing a distance matrix. The matrix needs to be similar to the format of “1000bp1L\_Unweighted RF-distance.out” – a lower triangle tree-to-tree distance matrix. Negative values are not allowed.
- 2- **[Input Format]** – Select whether the input file is formatted for distances (the vast majority of cases) or 2D/3D coordinates (when pre-computed).
- 3- **[Load Distance/Coordinates]** – Load the distance matrix or coordinates specified in the “Input File” into memory.
- 4- **[Method]** – If computing tree-to-tree distances from scratch, choose a distance. Options include Unweighted Robinson-Foulds, Weighted Robinson-Foulds, Matching, and SubtreePrune-Regraft (SPR) distances. As far as we know, TreeScaper is currently the only program available that can calculate all of these distances. Click “Distance” to compute the distance matrix.

#### 4.1.6 Convert Tree-to-Tree Distances to Affinities

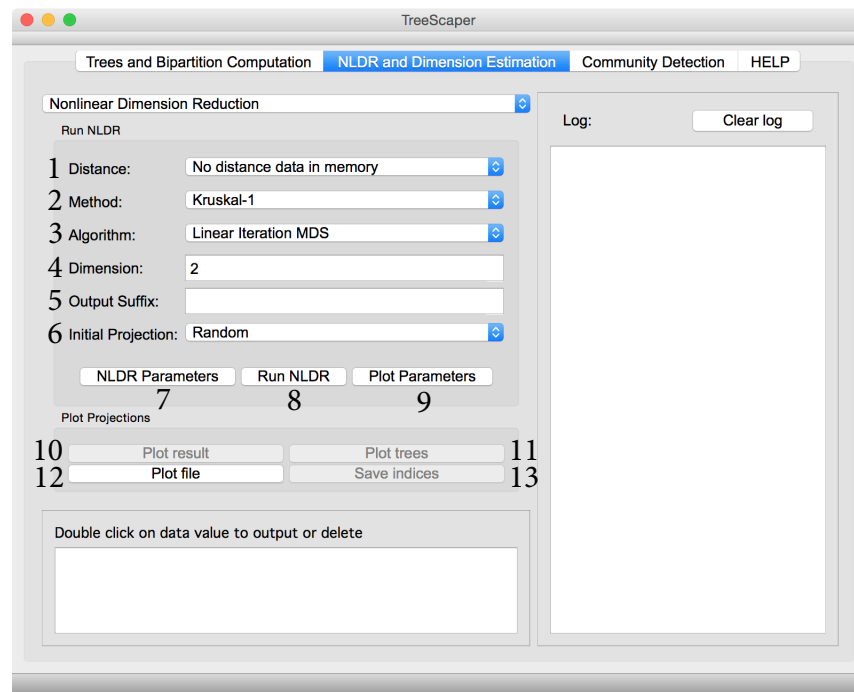


In the topological networks used by TreeScaper, nodes correspond to sampled trees and edge weights correspond to the affinity (or similarity) between pairs of sampled trees. Before community detection can be performed on these networks, tree-to-tree distances must be converted to affinities. TreeScaper allows users both to load pre-computed affinities from a file (options 1 and 2 below) or to calculate affinities directly from distances in memory (options 3 and 4 below).

- 1- **[Affinity File]** – Although affinities may be calculated from a distance matrix stored in memory, its also possible that youve previously calculated affinities and output that information to a file. In such cases, you may select that file here using browse.
- 2- **[Load Affinity File]** – Load the chosen “Affinity File” into memory.
- 3- **[Distance in memory]** – Select which distance matrix to use. If there is no distance matrix in memory, the drop down box will read “No distance data”.
- 4- **[Affinity type]** – Choose how to transform the distance matrix to an affinity matrix. “Reciprocal” computes the reciprocal of distance. “Exponential” computes an exponential transformation of the distance.

## 4.2 Nonlinear Dimensionality Reduction and Dimension Estimation

### 4.2.1 Nonlinear Dimensionality Reduction (NLDR)



NLDR seeks to find a low-dimensional representation that accurately reflects a set of high-dimensional distances. TreeScaper allows users to run NLDR for any specified number of dimensions, although points can only be visualized in 1 – 3 dimensions. If more than 3 dimensions are used when NLDR is run, only the first 3 axes will be used in the plot.

#### *Run NLDR Panel*

- 1- **[Distance]** – Select a distance matrix from memory. See options in the “Trees and Bipartition Computation” tab to compute a distance matrix if one is not currently available in memory.
- 2- **[Method]** – Select the method used for NLDR. Classic Scaling uses singular value decomposition (SVD). Kruskal–1 uses Kruskal stress–1 as the stress function. Normalized uses normalized stress. NLM uses Sammon nonlinear mapping. CCA uses curvilinear component analysis.
- 3- **[Algorithm]** – Select the optimization algorithm. For Classic Scaling, the algorithm used is always “SVD”. For Kruskal–1, the “Stochastic” algorithm is not recommended because it is slow. For Normalized, NLM and CCA, the Linear Iteration MDS algorithm is not available. Linear iteration is used in the Hillis et al. paper ???. “Stochastic” uses a stochastic gradient descent algorithm and appears to have the best performance.
- 4- **[Dimension]** – Select the number of dimensions for the low-dimensional projection. Any number of dimensions may be used, but only 1 – 3 dimensions can be visualized.
- 5- **[Output Suffix]** – Choose a suffix to distinguish the output files when they have the same name.

- 6- **[Initial Projection]** – This is the initial matrix used for the iteration. “Random” means a uniform random number is used. ”Classic scaling” means the initial matrix is the result of classic scaling:

[https://en.wikipedia.org/wiki/Multidimensional\\_scaling#Classical\\_multidimensional\\_scaling](https://en.wikipedia.org/wiki/Multidimensional_scaling#Classical_multidimensional_scaling)

Usually, initialization with classic scaling makes the algorithm converge quickly.

