iGraphDocs

**Basic Intro to R package igraph**

***Helen Kettle***

***March 20, 2018***

**Plotting Networks with igraph**

**Official website:**

<http://igraph.org/r/>

**Excellent tutorial:**

<http://kateto.net/networks-r-igraph>

**How I’ve used it for visualising the microbial groups in my microPop R package:**

<https://helenkett.shinyapps.io/MicrobialNetworks/>

**Example**

**Simplified Microbial Network for Methane Production**

Fermenting bacteria (‘**Fermenters**’) consume carbohydrates and convert them into bacterial growth, carbon dioxide, hydrogen and other products (e.g. volatile fatty acids).

Methane producing archea (‘**Methanogens**’) use the hydrogen and the carbon dioxide to grow, producing methane and water in the process.

This process is typically found in the rumen (e.g. in cows, goats, horses, sheep etc ) and sometimes in the human gut.

**Define Nodes**

nodes=cbind('id'=c('Fermenters','Methanogens','carbs','CO2','H2','other','CH4','H2O'),

'type'=c(rep('Microbe',2),rep('nonBio',6)))

nodes

## id type

## [1,] "Fermenters" "Microbe"

## [2,] "Methanogens" "Microbe"

## [3,] "carbs" "nonBio"

## [4,] "CO2" "nonBio"

## [5,] "H2" "nonBio"

## [6,] "other" "nonBio"

## [7,] "CH4" "nonBio"

## [8,] "H2O" "nonBio"

**Define Links**

links=cbind('from'=c('carbs',rep('Fermenters',3),rep('Methanogens',2),'CO2','H2'),

'to'=c('Fermenters','other','CO2','H2','CH4','H2O',rep('Methanogens',2)),

'type'=c('uptake',rep('output',5),rep('uptake',2)),

'weight'=rep(1,8))

links

## from to type weight

## [1,] "carbs" "Fermenters" "uptake" "1"

## [2,] "Fermenters" "other" "output" "1"

## [3,] "Fermenters" "CO2" "output" "1"

## [4,] "Fermenters" "H2" "output" "1"

## [5,] "Methanogens" "CH4" "output" "1"

## [6,] "Methanogens" "H2O" "output" "1"

## [7,] "CO2" "Methanogens" "uptake" "1"

## [8,] "H2" "Methanogens" "uptake" "1"

**Make the network**

library(igraph)

net = graph\_from\_data\_frame(links,vertices = nodes,directed = T)

plot(net)

**Change Appearance**

colrs.v = c(nonBio = "lightblue",Microbe = "gold") #node colours

V(net)$color = colrs.v[V(net)$type]

colrs.e = c(output = "grey", uptake = "magenta") #edge colours

E(net)$color = colrs.e[E(net)$type]

plot(net, edge.curved=0.2,vertex.size=30) #make nodes bigger, curve arrows

# Social Network Analysis for Anthropologists

## Materials for Short Course

[View on GitHub](https://github.com/eehh-stanford/SNA-workshop)

## Introduction to igraph

* [Getting Started](file:///G:\CSCI6444-Fall2020\iGraphDocs\Introduction%20to%20igraph%20Social%20Network%20Analysis%20for%20Anthropologists.html#getting-started)
* [Graphs](file:///G:\CSCI6444-Fall2020\iGraphDocs\Introduction%20to%20igraph%20Social%20Network%20Analysis%20for%20Anthropologists.html#graphs)
  + [Some Definitions](file:///G:\CSCI6444-Fall2020\iGraphDocs\Introduction%20to%20igraph%20Social%20Network%20Analysis%20for%20Anthropologists.html#some-definitions)
  + [From Graphs to People and Relationships](file:///G:\CSCI6444-Fall2020\iGraphDocs\Introduction%20to%20igraph%20Social%20Network%20Analysis%20for%20Anthropologists.html#from-graphs-to-people-and-relationships)
* [Various Ways to Specify Graphs in igraph](file:///G:\CSCI6444-Fall2020\iGraphDocs\Introduction%20to%20igraph%20Social%20Network%20Analysis%20for%20Anthropologists.html#various-ways-to-specify-graphs-in-igraph)
  + [Encoding a Graph by Hand](file:///G:\CSCI6444-Fall2020\iGraphDocs\Introduction%20to%20igraph%20Social%20Network%20Analysis%20for%20Anthropologists.html#encoding-a-graph-by-hand)
  + [Special Graphs: Empty, Full, Ring](file:///G:\CSCI6444-Fall2020\iGraphDocs\Introduction%20to%20igraph%20Social%20Network%20Analysis%20for%20Anthropologists.html#special-graphs-empty-full-ring)
  + [Special Graphs: Lattice, Tree, Star](file:///G:\CSCI6444-Fall2020\iGraphDocs\Introduction%20to%20igraph%20Social%20Network%20Analysis%20for%20Anthropologists.html#special-graphs-lattice-tree-star)
  + [Special Graphs: Erdos-Renyi & Power-Law](file:///G:\CSCI6444-Fall2020\iGraphDocs\Introduction%20to%20igraph%20Social%20Network%20Analysis%20for%20Anthropologists.html#special-graphs-erdos-renyi-power-law)
* [Putting Graphs Together](file:///G:\CSCI6444-Fall2020\iGraphDocs\Introduction%20to%20igraph%20Social%20Network%20Analysis%20for%20Anthropologists.html#putting-graphs-together)
* [Rewiring](file:///G:\CSCI6444-Fall2020\iGraphDocs\Introduction%20to%20igraph%20Social%20Network%20Analysis%20for%20Anthropologists.html#rewiring)
* [Vertex and Edge Attributes](file:///G:\CSCI6444-Fall2020\iGraphDocs\Introduction%20to%20igraph%20Social%20Network%20Analysis%20for%20Anthropologists.html#vertex-and-edge-attributes)
* [Adjacency Matrices](file:///G:\CSCI6444-Fall2020\iGraphDocs\Introduction%20to%20igraph%20Social%20Network%20Analysis%20for%20Anthropologists.html#adjacency-matrices)
* [Community Structure](file:///G:\CSCI6444-Fall2020\iGraphDocs\Introduction%20to%20igraph%20Social%20Network%20Analysis%20for%20Anthropologists.html#community-structure)
* [Laying Out Graphs By Hand](file:///G:\CSCI6444-Fall2020\iGraphDocs\Introduction%20to%20igraph%20Social%20Network%20Analysis%20for%20Anthropologists.html#laying-out-graphs-by-hand)
* [Plotting Affiliation Graphs](file:///G:\CSCI6444-Fall2020\iGraphDocs\Introduction%20to%20igraph%20Social%20Network%20Analysis%20for%20Anthropologists.html#plotting-affiliation-graphs)

Download the R script for this tutorial [here](https://eehh-stanford.github.io/SNA-workshop/intro-igraph.R).

Back to [main page](https://eehh-stanford.github.io/SNA-workshop/).

### Getting Started

* igraph is a package that provides tools for the analysis and visualization of networks

library(igraph)

### Graphs

#### Some Definitions

**Graph:** A collection of vertices (or nodes) and undirected edges (or ties), denoted (V, E), where V is a the vertex set and E is the edge set.

**Digraph (Directed Graph):** A collection of vertices (or nodes) and directed edges.

**Bipartite Graph:** Graph where all the nodes of a graph can be partitioned into two sets 𝒱1 and 𝒱2 such that for all edges in the graph connects and unordered pair where one vertex comes from 𝒱1 and the other from 𝒱2. Often called an “affiliation graph” as bipartite graphs are used to represent people’s affiliations to organizations or events.

#### From Graphs to People and Relationships

* The vertices of the graph represent the actors in the social system. These are usually individual people, but they could be households, geographical localities, institutions, or other social entities.
* The edges of the graph represent the relations between these entities (e.g., “is friends with” or “has sexual intercourse with” or “sends money to”). These edges can be directed undirected (e.g., “within 2 meters of”) or directed (e.g., “sends money to”), in the case of a **digraph**.
* Graphs (and digraphs) can be binary (i.e., presence/absence of a relationship) or valued (e.g., “groomed five times in the observation period”, “sent $100”).
* A graph (with no self-loops) with n vertices has ${n \choose 2} = n(n-1)/2$ possible unordered pairs. This number (which can get very big!) is important for defining the density of a graph, i.e., the fraction of all possible relations that actually exist in a network.

