TreeScaper Workshop Activity 2: Community Detection Methods

Jeremy Ash

January 8, 2017

Introduction

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

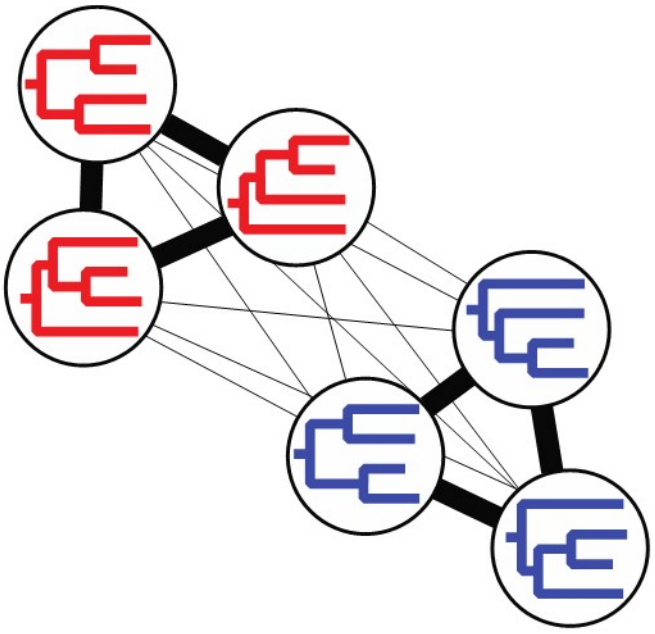

Figure 1: A cartoon topological affinity network. Circles are nodes in the network, each of which corresponds to one tree from a tree set. Edges represent the affinity (or similarity) between the trees, with thicker lines indicating greater affinity. Tree colors correspond to one intuitive definition of communities in this network.

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 (Figure 1).

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, Wang, and Warnow 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 (Figure 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 (Figure 3).

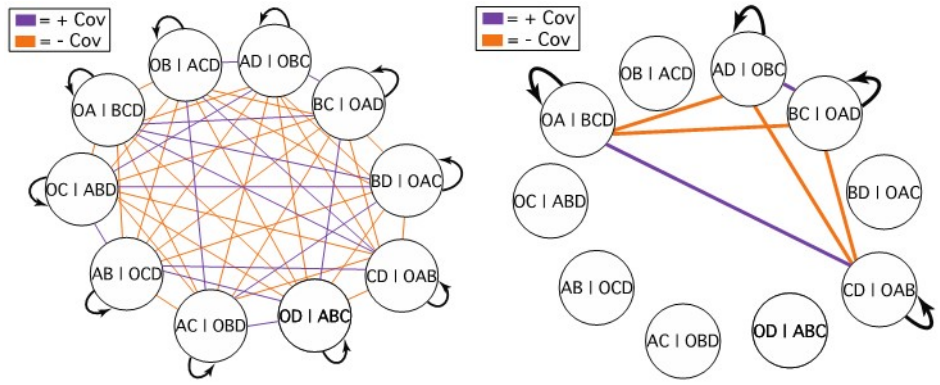

Figure 2: Two example bipartition covariance networks for sets of unrooted, 5-taxon trees. The network on the left corresponds to a tree set with a uniform distribution of frequencies across all possible tree topologies. Some weak covariances exist in this network, because some pairs of bipartitions 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.

Formatting Conventions

Throughout this tutorial, we format the text differently when referring to different TreeScaper components for clarity. *Tabs and dropdown menus* are italicized, **buttons** are bolded, "options" are in quotes.

Practical: Community Detection

The Data

The tree set that will be used during 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. Figure 3 shows the two

topologies, corresponding to the first and second halves of the alignment, respectively. Bipartitions that conflict between these topologies are in color.

