Entrega: Tarea 2

Laura Basalo Tur, Camila Pérez Arévalo, Josep Roman Cardell

We shall consider again the undirected Facebook friendship network considered in the last handout. The links in this network are contained in the file *facebook_sample_anon.txt*. Download it on your computer and upload it to R as a dataframe. Define an undirected graph with this list of edges.

```
#Cargamos los datos del fichero .txt en un dataframe
dataframe = read.table("data/facebook_sample_anon.txt",
                       header = FALSE,
                       col.names = c("nodeA", "nodeB"),
                       sep = " ")

#Generamos el grafo no dirigido desde los datos del dataframe
undirected_graph = graph_from_data_frame(dataframe, directed=F)
```

1) It has been observed in many networks an association between "centrality" and "lethality," defined as the fatal disconnection of the network when nodes are removed. Let's study this association on this network.

a) Repeat 1000 times the procedure of removing a random 0.1% of its set of nodes, and compute the average number of connected components of the resulting networks and the average fraction of the network represented by the largest component. Use *set.seed* to make your results reproducible.

```
#Indicamos la semilla a utilizar para que no cambien los resultados:
set.seed(4321)

#Número de repeticiones
n = 1000

compute_graph_random = function(g, n, perc){

  #Inicializamos dos listas vacías donde se guardarán los datos a calcular
  avrg_connected = c()
  avrg_largest_component = c()
  #Calculamos el porcentaje de nodos a eliminar
  perc = (0.1/100)*vcount(g)

  for(i in 1:n+1){
    #Obtenemos el 0.1% de vértices aleatorios a eliminar
    v_to_delete = sample(V(g), perc)
    g1 = delete_vertices(g, v_to_delete)
    #Obtenemos información de los componentes del grafo
    components = components(g1)
```

```
    #Guardamos el número de componentes conexos
    avrg_num_components = c(avrg_connected, components$no)
    #Guardamos la fracción del componente conexo más grande
    avrg_largest_component = c(avrg_largest_component, max(components$csize)/vcount(g1))
  }

  result = list("Average number of connected components" = mean(avrg_num_components),
                "Average fraction by largest component" = mean(avrg_largest_component))
  return(result)
}

#Ejecutamos la función del cálculo con el grafo no dirigido
compute_graph_random(undirected_graph, n, perc)
```

```
## $`Average number of connected components`
## [1] 1
##
## $`Average fraction by largest component`
## [1] 0.9999527
```

Como podemos observar en los resultados y después de un cálculo de 1000 iteraciones, la media del número de componentes en el grafo es de 1, mientras que la media de la fracción de representada por el componente más grande de la red es de 0.9999527. En el apartado siguiente realizaremos un análisis de estos resultados y su comparación eliminando vértices escogidos con diferentes métricas de centralización.

b) Now, compute the number of connected components and the fraction represented by the largest component of the networks obtained after removing the most central 0.1% of nodes, for the following centrality indices (of course, if the most central 0.1% of nodes for two indices are the same set of nodes, you need not waste your time considering twice the same network): *degree*; *closeness*; *betweenness*; *page.rank*. (*Hint*: It might be convenient to define first a function that removes a given set of nodes of this graph and computes the number of connected components and the fraction represented by the largest component of the resulting network; then you will only need to apply it to the required different sets of most central nodes.) Is it what you expected?

Para calcular el número de componentes y la fracción representada por el componente más grande de la red después de eliminar el 0.1% de los vértices o nodos más centrales según las diferentes métricas, creamos una función que recoge como parámetros el grafo sobre el cuál se han de borrar los vértices y el vector que representa los vértices a eliminar según las diferentes métricas:

```
remove_nodes = function(g, vertices){
  g1 = delete_vertices(g, vertices)
  num_components = components(g1)$no
  frac_largest_component = max(components(g1)$csize)/vcount(g1)
  return(list("Number of connected components" = num_components,
              "Fraction represented by the largest component" = frac_largest_component))
}

#Porcentaje de nodos a eliminar
perc = (0.1/100)*vcount(undirected_graph)
```

