UNIVERSIDAD NACIONAL DE COLOMBIA SEDE BOGOTA
ANALISIS ESTADISTICO DE REDES SOCIALES.
Taller 3: Graficas y Afiliacion de Redes

Cesar A Prieto Sarmiento - ceprieto@unal.edu.co
Alejandro Urrego Lopez - aurrego@unal.edu.co

24 de marzo de 2024

1. **Sintetizar y replicar el Capítulo 6 de Luke (2015).**

Chapter 6
Advanced Network Graphics

One eye sees, the other feels. (Paul Klee)

As the previous two chapters demonstrate, both statnet and igraph have sophisticated plotting capabilities that can produce a very wide variety of network graphics. However, these plotting functions cannot meet all of the analytic or presentation needs. In particular, network scientists may wish to produce more specialized network graphics. Also, while statnet and igraph excel at producing high-quality publication ready network graphics, these graphics are static. Fortunately, developers have started exploring how to take network graphics and deliver them to web-based platforms where users can interact with the diagrams. This chapter explores a few of these more specialized network graphic techniques, as well as demonstrating how to produce some simple web-based interactive network diagrams.

6.1 Interactive Network Graphics

One of the useful features of many other network analysis packages such as UCINet and Pajek is the ability to produce network diagrams that are interactive at some level. For example, in Pajek a network visualization can be produced in a separate 'Draw' window, and then the user can interact with that window in various ways to edit or change the network graphic. These capabilities can be very useful for exploring the network, as well as fine-tuning a network graphic for subsequent dissemination.

Although R's programmatic framework allows for detailed control over all the elements of a network graphic, this is generally not made available to the user in an interactive way. There are a few exceptions to this, as well as some new packages that allow for creating interactive network diagrams that can be published to the web. In this section a few of these options are demonstrated.

6.1.1 Simple Interactive Networks in igraph

The igraph package includes the tkplot () function which supports simple interactive network plots through a Tk graphics window. Only some features of the network graphics can be modified. A typical use for this feature is to produce the interactive graphic, adjust the node positions to improve the network layout, save the node position coordinates and then use the coordinates to produce a final (noninteractive) network diagram. This work flow is illustrated below with the Bali network, see Chap. 8 for a more in-depth example with these data.

```
library(intergraph)
library(igraph)
```

```
data(Bali)
iBali <- asIgraph(Bali)
Coord <- tkplot(iBali, vertex.size=3,
vertex.label=V(iBali)$role,
vertex.color="darkgreen")
# Edit plot in Tk graphics window before
# running next two commands.
MCoords <- tkplot.getcoords(Coord)
plot(iBali, layout=MCoords, vertex.size=5,
vertex.label=NA, vertex.color="lightblue")
```

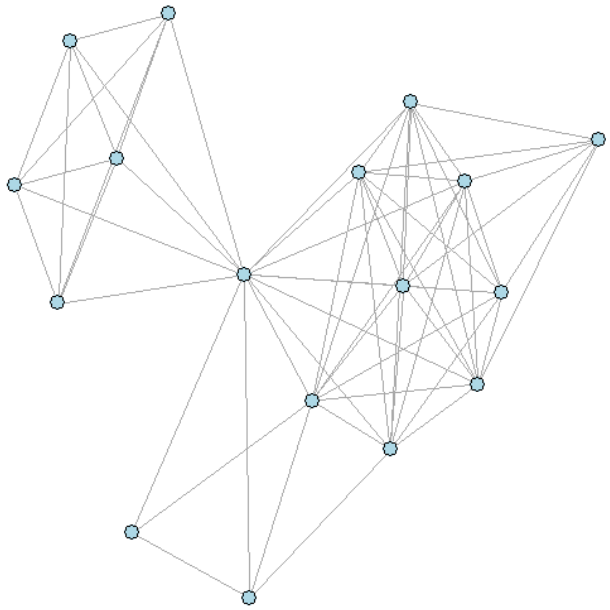


Figura 1

6.1.2 Publishing Web-Based Interactive Network Diagrams

Instead of building interactive network graphics within R itself, more people are beginning to look at ways to produce interactive graphics that are published on the Web, using frameworks like the D3 JavaScript library (http://http://d3js . org/) and Shiny (http: / /shiny .rstudio.com/).

None of these approaches yet have come close to matching what a fully-developed network graphics application such as Gephi can do. However, I anticipate that we will be seeing rapid development of more R-connected approaches to web-based network visualization in the next few years.

The networkD3 package is a small set of functions that can be used to build simple interactive network graphics that can be displayed in shiny-aware documents (i.e., RStudio) or in HTML web-pages. The following code shows how simple it is to produce an interactive graphic. The first set of lines will send a graphic to the Viewer window if you run the commands within RStudio. The simpleNetwork() function expects the network data in the form of an edgelist stored in a dataframe. (The output from the examples in this section is not shown here, because it requires RStudio or a web browser to view.)

```
library(networkD3)
```

```
src <- c("A","A","B","B","C","E")
target <- c("B","C","C","D","B","C")
net_edge <- data.frame(src, target)
simpleNetwork(net_edge)
```

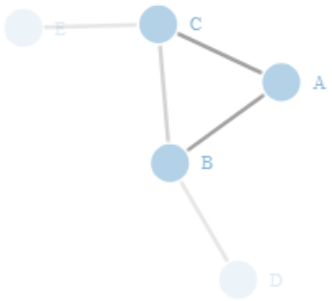


Figura 2

To save the interactive network to a freestanding HTML file, use the following code.

```
net_D3 <- simpleNetwork(net_edge)
saveNetwork(net_D3,file = 'Net_test1.html',
selfcontained=TRUE)
```

The output from simpleNetwork is so simple that it mainly is useful as a proof-of-concept or tech demo. Slightly more sophisticated network graphics can be produced using the forceNetwork ( ) function. For this example, we are using the Bali network again. The function expects data to be passed to it in two data frames. The 'links' dataframe will have the network data in edgelist format. The 'nodes' dataframe will have the node id and properties of the nodes. Currently only a categorical grouping variable is allowed. If the nodes have numeric ids, they must start at 0 . So, the main work to use the function is putting the data into the correct format.

```
iBali_edge <- get.edgelist(iBali)
iBali_edge <- iBali_edge - 1
iBali_edge <- data.frame(iBali_edge)
iBali_nodes <- data.frame(NodeID=as.numeric(V(iBali)-1),
Group=V(iBali)$role,
Nodesize=(degree(iBali)))
forceNetwork(Links = iBali_edge, Nodes = iBali_nodes,
Source = "X1", Target = "X2",
NodeID = "NodeID",Nodesize = "Nodesize",
```

```
radiusCalculation="Math.sqrt(d.nodesize)*3",
Group = "Group", opacity = 0.8,
legend=TRUE)
```

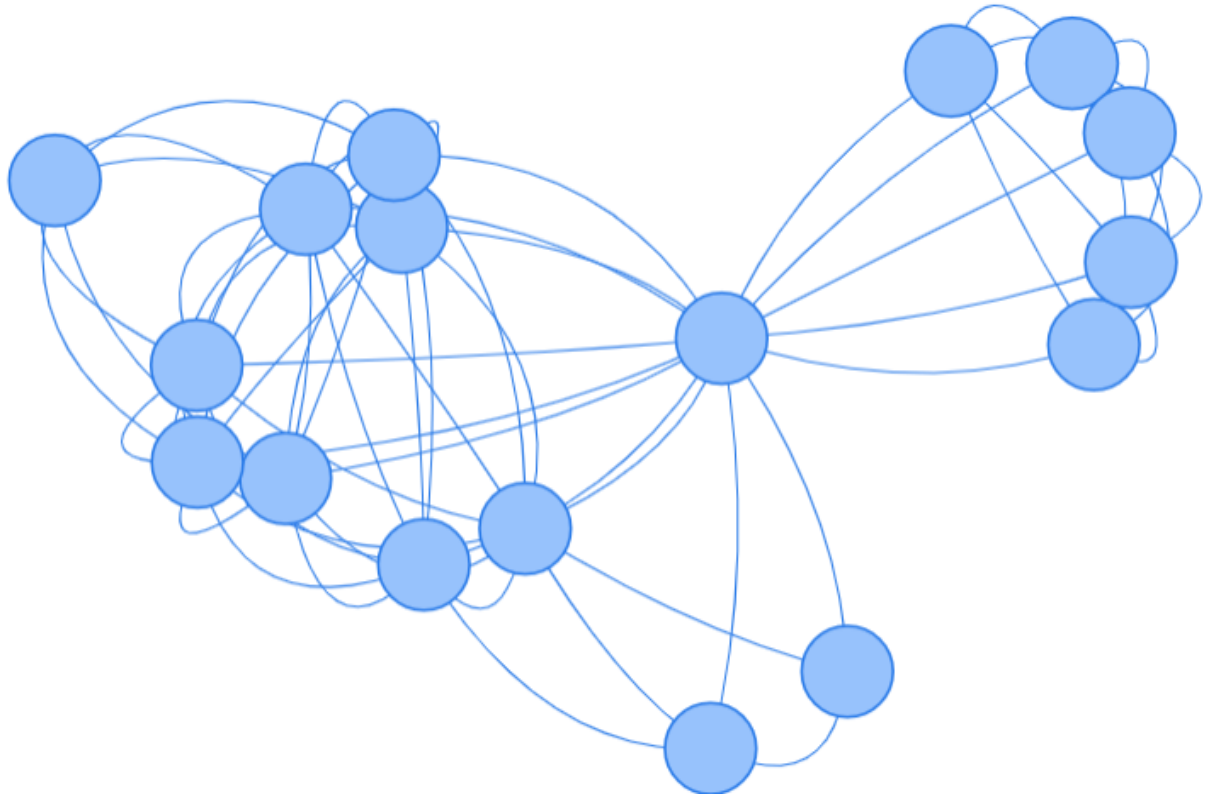


Figura 3

Once again, this can be saved to an external file. Be careful, you will get an error if you try to overwrite an existing file, even if it is not open in your browser.

```
net_D3 <- forceNetwork(Links = iBali_edge,
Nodes = iBali_nodes,
Source = "X1", Target = "X2",
NodeID = "NodeID",Nodesize = "Nodesize",
radiusCalculation="Math.sqrt(d.nodesize)*3",
Group = "Group", opacity = 0.8,
legend=TRUE)
saveNetwork(net_D3,file = 'Net_test2.html',
    selfcontained=TRUE)
```

The visNetwork package is a similar set of tools that uses the vis.js javascript library (http://visjs.org/) to produce web-based interactive network graphics.

