GeNetwork:

A program to facilitate individual-based network analyses on genetic data

Last updated: June 15, 2012

by:

Timothy R. Frasier
Department of Biology & Forensic Sciences Program
Saint Mary's University
923 Robie Street
Halifax, Nova Scotia B3H 3C3
Canada

E-mail: timothy.frasier@smu.ca

Contents

1 Overview

GeNetwork is a program written in C that provides an important "middle step" for conducting individual-based network analyses on genetic data, and for testing hypotheses regarding the resulting networks. Briefly, GeNetwork can create pairwise matrices of relatedness or allele sharing in a format that can subsequently be analyzed using the **igraph** library in the R statistical environment (Csárdi & Nepusz 2006). Once the user has obtained the desired characteristics of their network, they can test hypotheses of their data using the simulation functions in GeNetwork. These generate simulated genotypes based on the user's allele frequency file, on which pairwise matrices of relatedness or allele sharing are created. These simulated data can then be analyzed with igraph to test hypotheses about the observed data. A complete walk-through of this process is provided later in this manual.

2 Legal Stuff

I tested GeNetwork extensively throughout its development, and subsequently tested it using several data sets upon completion. It appears to be functioning properly, and I am currently using it to analyze data from projects that I am working on. However, I cannot guarantee that it does not contain any errors, or that its results will always be reliable.

3 Installation

GeNetwork can be run on LINUX, MAC, or Windows operating systems. With LINUX and MAC systems, you should run the program from the command line, even though it may appear to be working properly when you click on the icon. For some reason (I don't know why yet) it does not work properly when you click on the icon in these systems. However, clicking on the icon works fine with Windows. For all three systems you can download and install an executable file (and associated files). You can also download the source code and compile it yourself on any system.

3.1 Windows

Unzip the zipped folder.

The new folder, called "GeNetwork_1.0_Windows", will contain:

1. The executable file
2. A folder containing example files
3. Other files necessary for the program to run (don't move or delete them!)

Put any relevant infiles into the same folder as the program.

Launch the program by double-clicking the executable file.

3.2 Mac OS X

To unzip the file, just double-click on it. This will make a folder called "GeNetwork_1.0_Mac".

Navigate into that folder through the command line. There will be a folder called "Example_File", and a file called "GeNetwork"

Put any relevant infiles into the same folder as the program.

The program should work properly by typing:
`./GeNetwork`

If that doesn't work, turn this file into an executable by typing:
`>chmod a+x GeNetwork`

You can now start the program by typing:
`>./GeNetwork`

3.3 Linux

To unzip the file, type:
`>gunzip GeNetwork_1.0_Linux.tar.gz`

Then:
`>tar -xvf GeNetwork_1.0_Linux.tar`

This will make a folder called GeNetwork_1.0_Linux.

Navigate into that folder through the command line:
`>cd GeNetwork_1.0_Linux`

There will be a folder called "Example_Files", and a file called "GeNetwork"

Turn this file into an executable by typing:
`>chmod a+x GeNetwork`

Put any relevant infiles into the same folder as the program.

You can now start the program by typing:
`>./GeNetwork`

4 Infiles

There are only two infiles that you need for using GeNetwork: an allele frequency file and a genotype file. The allele frequency file is needed for all analyses, and the genotype file is only needed for analyses of your observed data. As with all programs, these infiles must be in a specific format for the program to work properly. Both file types should be saved as tab-delimited text files.

4.1 Allele Frequency File

GeNetwork can estimate allele frequencies for you (option #1 in the main menu) and format them appropriately for subsequent analyses. However, if you want to manually create your allele frequency file, there are a few rules. Your allele frequency file should contain one column for each locus, with each column containing the allele frequencies for that locus. The loci MUST be in the same order as they occur in your genotypes, meaning that the first column of allele frequencies should represent the locus that occurs first in your genotypes, and so on. A zero (0) needs to be present at the top of each column, and you cannot include locus names. ALL columns must have values for the same number of rows. This means that the number of rows will equal the number of alleles in your most polymorphic locus. You MUST fill these rows with integers > 1 as place holders for those loci with fewer alleles. The format of the example allele frequency file supplied with the program is below. It contains data for 3 loci. The first locus has 7 alleles, the second locus has 5 alleles, and the third locus has 8 alleles. In this case, 5 was used as a placeholder.

0	0	0
0.0556	0.250	0.111
0.167	0.250	0.167
0.111	0.100	0.111
0.111	0.250	0.0556
0.167	0.150	0.111
0.278	5	0.167
0.111	5	0.111
5	5	0.167

4.2 Genotype File

The format needed for the genotype file should be the format that you typically have your data in anyway, with one column of individuals IDs, followed by two columns for each locus you used (one column for each allele). Individuals IDs can be text, numbers, or any combination of both, but they MUST have 19 characters or less. You cannot include a row of locus names in your genotype file, and ALL missing data MUST be recorded as zero (0). The format of the example genotype file supplied with the program is below. It contains data for 10 individuals typed at 3 loci.

Homer	100	102	203	203	187	191
Marge	102	104	205	207	189	193
Bart	100	104	203	207	0	0
Lisa	102	102	203	205	189	191
Maggie	100	102	203	205	187	189
Peter	80	82	205	213	203	209
Lois	84	86	213	219	211	213
Chris	0	0	205	219	203	213
Meg	82	84	213	213	209	211
Stewie	82	86	213	219	209	213

5 Analyses of Observed Data

GeNetwork can create pairwise matrices of your genotypes based on relatedness or allele sharing. Both approaches should obviously produce similar results, but there are different situations where one is preferable over the other, so both were included. The outputs are comma-delimited pairwise matrices that are "square", meaning that the pairwise values are given (and the same) above and below the diagonal.