- 7- **[NLDR parameters]** – Clicking this button opens a dialog box with a variety of specific parameters that can be adjusted for each of the NLDR methods.
- 8- **[Run NLDR]** – Clicking this button uses the chosen NLDR method and optimization algorithm to project trees into low-dimensional space. The projection can then be visualized in TreeScaper directly (see below) or TreeScaper can output coordinate files, so that you can use third party plotting packages like Matplotlib to create publication quality figures in a number of output formats (See output files section).
- 9- **[Plot parameters]** – Click this and a dialog box will appear. Options in this box allow the user to customize plots of NLDR projections. Click “apply” to save any changes made to the parameters. Here are details on some parameters of greatest practical interest:

A drop down box in the “Plot” section that allows the user to select Points or Convex\_Hull. Points displays each tree as a separate point, while Convex\_Hull displays each cluster of trees as a convex hull.

In the “Plot” section, the user can define a desired number of clusters in which to group the trees. Each cluster will be given a unique color. Click **OK** to save the number of clusters.

The user can set the range of tree indices in each cluster manually using the “range” column under “Cluster index”. These ranges can either be consecutive with only the beginnings and endings specified (e.g., 1 – 10) or they can consist of a comma-separated list of nonconsecutive indices (e.g., 1, 3, 6, 9, 14). Alternatively, users can check “step size” and simply specify the number of trees to be in each cluster. The “step size” option will start at 0 and increase by the step size to create the tree index range for each cluster. If the largest index is greater than the number of trees in the tree set, this error will appear in the log: “error: the largest index should not be greater than the number of points!” If the largest index is less than the number of trees in the tree set, the trees whose indices were excluded will not be plotted in the NLDR projection. If the parameters are changed such that indices are missing between 0 and the largest index, TreeScaper will most likely crash. *Make sure that all tree indices are assigned to a cluster.*

### *Plot Projections Panel*

- 10- **[Plot Result]** – Creates a 1–, 2–, or 3–dimensional representation of the NLDR projection. To the left of the figure is a legend showing the colors assigned to the clusters. On a Mac, clicking and dragging allows the plot to be rotated in 3 dimensions. Holding down the Control

key while clicking and dragging rotates the plot in 2 dimensions. Holding down the Shift key while clicking and dragging repositions the plot. Pressing the Shift and R keys together will toggle in between two functions for clicking and dragging: (1) rotating the plot or (2) selecting points in the plot.

- 11- **[Plot trees]** – *Note: this function only works for the Mac version currently.* If points in the plot have been selected (see 10 above), clicking this button will separately display each selected tree. For each displayed tree, scrolling up and down will zoom in and out. Holding down either the Option or Command keys while clicking and dragging will reposition the tree. If some labels are not initially visible, try zooming in.
- 12- **[Plot File]** – Select a file containing a previously computed NLDR projection and plot it. These files contain the coordinates in low-dimensional space and have `_COR_` in their names.
- 13- **[Save Indices]** – If points in the plot have been selected (see 10 above), clicking this button will output the selected trees to a file.

### *Output Files*

Each time an NLDR analysis is run in TreeScaper, a series of output files are automatically created. These files have names indicating the name of the tree set, the tree-to-tree distance, the NLDR method, and the optimization algorithm. If NLDR is run multiple times and any of these parameters are changed, a new set of files will be created. If NLDR is run multiple times with the same parameters, only the final run will be saved.

**\*\_\*D\_\*\_COR\_\*.out:** The `*s` represent the filename, dimension, method and algorithm. This file contains the NLDR coordinates. Each row represents a sampled tree and is depicted as a point in low-dimensional space. *For more customized NLDR plots, users can simply read these coordinates into software packages with greater plotting flexibility (e.g., R) and adjust options as they see fit.*

**\*\_\*D\_\*\_DIS\_\*.out:** The Euclidean distance matrix of the NLDR coordinates.

**\*\_\*D\_\*\_STR\_\*.out:** The value of stress function after optimization.

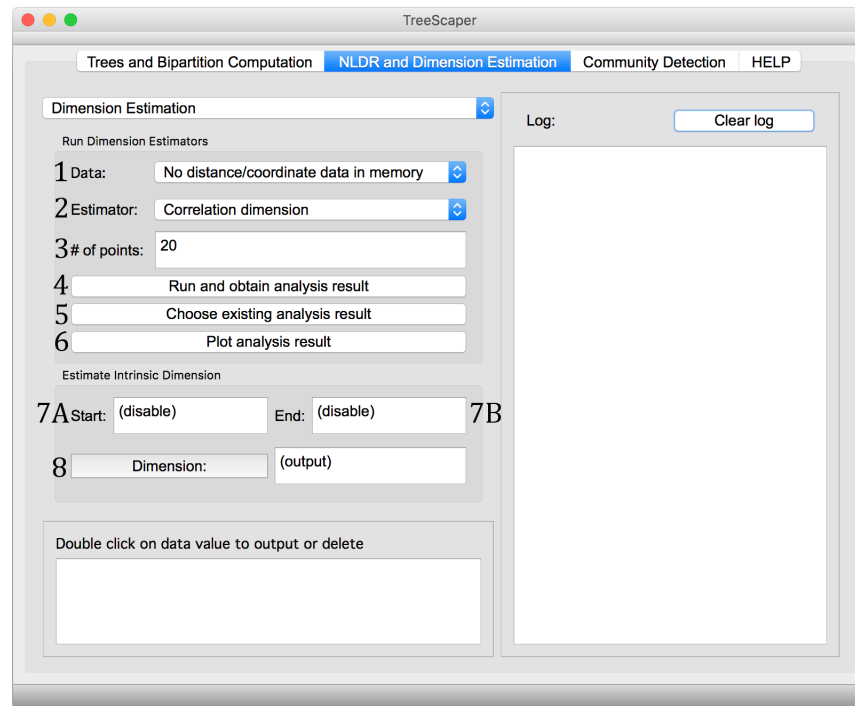
**\*\_\*D\_\*\_TIM\_\*.out:** The time cost.

