Tutorial de igraph (analisis de redes sociales)

Análisis de una comunidad de windsurfers

Para este tutorial vamos a utilizar los datos de dos comunidades de windsurfers en una playa del sur de California, en 1986. Los investigadores que llevaron a cabo este trabajo observaron a un grupo de 43 windsurfers durante 31 días. A partir de estas observaciones determinaron la red de contactos y además al final de las observaciones realizaron una encuesta para determinar cuáles eran las relaciones que reportaban los individuos observados.

Es decir que se pueden construir dos redes, una con las interacciones observadas y otra con las interacciones reportadas o percibidas.

Los investigadores encontraron que esta comunidad de windsurfers estaba dividida en dos sub-comunidades. Los objetivos de nuestro análisis van a ser explorar estas redes, realizar una caracterización topológica, e intentar demostrar la presencia de las sub-comunidades.

Pasos previos

Si no está instalada, necesitamos instalar igraph

Preparación de los datos

Instalar y/o cargar el paquete igraph

```
# Si igraph está instalado, lo carga; si no, lo instala y luego lo carga
if(!require(igraph)) install.packages("igraph"); require(igraph)
```

```
## Loading required package: igraph
```

```
##
## Attaching package: 'igraph'
```

```
## The following objects are masked from 'package:stats':
##
##     decompose, spectrum
```

```
## The following object is masked from 'package:base':
##
##     union
```

```
if(!require(ggplot2)) install.packages("ggplot2"); require(ggplot2)
```

```
## Loading required package: ggplot2
```

Descargar los datos

Una breve explicación del dataset está disponible aqui. Los datos están disponibles en formato DL, pero debido a que no hay una especificación formal de este formato, el parser de igraph no puede leer correctamente este dataset. Lo vamos a procesar a mano.

La información está registrada en dos matrices de adyacencia, una a continuación de la otra. La primera indica las interacciones observadas y la segunda las percibidas.

Las primeras siete lineas del archivo contienen información sobre estas matrices:

DL N=43 NM=2 FORMAT = FULLMATRIX DIAGONAL PRESENT LEVEL LABELS: “bb” “bc” DATA:

Lo importante aqui es que hay 43 individuos, que ambas matrices son simétricas, esto va a determinar un grafo no dirigido. Las primeras siete lineas la vamos a saltear y leemos los datos en una dataframe.

```
download.file("http://moreno.ss.uci.edu/beach.dat", destfile = "windsurfers.dat")
ws <- read.table("windsurfers.dat", skip = 7)
dim(ws)
```

```
## [1] 86 43
```

Ahora tenemos que separar las dos matrices

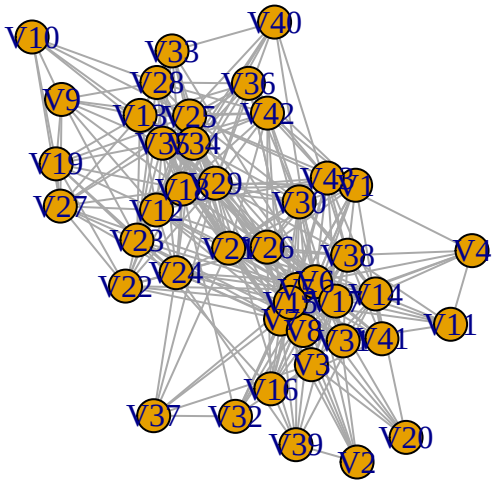
```
ws.obs <- as.matrix(ws[1:43, ])
ws.per <- as.matrix(ws[44:86, ])
```

Construcción de las redes

La primera matriz tiene números enteros que representan el tiempo que conversaron los individuos i y j. La segunda matriz tiene números reales que representan la percepción de la cercanía que tiene el grupo sobre dos dados individuos.

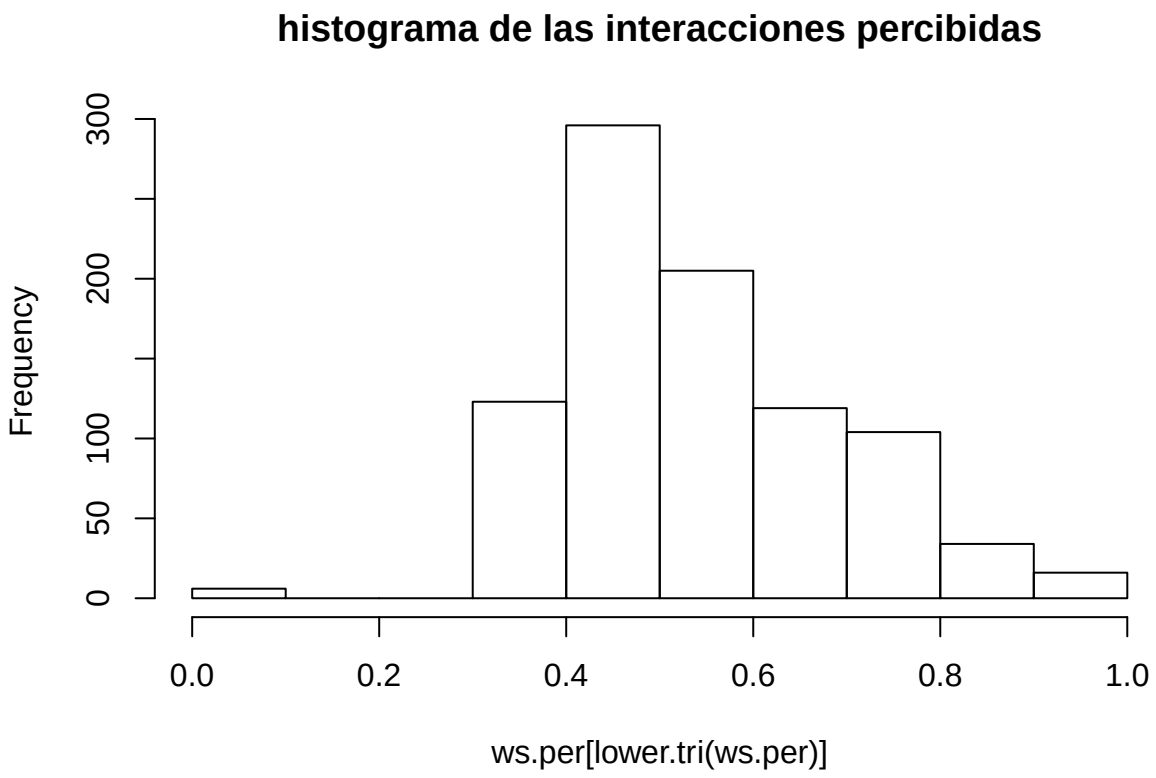
La diagonal principal de la matriz ws.obs contiene datos, pero no existe información sobre su significado.

```
ws.obs.red <- graph.adjacency(ws.obs, mode="undirected", diag=FALSE, weighted = T)
plot(ws.obs.red)
```