5.1 Relatedness

This calculation of relatedness is based on the method described in Li et al. (1993), with each locus weighted using the method described in Lynch & Ritland (1999) and Van de Casteele et al. (2001). This method was chosen out of the many available approaches for calculating relatedness because it is unbiased, it is never undefined, and it consistently performs well in a variety of situations (and often out-performs all other estimators) (Van de Casteele et al. 2001; Wang 2002; Krützen et al. 2003). For creating the pairwise matrices, **GeNetwork truncates relatedness values at zero (0)**, meaning that any values < 0 are just given the value of zero. This is because negative values cannot be handled in network analyses, and negative relatedness values are not biologically meaningful anyway (for estimating actual relatedness). The equation for relatedness at each locus (l) is:

$$r_{xy}(l) = \frac{S_{xy} - S_o}{1 - S_o}$$

Where $S_{xy} = 1$ when genotype $x = ii$ and genotype $y = ii$, or when $x = ij$ and $y = ij$. $S_{xy} = 0.75$ when $x = ii$ and $y = ij$ or vice versa. $S_{xy} = 0.5$ when $x = ij$ and $y = ik$. $S_{xy} = 0$ when $x = ij$ and $y = kl$, where i, j, k, and l represent different alleles.

$$S_o = \sum_{i=1}^{n} p_i^2 (2 - p_i)$$

Where p_i is the population allele frequency for allele i. Multilocus relatedness values can be obtained by multiplying r_{xy} for each locus by the weight for that locus, summing this value across loci, and then dividing by the sum of all weights used. The weight for each locus is calculated based on the number of alleles in each locus.

$$w(l) = \frac{n_j - 1}{\sum n_j - 1}$$

Where n_j is the number of alleles at locus j.

So, the total relatedness is:

$$r_{xy} = \frac{1}{W} \sum w(l) r_{xy}(l)$$

Where W is the sum of the weights for all loci used.

Let's walk through an example, using the first five individuals in the example genotype file, and the data from the example allele frequency file (both provided above). Note that the weights will have to be calculated differently for those comparisons where individuals are missing data, to ensure that only data for loci being compared are used.

S_o for locus 1 = 0.0060 + 0.051 + 0.023 + 0.023 + 0.051 + 0.13 + 0.023 = **0.31**
S_o for locus 2 = 0.11 + 0.11 + 0.019 + 0.11 + 0.042 = **0.39**
S_o for locus 3 = 0.023 + 0.051 + 0.023 + 0.0060 + 0.023 + 0.051 + 0.023 + 0.051 = **0.25**

Weight for locus 1 = 6/19 = **0.32**
Weight for locus 2 = 4/19 = **0.21**

Weight for locus 3 = 7/19 = **0.37**

Total weights (W) = 0.89

Results:

Pair	S_{xy} locus 1	S_{xy} locus 2	S_{xy} locus 3	r_{xy} locus 1	r_{xy} locus 2	r_{xy} locus 3	r-value	truncated
Homer & Marge	0.5	0	0	0.27	-0.64	-0.34	-0.19	0
Homer & Bart	0.5	0.75	NA	0.27	0.59	NA	0.40	0.40
Homer & Lisa	0.75	0.75	0.5	0.64	0.59	0.33	0.50	0.50
Homer & Maggie	1	0.75	0.5	1	0.59	0.33	0.63	0.63
Marge & Bart	0.5	0.5	NA	0.27	0.18	NA	0.24	0.24
Marge & Lisa	0.75	0.5	0.5	0.64	0.18	0.33	0.40	0.40
Marge & Maggie	0.5	0.5	0.5	0.27	0.18	0.33	0.28	0.28
Bart & Lisa	0	0.5	NA	-0.45	0.18	NA	-0.20	0
Bart & Maggie	0.5	0.5	NA	-.27	0.18	NA	0.24	0.24
Lisa & Maggie	0.75	1	0.5	0.64	1	0.33	0.60	0.60

5.2 Allele Sharing

Another way to quantify how similar two genotypes are is by simply calculating the proportion of alleles that they share, which is the number of alleles shared divided by the number of alleles compared. This metric has the desirable properties of ranging from 0 (no alleles shared) to 1 (all alleles shared). At any locus, two individuals can share 0, 1, or 2 alleles. One feature of this calculation is that it does not require allele frequency information. This is good if your sample sizes are small, which could result in your allele frequency estimates being inaccurate and would lead to biased relatedness values. However, if you have decent allele frequency estimates, then weighting allele sharing by allele frequencies (e.g. calculating relatedness) will likely provide more valuable information on how more/less similar two genotypes are above and beyond "background" allele sharing in the population.

Let's walk through the same example as above (using the first 5 genotypes in the example genotype file) and calculate allele sharing. The table below shows the number of alleles shared (abbreviated as "AS") and the number of alleles compared (abbreviated as "AC") for each locus, as well as the overall value of allele sharing.

