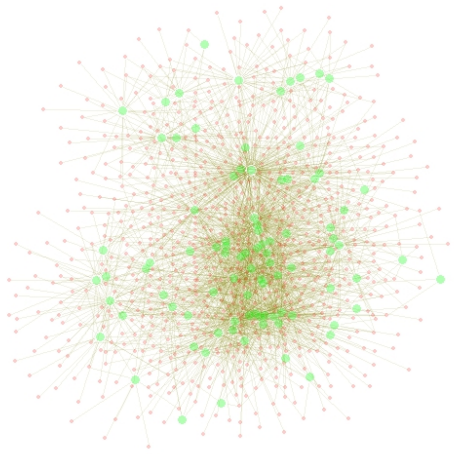
Affiliation Data



A network can consist of different 'classes' of nodes. For example, a two-mode network might consist of people (the first mode) and groups in which they are members (the second mode). Another very common example of two-mode network data consists of users on a particular website who communicate in the same forum thread.

Here's a short example of this kind of data. Run this in R for yourself - just copy an paste into the command line or into a script and it will generate a dataframe that we can use for illustrative purposes:

```
df <- data.frame( person =
c('Sam','Sam','Sam','Greg','Tom','Tom','Tom','Mary','Mary'), group =
c('a','b','c','a','b','c','d','b','d'), stringsAsFactors = F)
```

Here's what the data looks like if we run the command above in R, and then type df, which prints the data:

```
> df
  person group
1    Sam     a
2    Sam     b
3    Sam     c
4   Greg     a
5    Tom     b
6    Tom     c
7    Tom     d
8   Mary     b
9   Mary     d
```

Fast, efficient two-mode to one-mode conversion in R

Suppose we wish to analyze or visualize how the people are connected directly - that is, what if we want the network of people where a tie between two people is present if they are both members of the same group? We need to perform a two-mode to one-mode conversion.

This is easy to do using the matrix algerbra functions included in R. But first, you need to restructure your network data in matrix format. This is really easy to do in R, just use the table function and then create a matrix:

```
M = as.matrix( table(df) )
```

By the way, don't let the notation confuse you here. What we are doing is nesting the table function within the as.matrix function, just doing essentially two lines worth of coding in one line of code. It's equivalent to:

```
m = table( df )
M = as.matrix( m )
```

If you are using the network or sna packages, a network object be coerced via `as.matrix(your-network)`; with the igraph package use `get.adjacency(your-network)`.

Relevant Online Resources
R project website
get R and relevant sna packages here, plus many other resources
R - getting started
list of online books, short guides, and reference cards to get started using R
Intro to Social Network Methods
an excellent and free online book on social network analysis
statnet website
a series of R packages for network analysis, ideal for network/regression models
igraph website
an excellent package for working with network data and network visualization

We will either convert to the mode represented by the columns or by the rows.

To get the one-mode representation of ties between rows (people in our example), multiply the matrix by its transpose. Note that you must use the matrix-multiplication operator `%*%` rather than a simple astrisk. The R code is:

```
Mrow = M %*% t(M)
```

Mrow will be the one-mode matrix formed by the row entities (people with ties to each other if they are in the same group, in our example). Here's what it looks like:

```
> Mrow
      person
person Greg Mary Sam Tom
  Greg    1    0   1   0
  Mary    0    2   1   2
  Sam     1    1   3   2
  Tom     0    2   2   3
```

To get the one-mode matrix formed by the column entities (i.e. the number of people the enter the following command:

```
Mcol = t(M) %*% M
```

And the resulting co-membership matrix is as follows:

```
> Mcol
     group
group a b c d
    a 2 1 1 0
    b 1 3 2 2
    c 1 2 2 1
    d 0 2 1 2
```

Although we've used a very small network for our example, this code is highly extensible to the analysis of larger networks with R. One could read-in a huge network of, for instance genes and classifiers, convert the data to memory-efficient sparse matrix representations using the Matrix or SpareM packages, and still use the same matrix algerbra operation above. Both packages have routines to handle matrix transposition and multiplication.

Analysis of Two Mode Data and Mobility

Let's work with some actual affiliation data, collected by Dan McFarland on student extracurricular affiliations. It's a longitudinal data set, with 3 waves - 1996, 1997, 1998. It consists of students (anonymized) and the student organizations in which they are members (e.g. National Honor Society, wrestling team, cheerleading squad, etc.).