This package also requires network data to be provided in a nodes data frame and an edges data frame. The nodes data frame should include an id column, and the edges data frame should have from and columns. Using the Bali network, the following code sets up the data and produces a minimal example of an interactive network graphic. Like in the previous example, this code produces an interactive network in the Viewer window of RStudio.

```
library(visNetwork)
iBali_edge <- get.edgelist(iBali)
```

```
iBali_edge <- data.frame(from = iBali_edge[,1],
to = iBali_edge[,2])
iBali_nodes <- data.frame(id = as.numeric(V(iBali)))
visNetwork(iBali_nodes, iBali_edge, width = "100%")
```

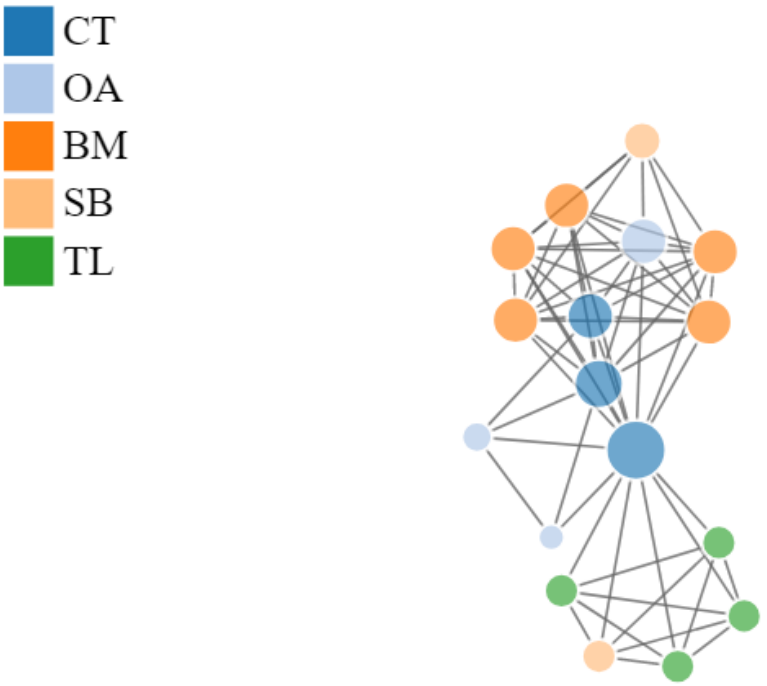


Figura 4

The visNetwork package has a large number of options that can be used to control the appearance of the network diagram, as well as for controlling how the plot can be embedded in Shiny web applications. See the package help file for more information, as well as a more in-depth demonstration of its capabilities available at http://dataknowledge.github. io/visNetwork/. The next code shows off some of these options.

```
iBali_nodes$group <- V(iBali)$role
iBali_nodes$value <- degree(iBali)
net <- visNetwork(iBali_nodes, iBali_edge,
width = "100%",legend=TRUE)
visOptions(net,highlightNearest = TRUE)
```

First, some of the display options are controlled by saving node or edge information into the nodes or edges data frames. Here, the group variable stores the 'role' attribute, and the value variable is used to store the node sizes (in this case, the degree). The visNetwork ( ) and visoptions ( ) functions are used to display the network, add a legend based on the grouping variable, set default colors for each group, and then allow for the user to highlight individual nodes and their immediate neighbors when clicking on a node in the diagram.

As before, these interactive plots will appear in a plot window if you are using RStudio. Once the plot has been designed, it can be exported to a freestanding webpage or embedded in other web platforms (e.g., with Shiny). This last example shows how to save the plot in a separate web file, using the saveWidget () function from the htmlwidgets package, which is installed when you install the

visNetwork package. This example adds a set of navigation buttons to the final network plot that allows moving the network and zooming in or out.

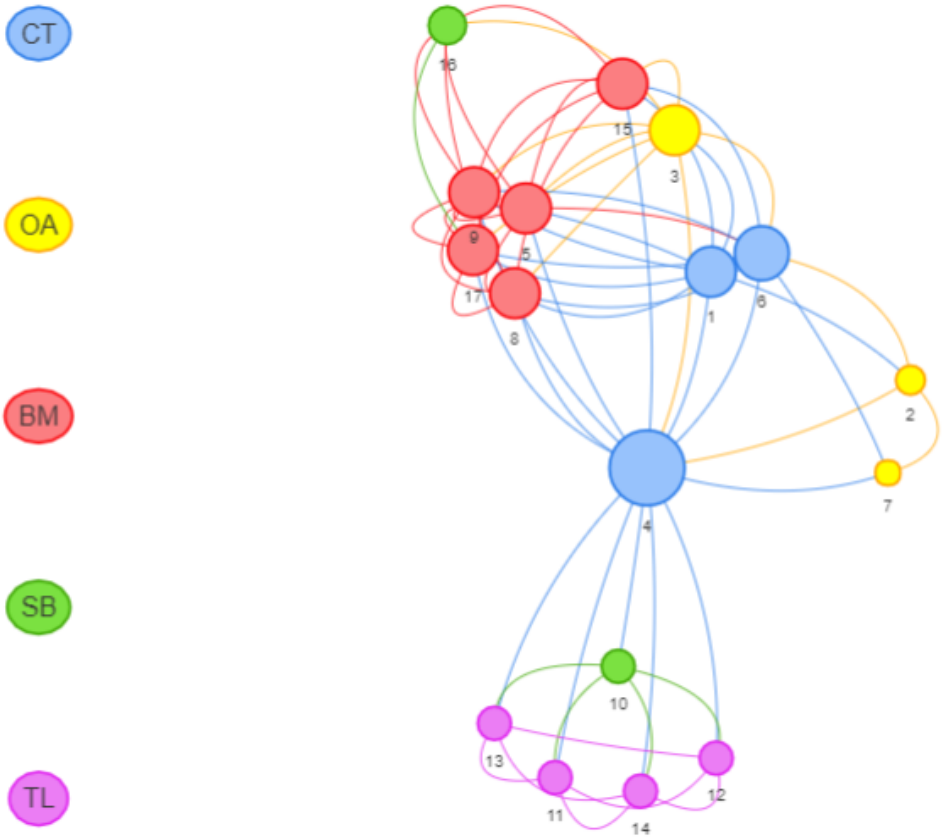


Figura 5

```
net <- visNetwork(iBali_nodes, iBali_edge,
width = "100%",legend=TRUE)
net <- visOptions(net,highlightNearest = TRUE)
net <- visInteraction(net,navigationButtons = TRUE)
library(htmlwidgets)
saveWidget(net, "Net_test3.html")
```

6.1.3 Statnet Web: Interactive statnet with shiny

As evidence of the rapid development of interactive network tools, the Statnet development team has recently published a web-based version of their R network analytic tools using the shiny web application framework.

Statnet Web can be used by connecting directly to the shinyapps.io server at https://statnet. shinyapps.io/ statnetWeb. Or, the tools can be run locally by installing the statnetWeb package. In addition to producing basic network plots by selecting parameters and options from drop-down boxes, statnetWeb can produce a variety of network statistics as well as fit and test ERGMs (see Chap. 11). Although web-based statnet does not give as much control over or reproducibility of network analytic results as a programming approach does, it is an impressive platform for quickly exploring network characteristics and will be useful for teaching as well as disseminating network analytic results.

```
library (statnetWeb)
run-sw ( )
```

6.2 Specialized Network Diagrams

Traditionally, network diagrams are plotted to illustrate fundamental network and node properties such as prominence (see Chap.4). However, there are a number of more specialized plotting techniques

that can be used that are appropriate for highlighting other important or interesting aspects of the networks. Three of these approaches are demonstrated in this section: arc diagrams, chord diagrams, and heatmaps.

6.2.1 Arc Diagrams

Arc diagrams can be used when the positioning of nodes in a network is of less interest than the pattern of ties. Here is a simple example of an arc diagram, using the arcdiagram package. Note that this has to be installed using GitHub.

```
library(devtools)
install_github("gastonstat/arcdiagram")
```

The set-up for this example includes loading all the required libraries, then creating an edgelist object for the arcdiagram ( ) function. For this example, we are using the Simpsons dataset, which contains a set of (fictitious) network data that shows the primary interaction ties between 15 of the characters on the Simpsons television show.

```
library(arcdiagram)
library(igraph)
library(intergraph)
data(Simpsons)
iSimp <- asIgraph(Simpsons)
simp_edge <- get.edgelist(iSimp)
```

A basic arc diagram can be produced with one function call (Fig. 6.1).

```
arcplot (simp_edge)
```

The arc diagram can be enhanced in a number of ways to highlight node and other network characteristics. Here we define some subgroups in the network (1 = family, 2 = work, 3 = school, 4 = neighborhood) and use colors to distinguish the groups (colors taken from a palette at colorbrewer 2 .org). Also, the degree of each node is used to adjust its size (Fig. 6.2).

```
s_grp <- V(iSimp)$group
s_col = c("#a6611a", "#dfc27d","#80cdc1","#018571")
cols = s_col[s_grp]
node_deg <- igraph::degree(iSimp)


arcplot(simp_edge, lwd.arcs=2, cex.nodes=node_deg/2,
        labels=V(iSimp)$vertex.names,
        col.labels="darkgreen",font=1,
        pch.nodes=21,line=1,col.nodes = cols,
        bg.nodes = cols, show.nodes = TRUE)
```

Fig. 6.1 Simpsons contact network

6.2.2 Chord Diagrams

Chord diagrams are a specialized type of information graphic that uses a circular layout to display the interrelationships between data in a matrix. They have become particularly popular in genetics research. Because network information can be organized in matrices, chord diagrams are an interesting

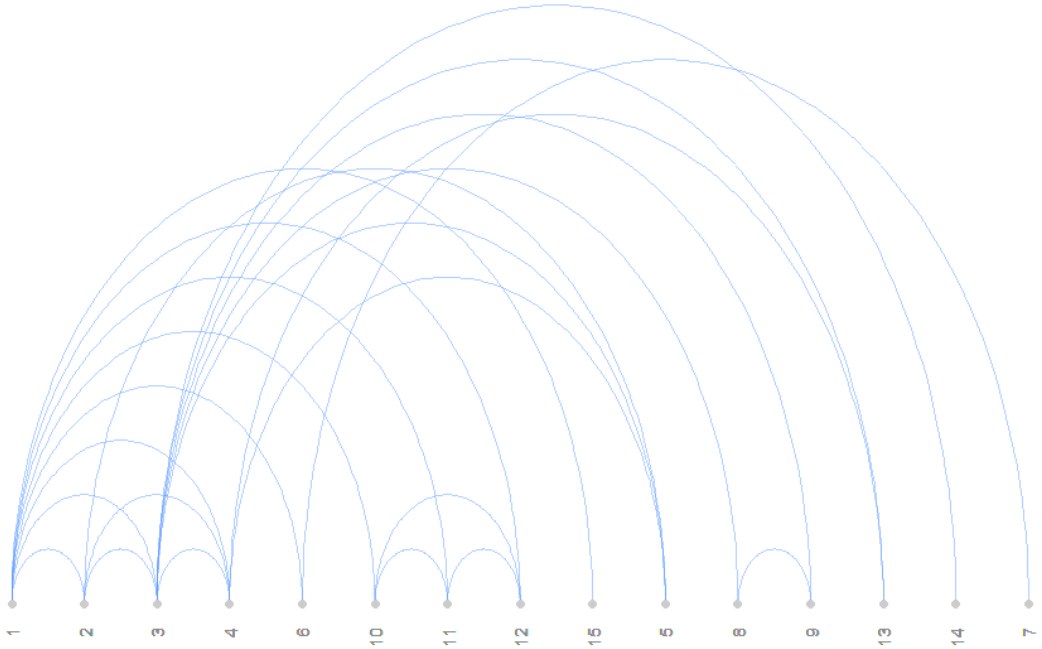


Figura 6

graphic option for network plots. This is especially true for valued (weighted) and directed networks, where the amount and direction of the 'flows' are of interest.

The circlize package, by Zuguang Gu, implements a variety of circular graphics, including chord diagrams. The package has a lot of features, giving the user great control over the graphical appearance. The included vignette, `circular_visualization_of_matrix` is suggested reading.

In this example, we return to the network of the 2010 Netherlands World Cup soccer team. Although Fig. 1.2 shows the basic pattern of passing flows between the eleven members of the team, it ignores the number of passes (stored in the vertex attribute passes). Here we will create a chord diagram to further examine these patterns.