**\*\_\*D\_\*\_1NN\_\*.out:** The first element is the 1nn percentage of the original distance matrix. The second element is the 1nn percentage of the result distance matrix.

**\*\_\*D\_\*\_CON\_\*.out:** The continuity. The first column is the number of neighbors considered. The second column is the continuities.

**\*\_\*D\_\*\_TRU\_\*.out:** The trustworthiness. The first column is the number of neighbors considered. The second column is the trustworthiness.

### 4.2.2 Dimension Estimation



In addition to conducting NLDR analyses, TreeScaper allows users to estimate the intrinsic dimensionality of their data. By doing so, one can gain a sense for how much information is lost when projecting into lower dimensional space.

#### *Run Dimension Estimators*

- 1- **[Data]** – Select a set of tree-to-tree distances from this dropdown menu.
- 2- **[Estimator]** – Select one of three dimension estimators: correlation dimension, maximum likelihood, or nearest neighbor estimator.
- 3- **[# of points]** – For the maximum likelihood and nearest neighbor estimators, this value determines the number of neighbors considered. For correlation dimension, this value represents the number of points used in the interval.
- 4- **[Run and obtain analysis results]** – Run the dimension estimator.
- 5- **[Choose existing analysis result]** – Choose a previous dimension estimation output file for analysis.
- 6- **[Plot analysis result]** – Plot a figure of the output file.

#### *Estimate Intrinsic Dimension*

- 7 **A- [Start] and B- [End]** – Choose the range of the interval used to compute dimension.

- 8- **[Dimension]** – Estimate the dimension in the given interval. For the nearest neighbor and correlation dimension estimators, the dimension is the slope. For the maximum likelihood estimator, the dimension is the mean.

*Output files*

**\*\_CORR\_DIM\_logvslog.out:** Correlation dimension output. The \* represents the filename. If one plots a figure of the first column versus the second column, the slope of the curve is the correlation dimension.

**\*\_CORR\_DIM\_derilogvslog.out:** Correlation dimension output. This is the slope of “\*\_CORR\_DIM\_logvslog.out” and is given by using the central difference method.

**\*\_MLE\_DIM\_k.out:** Maximum likelihood estimator output. The  $i^{th}$  row represents the dimension given by considering the  $k$  nearest neighbor. The dimension is the mean of some interval of the neighborhood.

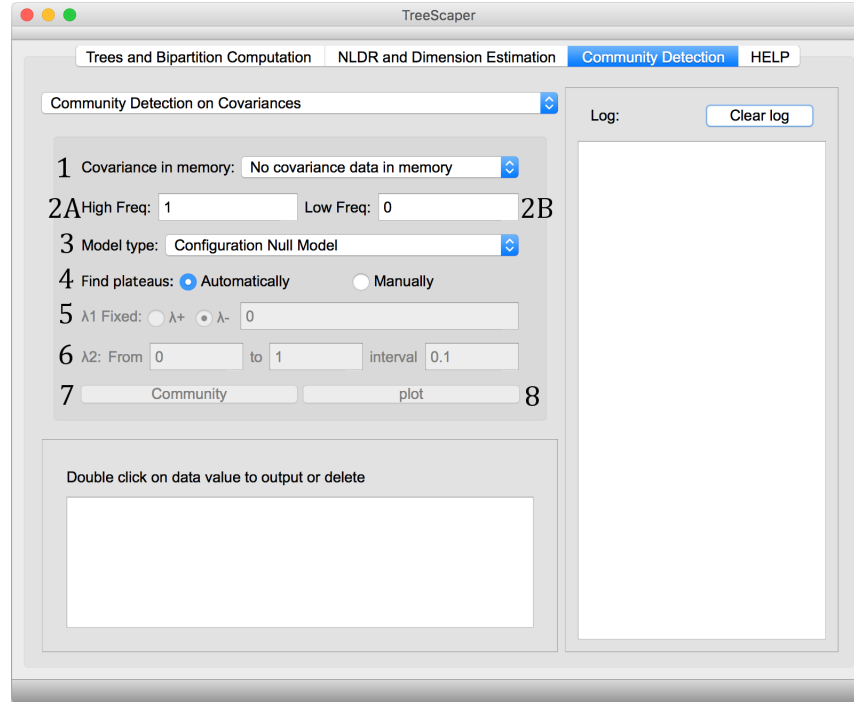
**\*\_MLE\_DIM\_dim.out:** Maximum likelihood estimator output. It is the mean of the \*\_MLE\_DIM\_k.out.

**\*\_NN\_DIM\_logvslog.out:** Nearest neighbor estimator output. If one plots a figure of the first column versus the second column, the slope of the curve is the nearest neighbor dimension.

**\*\_NN\_DIM\_derilogvslog.out:** Nearest neighbor estimator output. This is the slope of “\*\_NN\_DIM\_logvslog.out and is given by using the central difference method.

## 4.3 Community Detection

### 4.3.1 Community Detection in Bipartition-Covariance Networks



In the bipartition covariance network, nodes correspond to bipartitions and the weights of edges connecting these nodes correspond to their covariance in presence/absence across tree samples (Fig. 3). Community detection in these covariance networks attempts to group bipartitions that have positive covariances and separate bipartitions that have negative covariances.