What we'll do is to read in the data, make some mode conversions, visualize the networks in various ways, compute some centrality measures, and then compute transition probabilities (the probability that a member of one group will stay a member of the same group or become a member of a new group

```
# Load the "igraph" library
library("igraph")

# (1) Read in the data files, NA data objects coded as "na"
magact96 = read.delim("http://stanford.edu/~messing/mag_act96.txt",
    na.strings = "na")
magact97 = read.delim("http://stanford.edu/~messing/mag_act97.txt",
    na.strings = "na")
magact98 = read.delim("http://stanford.edu/~messing/mag_act98.txt",
    na.strings = "na")
```

Missing data is coded as "na" in this data, which is why we gave R the command `na.strings = "na"`. These files consist of four columns of individual-level attributes (ID, gender, grade, race), then a bunch of group membership dummy variables (coded "1" for membership, "0" for no membership). We need to set aside the first four columns (which do not change from year to year).

```
magattrib = magact96[,1:4]

g96 = as.matrix(magact96[,-(1:4)]); row.names(g96) = magact96$ID.
g97 = as.matrix(magact97[,-(1:4)]); row.names(g97) = magact97$ID.
g98 = as.matrix(magact98[,-(1:4)]); row.names(g98) = magact98$ID.
```

By using the [,-(1:4)] index, we drop those columns so that we have a square incidence matrix for each year, and then tell R to set the row names of the matrix to the student's ID. Note that we need to keep the "." after ID in this dataset (because it's in the name of the variable).

Now we load these two-mode matrices into igraph:

```
i96 <- graph.incidence(g96, mode=c("all") )
i97 <- graph.incidence(g97, mode=c("all") )
i98 <- graph.incidence(g98, mode=c("all") )
```

Ploting two-mode networks

Now, let's plot these graphs. The igraph package has excellent plotting functionality that allows you to assign visual attributes to igraph objects before you plot. The alternative is to pass 20 or so arguments to the plot.igraph() function, which gets really messy.

Let's assign some attributes to our graph. First we set vertex attributes, making sure to make them slightly transparent by altering the gamma, using the `rgb(r,g,b,gamma)` function to set the color. This makes it much easier to look at a really crowded graph, which might look like a giant hairball otherwise.

You can read up on the RGB color model here.

Each node (or "vertex") object is accessible by calling `V(g)`, and you can call (or create) a node attribute by using the $ operator so that you call `V(g)$attribute`. Here's how to set the color attribute for a set of nodes in a graph object:

```
V(i96)$color[1:1295] = rgb(1,0,0,.5)
V(i96)$color[1296:1386] = rgb(0,1,0,.5)
```

Notice that we index the `V(g)$color` object by a seemingly arbitrary value, 1295. This marks the end of the student nodes, and 1296 is the first group node. You can view which nodes are which by typing `V(i96)`. R prints out a list of all the nodes in the graph, and those with a number are obviously different from those that consist of a group name.

Now we'll set some other graph attributes:

```
V(i96)$label = V(i96)$name
V(i96)$label.color = rgb(0,0,.2,.5)
V(i96)$label.cex = .4
V(i96)$size = 6
V(i96)$frame.color = NA
```

You can also set edge attributes. Here we'll make the edges nearly transparent and slightly yellow because there will be so many edges in this graph:

