Creating an igraph object

Here you will learn how to create an igraph 'object' from data stored in an edgelist. The data are friendships in a group of students. You will also learn how to make a basic visualization of the network.

Each row of the friends dataframe represents an edge in the network.

Instructions

100xp

Instructions

100xp

Inspect the first few rows of the dataframe friends using the function head().

Create new object friends.mat from the dataframe friends using as.matrix().

Convert variable to an igraph object g using graph.edgelist().

Make a basic plot of the network using plot().

Take Hint (-30xp)

> # Load igraph

> library(igraph)

Attaching package: 'igraph'

The following objects are masked from 'package:stats':

decompose, spectrum

The following object is masked from 'package:base':

union

>

> # Inspect the first few rows of the dataframe 'friends'

> head(friends)

name1 name2

1 Jessie Sidney

2 Jessie Britt

3 Sidney Britt

4 Sidney Donnie

5 Karl Berry

6 Sidney Rene

>

> # Convert friends dataframe to a matrix

> friends.mat <- as.matrix(friends)

>

> # Convert friends matrix to an igraph object

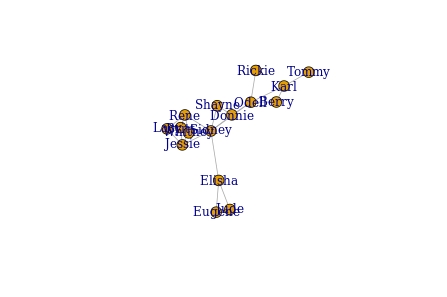
> g <- graph.edgelist(friends.mat, directed = FALSE)

>

>

> # Make a very basic plot of the network

> plot(g)



Counting vertices and edges

A lot of basic information about a network can be extracted from an igraph object. In this exercise you will learn how to count the vertices and edges from a network by applying several functions to the graph object g.

Each row of the friends dataframe represents an edge in the network.

Instructions

100xp

Use V() and E() to view the vertices and edges respectively of the network.

Use gsize() to count the number of edges in a network.

Use gorder() to count the number of vertices in a network.

Take Hint (-30xp)

> # Load igraph

> library(igraph)

>

> # Subset vertices and edges

> V(g)

+ 16/16 vertices, named:

[1] Jessie Sidney Britt Donnie Karl Berry Rene Shayne Elisha

[10] Whitney Odell Lacy Eugene Jude Rickie Tommy

> E(g)

+ 27/27 edges (vertex names):

[1] Jessie --Sidney Jessie --Britt Sidney --Britt Sidney --Donnie

[5] Karl --Berry Sidney --Rene Britt --Rene Sidney --Shayne

[9] Sidney --Elisha Sidney --Whitney Jessie --Whitney Donnie --Odell

[13] Sidney --Odell Rene --Whitney Donnie --Shayne Jessie --Lacy

[17] Rene --Lacy Elisha --Eugene Eugene --Jude Berry --Odell

[21] Odell --Rickie Karl --Odell Britt --Lacy Elisha --Jude

[25] Whitney--Lacy Britt --Whitney Karl --Tommy

>

> # Count number of edges

> gsize(g)

[1] 27

>

> # Count number of vertices

> gorder(g)

[1] 16

>

Node attributes and subsetting

In this exercise you will learn how to add attributes to vertices in the network and view them.

Instructions

100xp

Create a new vertex attribute called 'gender' from the vector genders using set\_vertex\_attr().

Create a new vertex attribute called 'age' from the vector ages using set\_vertex\_attr().

View all vertex attributes using vertex\_attr().

View the attributes of the first five vertices in a dataframe using V(g)[[]].

Take Hint (-30xp)

> library(igraph)

>

> # Inspect the objects 'genders' and 'ages'

> genders

[1] "M" "F" "F" "M" "M" "M" "F" "M" "M" "F" "M" "F" "M" "F" "M" "M"

> ages

[1] 18 19 21 20 22 18 23 21 22 20 20 22 21 18 19 20

> # Create new vertex attribute called 'gender'

> g <- set\_vertex\_attr(g, "gender", value = genders)

>

> # Create new vertex attribute called 'age'

> g <- set\_vertex\_attr(g, "age", value = ages)

>

> # View all vertex attributes in a list

> vertex\_attr(g)

$name

[1] "Jessie" "Sidney" "Britt" "Donnie" "Karl" "Berry" "Rene"

[8] "Shayne" "Elisha" "Whitney" "Odell" "Lacy" "Eugene" "Jude"

[15] "Rickie" "Tommy"

$gender

[1] "M" "F" "F" "M" "M" "M" "F" "M" "M" "F" "M" "F" "M" "F" "M" "M"

$age

[1] 18 19 21 20 22 18 23 21 22 20 20 22 21 18 19 20

>

> # View attributes of first five vertices in a dataframe

> V(g)[[1:5]]

+ 5/16 vertices, named:

name gender age

1 Jessie M 18

2 Sidney F 19

3 Britt F 21

4 Donnie M 20

5 Karl M 22

>Edge attributes and subsetting

In this exercise you will learn how to add attributes to edges in the network and view them. For instance, we will add the attribute 'hours' that represents how many hours per week each pair of friends spend with each other.

Instructions

100xp

Create a new edge attribute called 'hours' from the vector hours using set\_edge\_attr().

View all edge attributes using edge\_attr().

View all edges that include the person "Britt".

View all edges where the attribute hours is greater than or equal to 4 hours.

Take Hint (-30xp)

> library(igraph)

>

> # View hours

> hours

[1] 1 2 2 1 2 5 5 1 1 3 2 1 1 5 1 2 4 1 3 1 1 1 4 1 3 3 4

>

> # Create new edge attribute called 'hours'

> g <- set\_edge\_attr(g, "hours", value = hours)

>

> # View edge attributes of graph object

> edge\_attr(g)

$hours

[1] 1 2 2 1 2 5 5 1 1 3 2 1 1 5 1 2 4 1 3 1 1 1 4 1 3 3 4

>

> # Find all edges that include "Britt"

> E(g)[[inc('Britt')]]

+ 5/27 edges (vertex names):

tail head tid hid hours

2 Britt Jessie 3 1 2

3 Britt Sidney 3 2 2

7 Rene Britt 7 3 5

23 Lacy Britt 12 3 4

26 Whitney Britt 10 3 3

>

> # Find all pairs that spend 4 or more hours together per week

> E(g)[[hours>=4]]

+ 6/27 edges (vertex names):

tail head tid hid hours

6 Rene Sidney 7 2 5

7 Rene Britt 7 3 5

14 Whitney Rene 10 7 5

17 Lacy Rene 12 7 4

23 Lacy Britt 12 3 4

27 Tommy Karl 16 5 4

Visualizing attributes

In this exercise we will learn how to create igraph objects with attributes directly from dataframes and how to visualize attributes in plots. We will use a second network of friendship connections between students.

Instructions

100xp

Instructions

100xp

Create a new igraph object with graph\_from\_data\_frame(). Two dataframes need to be provided - friends1\_edges contains all edges in the network with attributes and friends1\_nodes contains all vertices in the network with attributes.

View all edges where the attribute hours is greater than or equal to 5 hours.

Create a new vertex attribute containing color names: orange for females and dodgerblue for males.

Plot the network with vertices colored by gender and make label names black.

Take Hint (-30xp)

> library(igraph)

>

> # Create an igraph object with attributes directly from dataframes

> g1 <- graph\_from\_data\_frame(d = friends1\_edges, vertices = friends1\_nodes, directed = FALSE)

>

>

> # Subset edges greater than or equal to 5 hours

> E(g1)[[hours >= 5]]

tail head tid hid hours

5 Valentine Kelley 6 3 5

8 Jasmine Ronald 8 4 5

12 Perry Valentine 15 6 5

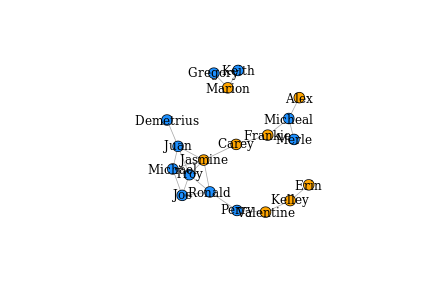
15 Juan Jasmine 9 8 6

>

> # Plot network and color vertices by gender

> V(g1)$color <- ifelse(V(g1)$gender == "F", "orange", "dodgerblue")

> plot(g1, vertex.label.color = "black")



igraph network layouts

The igraph package provides several built in layout algorithms for network visualization. Depending upon the size of a given network different layouts may be more effective in communicating the structure of the network. Ideally the best layout is the one that minimizes the number of edges that cross each other in the network. In this exercise you will explore just a few of the many default layout algorithms. Re-executing the code for each plot will lead to a slightly different version of the same layout type. Doing this a few times can help to find the best looking visualization for your network.