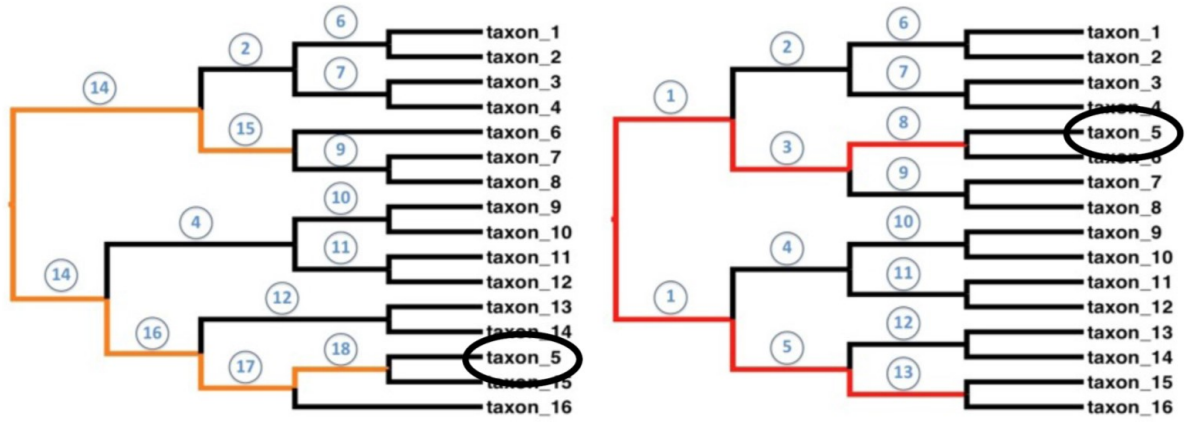

Figure 3: Guide trees used to generate the sequence alignment

A bootstrap analysis was then performed in Garli to produce the 100 trees in this tree set (Figure 4). In this tutorial, we will analyze this tree set in TreeScaper to explore the two conflicting signals present in the data.

Tutorial files

The additional files referred to during this tutorial can be found in the activities/com_detection_files directory on the github page.

Loading the Tree Set and Basic Computations

Loading the Tree Set

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.

Constructing the Bipartition Matrix

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., internal 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.

Creating a Consensus Tree

Next, we'll 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**

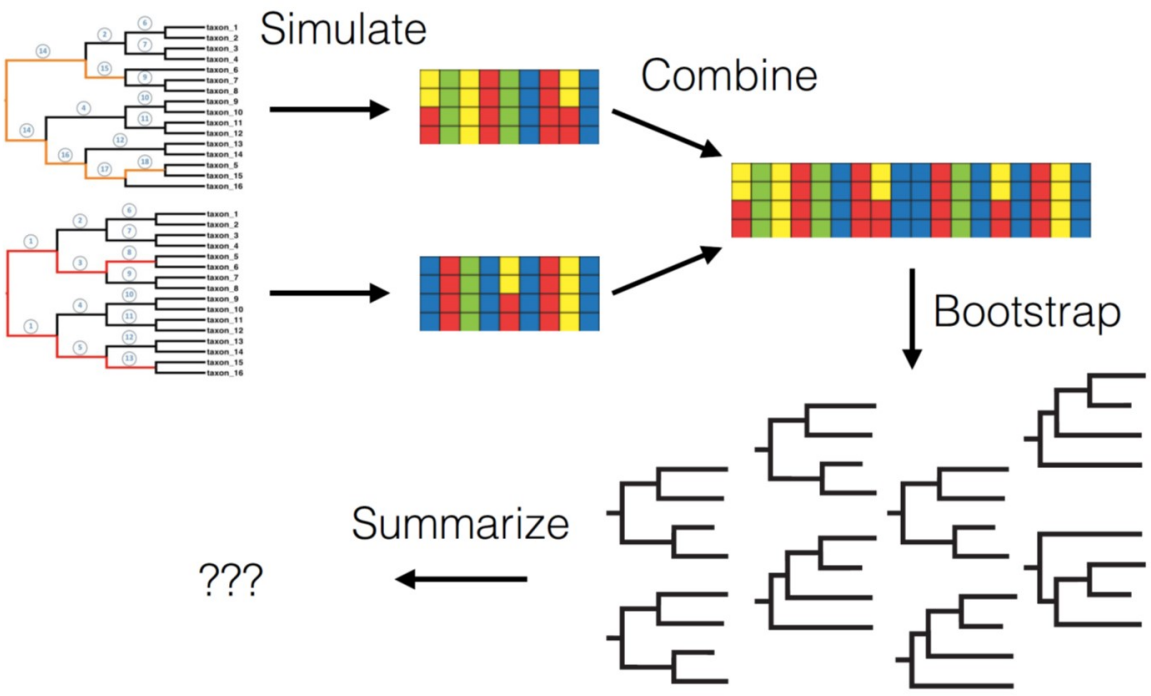

Figure 4: Simulation scheme used to generate the sequence alignment

Tree and then **Plot**. While examining the plot, you can zoom in and out by scrolling up or down. Holding down the Ctrl key (option key on Mac) 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.