### Various Ways to Specify Graphs in igraph

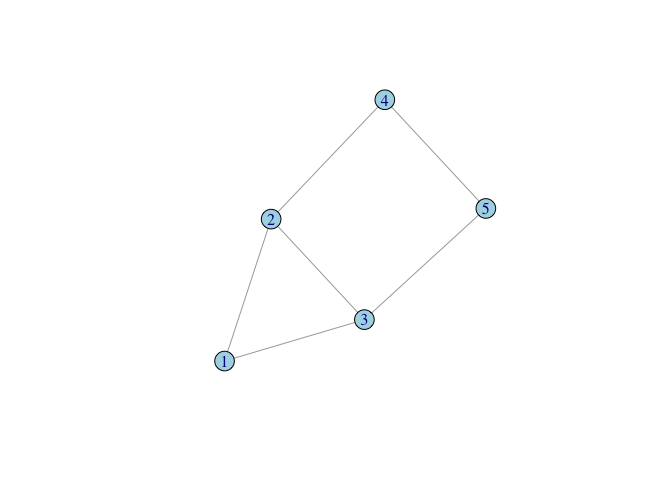
#### Encoding a Graph by Hand

* Create a small, undirected graph of five vertices from a vector of vertex pairs

require(igraph)

g <- make\_graph(c(1,2, 1,3, 2,3, 2,4, 3,5, 4,5), n=5, dir=FALSE)

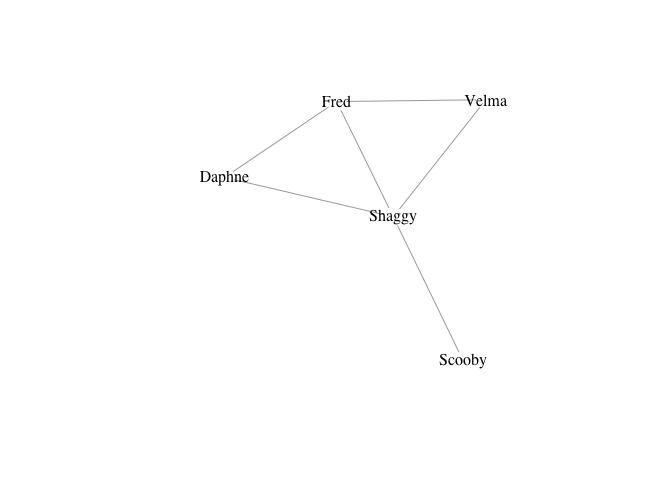
plot(g, vertex.color="lightblue")



* Create a small graph using graph\_from\_literal()
* Undirected edges are indicated with one or more dashes -, --, etc. It doesn’t matter how many dashes you use – you can use as many as you want to make your code more readable.
* The colon operator : links “vertex sets” – i.e., creates ties between all members of two groups of vertices

g <- graph\_from\_literal(Fred-Daphne:Velma-Shaggy, Fred-Shaggy-Scooby)

plot(g, vertex.shape="none", vertex.label.color="black")



* Make directed edges using -+ where the plus indicates the direction of the arrow, i.e., A --+ B creates a directed edge from A to B
* A mutual edge can be created using +-+

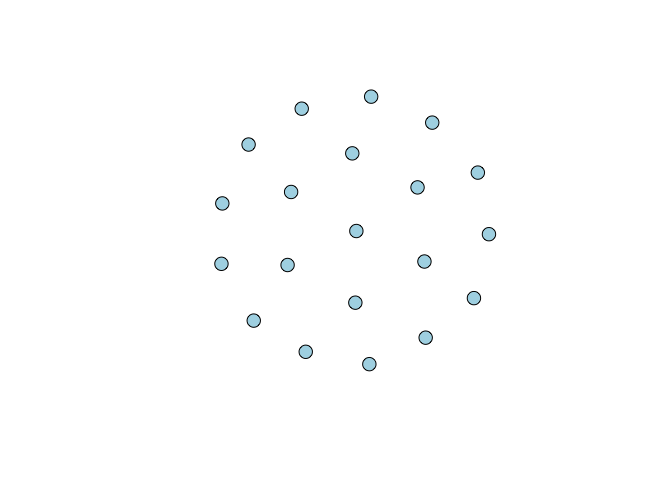
#### Special Graphs: Empty, Full, Ring

* I really don’t like the current default color in igraph, so I set the vertex color for every plot

# empty graph

g0 <- make\_empty\_graph(20)

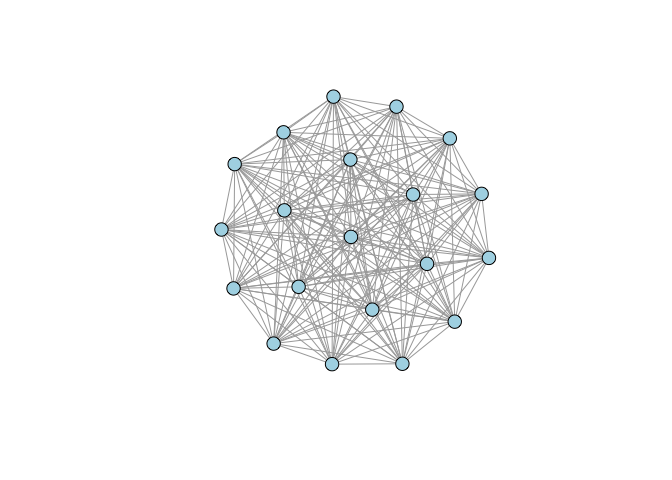
plot(g0, vertex.color="lightblue", vertex.size=10, vertex.label=NA)



# full graph

g1 <- make\_full\_graph(20)

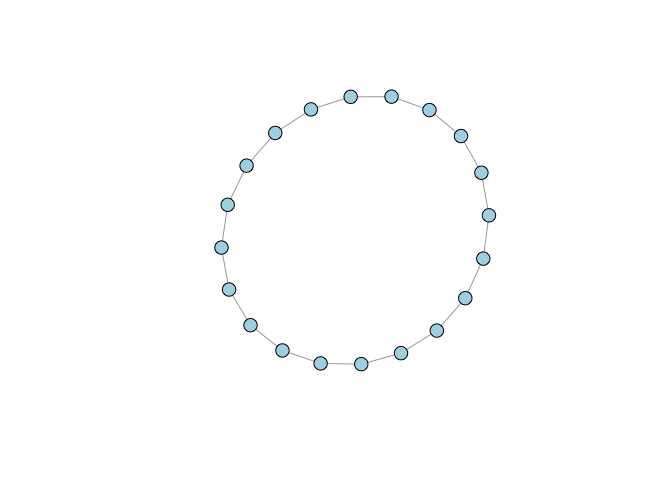
plot(g1, vertex.color="lightblue", vertex.size=10, vertex.label=NA)



# ring

g2 <- make\_ring(20)

plot(g2, vertex.color="lightblue", vertex.size=10, vertex.label=NA)

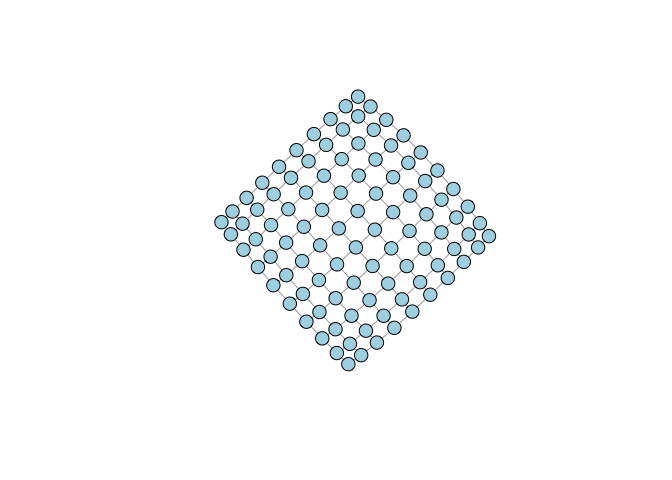


#### Special Graphs: Lattice, Tree, Star

# lattice

g3 <- make\_lattice(dimvector=c(10,10))