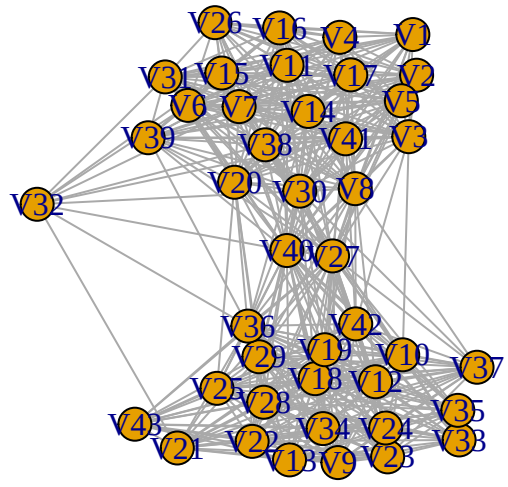


La métrica para medir la cercanía se obtiene a partir de cuestionarios. Este valor varía entre 0 y 1. Un problema con esta codificación es que aun para individuos con muy poca interacción percibida esta variable va a tomar un valor cercano a cero, pero no cero. Al construir el grafo se tenderá una arista entre dos individuos, cuando en la practica esa interacción debería considerarse nula. Para solucionar esto vamos a considerar que cuando el valor de interacción percibida es menor de 0.5 lo convertimos a cero; esto es, no hay una interacción efectiva.

```
hist(ws.per[lower.tri(ws.per)], main = "histograma de las interacciones percibidas")
```




```
umbral <- 0.5
ws.per.2 <- ws.per
ws.per.2[which(ws.per.2 <= umbral)] <- 0
ws.per.red <- graph.adjacency(ws.per.2, mode="undirected", diag=FALSE, weighted = T)
plot(ws.per.red)
```



Caracterización y análisis de la red

Características generales, número de nodos (V) y aristas (E)

```
summary(ws.obs.red)
```

```
## IGRAPH 8f308b4 UNW- 43 336 --
```

```
## + attr: name (v/c), weight (e/n)
```

```
summary(ws.per.red)
```

```
## IGRAPH 8904e04 UNW- 43 478 --
## + attr: name (v/c), weight (e/n)
```

```
# Para información más detallada
# str(ws.obs.red)

# Si necesito sólo los recuentos
vcount(ws.obs.red)
```

```
## [1] 43
```

```
ecount(ws.obs.red)
```

```
## [1] 336
```

```
# Para ver los nodos y aristas
V(ws.obs.red)
```

```
## + 43/43 vertices, named, from 8f308b4:
##  [1] V1  V2  V3  V4  V5  V6  V7  V8  V9  V10 V11 V12 V13 V14 V15 V16 V17
## [18] V18 V19 V20 V21 V22 V23 V24 V25 V26 V27 V28 V29 V30 V31 V32 V33 V34
## [35] V35 V36 V37 V38 V39 V40 V41 V42 V43
```

```
E(ws.obs.red)
```

```
## + 336/336 edges from 8f308b4 (vertex names):
##  [1] V1--V5  V1--V6  V1--V8  V1--V15 V1--V16 V1--V18 V1--V21 V1--V25
##  [9] V1--V26 V1--V29 V1--V30 V1--V36 V1--V38 V1--V41 V1--V42 V2--V3
## [17] V2--V5  V2--V6  V2--V7  V2--V8  V2--V17 V2--V39 V3--V5  V3--V6
## [25] V3--V7  V3--V8  V3--V14 V3--V15 V3--V17 V3--V20 V3--V21 V3--V30
## [33] V3--V32 V3--V36 V3--V37 V3--V38 V4--V6  V4--V7  V4--V11 V4--V14
## [41] V4--V17 V4--V38 V4--V43 V5--V6  V5--V7  V5--V8  V5--V11 V5--V12
## [49] V5--V14 V5--V15 V5--V16 V5--V17 V5--V18 V5--V21 V5--V22 V5--V23
## [57] V5--V24 V5--V26 V5--V29 V5--V30 V5--V31 V5--V32 V5--V38 V5--V39
## [65] V5--V41 V5--V43 V6--V7  V6--V8  V6--V11 V6--V14 V6--V15 V6--V16
## [73] V6--V17 V6--V18 V6--V20 V6--V21 V6--V24 V6--V25 V6--V26 V6--V29
## + ... omitted several edges
```

```
V(ws.per.red)
```

```
## + 43/43 vertices, named, from 8904e04:
##  [1] V1  V2  V3  V4  V5  V6  V7  V8  V9  V10 V11 V12 V13 V14 V15 V16 V17
## [18] V18 V19 V20 V21 V22 V23 V24 V25 V26 V27 V28 V29 V30 V31 V32 V33 V34
## [35] V35 V36 V37 V38 V39 V40 V41 V42 V43
```

```
E(ws.per.red)
```

```
## + 478/478 edges from 8904e04 (vertex names):
##  [1] V1--V3  V1--V4  V1--V5  V1--V6  V1--V7  V1--V8  V1--V11 V1--V14
##  [9] V1--V15 V1--V16 V1--V17 V1--V20 V1--V26 V1--V27 V1--V30 V1--V31
## [17] V1--V38 V1--V41 V2--V3  V2--V4  V2--V5  V2--V6  V2--V7  V2--V8
## [25] V2--V11 V2--V14 V2--V15 V2--V16 V2--V17 V2--V20 V2--V26 V2--V27
## [33] V2--V30 V2--V31 V2--V38 V2--V39 V2--V40 V2--V41 V3--V4  V3--V5
## [41] V3--V6  V3--V7  V3--V8  V3--V10 V3--V11 V3--V14 V3--V15 V3--V16
## [49] V3--V17 V3--V20 V3--V26 V3--V27 V3--V30 V3--V31 V3--V38 V3--V39
## [57] V3--V40 V3--V41 V3--V42 V4--V5  V4--V6  V4--V7  V4--V8  V4--V11
```

```
## [65] V4--V14 V4--V15 V4--V16 V4--V17 V4--V20 V4--V26 V4--V30 V4--V31
## [73] V4--V38 V4--V39 V4--V40 V4--V41 V5--V6  V5--V7  V5--V8  V5--V11
## + ... omitted several edges
```

¿Las redes tienen loops o aristas múltiples? ¿Son redes completamente conectadas?

```
is.simple(ws.obs.red)
```

```
## [1] TRUE
```

```
is.simple(ws.per.red)
```

```
## [1] TRUE
```

```
is.connected(ws.obs.red)
```

```
## [1] TRUE
```

```
is.connected(ws.per.red)
```

```
## [1] TRUE
```

La función *is.connected()* admite un argumento extra "mode", que puede tomar dos valores, "weak" o "strong". Esto se aplica a grafos dirigidos, y sirve para demostrar si el grafo es conectado al considerar aristas sin dirección (weak) o también cuando se considera la dirección ("strong").

