Social Network Analysis Tutorial: Part 1

Created by: Kim VanderWaal, May 8, 2015; Last updated: Feberuary 3, 2019

We will be working with a small subset of cattle movement data.

First, make sure that you have the proper packages installed and loaded. In R, anything that is writted after a # symbol are considered comments (not code)

```
#install.packages("igraph")
#install.packages("RColorBrewer")
#install.packages("EpiContactTrace")
#install.packages("ggplot2")
#install.packages("reshape")
library(igraph)
library(EpiContactTrace)
library(RColorBrewer)
library(ggplot2)
```

This tutorial includes some .Rdata formatted data files. The files are also provided in the SNA folder in .csv format so you can see how to organize your data for SNA.

Load the .Rdata files using the following code:

```
#load(file.choose())  #load the edgelist_farms.Rdata
#load(file.choose())  #load attr.farms.Rdata

#If your data is in a csv file, you can read it in with the following code (not necesssa
ry for this tutorial).
#read.csv(file.choose(),strings.As.Factors=F)
```

Making a igraph object from the edgelist.

Graph.data.frame is the EASIEST way to get your data into network form. The first two columns are the sender and reciever. The remaining columns are edge-level attributes (i.e., weights)

```
head(edges)
```

```
##   Origin Dest       Date breeding.cows steers heifers calves batch.size
## 1      1   25 2008-12-05             0     43       0      0         43
## 2      1   25 2008-12-05             0     41       0      0         41
## 3      3   88 2009-07-25             0      0       0     24         24
## 4      2   25 2008-12-05             0     42       0      0         42
## 5      5   13 2008-12-15             0      0      12      0         12
## 6      5    7 2008-12-15             0      0       8      0          8
```

```
# create an igraph object
net <- graph.data.frame(edges,directed=T)
net
```

```
## IGRAPH DN-- 120 356 --
## + attr: name (v/c), Date (e/c), breeding.cows (e/n), steers (e/n),
##   heifers (e/n), calves (e/n), batch.size (e/n)
```

The next key thing to know how to do in igraph is how add node-level attributes from another file. The attr1 file is a table of attributes.

```
head(attr1)
```

```
##   farm.id      type size    long      lat
## 1       1 Fattening  354 6193999 567613.7
## 2       3     Dairy  203 6181075 559135.0
## 3       2 Fattening  250 6196082 562319.0
## 4       5     Dairy  994 6207385 474646.3
## 5      15     Dairy  544 6221101 413726.6
## 6      13     Dairy   87 6205887 421180.9
```

```
#add production type to each node.
```

The V(net) notation indicates a vertex attribute of the specified network. V(net)$name indicates what we would like to name the attribute. The “V” stands for vertex, which is another name for node. The match command ensures that the production type added to each node is the correct one for that node by matching the name of the node to the corresponding farm.id in the attribute file

```
V(net)$type <- as.character(attr1$type[match(V(net)$name,attr1$farm.id)])
```

note that “type” now appears as a node-level (v/c) a attribute. Also note that edge-level attributes are denoted as (e/n)

```
net
```

```
## IGRAPH DN-B 120 356 --
## + attr: name (v/c), type (v/c), Date (e/c), breeding.cows (e/n),
##   steers (e/n), heifers (e/n), calves (e/n), batch.size (e/n)
```

```
#add herd size
V(net)$farm.size <- attr1$size[match(V(net)$name,attr1$farm.id)]
```