Pair	AS locus 1	AS locus 2	AS locus 3	AC locus 1	AC locus 2	AC locus 3	allele sharing
Homer & Marge	1	0	0	2	2	2	0.167
Homer & Bart	1	1	NA	2	2	0	0.500
Homer & Lisa	1	1	1	2	2	2	0.500
Homer & Maggie	2	1	1	2	2	2	0.667
Marge & Bart	1	1	NA	2	2	0	0.500
Marge & Lisa	1	1	1	2	2	2	0.500
Marge & Maggie	1	1	1	2	2	2	0.500
Bart & Lisa	0	1	NA	2	2	0	0.250
Bart & Maggie	1	1	NA	2	2	0	0.500
Lisa & Maggie	1	2	1	2	2	2	0.667

6 Simulations

A network can be created with the output of the above two options, and different metrics of that network can be calculated, using igraph. However, obtaining the data on your observed network is just the first step. What is still needed is a way to test hypotheses about that network. Of course, the best way to do this depends on the hypothesis being tested. However, I imagine that the common hypothesis that users will be testing on their network is "do I see more clustering in my observed data than would be expected for a data set of random individuals given my samples sizes and locus information". To test this hypothesis (or others based on a similar premise), GeNetwork has a simulation function.

The program allows the user to simulate a given number of unrelated individuals based on the allele frequency file, and then create pairwise matrices of relatedness or allele sharing based on those simulated data. The program allows you to easily do this many times (e.g. for 1,000 iterations), which can be used for hypothesis testing. Each matrix of simulated data is written in a separate outfile (named "out1", "out2", etc.). At the moment you can **perform up to 1,000 iterations**. Please let me know if this is limiting your work, and I can change the code accordingly.

For example, suppose that you have a data set of 100 samples genotyped at 10 loci. You then create a network based on relatedness for your data set. When you visualize this network, you see that your samples clearly fall out into two unlinked clusters, which suggests that there is strong subdivision within your population (or that you have samples from more than one population). Suppose that you have calculated the modularity (see below) for your network, and it is 0.782. You now want to see if this is higher (if your network is more clustered) than expected if your samples represent one interbreeding population, given your allele frequencies and sample sizes. You use GeNetwork to generate 1,000 simulated data sets, each of 100 individuals. Therefore, each iteration is one realization of your data set under the null hypothesis of no structuring (e.g. one interbreeding population). You calculate the modularity for these 1,000 simulated data sets, and find that only 3 have a modularity greater than or equal to your observed data. Thus, your observed data shows significantly more clustering than expected if it represents one interbreeding population, with $p < 0.004$ (you observed fewer than 4 iterations—out of 1,000—with a modularity equal to or greater than the observed data).

Note: this is just *one* of the types of analyses that you can conduct combining GeNetwork with igraph. However, there are many many very informative analyses that you could conduct to learn about, and test hypotheses, on your data. Indeed, network analyses provide, what I think, is the most powerful tool for assessing population genetic data. I encourage you to learn igraph, and explore the ways that it can be useful for your population genetic work.

7 Walk-Through With Example Data

I realize that network analyses, and graph theory, are likely new topics for most people interested in using GeNetwork. I also realize that many people are not familiar with the R statistical environment. Therefore, here I will walk through *all* of the steps in conducting one type of analysis of genetic data, using the example files that are provided with GeNetwork. I do not describe all of the commands in detail, but more information about them can be found in the igraph manual. Lastly, at the end of this manual I provide what (in my opinion) are good starting points for users who are interested in learning more about network analyses and R.

There are many programs available for visualizing networks. I use Cytoscape, which is a powerful and open source program. Therefore, this example will also walk through how to visualize your data using Cytoscape.

7.1 Getting Started with GeNetwork

Let's build a network based on the relatedness values of the data in the example infiles. This can be done by launching GeNetwork, and choosing the appropriate options (option 2 in this case). Below, the questions from GeNetwork are in blue, and the response (what you would type) is in red. Note that all relevant infiles need to be in the same folder as the program.

```
How many alleles do you have in your most polymorphic locus (not including 0)?  8
How many loci did you use?  3
What is the name of your allele frequency file (including extension?)  frequencies.txt
How many individuals do you have in your genotype file?  10
What is the name of your genotype file (including extension?)  genotypes.txt
What do you want to call your outfile?  rel_out
```

Next, you need to copy your outfile, and paste it into the directory where R will look for it. The default is in your home folder, but you may have changed the path to something different.

7.2 Creating the Network in R

Prior to conducting these analyses, you need to install the igraph package. You can find information on how to do this here.

Below, all of the commands that are typed into R are given in blue.

First, we need to read our data into R. We can do this with the `read.table` command. Here, our infile is comma-delimited and has a row of header names, so we need to tell R all of this. Below is an example, and I have called our data in R "testdata".

```
>testdata <- read.table("rel_out", header=TRUE, sep=",")
```

Now, igraph needs our data to be formatted as a matrix in order to analyze it properly. The first thing we need to do to convert it properly is take the column header names, and make them also the row names (e.g. right now our data just has labels (individual names) across the top for the column, but these also need to be present for each row of the pair-wise matrix). This can be done with the following commands.

```
>labels <- cbind(colnames(testdata))
>row.names(testdata) <- labels
```

Lastly, we can convert our data to a matrix using the following command.

```
>testmatrix <- as.matrix(testdata)
```

Now, we need to create a graph (network) of our data. We can do this with the `graph.adjacency` command in igraph. Here, we want to create an "undirected" graph, because our relatedness values are symmetrical (e.g. they are the same in both directions between a pair of individuals). We also want the graph to be "weighted", meaning that the edge lengths are proportional to the relatedness values. Below are the commands, including loading the igraph library.

```
>library(igraph)
>testgraph <- graph.adjacency(testmatrix, mode=c("undirected"), weighted=TRUE, diag=FALSE)
>summary(testgraph)
```

The summary should tell you that your data has 10 vertices (nodes, which are the individuals), and 16 edges connecting them.