Otras características topológicas de la red

¿Cuál es el diámetro de la red? ¿Cuáles son los vertices que determinan ese diámetro? La función para calcular el diámetro tienen en cuenta los pesos de las aristas. La función *get.diameter()* devuelve los nodos que conforman el camino del diámetro máximo.

```
# Red de interacciones observadas
diameter(ws.obs.red)
```

```
## [1] 4
```

```
get.diameter(ws.obs.red)
```

```
## + 3/43 vertices, named, from 8f308b4:
## [1] V2  V5  V23
```

```
# Red de interacciones percibidas
diameter(ws.per.red)
```

```
## [1] 1.626
```

```
get.diameter(ws.per.red)
```

```
## + 4/43 vertices, named, from 8904e04:
## [1] V16 V40 V9  V43
```

La densidad de un grafo es el cociente entre el número de aristas de un grafo y el número de todos las posibles aristas. Como nuestras dos redes tienen el mismo número de nodos, estamos determinando algo que ya habíamos visto antes, la red de relaciones percibidas tiene más conexiones que la red de relaciones observadas.

```
graph.density(ws.obs.red)
```

```
## [1] 0.372093
```

```r
graph.density(ws.per.red)
```

```
## [1] 0.5293466
```

La densidad de la segunda red es alta comparada con lo común para redes sociales.

El coeficiente de clustering global que vimos en clase se calcula en *igraph* con la función *transitivity()*. esta función por defecto utiliza los pesos en los cálculos del clustering. Si al argumento *type* se le pasa el valor "local" calcula los coeficientes de clustering para cada vértice, y con el valor "global", calcula un valor para el grafo completo.

```r
head (transitivity(ws.obs.red, type = "local"))
```

```
## [1] 0.6666667 0.9047619 0.5714286 0.8095238 0.4861538 0.4172043
```

```r
head (transitivity(ws.per.red, type = "local"))
```

```
## [1] 0.9803922 0.9736842 0.8379447 0.9684211 0.9619048 0.9047619
```

```r
# red de interacciones observadas
transitivity(ws.obs.red, type = "global")
```

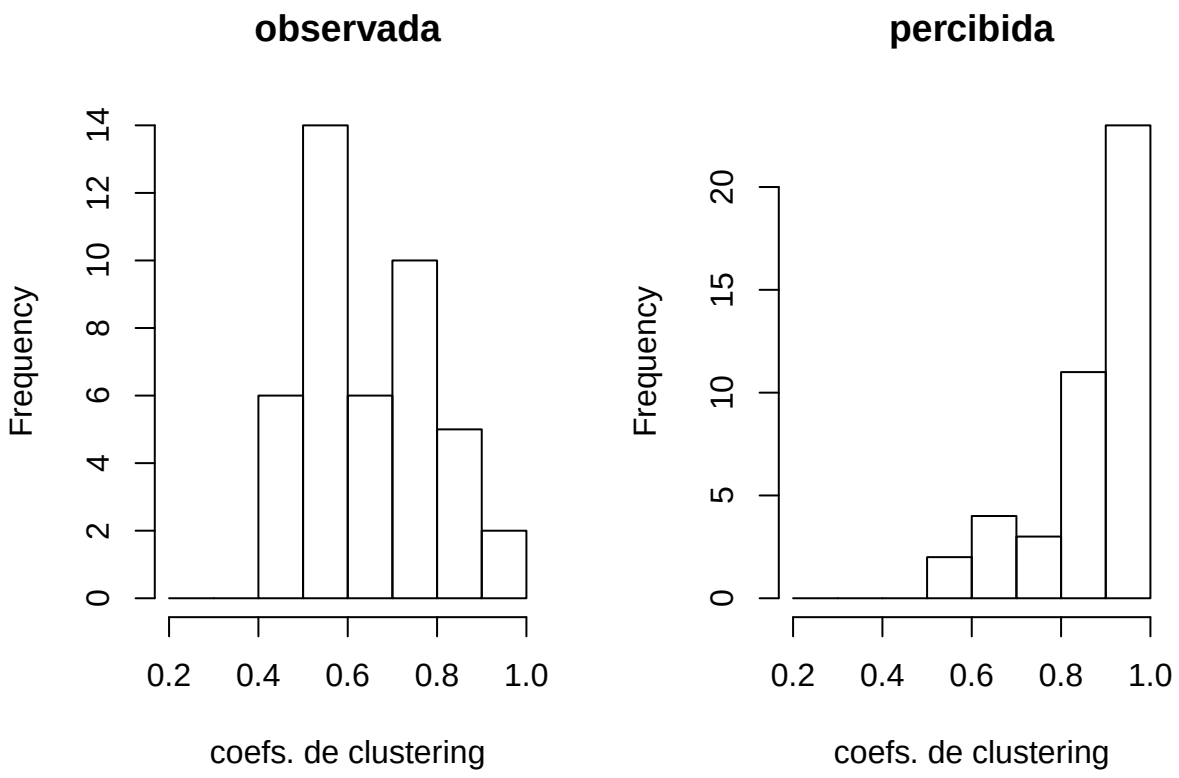
```
## [1] 0.5638827
```

```r
# red de interacciones percibidas
transitivity(ws.per.red, type = "global")
```

```
## [1] 0.8194929
```

La red de interacciones percibidas tiene un coeficiente de clustering mayor que la observada. indicando que los individuos asumen una intensidad de interacciones mayor que la registrada durante los 31 días de observaciones. En particular lo que podemos ver es que en la red de interacciones percibidas hay bastantes individuos para quienes el grupo supone que tienen contactos que a su vez están muy conectados entre sí.

```r
par(mfrow = c(1,2))
hist(transitivity(ws.obs.red, type = "local"), main = "observada",
     breaks = seq(0.2, 1, 0.1), xlab = "coefs. de clustering")
hist(transitivity(ws.per.red, type = "local"), main = "percibida",
     breaks = seq(0.2, 1, 0.1), xlab = "coefs. de clustering")
```




```
# El display gráfico vuelve a la configuración de un gráfico por panel
par(mfrow = c(1,1))
```