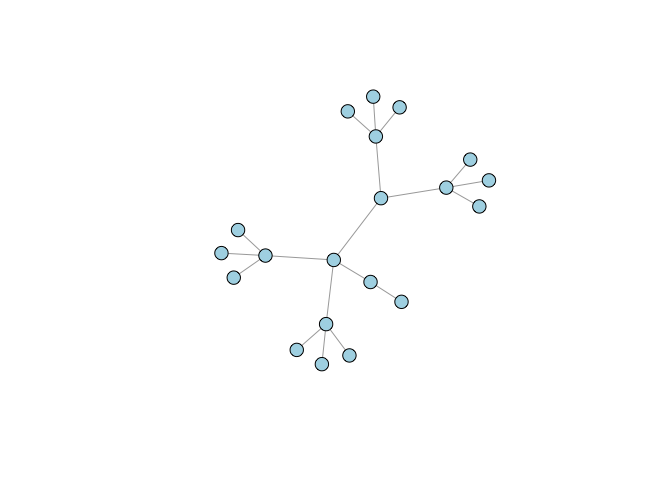
plot(g3, vertex.color="lightblue", vertex.size=10, vertex.label=NA)



# tree

g4 <- make\_tree(20, children = 3, mode = "undirected")

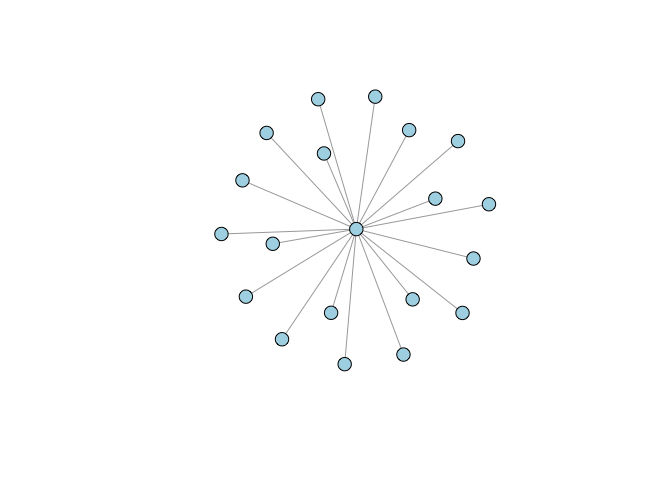
plot(g4, vertex.color="lightblue", vertex.size=10, vertex.label=NA)



# star

g5 <- make\_star(20, mode="undirected")

plot(g5, vertex.color="lightblue", vertex.size=10, vertex.label=NA)

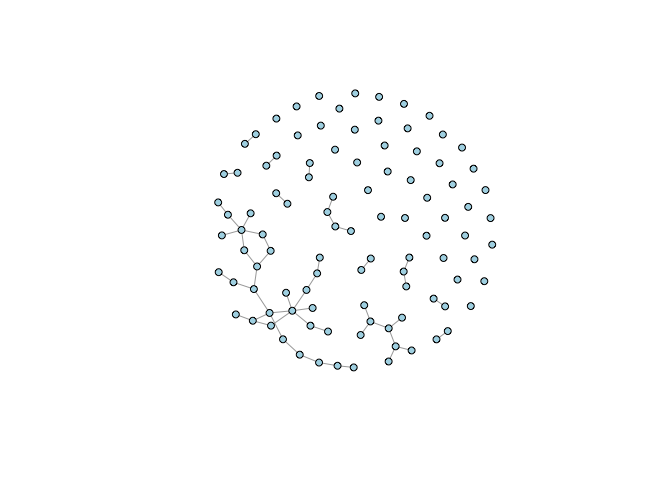


#### Special Graphs: Erdos-Renyi & Power-Law

# Erdos-Renyi Random Graph

g6 <- sample\_gnm(n=100,m=50)

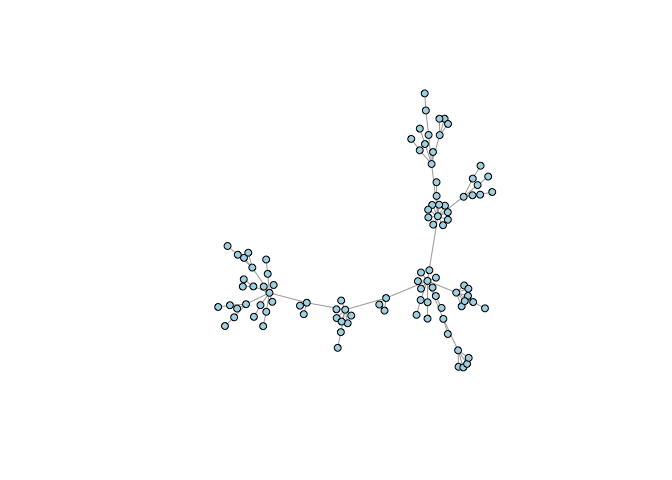
plot(g6, vertex.color="lightblue", vertex.size=5, vertex.label=NA)



# Power Law

g7 <- sample\_pa(n=100, power=1.5, m=1, directed=FALSE)

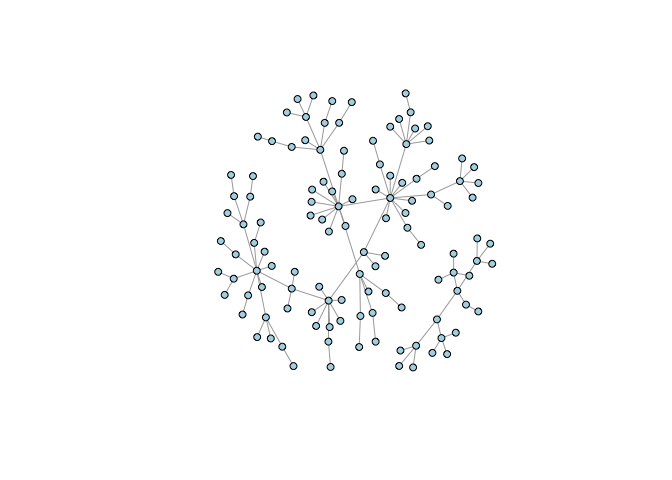
plot(g7, vertex.color="lightblue", vertex.size=5, vertex.label=NA)



### Putting Graphs Together

* Sometimes you want to plot two (or more) graphs together
* The disjoint union operator allows you to merge two graphs with different vertex sets

plot(g4 %du% g7, vertex.color="lightblue", vertex.size=5, vertex.label=NA)



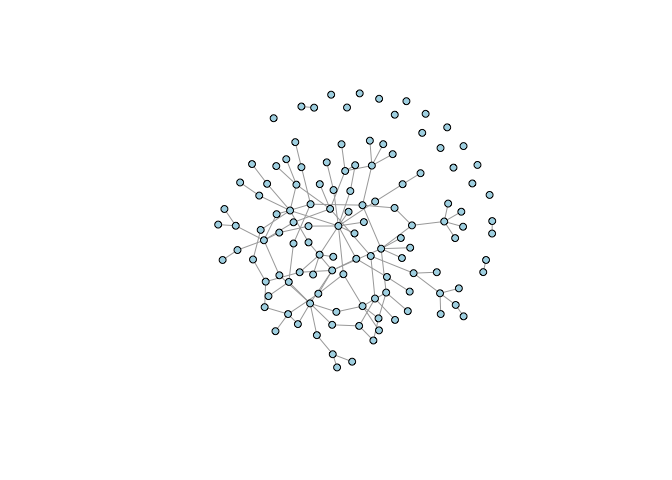
### Rewiring

* Rewiring means rearranging the ties in a graph. It randomizes the connections between nodes without changing the degree distribution

gg <- g4 %du% g7

gg <- rewire(gg, each\_edge(prob = 0.3))

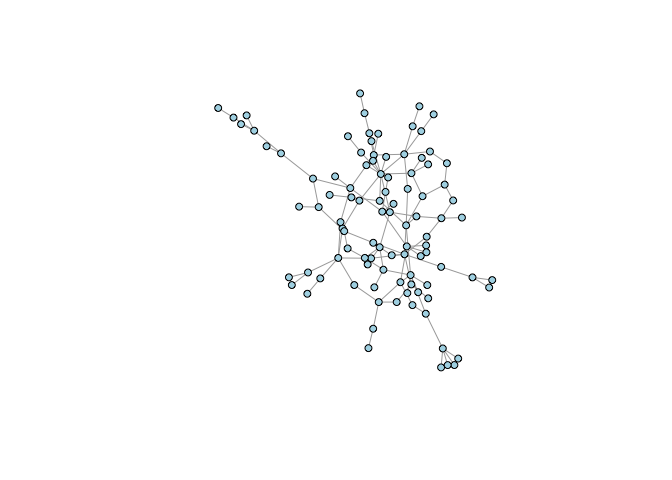
plot(gg, vertex.color="lightblue", vertex.size=5, vertex.label=NA)