Now, we will move to visualizing the network, but you should keep R open because we will come back to it for analyzing the network. We are just going to peek at it now. To visualize your network, you need to organize the data appropriately and export from R to a saved file. Here, we will save the file as testnetwork.

```
>edges <- as.data.frame(get.edgelist(testgraph))
>weights <- E(testgraph)$weight
>testnetwork <- cbind(edges, weights)
>write.csv(testnetwork, "testnetwork.csv")
```

7.3 Visualizing Network in Cytoscape

Launch Cytoscape, and import your data by selecting File → Import → Network from Table (Text/MS Excel) (**Figure 1**).

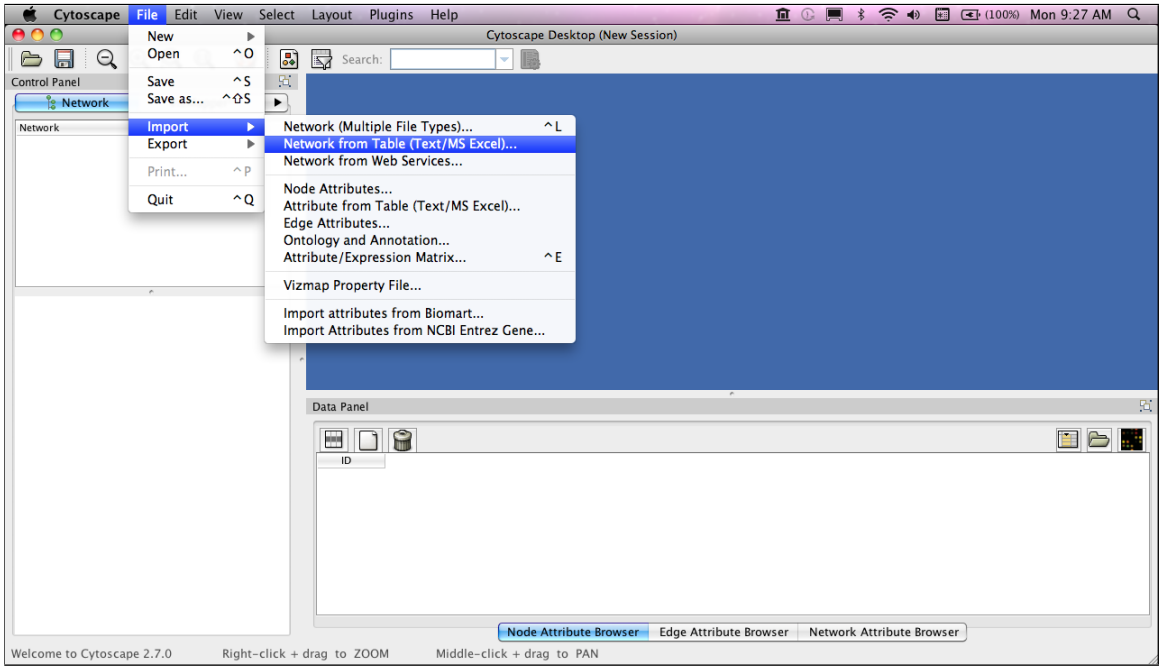

Figure 1: Loading data into Cytoscape

From this option, you can click on the "Select File(s)" button, which will allow you to browse to wherever your network file is. Once you have selected your file, a new bunch of options will appear. In these you want to select the following three: (1) check the box for "Show Text File Import Options" - this opens up more options; (2) check the box for "comma" as the delimiter; and (3) check the box that reads "Transfer first line as attribute names" (**Figure 2**).