La primera métrica a analizar será la de *degree*, la cual nos indica el número de grados de cada vértice del grafo, es decir, su número de enlaces.

```r
order_by_degree = order(centr_degree(undirected_graph)$res, decreasing = TRUE)
v_to_delete = order_by_degree[1:perc] #100 1535 1723 3156
remove_nodes(undirected_graph, v_to_delete)
```

```
## $`Number of connected components`
## [1] 41
##
## $`Fraction represented by the largest component`
## [1] 0.9878563
```

La siguiente métrica a analizar es la de *closeness*, que nos devuelve para cada vértice el número de pasos requeridos para acceder a todos los demás vértices del grafo.

```r
order_by_clos = order(centr_clo(undirected_graph)$res, decreasing = TRUE)
v_to_delete = order_by_clos[1:perc] #100  52 366 493
remove_nodes(undirected_graph, v_to_delete)
```

```
## $`Number of connected components`
## [1] 12
##
## $`Fraction represented by the largest component`
## [1] 0.9972739
```

La métrica *betwennes* nos indica para cada vértice el número de veces que este actúa como puente a lo largo del camino más corto entre dos nodos del grafo.

```r
order_by_betwennes = order(centr_betw(undirected_graph)$res, decreasing = TRUE)
v_to_delete = order_by_betwennes[1:perc]
#100 1535 3156 1723 -> Same nodes as with the Degree metric.
remove_nodes(undirected_graph, v_to_delete)
```

```
## $`Number of connected components`
## [1] 41
##
## $`Fraction represented by the largest component`
## [1] 0.9878563
```

Por último, hacemos el cálculo utilizando la métrica *Page Rank*, la cual proporcionar a cada vértice una puntuación según el número de aristas entrantes que este posee y de la calidad de los vértices de los cuales provienen dichos enlaces.

```r
page_rank_result = page_rank(undirected_graph, directed = FALSE, algo = "power")
order_by_pagerank = order(page_rank_result$vector, decreasing = TRUE)
v_to_delete = order_by_pagerank[1:perc] # 3156  100 1535    1
remove_nodes(undirected_graph, v_to_delete)
```

```
## $`Number of connected components`
## [1] 52
##
## $`Fraction represented by the largest component`
## [1] 0.983891
```

Es de esperar que, en el apartado A, al eliminar aleatoriamente una fracción de los vértices de un grafo el número de componentes, en media, se mantenga sobre 1. Esto se debe a que la mayoría de los nodos eliminados no tienen una importancia significativa para la conectividad del grafo.

Por otro lado y, según vemos en los resultados del apartado B, los nodos de mayor grado tienen la función de conectores dentro del grafo, por lo que si los vértices eliminados son los de mayor grado, un elevado número de conexiones se eliminarán, provocando la división del grafo en varias componentes, en este caso 41.

Lo mismo se puede aplicar para las otras métricas analizadas, *Closeness* y *Betweennes*, ambas miden la centralidad de los nodos por lo que los vértices con un valor más elevado de las anteriores propiedades será un nodo con elevada influencia sobre la conectividad de los otros nodos en el grafo. Las métricas son tan similares que los nodos con valores más elevados de betweennes y degree son los mismos.

En el caso del *PageRank* los nodos son evaluados según su grado de aristas entrantes, además de la calidad de los nodos que forman parte de estas aristas entrantes. Esta métrica se entiende como: *partiendo de nodos aleatorios y siguiendo caminos aleatorios cuáles son los nodos del grafo por los que se pasaría con más frecuencia.* De este modo, si eliminamos los nodos con mayor frecuencia de paso, resulta aún mayor la desconexión de la red ya que en este caso aparecen 52 componentes conectadas.