## retain only the connected component

gg <- induced.subgraph(gg, subcomponent(gg,1))

plot(gg, vertex.color="lightblue", vertex.size=5, vertex.label=NA)



### Vertex and Edge Attributes

* You can add arbitrary attributes to both vertices and edges. Generally, you do this to store information for plotting: colors, edge weights, names, etc.
* Some attributes are automatically created when you construct an graph object (e.g., “name” or “weight” if you load a weighted adjacency matrix)
* V(g) accesses vertex attributes
* E(g) accesses edge attributes

## look at the structure

g4

## IGRAPH 52d3b03 U--- 20 19 -- Tree

## + attr: name (g/c), children (g/n), mode (g/c)

## + edges from 52d3b03:

## [1] 1-- 2 1-- 3 1-- 4 2-- 5 2-- 6 2-- 7 3-- 8 3-- 9 3--10 4--11 4--12

## [12] 4--13 5--14 5--15 5--16 6--17 6--18 6--19 7--20

V(g4)$name <- LETTERS[1:20]

## see how it's changed

g4

## IGRAPH 52d3b03 UN-- 20 19 -- Tree

## + attr: name (g/c), children (g/n), mode (g/c), name (v/c)

## + edges from 52d3b03 (vertex names):

## [1] A--B A--C A--D B--E B--F B--G C--H C--I C--J D--K D--L D--M E--N E--O

## [15] E--P F--Q F--R F--S G--T

## see what I did there?

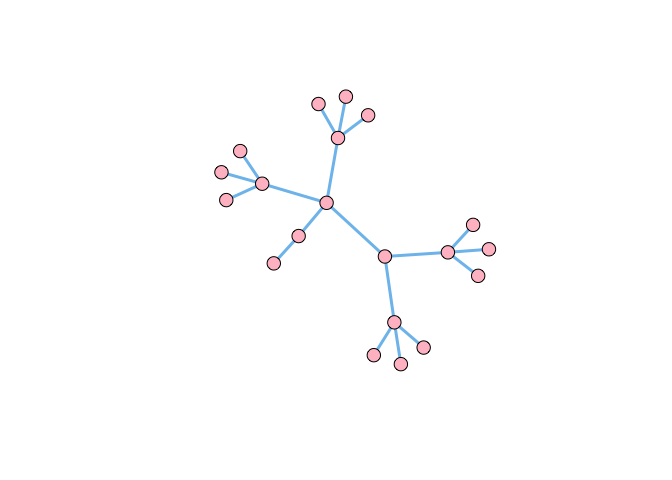
## do some other stuff

V(g4)$vertex.color <- "Pink"

E(g4)$edge.color <- "SkyBlue2"

plot(g4, vertex.size=10, vertex.label=NA, vertex.color=V(g4)$vertex.color,

edge.color=E(g4)$edge.color, edge.width=3)



### Adjacency Matrices

* Most primatologists/behavioral ecologists probably have experience thinking in terms of adjacency matrices
* An example of an adjacency matrix is the pairwise interaction matrices (e.g., agonistic or affiliative interactions) that we construct from behavioral observations
* A very important potential gotcha: when you read data into R, it will be in the form of a data frame. Converting an adjacency matrix to an igraph graph object requires the data to be in the matrix class. Therefore, you need to coerce the data you read in by wrapping your read.table() in an as.matrix() command.

kids <- as.matrix(

read.table("http://web.stanford.edu/class/ess360/data/strayer\_strayer1976-fig2.txt",

header=FALSE)

)

kid.names <- c("Ro","Ss","Br","If","Td","Sd","Pe","Ir","Cs","Ka",

"Ch","Ty","Gl","Sa", "Me","Ju","Sh")

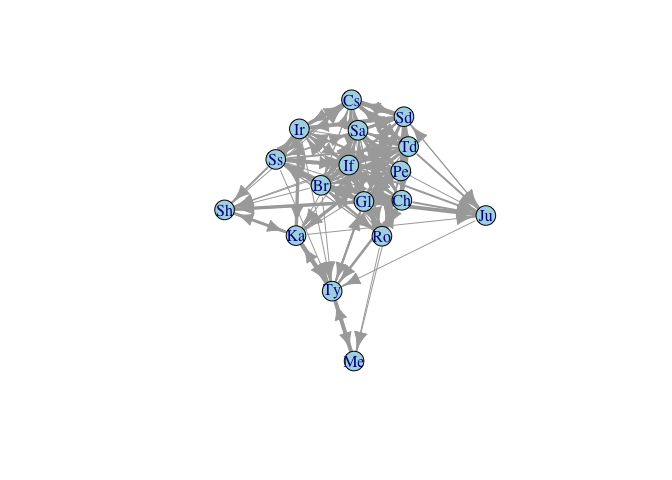
colnames(kids) <- kid.names

rownames(kids) <- kid.names

g <- graph\_from\_adjacency\_matrix(kids, mode="directed", weighted=TRUE)

lay <- layout\_with\_fr(g)

plot(g,edge.width=log2(E(g)$weight)+1, layout=lay, vertex.color="lightblue")



* Adjacency matrices are actually very inefficient
* Most sociomatrices are quite sparse
* Cost of an adjacency matrix increases as k2
* Edge Lists are much more efficient

### Community Structure

* Various algorithms for detecting clusters of similar vertices – i.e., “communities”
* Use fastgreedy.community() to identify clusters in Kapferer’s tailor shop and color the vertices based on their membership
* fastgreedy.community() identifies four clusters
* These clusters are listed as numbers in fg$membership
* Use this vector to index vertex colors

A <- as.matrix(

read.table(file="http://web.stanford.edu/class/ess360/data/kapferer-tailorshop1.txt",

header=TRUE, row.names=1)

)

G <- graph.adjacency(A, mode="undirected", diag=FALSE)

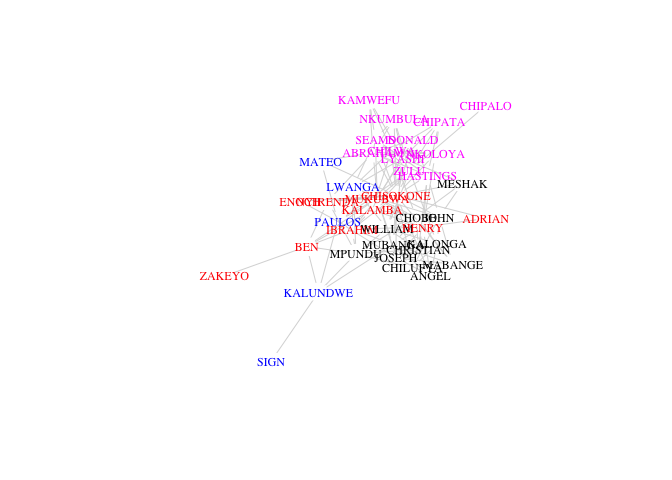
fg <- fastgreedy.community(G)

cols <- c("blue","red","black","magenta")

plot(G, vertex.shape="none",

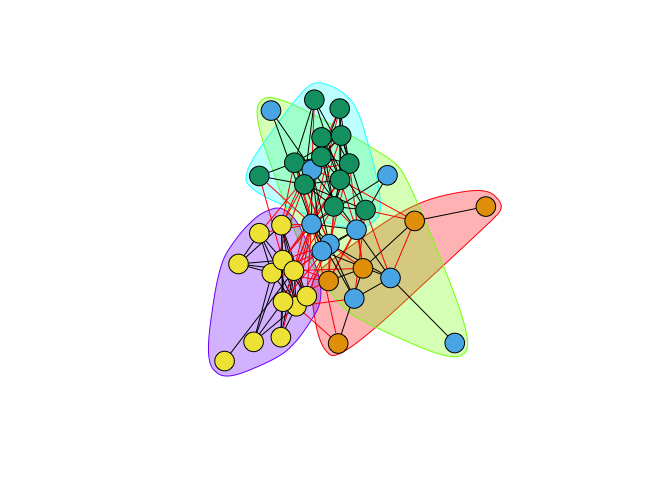
vertex.label.cex=0.75, edge.color=grey(0.85),

edge.width=1, vertex.label.color=cols[fg$membership])