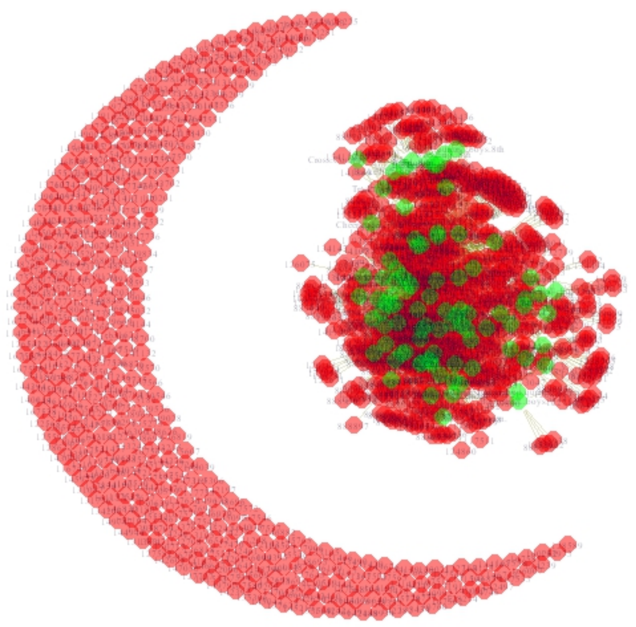
```
E(i96)$color = rgb(.5,.5,0,.2)
```

Now, we'll open a pdf "device" on which to plot. This is just a connection to a pdf file. Note that the code below will take a minute or two to execute (or longer if you have a pre- Intel dual-core processor).

```
pdf("i96.pdf")
plot(i96, layout=layout.fruchterman.reingold)
dev.off()
```

Note that we've used the Fruchterman-Reingold force-directed layout algorithm here. Generally speaking, the when you have a ton of edges, the Kamada-Kawai layout algorithm works well but, it can get really slow for networks with a lot of nodes. Also, for larger networks, layout.fruchterman.reingold.grid is faster, but can fail to produce a plot with any meaninful pattern if you have too many isolates, as is the case here. Experiment for yourself.

Here's what we get:

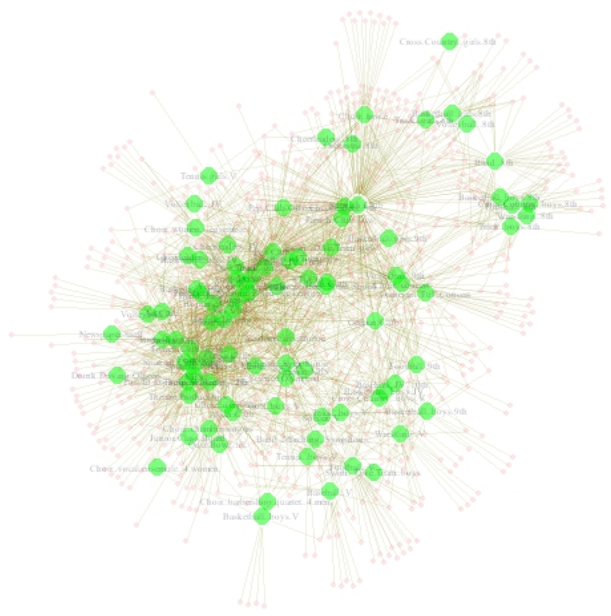


It's oddly reminiscent of a cresent and star, but impossible to read. Now, if you open the pdf output, you'll notice that you can zoom in on any part of the graph ad infinitum without losing any resolution. How is that possible in such a small file? It's possible because the pdf device output consists of data based on vectors: lines, polygons, circles, elipses, etc., each specified by a mathematical formula that your pdf program renders when you view it. Regular bitmap or jpeg picture output, on the other hand, consists of a pixel-coordinate mapping of the image in question, which is why you lose resolution when you zoom in on a digital photograph or a plot produced with most other programs.

Let's remove all of the isolates (the cresent), change a few aesthetic features, and replot. First, we'll remove isloates, by deleting all nodes with a degree of 0, meaning that they have zero edges. Then, we'll suppress labels for students and make their nodes smaller and more transparent. Then we'll make the edges more narrow more transparent. Then, we'll replot using various layout algorithms:

```
i96 = delete.vertices(i96, V(i96)[ degree(i96)==0 ])
V(i96)$label[1:857] = NA
V(i96)$color[1:857] =  rgb(1,0,0,.1)
V(i96)$size[1:857] = 2

E(i96)$width = .3
E(i96)$color = rgb(.5,.5,0,.1)

pdf("i96.2.pdf")
plot(i96, layout=layout.kamada.kawai)
dev.off()

pdf("i96.3.pdf")
plot(i96, layout=layout.fruchterman.reingold.grid)
dev.off()

pdf("i96.4.pdf")
plot(i96, layout=layout.fruchterman.reingold)
dev.off()
```

I personally prefer the Fruchterman-Reingold layout in this case. The nice thing about this layout is that it really emphasizes centrality--the nodes that are most central are nearly always placed in the middle of the plot. Here's what it looks like:



Very pretty, but you can't see which groups are which at this resolution. Zoom in on the pdf output, and you can see things pretty clearly.