Instructions

100xp

Instructions

100xp

In the plot function, change the layout argument to layout\_in\_circle() to produce a circle network.

In the plot function, change the layout argument to layout\_with\_fr() to produce a network with the Fruchterman-Reingold layout.

You can also stipulate the layout by providing a matrix of (x, y) coordinates for each vertex. Here you use the layout\_as\_tree() function to generate the matrix m of coordinates. Then pass m to the layout function in plot() to plot.

Choosing a correct layout can be bewildering. Fortunately igraph has a function layout\_nicely() that tries to choose the most appropriate layout function for a given graph object. Use this function to produce the matrix m1 and plot the network using these coordinates.

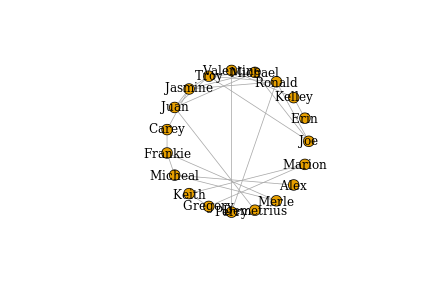
Take Hint (-30xp)

> library(igraph)

>

> # Plot the graph object g1 in a circle layout

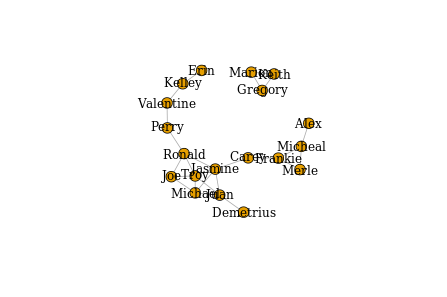
> plot(g1, vertex.label.color = "black", layout = layout\_in\_circle(g1))



>

> # Plot the graph object g1 in a Fruchterman-Reingold layout

> plot(g1, vertex.label.color = "black", layout = layout\_with\_fr(g1))

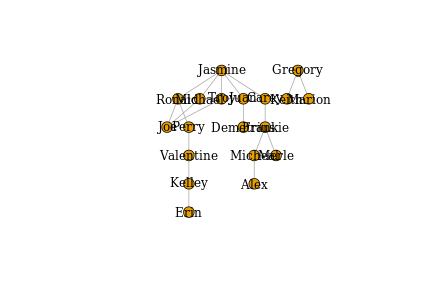


>

> # Plot the graph object g1 in a Tree layout

> m <- layout\_as\_tree(g1)

> plot(g1, vertex.label.color = "black", layout = m)

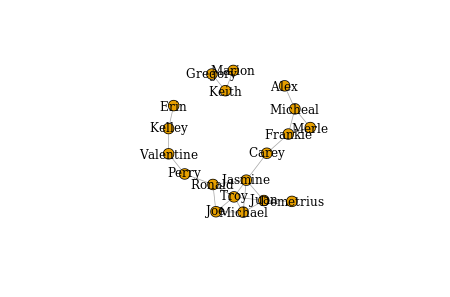


>

> # Plot the graph object g1 using igraph's chosen layout

> m1<- layout\_nicely(g1)

> plot(g1, vertex.label.color = "black", layout = m1)



Visualizing edges

In this exercise you will learn how to change the size of edges in a network based on their weight, as well as how to remove edges from a network which can sometimes be helpful in more effectively visualizing large and highly clustered networks. In this introductory chapter, we have just scratched the surface of what's possible in visualizing igraph networks. You will continue to develop these skills in future chapters.

Instructions

100xp

Instructions

100xp

Create a vector w1 of edge weights based on the number of hours friends spend together.

Plot the network ensuring that the edge.width is set to the vector of weights you just created. Using edge.color = 'black' ensures that all edges will be black.

Next, create a new graph object g2 that is the g1 network but with all edges of that are of weight less than two hours removed. This is done by using delete\_edges() which takes two arguments. The first is the graph object and the second is the subset of edges to be removed. In this case, you will remove any edges that have a value of less than two hours.

Finally, plot the new network g2 using the appropriate vector of edge widths and layout.

Take Hint (-30xp)

library(igraph)

# Create a vector of weights based on the number of hours each pair spend together

w1 <- E(g1)$hours

# Plot the network varying edges by weights

m1 <- layout\_nicely(g1)

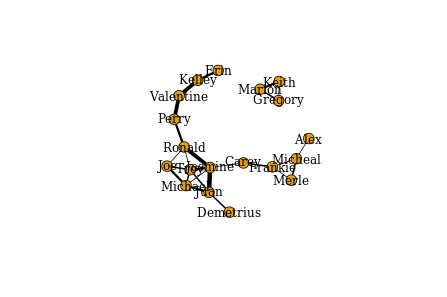
plot(g1,

vertex.label.color = "black",

edge.color = 'black',

edge.width = w1,

layout = m1)



# Create a new igraph object only including edges from the original graph that are greater than 2 hours long

g2 <- delete\_edges(g1, E(g1)[hours < 2])

# Plot the new graph

w2 <- E(g2)$hours

m2 <- layout\_nicely(g2)

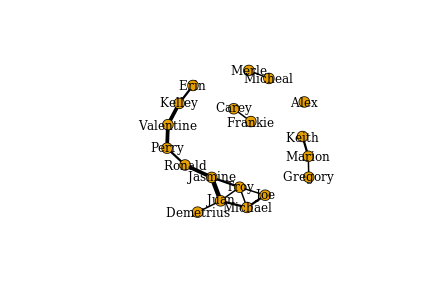
plot(g2,

vertex.label.color = "black",

edge.color = 'black',

edge.width = w2,

layout = m2)



Directed igraph objects

In this exercise you will learn how to create a directed graph from a dataframe, how to inspect whether a graph object is directed and/or weighted and how to extract those vertices at the beginnning and end of directed edges.

Instructions

100xp

Instructions

100xp

Convert the dataframe measles into an igraph graph object using the function graph\_from\_data\_frame() and ensure that it will be a directed graph by setting the second argument to TRUE.

Check if the graph object is directed by using is.directed().

Examine whether the edges of the graph object are already weighted by using is.weighted().

Subset each vertex from which each edge originates by using head\_of(). This function takes two arguments, the first being the graph object and the second the edges to examine. For all edges you can use E(g).

Take Hint (-30xp)

> library(igraph)

>

> # Get the graph object

> g <- graph\_from\_data\_frame(measles, directed = TRUE)

>

> # is the graph directed?

> is.directed(g)

[1] TRUE

>

> # Is the graph weighted?

> is.weighted(g)

[1] FALSE

>

> # Where does each edge originate from?

> table(head\_of(g, E(g)))

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26

30 2 7 3 4 4 8 2 2 4 3 3 7 4 1 6 1 3 3 1 4 2 3 2 1 2

27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52

1 3 2 4 1 1 3 1 1 7 2 3 2 3 2 3 1 2 1 1 1 3 2 6 1 2

53 54 55 56 57 58 59 60 61 62

1. 2 2 1 1 1 1 1 1 2

Identifying edges for each vertex

In this exercise you will learn how to identify particular edges. You will learn how to determine if an edge exists between two vertices as well as finding all vertices connected in either direction to a given vertex.

Instructions

100xp

Instructions

100xp

First make a visualization of this network using plot(). You will improve this visualization later. It can be useful to visualize the network before analysis. To improve visibility of the plot of this network, you should make the vertex size equal to 0 and the edge arrow size equal to 0.1.

Check if there is an edge going in each direction between vertex 184 to vertex 178 using single brackets subsetting of the graph object. If a 1 is returned that indicates TRUE there is an edge. If a 0 is returned that indicates FALSE there is not an edge.