Plotting a network

Let’s first create an attribute for node color

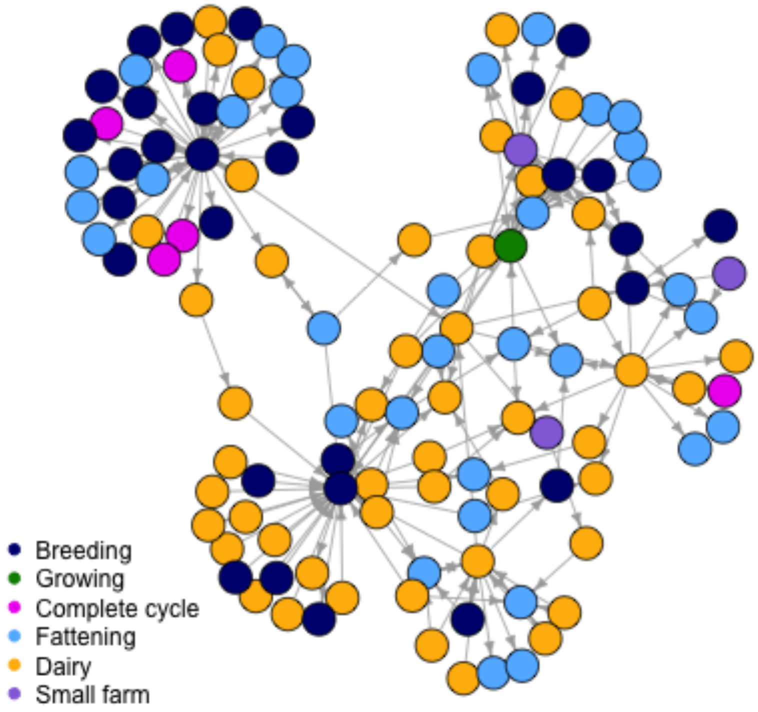
```
#Prepare custom colors for plotting by creating an attribute for vertex (node) color
V(net)$color <- V(net)$type
V(net)$color <- gsub("Breeding","navy",V(net)$color)
V(net)$color <- gsub("Complete cycle","magenta2",V(net)$color)
V(net)$color <- gsub("Growing","green4",V(net)$color)
V(net)$color <- gsub("Fattening","steelblue1",V(net)$color)
V(net)$color <- gsub("Dairy","darkgoldenrod1",V(net)$color)
V(net)$color <- gsub("Small farm","mediumpurple",V(net)$color)

#You could similarly specify shapes by creating a shape attribute.
```

Plot the network using the following command, and create a legend.

```
plot.igraph(simplify(net),
            layout=layout.fruchterman.reingold,
            vertex.label=NA,
            vertex.color=V(net)$color,
            vertex.size=10,
            edge.arrow.size=.5)

#make a legend
legend("bottomleft",
       legend=c("Breeding","Growing","Complete cycle","Fattening","Dairy","Small farm"),
       col=c("navy","green4","magenta2","steelblue1","darkgoldenrod1","mediumpurple"),
       pch=19,cex=1,bty="n")
```



plot of chunk unnamed-chunk-8

Assignment question 1. Try any four different layouts, such as fruchterman.reingold, kamada.kawai, etc. These can be entered in the "layout=" as part of the plot.igraph statement. Export these (you can do so in the menu of the Rstudio "Plots" window), and paste them into a word doc (You can also complete this assignment in R Markdown, if preferred). Describe in a few sentences how the different layouts convey different impressions about the structure of the network.

```
?layout.fruchterman.reingold  #to see different layout options
```

On your own, try making the vertex size scale with the log of the farm size. Do you notice any patterns about where large farms are located within the network?

Edge criteria

What if we want to change the way edges are determined? Right now, there is a row for every movement, but not all movements consist of the same types of animals.

```
head(edges)
```

```
##   Origin Dest       Date breeding.cows steers heifers calves batch.size
## 1      1   25 2008-12-05             0     43       0      0         43
## 2      1   25 2008-12-05             0     41       0      0         41
## 3      3   88 2009-07-25             0      0       0     24         24
## 4      2   25 2008-12-05             0     42       0      0         42
## 5      5   13 2008-12-15             0      0      12      0         12
## 6      5    7 2008-12-15             0      0       8      0          8
```