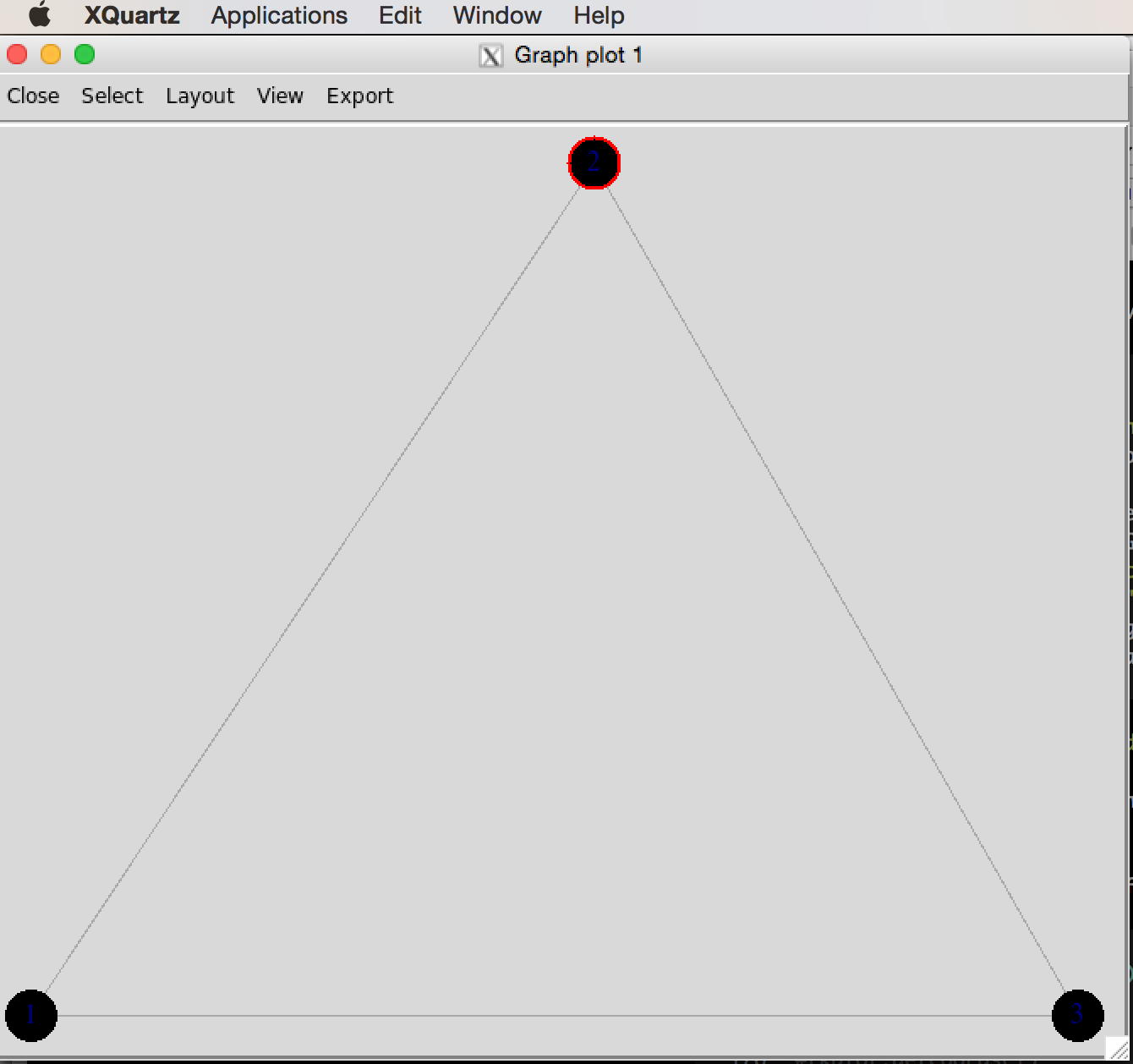
# another approach to visualizing

plot(fg,G,vertex.label=NA)



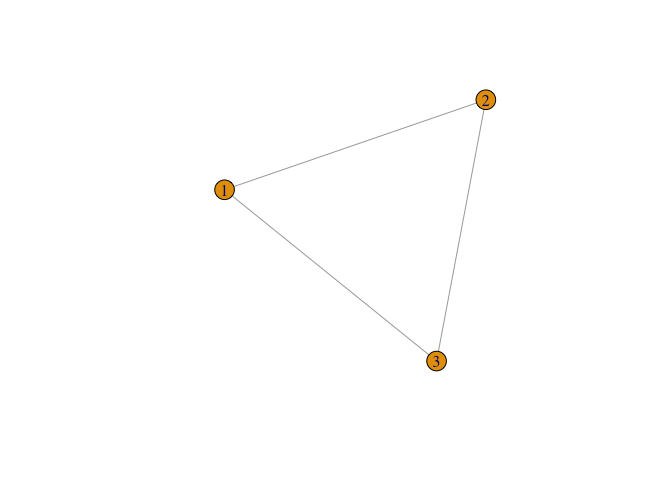
### Laying Out Graphs By Hand

* The layout is of any given plot is random (e.g., plot the same graph repeatedly and you’ll see that the layout changes with each plot)
* igraph provides a tool for tinkering with the layout called tkplot()
* Call tkplot() and it will open an X11 window (on Macs at least)
* Select and drag the vertices into the layout you want, then use tkplot.getcoords(gid) to get the coordinates (where gid is the graph id returned when calling tkplot() on your graph)



g <- graph( c(1,2, 2,3, 1,3), n=3, dir=FALSE)

plot(g)



#tkplot(g)

#tkplot.getcoords(1)

### do some stuff with tkplot() and get coords which we call tri.coords

## tkplot(g)

## tkplot.getcoords(1) ## the plot id may be different depending on how many times you've called tkplot()

## [,1] [,2]

##[1,] 228 416

##[2,] 436 0

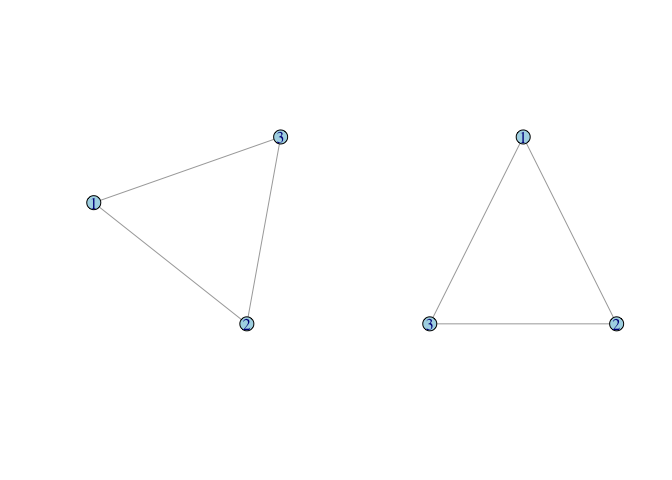
##[3,] 20 0

tri.coords <- matrix( c(228,416, 436,0, 20,0), nr=3, nc=2, byrow=TRUE)

par(mfrow=c(1,2))

plot(g, vertex.color="lightblue")

plot(g, layout=tri.coords, vertex.color="lightblue")



### Plotting Affiliation Graphs

davismat <- as.matrix(

read.table(file="http://web.stanford.edu/class/ess360/data/davismat.txt",

row.names=1, header=TRUE)

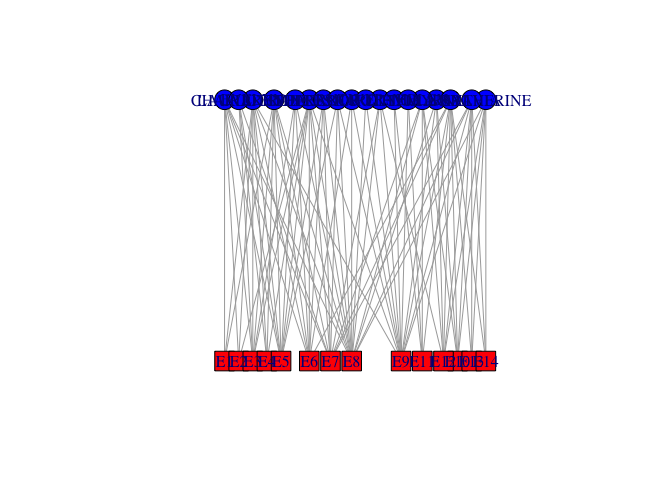
)

southern <- graph\_from\_incidence\_matrix(davismat)

V(southern)$shape <- c(rep("circle",18), rep("square",14))