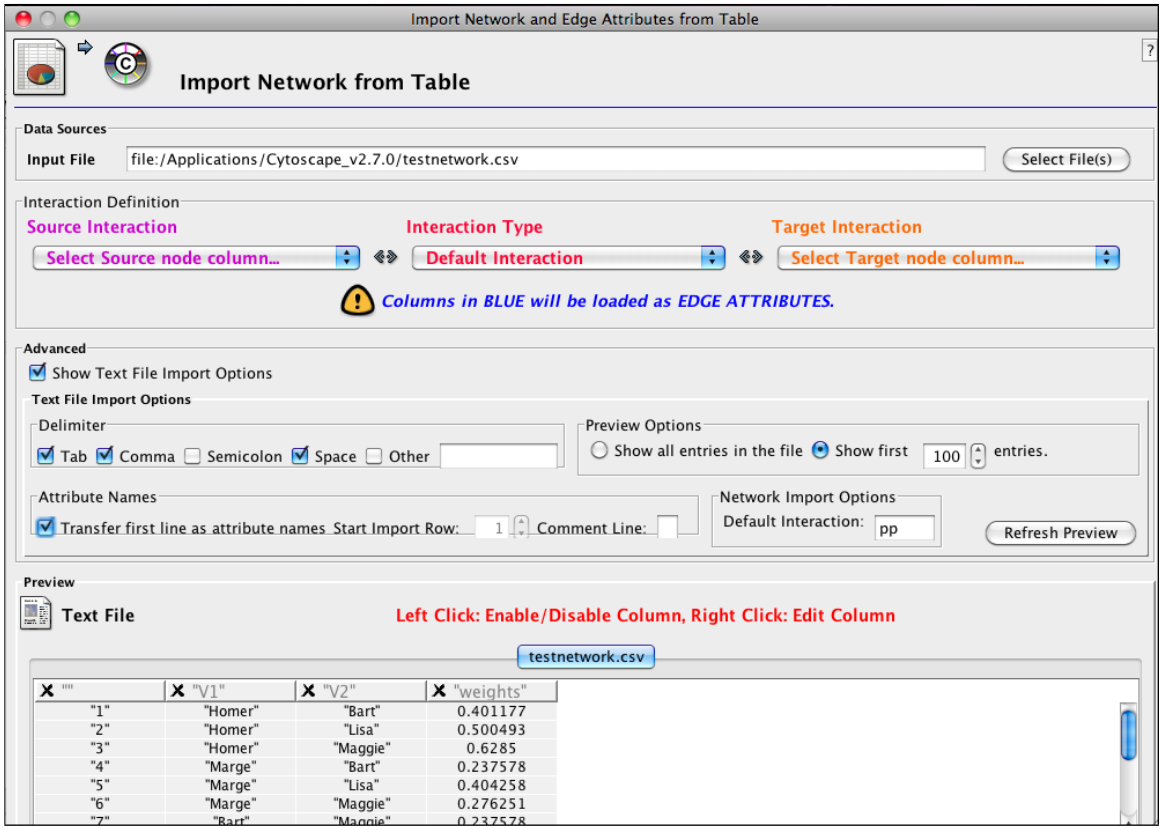

Figure 2: Appropriate check boxes for importing data

Now, you just need to tell Cytoscape which fields contain your nodes (individuals), and which one contains your edges. Select the "Source Interaction" drop-down menu and select "Column 2". Then select the "Interaction Type" drop-down menu, and select "Column 4". This is telling Cytoscape to use your edge weights as the "interaction" between nodes. Lastly, select the "Target Interaction" drop-down menu and select "Column 3". Next, click the "Import" button on the bottom of your menu - and your network should open up, and look something like this (**Figure 3**).

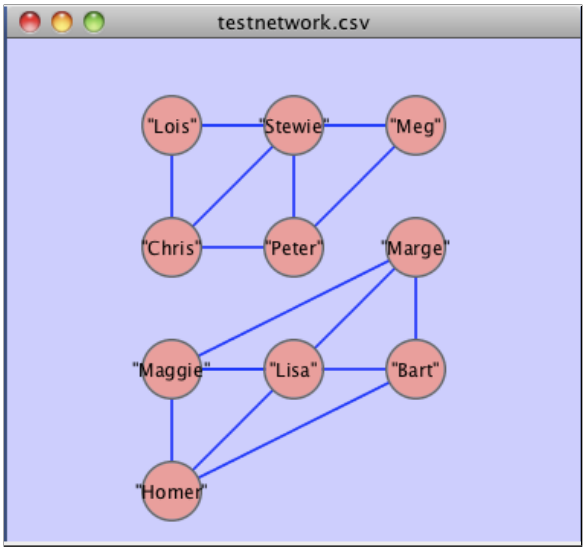

Figure 3: Initial network for test data.

This network isn't very informative yet. There are many ways to view your network. I find the most useful one is using the "spring embedded" option. With this, the program treats each edge like a "spring", that repels or attracts the different nodes relative to the weight of each edge. The program tries to find an arrangement of nodes that best fits the collected interactions between all nodes. To view your network in this way go to Layout → Cytoscape Layouts → Spring Embedded (**Figure 4**). The resulting graph should look like that in **Figure 5**, although I have played around a bit with colours.