Visualizing Tree Space

** This section will guide you through the steps to construct the NLDR visualization of the weighted RF distance matrix for this data set shown in Figure 1. The steps will be similar to Activity 1. You can skip ahead to the interpretation of the result if you would not like to do this again.

NLDR

From the main dropdown menu, select "Load Distance/Coordinate Data or Compute Tree-to-Tree Distances". From the *Method* dropdown select "Weighted Robinson Foulds", then click *Distance* to compute a matrix of pairwise distances between trees in your tree set.

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* dropdowns, 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 dialogue's 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. 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.

If you are using a Mac, 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.

NLDR Interpretation

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 set's topological affinity network. After rotation, your NLDR plot should look something like Figure 5. In this diagram, trees are colored by affinity community, which we will find in the next step of the tutorial.

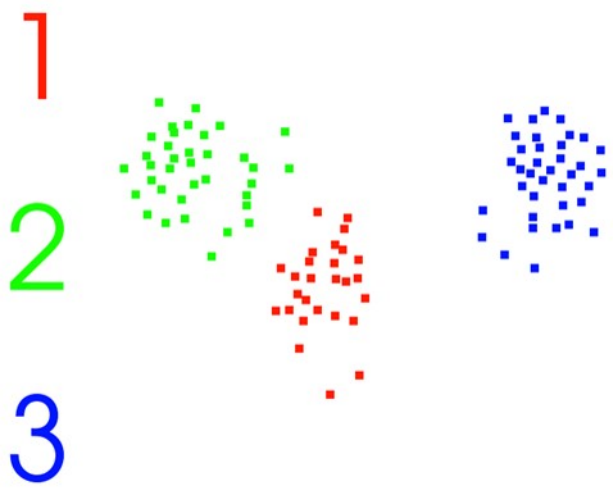

Figure 5: One rotation of the 3D NLDR projection outlined in the tutorial. Points are colored according to the output of topological affinity community detection analysis, which is described below.

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.

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.

Compute Distance Matrix In the topmost tab menu, go back to the *Trees and Bipartition Computation* tab. From the main dropdown menu, select "Load Distance/Coordinate Data or Compute Tree-to-Tree Distances". From the *Method* dropdown select "Weighted Robinson Foulds", then click *Distance* to compute a matrix of pairwise distances between trees in your tree set

Compute Affinity Matrix Select "Load/Convert Tree-to-Tree Distances to Affinities" from the main dropdown menu in this tab. Now, 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.

Find the Plateau Community Structure 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 variety of specific information about the community detection analysis is printed to the log. For now, we will ignore most of this information. The basic idea is to vary a tuning parameter (λ) 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.

Plot Community Detection Results 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 λ. You should see a plot that resembles Figure 6.

S 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. For the "Labels for Communities" line, the community label is incremented by one each time a new community structure is discovered.

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 λ is between 0.6 and 0.9. See Section 6.3 of the manual and the Appendix 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.

** The next two sections are optional because they are not very user friendly at the moment. We are working on making these texts easier for users.

Optional: Create Consensus Trees of Affinity Communities Using the indices of trees assigned to different communities, you can create community-specific consensus trees or color points by community assignmemnt 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 premade 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. Figure 7 shows 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.

