Social Network Analysis
With igraph & R

Ofrit Lesser

ofrit.lesser@gmail.com

December 11th, 2014

אוניברסיטת בן-גוריון בנגב
Ben-Gurion University of the Negev

Outline

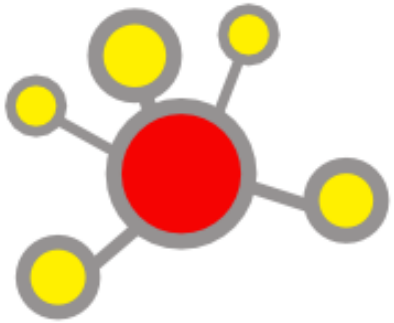
- The igraph R package
- Basic graph concepts
- What can you do with igraph?
  - Construction
  - Attributes
  - Centrality measure
  - Community detection
  - Plotting, and more…

The igraph software package

- igraph - An open source library for the analysis of large networks.
- Free for academic and commercial use (GPL).
- State of the art data structures and algorithms, works well with large graphs.
- Core functionality is implemented as a C library.
- Can be programmed in GNU R, Python and C/C++.

- For details & documentation see

  http://igraph.org/



Scalability - How LARGE?

- Well, it depends what you want to calculate.
- For graph creation and manipulation, it is enough if it fits into the memory.

- igraph (typically) needs 32 bytes per edge and 16 bytes per vertex.
- A graph with one million vertices and ten million edges needs about 320 Mbytes.

Installation

- http://igraph.org/r/

Installation

igraph is on CRAN and can be installed from within R:

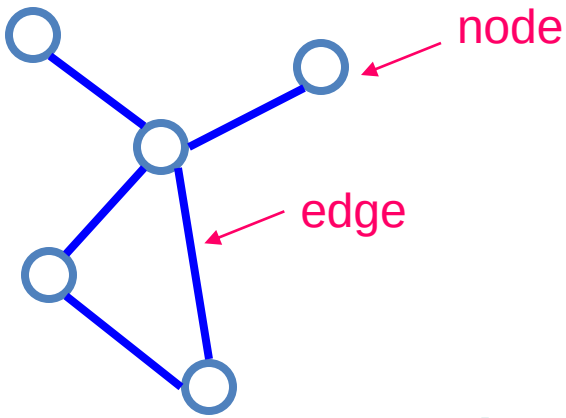
```
## Download and install the package
install.packages("igraph")

## Load package
library(igraph)
```

What are networks?

- Networks are sets of nodes connected by edges.

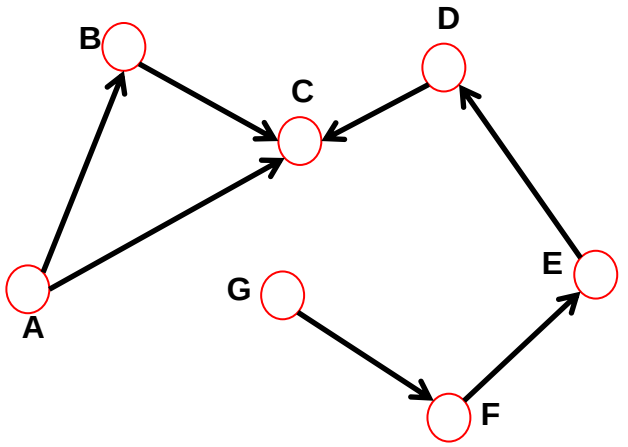
“Network” ≡ “Graph”



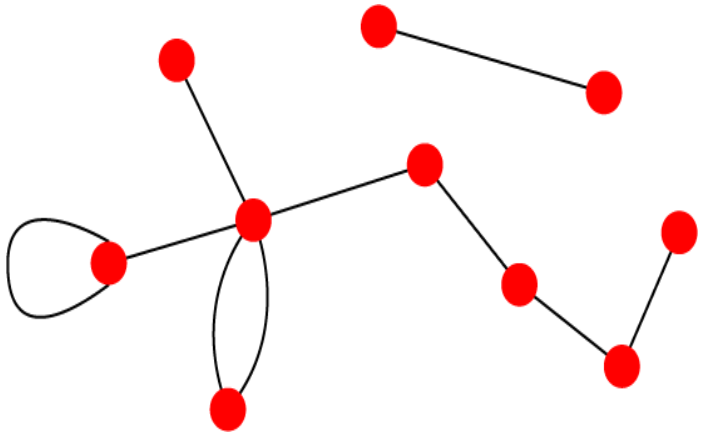
points	**lines**	
vertices	edges, arcs	math
nodes	edges, links	computer science
sites	bonds	physics
actors	ties, relations	sociology