Two mode to one mode data transformation

We've emphasized groups in this visualization so much, that we might want to just create a network consisting of group co-membership. First we need to create a new network object. We'll do that the same way for this network as for our example at the top of this page:

```
g96e = t(g96) %*% g96
g97e = t(g97) %*% g97
g98e = t(g98) %*% g98

i96e = graph.adjacency(g96e, mode = "undirected")
```

Now we need to tansform the graph so that multiple edges become an attribute (`E(g)$weight`) of each unique edge:

```
E(i96e)$weight <- count.multiple(i96e)
i96e <- simplify(i96e)
```

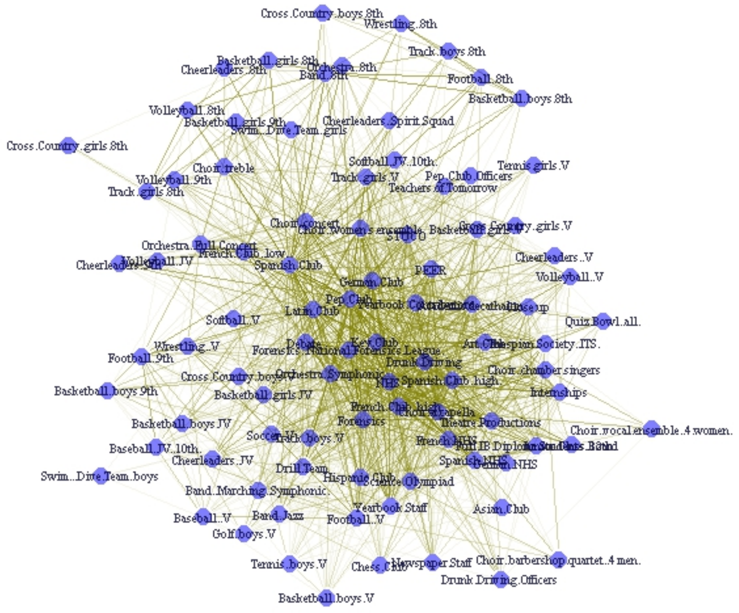
Now we'll set the other plotting parameters as we did above:

```
# Set vertex attributes
V(i96e)$label = V(i96e)$name
V(i96e)$label.color = rgb(0,0,.2,.8)
V(i96e)$label.cex = .6
V(i96e)$size = 6
V(i96e)$frame.color = NA
V(i96e)$color = rgb(0,0,1,.5)

# Set edge gamma according to edge weight
egam = (log(E(i96e)$weight)+.3)/max(log(E(i96e)$weight)+.3)
E(i96e)$color = rgb(.5,.5,0,egam)
```

We set edge gamma as a function of how many edges exist between two nodes, or in this case, how many students each group has in common. For illustrative purposes, let's compare how the Kamada-Kawai and Fruchterman-Reingold algorithms render this graph:

```
pdf("i96e.pdf")
plot(i96e, main = "layout.kamada.kawai", layout=layout.kamada.kawai)
plot(i96e, main = "layout.fruchterman.reingold",
layout=layout.fruchterman.reingold)
dev.off()
```

I like the Kamada-Kawai layout for this graph, because the center of the graph is too busy otherwise. And here's what the resulting plot looks like:



You can check out the difference between each layout yourself. Here's what the pdf output looks like. Page 1 shows the Kamada-Kawai layout and page 2 shows the Fruchterman Reingold layout.

Group overlap networks and plots

Now we might also be interested in the percent overlap between groups. Note that this will be a directed graph, because the percent overlap will not be symmetric across groups--for example, it may be that 3/4 of Spanish NHS members are in NHS, but only 1/8 of NHS members are in the Spanish NHS. We'll create this graph for all years in our data (though we could do it for one year only).

First we'll need to create a percent overlap graph. We start by dividing each row by the diagonal (this is really easy in R):

```
ol96 = g96e/diag(g96e)
ol97 = g97e/diag(g97e)
ol98 = g98e/diag(g98e)
```

Next, sum the matricies and set any NA cells (caused by dividing by zero in the step above) to zero:

```
magall = ol96 + ol97 + ol98
magall[is.na(magall)] = 0
```

Note that `magall` now consists of a percent overlap matrix, but because we've summed over 3 years, the maximun is now 3 instead of 1.