V(southern)$color <- c(rep("blue",18), rep("red", 14))

plot(southern, layout=layout.bipartite)



## not so beautiful

## did some tinkering using tkplot()...

x <- c(rep(23,18), rep(433,14))

y <- c(44.32432, 0.00000, 132.97297, 77.56757, 22.16216, 110.81081, 155.13514,

199.45946, 177.29730, 243.78378, 332.43243, 410.00000, 387.83784, 354.59459,

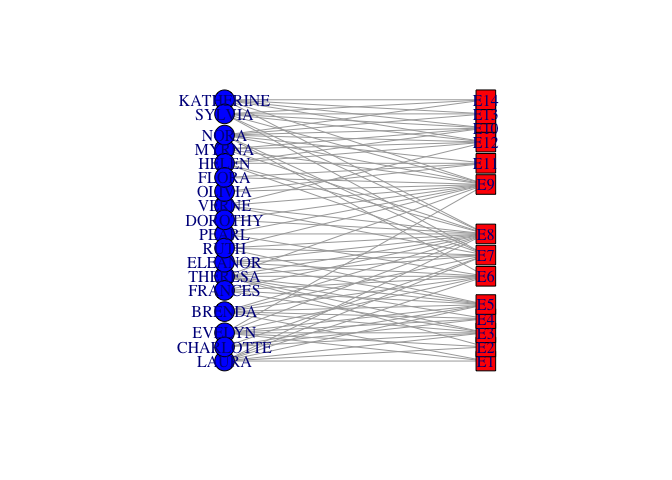
310.27027, 221.62162, 265.94595, 288.10811, 0.00000, 22.16216, 44.32432,

66.48649, 88.64865, 132.97297, 166.21622, 199.45946, 277.02703, 365.67568,

310.27027, 343.51351, 387.83784, 410.00000)

southern.layout <- cbind(x,y)

plot(southern, layout=southern.layout)



* The incidence matrix is n × k, where n is the number of actors and k is the number of events
* Project the incidence matrix X into social space, creating a sociomatrix A, **A** = **X** **X**T
* This transforms the n × k into an n × n sociomatrix

#Sociomatrix

(f2f <- davismat %\*% t(davismat))

## EVELYN LAURA THERESA BRENDA CHARLOTTE FRANCES ELEANOR PEARL RUTH

## EVELYN 8 6 7 6 3 4 3 3 3

## LAURA 6 7 6 6 3 4 4 2 3

## THERESA 7 6 8 6 4 4 4 3 4

## BRENDA 6 6 6 7 4 4 4 2 3

## CHARLOTTE 3 3 4 4 4 2 2 0 2

## FRANCES 4 4 4 4 2 4 3 2 2

## ELEANOR 3 4 4 4 2 3 4 2 3

## PEARL 3 2 3 2 0 2 2 3 2

## RUTH 3 3 4 3 2 2 3 2 4

## VERNE 2 2 3 2 1 1 2 2 3

## MYRNA 2 1 2 1 0 1 1 2 2

## KATHERINE 2 1 2 1 0 1 1 2 2

## SYLVIA 2 2 3 2 1 1 2 2 3

## NORA 2 2 3 2 1 1 2 2 2

## HELEN 1 2 2 2 1 1 2 1 2

## DOROTHY 2 1 2 1 0 1 1 2 2

## OLIVIA 1 0 1 0 0 0 0 1 1

## FLORA 1 0 1 0 0 0 0 1 1

## VERNE MYRNA KATHERINE SYLVIA NORA HELEN DOROTHY OLIVIA FLORA

## EVELYN 2 2 2 2 2 1 2 1 1

## LAURA 2 1 1 2 2 2 1 0 0

## THERESA 3 2 2 3 3 2 2 1 1

## BRENDA 2 1 1 2 2 2 1 0 0

## CHARLOTTE 1 0 0 1 1 1 0 0 0

## FRANCES 1 1 1 1 1 1 1 0 0

## ELEANOR 2 1 1 2 2 2 1 0 0

## PEARL 2 2 2 2 2 1 2 1 1

## RUTH 3 2 2 3 2 2 2 1 1

## VERNE 4 3 3 4 3 3 2 1 1

## MYRNA 3 4 4 4 3 3 2 1 1

## KATHERINE 3 4 6 6 5 3 2 1 1

## SYLVIA 4 4 6 7 6 4 2 1 1

## NORA 3 3 5 6 8 4 1 2 2

## HELEN 3 3 3 4 4 5 1 1 1

## DOROTHY 2 2 2 2 1 1 2 1 1

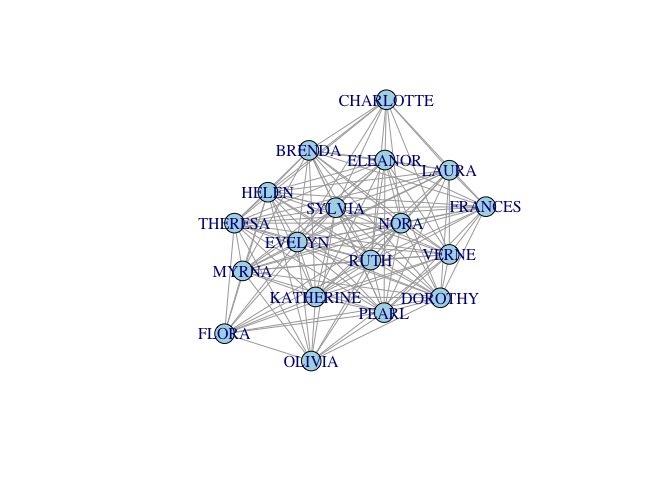
## OLIVIA 1 1 1 1 2 1 1 2 2

## FLORA 1 1 1 1 2 1 1 2 2

gf2f <- graph\_from\_adjacency\_matrix(f2f, mode="undirected", diag=FALSE, add.rownames=TRUE)

gf2f <- simplify(gf2f)

plot(gf2f, vertex.color="lightblue")



## who is the most central?

cb <- betweenness(gf2f)

#plot(gf2f,vertex.size=cb\*10, vertex.color="lightblue")

plot(gf2f,vertex.label.cex=1+cb/2, vertex.shape="none")



* Project the matrix into event space

### this gives you the number of women at each event (diagonal) or mutually at 2 events

(e2e <- t(davismat) %\*% davismat)

## E1 E2 E3 E4 E5 E6 E7 E8 E9 E10 E11 E12 E13 E14

## E1 3 2 3 2 3 3 2 3 1 0 0 0 0 0

## E2 2 3 3 2 3 3 2 3 2 0 0 0 0 0

## E3 3 3 6 4 6 5 4 5 2 0 0 0 0 0

## E4 2 2 4 4 4 3 3 3 2 0 0 0 0 0

## E5 3 3 6 4 8 6 6 7 3 0 0 0 0 0

## E6 3 3 5 3 6 8 5 7 4 1 1 1 1 1

## E7 2 2 4 3 6 5 10 8 5 3 2 4 2 2

## E8 3 3 5 3 7 7 8 14 9 4 1 5 2 2

## E9 1 2 2 2 3 4 5 9 12 4 3 5 3 3

## E10 0 0 0 0 0 1 3 4 4 5 2 5 3 3

## E11 0 0 0 0 0 1 2 1 3 2 4 2 1 1

## E12 0 0 0 0 0 1 4 5 5 5 2 6 3 3

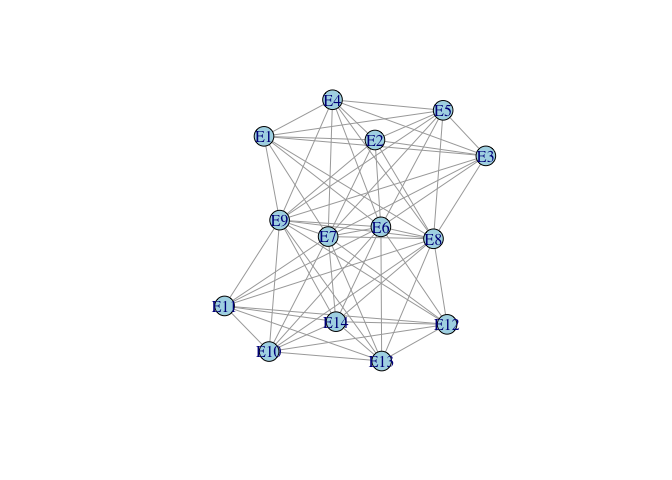
## E13 0 0 0 0 0 1 2 2 3 3 1 3 3 3

## E14 0 0 0 0 0 1 2 2 3 3 1 3 3 3

ge2e <- graph\_from\_adjacency\_matrix(e2e, mode="undirected", diag=FALSE, add.rownames=TRUE)

ge2e <- simplify(ge2e)

plot(ge2e, vertex.color="lightblue")



# Introduction to social network analysis with R

#### Pablo Barbera

#### June 27, 2017

#### Importing network data into R

In this training session we will be using a small network that indicates [interactions in the movie Star Wars Episode IV](http://evelinag.com/blog/2015/12-15-star-wars-social-network/). Here, each node is a character and each edge indicates whether they appeared together in a scene of the movie. Edges here are thus undirected and they also have weights attached, since they can appear in multiple scenes together.