Using the function incident() identify those edges that go in either direction from vertex 184 or only those going out from vertex 184. The first argument should be the graph object, the second should be the vertex to examine and the third argument the mode indicating the direction. Choose from all, in and out.

Take Hint (-30xp)

> library(igraph)

>

> # Make a basic plot

> plot(g,

vertex.label.color = "black",

edge.color = 'gray77',

vertex.size = 0,

edge.arrow.size = 0.1,

layout = layout\_nicely(g))

>

> # Is there an edge going from vertex 184 to vertex 178?

> g['184', '178']

[1] 1

>

> # Is there an edge going from vertex 178 to vertex 184?

> g['178', '184']

[1] 0

>

> # Show all edges going to or from vertex 184

> incident(g, '184', mode = c("all"))

+ 6/184 edges (vertex names):

[1] 184->45 184->182 184->181 184->178 184->183 184->177

>

> # Show all edges going out from vertex 184

> incident(g, '184', mode = c("out"))

+ 6/184 edges (vertex names):

[1] 184->45 184->182 184->181 184->178 184->183 184->177

>

>

>

Neighbors

Often in network analysis it is important to explore the patterning of connections that exist between vertices. One way is to identify neighboring vertices of each vertex. You can then determine which neighboring vertices are shared even by unconnected vertices indicating how two vertices may have an indirect relationship through others. In this exercise you will learn how to identify neighbors and shared neighbors between pairs of vertices.

Instructions

100xp

Instructions

100xp

Using the function neighbors() identify the vertices that are connected in any manner to vertex 12, those vertices that direct an edge to vertex 12 and those vertices that receive a directed edge from vertex 12. This can be achieved by choosing the correct value in the argument mode. Choose from all, in and out.

Determine if vertices 42 and 124 have a neighbor in common. Create a vector n1 of those vertices that receive an edge from vertex 42 and a vector n2 of those vertices that direct an edge to vertex 124 using neighbors(). Next use intersection() to identify if there are any vertices that exist in both n1 and n2.

Take Hint (-30xp)

> library(igraph)

>

> # Identify all neighbors of vertex 12 regardless of direction

> neighbors(g, '12', mode = c('all'))

+ 5/187 vertices, named:

[1] 45 13 72 89 109

>

> # Identify other vertices that direct edges towards vertex 12

> neighbors(g, '12', mode = c('in'))

+ 1/187 vertex, named:

[1] 45

>

> # Identify any vertices that receive an edge from vertex 42 and direct an edge to vertex 124

> n1 <- neighbors(g, '42', mode = c('out'))

> n2 <- neighbors(g, '124', mode = c('in'))

> intersection(n1, n2)

+ 1/187 vertex, named:

[1] 7

Distances between vertices

The inter-connectivity of a network can be assessed by examining the number and length of paths between vertices. A path is simply the chain of connections between vertices. The number of intervening edges between two vertices represents the geodesic distance between vertices. Vertices that are connected to each other have a geodesic distance of 1. Those that share a neighbor in common but are not connected to each other have a geodesic distance of 2 and so on. In directed networks, the direction of edges can be taken into account. If two vertices cannot be reached via following directed edges they are given a geodesic distance of infinity. In this exercise you will learn how to find the longest paths between vertices in a network and how to discern those vertices that are within n

connections of a given vertex. For disease transmission networks such as the measles dataset this helps you to identify how quickly the disease spreads through the network.

Instructions

100xp

Instructions

100xp

Find the length of the longest path in the network using farthest\_vertices().

Identify the sequence of the path using get\_diameter(). This demonstrates the individual children that passed the disease the furthest through the network.

Use ego() to find all vertices that are reachable within 2 connections of vertex 42 and then those that can reach vertex 42 within two connections. The first argument of ego() is the graph object, the second argument is the maximum number of connections between the vertices, the third argument is the vertex of interest, and the fourth argument determines if you are considering connections going out or into the vertex of interest.

Take Hint (-30xp)

> library(igraph)

>

> # Which two vertices are the furthest apart in the graph ?

> farthest\_vertices(g)

$vertices

+ 2/187 vertices, named:

[1] 184 162

$distance

[1] 5

>

> # Shows the path sequence between two furthest apart vertices.

> get\_diameter(g)

+ 6/187 vertices, named:

[1] 184 178 42 7 123 162

>

> # Identify vertices that are reachable within two connections from vertex 42

> ego(g, 2, '42', mode = c('out'))

[[1]]

+ 13/187 vertices, named:

[1] 42 7 106 43 123 101 120 124 125 128 129 108 127

>

> # Identify vertices that can reach vertex 42 within two connections

> ego(g, 2, '42', mode = c('in'))

[[1]]

+ 3/187 vertices, named:

[1] 42 178 184

>

Identifying key vertices

Perhaps the most straightforward measure of vertex importance is the degree of a vertex. The out-degree of a vertex is the number of other individuals to which a vertex has an outgoing edge directed to. The in-degree is the number of edges received from other individuals. In the measles network, individuals that infect many other individuals will have a high out-degree. In this exercise you will identify whether indviduals infect equivalent amount of other children or if there are key children who have high out-degrees and infect many other children.

Instructions

100xp

Instructions

100xp

Calculate the out-degree of each vertex using the function degree(). The first argument is the network graph object and the second argument is the mode which should be one of out, in or all. Assign the output of this function to the object g.outd.

View a summary of the out-degrees of all individuals using the function table() on the vector object g.outd.

Make a histogram of the out-degrees using the function hist() on the vector object g.outd.

Determine which vertex has the highest out-degree in the network using the function which.max() on the vector object g.outd.

Take Hint (-30xp)

> library(igraph)

>

> # Calculate the out-degree of each vertex

> g.outd <- degree(g, mode = c("out"))

>

> # View a summary of out-degree

> table(g.outd)

g.outd

0 1 2 3 4 6 7 8 30

125 21 16 12 6 2 3 1 1

>

> # Make a histogram of out-degrees

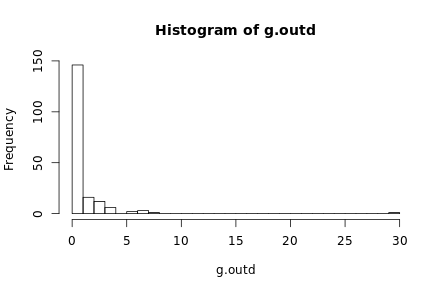
> hist(g.outd, breaks = 30)

>

> # Find the vertex that has the maximum out-degree

> which.max(g.outd)

45



Betweenness

Another measure of the importance of a given vertex is its betweenness. This is an index of how frequently the vertex lies on shortest paths between any two vertices in the network. It can be thought of as how critical the vertex is to the flow of information through a network. Individuals with high betweenness are key bridges between different parts of a network. In our measles transmission network, vertices with high betweenness are those children who were central to passing on the disease to other parts of the network. In this exercise, you will identify the betweenness score for each vertex and then make a new plot of the network adjusting the vertex size by its betweenness score to highlight these key vertices.

Instructions

100xp

Calculate the betweenness of each vertex using the function betweenness() on the graph object g. Ensure that the scores are calculated for a directed network. The results of this function will be assigned as g.b.

Visually examine the distribution of betweenness scores using the function hist().

Use plot() to make a plot of the network based on betweenness scores. The vertex labels should be made NA so that they do not appear. The vertex size attribute should be one plus the square-root of the betweenness scores that are in object g.b. Given the huge disparity in betweenness scores in this network, normalizing the scores in this manner ensures that all nodes can be viewed but their relative importance is still identifiable.