Network elements: edges

- Directed links (edges)
  - URLs on the www
  - phone calls
  - metabolic reactions

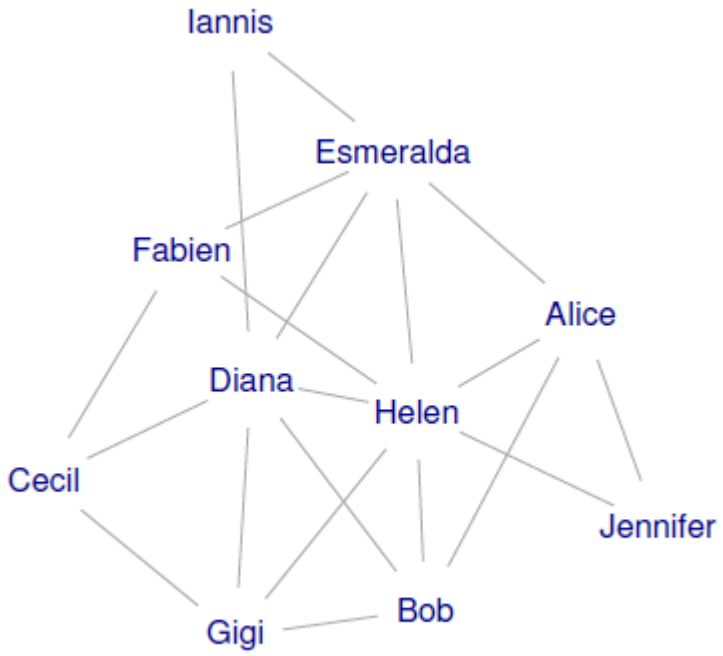


- Undirected links (edges
  - Co-authorship links
  - Actor network
  - protein interactions



Graph representation

- There is no best format for network data, everything depends on what kind of questions one wants to ask.



Graph representation

- Edge list. Readable, but not too efficient.

Alice	Bob
Bob	Diana
Cecil	Diana
Alice	Esmeralda
Diana	Esmeralda
Cecil	Fabien
Esmeralda	Fabien
Bob	Gigi
Cecil	Gigi
Diana	Gigi
Alice	Helen
Bob	Helen
Diana	Helen
Esmeralda	Helen
Fabien	Helen
Gigi	Helen
Diana	Iannis
Esmeralda	Iannis
Alice	Jennifer
Helen	Jennifer

Graph representation

- Adjacency matrix
- Good for questions like: is ’Alice’ connected to ’Bob’?

	Alice	Bob	Cecil	Diana	Esmeralda	Fabien	Gigi	Helen	Iannis	Jennifer
Alice	0	1	0	0	1	0	0	1	0	1
Bob	1	0	0	1	0	0	1	1	0	0
Cecil	0	0	0	1	0	1	1	0	0	0
Diana	0	1	1	0	1	0	1	1	1	0
Esmeralda	1	0	0	1	0	1	0	1	1	0
Fabien	0	0	1	0	1	0	0	1	0	0
Gigi	0	1	1	1	0	0	0	1	0	0
Helen	1	1	0	1	1	1	1	0	0	1
Iannis	0	0	0	1	1	0	0	0	0	0
Jennifer	1	0	0	0	0	0	0	1	0	0

Graph representation

- Adjacency lists. quickly retrieve all neighbors for a node.

Alice	**Bob, Esmeralda, Helen, Jennifer**
Bob	**Alice, Diana, Gigi, Helen**
Cecil	**Diana, Fabien, Gigi**
Diana	**Bob, Cecil, Esmeralda, Gigi, Helen, Iannis**
Esmeralda	**Alice, Diana, Fabien, Helen, Iannis**
Fabien	**Cecil, Esmeralda, Helen**
Gigi	**Bob, Cecil, Diana, Helen**
Helen	**Alice, Bob, Diana, Esmeralda, Fabien, Gigi, Jennifer**
Iannis	Diana, Esmeralda
Jennifer	Alice, Helen