Grados totales, de entrada y de salida

```
degree(ws.obs.red)
```

```
##  V1  V2  V3  V4  V5  V6  V7  V8  V9 V10 V11 V12 V13 V14 V15 V16 V17 V18
##  15   7  15   7  26  31  28  17  10   7   7  19  19  17  25  11  18  26
## V19 V20 V21 V22 V23 V24 V25 V26 V27 V28 V29 V30 V31 V32 V33 V34 V35 V36
##  11   6  21  12  17  19  23  25  12  13  24  17  12   8  10  19  20  13
## V37 V38 V39 V40 V41 V42 V43
##   7  20   9   8  12  15  14
```

```
sort(degree(ws.obs.red), decreasing = T)
```

```
##  V6  V7  V5 V18 V15 V26 V29 V25 V21 V35 V38 V12 V13 V24 V34 V17  V8 V14
##  31  28  26  26  25  25  24  23  21  20  20  19  19  19  19  18  17  17
## V23 V30  V1  V3 V42 V43 V28 V36 V22 V27 V31 V41 V16 V19  V9 V33 V39 V32
##  17  17  15  15  15  14  13  13  12  12  12  12  11  11  10  10   9   8
## V40  V2  V4 V10 V11 V37 V20
##   8   7   7   7   7   7   6
```

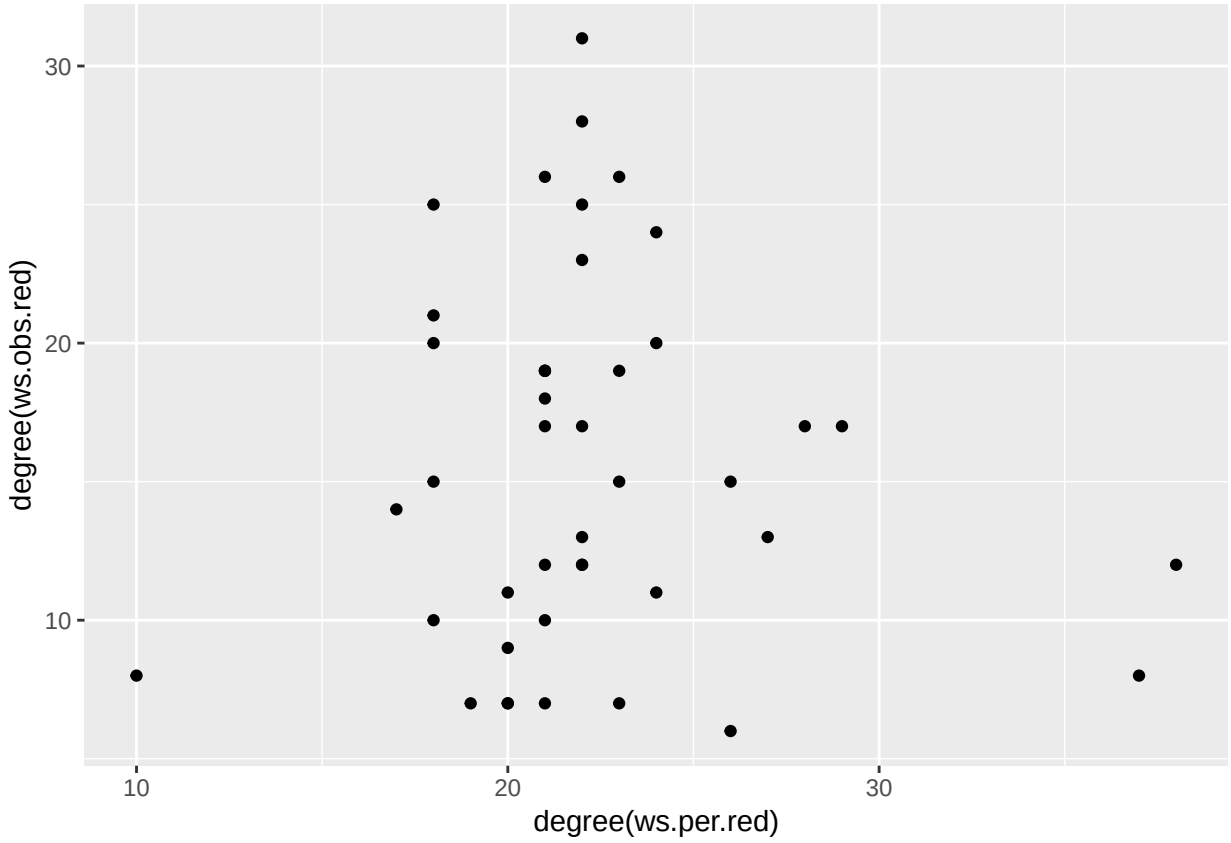
Si el grafo fuese dirigido también podríamos consultar los grados de entrada y de salida.

```
degree(ws.obs.red, mode = "in")
degree(ws.obs.red, mode = "out")
```

Uno de los objetivos originales de este paper era mostrar que había similitud entre las interacciones observadas y las cercanías percibidas. Podemos probar esto con un gráfico y analizando la correlación.

Podemos comenzar viendo qué sucede a nivel de grados:

```
qplot(degree(ws.per.red), degree(ws.obs.red))
```




```
cor(degree(ws.per.red), degree(ws.obs.red))
```

```
## [1] -0.04875002
```

No se observa una relación entre el grado que predice el grupo para cada individuo, y el grado que se observado.

Vamos a analizar con más detalle las distribuciones de grado. La función *degree.distribution()* calcula las distribuciones de los grados y las distribuciones acumuladas. A partir de éstas, podemos realizar los correspondientes gráficos.