Take Hint (-30xp)

> library(igraph)

>

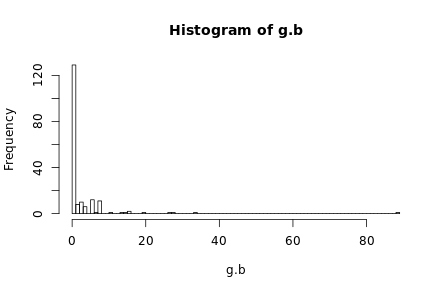
> # Calculate betweenness of each vertex

> g.b <- betweenness(g, directed = TRUE)

>

> # Show histogram of vertex betweenness

> hist(g.b, breaks = 80)



>

> # Create plot with vertex size determined by betweenness score

> plot(g,

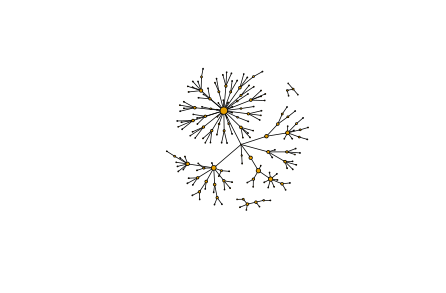
vertex.label = NA,

edge.color = 'black',

vertex.size = sqrt(g.b)+1,

edge.arrow.size = 0.05,

layout = layout\_nicely(g))



>Visualizing important nodes and edges

One issue with the measles dataset is that there are three individuals for whom no information is known about who infected them. One of these individuals (vertex 184) appears ultimately responsible for spreading the disease to many other individuals even though they did not directly infect too many indviduals. However, because vertex 184 has no incoming edge in the network they appear to have low betweenness. One way it is possible to explore the importance of this vertex is by visualizing the geodesic distances of connections going out from this individual. In this exercise you shall create a plot of these distances from this patient zero.

Instructions

100xp

Instructions

100xp

Use make\_ego\_graph() to create a subset of our network comprised of vertices that are connected to vertex 184. The first argument is the original graph g. The second argument is the maximal number of connections that any vertex needs to be connected to our vertex of interest. In this case we can use diameter() to return the length of the longest path in the network. The third argument is our vertex of interest which should be 184. The final argument is the mode. In this instance you can include all connections regardless of direction.

Create an object dists that contains the geodesic distance of every vertex from vertex 184. Use the function distances() to calculate this.

Assign the attribute color to each vertex. Each color will be selected based on its geodesic distance. The color palette colors is a length equal to the maximal geodesic distance plus one. This is so that vertices of the same distance are plotted in the same color and patient zero also has its own color.

Use plot() to visualize the network g184. The vertex label should be the geodesic distances dists.

Take Hint (-30xp)

>

> # Make an ego graph

> g184 <- make\_ego\_graph(g, diameter(g), nodes = '184', mode = c("all"))[[1]]

>

> # Get a vector of geodesic distances of all vertices from vertex 184

> dists <- distances(g184, "184")

>

> # Create a color palette of length equal to the maximal geodesic distance plus one.

> colors <- c("black", "red", "orange", "blue", "dodgerblue", "cyan")

>

> # Set color attribute to vertices of network g184.

> V(g184)$color <- colors[dists+1]

>

> # Visualize the network based on geodesic distance from vertex 184 (patient zero).

> plot(g184,

vertex.label = dists,

vertex.label.color = "white",

vertex.label.cex = .6,

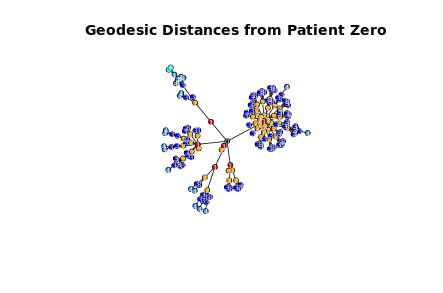
edge.color = 'black',

vertex.size = 7,

edge.arrow.size = .05,

main = "Geodesic Distances from Patient Zero"

)



Forrest Gump network

In this chapter you will use a social network based on the movie Forrest Gump. Each edge of the network indicates that those two characters were in at least one scene of the movie together. Therefore this network is undirected. To familiarize yourself with the network, you will first create the network object from the raw dataset. Then, you will identify key vertices using a measure called eigenvector centrality. Individuals with high eigenvector centrality are those that are highly connected to other highly connected individuals. You will then make an exploratory visualization of the network.

Instructions

100xp

Instructions

100xp

Inspect the first few rows of the dataframe gump using head().

Make an undirected network using graph\_from\_data\_frame().

Identify the key vertices using the function eigen\_centrality() and assign the results of this to the object g.ec. Next identify which individual has the highest eigenvector centrality using which.max(). The values of the centrality scores are stored in g.ec$vector.

Make a plot of the Forrest Gump Network using plot(). Make the size of the vertices equal to 25 times the eigenvector centrality values that are stored in g.ec$vector.

> head(gump)

V1 V2

1 ABBIE HOFFMAN JENNY

2 ABBIE HOFFMAN POLICEMAN

3 ANCHORMAN FORREST

4 ANCHORMAN LT DAN

5 ANCHORMAN MARGO

6 ANCHORMAN MRS GUMP

>

> # Make an undirected network

> g <- graph\_from\_data\_frame(gump, directed = FALSE)

>

> # Identify key nodes using eigenvector centrality

> g.ec <- eigen\_centrality(g)

> which.max(g.ec$vector)

FORREST

36

>

> # Plot Forrest Gump Network

> plot(g,

vertex.label.color = "black",

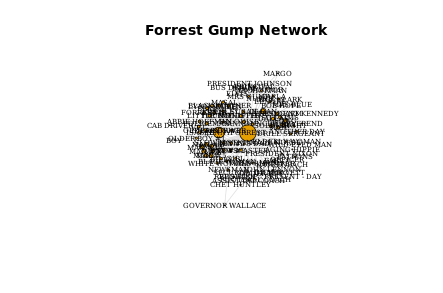
vertex.label.cex = 0.6,

vertex.size = 25\*(g.ec$vector),

edge.color = 'gray88',

main = "Forrest Gump Network"

)



Network density and average path length

The first graph level metric you will explore is the density of a graph. This is essentially the proportion of all potential edges between vertices that actually exist in the network graph. It is an indicator of how well connected the vertices of the graph are.

Another measure of how interconnected a network is average path length. This is calculated by determining the mean of the lengths of the shortest paths between all pairs of vertices in the network. The longest path length between any pair of vertices is called the diameter of the network graph. You will calculate the diameter and average path length of the original graph g.

Instructions

100xp

Instructions

100xp

Using the function edge\_density() calculate the density of the graph gand assign this value to the vector gd.

Use diameter() to calculate the diameter of the original graph g.

Assign the average path length of g to g.apl with the function mean\_distance().

Take Hint (-30xp)

> library(igraph)

>

> # Get density of a graph

> gd <- edge\_density(g)

>

> # Get the diameter of the graph g

> diameter(g, directed = FALSE)

[1] 4

>

> # Get the average path length of the graph g

> g.apl <- mean\_distance(g, directed = FALSE)

> g.apl

[1] 1.994967

Random graphs

Generating random graphs is an important method for investigating how likely or unlikely other network metrics are likely to occur given certain properties of the original graph. The simplest random graph is one that has the same number of vertices as your original graph and approximately the same density as the original graph. Here you will create one random graph that is based on the original Forrest Gump Network.

Instructions

100xp

Instructions

100xp

Generate a random graph using the function erdos.renyi.game(). The first argument n should be the number of nodes of the graph g which can be calculated using gorder(), the second argument p.or.m should be the density of the graph g which you previously stored as the obejct gd. The final argument is set as type='gnp' to tell the function that you are using the density of the graph to generate a random graph. Store this new graph as the vector g.random.

Get the density of the random graph g.random. You will notice if you generate a random graph a few times that this value will slightly vary but be approximately equal to the density of your original graph g from the previous exercise stored in the object gd.

Calculate the average path length of the random graph g.random.