The first steps are to load the required packages and prepare the data. The main requirement is to have the network data in the form of a sociomatrix, with the entries corresponding to the strength or size of the tie if it is a valued network. The matrix will also have to have names assigned for the rows and columns. (In this example

Fig. 6.2 Simpsons contact network - Version 2

we have an N × N matrix, so the names will be the same for rows and columns. The circlize package can also be used for N × k matrices, so chord diagrams will also be useful for 2-mode affiliation networks, such as those discussed in Chap. 9.)

```
library(statnet)
library(circlize)
load("C:/Users/HP/Downloads/Taller3a/FIFA_Nether.rda")
FIFAm <- as.sociomatrix(FIFA_Nether,attrname='passes')
names <- c("GK1","DF3","DF4","DF5","MF6",
           "FW7","FW9","MF10","FW11","DF2","MF8")
rownames(FIFAm) = names
colnames(FIFAm) = names
FIFAm
```

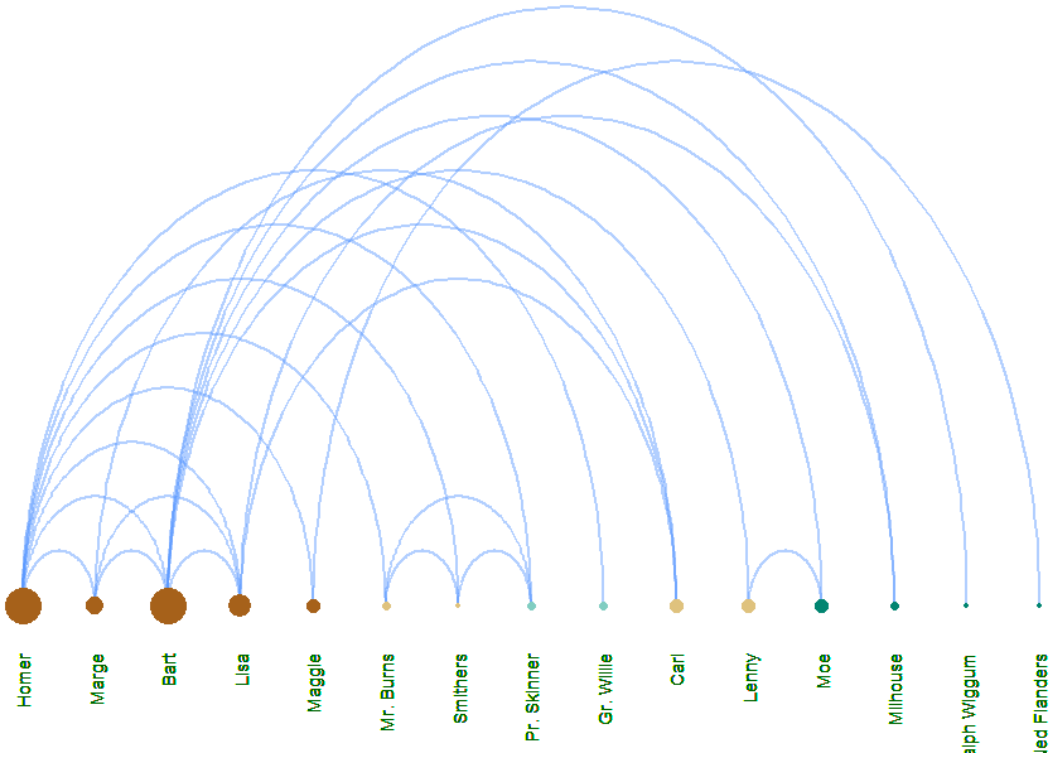


Figura 7

The sociomatrix reveals a number of ties that have very low numbers of passes. To make the subsequent graphics a little easier to interpret we drop all ties with less than ten passes.

```
FIFAm[FIFAm < 10] <- 0
FIFAm
```

With a sociomatrix that has names assigned, a basic chord diagram can be produced by a simple call to the chordDiagram () function (Fig. 6.3).

```
chordDiagram(FIFAm)
```

Chord diagrams can contain a lot of information, especially for larger networks, so it is usually important to fine tune the plot to highlight the most important information. In this next plot, a number of options are used to make the graphic a little easier to interpret. First, colors are set so that players in the same position (Forward, Midfielder, etc.) have the same color. Then, because this is a directed network, flows (passes, in this case) go in both directions. The directional option is used so that the departing passes start further away from outer circle, making it easier to see the difference between passes sent and passes received. Finally, the order option is used to sort the players by their position.

```
grid.col <- c("#AA3939",rep("#AA6C39",4),
              rep("#2D882D",3),rep("#226666",3))
chordDiagram(FIFAm,directional = TRUE,
             grid.col = grid.col,
             order=c("GK1","DF2","DF3","DF4","DF5",
                     "MF6","MF8","MF10","FW7",
                     "FW9","FW11"))
```

	GK1	DF3	DF4	DF5	MF6	FW7	FW9	MF10	FW11	DF2	MF8
GK1	0.00	42.00	67.00	21.00	2.00	27.00	7.00	5.00	2.00	17.00	3.00
DF3	30.00	0.00	44.00	14.00	42.00	15.00	8.00	7.00	10.00	36.00	29.00
DF4	38.00	43.00	0.00	57.00	18.00	11.00	7.00	21.00	1.00	7.00	28.00
DF5	6.00	14.00	47.00	0.00	11.00	50.00	20.00	40.00	1.00	4.00	42.00
MF6	9.00	28.00	25.00	10.00	0.00	41.00	28.00	37.00	14.00	34.00	21.00
FW7	4.00	12.00	1.00	21.00	21.00	0.00	15.00	33.00	9.00	25.00	18.00
FW9	0.00	0.00	1.00	8.00	7.00	12.00	0.00	31.00	16.00	7.00	2.00
MF10	1.00	11.00	11.00	22.00	43.00	29.00	20.00	0.00	28.00	13.00	21.00
FW11	3.00	2.00	2.00	3.00	7.00	6.00	11.00	15.00	0.00	21.00	12.00
DF2	29.00	38.00	8.00	3.00	45.00	38.00	10.00	18.00	26.00	0.00	15.00
MF8	12.00	25.00	26.00	38.00	23.00	13.00	12.00	32.00	11.00	24.00	0.00

	GK1	DF3	DF4	DF5	MF6	FW7	FW9	MF10	FW11	DF2	MF8
GK1	0.00	42.00	67.00	21.00	0.00	27.00	0.00	0.00	0.00	17.00	0.00
DF3	30.00	0.00	44.00	14.00	42.00	15.00	0.00	0.00	10.00	36.00	29.00
DF4	38.00	43.00	0.00	57.00	18.00	11.00	0.00	21.00	0.00	0.00	28.00
DF5	0.00	14.00	47.00	0.00	11.00	50.00	20.00	40.00	0.00	0.00	42.00
MF6	0.00	28.00	25.00	10.00	0.00	41.00	28.00	37.00	14.00	34.00	21.00
FW7	0.00	12.00	0.00	21.00	21.00	0.00	15.00	33.00	0.00	25.00	18.00
FW9	0.00	0.00	0.00	0.00	0.00	12.00	0.00	31.00	16.00	0.00	0.00
MF10	0.00	11.00	11.00	22.00	43.00	29.00	20.00	0.00	28.00	13.00	21.00
FW11	0.00	0.00	0.00	0.00	0.00	0.00	11.00	15.00	0.00	21.00	12.00
DF2	29.00	38.00	0.00	0.00	45.00	38.00	10.00	18.00	26.00	0.00	15.00
MF8	12.00	25.00	26.00	38.00	23.00	13.00	12.00	32.00	11.00	24.00	0.00

In the resulting chord diagram (Fig. 6.4), it is much easier to see the patterns of passes among the players. We can see that FW7 receives more than twice the number of passes than the other two forwards. Similarly, we can see that the goalkeeper's favorite target is DF4, and that DF4 likes to pass frequently to DF5.

6.2.3 Heatmaps for Network Data

Heatmaps are another example of a specialized graphic that can be used for networks, especially valued or weighted networks. Here, a heatmap is produced to highlight the players who are passing or receiving the most among the Netherlands teammates.

First, a sociomatrix is created with the cells reflecting the tie weight, in this case 'passes.' Row and column names are defined for the margin labels.

```
load("C:/Users/HP/Downloads/Taller3a/FIFA_Nether.rda")
FIFAm <- as.sociomatrix(FIFA_Nether,attrname='passes')
colnames(FIFAm) <- c("GK1","DF3","DF4","DF5",
                     "MF6","FW7","FW9","MF10",
                     "FW11","DF2","MF8")
rownames(FIFAm) <- c("GK1","DF3","DF4","DF5",
                     "MF6","FW7","FW9","MF10",
                     "FW11","DF2","MF8")
```

Once the data are set up, the heatmap is relatively easy to produce (Fig. 6.5). The colorRampPalette () function is used to designate a color range that will be used for the low and high ends of the values in the sociomatrix. (The color ranges chosen here were taken from color chooser tools at paletton.com.) The network data are directed, so it is important to remember that here the rows are the 'passers' and the columns are the 'receivers.'

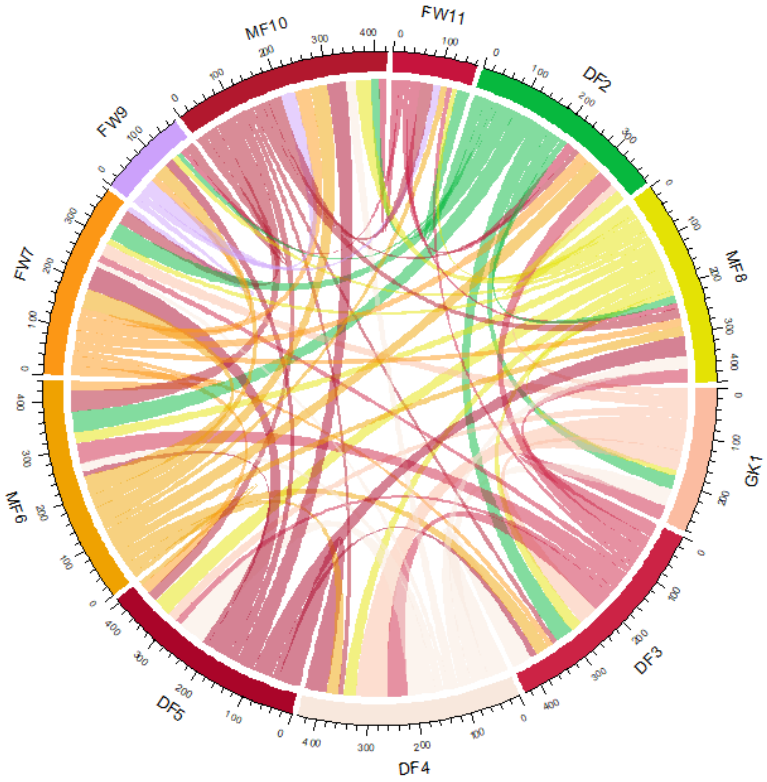


Figura 8

```
palf <- colorRampPalette(c("#669999", "#003333"))
heatmap(FIFAm[,11:1],Rowv = NA,Colv = NA,col = palf(60),
        scale="none", margins=c(11,11) )
```

6.3 Creating Network Diagrams with Other R Packages

6.3.1 Network Diagrams with ggplot2

Although ggplot2 is not designed to handle all of the requirements of a fullfledged network visualization package, some of its advanced graphics capabilities can be used to create specialized network plotting routines. The following example

is based on code developed by David Sparks, and posted on the blog he runs with his colleague Christopher DeSante, is.R() (http://is-r.tumblr.com).