- 1- **[Covariance in memory]** – Select the covariance matrix in memory to use for community detection. If there is no covariance matrix in memory, the drop down box will read “No covariance data”.
- 2- **A- [High Freq] and B- [Low Freq]** – High and low frequency bipartitions often have small covariances with every other bipartition in the tree set, making them very difficult to assign to a particular community. Because these bipartitions do not represent strongly supported conflicting signal, the user may want to filter these bipartitions out of the analysis. “High Freq” specifies the frequency above which bipartitions are filtered out. “Low Freq” specifies the frequency below which bipartitions are filtered out.
- 3- **[Model type]** – Select the model to use for community detection. Options are the Configuration Null Model, the Constant Potts Model, the Erdős-Rényi Model, and the No Null Model. Each of these models is described in more detail in Appendix A.
- 4- **[Find Plateaus]** – Several community detection models depend on tuning parameters to determine the scale at which communities are detected. In order to determine which values of these parameters are most appropriate, we can look for parameter ranges where community structure remains stable. When plotting community number against parameter values, the largest of



these plateaus of intrinsic stability is preferred. If “Automatically” is selected, TreeScaper tries to find the intrinsic community structure of the network by efficiently testing different values of the tuning parameters. In this automated setting, TreeScaper looks for the largest plateau across values of tuning parameters that produce nontrivial community structures. Community structures are trivial if all bipartitions are placed in a single community or if every bipartition is placed in its own community. If “Manually” is selected, the user searches for the intrinsic community structure by specifying which parameter values are to be tested.

If “Manually” is selected in the “Find Plateaus” section:

- 5- [ **$\lambda 1$  fixed**] – If the Constant Potts Model (CPM) is used, or the Configuration Null Model (CNM) or Erdos-Rnyi Null Model (ERNM) are used and the trees are fully binary, only the difference between  $\lambda^+$  and  $\lambda^-$  matters for community detection. Therefore, if one wishes to vary the  $\lambda$  values to see how they affect community detection, one can fix one of the  $\lambda$  values and vary the other. This option allows the user to select whether to fix  $\lambda^+$  or  $\lambda^-$ .
- 6- [ **$\lambda 2$** ] – For the  $\lambda$  parameter that will be varied, the user can choose the range of values to test and the interval separating consecutive values. To find the intrinsic community structure of the network, vary the  $\lambda$  parameter such that at one extreme, each node is in its own community, and at the other extreme, all the nodes are grouped into a single community. As with automatic community detection, the intrinsic community structure can be chosen by looking for the largest range of  $\lambda$  values that produces the same non-trivial community structure (i.e., the largest plateau).
- 7- [**Community**] – Runs community detection for a set of  $\lambda$  values.

If “Automatically” is selected, the lambda values tested and the largest plateaus are output to the log. For the plateaus, the “Lower Bound” range is a conservative estimate for the size of the plateau, while the “Upper Bound” is each plateaus largest possible size. If the lower bound of the largest plateau is larger than the upper bound of the second largest plateau, the user can be confident in the designation of a single plateau as the largest. Automatic search does not work for non-binary trees when the CNM or ERNM models are used. If “Manually” is selected, the community structure detected for each  $\lambda$  value (i.e., the node indices placed in each community) is also output to the log.

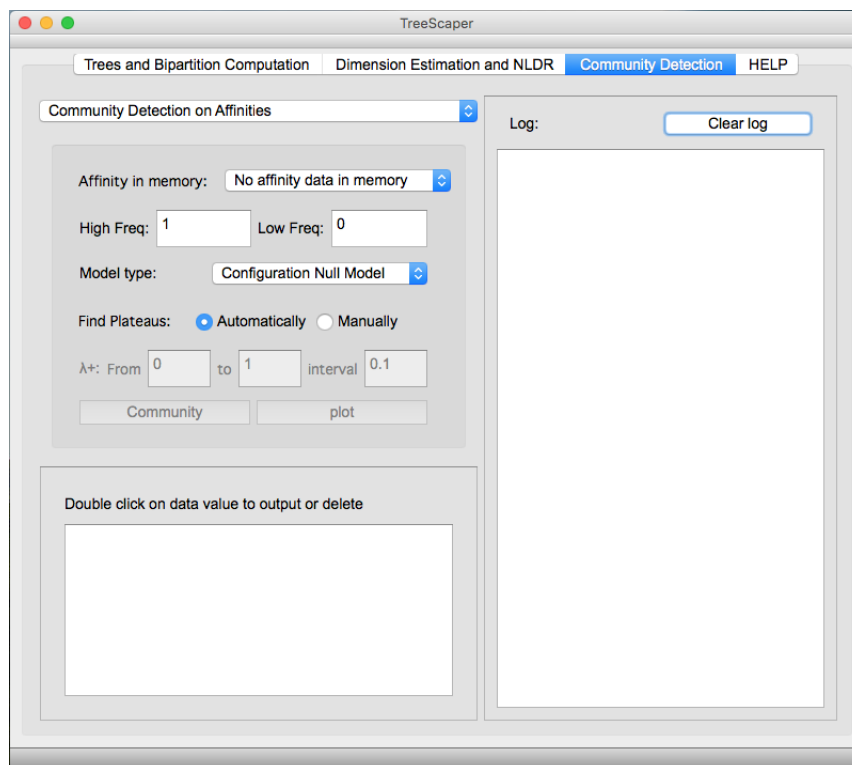
Once community detection is complete, a summary of the community detection results will be written to file. The name of the file is listed at the end of the log. The summary will be a tab delimited file in the format of “1000bp1L\_unrooted\_unweighted\_Covariance\_Matrix\_community\_auto\_results.out”. The first row contains numbers used to label each unique community structure. This number starts at 0 and increases by 1 every time a unique community structure is found. The second row contains each of the tested  $\lambda 2$  values. The third row contains the number of communities.