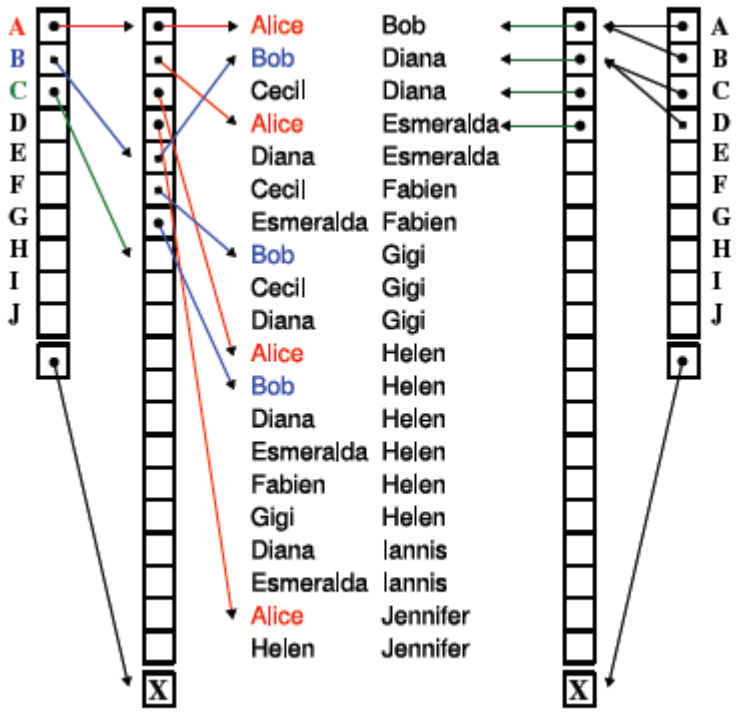
Graph representation - igraph

- Flat data structures, indexed edge lists.
- Easy to handle, good for large sparse graphs.

Alice	Bob
Bob	Diana
Cecil	Diana
Alice	Esmeralda
Diana	Esmeralda
Cecil	Fabien
Esmeralda	Fabien
Bob	Gigi
Cecil	Gigi
Diana	Gigi
Alice	Helen
Bob	Helen
Diana	Helen
Esmeralda	Helen
Fabien	Helen
Gigi	Helen
Diana	Iannis
Esmeralda	Iannis
Alice	Jennifer
Helen	Jennifer

Graph representation - igraph

- Flat data structures, indexed edge lists.
- Easy to handle, good for many kind of questions.



Vertex and edge ids

- Vertices are numbered from 1 to |V| (used to be 0 to |V|-1).

  V = {A,B,C,D,E}

  E = ((A,B), (A,C), (B,C), (C,E)).

  A = 1,B = 2,C = 3,D = 4,E = 5.

```
g <- graph( c(1,2, 1,3, 2,3, 3,5), n=5 )
g
```

```
## IGRAPH D--- 5 4 --
```

```
str(g)
```

```
## IGRAPH D--- 5 4 --
## + edges:
## [1] 1->2 1->3 2->3 3->5
```

Creating igraph graphs

- igraph objects
- summary(), is.igraph()

```
is.igraph(g)
```

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

```
summary(g)
```

```
## IGRAPH D--- 5 4 --
```

igraph graphs operations

- Basic manipulations

```
is.directed(g)
```

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

```
vcount(g)
```

```
## [1] 5
```

```
ecount(g)
```

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

Vertex and edge sequences

- Vertex and edge sequences and iterators

```
V(g)
```

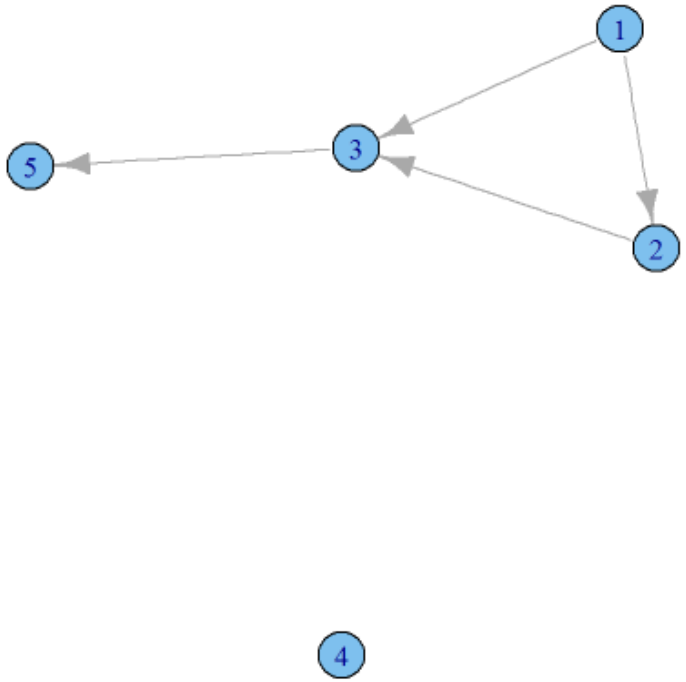
```
## Vertex sequence:
## [1] 1 2 3 4 5
```

```
E(g)
```

```
## Edge sequence:
##
## [1] 1 -> 2
## [2] 1 -> 3
## [3] 2 -> 3
## [4] 3 -> 5
```