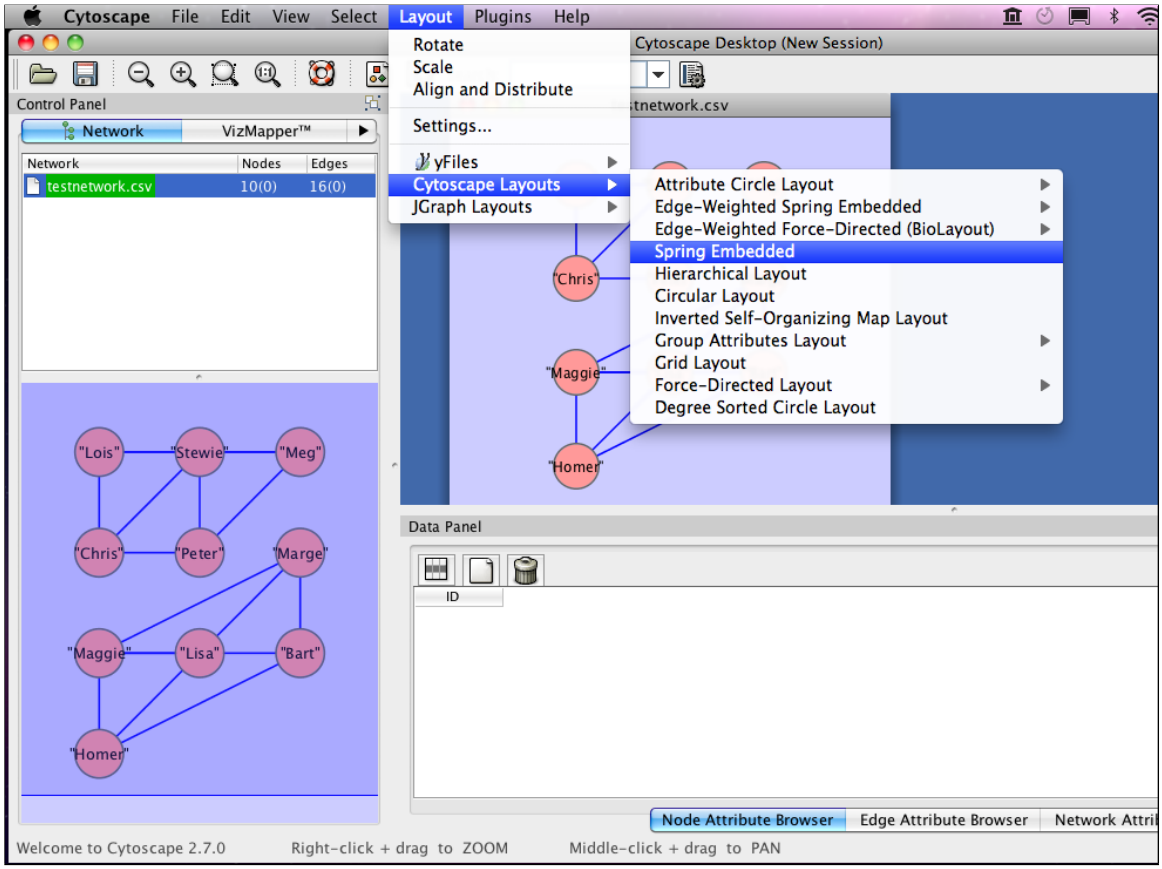

Figure 4: Selecting the "spring embedded" option in Cytoscape.

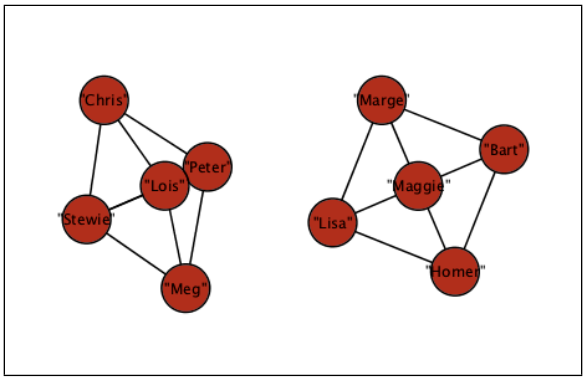

Figure 5: Resulting network.

From this network, it is very easy to see any patterns in your data. For example, our data clearly consists of 2 groups of fairly related individuals, and there seems to be little (or no) gene flow between them. Visualizing your data in this way can be very informative. You should note that you can view other options of your data as well. For example, if you have gender data, you could plot males as one colour and females as another, and compare the resulting statistics of each gender separately. There is tremendous power and versatility to this approach, but it takes a little while to learn the ropes.

7.4 Analyzing Your Data in R

Now that we've been able to visually (qualitatively) assess the data, let's see what sort of characteristics it has, from a quantitative perspective. Hopefully you did not close R before, and can just open that again and pick up where we left off.

Suppose that you are interested in how many clusters you have. Visually, it looked like 2, but we should assess that quantitatively. Newman (2006) reviewed and compared several cluster-detecting algorithms, as well as presented a new method that out-performed the others. His method can be implemented in igraph using the `eigenvector communities` function. We'll use two different methods for detecting clusters: first we'll use the standard "clusters" command in igraph, and then we'll use Newman's "eigenvector communities" approach.

Note that in our example, our clusters were not connected by any edges. This need not be the case. Indeed, in many genetic data sets, there will not be any completely separate clusters; instead, the individuals will be subdivided into relatively compact (tightly linked) clusters that are only moderately linked (few edges) to other such clusters. The approaches here work the same for detecting clusters in those data sets as well. Our example is just a very simple and clear data set, for initiation purposes.

For the first approach, we can use the `clusters` function in igraph. Below, the commands are in blue, and the results are in red.

```
>clusters(testgraph)
$membership
[1] 0 0 0 0 0 1 1 1 1 1
$csize
[1] 5 5
$no
[1] 2
```

The first line of the results ($membership) gives the cluster assignment for each individual (node). The first cluster is called cluster 0, the second is cluster 1, and so on. So, looking at these data, this approach assigned the first 5 individuals into cluster 0, and the second 5 individuals into cluster 1, which is consistent with our visualization of the network. The second line of the results ($csize) gives the size of each cluster. Here, cluster 0 has 5 individuals and cluster 1 has 5 individuals. The last line ($no) gives the number of clusters, which is 2. So, these data are completely consistent with our observations of the network.

In many situations, it would be helpful to combine these results with your individual names, so that you could have a list of each individual, along with which cluster they were assigned to. This is easy to do. First, instead of just writing the results to the screen, we want to write the results from the cluster analysis to a file: in this case, let's call it testout1. The resulting commands would be:

```
>testout1 <- clusters(testgraph)
>individuals <- labels
>clusters <- testout1$membership
>testclusters <- cbind(individuals, clusters)
>write.csv(testclusters, "testclusters.csv")
```

The resulting .csv file could then be opened in Excel, or other spreadsheet program, for visualization and analysis.

We can conduct a similar analysis using Newman's (2006) approach. The commands for this are:

```
>testout2 <- leading.eigenvector.community(testgraph, steps=1, options=igraph.arpack.default)
>community.le.to.membership(testout2$merges, testout2$steps, testout2$membership)
$membership
[1] 0 0 0 0 0 1 1 1 1 1
$csize
```

```
[1] 5 5
```

This approach resulted in the same results as before. This will not always be the case, however, and the eigenvector community approach should provide better resolution on more complicated data sets. A file combining individual names with their cluster assignments could be made the same as in the first example.