Take Hint (-30xp)

> library(igraph)

>

> # Create one random graph with the same number of nodes and edges as g

> g.random <- erdos.renyi.game(n = gorder(g), p.or.m = gd, type = "gnp")

>

> g.random

IGRAPH U--- 94 126 -- Erdos renyi (gnp) graph

+ attr: name (g/c), type (g/c), loops (g/l), p (g/n)

+ edges:

[1] 6--11 8--16 4--20 16--21 2--22 21--22 15--25 14--27 15--27 19--27

[11] 26--28 19--30 28--30 8--31 24--32 27--32 25--33 8--34 12--34 31--34

[21] 33--34 14--35 25--38 34--39 40--42 15--43 24--43 32--44 34--44 32--45

[31] 43--45 14--46 20--47 9--50 11--50 31--50 11--51 34--52 29--53 9--54

[41] 46--54 51--54 16--55 52--56 1--57 27--58 4--59 24--59 40--59 9--60

[51] 12--60 46--60 50--60 55--60 49--61 19--62 7--63 38--63 12--64 27--64

[61] 41--65 56--65 38--68 46--69 48--69 67--71 3--72 4--72 67--72 14--73

[71] 28--73 53--73 22--74 12--75 39--75 4--76 21--76 70--76 1--77 22--77

+ ... omitted several edges

>

> plot(g.random)

>

> # Get density of new random graph `g.random`

> edge\_density(g.random)

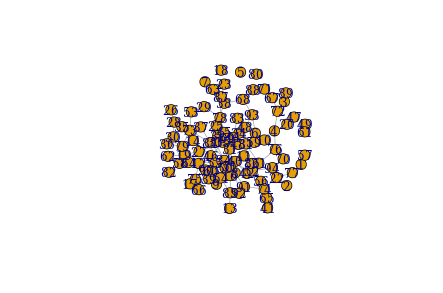
[1] 0.02882636

>

> #Get the average path length of the random graph g.random

> mean\_distance(g.random, directed = FALSE)

[1] 4.607608



Network randomizations

In the previous exercise you may have noticed that the average path length of the Forrest Gump network was smaller than the average path length of the random network. If you ran the code a few times you will have noticed that it is nearly always lower in the Forrest Gump network than the random network. What this suggests is that the Forrest Gump network is more highly interconnected than each random network even though the random networks have the same number of vertices and approximately identical graph densities. Rather than re-running this code many times, you can more formally address this by creating 1000 random graphs based on the number of vertices and density of the original Forrest Gump graph. Then, you can see how many times the average path length of the random graphs is less than the original Forrest Gump network. This is called a randomization test.

Instructions

100xp

Instructions

100xp

Generate 1000 random graphs of the original graph g by executing the code that creates the list object gl and the for loop.

Calculate the average path length of the 1000 random graphs using lapply(). Create a vector gl.apls of these 1000 values by executing the code that uses unlist().

Plot a histogram of the average path lengths of the 1000 random graphs using hist() on the vector gl.apls. Add a red dashed line to the plot using abline() with the x-intercept being the value of the average path length of the original graph g.apl. You calculated this value in the previous exercise.

Calculate the proportion of times that the values of the average path length of random graphs gl.apls are lower than the value of the original graph g.apl. This is essentially the probability that we would expect our observed average path length by chance given the original density and number of vertices of the original graph.

Take Hint (-30xp)

> library(igraph)

>

> # Generate 1000 random graphs

> gl <- vector('list',1000)

>

> for(i in 1:1000){

gl[[i]] <- erdos.renyi.game(n = gorder(g), p.or.m = gd, type = "gnp")

}

>

> # Calculate average path length of 1000 random graphs

> gl.apl <- lapply(gl, mean\_distance, directed = FALSE)

> gl.apls <- unlist(gl.apl)

>

> # Plot the distribution of average path lengths

> hist(gl.apls, xlim = range(c(1.5, 6)))

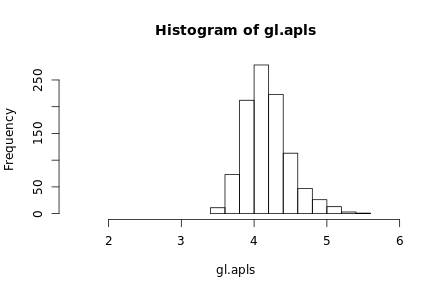
> abline(v = g.apl, col = "red", lty = 3, lwd=2)

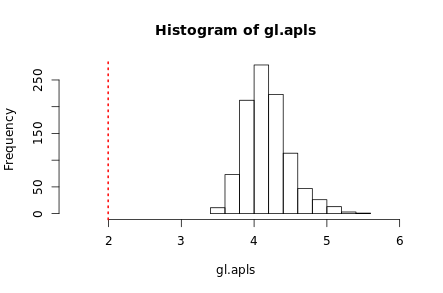
>

> # Calculate the proportion of graphs with an average path length lower than our observed

> sum(gl.apls < g.apl)/1000

[1] 0





Great work! As you can see, the Forrest Gump network is far more interconnected than we would expect by chance as zero random networks have an average path length smaller than the Forrest Gump network's average path length.

Triangles and transitivity

Another important measure of local connectivity in a network graph involves investigating triangles (also known as triads). In this exercise you will find all closed triangles that exist in a network. This means that an edge exists between three given vertices. You can then calculate the transitivity of the network. This is equivalent to the proportion of all possible triangles in the network that are closed. You will also learn how to identify the number of closed triangles that any given vertex is a part of and its local transitivity - that is, the proportion of closed triangles that the vertex is a part of given the theoretical number of triangles it could be a part of.

Instructions

100xp

Instructions

100xp

Show a matrix of all possible triangles in the Forrest Gump network g using the function triangles().

Using the function count\_triangles(), find how many triangles that the vertex "BUBBA" is a part of. The vids argument refers to the id of the vertex.

Calculate the global transitivity of the network g using transitivity().

Find the local transitivity of vertex "BUBBA" also using the function transitivity(). The type is defined as local to indicate that you are calculating a local rather than global transitivity.

Take Hint (-30xp)

> library(igraph)

>

> # Show all triangles in the network.

> matrix(triangles(g), nrow = 3)

[,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,14]

[1,] 36 36 36 36 36 36 36 36 36 36 36 36 36 36

[2,] 1 1 1 1 2 4 4 6 6 6 6 7 7 8

[3,] 83 38 39 66 68 57 24 27 75 40 45 8 69 69

[,15] [,16] [,17] [,18] [,19] [,20] [,21] [,22] [,23] [,24] [,25] [,26]

[1,] 36 36 36 36 36 36 36 36 36 36 36 36

[2,] 11 11 11 12 12 13 14 14 14 14 14 14

[3,] 12 13 70 70 13 70 4 19 24 71 65 57

[,27] [,28] [,29] [,30] [,31] [,32] [,33] [,34] [,35] [,36] [,37] [,38]

[1,] 36 36 36 36 36 36 36 36 36 36 36 36

[2,] 14 14 14 15 15 17 17 18 18 19 19 21

[3,] 62 63 64 21 72 22 42 5 28 71 63 72

[,39] [,40] [,41] [,42] [,43] [,44] [,45] [,46] [,47] [,48] [,49] [,50]

[1,] 36 36 36 36 36 36 36 36 36 36 36 36

[2,] 22 24 26 26 26 26 26 26 27 27 27 28

[3,] 42 57 73 52 47 48 49 50 75 45 40 5

[,51] [,52] [,53] [,54] [,55] [,56] [,57] [,58] [,59] [,60] [,61] [,62]

[1,] 36 36 36 36 36 36 36 36 36 36 36 36

[2,] 28 30 30 30 34 38 38 38 39 39 40 40

[3,] 90 84 61 51 88 83 66 39 83 66 75 45

[,63] [,64] [,65] [,66] [,67] [,68] [,69] [,70] [,71] [,72] [,73] [,74]

[1,] 36 36 36 36 36 36 36 36 36 36 36 36

[2,] 41 41 41 41 41 41 41 41 41 41 41 41

[3,] 1 3 6 7 8 11 12 13 26 27 30 32

[,75] [,76] [,77] [,78] [,79] [,80] [,81] [,82] [,83] [,84] [,85] [,86]

[1,] 36 36 36 36 36 36 36 36 36 36 36 36

[2,] 41 41 41 41 41 41 41 41 41 41 41 41

[3,] 33 86 37 38 39 40 43 44 45 47 48 49

[,87] [,88] [,89] [,90] [,91] [,92] [,93] [,94] [,95] [,96] [,97] [,98]

[1,] 36 36 36 36 36 36 36 36 36 36 36 36

[2,] 41 41 41 41 41 41 41 41 41 41 41 41

[3,] 50 51 52 53 54 56 58 61 66 69 70 73

[,99] [,100] [,101] [,102] [,103] [,104] [,105] [,106] [,107] [,108]