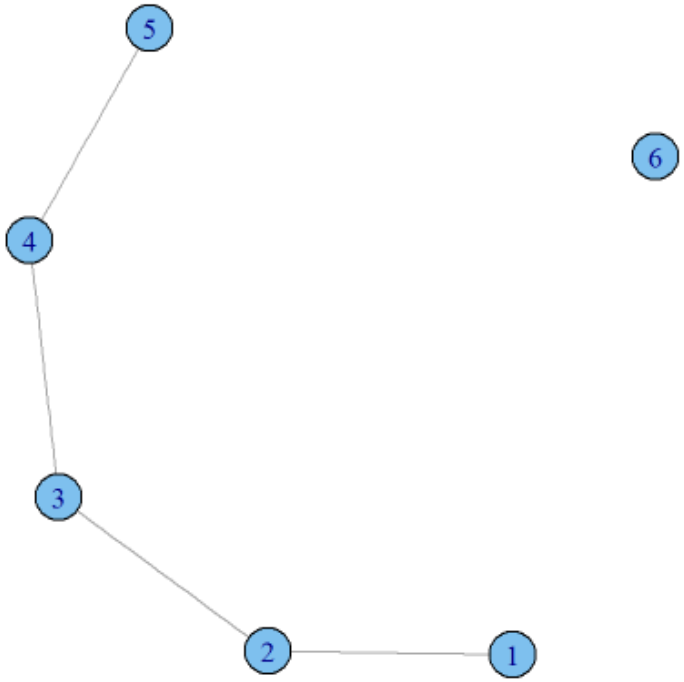
Visualizing graphs

```
g <- graph( c(1,2, 1,3, 2,3, 3,5), n=5 )
plot(g)
```



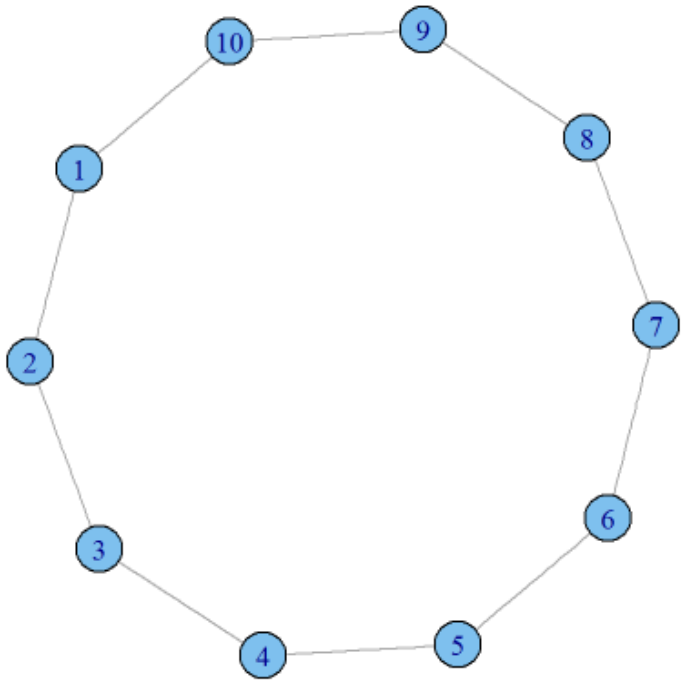
Other graph types - undirected

```
g_nd <- graph( c(1,2, 2,3, 3,4, 4,5), n=6, directed=FALSE )
plot(g_nd)
```



