Freie Universität Berlin

Introducing the igraph library

Course: Complex Network Analysis mit GNU R

Prof. Dr. Claudia Müller-Birn

Institute for Computer Science, Networked Information Systems

February 23, 2012

This chapter is mainly based on the documentation written by Gábor Csárdi, Tamás Nepusz: The igraph software package for complex network research. InterJournal Complex Systems, 1695, 2006.

tapply

```
tapply(vector, grouping, f)
```

Input: Vector, grouping (or factor or list of factors), and a function f

Output: Matrix/array, where an element in the matrix/array is the value of *f* at a grouping *g* of the vector, and *g* gets pushed to the row/col names

Example:

```
> ages <- c(25,26,55,37,21,42)
> affils <- c("R", "D","D","R","U","D")
> tapply(ages,affils,mean)
 D  R  U
41 31 21
```

What does this example mean?

```
# the second parameter is treated as a factor
# a factor can be seen as vector with some more information
> affils <- factor(affils)
> str(affils)
 Factor w/ 3 levels "D","R","U": 2 1 1 2 3 1
```

tapply (*extended example*)

```
> d <- data.frame(list(gender=c("M", "M", "F", "M", "F", "M"),
+ age = c(47,59,21,32,33,24), income=c(55000,88000,32450,76500,123000,45650))) # create new dataframe
> d
  gender age income
1      M  47  55000
2      M  59  88000
3      F  21  32450
4      M  32  76500
5      F  33 123000
6      M  24  45650
> d$over25 <- ifelse(d$age > 25,1,0) # extend dataframe by additional column “over25” and select
                                     entries that fullfil the condition “>25” in column “age”
> head(d)
  gender age income over25
1      M  47  55000      1
2      M  59  88000      1
3      F  21  32450      0
4      M  32  76500      1
5      F  33 123000      1
6      M  24  45650      0
> tapply(d$income,list(d$gender,d$over25),mean) # compose two groups based on gender and age and
      0         1                               calculate the mean in each group
F 39050 123000.00
M    NA  73166.67
```

Overview measures of degree centrality

```
##########
# vertex degree
> degree(graph)[]

##########
# normalized vertex degree – degree centrality
> n <- vcount(graph)
> degrees <- degree(graph)/(n-1)

##########
# degree centralization
> centralization.degree <- function(graph) {

> n <- vcount(graph)
> degrees <- degree(graph)/(n-1)

> numerator <- sum(max(degrees) - degrees)
> denominator <- (n-1)*(n-2)

> return(numerator/denominator)
}
```

Overview measures of closeness centrality

```
##########
# vertex closeness
> closeness(graph, mode="out")* vcount(graph) # directed
> closeness(graph) * vcount(graph) # undirected



##########
# normalized vertex closeness - closeness centrality
> closenesseses <- closeness(graph, mode="out")  # directed
> closenesseses <- closeness(graph)  # undirected


##########
# closeness centralization
> centralization.closeness <- function(graph) {
> n <- vcount(graph)
> closenesses <- closeness(graph, mode="out")  # directed
> closenesses <- closeness(graph)  # undirected

> numerator <- sum(max(closenesses)-closenesses)
> denominator <- (n-2)*(n-1)/(2*n-3)

> return(numerator/denominator)
}
```

Overview measures of betweenness centrality

```
##########
# vertex betweenness
> betweenness(graph)   # undirected
> betweenness(graph, directed = TRUE)  # directed

##########
# normalized vertex betweenness - betweenness centrality
> n <- vcount(graph)
> betweennesses <- betweenness(graph)/(0.5*(n-1)*(n-2))  # undirected
> betweennesses <- betweenness(graph, directed = TRUE)/(0.5*(n-1)*(n-2)) # directed

##########
# betweenness centralization
> centralization.betweenness <- function(graph) {

> n <- vcount(graph)
> betweennesses <- betweenness(graph)/(0.5*(n-1)*(n-2))

> numerator <- 2*sum(max(betweennesses)-betweennesses)
> denominator <- (n-2)^2*(n-1)

> return(numerator/denominator)
}
```

Another network example:

```
> vertices <- read.csv("http://cneurocvs.rmki.kfki.hu/igraph/judicial.csv") # load vertices
> edges <- read.table("http://cneurocvs.rmki.kfki.hu/igraph/allcites.txt")  # load edges

> jg <- graph.data.frame(edges, vertices=vertices, dir=TRUE)

> summary(jg)
Vertices: 30288
Edges: 216738
Directed: TRUE
No graph attributes.
Vertex attributes: name, usid, parties,
hub, hubrank, auth, authrank, between, i
No edge attributes.

> is.connected(jg) # decides whether the graph is connected
[1] FALSE

> no.clusters(g, mode='weak') # number of weak clusters
[1] 15
> no.clusters(g, mode='strong') # number of strong clusters
[1] 15
```

graph.data.frame creates igraph graphs from one or two data frames.
If vertices is not NULL, then it must be a data frame giving vertex metadata. **The first column of vertices is assumed to contain symbolic vertex names, this will be added to the graphs as the 'name' vertex attribute.** Other columns will be added as additional vertex attributes. The symbolic edge list is checked to contain only vertex names listed in vertices.