[1,] 36 36 36 36 36 36 36 36 36 36

[2,] 41 41 41 41 41 41 43 43 43 44

[3,] 74 75 79 82 83 84 82 54 53 2

[,109] [,110] [,111] [,112] [,113] [,114] [,115] [,116] [,117] [,118]

[1,] 36 36 36 36 36 36 36 36 36 36

[2,] 44 44 44 44 44 44 44 44 44 44

[3,] 3 9 14 17 19 22 82 71 42 43

[,119] [,120] [,121] [,122] [,123] [,124] [,125] [,126] [,127] [,128]

[1,] 36 36 36 36 36 36 36 36 36 36

[2,] 44 44 44 44 44 45 47 47 47 47

[3,] 53 62 63 64 65 75 73 52 50 48

[,129] [,130] [,131] [,132] [,133] [,134] [,135] [,136] [,137] [,138]

[1,] 36 36 36 36 36 36 36 36 36 36

[2,] 47 48 48 48 48 49 49 49 50 50

[3,] 49 73 52 50 49 73 52 50 73 52

[,139] [,140] [,141] [,142] [,143] [,144] [,145] [,146] [,147] [,148]

[1,] 36 36 36 36 36 36 36 36 36 36

[2,] 51 51 52 53 54 54 56 58 59 60

[3,] 84 61 73 82 87 56 89 79 92 2

[,149] [,150] [,151] [,152] [,153] [,154] [,155] [,156] [,157] [,158]

[1,] 36 36 36 36 36 36 36 36 36 36

[2,] 60 60 60 60 60 60 61 62 62 62

[3,] 20 23 25 31 81 43 84 71 19 35

[,159] [,160] [,161] [,162] [,163] [,164] [,165] [,166] [,167] [,168]

[1,] 36 36 36 36 36 36 36 36 36 36

[2,] 62 63 64 64 64 64 64 64 65 65

[3,] 63 71 3 71 19 63 62 46 4 71

[,169] [,170] [,171] [,172] [,173] [,174] [,175] [,176] [,177] [,178]

[1,] 36 36 36 36 36 36 36 41 41 41

[2,] 65 65 65 65 65 65 66 1 1 1

[3,] 19 24 64 63 57 62 83 83 38 39

[,179] [,180] [,181] [,182] [,183] [,184] [,185] [,186] [,187] [,188]

[1,] 41 41 41 41 41 41 41 41 41 41

[2,] 1 6 6 6 6 7 7 8 11 11

[3,] 66 27 75 40 45 8 69 69 12 13

[,189] [,190] [,191] [,192] [,193] [,194] [,195] [,196] [,197] [,198]

[1,] 41 41 41 41 41 41 41 41 41 41

[2,] 11 12 12 13 26 26 26 26 26 26

[3,] 70 70 13 70 73 52 47 48 49 50

[,199] [,200] [,201] [,202] [,203] [,204] [,205] [,206] [,207] [,208]

[1,] 41 41 41 41 41 41 41 41 41 41

[2,] 27 27 27 30 30 30 38 38 38 39

[3,] 75 45 40 84 61 51 83 66 39 83

[,209] [,210] [,211] [,212] [,213] [,214] [,215] [,216] [,217] [,218]

[1,] 41 41 41 41 41 41 41 41 41 41

[2,] 39 40 40 43 43 43 44 44 44 44

[3,] 66 75 45 82 54 53 3 82 43 53

[,219] [,220] [,221] [,222] [,223] [,224] [,225] [,226] [,227] [,228]

[1,] 41 41 41 41 41 41 41 41 41 41

[2,] 45 47 47 47 47 47 48 48 48 48

[3,] 75 73 52 50 48 49 73 52 50 49

[,229] [,230] [,231] [,232] [,233] [,234] [,235] [,236] [,237] [,238]

[1,] 41 41 41 41 41 41 41 41 41 41

[2,] 49 49 49 50 50 51 51 52 53 54

[3,] 73 52 50 73 52 84 61 73 82 56

[,239] [,240] [,241] [,242] [,243] [,244] [,245] [,246] [,247] [,248]

[1,] 41 41 41 41 44 44 44 44 44 44

[2,] 58 58 61 66 2 14 14 14 14 14

[3,] 10 79 84 83 67 19 71 65 62 63

[,249] [,250] [,251] [,252] [,253] [,254] [,255] [,256] [,257] [,258]

[1,] 44 44 44 44 44 44 44 44 44 44

[2,] 14 17 17 19 19 22 43 43 53 62

[3,] 64 22 42 71 63 42 82 53 82 71

[,259] [,260] [,261] [,262] [,263] [,264] [,265] [,266] [,267] [,268]

[1,] 44 44 44 44 44 44 44 44 44 44

[2,] 62 62 63 64 64 64 64 64 65 65

[3,] 19 63 71 3 71 19 63 62 71 19

[,269] [,270] [,271] [,272] [,273] [,274] [,275] [,276] [,277] [,278]

[1,] 44 44 44 14 14 14 14 14 14 14

[2,] 65 65 65 4 4 19 19 24 65 65

[3,] 64 63 62 57 24 71 63 57 4 71

[,279] [,280] [,281] [,282] [,283] [,284] [,285] [,286] [,287] [,288]

[1,] 14 14 14 14 14 14 14 14 14 14

[2,] 65 65 65 65 65 65 62 62 62 63

[3,] 19 24 64 63 57 62 71 19 63 71

[,289] [,290] [,291] [,292] [,293] [,294] [,295] [,296] [,297] [,298]

[1,] 14 14 14 14 65 65 65 65 65 65

[2,] 64 64 64 64 4 4 19 19 24 64

[3,] 71 19 63 62 57 24 71 63 57 71

[,299] [,300] [,301] [,302] [,303] [,304] [,305] [,306] [,307] [,308]

[1,] 65 65 65 65 65 65 65 64 64 64

[2,] 64 64 64 63 62 62 62 19 19 63

[3,] 19 63 62 71 71 19 63 71 63 71

[,309] [,310] [,311] [,312] [,313] [,314] [,315] [,316] [,317] [,318]

[1,] 64 64 64 62 62 62 19 26 26 26

[2,] 62 62 62 19 19 63 63 52 47 47

[3,] 71 19 63 71 63 71 71 73 73 52

[,319] [,320] [,321] [,322] [,323] [,324] [,325] [,326] [,327] [,328]

[1,] 26 26 26 26 26 26 26 26 26 26

[2,] 47 47 47 48 48 48 48 49 49 49

[3,] 50 48 49 73 52 50 49 73 52 50

[,329] [,330] [,331] [,332] [,333] [,334] [,335] [,336] [,337] [,338]

[1,] 26 26 47 47 47 47 47 47 47 47

[2,] 50 50 52 50 50 48 48 48 48 49

[3,] 73 52 73 73 52 73 52 50 49 73

[,339] [,340] [,341] [,342] [,343] [,344] [,345] [,346] [,347] [,348]

[1,] 47 47 48 48 48 48 48 48 49 49

[2,] 49 49 52 50 50 49 49 49 52 50

[3,] 52 50 73 73 52 73 52 50 73 73

[,349] [,350] [,351] [,352] [,353] [,354] [,355] [,356] [,357] [,358]

[1,] 49 50 43 1 1 1 1 1 1 6

[2,] 50 52 53 38 38 38 39 39 66 27

[3,] 52 73 82 83 66 39 83 66 83 75

[,359] [,360] [,361] [,362] [,363] [,364] [,365] [,366] [,367] [,368]

[1,] 6 6 6 6 6 27 27 27 38 38

[2,] 27 27 40 40 45 45 40 40 66 39

[3,] 45 40 75 45 75 75 75 45 83 83

[,369] [,370] [,371] [,372] [,373] [,374] [,375] [,376] [,377] [,378]

[1,] 38 39 40 4 11 11 11 12 30 30

[2,] 39 66 45 24 12 12 13 13 61 51

[3,] 66 83 75 57 70 13 70 70 84 84

[,379] [,380] [,381] [,382] [,383] [,384]

[1,] 30 51 7 17 18 15

[2,] 51 61 8 22 28 21

[3,] 61 84 69 42 5 72

>

> # Count the number of triangles that vertex "BUBBA" is in.