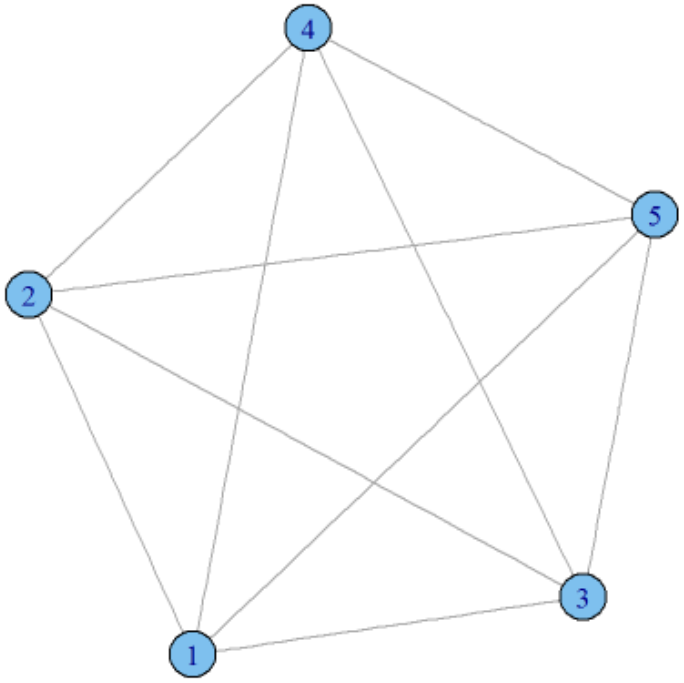
Ring graph

```
g_r <- graph.ring(10)
plot(g_r)
```



Complete graph

```
g_f <- graph.full(5)
plot(g_f)
```



Visualization layouts

- Possible to force directed layouts. Examples:

  plot(g, layout=layout.fruchterman.reingold)

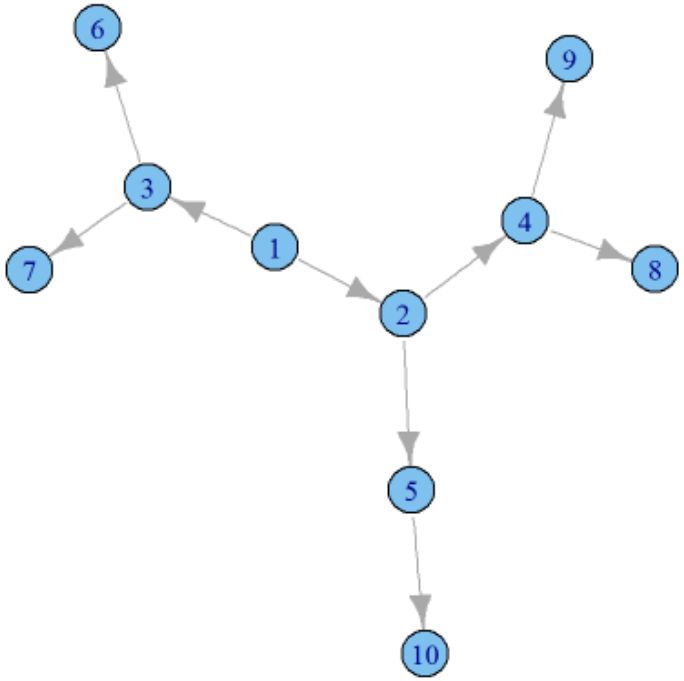
  plot(g, layout=layout.graphopt)

  plot(g, layout=layout.kamada.kawai)

- More on drawing graphs, see: http://igraph.org/r/doc/plot.common.html

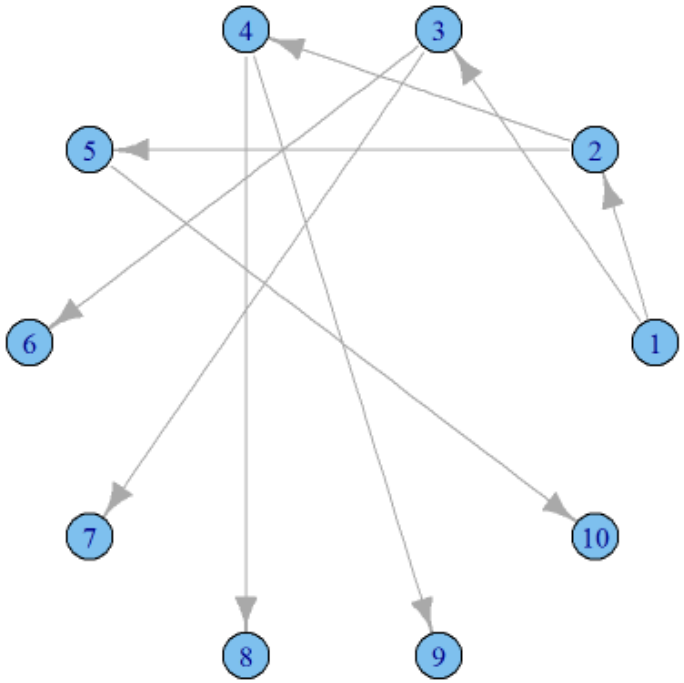
Tree graph

```
g_t <- graph.tree(10, 2)
plot(g_t)
```