The first step is to read the list of edges and nodes in this network:

edges <- read.csv("data/star-wars-network-edges.csv")

head(edges)

## source target weight

## 1 C-3PO R2-D2 17

## 2 LUKE R2-D2 13

## 3 OBI-WAN R2-D2 6

## 4 LEIA R2-D2 5

## 5 HAN R2-D2 5

## 6 CHEWBACCA R2-D2 3

nodes <- read.csv("data/star-wars-network-nodes.csv")

head(nodes)

## name id

## 1 R2-D2 0

## 2 CHEWBACCA 1

## 3 C-3PO 2

## 4 LUKE 3

## 5 DARTH VADER 4

## 6 CAMIE 5

For example, we learn that C-3PO and R2-D2 appeared in 17 scenes together.

How do we convert these two datasets into a network object in R? There are multiple packages to work with networks, but the most popular is igraph because it’s very flexible and easy to do, and in my experience it’s much faster and scales well to very large networks. Other packages that you may want to explore are sna and networks.

Now, how do we create the igraph object? We can use the graph\_from\_data\_frame function, which takes two arguments: d, the data frame with the edge list in the first two columns; and vertices, a data frame with node data with the node label in the first column. (Note that igraph calls the nodes vertices, but it’s exactly the same thing.)

library(igraph)

g <- graph\_from\_data\_frame(d=edges, vertices=nodes, directed=FALSE)

g

## IGRAPH UNW- 22 60 --

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

## + edges (vertex names):

## [1] R2-D2 --C-3PO R2-D2 --LUKE

## [3] R2-D2 --OBI-WAN R2-D2 --LEIA

## [5] R2-D2 --HAN R2-D2 --CHEWBACCA

## [7] R2-D2 --DODONNA CHEWBACCA --OBI-WAN

## [9] CHEWBACCA --C-3PO CHEWBACCA --LUKE

## [11] CHEWBACCA --HAN CHEWBACCA --LEIA

## [13] CHEWBACCA --DARTH VADER CHEWBACCA --DODONNA

## [15] LUKE --CAMIE CAMIE --BIGGS

## + ... omitted several edges

What does it mean? - U means undirected  
- N means named graph  
- W means weighted graph  
- 22 is the number of nodes  
- 60 is the number of edges  
- name (v/c) means name is a node attribute and it’s a character  
- weight (e/n) means weight is an edge attribute and it’s numeric

This is how you access specific elements within the igraph object:

V(g) # nodes

## + 22/22 vertices, named:

## [1] R2-D2 CHEWBACCA C-3PO LUKE DARTH VADER

## [6] CAMIE BIGGS LEIA BERU OWEN

## [11] OBI-WAN MOTTI TARKIN HAN GREEDO

## [16] JABBA DODONNA GOLD LEADER WEDGE RED LEADER

## [21] RED TEN GOLD FIVE

V(g)$name # names of each node

## [1] "R2-D2" "CHEWBACCA" "C-3PO" "LUKE" "DARTH VADER"

## [6] "CAMIE" "BIGGS" "LEIA" "BERU" "OWEN"

## [11] "OBI-WAN" "MOTTI" "TARKIN" "HAN" "GREEDO"

## [16] "JABBA" "DODONNA" "GOLD LEADER" "WEDGE" "RED LEADER"

## [21] "RED TEN" "GOLD FIVE"

vertex\_attr(g) # all attributes of the nodes

## $name

## [1] "R2-D2" "CHEWBACCA" "C-3PO" "LUKE" "DARTH VADER"

## [6] "CAMIE" "BIGGS" "LEIA" "BERU" "OWEN"

## [11] "OBI-WAN" "MOTTI" "TARKIN" "HAN" "GREEDO"

## [16] "JABBA" "DODONNA" "GOLD LEADER" "WEDGE" "RED LEADER"

## [21] "RED TEN" "GOLD FIVE"

##

## $id

## [1] 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21

E(g) # edges

## + 60/60 edges (vertex names):

## [1] R2-D2 --C-3PO R2-D2 --LUKE

## [3] R2-D2 --OBI-WAN R2-D2 --LEIA

## [5] R2-D2 --HAN R2-D2 --CHEWBACCA

## [7] R2-D2 --DODONNA CHEWBACCA --OBI-WAN

## [9] CHEWBACCA --C-3PO CHEWBACCA --LUKE

## [11] CHEWBACCA --HAN CHEWBACCA --LEIA

## [13] CHEWBACCA --DARTH VADER CHEWBACCA --DODONNA

## [15] LUKE --CAMIE CAMIE --BIGGS

## [17] LUKE --BIGGS DARTH VADER--LEIA

## [19] LUKE --BERU BERU --OWEN

## + ... omitted several edges

E(g)$weight # weights for each edge

## [1] 17 13 6 5 5 3 1 7 5 16 19 11 1 1 2 2 4 1 3 3 2 3 18

## [24] 2 6 17 1 19 6 1 2 1 7 9 26 1 1 6 1 1 13 1 1 1 1 1

## [47] 1 2 1 1 3 3 1 1 3 1 2 1 1 1

edge\_attr(g) # all attributes of the edges

## $weight

## [1] 17 13 6 5 5 3 1 7 5 16 19 11 1 1 2 2 4 1 3 3 2 3 18

## [24] 2 6 17 1 19 6 1 2 1 7 9 26 1 1 6 1 1 13 1 1 1 1 1

## [47] 1 2 1 1 3 3 1 1 3 1 2 1 1 1

g[] # adjacency matrix

## 22 x 22 sparse Matrix of class "dgCMatrix"

## [[ suppressing 22 column names 'R2-D2', 'CHEWBACCA', 'C-3PO' ... ]]

##

## R2-D2 . 3 17 13 . . . 5 . . 6 . . 5 . . 1 . . . . .

## CHEWBACCA 3 . 5 16 1 . . 11 . . 7 . . 19 . . 1 . . . . .

## C-3PO 17 5 . 18 . . 1 6 2 2 6 . . 6 . . . . . 1 . .

## LUKE 13 16 18 . . 2 4 17 3 3 19 . . 26 . . 1 1 2 3 1 .

## DARTH VADER . 1 . . . . . 1 . . 1 1 7 . . . . . . . . .

## CAMIE . . . 2 . . 2 . . . . . . . . . . . . . . .

## BIGGS . . 1 4 . 2 . 1 . . . . . . . . . 1 2 3 . .

## LEIA 5 11 6 17 1 . 1 . 1 . 1 1 1 13 . . . . . 1 . .

## BERU . . 2 3 . . . 1 . 3 . . . . . . . . . . . .

## OWEN . . 2 3 . . . . 3 . . . . . . . . . . . . .

## OBI-WAN 6 7 6 19 1 . . 1 . . . . . 9 . . . . . . . .

## MOTTI . . . . 1 . . 1 . . . . 2 . . . . . . . . .

## TARKIN . . . . 7 . . 1 . . . 2 . . . . . . . . . .

## HAN 5 19 6 26 . . . 13 . . 9 . . . 1 1 . . . . . .

## GREEDO . . . . . . . . . . . . . 1 . . . . . . . .

## JABBA . . . . . . . . . . . . . 1 . . . . . . . .

## DODONNA 1 1 . 1 . . . . . . . . . . . . . 1 1 . . .

## GOLD LEADER . . . 1 . . 1 . . . . . . . . . 1 . 1 1 . .

## WEDGE . . . 2 . . 2 . . . . . . . . . 1 1 . 3 . .

## RED LEADER . . 1 3 . . 3 1 . . . . . . . . . 1 3 . 1 .

## RED TEN . . . 1 . . . . . . . . . . . . . . . 1 . .

## GOLD FIVE . . . . . . . . . . . . . . . . . . . . . .

g[1,] # first row of adjacency matrix

## R2-D2 CHEWBACCA C-3PO LUKE DARTH VADER CAMIE

## 0 3 17 13 0 0

## BIGGS LEIA BERU OWEN OBI-WAN MOTTI

## 0 5 0 0 6 0

## TARKIN HAN GREEDO JABBA DODONNA GOLD LEADER

## 0 5 0 0 1 0

## WEDGE RED LEADER RED TEN GOLD FIVE

## 0 0 0 0

#### Network visualization

How can we visualize this network? The plot() function works out of the box, but the default options are often not ideal:

par(mar=c(0,0,0,0))

plot(g)