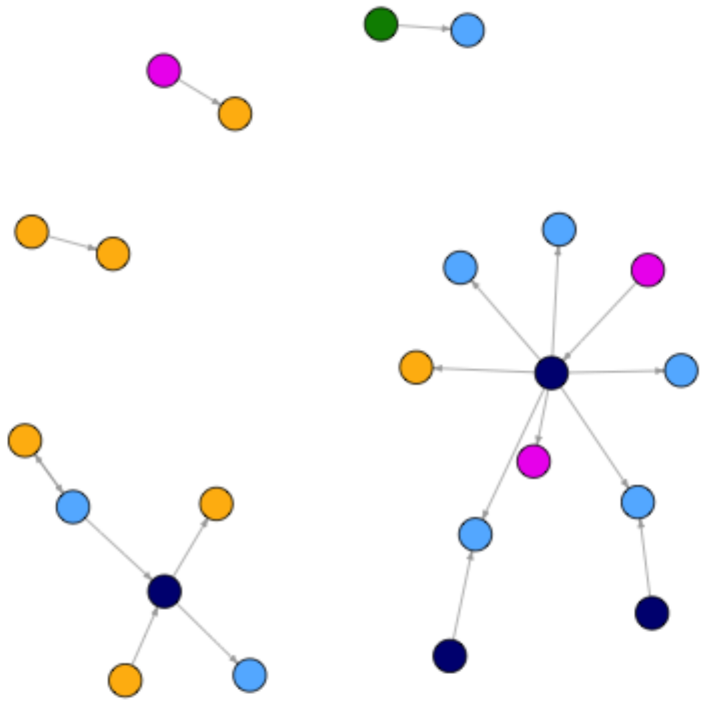
Let’s recreate our network so only movements of >50 animals are considered

```
#Filter edges
e2 <- subset(edges,batch.size>50)

net.c <- graph.data.frame(e2)

#re-add attributes and change colors
V(net.c)$farm.size <- (attr1$size[match(V(net.c)$name,attr1$farm.id)]  )
V(net.c)$type <- as.character(attr1$type[match(V(net.c)$name,attr1$farm.id)]  )
V(net.c)$color <- V(net.c)$type
V(net.c)$color <- gsub("Breeding","navy",V(net.c)$color)
V(net.c)$color <- gsub("Complete cycle","magenta2",V(net.c)$color)
V(net.c)$color <- gsub("Growing","green4",V(net.c)$color)
V(net.c)$color <- gsub("Fattening","steelblue1",V(net.c)$color)
V(net.c)$color <- gsub("Dairy","darkgoldenrod1",V(net.c)$color)
V(net.c)$color <- gsub("Small farm","mediumpurple",V(net.c)$color)

#You could similarly specify shapes by creating a shape attribute.
plot.igraph(simplify(net.c),
            layout=layout.fruchterman.reingold,
            vertex.label=NA,
            vertex.color=V(net.c)$color,
            vertex.size=10,
            edge.arrow.size=.3)
```



plot of chunk unnamed-chunk-11

How does the network change?

Re-try this with filtering only for movements of heifers. How does the network change?

Subgraphs

How to make a subset of the full graph. This is useful for contact tracing or for “zooming in” on areas of the network of interest.

An “ego network” is a type of subgraph that is created based on the all the connections of a single node that are within k steps in the network

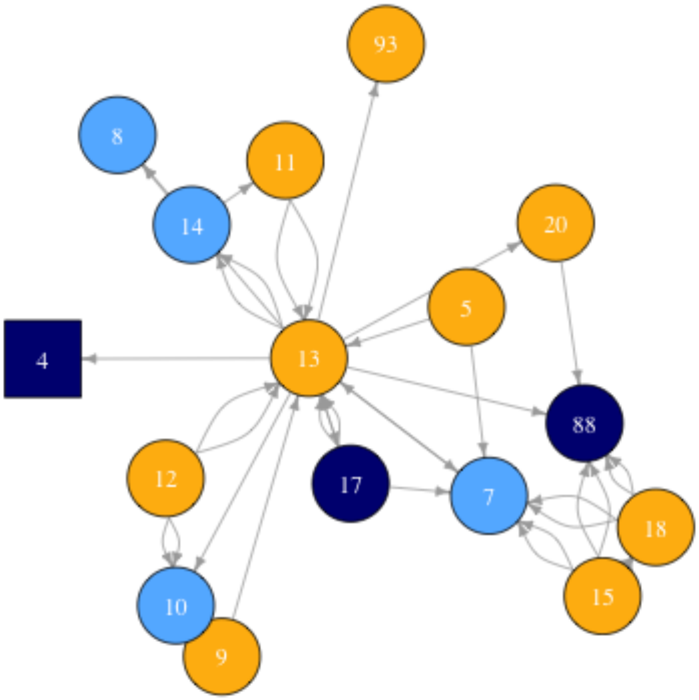
```
?graph.neighborhood
net3 <- graph.neighborhood(net,2,nodes=4)  #find the second-order neighborhood of node #
4

#Output is in list format.  So extract the network from the first position in the list
net3 <- net3[[1]]
```