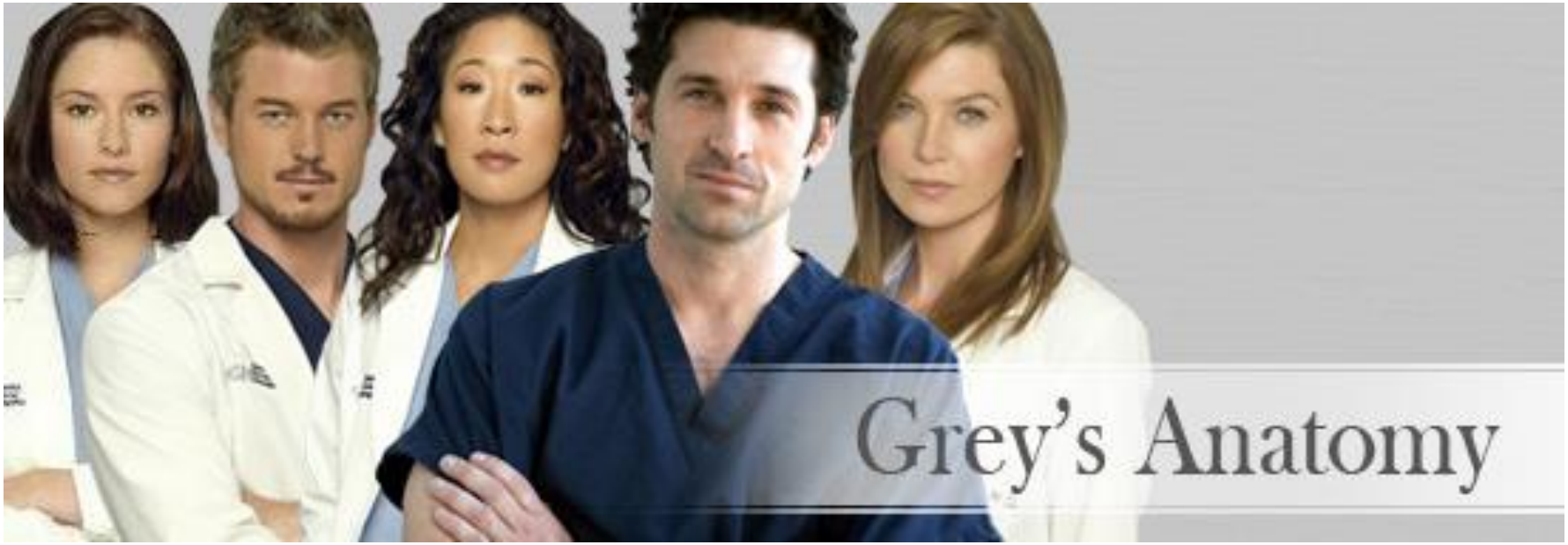


Tree graph with circular layout

```
plot(g_t, layout=layout.circle)
```



Grey's Anatomy Network of Romance Example



Inspired by the post "Grey's Anatomy Network of Sexual Relations", by Gary Weissman. Available online at http://www.babelgraph.org/wp/?p=1

Create Grey's Anatomy Network

- Creating graphs using graph.data.frame()

```
library(igraph)
ga.data <- read.csv('ga_edgelist.csv', header=TRUE)
head(ga.data)
```

```
##    from     to
## 1  lexi  sloan
## 2  lexi  karev
## 3  owen   yang
## 4  owen altman
## 5 sloan torres
## 6 sloan altman
```

```
g <- graph.data.frame(ga.data, directed=FALSE)
summary(g)
```

```
## IGRAPH UN-- 32 34 --
## attr: name (v/c)
```