The edgeMaker ( ) function and supporting code can be used to create attractive and functional plots of directed networks using 'tapered-intensity-curved' edges. The bulk of the work is done by the edgeMaker ( ) function which creates the curved ties between each connected dyad.

```
  edgeMaker <- function(whichRow, layoutCoordinates, adjacencyList, len = 100, curved = TR
fromC <- layoutCoordinates[adjacencyList[whichRow, 1], ]
toC <- layoutCoordinates[adjacencyList[whichRow, 2], ]
graphCenter <- colMeans(layoutCoordinates)
bezierMid <- c(fromC[1], toC[2])
distance1 <- sum((graphCenter - bezierMid)^2)

if (distance1 < sum((graphCenter - c(toC[1], fromC[2]))^2)) {
  bezierMid <- c(toC[1], fromC[2])
}
```

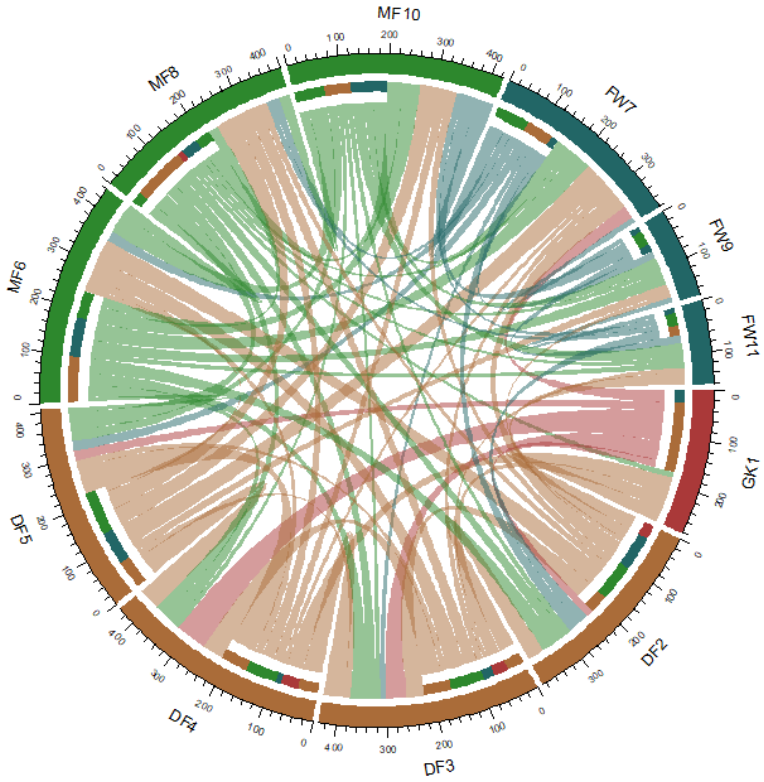


Figura 9

```
  bezierMid <- (fromC + toC + bezierMid) / 3

  if (!curved) {
    bezierMid <- (fromC + toC) / 2
  }

  edge <- data.frame(bezier(c(fromC[1], bezierMid[1], toC[1]),
                            c(fromC[2], bezierMid[2], toC[2]),
                            evaluation = len))
  edge$Sequence <- 1:len
  edge$Group <- paste(adjacencyList[whichRow, 1:2], collapse = ">")

  return(edge)
}
```

In addition to the core sna and ggplot 2 packages, the Hmisc package is used which provides the bezier () function used by edgeMaker.

```
library(sna)
library(ggplot2)
library(Hmisc)
```

As has been typical with the examples in this chapter, the network data has to be transformed to an edgelist format prior to using the plotting functions. For this example, we also drop the ties that have the associated weight 'passes' less than 10. Finally, the edgeMaker function expects the edgelist object to be named 'adjacencyList.'

```
load("C:/Users/HP/Downloads/Taller3a/FIFA_Nether.rda")
```

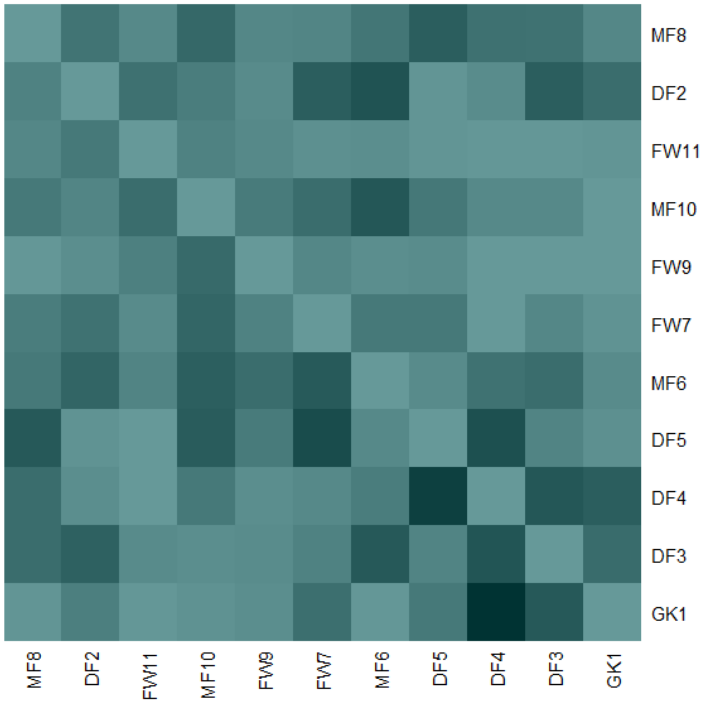


Figura 10

```
fifa <- FIFA_Nether
fifa.edge <- as.edgelist.sna(fifa,attrname='passes')
fifa.edge <- data.frame(fifa.edge)
names(fifa.edge)[3] <- "value"
fifa.edge <- fifa.edge[fifa.edge$value > 9,]
adjacencyList <- fifa.edge
```

Now, we use edgeMaker to create the curved edges. Also, gplot (from sna) is called once to store the layout coordinates for the ggplot 2 function. (This means that any set of coordinates can be fed to ggplot2.)

```
layoutCoordinates <- gplot(network(fifa.edge))
allEdges <- lapply(1:nrow(fifa.edge), function(i) {
  edgeMaker(i, layoutCoordinates, fifa.edge, len = 500, curved = TRUE)
})
allEdges <- do.call(rbind, allEdges)
```

Before producing the plot, we create an empty ggplot 2 theme. This is used to clean up after producing the plot.

```
new_theme_empty <- theme_bw()
new_theme_empty$line <- element_blank()
new_theme_empty$rect <- element_blank()
new_theme_empty$strip.text <- element_blank()
new_theme_empty$axis.text <- element_blank()
new_theme_empty$plot.title <- element_blank()
new_theme_empty$axis.title <- element_blank()
new_theme_empty$plot.margin <- structure(c(0,0,-1,-1),
```

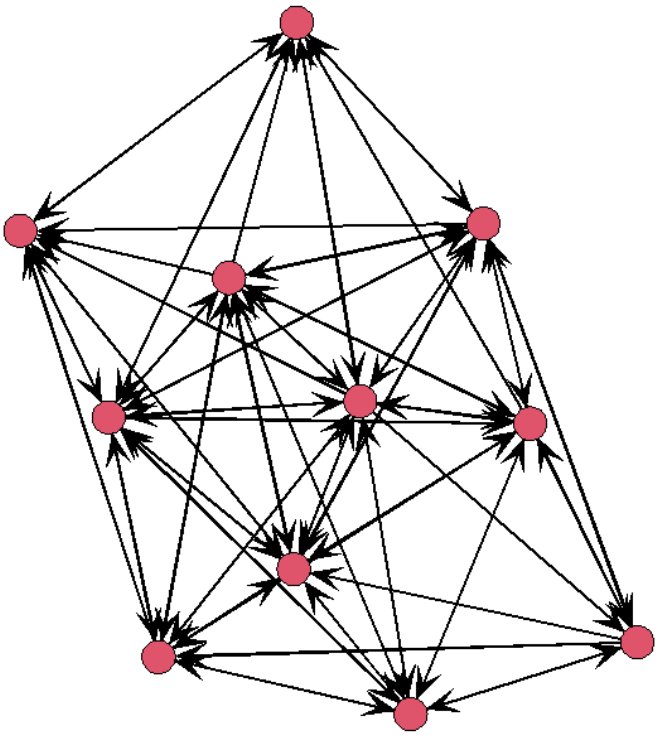


Figura 11

```
                                        unit = "lines",
                                        valid.unit = 3L,
                                        class = "unit")

zp1 <- ggplot(allEdges)
zp1 <- zp1 + geom_path(aes(x = x, y = y, group = Group,
                           colour=Sequence, size=-Sequence))
zp1 <- zp1 + geom_point(data =
                          data.frame(layoutCoordinates),
                        aes(x = x, y = y),
                        size = 4, pch = 21,
                        colour = "black", fill = "gray")
zp1 <- zp1 + scale_colour_gradient(low = gray(0),
                                   high = gray(9/10),
                                   guide = "none")
zp1 <- zp1 + scale_size(range = c(1/10, 1.5),
                        guide = "none")
zp1 <- zp1 + new_theme_empty
print(zp1)
```

And now the final step is to create the plot using ggplot (). Familiarity with ggplot 2 will help in understanding this code. The scale-colour-gradient option controls the intensity of the gradient, and the scale-size option controls the amount of the taper (Fig. 6.6).

2. **Sintetizar y replicar el Capítulo 9 de Luke (2015).**

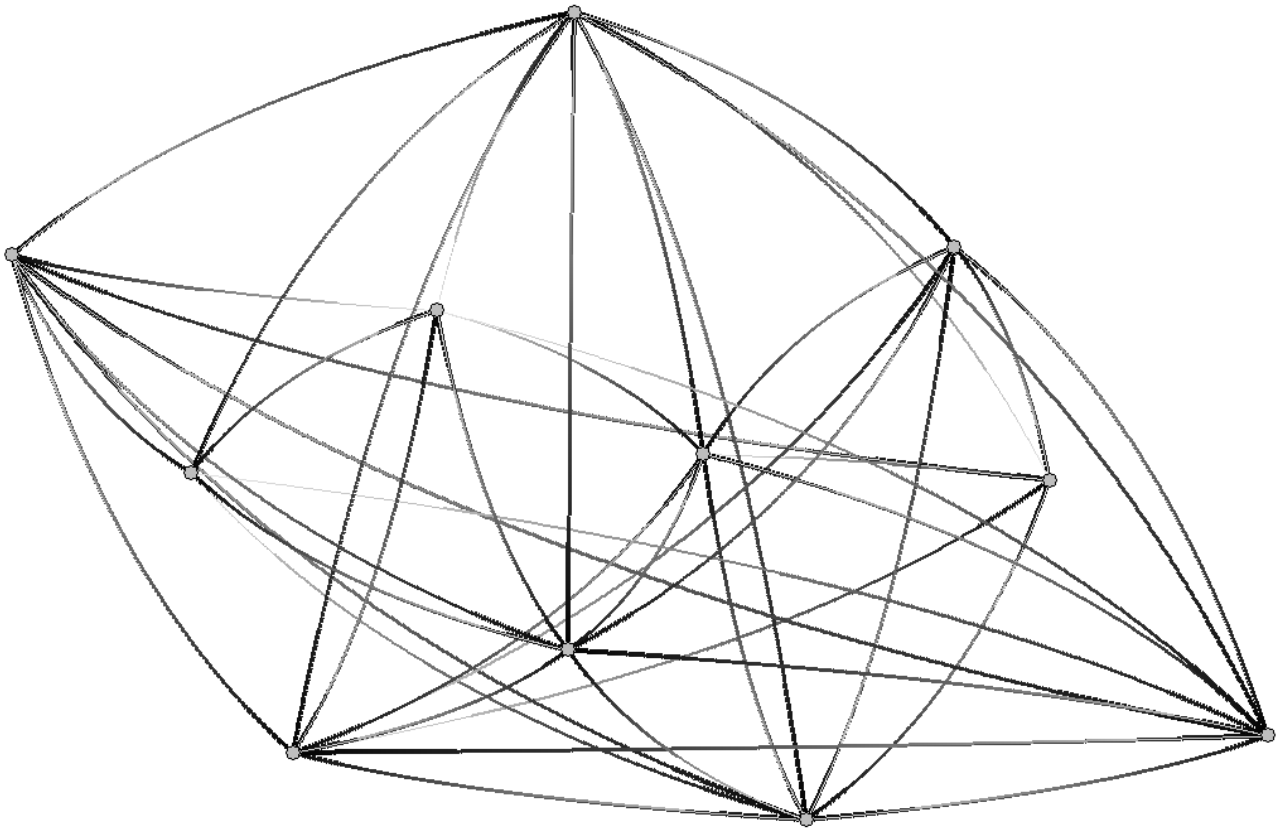


Figura 12

Chapter 9
Affiliation Networks

9.1 Defining Affiliation Networks

Until now, all the networks that we have examined are based on direct ties. That is, the social ties connecting the actors in the social network have been confirmed through self-report, direct observation, or some other type of data collection that tells us how actors are directly connected to one another.

However, social scientists are often interested in situations where there may be the opportunity for social relationships, but these relationships cannot be directly observed. However, by virtue of occupying the same social situation, we may infer that there is an opportunity or potential for social connections.

We call this new type of social network an affiliation network. An affiliation network is a network where the members are affiliated with one another based on co-membership in a group, or co-participation in some type of event. For example, students who all belong to the same class can be thought of as being connected to one another, although we may not know whether they actually have direct social ties.

The classic example from network science is the case of corporate interlocks. Company directors have the opportunity to interact with each other when they sit together on the same corporate board of directors. Moreover, the companies themselves can be seen to be connected through their shared director memberships. That is, when the same director sits on two different company boards, those companies are connected through that director. Sociologists and political scientists have used these types of affiliation networks to explain how companies tend to behave in similar ways to one another (Galaskiewicz 1985).