Plot the network

```
#highlight the focal node
V(net3)$shape <- "circle"
V(net3)$shape[V(net3)$name==4] <- "square"

plot.igraph(net3,
            vertex.color=V(net3)$color,
            vertex.label.color="white",
            vertex.shape=V(net3)$shape,
            vertex.size=25,
            vertex.shape=V(net3)$shape,
            edge.arrow.size=.5,
            layout=layout.fruchterman.reingold)
```



plot of chunk unnamed-chunk-13

Node-level statistics

Let’s calculate some node-level centrality metrics. Try calculating degree, betweenness, and closeness on the subgraph net3

```
degree(net3,mode="out")
```

```
##  5 15 13 18  9  4 11 12 14 17 20  7 88 93  8 10
##  2  7 10  4 12  0  2  4  2  4  1  1  0  0  0  0
```

Degree is defined as the number of (unique) contacts that a node engages in. Check the out-degree of the focal node (#11). Is it right?

This is an important consideration for using edgelists. If its possible for the same edge to appear more than once, then the degree command will return the total number of edges, not the total number of unique contacts. This should always be checked when performing network analyses.

Use the simplify command to eliminate duplicate edges

```
net4 <- simplify(net3)
degree(net4,mode="out")
```

```
##  5 15 13 18  9  4 11 12 14 17 20  7 88 93  8 10
##  2  3  9  2  2  0  1  2  2  2  1  1  0  0  0  0
```

Replot net4. What changes? Is the degree properly calculated now?

Try betweenness and closeness.

```
betweenness(net4)
```

```
##  5 15 13 18  9  4 11 12 14 17 20  7 88 93  8 10
##  0  0 76  0  0  0  8  0  8  0  0 18  0  0  0  0
```

```
closeness(net4)
```

```
##           5          15          13          18           9           4 
## 0.011764706 0.012820513 0.010989011 0.010752688 0.011764706 0.004166667 
##          11          12          14          17          20           7 
## 0.010101010 0.011764706 0.009523810 0.010101010 0.004444444 0.010000000 
##          88          93           8          10 
## 0.004166667 0.004166667 0.004166667 0.004166667
```

Betweenness and closeness values can be dependent on the overall size of the network. So, to compare across networks, its usually a good idea to use “normalized” values to account for network size.

```
betweenness(net4,normalized=T)
```

Make a table to summarize these data. This table can now be written to a .csv file using write.csv()

```
data <- data.frame(name=V(net4)$name,
                   in.deg = degree(net4,mode="in"),
                   out.deg = degree(net4,mode="out"),
                   betw = betweenness(net4),
                   size = V(net4)$farm.size)
```

Assignment 2: Construct a data frame as shown above for the full network, and additionally include closeness. Use the “pairs” command to visualize correlations between the centrality metrics in the “data” object. Hint: use ?pairs to learn more about this command. Also calculate Spearman’s correlations amongst the centrality metrics (function: cor). Include this output in your assignment.

Find the shortest path between two nodes

Sometimes, we may want to find the distance (# of steps in the network) between two farms. This could be useful when determining how likely a transmission chain is to occur between two nodes. Let’s explore this using our simplified sub-network, net4. Calculate shortest paths between select nodes (say node 5 to 4)

```
shortest.paths(net4,v="5",to="4",mode="out")
```

```
##   4
## 5 2
```

Try doing this for mode “out” and “in”. What is the difference and why? What happens if you do not specifiy a mode

Important: For directed networks, you should always specify a “mode” for path-based analsyes (including betweenness and shortest paths). Otherwise, igraph will ignore the direction of the arrows, and find paths that should be impossible based on the direction of movement.