Vertex and edge sequences

- List of nodes' names

```
V(g)$name
```

```
##  [1] "lexi"         "owen"       "sloan"         "torres"
##  [5] "derek"        "karev"      "o'malley"      "yang"
##  [9] "grey"         "chief"      "ellis grey"    "susan grey"
## [13] "bailey"       "izzie"      "altman"        "arizona"
## [17] "colin"        "preston"    "kepner"        "addison"
## [21] "nancy"        "olivia"     "mrs. seabury"  "adele"
## [25] "thatch grey"  "tucker"     "hank"          "denny"
## [29] "finn"         "steve"      "ben"           "avery"
```

Vertex and edge sequences

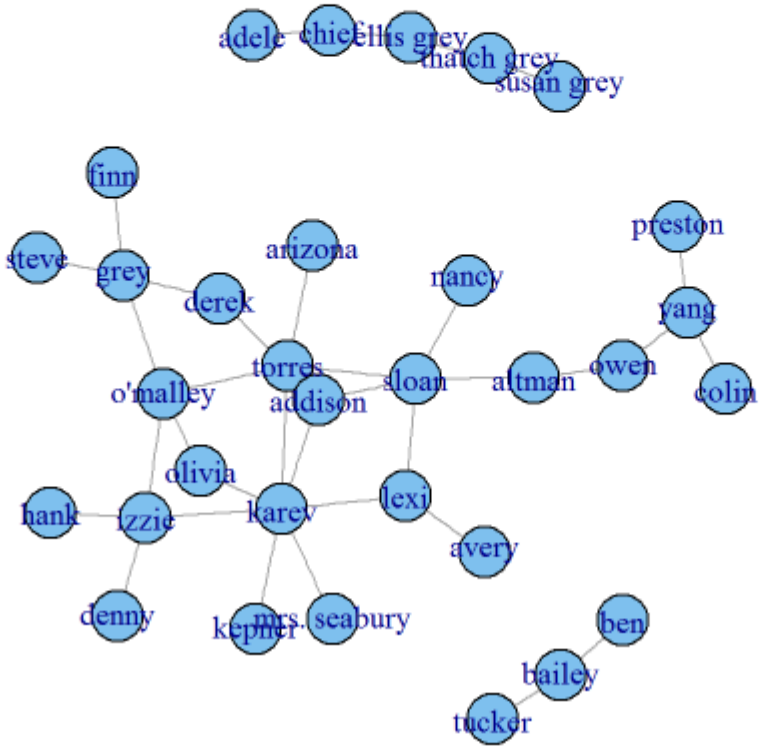
- And a list of edges…

```
head(E(g))
```

```
## Edge sequence:
##
## [1] sloan  -- lexi
## [2] karev  -- lexi
## [3] yang   -- owen
## [4] altman -- owen
## [5] torres -- sloan
## [6] altman -- sloan
```

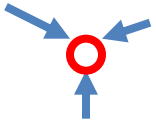
Choose a layout scheme and plot

```
g$layout <- layout.fruchterman.reingold(g)
plot(g)
```

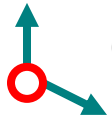


Node Degree

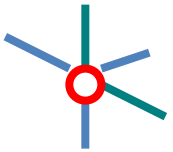
- Node network properties
  - from immediate connections
    - indegree
      how many directed edges (arcs) are incident on a node
    - outdegree
      how many directed edges (arcs) originate at a node
    - degree (in or out)
      number of edges incident on a node



indegree=3



outdegree=2



degree=5

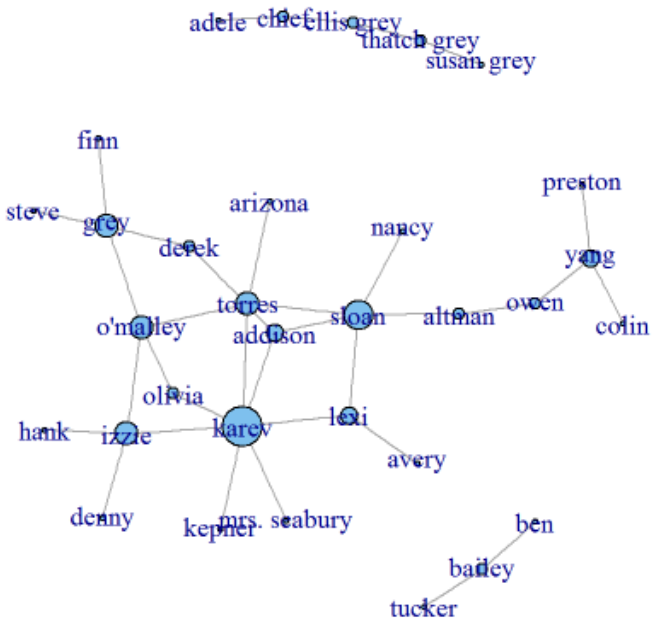
Calculate degree centrality

```
degr.score <- degree(g)
degr.score
```

```
##         lexi         owen        sloan       torres        derek
##            3            2            5            4            2
##        karev     o'malley         yang         grey        chief
##            7            4            3            4            2
##   ellis grey   susan grey       bailey       izzie       altman
##            2            1            2            4            2
##      arizona        colin      preston      kepner      addison
##            1            1            1            1            3
##        nancy       olivia mrs. seabury       adele  thatch grey
##            1            2            1            1            2
##       tucker         hank        denny        finn        steve
##            1            1            1            1            1
##          ben        avery
##            1            1
```