Let's compute average club size, by taking the mean across each value in each diagonal:

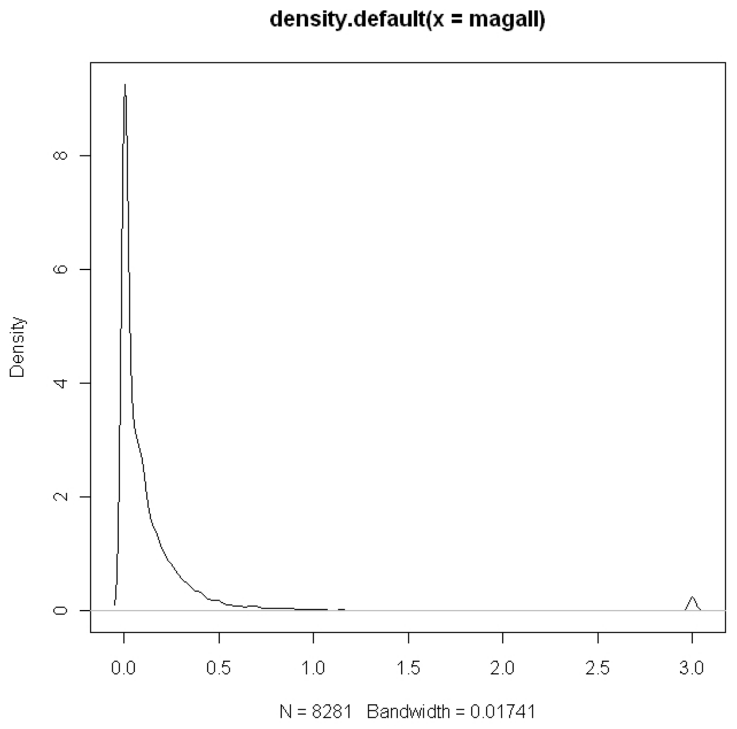
```
magdiag = apply(cbind(diag(g96e), diag(g97e), diag(g98e)), 1, mean )
```

Finally, we'll generate centrality measures for magall. When we create the igraph object from our matrix, we need to set `weighted=T` because otherwise igraph dichotomizes edges at 1. This can distort our centrality measures because now edges represent more than binary connections--they represent the percent of membership overlap.

```
magallg = graph.adjacency(magall, weighted=T)

# Degree
V(magallg)$degree = degree(magallg)

# Betweenness centrality
V(magallg)$btwcnt = betweenness(magallg)
```

Before we plot this, we should probably filter some of the edges, otherwise our graph will probably be too busy to make sense of visually. Take a look at the distribution of connection strength by plotting the density of the magall matrix:

```
plot(density(magall))
```



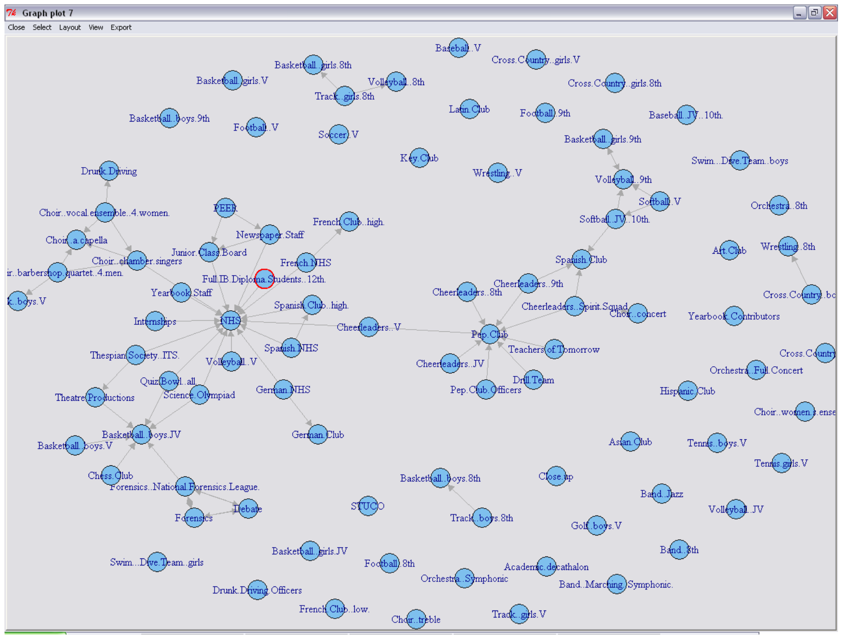
Nearly all of the edge weights are below 1--or in other words, the percent overlap for most clubs is less than 1/3. Let's filter at 1, so that an edge will consists of group overlap of more than 1/3 of the group's members in question.

```
magallgt1 = magall
magallgt1[magallgt1<1] = 0
magallggt1 = graph.adjacency(magallgt1, weighted=T)

# Removes loops:
magallggt1 <- simplify(magallggt1, remove.multiple=FALSE, remove.loops=TRUE)
```