Another network example:

```
> table(clusters(jg)$csize)
```

1	3	4	25389
4871	8	1	1

isolates giant component

```
# Now, we want to find the isolated vertices in this graph. For this, we must write our own function.
# We also want to take loops into account.

> g <- simplify(jg, remove.multiple = TRUE, remove.loops = TRUE) # remove all multiple edges and
                                                                 loops

> isolates <- function(jg){return(which(degree(jg)==0)-1)} # our complete function

> i <- isolates(jg)

> length(i)
[1] 4871
```

The “isolate” function in more detail

```
> isolates <- function(jg){return(which(degree(jg)==0)-1)} # our complete function

> degreeZeroNodes <- degree(jg)==0                          # all nodes with a degree equal zero

> head(degreeZeroNodes)  # vector with Booleans
[1]  TRUE  TRUE  TRUE  TRUE FALSE  TRUE

> isolates <- which(degreeZeroNodes))  # returns the indices of the vector that are equal to TRUE

# the long way, probably shorter is the function

> jg2 <- decompose.graph(jg, min.vertices = 2)  # decompose graph and ignore components with less
                                                  than 2 vertices

> table(clusters(jg2[[1]])$csize)
25389
    1
```

Function “subgraph” and “which.max”

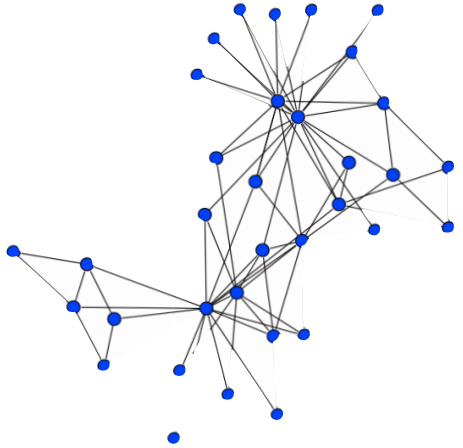
```
> cl <- clusters(jg)
> jg3 <- subgraph(jg, which(cl$membership == which.max(cl$csize)-1)-1)  # the complete function

# now in more detail
# with “cl$csize” we address the list component “csize” and apply the which.max function

> cl$csize[1:10]
 [1]     1     1     1     1 25389     1     1     1     1     1

> maxcl <- which.max(cl$csize)
> maxcl
[1] 5
```

returns an index of the maximal element of a vector
which.min() is also available

```
> cl$membership[1:14] # returns a vector that contains the nodes with their cluster ids
 [1]  0  1  2  3  4  5  6  7  8  9 10 11  4 12

> subjg <- which(cl$membership == which.max(cl$csize)-1)  # select those vertices where there
> subjg[1:14]                                             membership ids are equal to the id of the
[1]  5 13 21 22 23 24 25 28 31 33 40 42 44 46             maximal cluster

> summary(jg3)
Vertices: 25389
Edges: 216718
```

Karate club data set

```
> gk <- read.graph("http://cneurocvs.rmki.kfki.hu/igraph/karate.net", format="pajek") # load data

> summary(gk)
Vertices: 34
Edges: 78
Directed: FALSE
No graph attributes.
No vertex attributes.
No edge attributes.

> transitivity(gk, type="global") # determine transitivity
[1] 0.2476008

> localCC <- transitivity(gk, type="local", vids=NULL) #calculate the Global Clustering Coefficient

> sum <- sum(localCC, na.rm=TRUE)

> globalCC <- 1/vcount(gk)*sum

> globalCC
[1] 0.5650746
```