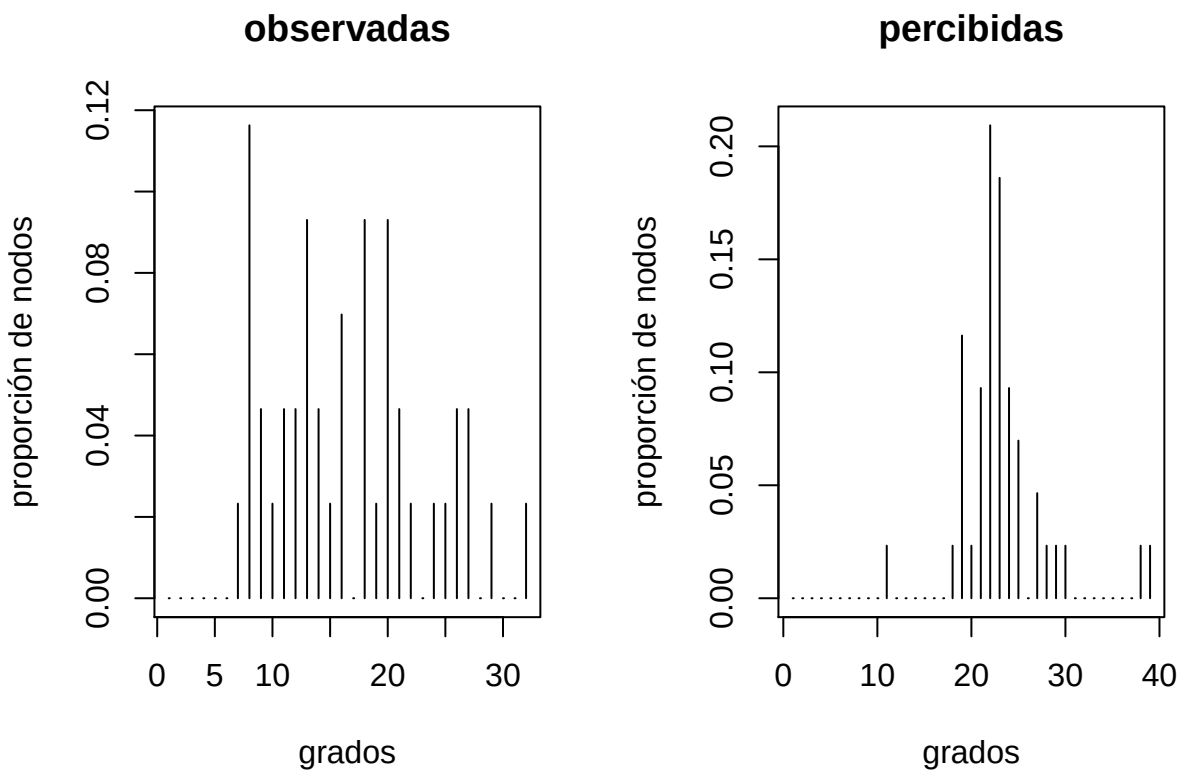
```
head( degree.distribution(ws.obs.red ), 15)
```

```
##  [1] 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
##  [7] 0.02325581 0.11627907 0.04651163 0.02325581 0.04651163 0.04651163
## [13] 0.09302326 0.04651163 0.02325581
```

```
head( degree.distribution(ws.obs.red, cumulative = T ))
```

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

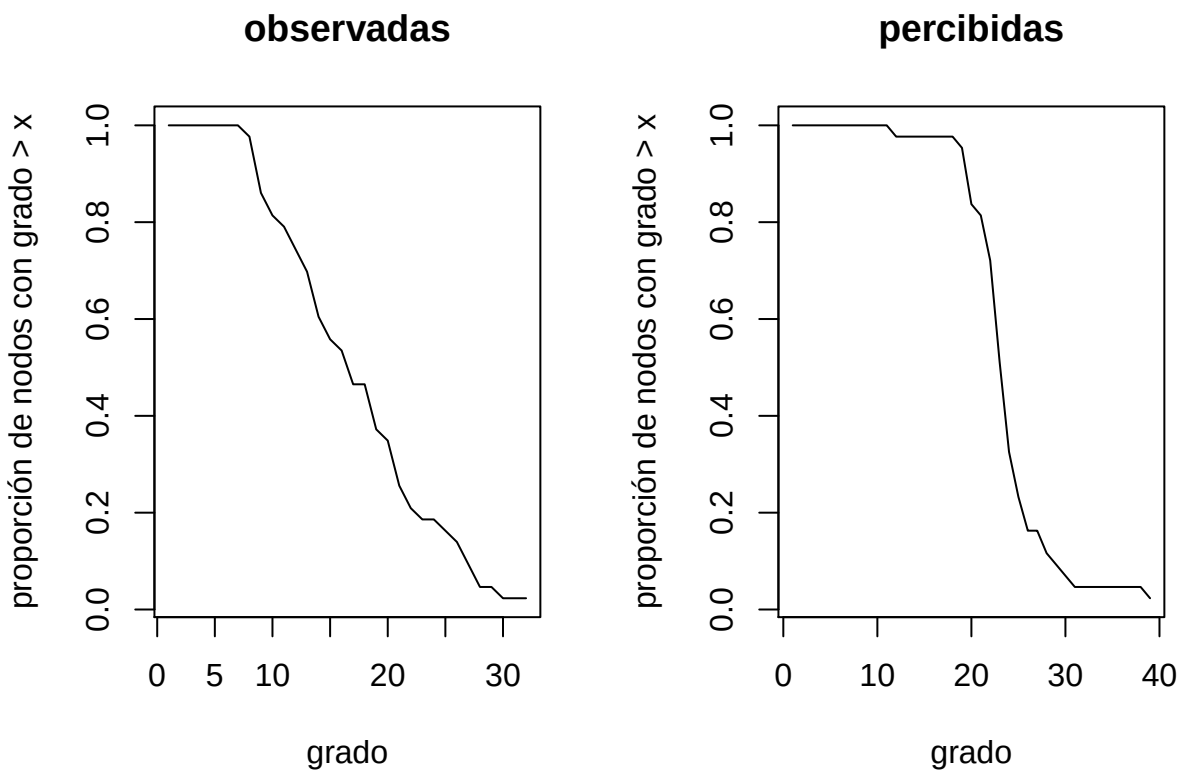
```
par( mfrow = c(1,2) )
plot( degree.distribution(ws.obs.red),
      xlab = "grados", ylab = "proporción de nodos", type = "h", main = "observadas")

plot( degree.distribution(ws.per.red),
      xlab = "grados", ylab = "proporción de nodos", type = "h", main = "percibidas")
```




```
# El display gráfico vuelve a la configuración de un gráfico por panel
par( mfrow = c(1,2) )
```

En ambas redes vemos que hay un número bajo de nodos de alto grado y varios de grado menor. Los siguientes gráficos muestran como se observa este fenómeno en los gráficos de distribuciones acumuladas. También podemos ver que la variación en ambas curvas no es la misma.

```
par( mfrow = c(1,2) )

plot( degree.distribution(ws.obs.red, cumulative = T), type = "l", xlab = "grado", ylab = "proporción d

plot( degree.distribution(ws.per.red, cumulative = T), type = "l", xlab = "grado", ylab = "proporción d
```




```
# El display gráfico vuelve a la configuración de un gráfico por panel
par( mfrow = c(1,1))
```

Mundos pequeños

Estas redes son relativamente chicas como para intentar probar si son de mundo pequeño, pero podemos probar.

```
ws.obs.red.plf <- power.law.fit(degree(ws.obs.red))
ws.per.red.plf <- power.law.fit(degree(ws.per.red))
```

¿El parámetro alfa de la función es mayor que 1? ¿Si? OK

```
ws.obs.red.plf$alpha
```

```
## [1] 10.6483
```

```
ws.per.red.plf$alpha
```

```
## [1] 8.001056
```

¿El test de ajuste de Kolmogorov-Smirnov es no significativo? OK.

```
ws.obs.red.plf$KS.p
```

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

```
ws.per.red.plf$KS.p
```

```
## [1] 0.9999221
```