Lastly, for hypothesis testing, it would be good to have one metric that we could use that quantified how clustered our data are. This could then be compared to simulated data to test if we observe more clustering in our data set than expected if mating is random. There are a large number of network measures that are suitable for this. For now, we will just calculate the "modularity", which is based on Newman's eigenvector communities. Modularity is a quantification of how connected the identified clusters are to each other, which is what we are interested in for this example analysis. However, other metrics would be useful for testing other hypotheses and/or for other data sets. It will be important for you to think about which metric is best suited to your hypothesis, and choose it accordingly. The commands for this are:

```
>modularity(testgraph, testout2$membership, weights=E(testgraph)$weight)
[1] 0.4951481
```

Note that in addition to the file containing our network ("testgraph"), we also needed the file generated earlier to identify the eigenvector communities ("testout2"). Next we can compare these to values obtained through simulations, for hypothesis testing.

A third method for identifying clusters, and the one that I have found to perform far better than the other available methods, is the spinglass community method described by Reichardt & Bornholdt (2006). The commands for conducting this analysis are below. Note however that the values that I enter here are generic, and you should figure out what values are most appropriate for your study. You cannot actually use this approach on the test data, however, because this approach *can only be used on graphs that are connected* (i.e. the graph cannot be unconnected). However, this approach works very well on most genetic data sets that I have analyzed, which are likely to not have any unconnected nodes. The output from this approach will be similar to that from the `clusters` function.

```
>testout3 <- spinglass.community(testgraph, weights=E(testgraph)$weight, spins=25,
parupdate=FALSE, start.temp=1, stop.temp=0.1, cool.fact=0.99, update.rule=c("simple"),
gamma=1)
```

7.5 Conducting and Analyzing Simulations

To conduct the simulations, we will go back to GeNetwork, and create 100 simulated data sets, each containing 10 individuals (like our observed data set), based on our allele frequency data, and create matrices of these based on relatedness (option 4 in the main menu). The appropriate commands would be:

```
How many iterations do you want to perform?  100
How many individuals do you want to generate?)  10
How many alleles do you have in the most polymorphic locus (not including 0)?  8
How many loci did you use?  3
What is the name of your allele frequency file (including extension?)  frequencies.txt
```

The results will be 100 outfiles (named out1, out2, etc.). These should be copied and pasted into the folder where R will look for them (e.g. your home folder). Don't worry, you won't have to analyze each of these independently. Instead, we will have R step through them sequentially and analyze them, while recording the results in a single file.

The first thing you need to do is make a file containing a list of all of the infile names. This can easily be done in Excel, where one column can be made as a list that reads, "out1, out2, out3, etc.". For our example, this list should go up to "out100". This list should be saved as a .csv file, and be saved in the

folder where R will look for files. Here, I have called this file "infiles.csv". Again, these are the names of the simulated output files, and we will refer R to this list of names to identify which file to open next.

Next, we just need to type in the appropriate commands into R. Below, I will number the lines, and then explain what each line is doing. Note that in this example the `leading.eigenvector.community` function is being used to identify clusters. However, you can substitute this for the method of your choice when analyzing your data.

```
1.  >guide <- read.table("infiles.csv", header=FALSE, sep=",")
2.  >guide2 <- as.matrix(guide)
3.  >counter <- 0
4.  >module <- (1:100)
5.  >while (counter < 100) {
6.  + outfile <- read.table(guide2[counter+1], header=TRUE, sep=",")
7.  + labels <- cbind(colnames(outfile))
8.  + row.names(outfile) <- labels
9.  + data1 <- as.matrix(outfile)
10.  + data2 <- graph.adjacency(data1, mode=c("undirected"), weighted=TRUE, diag=FALSE)
11.  + simout <- leading.eigenvector.community(data2, steps=1, options=igraph.arpack.default)
12.  + module[counter+1] <- modularity(data2, simout$membership, weights=E(data2)$weight)
13.  + counter <- counter+1
14.  + }
```

Explanation:

1. Line1 is reading the list of simulation outfile names into R.
2. Line2 is making this file a matrix
3. Line 3 is initiating a counter (called "counter"), that we will use to tell R how many times to cycle through these commands. We will start it out as 0, and then increment it by 1.
4. Line 4 is creating a vector, called "module" that will hold 100 values. These will be the estimates of the modularity for each of the simulated data sets.
5. Line 5 is setting up a "while" loop. That is, as long as `counter` is less than 100, conduct the commands between the parentheses.
6. Line 6 is reading the first outfile from the simulations into R. The second time through the "while" loop, it will read the second outfile ("out2"), and so on.
7. Lines 7 & 8 are converting the column names to row names.
8. Line 9 is converting the data file into a matrix.
9. Line 10 is creating the network for the simulated data set.
10. Lines 11 & 12 are calculating the modularity of the simulated data set.
11. Line 13 is incrementing our counter: `counter`. Thus, the first time through the commands, it will analyze the first outfile, the second time through it will analyze the second outfile, and so one. Each time, it adds the results from those analyses to the vectors containing the results.
12. Line 14 closes the parentheses.

The last thing we need to do is see how our observed values compare with the "expected" values generated by the simulations. Specifically, we want to identify how many of the simulated data sets had a modularity higher than the observed value. This can be done as is shown below.

```
>sum(module >= 0.4951481)
[1] 0
```

This means that none of the simulated data sets had a modularity value equal to or greater than our observed value. Thus, $p < 0.01$ because we observed fewer than 1 out of 100 values that was higher than the

observed value.