Respecto a la fracción del componente más grande, realizamos el siguiente cálculo para visualizar de manera más clara el número de vértices que lo formarían:

```
#Resultado eliminado vértices aleatorios
0.9999527*(vcount(undirected_graph)-4)
```

```
## [1] 4034.809
```

```
#Resultados métricas Degree y Betweennes
0.9878563*(vcount(undirected_graph)-4)
```

```
## [1] 3986
```

```
#Resultado métrica Closeness
0.9972739*(vcount(undirected_graph)-4)
```

```
## [1] 4024
```

```
#Resultado métrica PageRank
0.983891*(vcount(undirected_graph)-4)
```

```
## [1] 3970
```

El componente conexo más grande eliminando los nodos con mayor valor de PR tiene unos 50 nodos menos que el correspondiente eliminando los nodos de mayor CLC, por lo que vemos que con PageRank se han eliminado vértices que proporcionaban más conexión al grafo.

En cuanto al resultado obtenido en el apartado A (eliminando vértices aleatorios) esperábamos que este fuese el mayor que utilizando las métricas de centralidad para eliminar, ya que con estas últimas se eliminan aquellos vértices que cuentan con una mayor conectivadad. Es decir, se eliminan los vértices que haría más probable que el grafo se divida en más componentes y, como resultado, el tamaño del componente más largo se vea disminuido.

2) Now, consider the same graph as a directed one, and find the hubs and authorities scores. Compare with the page rank score.

```
#Generamos el grafo dirigido desde los datos del dataframe
directed_graph = graph_from_data_frame(dataframe, directed=T)
```

A continuación calculamos el algoritmo HITS con el grafo dirigido. Este algoritmo nos devuelve dos vectores de puntuaciones: hub y authorities.

```
#Hub vector
hub_score = hub_score(directed_graph)
head(hub_score$vector,10) # Top 10 PR
```

```
##            0            1            2            3            4            5
## 4.881624e-05 7.825648e-06 4.112081e-08 3.544931e-07 4.154800e-08 1.520143e-05
##            6            7            8            9
## 2.267623e-08 2.664711e-05 3.174368e-08 2.400849e-05
```

```
#Authorities vector
authority_score = authority_score(directed_graph)
head(authority_score$vector,10) # Top 10 PR
```

```
##            0            1            2            3            4            5
## 0.000000e+00 5.707246e-07 5.707246e-07 5.707246e-07 5.707246e-07 5.707246e-07
##            6            7            8            9
## 5.707246e-07 5.707246e-07 5.707246e-07 5.748690e-07
```

El siguiente a calcular es el algoritmo PageRank, el cuál tiene en cuenta las aristas entrantes.

```
PR = page_rank(directed_graph, directed = TRUE) #iteraciones = 1000
head(PR$vector, decreasing = TRUE,10) # Top 10 PR
```

```
##            0            1            2            3            4            5
## 7.730367e-05 7.749303e-05 7.749303e-05 7.749303e-05 7.749303e-05 7.749303e-05
##            6            7            8            9
## 7.749303e-05 7.749303e-05 7.749303e-05 8.160984e-05
```

Tanto el algoritmo de PageRank como el de HITS son utilizados para analizar los enlaces de un grafo y dar una puntuación a cada vértice de este. No obstante, entre ellos se pueden observar algunas diferencias como el alcance de nodos utilizados para el cálculo. Mientras HITS se aplica a un subgrafo o a los vecinos de un nodo, PageRank se aplica al grafo entero. De la misma forma, HITS trabaja dando puntuación a cada nodo según las aristas de entrada (*hub_score*) y los de salida (*hub_authority*), y PageRank solo a las aristas entrantes.

Como podemos ver en los resultados obtenidos y en la figura 1, centrándonos en el nodo *0*, vemos como su puntuación en el vector *authorities* es 0, ya que no tiene ninguna arista entrante. No obstante, si nos fijamos en la puntuación del vector *hubs* se observa que su valor es más elevado en comparación a sus vértices vecinos ya que hay más aristas que salen de este nodo. Por otro lado, las puntuaciones de los primeros nodos en el vector obtenido con PageRank son valores bastante similares.

En relación a tiempo de ejecución, HITS realiza los cálculos en tiempo búsqueda, pero PageRank realiza el cálculo de un nodo en el momento de indexación, lo que lo hace más eficiente en términos de complejidad.