9.1.1 Affiliations as 2-Mode Networks

As a simple example of an affiliation network, consider the following data table of students grouped in classes (Table 9.1).

```
# Definir los vectores en Python
C1 = [1, 1, 1, 0, 0, 0]
C2 = [0, 1, 1, 1, 0, 0]
C3 = [0, 0, 1, 1, 1, 0]
C4 = [0, 0, 0, 0, 1, 1]

# Crear el dataframe
aff_df = pd.DataFrame({'C1': C1, 'C2': C2, 'C3': C3, 'C4': C4})

# Establecer los nombres de fila
aff_df.index = ['S1', 'S2', 'S3', 'S4', 'S5', 'S6']
```

	C1	C2	C3	C4
S1	1	0	0	0
S2	1	1	0	0
S3	1	1	1	0
S4	0	1	1	0
S5	0	0	1	1
S6	0	0	0	1

Table 9.1 Students grouped by classes

This type of data matrix is called an incidence matrix, and it depicts how n actors belong to g groups. In this case we have six students grouped into four classes. An incidence matrix is similar to an adjacency matrix, but an adjacency matrix is an nxn square matrix where each dimension refers to the actors in the network. An incidence matrix, on the other hand, is an nxg rectangular matrix with two different dimensions: actors and groups. For this reason, affiliation networks are also known as two-mode networks.

9.1.2 Bipartite Graphs

In affiliation networks, there are always two types of nodes: one type for the actors, and another type for the groups or events to which the actors belong. Ties then connect the actors to those groups. One consequence of this is that there are no direct ties among actors, and there are no direct ties between the groups. Figure 9.1, which shows the example student affiliation network, illustrates this defining characteristic of a bipartite graph.

Both `statnet` and `igraph` have functionality built in to recognize and operate on affiliation networks. The process with igraph is a little more straightforward, so it will be used in this chapter.

```
g = ig.Graph.Biadjacency(aff)
g.vs['name']=['S1', 'S2', 'S3', 'S4', 'S5', 'S6','C1', 'C2', 'C3', 'C4']
# al parecer se agregan primero las filas y posteriormente las columnas

plt_x = [2] * 6 + [4] * 4
plt_y = list(range(7, 1, -1)) + list(range(6, 2, -1))
layout = [(x, y) for x, y in zip(plt_x, plt_y)]
# layout adaptado con chat gpt

shapes = ['circle', 'square']
```

```
colors = ['blue', 'red']
# se crea una lista de formas

shape=[]
color=[]

# Assigning shapes and colors to vertices
for type in (g.vs['type']):
    shape.append(shapes[type]) # si es un elemento de las filas es 0 y por el contrario es 1
    color.append(colors[type]) # si es un elemento de las filas es 0 y por el contrario es 1

# se agrega el atrivuto a los vertices
g.vs['color']=color
g.vs['shape']= shape

ig.plot(g,vertex_label=g.vs['name'],layout=layout)
```

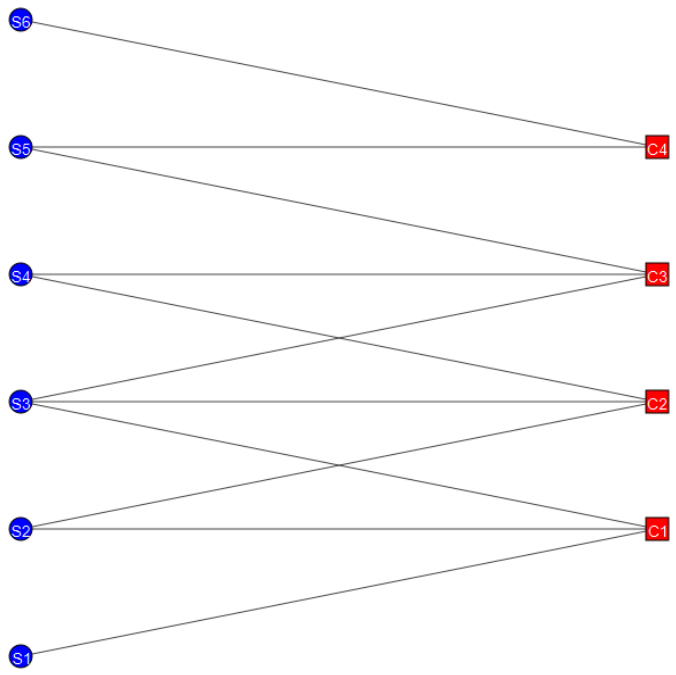


Figura 13: Fig. 9.1 Affiliation network as bipartite graph

9.2 Affiliation Network Basics

9.2.1 Creating Affiliation Networks from Incidence Matrices

An affiliation network can be stored as an igraph object in a few different ways. If the underlying data are available as an incidence matrix (e.g., Table 9.1), then this can be done in one line of code.

```
for es in g.es:
print(str(aff_df.index[es.tuple[0]]) + ' ~ ' str(aff_df.columns[es.tuple[1]-aff_df.shape[0]]
# como los vertices se guardan como una lista en la que primero van las
#   filas y luego las columnas para ver cuales son los vertices y vemos su
#   posicion en la df original

S1 ~ C1
S2 ~ C1
S2 ~ C2
S3 ~ C1
S3 ~ C2
S3 ~ C3
S4 ~ C2
S4 ~ C3
S5 ~ C3
S5 ~ C4
S6 ~ C4
```

The graph. incidence function takes a matrix or data.frame and transforms it into an affiliation network, reading the rows as actors and the columns as the groups or events. Note that both the row and column names should be defined so that they can be correctly assigned to the individual actors and groups.

By typing in the name of the igraph object, we get a cryptic two line summary of the network. The ' B ' in the 'UN-B' string tells us that this is a bipartite network. Furthermore, the second line shows that this network has two vertex attributes: name stores the name of the vertex, and type is a logical vector that igraph uses to distinguish between the two different types of nodes (students and classes, in this case).

More information about the affiliation network can be obtained using traditional igraph functions.

Here we can see the underlying incidence matrix, and the vertex names and types. The correspondence between the names and the logical type vectors is clear, where all the students have type = FALSE, and the class nodes have type = TRUE.

9.2.2 Creating Affiliation Networks from Edge Lists

For larger networks, it is more common to have the underlying data available as an edge list. Edge lists can also be translated into an affiliation network, as long nodes of one type (e.g., students) are only connected to nodes of the other type (e.g., classes). The following code constructs the same example affiliation network from edge list data.

```
##  [1] S1--C1 S2--C1 S2--C2 S3--C1 S3--C2 S3--C3
##  [7] S4--C2 S4--C3 S5--C3 S5--C4 S6--C4
```

The above creates an network object, but igraph does not know that it is a bipartite graph. (Note that the network description lacks the ' B ' that indicates a bipartite graph.) To fix this, we can simply set the type vertex attribute. Once this is done, we have an affiliation network object that is formed from a bipartite graph.

9.2.3 Plotting Affiliation Networks

As with any type of network, affiliation networks can be plotted for visual inspection. However, it is useful to designate different node shapes and colors to make the affiliation structure easier to interpret. Here we will set the students to be blue circles, and the classes to be red squares. The code shows how the type attribute can be used as an index into both a shapes and a colors vector to select the appropriate shape and color for each node. Note that 1 is added to type index because as a logical vector it starts at 0 , whereas we want to select either the first or second elements of the shapes/colors vectors.

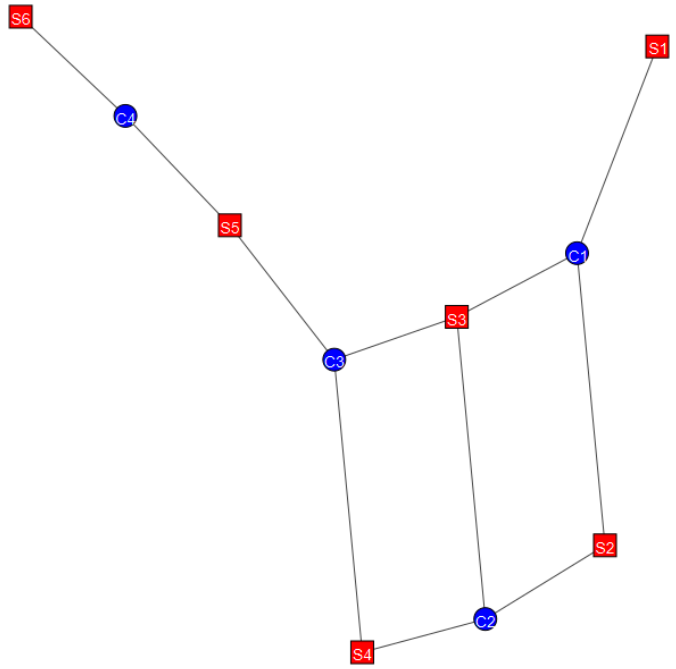


Figura 14: Fig. 9.2 Simple plot of affiliation network

9.2.4 Projections

By examining Fig. 9.2 we can see how classes are indirectly connected through their shared students. For example, classes 2 and 3 are indirectly connected through two shared students (S3 and S4). In comparison, classes 3 and 4 are also indirectly connected, but only via one shared student (S5). We can also focus the examination on the individual-level nodes. Here the figure reveals that students 2 and 4 are indirectly connected by their co-affiliation in class 2 .

Examining both types of nodes in a two-mode network graphic is often the first step in studying an affiliation network. However, it is also useful to examine the direct connections among the nodes of one type at a time (classes and students, in this case). This can be done by extracting and visualizing the one-mode projections of the two-mode affiliation network. Every actor by event affiliation network can produce two one-mode networks, one of actors and one of the events or affiliations.

In igraph the projections can be obtained again by just one line of code. The bipartite.projection function returns a list of two igraph network objects. The first network is made up of the direct ties among the first mode (in our case students), and the second network shows the ties among the second mode (classes).

The adjacency matrix of each one-mode projection can be obtained with the get.adjacency function. In the code below, notice how the edge attribute weight is specified. This produces a valued adjacency matrix, where the values indicate how many ties connect any of the nodes. So, for example, the Class adjacency matrix indicates that classes 2 and 3 have a weight of 2 . This reflects the observation we made earlier that classes 2 and 3 share two students (S3 and S4).

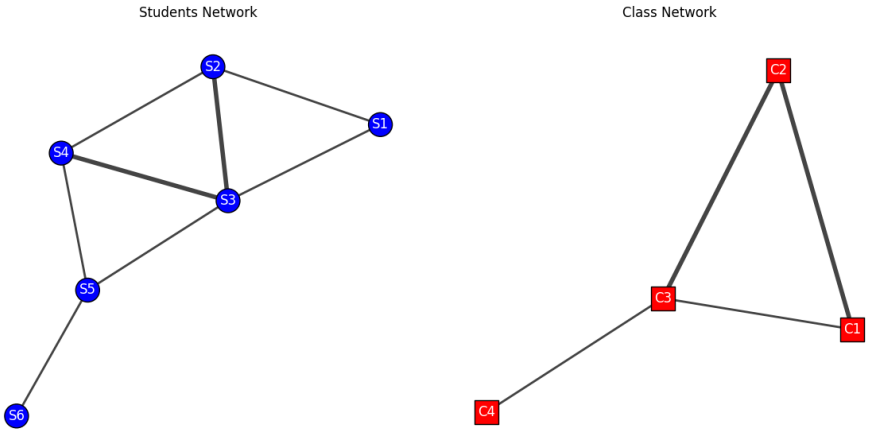


Figura 15: Fig. 9.3 Simple plot of affiliation network

9.3 Example: Hollywood Actors as an Affiliation Network

The data file hwd in the UseNetR package contains a larger and more interesting affiliation network that can be explored using these techniques. Hollywood actors are a good example of an affiliation network, actors are connected to one another through the movies in which they appear together. The hwd dataset is an igraph bipartite graph object. The data are originally from IMDB (www.imdb.com). The dataset contains the ten most popular movies (as judged by IMBD users) for

each year from 1999 to 2014, and the first ten actors listed on each movie's IMDB page. In addition to the movie and actor names, each movie has the year of its release, its IMDB user rating, and the MPAA movie rating (i.e., G, PG, PG-13, and R) stored as a node characteristic.