Before we do anything else, we'll create a custom layout based on Fruchterman.-Ringold wherein we adjust the coordates by hand using the `tkplot` gui tool to make sure all of the labels are visible. This is very useful if you want to create a really sharp-looking network visualization for publication.

```
magallggt1$layout = layout.fruchterman.reingold(magallggt1)
V(magallggt1)$label = V(magallggt1)$name
tkplot(magallggt1)
```

Let the plot load, then maximize the window, and select to View -> Fit to Screen so that you get maximum resolution for this large graph. Now hand-place the nodes, making sure no labels overlap:



Pay special attention to whether the labels overlap (or might overlap if the font was bigger) along the vertical. Save the layout coordinates to the graph object:

```
magallggt1$layout = tkplot.getcoords(1)
```

We use "1" here because only if this was the first tkplot object you called. If you called tkplot a few times, use the last plot object. You can tell which object is visible because at the top of the `tkplot` interface, you'll see something like "Graph plot 1" or in the case of my screenshot above "Graph plot 7" (it was the seventh time I called tkplot).

```
# Set vertex attributes
V(magallggt1)$label = V(magallggt1)$name
V(magallggt1)$label.color = rgb(0,0,.2,.6)
V(magallggt1)$size = 6
V(magallggt1)$frame.color = NA
V(magallggt1)$color = rgb(0,0,1,.5)

# Set edge attributes
E(magallggt1)$arrow.size = .3

# Set edge gamma according to edge weight
egam = (E(magallggt1)$weight+.1)/max(E(magallggt1)$weight+.1)
E(magallggt1)$color = rgb(.5,.5,0,egam)
```

One thing that we can do with this graph is to set label size as a function of degree, which adds a "tag-cloud"-like element to the visualization:

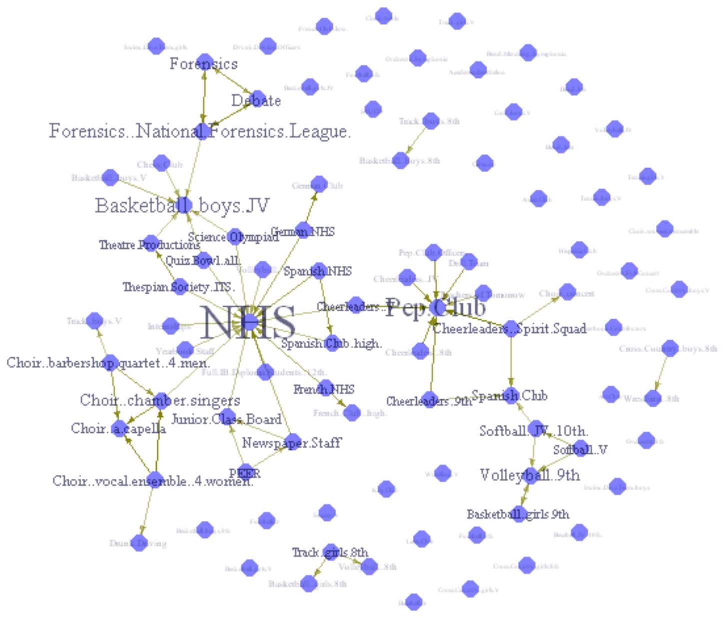
```
V(magallggt1)$label.cex = V(magallggt1)$degree/(max(V(magallggt1)$degree)/2)+ .3
#note, unfortunately one must play with the formula above to get the
#ratio just right
```

Let's plot the results:

```
pdf("magallggt1customlayout.pdf")
plot(magallggt1)
dev.off()
```

Note that we used the custom layout, which because we made part of the igraph object `magallggt1`, we did not need to specify in plot command.

Here's the pdf output, and here's what it looks like:



This visualization reveals much more information about our network than our cresent-star visualization.

I'll soon post more on transition probabilities and mobility in two mode networks, using this network as an example.