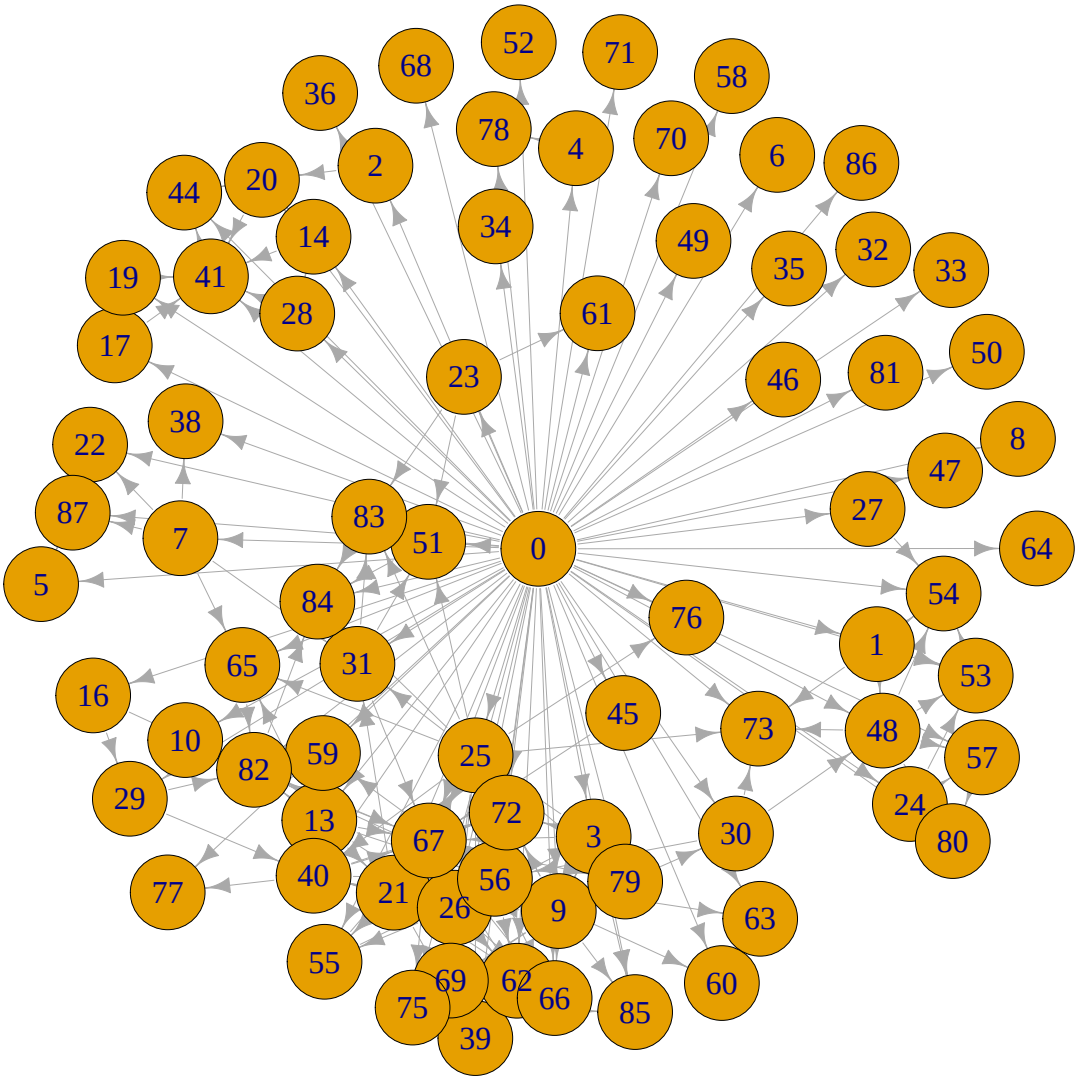

Figure 1: Representación gráfica parcial del nodo 0 como 'Hub'.