We can also calculate shortest paths among a list of nodes (i.e., all infected farms)

```
v.list <- c("15","17","8","9")
shortest.paths(net4,v= v.list,to= v.list,mode="in")
```

```
##     15  17   8   9
## 15   0 Inf Inf Inf
## 17   3   0 Inf   2
## 8    3   2   0   2
## 9  Inf Inf Inf   0
```

“Inf” means that a path between two nodes does not exist.

Sometimes, we don’t just want the pathlength, but would like to utilize weighted data (i.e., how many animals moved?). You can specify weights of edges.

```
shortest.paths(net3,v=v.list,to=v.list,mode="in",weights=1/E(net3)$batch.size)
```

```
##           15       17   8        9
## 15 0.0000000      Inf Inf      Inf
## 17 0.3666667 0.000000 Inf 1.166667
## 8  0.3250000 0.162037   0 1.125000
## 9        Inf      Inf Inf 0.000000
```

We use the inverse as the weight because analyses relying on weights are designed to use weight as the "COST" of traversing an edge. Low cost means that the edge is easier to traverse (i.e., transmission is more likely). Because we want transmission to be more likely in movements with more cattle, we take the inverse of the batch.size so that movements with large numbers of animals are low cost for transmission

We can also calculate the distance between all farms in a network by not specifying any nodes

```
s <- shortest.paths(net4,mode="in")
```

Matrices can be hard to read, so change to a data.frame

```
library(reshape2)
s1 <- melt(s)
head(s1)
```

```
##   Var1 Var2 value
## 1    5    5     0
## 2   15    5   Inf
## 3   13    5     1
## 4   18    5   Inf
## 5    9    5   Inf
## 6    4    5     2
```

Var1 and Var2 are the node neames, and value is the distance between nodes.

Network summary statistics

Let's calculate some network summary statistics using our full network (net). You may want to replot the full network so that you can look at it as you go along

Try running the following

Network-level statistics should usually be calculated with a simplified network

Graph density: Number of edges occurring / total number possible

```
net2 <- simplify(net)
graph.density(net2)
```

```
## [1] 0.01428571
```

Transitivity: A measure of clustering. Number of triangle / Number of triplets in network.

```
transitivity(net2)
```

```
## [1] 0.08549715
```

Tracing contacts

This is useful for contact tracing in dynamic networks (i.e., each edge occurs at a specific data in time). The package “EpiContactTrace” is useful because it traces contacts, taking into account the temporal sequence in which edges occur. igraph’s subgraph functions do not do this.

To use this package, edgelists must be in specific format with specific column names (“source”,“origin”,“t”). Here is some code to modify our current edgelist into the proper format

If you are uploading from a csv file, make sure that the date in the file is recorded a yyyy-mm-dd. If not, change it in excel before importing into R.

```
edges.trace <- subset(edges,select=c(Origin,Dest,Date))#select only the columns we want
names(edges.trace)<- c("source","destination","t")#rename the columns
edges.trace$source <- as.integer(edges.trace$source)#make the source and destination into integers
edges.trace$destination <- as.integer(edges.trace$destination)
```

Run the contact tracing. This will trace all contacts within the previous two years for node 13

```
tr <- Trace(edges.trace,root=13,tEnd="2013-01-01",days=365*2)
tr
```

```
## Root: 13
##
## <<< In contacts <<<
## In begin date: 2011-01-02
## In end date:   2013-01-01
## In days: 730
## In degree: 3
## Ingoing contact chain: 5
##
## 13 <-- 12
## 13 <-- 17
## 13 <--  7
##         7 <-- 15
##         7 <-- 17
##         7 <-- 18
##
## >>> Out contacts >>>
## Out begin date: 2011-01-02
## Out end date:   2013-01-01
## Out days: 730
## Out degree: 5
## Outgoing contact chain: 18
##
##  13 -->  14
##          14 -->   8
##  13 -->  20
##          20 --> 102
##                 102 --> 100
##                 102 --> 111
##                         111 --> 113
##                 102 -->  82
##          20 --> 116
##                 116 --> 112
##                 116 --> 113
##                         113 --> 111
##                 116 --> 120
##                 116 -->  32
##                 116 -->  34
##          20 -->  22
##                  22 -->  21
##          20 -->  33
##          20 -->  88
##  13 -->   8
##  13 -->  88
##  13 -->  93
```