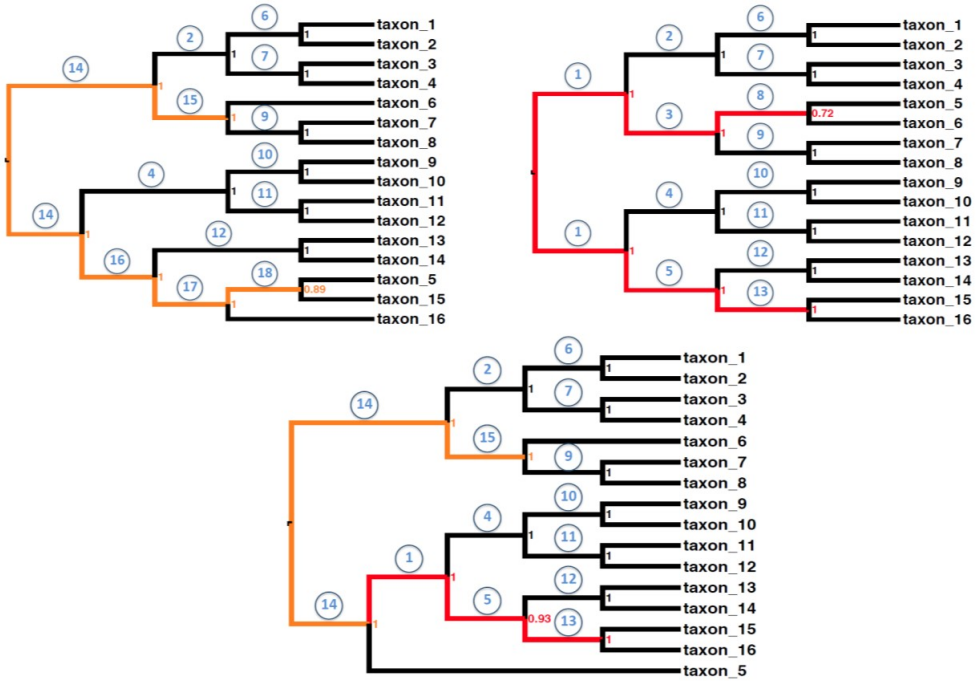

Figure 6: Consensus trees of the three affinity communities

Optional: Cluster NLDR Points by Affinity Community 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. 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 Figure 3.

Community Detection on a Bipartition-Covariance Network

Reload Tree Set Next, we will construct a bipartition covariance network. Reload the tree set, with one important difference: do not select the "weighted trees" option. You should not 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**.

Compute Covariances 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.

Find Plateau Community 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 .95 and the "Low Freq" to .05. These settings will filter out high and low frequency bipartitions, because they are difficult for the optimization algorithm to assign to a community. These bipartitions are not of interest because they are not strongly connected to any node in the network. In the *Model Type* dropdown select "Constant Potts Model", and for "Find Plateaus" select "Automatically". Click **Community**. The λ 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.

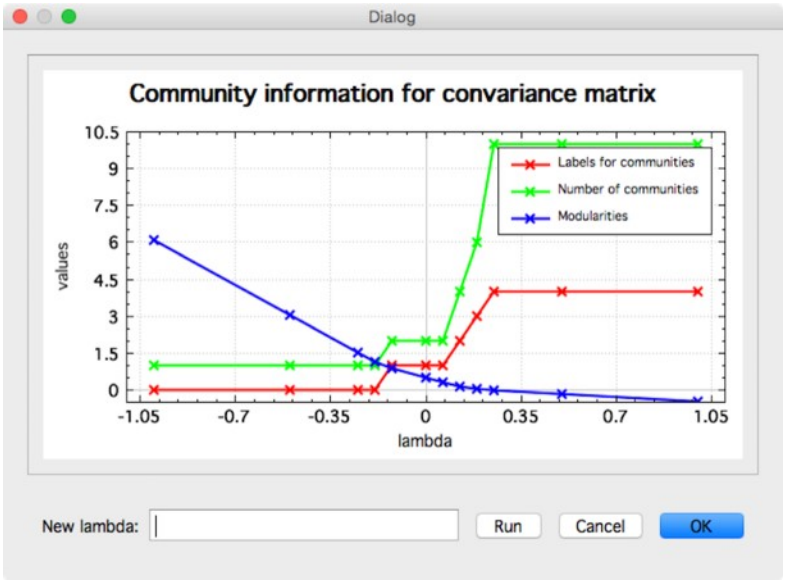

Figure 7: Example output from automatic community detection on the bipartition covariance network discussed in the tutorial. Note the trivial community structures represented by the plateaus at 1 and 10 communities.

Plot Results When community detection is finished, click **Plot** (Figure 8). 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.

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 "$\lambda 1$ 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.

Interpret Results 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 6.1.2 of the manual for an explanation of how to read the bipartition matrix output.