**Note:** TreeScaper currently does not support use of the CNM or ERNM models for tree sets with non-binary trees. In these cases, the simple difference between  $\lambda^+$  and  $\lambda^-$  is not all that matters. Both  $\lambda^+$  and  $\lambda^-$  need to be varied independently to find a two-dimensional plateau. Given the

complexity of this problem, we have not yet attempted an implementation.

- 8- **[Plot]** – Generate a plot of the community labels, number of communities, and the modularities for different values of  $\lambda$ . These plots help the user to identify plateaus.

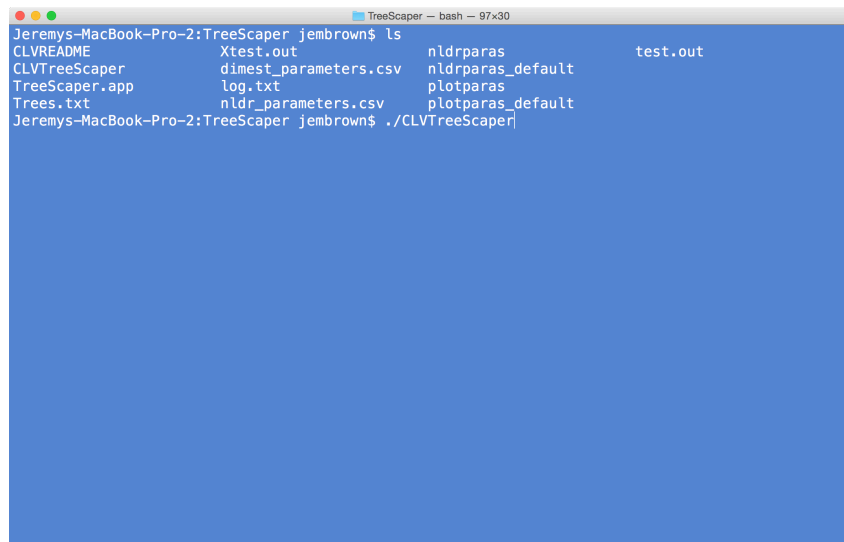
### 4.3.2 Community Detection in Topological Affinity Networks



In topological networks, nodes represent individual topologies and edge weights correspond to the similarity (or affinity) between each pair of topologies (Fig. 2). Community detection in these affinity networks attempts to find community boundaries that group similar topologies and separate those that are dissimilar.

Most of the community detection menus for the affinity matrices are the same as those for the covariance matrices. The community detection models are also the same, with two exceptions. First, the  $\lambda^-$  values are not applicable because there are no negative edges in the affinity matrices. Second, *the most interesting community structure for affinity matrices often corresponds to the second largest plateau*. This plateau is of particular interest when there are many trees in the tree set with identical topologies, and a discrete tree-to-tree distance (e.g., unweighted Robinson-Foulds, matching, or SPR) is used. In this situation, all duplicate tree topologies will have a zero distance. If a wide range of parameter space is explored, the largest plateau will often correspond to a community structure where every unique topology is placed in its own community, which should generally be considered a trivial solution. This community structure is not usually of much biological interest, because it is equivalent to a simple list of the number of unique topologies and their frequencies with no information about affinity.

## 5 COMMAND-LINE TREESCAPER



```
Jeremys-MacBook-Pro-2:TreeScaper jembrown$ ls
CLVREADME      Xtest.out      nldrparas      test.out
CLVTreeScaper  dimest_parameters.csv  nldrparas_default
TreeScaper.app log.txt        plotparas
Trees.txt      nldr_parameters.csv  plotparas_default
Jeremys-MacBook-Pro-2:TreeScaper jembrown$ ./CLVTreeScaper
```

There are three general run modes for the command-line version of TreeScaper (“CLVTreeScaper”). Using one of these three flags as the first command-line argument sets the mode (e.g., CLVTreeScaper trees).

### (1) **-trees**

In this mode, users can compute a majority rule/strict consensus tree, distance matrix, bipartition matrix, covariance matrix, affinity matrix, or detect communities in an affinity or covariance network. Relevant arguments include:

**-f:** Provide the name of the file that contains the data

**-ft:** The file type. Options are:

‘**Trees**’: The file contains trees. The tree format can be either Newick or Nexus.

‘**Dist**’: The file contains distance matrix which can be used to compute affinity matrix or communities.

‘**Cova**’: The file contains covariance matrix which can be used to compute communities.

**-w:** Indicate whether trees are weighted. Options are:

‘**1**’: weighted

‘**0**’: unweighted

**-r:** Indicate whether trees are rooted. Options are:

‘**1**’: rooted

‘**0**’: unrooted

**-o:** This option is used to indicate what output the user is interested in. Options are:

‘**BipartMatrix**’

‘**Consensus**’

**‘Dist’**  
**‘Affinity’**  
**‘Covariance’**  
**‘Community’**

When outputting a bipartition matrix (-o BipartMatrix):

**-bfm:** Bipartition matrix output type. Options are:

**‘list’:** Output sparse matrix in the form (row, column, value)  
**‘matrix’:** Output as if it is a full matrix

When computing a majority-rule or strict consensus tree (-o Consensus), use the -if, -ct, and/or -cfm flags:

**-if:** The name of a list file. Consensus tree computations will only consider the trees indicated in the file.

**-ct:** The type of consensus tree to be computed. Options are:

**‘Majority’:** Majority consensus tree  
**‘Strict’:** Strict consensus tree

**-cfm:** Format of the consensus tree file. Options are:

**‘Newick’**  
**‘Nexus’**

When computing a distance matrix (-o Dist):