There are a lot of other cool functionalites, which we will not explore extensively, but here are a few

```
#NetworkSummary(tr)     #get a one-line summary of the trace
#NetworkStructure(tr)   #essentially gives the information needed to construct an edgeli
st
```

Finding communities

Return to original net object for this. There are many ways to find clusters of nodes that are more interlinked with one another then with nodes outside their cluster. We will briefly explore one such method.

```
#Create a community object.
c <- walktrap.community(net,weights=1/E(net)$batch.size,steps=7)
plot(c, net, vertex.size=5, edge.arrow.size=.25,vertex.label=NA)
```

```
## Graph community structure calculated with the walktrap algorithm
## Number of communities (best split): 10
## Modularity (best split): 0.7105439
## Membership vector:
##   1   3   2   5  15  13  23  18   9   4  11  35  12  19  14  17  30  20
##   6   5   6   1   7   1  10   7   1   2   1   2   1   7   1   1   5   1
##  28  51  44  46  48  45  37  40  41  55  38  53 111  67  70 105 119  95
##   1   3   6   3   3   3   2   6   5   3   6   3   4   3   3   2   9   5
## 113  91 115 116  96  71  76  79  78  97  85  81  69  66  43 114  73  36
##   4   3   4   2   2   3   3   8   3   5   5   5   3   3   5   4   3   3
##  27  24  32  22   7  31  42  52  99  87  90  84  92 112  75  98 108  89
##   6   5   2  10   1   1   5   3   4   5   5   5   5   2   3   4   2   5
##  88  56  68 102  80 109  93 106  26 107  33  25   8  10  21  29  49 120
##   5   3   3   4   8   4   1   7   7   4   1   6   1   1  10   2   3   2
##  64  57  61  63  72  74  77  60  50  65  54  58  47  62  59 104  39 101
##   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   9   5   4
##  34 103 118  16   6 100  86  83  94 110  82 117
##   2   2   2  10   2   4   5   5   5   4   4   4
```

The Modularity score listed at the top of the output is a measure ranging from 0-1 of how good the community structure (how many edges are directed within a community versus between communities). Typically, values of 0.2 - 0.7 are considered indicative of moderate to strong community structure.

The Membership vector in the ouptput lists the community ID number for each node in your network.

Let’s re-plot this community structure in another way. Reset color palette so there are more default colors in the palette. RColorBrewer package helps select contrasting colors. You can google RColorBrewer to get a list of possible color schemes.

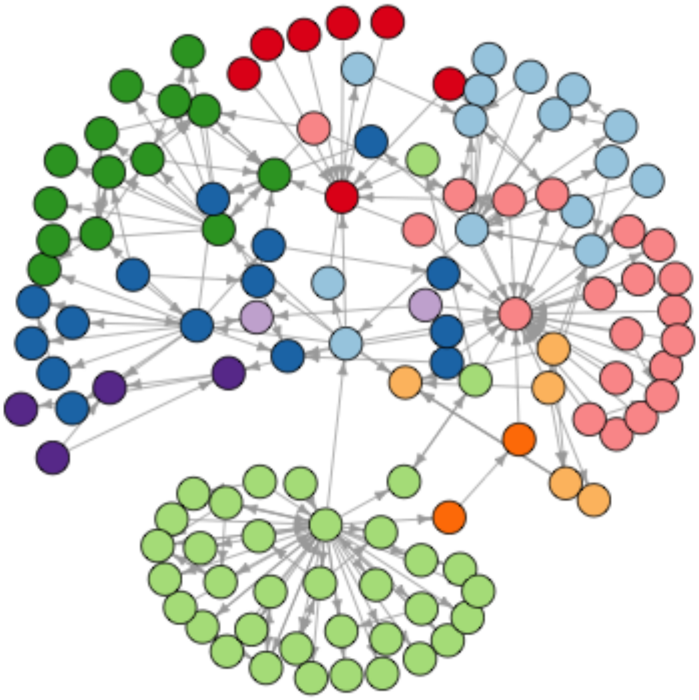
```
palette(brewer.pal(length(c),"Paired"))
```