Los valores de *xmin* se corresponden a valores de grado bajos

```
ws.obs.red.plf$xmin
```

```
## [1] 25
```

```
ws.per.red.plf$xmin
```

```
## [1] 21
```

No son valores bajos. Esto significa que unos pocos valores se utilizaron para hacer el ajuste, posiblemente se deba al pequeño tamaño de estas redes. Podemos forzar la función para que use un valor más bajo.

```
ws.obs.red.plf.2 <- power.law.fit(degree(ws.obs.red), xmin = 9)
ws.obs.red.plf.2$alpha
```

```
## [1] 2.445363
```

```
ws.obs.red.plf.2$KS.p
```

```
## [1] 0.0702021
```

```
ws.per.red.plf.2 <- power.law.fit(degree(ws.per.red), xmin = 17)
# Elegimos 17 porque los grados mínimos en ws.per.red son mayores
ws.per.red.plf.2$alpha
```

```
## [1] 4.309512
```

```
ws.per.red.plf.2$KS.p
```

```
## [1] 0.005393956
```

En este segundo análisis vemos que podemos seguir considerando a la red de interacciones observadas cumple con los requisitos en cuanto a ajuste a una ley de potencias. La otra red, no.

El otro criterio para probar que una red es de mundo pequeño es demostrar que sus coeficientes de clustering son mayores que grafos al azar con características topológicas similares. Esto lo vamos a probar para la red de interacciones observadas simulando 1000 redes al azar.

con la función *barabasi.game()* creamos grafos al azar que siguen el modelo de Barabási–Albert (un modelo de redes libres de escala). Y luego con la función *sample_gnm()* generamos grafos de acuerdo al modelo de Erdos-Renyi, en este caso pasamos como argumento el número de nodos y aristas, y se van creando aristas al azar.

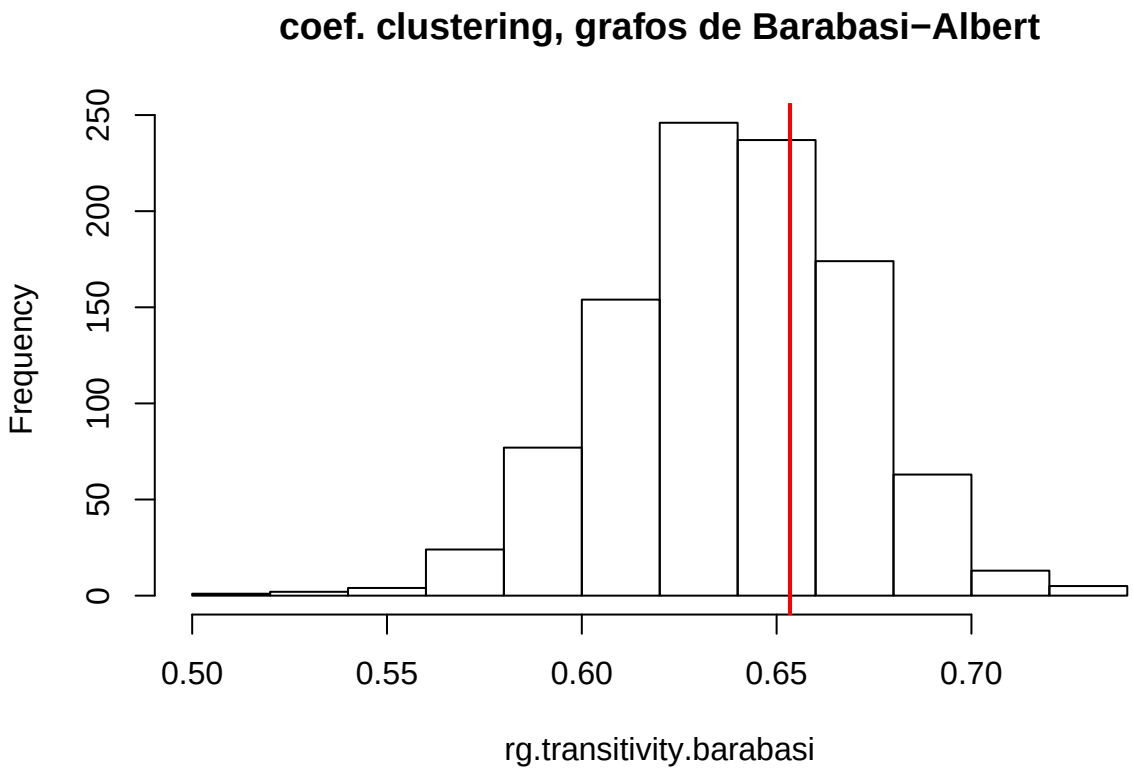
```
rg.transitivity.barabasi <- array()
rg.transitivity.erdos <- array()

for(i in 1:1000){
  rg.1 <- barabasi.game(43, power = 2.4, m=8, directed = F)
  rg.2 <- sample_gnm(43, 336)
  rg.transitivity.barabasi[i] <-  mean(transitivity(rg.1, "local", isolates="zero"))
  rg.transitivity.erdos[i] <- mean(transitivity(rg.2, "local", isolates="zero"))
}
red.transitivity <- mean(transitivity(ws.obs.red, "local", isolates="zero"))
```

La comparación del promedio de los coeficientes de clusters contra los grafos con el modelo de Barabási–Albert:

```
table(red.transitivity > rg.transitivity.barabasi)
```

```
##
## FALSE  TRUE
##   322   678
```

```
hist(rg.transitivity.barabasi, main = "coef. clustering, grafos de Barabasi-Albert")
abline( v = red.transitivity, col ="red", lwd= 2)
```