**-dm:** Indicates the distance metric. Options are:

**‘URF’:** Unweighted Robinson-Foulds distance  
**‘RF’:** Weighted Robinson-Foulds distance  
**‘Mat’:** Matching distance  
**‘SPR’:** Subtree-Prune-Regraft

When computing an affinity matrix (-o Affinity):

**-dm:** Indicates the distance metric. Options are:

**‘URF’ :** Unweighted Robinson Foulds distance  
**‘RF’:** Weighted Robinson Foulds distance  
**‘Mat’:** Matching distance  
**‘SPR’:** Subtree-Prune and Regraft

**-am:** Indicates the distance to affinity transformation. Options are:

**‘Rec’:** Reciprocal  
**‘Exp’:** Exponential

When detecting communities (-o Community):

**-t:** Target matrix used to compute communities. Options are:

‘**Affinity**’: affinity matrix

‘**Covariance**’: covariance matrix

**-cm:** Model used to compute communities. Options are:

‘**CNM**’: Configuration Null Model

‘**CPM**’: Constant Potts Model

‘**ERNM**’: Erdos-Rnyi Null Model

‘**NNM**’: No Null Model

**-lm:** Method of plateau detection. Options are:

‘**auto**’: automatically choose lambdas and find plateaus

‘**manu**’: specify intervals by users to find plateaus

The following flags are used to specify values of lambda for manual searches:

**-lp:** Specify a fixed value of  $\lambda^+$ . Must be between 0 and 1. Used when -lpiv is zero (see below).

**-lps, -lpe, -lpiv:** Starting, ending, and sampling intervals for  $\lambda^+$ . Used to explore a range of possible values for  $\lambda^+$ .

**-ln:** Specify a fixed value of  $\lambda^-$ . Must be between 0 and 1. Used when -lniv is zero (see below).

**-lns, -lne, -lniv:** Starting, ending, and sampling intervals for  $\lambda^-$ . Used to explore a range of possible values for  $\lambda^-$ .

*Note: Either  $\lambda^+$  or  $\lambda^-$  must be fixed, because plateau detection is undefined when both vary.*

**-hf:** Frequency upper bound. A number between 0 and 1. Nodes with frequencies above this value are ignored.

**-lf:** Frequency lower bound. A number between 0 and 1. Nodes with frequencies below this value are ignored.

**Examples of command-line runs** *Options specified by the are given inside braces. When specific alternatives are available, they are separated by commas (e.g., {option1,option2}). When numbers can be specified anywhere in a continuous range, the bounds of the range are separated by a dash (e.g., {0-1}).*

*Compute a Bipartition Matrix:*

```
./CLVTreeScaper -trees -f trees.txt -ft Trees -w 1,0 -r 1,0 -o BipartMatrix -bfm {list,matrix}
```

*Compute a Consensus Tree:*

```
./CLVTreeScaper -trees -f {trees.txt} -ft Trees -w {1,0} -r {1,0} -o Consensus -if IndicesFileName -ct {Majority,Strict} -cfm {Newick,Nexus}
```

*Compute Distance Matrix:*

```
./CLVTreeScaper -trees -f {trees.txt} -ft Trees -w {1,0} -r {1,0} -o Dist -dm {URF,RF,Mat,SPR}
```

*Compute Affinity Matrix:*

```
./CLVTreeScaper -trees -f {trees.txt} -ft Trees -w {1,0} -r {1,0} -o Affinity -dm  
{URF,RF,Mat,SPR} -am {Exp,Rec}
```

*Compute Covariance Matrix:*

```
./CLVTreeScaper -trees -f {trees.txt} -ft Trees -w {1,0} -r {1,0} -o Covariance
```

*Compute Communities with  $\lambda^+$  Fixed:*

```
./CLVTreeScaper -trees -f {trees.txt} -ft Trees -w {1,0} -r {1,0} -o Community -t  
{Affinity,Covariance} -cm {CNM,CPM,ERNM,NNM} -lm manu -lp {AnyNumber} -lns  
{AnyNumber} -lne {AnyNumber} -lniv {AnyNumber} -hf {0-1} -lf {0-1}
```

*Compute Communities with  $\lambda^-$  Fixed:*

```
./CLVTreeScaper -trees -f {trees.txt} -ft Trees -w {1,0} -r {1,0} -o Community -t  
{Affinity,Covariance} -cm {CNM,CPM,ERNM,NNM} -lm manu -ln {AnyNumber} -lps  
{AnyNumber} -lpe {AnyNumber} -lpiv {AnyNumber} -hf {0-1} -lf {0-1}
```

*Compute Communities with Automatically Chosen Lambdas:*

```
./CLVTreeScaper -trees -f {trees.txt} -ft Trees -w {1,0} -r {1,0} -o Community -t  
{Affinity/Covariance} -cm {CNM/CPM/ERNM/NNM} -lm auto -hf {0-1} -lf {0-1}
```

*Load Distances and Compute Affinity Matrix:*

```
./CLVTreeScaper -trees -f {dist.txt} -ft Dist -o Affinity -am {Exp,Rec}
```

*Load Distances and Compute Affinity Communities Automatically:*

```
./CLVTreeScaper -trees -f {dist.txt} -ft Dist -o Community -t Affinity -am {Exp,Rec} -cm  
{CNM,CPM,ERNM,NNM} -lm auto -hf {0-1} -lf {0-1}
```

*Load Covariances and Compute Communities using Automatic Search on Lambda:*

```
./CLVTreeScaper -trees -f {cova.txt} -ft Cova -o Community -cm {CNM,CPM,ERNM,NNM}  
-lm auto -hf {0-1} -lf {0-1}
```