Also, specifying the velow code will preserve the layout each time you plot the network

```
lay.constant <- layout.kamada.kawai(net)

#add membership as a node attribute
V(net)$community <- c$membership

plot.igraph(simplify(net),
            layout=lay.constant,
            vertex.label=NA,
            vertex.color=c$membership,
            vertex.size=10,
            edge.arrow.size=.25)
```



plot of chunk unnamed-chunk-29

Assignment question 3: Change the color palette and try different layouts to see which one seems to capture underlying community structure the best. Show at least two plots in your assignment and discuss how the layout influences interpretation.

Plotting networks with ggplot

The GGalley package is a newer network visualiztion package that is based on ggplot2. As all ggplot2-related packages, this package can produce nicer and more easily customized visualizations.

```
#install.packages("GGally")
library(GGally)
```

The package depends on also having several other packages loaded: ggplot2, network, sna, and intergraph. The network and sna packages are similar to igraph and are used for network manipulation and analysis. They store networks as R objects in the “network” class, whereas igraph stored networks as “igraph” objects. However, the intergraph package can be used to translate your igraph object to a network object.

```
library(network)
library(sna)
library(ggplot2)
library(intergraph)
```

Convert our network to a network object

```
net.convert <- asNetwork(simplify(net))
```

Plot our network in ggplot

```
ggnet2(net.convert,  color="type")
#the alpha statement creates transparent nodes
ggnet2(net.convert,  color="type",alpha=.75)
```

The color scheme is not the best, so let’s change it. Go to https://www.r-graph-gallery.com/38-rcolorbrewers-palettes/ (https://www.r-graph-gallery.com/38-rcolorbrewers-palettes/) if you want to view a full set of possible palettes.

```
ggnet2(net.convert,  color="community", palette="Paired")
```

You can also introduce some node centrality calcuations right within the command line

```
ggnet2(net.convert,  color="community", palette="Paired",
    size="degree")
```

Using ggplot for networks has alot of advantages, and many cool costumizations. However, we won’t go into depth on those here. Additional resources: https://briatte.github.io/ggnet/ (https://briatte.github.io/ggnet/)

Statistics with networks

Network k-test Tutorial

Please cite the following when using th network k-test for research or educational purposes: VanderWaal, K. L., Enns, E. A., Picasso, C., Packer, C. & Craft, M. E. 2016. Evaluating empirical contact networks as potential transmission pathways for infectious diseases. Journal of the Royal Society Interface, 13, 20160166.

We will be working with a set of data related to social interactions among ground squirresl and infection data for Cryptosporidium.

First, make sure that you have the proper packages installed and loaded. In R, anything that is writted after a # symbol are considered comments (not code).

It also is a good idea to set your working directory to a known location using the “session” drop down menu. Make sure the files downloaded as part of the tutorial are in thew orking directory

```
#install.packages("igraph")

library(igraph)
```

Read in your source file. This should be located in your working directory. This will read in all the functions associated with this “package”

```
source("Cluster.test.source.parallel.R")
```

Making a igraph object for analysis.

An attribute file and an edgelist file are required for running the k-test. Both should be saved in .csv format. See side panel for details on format.

Edgelist file: a .csv file listing all edges/links in the network. One column should be titled ‘v1’ and another column should be titled ‘v2’. Names used in the edgelist should match the nodes listed in the attribute file. For directed networks, the sender should be in the first column, and the reciever in the second. Any additional columns will be interpreted as edge weights or edge attributes by igraph.