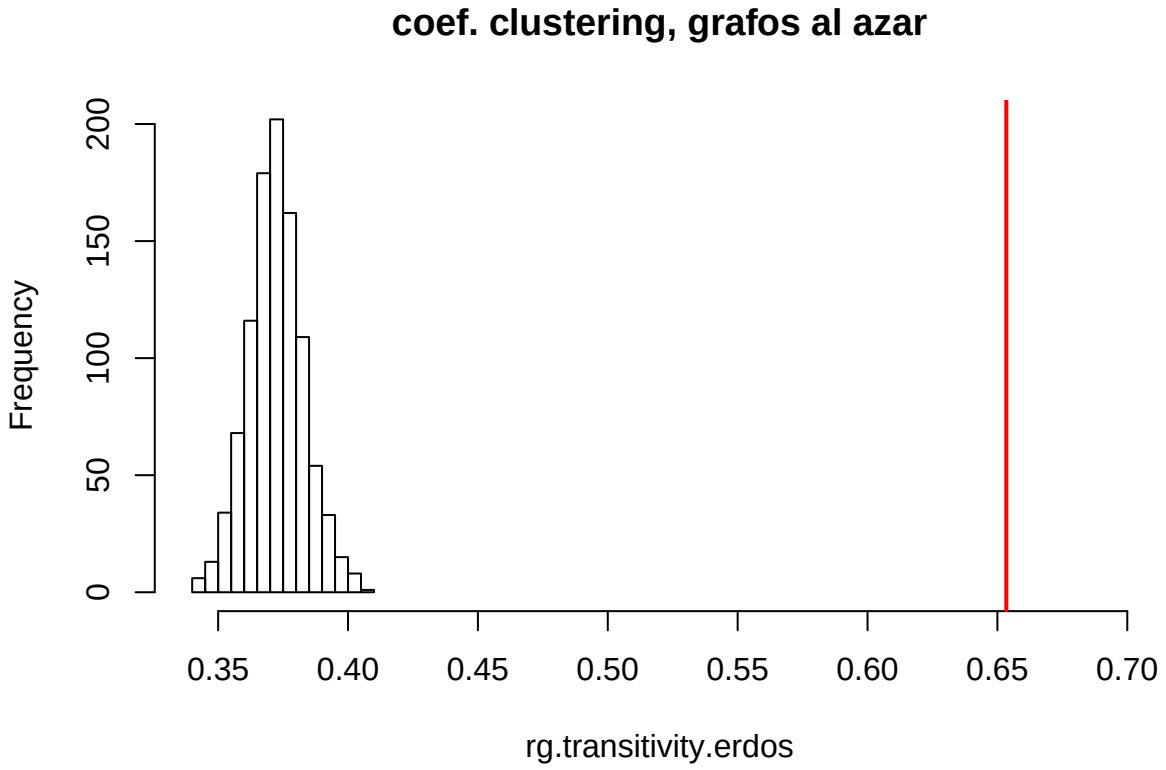
El coeficente de clustering del grafo de interacciones observadas (linea roja) se encuentra dentro del rango de valores de grafos simulados siguiendo un modelo de red libre de escala. Mirando la tabla, el valor de p = 711 / 1000 = 0.711.

Y la comparación contra grafos simulados con el modelo de Erdos-Renyi (grafos al azar):

```
table(red.transitivity > rg.transitivity.erdos)
```

```
##
## TRUE
## 1000
```

```
hist(rg.transitivity.erdos, xlim=c(0.34, 0.7), main = "coef. clustering, grafos al azar")
abline( v = red.transitivity, col ="red", lwd= 2)
```



Vemos que el promedio de los coeficientes de clustering del grafo es significativamente mayor que los de grafos que siguen modelos al azar, tal como se espera para un grafo de mundo pequeño. Si miramos nuevamente la tabla generada, el valor medido es mayor a cualquiera de los valores producidos por el azar, entocnes el valor de $p < 0.001$.

Por lo tanto, basados en el ajuste a una ley de potencias y al al distribución de los valores de coeficientes de clustering no podemos descartar la hipótesis de que la red de interacciones observadas sea un grafo de mundo pequeño.

Cálculo de la asortividad

```
assortativity.degree(ws.obs.red)
```

```
## [1] -0.1469652
```

```
assortativity.degree(ws.per.red)
```

```
## [1] -0.05034805
```

En ambos casos los valores de asortividad sugieren que no hay asociaciones preferenciales por un lado entre nodos de alto grado, y por el otro entre los de bajo grado.

Otras medias de centralidad

Con *igraph* es posible calcular las medias de centralidad que vimos en lcase (ademas de la centralidad de grado):

- Intermediación (betweenness)
- Cercanía (Closeness)
- Centralidad de autovectores (Eigenvector centrality)

En las salidas siguientes se muestran llamadas a las distintas funciones de centralidad, y se muestran los 10 individuos con valores mayores para cada medida y red.

```
# Intermediación
head( sort( betweenness(ws.obs.red), decreasing = T) )
```

```
##      V26      V18      V25      V30       V1      V24
## 76.62455 73.81422 62.78720 44.15074 42.58201 41.64934
```

```
head( sort( betweenness(ws.per.red), decreasing = T) )
```

```
##    V27    V40    V30    V32    V37    V39
## 186.00  85.25  82.75  20.00  13.00   9.50
```

```
# Cercania
head( sort( closeness(ws.obs.red), decreasing = T) )
```

```
##        V26        V18        V25         V1        V30        V14
## 0.01234568 0.01219512 0.01219512 0.01190476 0.01176471 0.01123596
```

```
head( sort( closeness(ws.per.red), decreasing = T) )
```

```
##        V27        V40        V30        V20         V8        V41
## 0.03926959 0.03833033 0.03385355 0.02996973 0.02982582 0.02966743
```

```
# Centralidad de autovectores
head( sort( eigen_centrality(ws.obs.red)$vector, decreasing = T) )
```

```
##        V6        V5       V15        V7        V8       V31
## 1.0000000 0.8164119 0.7831516 0.6447378 0.3896565 0.3491946
```

```
head( sort( eigen_centrality(ws.per.red)$vector, decreasing = T) )
```

```
##       V27       V40       V29       V36       V12       V19
## 1.0000000 0.9925283 0.9911198 0.9826181 0.9813275 0.9778097
```

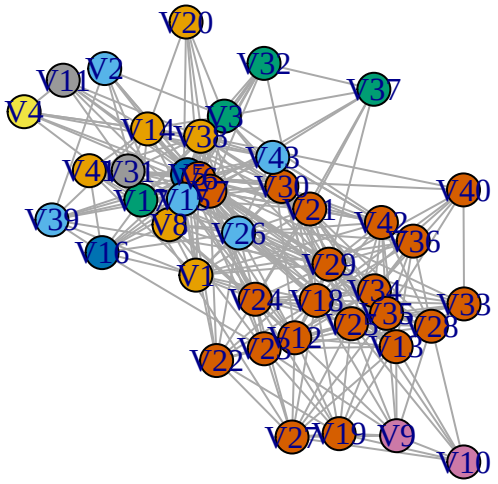
Clustering

```
ws.obs.red.cl.eb <- cluster_edge_betweenness(ws.obs.red, directed = F, merges = T)
```

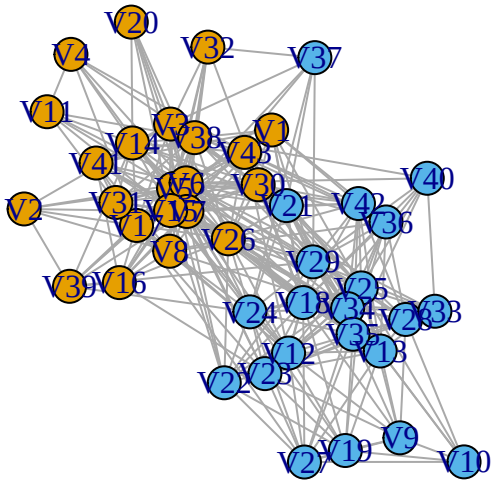
```
## Warning in cluster_edge_betweenness(ws.obs.red, directed = F, merges = T):
## At community.c:460 :Membership vector will be selected based on the lowest
## modularity score.
```

```
## Warning in cluster_edge_betweenness(ws.obs.red, directed = F, merges = T):
## At community.c:467 :Modularity calculation with weighted edge betweenness
## community detection might not make sense -- modularity treats edge weights
## as similarities while edge betwenness treats them as distances
```