9.3.1 Analysis of Entire Hollywood Affiliation Network

Para continuar con el capítulo 9 del libro de Luke, en lugar de hacerlo con la red bipartita Hollywood Affiliation Network, lo haremos con la del proyecto que se desarrolla en clase. En este caso, se trabajará con una red bipartita inducida por usuarios y series que estos observan, y en este caso, son animes.Los usuarios están conectados con las series que ven. Los datos provienen originalmente de este enlace y se trata de un conjunto de datos que contiene información sobre 17,562 animes y las preferencias de 325,772 usuarios diferentes de la página MyAnimeList. Dado que el conjunto de datos es tan extenso, se trabajará con los animes emitidos desde el 2020 hasta el primer semestre del 2021. Además de contener los nombres de los animes, incluye el rating de usuario que se utilizará como una variable nodal.

```
print(gm.es.graph)
IGRAPH UN-T 635 4568 --
+ attr: Bool type (v), name (v), type (v)
+ edges (vertex names):
               100279 -- Kami no Tou, Darwin's Game
                10058 -- Haikyuu!!: To the Top, Yahari Ore no Seishun Love
Comedy wa Machigatteiru. Kan, Re:Zero kara Hajimeru Isekai Seikatsu 2nd
Season, Kami no Tou, Haikyuu!!: To the Top 2nd Season, Sword Art Online:
Alicization - War of Underworld 2nd Season, Runway de Waratte
               102533 -- Darwin's Game, Kaguya-sama wa Kokurasetai?:
Tensai-tachi no Renai Zunousen, Otome Game no Hametsu Flag shika Nai Akuyaku
Reijou ni Tensei shiteshimatta..., Honzuki no Gekokujou: Shisho ni Naru Tame
ni wa Shudan wo Erandeiraremasen 2nd Season, Kanojo, Okarishimasu, Isekai
Quartet 2

for index in range(-10, 0):
    print(gm.vs['name'][index])

Eternity: Shinya no Nurekoi Channel
Asatir: Mirai no Mukashi Banashi
Chou Futsuu Toshi Kashiwa Densetsu R
Komatta Jiisan
Get Up! Get Live! #Geragera
Himitsukessha Taka no Tsume: Golden Spell
Jujutsu Kaisen (TV)
5-toubun no Hanayome
SK
Jaku-Chara Tomozaki-kun

for index in range(0,10):
    print(g.vs['type'][index])
```

```
usuario
usuario
usuario
usuario
usuario
usuario
usuario
usuario
usuario
usuario
```

La descripción resumida indica que la Red Usuario - Anime: MyAnimeList es de hecho una red bipartita. Podemos deducir que los vínculos conectan a cada usuario con las series que este ha visto. También revela que la red tiene 635 nodos y 4568 vínculos. Esto es un poco más difícil de descifrar para una red de afiliación, pero dado lo que ya sabemos, podemos deducir que hay 135 nodos de series, 500 nodos de actores y los 4568 vínculos surgen de cada series que he vista por un usuario.

La red completa es demasiado grande para mostrarla aquí, pero podemos examinar un pequeño subconjunto antes de realizar análisis más enfocados. Como primer paso, podemos aprovechar la capacidad de 'igraph' para almacenar informacimo. En este caso, el color y la forma del nodo pueden ser designados definiéndolos como atributos de vérteicdos.")

```
colors=[adjustcolor('red',0.3),adjustcolor('royalblue',0.2) ]
shapes=['square','circle']
shape = []
color = []

# Asignar formas y colores a los vértices
for type in gm.vs['type']:
    if type == 'anime':
        idx = 0
    else:
        idx = 1
    shape.append(shapes[idx])  # Acceder a la forma correspondiente
    color.append(colors[idx])  # Acceder al color correspondiente

# Asignar las formas y colores a los vértices
gm.vs['shape'] = shape
gm.vs['color'] = color
gm.vs['vertex_frame_color'] = color  # Asignar el color del marco de los vértices
```

Para el primer gráfico, vamos a analizar un subconjunto de los animes más importantes de las temporadas de invierno, primavera, verano y otoño de MyAnimeList del año 2020. Este ejemplo también ilustra cómo crear un subgrafo extrayendo solo las aristas que son incidentes a vértices de nuestro interés (en este caso, queremos tener los vértices que están únicamente relacionados con los animes más famosos del año 2020). La clave aquí es el método `select` de la subclase `Graph.vs` y los métodos `subgraph`, `neighborhood` de la clase `Graph`. El método `select` busca vértices con algunas características, en este caso, el nombre. Posteriormente, con el método `neighborhood`, se extraen los vértices adyacentes a algún vértice en particular, se crea una lista con esto y finalmente se crea un subgrafo con el método `subgraph`.

```
Names=(Identificados
    .query("Name.str.contains('Yahari Ore no Seishun Love Comedy wa Machigatteiru.
             |Kaguya-sama|Haikyuu!!: To the Top|Jujutsu Kaisen')")
    .query('~Name.str.contains("Haikyuu!!: To the Top 2nd Season")')
)['Name'].unique()
```

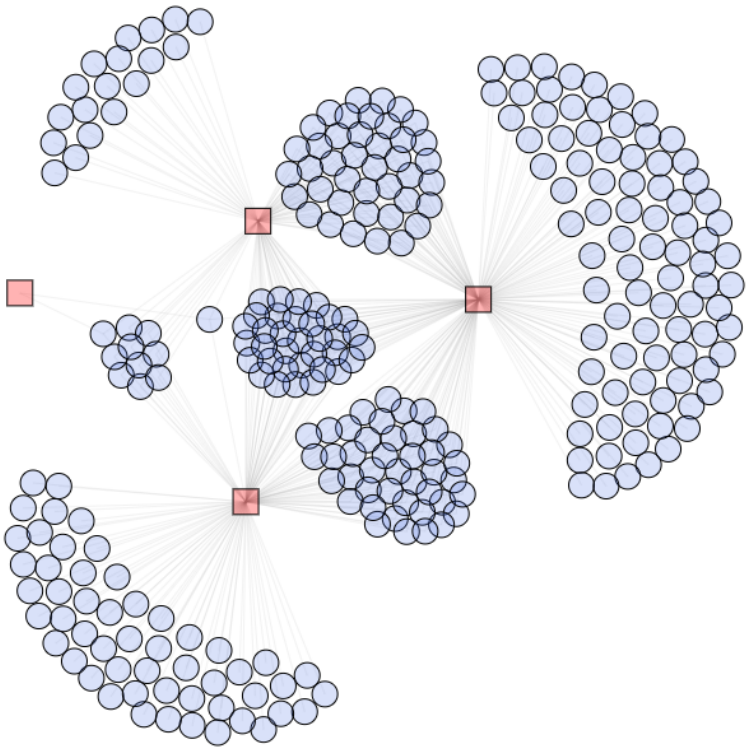


Figura 16: Grafo Animes Proyección

```
new_ID=gm.vs.select(name_in=Names)

opulares = []

# Iterar sobre los vértices y obtener sus vecindarios
for i in new_ID.indices:
    vecindario = gm.neighborhood(i)
    populares.extend(vecindario) # para ir añadiendo a la lista

random.seed(4212)
ig.plot(gm.subgraph(vertices=populares),
        background=None,edge_color =adjustcolor('gray',0.1))
```

Se puede aprender mucho de toda la red de animes. La mayoría de las estadísticas descriptivas de redes se pueden aplicar a las redes de afiliación, pero a menudo necesitan ser ajustadas ya sea en cómo están construidas o en cómo se interpretan. Por ejemplo, la densidad general de la red de afiliación se puede calcular fácilmente, pero no es muy significativa dada la forma en que se recopilaron los datos de la red (cada usuario, por definición, está conectado a un anime) y que no puede haber vínculos entre nodos de series o entre nodos de animes.

```
gm.density()
0.02269
```

En cambio, el grado de los nodos puede ser más informativo, al menos para los usuarios. El método `degree` y el método `select` permiten especificar qué vértices incluir, y se utiliza para seleccionar solo los actores (para los cuales la característica del nodo type es usuario).

```
np.mean(gm.degree(vertices=gm.vs.select(type_in='usuario')))

9.136
```

```
pd.DataFrame({'grado': gm.degree(vertices=gm.vs.select(type_in='usuario'))})[
'grado'].value_counts()
```

```
grado
1      100
2       52
4       37
3       35
6       30
7       27
5       23
11      22
```

Esto nos enseña que la mayoría de los usuarios han visto más de 10 animes que han salido durante el período de tiempo que se estudió. Si calculamos un promedio del grado, podemos observar que cada usuario ha visto 9.1 animes de la temporada. Esta informacion nos puede ayudar a identificar los usuarios mas activos.

Si quisiéramos evaluar la calificación promedio de los animes vistos por los usuarios, podríamos hacerlo accediendo a las características de cada anime que el usuario ha visto. Esto es un poco más complicado que el ejemplo anterior, pero se puede hacer utilizando las habilidades de igraph para identificar los vecinos adyacentes para cualquier nodo en el grafo.

El siguiente código recorre los nodos de actor en la red y suma el rating o calificación de todos los vecinos de cada nodo, calculando su respectiva media.).

```
# Inicializa una lista vacía para almacenar las medias de los usuarios
medias_usuarios = []

# Itera a través de los vértices
for i in gm.vs.select(type_in='usuario' , _degree_gt=9):
    n = gm.neighborhood(i)
    # Extrae nombres de los vertices adyacentes a ese usuario
    nombres = Identificados.query('Identificador_x in @n')['Name']
    # Calcula la media de puntuación para los animes que ve el usuario
    media_puntuacion = anime.query('Name in @nombres')['Score'].mean()
    # Añade el usuario y su media de puntuación a la lista
    medias_usuarios.append({'Usuario': i['name'], 'Media': media_puntuacion, 'Grado' :  gm.d

# Crea un DataFrame a partir de la lista de diccionarios
df_medias_usuarios = pd.DataFrame(medias_usuarios)
```

Una vez que tenemos esto, podemos examinar una vez más a los usuarios, lo cual se basa tanto en el número de animes vistos como en la media de calificación de estos. Finalmente, las características de la red siempre se pueden examinar utilizando enfoques gráficos y estadísticos más tradicionales. Por ejemplo, podemos ver si los usuarios que ven más animes también ven animes mejor o peor calificados, en promedio. Así, se puede examinar un simple gráfico de dispersión y regresión para ver la relación entre el número de animes y las calificaciones promedio de estos animes. Los resultados sugieren que no hay una relación clara entre qué tan activo es el usuario y la calificación de los animes que ve. El gráfico de dispersión también sugiere que a medida que se ven más animes, tienden a verse animes con una calificación menor.

```
import statsmodels.api as sm

# Definir variables independientes (X) y dependiente (y)
```

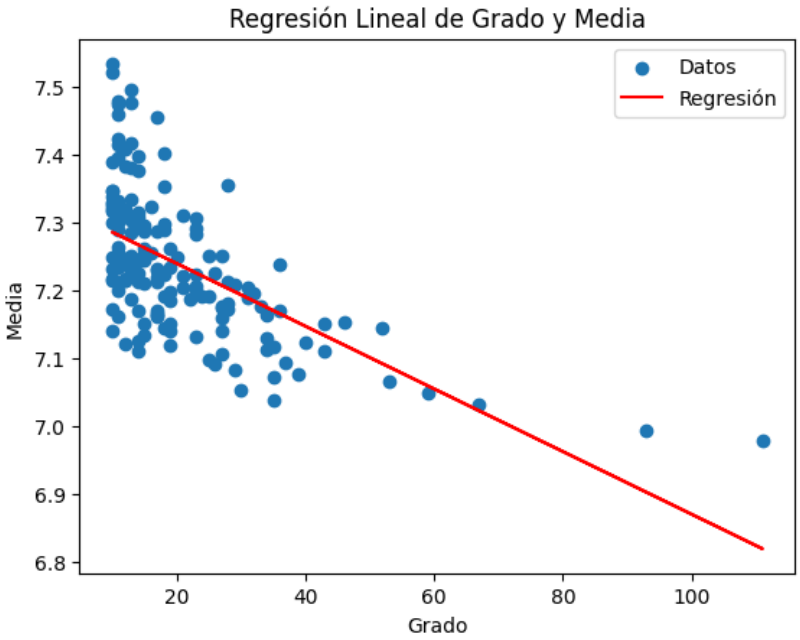
Usuario	Media	Grado
111155	7.53	10
149152	7.52	10
312395	7.49	13
28797	7.48	11
52752	7.48	13
⋮	⋮	⋮
253111	7.05	59
260925	7.04	35
284292	7.03	67
263012	6.99	93
251620	6.98	111