Let’s see how we can improve this figure. To see all the available plotting options, you can check ?igraph.plotting. Let’s start by fixing some of these.

par(mar=c(0,0,0,0))

plot(g,

vertex.color = "grey", # change color of nodes

vertex.label.color = "black", # change color of labels

vertex.label.cex = .75, # change size of labels to 75% of original size

edge.curved=.25, # add a 25% curve to the edges

edge.color="grey20") # change edge color to grey

Now imagine that we want to modify some of these plotting attributes so that they are function of network properties. For example, a common adjustment is to change the size of the nodes and node labels so that they match their importance (we’ll come back to how to measure that later). Here, strength will correspond to the number of scenes they appear in. And we’re only going to show the labels of character that appear in 10 or more scenes.

V(g)$size <- strength(g)

par(mar=c(0,0,0,0)); plot(g)

# taking the log to improve it

V(g)$size <- log(strength(g)) \* 4 + 3

par(mar=c(0,0,0,0)); plot(g)

V(g)$label <- ifelse( strength(g)>=10, V(g)$name, NA )

par(mar=c(0,0,0,0)); plot(g)

# what does `ifelse` do?

nodes$name=="R2-D2"

## [1] TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE

## [12] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE

ifelse(nodes$name=="R2-D2", "yes", "no")

## [1] "yes" "no" "no" "no" "no" "no" "no" "no" "no" "no" "no"

## [12] "no" "no" "no" "no" "no" "no" "no" "no" "no" "no" "no"

ifelse(grepl("R", nodes$name), "yes", "no")

## [1] "yes" "no" "no" "no" "yes" "no" "no" "no" "yes" "no" "no"

## [12] "no" "yes" "no" "yes" "no" "no" "yes" "no" "yes" "yes" "no"

We can also change the colors of each node based on what side they’re in (dark side or light side).

# create vectors with characters in each side

dark\_side <- c("DARTH VADER", "MOTTI", "TARKIN")

light\_side <- c("R2-D2", "CHEWBACCA", "C-3PO", "LUKE", "CAMIE", "BIGGS",

"LEIA", "BERU", "OWEN", "OBI-WAN", "HAN", "DODONNA",

"GOLD LEADER", "WEDGE", "RED LEADER", "RED TEN", "GOLD FIVE")

other <- c("GREEDO", "JABBA")

# node we'll create a new color variable as a node property

V(g)$color <- NA

V(g)$color[V(g)$name %in% dark\_side] <- "red"

V(g)$color[V(g)$name %in% light\_side] <- "gold"

V(g)$color[V(g)$name %in% other] <- "grey20"

vertex\_attr(g)

## $name

## [1] "R2-D2" "CHEWBACCA" "C-3PO" "LUKE" "DARTH VADER"

## [6] "CAMIE" "BIGGS" "LEIA" "BERU" "OWEN"

## [11] "OBI-WAN" "MOTTI" "TARKIN" "HAN" "GREEDO"

## [16] "JABBA" "DODONNA" "GOLD LEADER" "WEDGE" "RED LEADER"

## [21] "RED TEN" "GOLD FIVE"

##

## $id

## [1] 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21

##

## $size

## [1] 18.648092 19.572539 19.635532 22.439250 12.591581 8.545177 13.556229

## [8] 19.310150 11.788898 11.317766 18.567281 8.545177 12.210340 20.528107

## [15] 3.000000 3.000000 9.437752 9.437752 11.788898 13.259797 5.772589

## [22] -Inf

##

## $label

## [1] "R2-D2" "CHEWBACCA" "C-3PO" "LUKE" "DARTH VADER"

## [6] NA "BIGGS" "LEIA" NA NA

## [11] "OBI-WAN" NA "TARKIN" "HAN" NA

## [16] NA NA NA NA "RED LEADER"

## [21] NA NA

##

## $color

## [1] "gold" "gold" "gold" "gold" "red" "gold" "gold"

## [8] "gold" "gold" "gold" "gold" "red" "red" "gold"

## [15] "grey20" "grey20" "gold" "gold" "gold" "gold" "gold"

## [22] "gold"

par(mar=c(0,0,0,0)); plot(g)

# what does %in% do?

1 %in% c(1,2,3,4)

## [1] TRUE

1 %in% c(2,3,4)

## [1] FALSE

If we want to indicate what the colors correspond to, we can add a legend.

par(mar=c(0,0,0,0)); plot(g)

legend(x=.75, y=.75, legend=c("Dark side", "Light side", "Other"),

pch=21, pt.bg=c("red", "gold", "grey20"), pt.cex=2, bty="n")

Edge properties can also be modified. For example, here the width of each edge is a function of the log number of scenes those two characters appear together.

E(g)$width <- log(E(g)$weight) + 1

edge\_attr(g)

## $weight

## [1] 17 13 6 5 5 3 1 7 5 16 19 11 1 1 2 2 4 1 3 3 2 3 18

## [24] 2 6 17 1 19 6 1 2 1 7 9 26 1 1 6 1 1 13 1 1 1 1 1

## [47] 1 2 1 1 3 3 1 1 3 1 2 1 1 1

##

## $width

## [1] 3.833213 3.564949 2.791759 2.609438 2.609438 2.098612 1.000000

## [8] 2.945910 2.609438 3.772589 3.944439 3.397895 1.000000 1.000000

## [15] 1.693147 1.693147 2.386294 1.000000 2.098612 2.098612 1.693147

## [22] 2.098612 3.890372 1.693147 2.791759 3.833213 1.000000 3.944439

## [29] 2.791759 1.000000 1.693147 1.000000 2.945910 3.197225 4.258097

## [36] 1.000000 1.000000 2.791759 1.000000 1.000000 3.564949 1.000000

## [43] 1.000000 1.000000 1.000000 1.000000 1.000000 1.693147 1.000000

## [50] 1.000000 2.098612 2.098612 1.000000 1.000000 2.098612 1.000000

## [57] 1.693147 1.000000 1.000000 1.000000

par(mar=c(0,0,0,0)); plot(g)

Up to now, each time we run the plot function, the nodes appear to be in a different location. Why? Because it’s running a probabilistic function trying to locate them in the optimal way possible.

However, we can also specify the **layout** for the plot; that is, the (x,y) coordinates where each node will be placed. igraph has a few different layouts built-in, that will use different algorithms to find an optimal distribution of nodes. The following code illustrates some of these:

par(mfrow=c(2, 3), mar=c(0,0,1,0))

plot(g, layout=layout\_randomly, main="Random")

plot(g, layout=layout\_in\_circle, main="Circle")

plot(g, layout=layout\_as\_star, main="Star")

plot(g, layout=layout\_as\_tree, main="Tree")

plot(g, layout=layout\_on\_grid, main="Grid")

plot(g, layout=layout\_with\_fr, main="Force-directed")

Note that each of these is actually just a matrix of (x,y) locations for each node.

l <- layout\_randomly(g)

str(l)

## num [1:22, 1:2] -0.8468 -0.2524 0.1666 -0.0292 0.9343 ...

The most popular layouts are [force-directed](https://en.wikipedia.org/wiki/Force-directed_graph_drawing). These algorithms, such as Fruchterman-Reingold, try to position the nodes so that the edges have similar length and there are as few crossing edges as possible. The idea is to generate “clean” layouts, where nodes that are closer to each other share more connections in common that those that are located further apart. Note that this is a non-deterministic algorithm: choosing a different seed will generate different layouts.

par(mfrow=c(1,2))

set.seed(777)

fr <- layout\_with\_fr(g, niter=1000)

par(mar=c(0,0,0,0)); plot(g, layout=fr)

set.seed(666)

fr <- layout\_with\_fr(g, niter=1000)

par(mar=c(0,0,0,0)); plot(g, layout=fr)