It is easy to visualize these results in R. We can plot a histogram for the expected data sets using the `hist` function, and then add a line with our observed data to it. For the modularity data, the commands are below, and results shown in **Figure 6**.

```
>hist(module, xlab="Modularity")
>abline(v=0.4951481, col="red", lty="dashed")
```

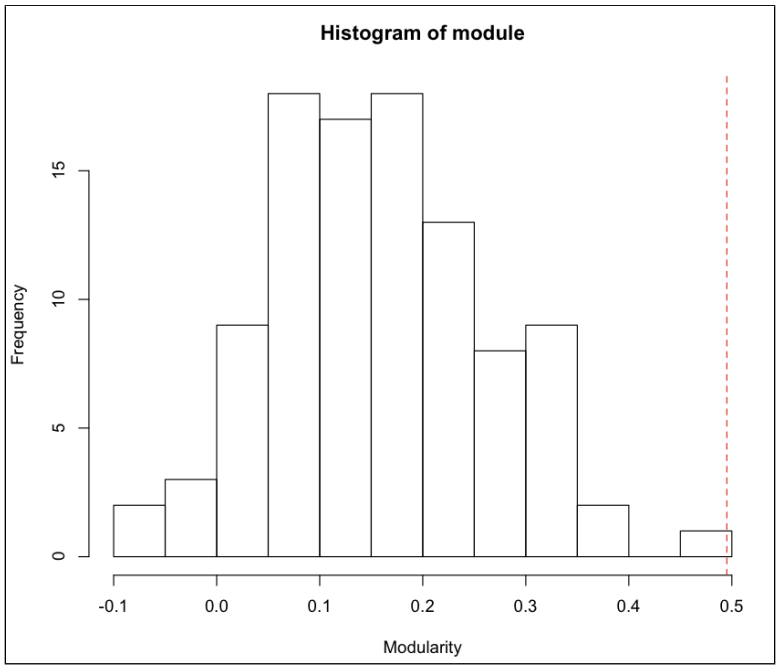

Figure 6: Observed (red line) and expected (histogram) values of modularity.

There are many metrics that are useful for quantifying the nature of your networks, and it is worth getting to know them. A good understanding of these is necessary in order to wisely choose which one(s) is most appropriate for testing the hypothesis that you are testing. It is a good idea to visually assess the network for your observed data, and identify what patterns you are interested in testing, identify what metric is best suited to testing such patterns, and test hypotheses based on that.

8 Suggested Reading

Learning network analyses can seem like a daunting task: requiring learning almost a new language and a new way to think about your data. However, it is extremely powerful, and worth the investment. For those users interested in learning more about network analyses, I highly recommend the book *Exploring Animal Social Networks* by Croft *et al.* 2008. It provides a very clear and concise introduction to network analyses for biologists. Also, in 2009 the journal *Behavioural Ecology and Sociobiology* published a special issue (**vol. 63(7)**) on network analyses, that provides very good information for new users. A few other useful review/summary papers include Costa *et al.* 2007, and Wey *et al.* 2008.

Becoming comfortable with R can also seem like a daunting task. Some people fear the command line, and feel at a loss when there are not icons to click, or drop-down menus to choose from. Do not fear the command line! R is extremely powerful, and is actually very easy (dare I say fun!?) to use once you get the hang of it. I have invested a significant amount of money in books on R, and have to say that no books even come close to the quality and clarity of Michael Crawley's *The R Book*. Even though it is expensive, it is an excellent deal, and is well worth it. For biologists, a very good introduction to R for biologists can

be found on Rodney Dyer's website (click here). There are also numerous helpful documents that you can find through internet searches, on of which can be found here.

9 References

Costa LDF, Rodrigues FA, Travieso G, Boas PRV (2007) Characterization of complex networks: A survey of measurements. *Advances in Physics* **56**: 167-242.

Crawley MJ (2007) *The R Book*. John Wiley & Sons.

Croft DP, James R, Krause J (2008) *Exploring Animal Social Networks*. Princeton University Press.

Csárdi G, Nepusz T (2006) The igraph software package for complex network research. *InterJournal Complex Systems*: 1695.

Krützen M, Sherwin WB, Connor RC, Barré LM, Van de Casteele T, Mann J, Brooks R (2003) Contrasting relatedness patterns in bottlenose dolphins (*Tursiops* sp.) with different alliance strategies. *Proceedings of the Royal Society of London* Series B **270**: 497-502.

Li CC, Weeks DE, Chakravarti A (1993) Similarity of DNA fingerprints due to chance and relatedness. *Human Heredity* **43**: 45-52.

Lynch M, Ritland K (1999) Estimation of pairwise relatedness with molecular markers. *Genetics* **152**: 1753-1766.

Newman MEJ (2006) Modularity and community structure in networks. *Proceedings of the National Academy of Sciences USA* **103**: 8577-8582.

Queller DC, Goodnight KF (1989) Estimating relatedness using genetic markers. *Evolution* **43**: 258-275.

Reichardt J, Bornholdt S (2006) Statistical mechanics of community detection. *Physical Review* **74**: 016110.

Van de Casteele T, Galbusera P, Matthysen E (2001) A comparison of microsatellite-based pairwise relatedness estimators. *Molecular Ecology* **10**: 1539-1549.

Wang J (2002) An estimator for pairwise relatedness using molecular markers. *Genetics* **160**: 1203-1215.

Wey T, Blumstein DT, Shen W, Jordán F (2008) Social network analysis of animal behaviour: a promising tool for the study of sociality. *Animal Behaviour* **75**: 333-344.