Tabla 1: Datos de usuarios con media y grado.

```
X = df_medias_usuarios[[ 'Grado']]
y = df_medias_usuarios['Media']
# Añadir constante a X (intercepto)
X = sm.add_constant(X)

# Ajustar el modelo de regresión lineal
modelo = sm.OLS(y, X).fit()

# Obtener los coeficientes de la regresión
coeficientes = modelo.params
```



9.3.2 Análisis de las Proyecciones

Siguiendo los mismos procedimientos presentados en la Sección 9.2.4, se pueden crear y analizar las dos proyecciones de la red de Red Usuario - Anime: MyAnimeList. Esto producirá una red de usuarios donde los usuarios tienen conexiones si vieron el mismo anime, y una red de animes donde los animes están conectados si fueron vistos por el mismo usuario. La proyección de usuarios tendrá así 500 nodos, y la red de proyección de animes tendrá 135 nodos.

```
projection=gm.bipartite_projection()
```

```
g_animes=projection[0]
ig.summary(g_animes)

IGRAPH UNWT 135 7582 --
+ attr: Bool type (v), color (v), name (v),, vertex_frame_color (v), weight (e)

g_usuarios=projection[1]
ig.summary(g_usuarios)

IGRAPH UNWT 500 66933 --
+ attr: Bool type (v), color (v),type (v), vertex_frame_color (v), weight (e)

calificacion=[]
for name in g_animes.vs['name']:
   calificacion.append(anime.query('Name == @name').reset_index()['Score'][0])

g_animes.vs['calificacion']=calificacion
```

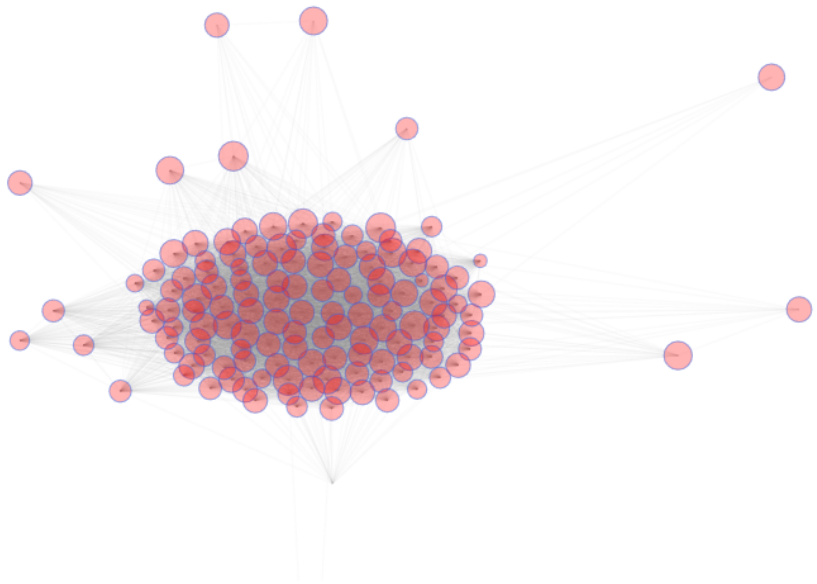


Figura 17: Red decorada por calificación

En esta figura, se presenta toda la red de animes, con el tamaño de los nodos basado en el rating o calificación, de modo que los animes más populares tienen nodos más grandes

```
random.seed(4212)
ig.plot(g_animes,
        edge_color =adjustcolor('gray',0.035),
        vertex_frame_color =adjustcolor('royalblue',0.5),
        vertex_shape='circle',
       vertex_size=[ 2.5*d for d in g_animes.vs['calificacion']])
```

Algunas descripciones básicas de la red proporcionan más información sobre la **Red Usuario - Anime: MyAnimeList**. Se puede observar que no hay ningún anime aislado, por lo tanto, estos forman una gran componente conectada.

```
g_animes.density()

0.838253178551686
```

```
len(g_animes.clusters())
```

```
1
```

La red completa de animes puede filtrarse para examinar el único gran componente como no hay componentes aisladas vamos a mirar las componentes que tengan un rating o calificacion mayor a 7.8. En la próxima figura, el ancho del borde se ha establecido igual a la raíz cuadrada del atributo de borde weight. Esto hace que los vínculos sean más gruesos para los animes que comparten mas usuarios entre ellas

```
i=0
vertices_8=[]
for c in g_animes.vs['calificacion']:
    if c > 7.8:
        vertices_8.append(i)
        i=i+1
    else :
        i=i+1
```

La siguiente figura de la red tiene una densidad relativamente alta, lo que hace que sea algo difícil interpretar cualquier característica estructural interesante. Para ayudar con eso, podemos identificar los núcleos de mayor densidad del grafo y usar eso para .acercarnos.a la parte más interconectada de la red. (Consulte el Capítulo 8 para obtener más información).

Esta red es lo suficientemente pequeña como para que podamos agregar etiquetas de nodos para ayudar con la interpretación. Esto nos ayuda a ver que las secciones más estrechamente conectadas de la red corresponden a series o películas populares, en particular, parece que hay un camino con componentes que tienen un peso mayor a los demás.

```
random.seed(4212)
```

```
ig.plot(g_animes.induced_subgraph(vertices_8),
        edge_color=adjustcolor('gray', 0.2),
        vertex_shape='circle',
        vertex_size=[2.5 * d for d in g_animes.induced_subgraph(vertices_8)
        .vs['calificacion']],
        vertex_label=g_animes.induced_subgraph(vertices_8).vs['name'],
        vertex_label_cex=1,
        edge_width=[0.1 * w for w in g_animes.es['weight']],
        vertex_label_color="purple",
        vertex_label_dist=0.8)
```

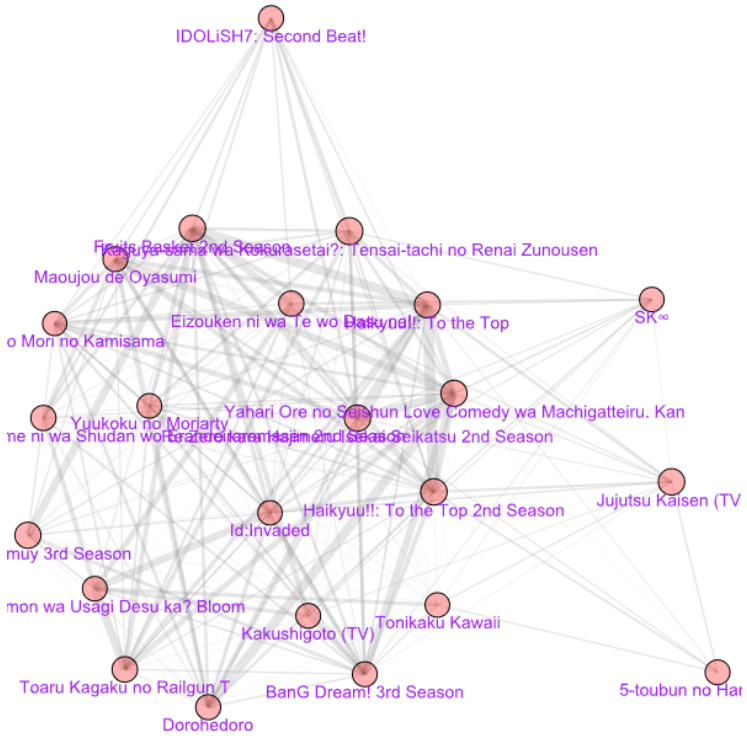


Figura 18: Zoom Grafo Animes Proyección

3. **Considere la base de datos de su interés que reportó en el Taller 2.**

   (a) Caracterizar la centralidad de los nodos.

   (b) Visualizar la red con un diseño adecuado teniendo en cuenta la centralidad de los nodos.

   (c) Identificar los puntos de articulación, los puntos aislados y las componentes.

   (d) Hacer la distribución de las distancia geodésica.

   (e) Determinar si la red es libre de escala.

   (f) Hacer un censo de los clanes y calcular el número clan.

   (g) Calcular la densidad junto con el coeficiente de agrupamiento de la red.

   (h) Particionar la red usando tres métodos de agrupamiento de su elección. Visualizar los resultados obtenidos.

   (i) Hacer un análisis de asortatividad de la red.

   (j) Interpretar los resultados.

Para caracterizar la centralidad de los nodos, vamos a calcular las tres medidas de centralidad, es decir: centralidad de cercanía, centralidad de intermediación y centralidad propia. Se calcularon sus versiones normalizadas y ponderadas para realizar una comparación entre todas. **Top 5 Animes con Mayor Closeness Weighted:**

No.	Name	Closeness Weighted
1	Oshiri Tantei 4th Season	0.85
2	Zhandou Wang Zhi Jufeng Zhan Hun 5: Heti Fanwa...	0.82
3	Semi wa Magic Cube 2nd Season	0.58
4	Kami no Tou	0.52
5	Re:Zero kara Hajimeru Isekai Seikatsu 2nd Season	0.52

Top 5 Animes con Mayor Closeness Unweighted:

No.	Name	Closeness Unweighted
1	Re:Zero kara Hajimeru Isekai Seikatsu 2nd Season	0.98
2	Kami no Tou	0.98
3	Kaguya-sama wa Kokurasetai?: Tensai-tachi no R...	0.98
4	The God of High School	0.97
5	Kanojo, Okarishimasu	0.97

Top 5 Animes con Mayor Betweenness Weighted:

No.	Name	Betweenness Weighted
1	Oshiri Tantei 4th Season	5,405.00
2	Zhandou Wang Zhi Jufeng Zhan Hun 5: Heti Fanwa...	4,427.91
3	Bite-Choicar	764.73
4	Yakusoku no Neverland 2nd Season	536.64
5	Semi wa Magic Cube 2nd Season	414.35

Top 5 Animes con Mayor Betweenness Unweighted:

No.	Name	Betweenness Unweighted
1	Re:Zero kara Hajimeru Isekai Seikatsu 2nd Season	215.27
2	Kami no Tou	172.31
3	Honzuki no Gekokujou: Shisho ni Naru Tame ni w...	113.38
4	Kaguya-sama wa Kokurasetai?: Tensai-tachi no R...	93.24
5	Dungeon ni Deai wo Motomeru no wa Machigatteir...	92.74