Karate club data set - edge betweenness

```
> bet <- edge.betweenness.community(gk)        # returns a numeric vector, the edge betweenness value
                                                 of the removed edges, the order is the same as in
                                                 removed.edges
> str(bet)
List of 4
 $ removed.edges   : num [1:78] 0 14 8 45 52 27 16 23 31 24 ...
 $ edge.betweenness: num [1:78] 68.5 62.8 83 82.2 123.5 ...
 $ merges          : num [1:33, 1:2] 32 29 25 24 23 6 5 4 3 2 ...
 $ bridges         : num [1:33] 78 77 75 74 72 69 68 66 65 64 ...


> V(gk)$name <- seq(vcount(g))-1       # apply ids as vertice names
> summary(gk)
Vertices: 34
Edges: 78
Directed: FALSE
No graph attributes.
Vertex attributes: name.
No edge attributes.

> V(gk)$name[1:10]
 [1] 0 1 2 3 4 5 6 7 8 9
```

Karate club data set - edge betweenness (*cont.*)

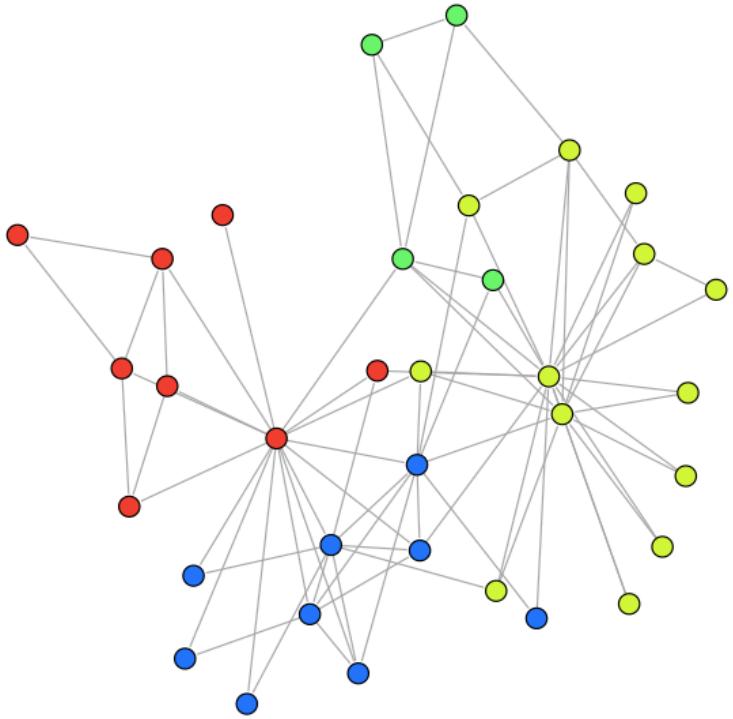
```
# the fastgreedy.modularization tries to find dense subgraph, also called communities in graphs via
directly optimizing a modularity score


> fcs <- fastgreedy.community(simplify(as.undirected(gk)))
> str(fcs)
List of 2
 $ merges    : num [1:33, 1:2] 5 6 0 4 10 29 33 23 27 25 ...
 $ modularity: num [1:34] -0.05043 -0.03812 -0.01417 -0.00169 0.01771 ...


# The merges performed by the algorithm will be stored here. Each merge is a row in a two-column
matrix and contains the ids of the merged communities. Communities are numbered from zero. In each
step a new community is created from two other communities and its id will be one larger than the
largest community id so far. This means that before the first merge we have n communities.


> fcs$merges
      [,1] [,2]
 [1,]    5   16
 [2,]    6   34
 [3,]    0   35
 [4,]    4   36
 [5,]   10   37
```

Karate club data set - edge betweenness (*cont.*)

```
# now we know which nodes are merged into clusters at each step and we know which modularity was
computed during each step.


# community.to.membership takes a merge matrix, a typical result of community structure detection
algorithms (here here fastgreedy.modularization) and creates a membership vector by performing a
given number of merges (here 30) in the merge matrix


# we will use this function to determine the max modularity in the graph


> memb <- community.to.membership(gk, fcs$merges, steps=which.max(fcs$modularity)-1)


> memb
$membership
 [1] 0 3 3 3 0 0 0 3 1 3 0 0 3 3 1 1 0 3 1 0 1 3 1 1 2 2 1 1 2 1 1 2 1 1

$csize
[1]  8 13  4  9
```

Karate club data set - visualization of the communities

```
> colbar <- rainbow(5)
> col <- colbar[memb$membership+1]
> l <- layout.kamada.kawai(g, niter=100)

> plot(g, layout=l, vertex.size=6, vertex.color=col, vertex.label=NA)
```



Homework assignment 5

Homework assignment 5: Modularity in networks

1. Write a function that determines the highest modularity in a network based on calculating the edge betweenness.

2. Use the co-authorship network provided in homework 3.

Additional: Visualize the result in R (as network or as diagram).

Deadline is next Monday Feb 27, 2012.