Read in the edgelist file (edges.A.csv):

```
e.A <- read.csv(file.choose(),stringsAsFactors=F)  #read edges.A.csv or edges.B.csv
```

Attribute file: a .csv file listing all nodes in the network (inlcuding isolates). One column should be titled ‘name’ and another column should be titled ‘state’. The names of columns should have no capitalization. Names can be either written as characters or numbers. The state column represents infection status: 0 = Non-infected; 1 = Infected

```
attr1 <- read.csv(file.choose(),stringsAsFactors=F) #read in attributes file, attr_squirrel.
```

Ultimately, this function requires your data to be in the iGraph format with a vertex attribute called “name” (as a character), and “state” (as an integer). All non-infected nodes should be zeros, and infected nodes should be 1.

Now, make an igraph object and add vertex attribute “state.” Here, we assume that the network in undirected. For directed networks, use “directed = T”.

Note that making the network from the edgelist in this way will eliminate any isolates. To include isolates, the vertices= argument can be used to specify a data.frame (i.e., such as the attribute dataframe) that includes all vertices including isolates.

```
netA <- graph.data.frame(e.A,directed=F) #directed=F if this is an undirected network
```

Add attribute for state as an integer.Non-infecteds should be 0, infecteds should be >0

```
V(netA)$state <- as.integer(attr1$state[match(V(netA)$name,attr1$name)] )
```

```
plot(simplify(netA),layout=layout.fruchterman.reingold,vertex.label=NA,vertex.color=V(ne
tA)$state)
```

Running the analysis

Now, run the k-test using the following code. You will get a few warning messages, but those do not affect the running of the analysis.

```
test2 <- k.test(netA,k=1,type="both",bin="full",iterations=20)

test2
```

The output is a list of three objects: test2[[1]] is a dataframe with the output of the permutations test2[[2]] is the observed mean and median k-statistics and the p-values. The ***mean*** should be used for interpretation of results test2[[3]] is a density plot of the null distribution of the k-statistic with the observed k-statistic in red

Your test can be saved as an .Rdata file

```
save(test2,file="test2.Rdata")

#to re-load results
load("test2.Rdata")
```

Assignment question 4: Load the data files “edges.lion.csv” and “attr.lions.csv”. These respresent an edgelist representing a contact network of lion prides in the Serengeti, and an attribute file that includes a state variable represnting which prides became infected during the first half of a canine distemper outbreak in 1993.

Run the k-test for this outbreak. Export the network plot you made above as well as density graph generated by the k-test. Record the p-value for the mean. Please put these into a word doc (or R Markdown), and, in your own words, interpret the k-test output. What is the null hypothesis being tested? What does the blue shaded area and red line represent in the plot? And how do you interpret the p-value?

A few notes about the options availabe

k=

Refers to how large of a neighborhood is considered for the k-statistic. k=1 refers to only direct connections of the infected node, where as k=2 refers to all connections within two steps. It is not advisable to do k=2 for very large networks

iterations=

How many permutations to perform. Default should be 1000

type=

The type option should be set to 'all' if using an undirected network. For directed networks, the k-statistic can be calculated for incoming paths, outgoing paths, or both. Using the 'all' command with a directed network will result in the edge directions being ignored

bin=

The bin option is set to 'full' by default, indicated a complete randomization of the pattern of infected nodes. For large networks, node infection statuses are randomized within each quartile of either degree or neighborhood size (all nodes with two steps) to preserve the overall connectivty levels of infected nodes while randomizing who is connected with whom

node.type=

The k-statistic can be calculated for nodes only of a given type if you have a vertex attribute called "node.type." Node.type attributes (e.g, Male or Female) can be added from an attribute table in the same way that we added the "state" attribute. If node.type = "Male", then all infected nodes (male or female) within k steps will be calculated for the infected male nodes (females are not considered focal).

par=

This indicates whether the user desires the permutations to be calculated via parallel processing. Default is true.

Other useful resources

https://sites.google.com/site/daishizuka/toolkits (https://sites.google.com/site/daishizuka/toolkits)