## (2) -nldr

In this mode, users can project trees into lower dimensional space using non-linear dimensionality reduction (NLDR). Relevant arguments include:

**-f:** Name of the file containing distance data.

**-t:** The type of distances contained in the file. Options are:

**‘DIS’:** A lower triangle matrix of original distances

**‘COR’:** Low-dimensional Euclidean coordinates (if already computed).

**-d:** The desired dimension of the Euclidean representation (usually 1, 2, or 3).

- c: The chosen cost function. Options are:
  - ‘CLASSIC\_MDS
  - ‘KRUSKAL1
  - ‘NORMALIZED
  - ‘SAMMON
  - ‘CCA
- a: The chosen NLDR algorithm. Options are:
  - ‘LINEAR\_ITERATION
  - ‘MAJORIZATION
  - ‘GAUSS\_SEIDEL
  - ‘STOCHASTIC
- i: The method for generating initial Euclidean coordinates. Options are:
  - ‘RAND’: Randomly choose coordinates for each point.
  - ‘CLASSIC\_MDS’: Generate initial coordinates using classic multi-dimensional scaling (MDS).
- o: The suffix for output file names.
- s: A random seed, if initial coordinates are generated randomly.

**Example command:**

```
./CLVTreeScaper -nldr -f {test.out} -t {DIS,COR} -d {1,2,3,}
-c {CLASSIC_MDS,KRUSKAL1,NORMALIZED,SAMMON,CCA}
-a {LINEAR_ITERATION,MAJORIZATION,GAUSS_SEIDEL,STOCHASTIC}
-i {RAND,CLASSIC_MDS} -o {run1} -s 1
```

### (3) -dimest

In this mode, users can estimate the intrinsic dimensionality of their data. This estimate can help in deciding on an appropriate number of dimensions to use when performing NLDR projections.

- f: Name of the file containing distance data.
- i: The type of distances contained in the file. Options are:
  - ‘DIS’: A lower triangle matrix of original distances
  - ‘COR’: Low-dimensional Euclidean coordinates (if already computed).
- e: The chosen estimator. Options are:
  - ‘CORR\_DIM’: Correlation dimension estimator
  - ‘NN\_DIM’: Nearest neighbor estimator
  - ‘MLE\_DIM’: Maximum likelihood estimator

**Example command:**

```
./CLVTreeScaper -dimest -f {test.out} -i {DIS,COR} -e {CORR_DIM,NN_DIM,MLE_DIM}
```



## A COMMUNITY DETECTION MODELS

Community detection includes a broad class of methods that attempt to find structure in networks, by identifying groups of nodes that are more densely or tightly connected to each other than they are to other nodes in the network [Newman, 2010]. These methods do not require the number and size of groups (known as communities) to be identified in advance, in contrast to graph partitioning approaches. Each of the community detection methods implemented in TreeScaper employ a quantity known as the Hamiltonian ( $\mathcal{H}$ ). Roughly analogous to the use of this term in quantum mechanics, the Hamiltonian represents the energy imposed by a given community structure. The structure with the minimum energy represents the most natural division of nodes. Below we provide definitions of  $\mathcal{H}$  for each of the methods in TreeScaper, as well as some explanation of how these definitions influence the detected communities, in order to help users efficiently explore parameter space and properly interpret model output. We also direct those interested to more in-depth explanations of these methods in papers by Fortunato, 2010; Reichardt and Bornholdt, 2006; Raghavan et al., 2007; Traag and Bruggeman, 2009; Traag et al., 2011; Traag, 2014.

### A.1 No Null Model

In the first case, which we term No Null Model (NNM, also known as the label propagation method),  $\mathcal{H}$  is defined for the set of all communities,  $\{\sigma\}$ , and is given by

$$\mathcal{H}(\{\sigma\}) = - \sum_{i,j} A_{i,j} \delta(\sigma_i, \sigma_j), \quad (1)$$

where the sum is over all nodes  $i$  and  $j$ ,  $A_{i,j}$  is the adjacency between nodes  $i$  and  $j$  (i.e., is there exists an edge connecting nodes  $i$  and  $j$ ), and  $\sigma_i$  is the community to which bipartition  $i$  belongs.  $\delta(\sigma_i, \sigma_j)$  is defined as 1 when  $i$  and  $j$  are in the same community, and 0 otherwise. The NNM contains no tunable parameters and is generally of the least interest, since its Hamiltonian has only one global optimum with all nodes in a single community. However, local optima could be of some interest.

One way to refine our approach to community detection beyond the NNM involves first defining an expectation for structure based on a stochastic model of network construction. In these cases, the existence of communities can be revealed by comparison between the actual density of edges in a subgraph and the density one would expect to have in the null subgraph without community structure. The expected edge density depends on the chosen null model. The two following methods are based on different choices of null model.

### A.2 Erdős-Rényi Model

For the Erdős-Rényi Model (ERM),  $\mathcal{H}$  is given by

$$\mathcal{H}(\{\sigma\}) = - \sum_{i,j} \left[ A_{i,j} - \left( p_{i,j}^+ \lambda^+ - p_{i,j}^- \lambda^- \right) \right] \delta(\sigma_i, \sigma_j), \quad (2)$$

where  $p_{i,j}$  is the probability of a positive ( $p_{i,j}^+$ ) or negative ( $p_{i,j}^-$ ) edge between  $i$  and  $j$  in the null case, while  $\lambda^+$  and  $\lambda^-$  are tunable parameters. All other definitions are as Equation (1) for the

NNM. For Erdős-Rényi's random graph model, the probability of occurrence for any particular positive edge ( $p_{i,j}^+$ ) is  $m^+/n^2$ , while the corresponding probability for any particular negative edge ( $p_{i,j}^-$ ) is  $m^-/n^2$ , where  $m^+$  is the sum of positive edge weights,  $m^-$  is the sum of the absolute value of the negative edge weights, and  $n$  is the number of nodes.