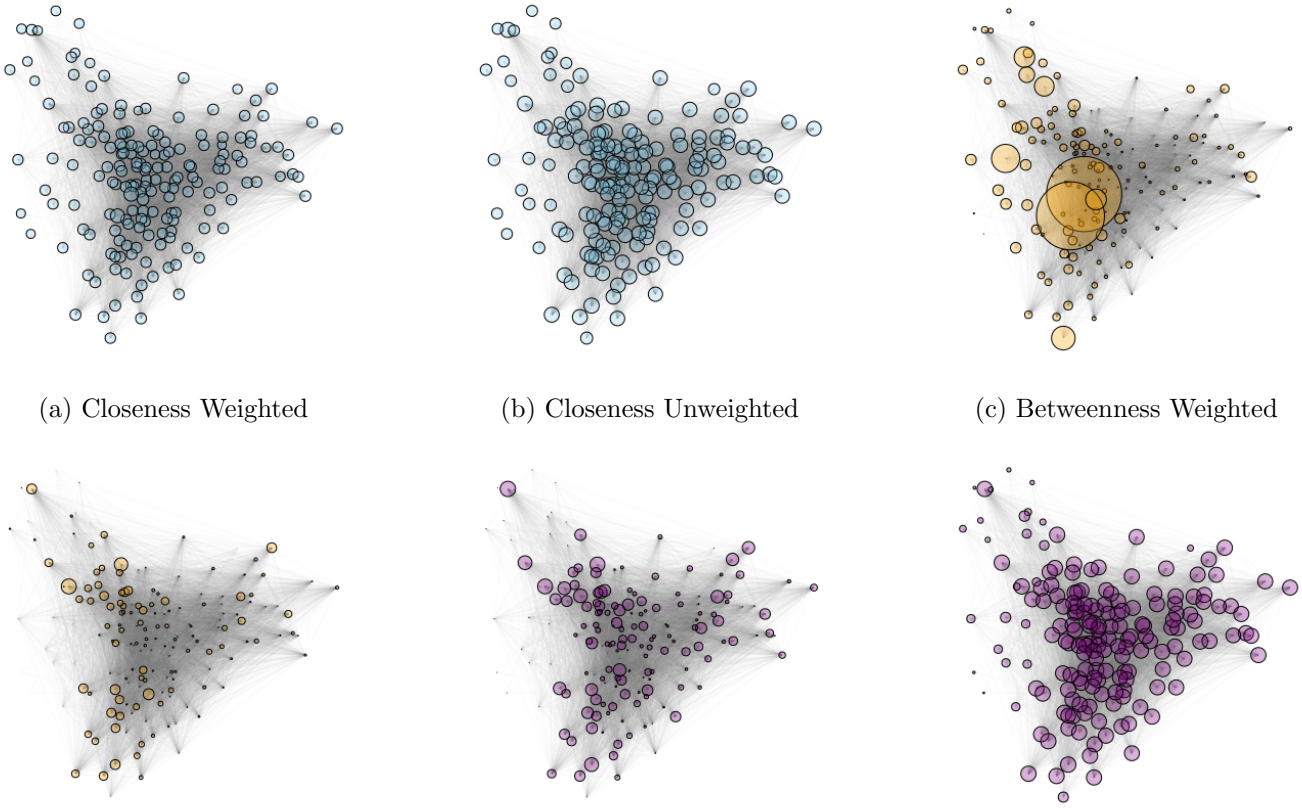
Top 5 Animes con Mayor Eigenvector Weighted:

No.	Name	Eigenvector Weighted
1	Kaguya-sama wa Kokurasetai?: Tensai-tachi no R...	1.00
2	Re:Zero kara Hajimeru Isekai Seikatsu 2nd Season	0.94
3	Kami no Tou	0.93
4	The God of High School	0.82
5	Maou Gakuin no Futekigousha: Shijou Saikyou no...	0.81

Top 5 Animes con Mayor Eigenvector Unweighted:

No.	Name	Eigenvector Unweighted
1	Kaguya-sama wa Kokurasetai?: Tensai-tachi no R...	1.00
2	Re:Zero kara Hajimeru Isekai Seikatsu 2nd Season	1.00
3	Kami no Tou	1.00
4	The God of High School	1.00
5	Kanojo, Okarishimasu	1.00

Se puede observar que dependiendo de la medida de centralidad, algunos animes parecen más “importantes” que otros. Por lo tanto, para resumir adecuadamente esta información, sería apropiado realizar un análisis multivariado, como un Análisis de Componentes Principales (ACP), para resumir esta información.



(a) Closeness Weighted (b) Closeness Unweighted (c) Betweenness Weighted

(d) Betweenness Unweighted (e) Eigenvector Weighted (f) Eigenvector Unweighted

Figura 19: Medidas de Centralidad

Se observa que no hay componentes aisladas, por lo tanto, el grafo tiene una sola componente. Al observar el grafo, esta característica es clara.

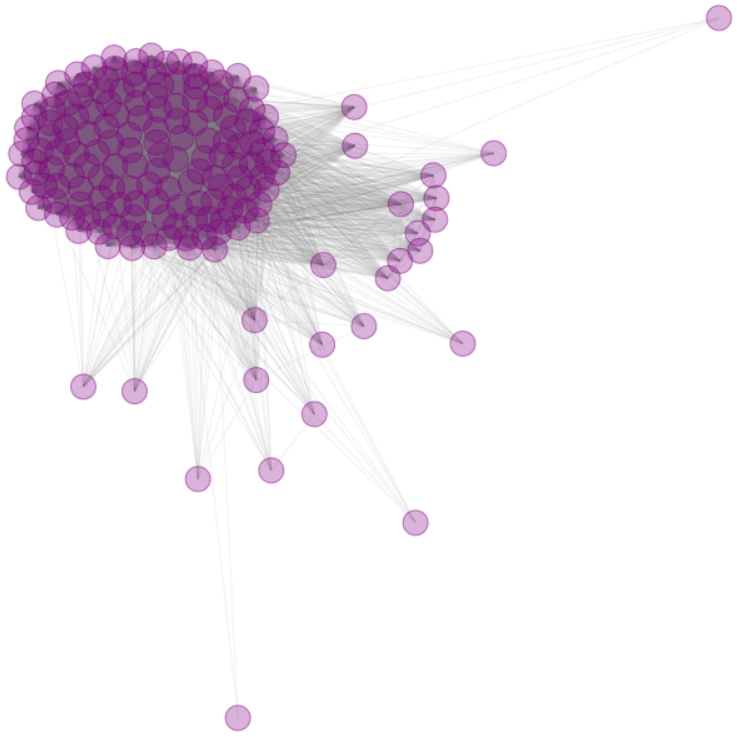
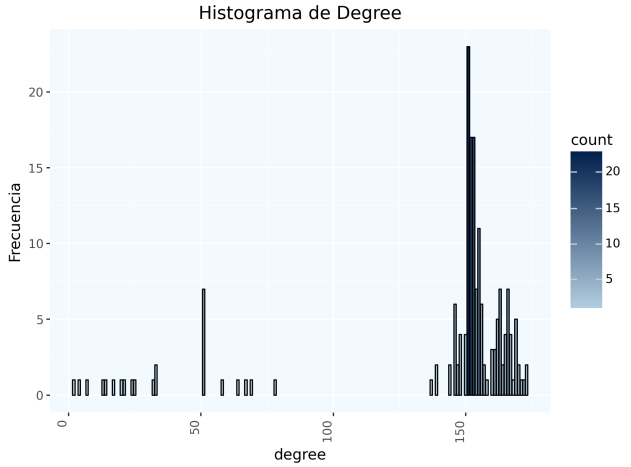
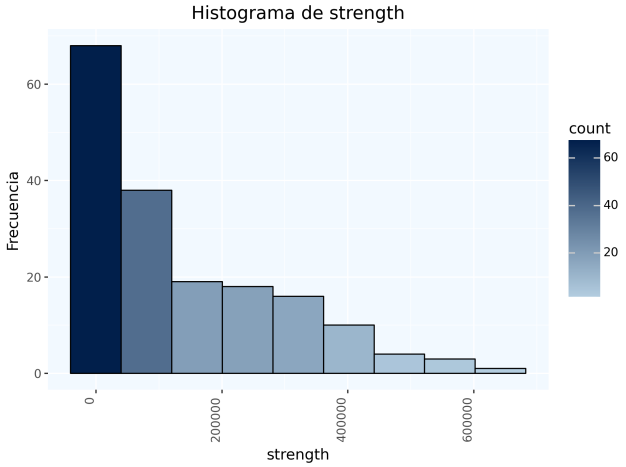


Figura 20: Grafo proyección Animes

También se puede determinar la conectividad del grafo y se encuentra que es 2-conectado, por lo tanto, no existen puntos de articulación.



(a) Distribución del grado



(b) Histograma de la Fuerza

Figura 21: Grado (distancia geodésica) y Fuerza

Se puede observar que la red no sigue una distribución libre de escala, incluso sin pasarla a escala logarítmica. También se puede notar que la mayoría de la fuerza se concentra en valores cercanos a cero.

Ahora al hacer un censo de los clanes podemos notar que hay un clan maximal de tamaño 147 lo que indica que la red es altamnente conectadoa como habia mostrado la densidad que es de 0.79 y la transitividad que es de 0.95.

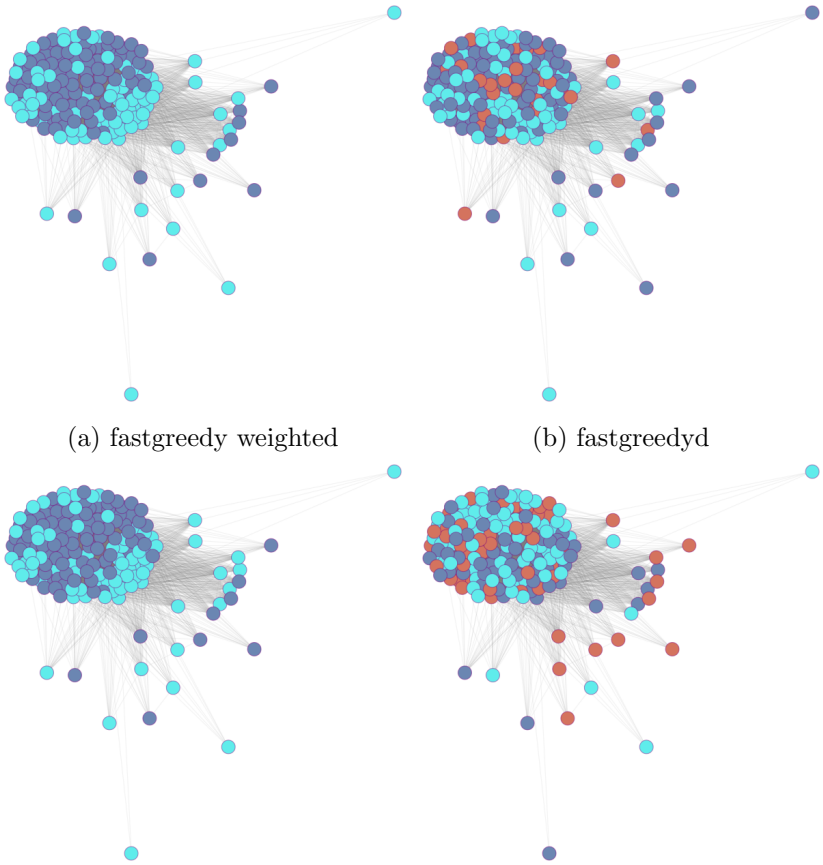
Tabla 2: Modularidad de diferentes algoritmos de detección de comunidades

Algoritmo	Modularidad
Fastgreedy	0.018
Edge Betweenness	0.003
Infomap	0.0
Label Propagation	0.0
Leading Eigenvector	0.014
Fastgreedy (ponderado)	0.072
Infomap (ponderado)	0.0
Leading Eigenvector (ponderado)	0.071

Al realizar varios métodos de agrupamiento tanto ponderados como no ponderados, se puede observar que la modularidad es bastante baja. Esto podría deberse a que estamos tomando partes de la red que se parecen mucho entre sí. Una posible solución sería realizar un muestreo de varias partes de esta red. Sin embargo, se decidió graficar los métodos de agrupamiento que tienen una mayor modularidad.

Finalmente, al calcular la asortatividad asociada con los grados de los vértices del grafo, se obtiene un coeficiente de -0.067, lo que indica que la red no es homofílica. Esto podría estar relacionado con los resultados de la clasificación.

En Conclusión tras analizar detenidamente la red, se pueden extraer conclusiones importantes. En primer lugar, al calcular la centralidad de los nodos, tanto en términos de cercanía, intermediación y propia, se destacó la influencia de ciertos nodos clave en la estructura general de la red. Además, al examinar la conectividad y los componentes de la red, se encontró una alta cohesión estructural al no detectar puntos de articulación y al constatar que la red es 2-conectada.En cuanto a la distribución de grados, se observó que la red no sigue una distribución libre de escala. Los métodos de agrupamiento revelaron una baja modula-



(a) fastgreedy weighted (b) fastgreedyd

(c) leading eigenvector weighted (d) leading eigenvector

Figura 22: Agrupamientos con mayor modularidad

ridad en la red, indicando similitudes entre sus partes, lo que podría abordarse mediante muestreo u otros métodos de agrupamiento. Finalmente, al calcular el coeficiente de asortatividad, se encontró que la red no es homofílica respecto al grado, lo que puede estar relacionado con los resultados de la clasificación y en la estructura general de la red.