Because examining these bipartition matrices can be tedious, I wrote a python script that will convert the bipartition matrix into the file "1000bp1L_comKey.out". This file 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 Figure 3 where conflicting bipartitions are shown in color). Figure 8 contains a schematic of the bipartition-covariance network. Node numbers correspond to the numbered bipartitions in the guide trees. 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.

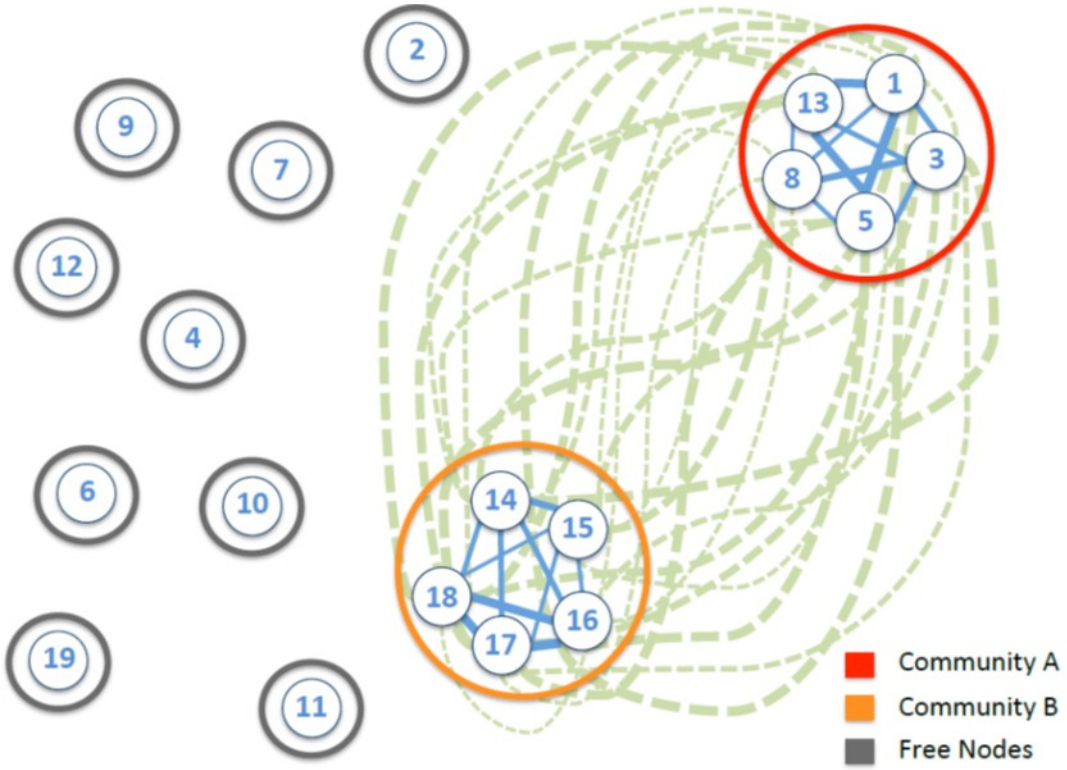

Figure 8: Schematic of the bipartition covariance network. Blue and green edges indicate positive and negative covariances between bipartitions, respectively. Line weights correspond to the magnitude of the covariances. Colored circles circumscribe the nodes assigned to each of the detected communities.

References

Gori, Kevin, Tomasz Suchan, Nadir Alvarez, Nick Goldman, and Christophe Dessimoz. 2016. "Clustering Genes of Common Evolutionary History." *Molecular Biology and Evolution* 33 (6). Oxford University Press: 1590–1605. doi:10.1093/molbev/msw038.

Lewitus, Eric, and Helene Morlon. 2016. "Characterizing and Comparing Phylogenies from their Laplacian Spectrum." *Systematic Biology* 65 (3). Oxford University Press: 495–507. doi:10.1093/sysbio/syv116.

Stockham, Cara, Li-San Wang, and Tandy Warnow. 2002. "Statistically based postprocessing of phylogenetic analysis by clustering." *Bioinformatics (Oxford, England)* 18 Suppl 1: S285–93. http://www.ncbi.nlm.nih.gov/pubmed/12169558.