> count\_triangles(g, vids='BUBBA')

[1] 37

>

> # Calculate the global transitivity of the network.

> g.tr <- transitivity(g)

> g.tr

[1] 0.1918082

>

> # Calculate the local transitivity for vertex BUBBA.

> transitivity(g, vids='BUBBA', type = "local")

[1] 0.6727273

Transitivity randomizations

As you did for the average path length, let's investigate if the global transitivity of the Forrest Gump network is significantly higher than we would expect by chance for random networks of the same size and density. You can compare Forrest Gump's global transitivity to 1000 other random networks.

Instructions

100xp

Instructions

100xp

One thousand random networks are stored in the list object gl. Using lapply() and transitivity() calculate the global transitivity of each of these networks. Assign these results to gl.tr.

Using unlist() convert gl.tr to a numeric vector gl.trs.

Investigate the summary statistics of the transitivities of the random networks using summary().

Calculate the proportion of random graphs that have a transitivity higher than the transitivity of Forrest Gump's network, which you previously calculated and assigned to g.tr.

Take Hint (-30xp)

> library(igraph)

>

> # Calculate average transitivity of 1000 random graphs

> gl.tr <- lapply(gl, transitivity)

> gl.trs <- unlist(gl.tr)

>

> # Get summary statistics of transitivity scores

> summary(gl.trs)

Min. 1st Qu. Median Mean 3rd Qu. Max.

0.00000 0.01983 0.02985 0.03094 0.04119 0.08876

>

> # Calculate the proportion of graphs with a transitivity score higher than Forrest Gump's network.

> sum(gl.trs > g.tr)/1000

[1] 0

Cliques

Identifying cliques is a common practice in undirected networks. In a clique every two unique nodes are adjacent - that means that every individual node is connected to every other individual node in the clique. In this exercise you will identify the largest cliques in the Forrest Gump network. You will also identify the number of maximal cliques of various sizes. A clique is maximal if it cannot be extended to a larger clique.

Instructions

100xp

Identify the largest cliques in the network using the function largest\_cliques().

Determine all the maximal cliques in the network using the function max\_cliques(). Assign the output of this function to the list object clq.

Calculate the length of each of the maximal cliques. Use lapply() to loop through the object clq determining the length() of each object in the list. Then unlist() and use table() to observe how large each of the maximal cliques are.

Take Hint (-30xp)

> library(igraph)

>

> # Identify the largest cliques in the network

> largest\_cliques(g)

Warning message: At cliques.c:908 :directionality of edges is ignored for directed graphs

[[1]]

+ 9/94 vertices, named:

[1] FORREST STRONGARM BUBBA DALLAS LT DAN MAN SGT SIMS

[8] SOLDIER SONG

[[2]]

+ 9/94 vertices, named:

[1] FORREST JENNY EMCEE MAN # MAN #1 MAN #2 MAN #3 MAN #5 MEN

>

> # Determine all maximal cliques in the network and assign to object 'clq'

> clq <- max\_cliques(g)

Warning message: At maximal\_cliques\_template.h:203 :Edge directions are ignored for maximal clique calculation

>

> # Calculate the size of each maximal clique.

> table(unlist(lapply(clq, length)))

2 3 4 5 6 7 9

12 24 7 2 4 2 2

Visualize largest cliques

Often in network visualization you will need to subset part of a network to inspect the inter-connections of particular vertices. Here, you will create a visualization of the largest cliques in the Forrest Gump network. In the last exercise you determined that there were two cliques of size 9. You will plot these side-by-side after creating two new igraph objects by subsetting out these cliques from the main network. The function subgraph() enables you to choose which vertices to keep in a new network object.

Instructions

100xp

Instructions

100xp

Assign the list of the largest cliques in the network to the object lc.

Create two new undirected subgraphs using the function subgraph(). The first, gs1, should contain only the vertices in the first largest clique. The second, gs2, should contain only the vertices in the second largest clique. This function is wrapped in as.undirected() to ensure that the subgraph is also undirected.

Visualize the two largest cliques side by side using plot(). First execute the code: par(mfrow=c(1,2)). This is to ensure that the two visualizations sit side-by-side. Make sure that the layout is set to layout.circle() to make the visualization easier to view.

Take Hint (-30xp)

> library(igraph)

>

> # Assign largest cliques output to object 'lc'

> lc <- largest\_cliques(g)

Warning message: At cliques.c:908 :directionality of edges is ignored for directed graphs

>

> # Create two new undirected subgraphs, each containing only the vertices of each largest clique.

> gs1 <- as.undirected(subgraph(g, lc[[1]]))

Warning message: At structural\_properties.c:1945 :igraph\_subgraph is deprecated from igraph 0.6, use igraph\_induced\_subgraph instead

> gs2 <- as.undirected(subgraph(g, lc[[2]]))

Warning message: At structural\_properties.c:1945 :igraph\_subgraph is deprecated from igraph 0.6, use igraph\_induced\_subgraph instead

>

>

> # Plot the two largest cliques side-by-side

>

> par(mfrow=c(1,2)) # To plot two plots side-by-side

>

> plot(gs1,

vertex.label.color = "black",

vertex.label.cex = 0.9,

vertex.size = 0,

edge.color = 'gray28',

main = "Largest Clique 1",

layout = layout.circle(gs1)

)

>

> plot(gs2,

vertex.label.color = "black",

vertex.label.cex = 0.9,

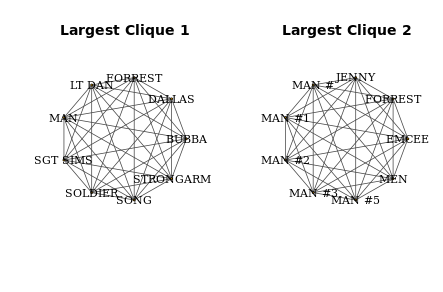
vertex.size = 0,

edge.color = 'gray28',

main = "Largest Clique 2",

layout = layout.circle(gs2)

)



Assortativity

In this exercise you will determine the assorativity() of the second friendship network from the first chapter. This is a measure of how preferentially attached vertices are to other vertices with identical attributes. You will also determine the degree assortativity which determines how preferentially attached are vertices to other vertices of a similar degree.

Instructions

100xp

Instructions

100xp

Make an exploratory plot of the friendship network object g1 using plot().

Convert the gender attribute of each vertex to a vector of numbers called values by factorizing and then using as.numeric().

Calculate the assortativity based on gender by using the function assortativity(). The first argument should be the graph object g1. The second argument are the values.

Calculate the degree assortativity of the network using assortativity.degree(). The first argument should be the graph object.

Take Hint (-30xp)

> # Plot the network

> plot(g1)

>

> # Convert the gender attribute into a numeric value

> values <- as.numeric(factor(V(g1)$gender))

>

> # Calculate the assortativity of the network based on gender

> assortativity(g1, values)

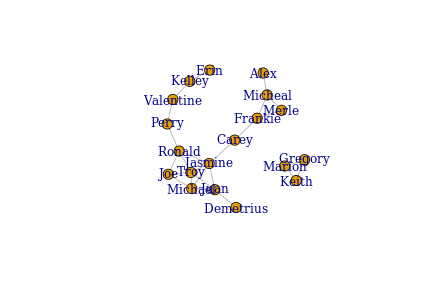
[1] 0.1319444

>

> # Calculate the assortativity degree of the network

> assortativity.degree(g1, directed = FALSE)

[1] 0.4615385



Using randomizations to assess assortativity

In this exercise you will determine how likely the observed assortativity in the friendship network is given the genders of vertices by performing a randomization procedure. You will randomly permute the gender of vertices in the network 1000 times and recalculate the assortativity for each random network.

Instructions

100xp

Instructions

100xp

Use assortativity() to calculate the assortativity of the graph object g1 based on gender using the object values calculated in the previous exercise, and assign this to the object observed.assortativity.

Inside the for loop calculate the assortativity of the network g1 using assortativity() while randomly permuting the object values each time with sample().

Plot the distribution of assortativity values from this permutation procedure using hist() and add a red vertical line for the original g1 network observed assortativity value that is stored in observed.assortativity.