If community detection is performed on a bipartition covariance network and there are no polytomies in the tree set, then for each bipartition, the sum of its covariances with all other bipartitions equals zero. Correspondingly, the sum of the absolute value of the positive covariances is equal to the sum of the absolute value of the negative covariances. In this case, Equation (2) simplifies to

$$\mathcal{H}(\{\sigma\}) = - \sum_{i,j} \left[ A_{i,j} - p_{i,j}^+ (\lambda^+ - \lambda^-) \right] \delta(\sigma_i, \sigma_j). \quad (3)$$

For this model, changing the  $\lambda$  tuning parameters affects the preferred community size by adjusting the reward and penalty for including positive and negative edges, respectively, in a community. In the simplified equation, if  $\lambda^+ > \lambda^-$ , then small communities are preferred. If  $\lambda^- > \lambda^+$ , then large communities are preferred.

### A.3 Configuration Null Model

For the Configuration Null Model (CNM),  $\mathcal{H}$  is given by

$$\mathcal{H}(\{\sigma\}) = - \sum_{i,j} \left[ A_{i,j} - \left( \frac{k_i^+ k_j^+}{2m^+} \lambda^+ - \frac{k_i^- k_j^-}{2m^-} \lambda^- \right) \right] \delta(\sigma_i, \sigma_j), \quad (4)$$

where  $k_i^+$  is the sum of all positive edges connecting nodes  $i$ ,  $k_i^-$  is the sum of the absolute value of all negative edges connecting to node  $i$ ,  $m^+$  is the sum of the positive edges in community  $\sigma$ , and  $m^-$  is the sum of the absolute value of all negative edges in community  $\sigma$ . All of the other terms are as above.

If bipartition-covariance community detection is performed and there are no polytomies in the tree set the above equation simplifies to:

$$\mathcal{H}(\{\sigma\}) = - \sum_{i,j} \left[ A_{i,j} - \frac{k_i^+ k_j^+}{2m^+} (\lambda^+ - \lambda^-) \right] \delta(\sigma_i, \sigma_j). \quad (5)$$

As with ERM, changing the  $\lambda$  tuning parameters affects the preferred community size by adjusting the reward and penalty for including positive and negative edges, respectively. In the simplified equation, if  $\lambda^+ > \lambda^-$ , then communities are more likely to include negative edges. If  $\lambda^- > \lambda^+$ , then communities are less tolerant of negative edges and more strongly favor only positive edges.

### A.4 Constant Potts Model

For the Constant Potts Model (CPM),  $\mathcal{H}$  is given by

$$\mathcal{H}(\{\sigma\}) = - \sum_{i,j} [A_{i,j} - (\lambda^+ - \lambda^-)] \delta(\sigma_i, \sigma_j), \quad (6)$$

where all terms are as above. As with the other models, changing the values of the  $\lambda$  tuning parameters affects the preferred community size. If  $\lambda^+ > \lambda^-$ , small communities are preferred. If  $\lambda^- > \lambda^+$ , large communities are preferred.

## References

- Amenta, N. and Klingner, J. (2002). Case study: visualizing sets of evolutionary trees. In *IEEE Symposium on Information Visualization, 2002. INFOVIS 2002.*, pages 71–74.
- Castoe, T. A., de Koning, A. P. J., Kim, H.-M., Gu, W., Noonan, B. P., Naylor, G., Jiang, Z. J., Parkinson, C. L., and Pollock, D. D. (2009). Evidence for an ancient adaptive episode of convergent molecular evolution. *Proceedings of the National Academy of Sciences*, 106(22):8986–8991.
- Fortunato, S. (2010). Community detection in graphs. *Physics Reports*, 486(3–5):75 – 174.
- Gori, K., Suchan, T., Alvarez, N., Goldman, N., and Dessimoz, C. (2016). Clustering genes of common evolutionary history. *Molecular Biology and Evolution*, 33(6):1590.
- Hillis, D. M., Heath, T. A., John, K. S., and Anderson, F. (2005). Analysis and visualization of tree space. *Systematic Biology*, 54(3):471.
- Lewitus, E. and Morlon, H. (2016). Characterizing and comparing phylogenies from their laplacian spectrum. *Systematic Biology*, 65(3):495.
- Newman, M. E. J. (2010). *Networks: An Introduction*. Oxford University Press, Inc., New York, NY, USA.
- Raghavan, U. N., Albert, R., and Kumara, S. (2007). Near linear time algorithm to detect community structures in large-scale networks. *Phys. Rev. E*, 76:036106.
- Reichardt, J. and Bornholdt, S. (2006). Statistical mechanics of community detection. *Phys. Rev. E*, 74:217 – 224.
- Stockham, C., Wang, L.-S., and Warnow, T. (2002). Statistically based postprocessing of phylogenetic analysis by clustering. *Bioinformatics*, 18(suppl\_1):S285.
- Traag, V. (2014). *Algorithms and Dynamical Models for Communities and Reputation in Social Networks*. Springer, Heidelberg.
- Traag, V. and Bruggeman, J. (2009). Community detection in networks with positive and negative links. *Physical review. E, Statistical, nonlinear, and soft matter physics*, 80:036115.
- Traag, V. A., Van Dooren, P., and Nesterov, Y. (2011). Narrow scope for resolution-limit-free community detection. *Phys. Rev. E*, 84:016114.
- Wilgenbusch, J. C., Huang, W., and Gallivan, K. A. (2017). Visualizing phylogenetic tree landscapes. *BMC Bioinformatics*, 18(1):85.