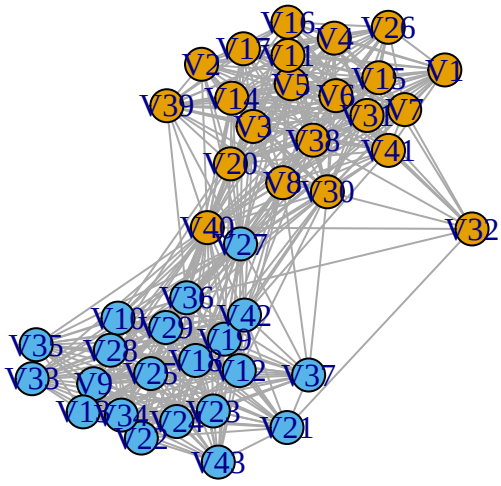
```
plot(ws.obs.red, vertex.color = ws.obs.red.cl.eb$membership)
```



```
ws.obs.red.cl.lo <- cluster_louvain(ws.obs.red, weights = E(ws.obs.red)$weight)
plot(ws.obs.red, vertex.color = ws.obs.red.cl.lo$membership)
```



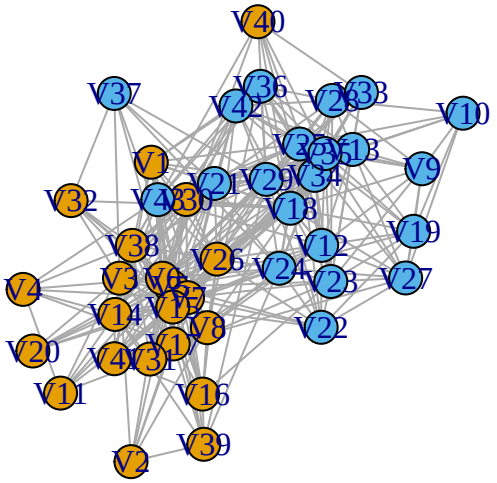
```
ws.per.red.cl.lo <- cluster_louvain(ws.per.red, weights = E(ws.per.red)$weight)
plot(ws.per.red, vertex.color = ws.per.red.cl.lo$membership)
```



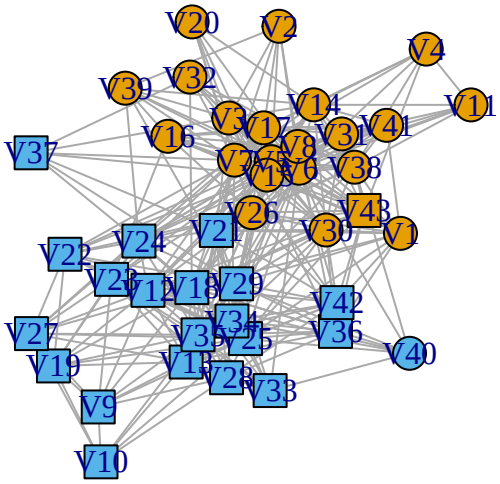
```
table(ws.per.red.cl.lo$membership, ws.obs.red.cl.lo$membership)
```

```
##
##      1  2
##   1 21  1
##   2  1 20
```

```
plot(ws.obs.red, vertex.color = ws.per.red.cl.lo$membership)
```



```
plot(ws.obs.red, vertex.color = ws.obs.red.cl.lo$membership, vertex.shape = ifelse(ws.per.red.cl.lo$mem
```



igraph incluye funciones para realizar análisis de agrupamientos con otros algoritmos. Por ejemplo,

- *cluster_walktrap()*
- *cluster_fast_greedy()*
- *cluster_spinglass()*
- etc. (revisar la documentación del paquete)

Cuando agrupamos los nodos por clusters, o cuando podemos asignar clases o atributos a los nodos nos puede interesar saber si esa división es significativa en términos del grafo. Es decir, queremos saber cuán bien se separan los nodos con diferentes clases o atributos teniendo en cuenta la estructura de la red. Este concepto se llama modularidad, y podemos calcularlo para el cluster que hicimos en el paso anterior.

Por ejemplo, nos puede interesar determinar si los dos clusters obtenidos a partir de la red de interacciones percibidas presenta buena modularidad sobre el grafo de interacciones observadas.

```r
modularity(ws.obs.red, ws.per.red.cl.lo$membership)
```

```
## [1] 0.2350083
```

La pregunta ahora es ¿Esta modularidad es significativa? Para contestar esto generamos 1000 agrupamiento al azar, calculamos su modularidad y luego comparamos contra el valor observado.

```r
# Un array de modularidades al azar
random.membership <- array()

# Valores de modularidad para 1000 agrupamientos al azar
for(i in 1:1000) random.membership[i] <- modularity( ws.obs.red, sample(1:2, 43, replace = T) )

# test del clustering basado en las modularidades
table( modularity(ws.obs.red, ws.per.red.cl.lo$membership) > random.membership )
```

```
##
## TRUE
## 1000
```

El clustering es altamente significativo.

Otra forma de comparar comunidades es utilizando el indice de Rand ajustado, de forma similar a la validación externa en los métodos de Clustering.

```r
# La funcion compare tambien tiene otras metricas como la de van Dongen.
compare(ws.obs.red.cl.lo,ws.per.red.cl.lo, method = "adjusted.rand")
```

```
## [1] 0.8182849
```

El resultado es un número entre 0 y 1, cuanto más cerca de 1 mejor. Para poder determinar si es significativo repetimos el procedimiento anterior generando re-sampleos de las etiquetas de las comunidades.

```r
# Defino una red con las propiedades de la red percibida.
tmp.comm <- ws.per.red.cl.lo

# Asigno etiquetas al azar y vuelvo a calcular
random.rand <- array()
for(i in 1:1000) {
  tmp.comm$membership <- as.numeric(sample(1:2, 43, replace = T))
  random.rand[i] <- compare( ws.obs.red.cl.lo, tmp.comm, method = "adjusted.rand" )
}

table( compare(ws.obs.red.cl.lo,ws.per.red.cl.lo, method = "adjusted.rand") > random.rand )
```

```
##
## TRUE
## 1000
```

Nuevamente el resultado es significativo.