Take Hint (-30xp)

# Calculate the observed assortativity

observed.assortativity <- assortativity(g1, values)

# Calculate the assortativity of the network randomizing the gender attribute 1000 times

results <- vector('list', 1000)

for(i in 1:1000){

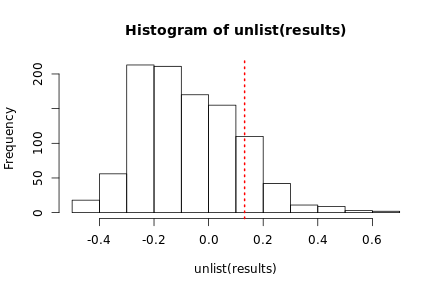
results[[i]] <- assortativity(g1, sample(values))

}

# Plot the distribution of assortativity values and add a red vertical line at the original observed value

hist(unlist(results))

abline(v = observed.assortativity, col = "red", lty = 3, lwd=2)



Reciprocity

The reciprocity of a directed network reflects the proportion of edges that are symmetrical. That is, the proportion of outgoing edges that also have an incoming edge. It is commonly used to determine how inter-connected directed networks are. An example of a such a network may be grooming exchanges in chimpanzees. Certain chimps may groom another but do not get groomed by that individual, whereas other chimps may both groom each other and so would have a reciprocal tie.

Instructions

100xp

In this example network of chimps grooming each other, make an exploratory plot of the network g using plot(). Make the arrow size 0.3 using the argument edge.arrow.size and the arrow width 0.5 using the argument edge.arrow.width.

Calculate the reciprocity of the graph using reciprocity().

Take Hint (-30xp)

>

> library(igraph)

>

> # Make a plot of the chimp grooming network

> plot(g,

edge.color = "black",

edge.arrow.size = 0.3,

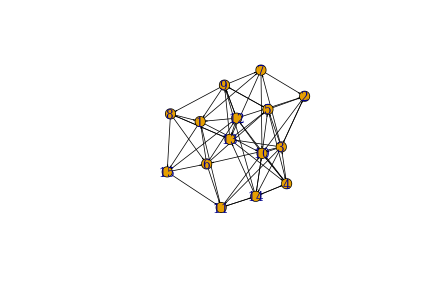
edge.arrow.width = 0.5)

>

> # Calculate the reciprocity of the graph

> reciprocity(g)

[1] 0.2711864



Fast-greedy community detection

The first community detection method you will try is fast-greedy community detection. You will use the Zachary Karate Club network. This social network contains 34 club members and 78 edges. Each edge indicates that those two club members interacted outside the karate club as well as at the club. Using this network you will determine how many sub-communities the network has and which club members belong to which subgroups. You will also plot the networks by community membership.

Instructions

100xp

Use the function fastgreedy.community() to create a community object. Assign this to the object kc.

Use the function sizes() on kc to determine how many communities were detected and how many club members are in each.

Display which club members are in which community using the function membership().

Make the default community plot by using the function plot(). The first argument should be the object kc and the second argument is the graph object g.

Take Hint (-30xp)

> # Perform fast-greedy community detection on network graph

> kc = fastgreedy.community(g)

>

> # Determine sizes of each community

> sizes(kc)

Community sizes

1 2 3

8 17 9

>

> # Determine which individuals belong to which community

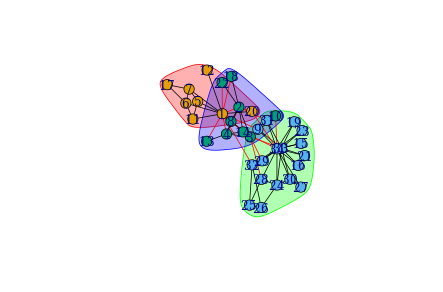
> membership(kc)

[1] 1 3 3 3 1 1 1 3 2 3 1 1 3 3 2 2 1 3 2 1 2 3 2 2 2 2 2 2 2 2 2 2 2 2

>

> # Plot the community structure of the network

> plot(kc, g)



Edge-betweenness community detection

An alternative community detection method is edge-betweenness. In this exercise you will repeat the community detection of the karate club using this method and compare the results visually to the fast-greedy method.

Instructions

100xp

Use the function edge.betweenness.community() on the graph object g to create the community igraph object gc.

Calculate the size and number of communities by using the function sizes on the community igraph object.

Plot each community plot next to each other using par(). The first plot should include the community object kc from the previous exercise. The second plot should include the community object gc.

Take Hint (-30xp)

> # Perform edge-betweenness community detection on network graph

> gc = edge.betweenness.community(g)

>

> # Determine sizes of each community

> sizes(gc)

Community sizes

1 2 3 4 5

10 6 5 12 1

>

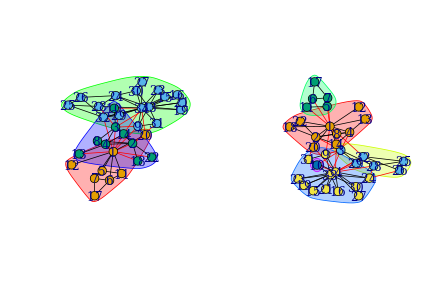
> # Plot community networks determined by fast-greedy and edge-betweenness methods side-by-side

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

> plot(kc, g)

> plot(gc, g)

>



Interactive networks with threejs

In this course you have exclusively used igraph to make basic static network plots. There are many packages available to make network plots. One very useful one is threejs which allows you to make interactive network visualizations. This package also integrates seamlessly with igraph. In this exercise you will make a basic interactive network plot of the karate club network using the threejs package. Once you have produced the visualization be sure to move the network around with your mouse. You should be able to scroll in and out of the network as well as rotate the network.

Instructions

100xp

First using set\_vertex\_attr() let's make a new vertex attribute called color that is dodgerblue.

Plot the network g using the threejs function graphjs(). The first argument should be the graph object g. Also make the vertex size equal to 1.

Take Hint (-30xp)

> library(igraph)

> library(threejs)

>

> # Set a vertex attribute called 'color' to 'dodgerblue'

> g <- set\_vertex\_attr(g, "color", value = "dodgerblue")

>

> # Redraw the graph and make the vertex size 1

> graphjs(g, vertex.size = 1)

THIS AMAZING PLOT IS SOMETHING YOU CAN ROTATE AND INTERACT WITH

AMAZING FOR PRESENTATION

Sizing vertices in threejs

As with all network visualizations it is often worth adjusting the size of vertices to illustrate their relative importance. This is also straightforward in threejs. In this exercise you will create an interactive threejs plot of the karate club network and size vertices based on their relative eigenvector centrality.

Instructions

100xp

Calculate the eigenvector centrality of each vertex using eigen\_centrality() and store the values in the object ec.

Using sqrt() adjust the values in ec to create a new vector of values v which is equal to five times the square root of the original eigenvector centrality.

Plot the network using the threejs function graphjs and making the argument vertex.size equal to the values in v.

Take Hint (-30xp)

> # Create numerical vector of vertex eigenvector centralities

> ec <- as.numeric(eigen\_centrality(g)$vector)

>

> # Create new vector 'v' that is equal to the square-root of 'ec' multiplied by 5

> v <- 5\*sqrt(ec)

>

> # Plot threejs plot of graph setting vertex size to v

> graphjs(g, vertex.size = v)

3D community network graph

Finally in this exercise you will create an interactive threejs plot with the vertices based on their community membership as produced by the fast-greedy community detection method.

Instructions

100xp

Use the function membership() on the community igraph object kc to generate a vector of community membership for each vertex.

Check how many communities there are using the function sizes() on the community igraph object kc.

Use set\_vertex\_attr() to add a vertex attribute called color to the graph object g. The values to add are the colors based on the membership assigned to object i.

Plot the three-dimensionsal graph by using the function graphjs() on the network object g.

Take Hint (-30xp)

> # Create an object 'i' containin the memberships of the fast-greedy community detection

> i <- membership(kc)

>

> # Check the number of different communities

> sizes(kc)

Community sizes

1 2 3

8 17 9

>

> # Add a color attribute to each vertex, setting the vertex color based on community membership

> g <- set\_vertex\_attr(g, "color", value = c("yellow", "blue", "red")[i])

>

> # Plot the graph using threejs

> graphjs(g)

AMAZING 3D MULTICOLORED INTERACTIVE GRAPH