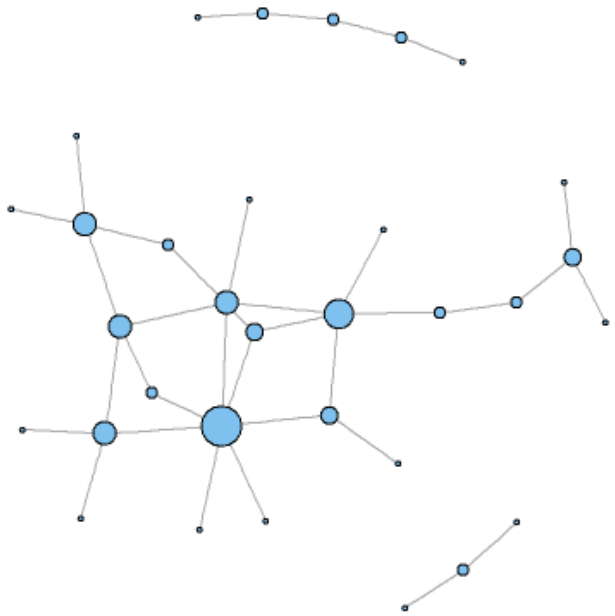
Nodes' size scales with degree

```
V(g)$size <- degree(g) * 2 # multiply by 2 for scale
plot(g)
```



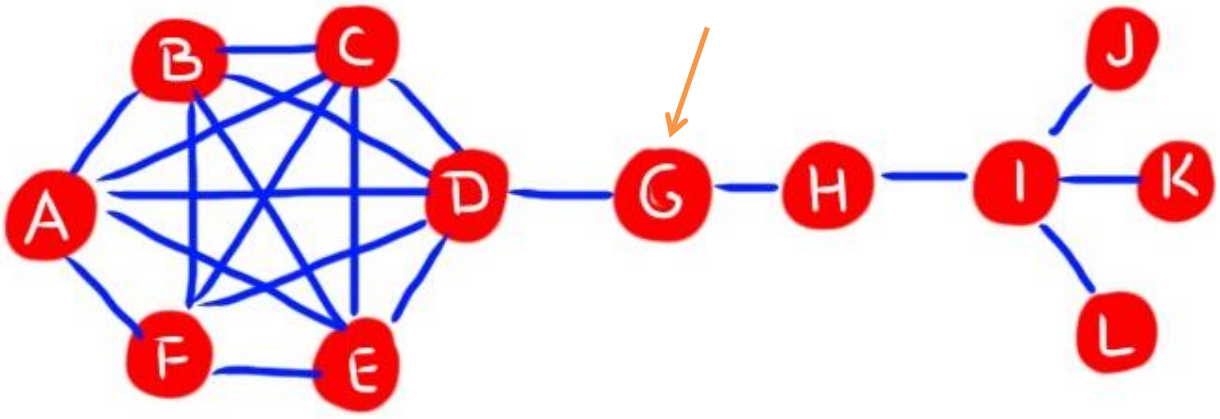
Remove nodes` labels

```
V(g)$label <- NA # remove labels for now
plot(g)
```



Centrality:
importance based on network position

Degree is only part of the story…



Closeness: definition

Closeness centrality of a vertex - the inverse of the average shortest paths to/from all the other vertices in the graph.

Closeness Centrality:

$$C_c(i) = \frac{1}{\sum_{j=1}^{N} d(i,j)}$$

Closeness Centrality normalized:

$$C_c(i) = \frac{N-1}{\sum_{j=1}^{N} d(i,j)}$$

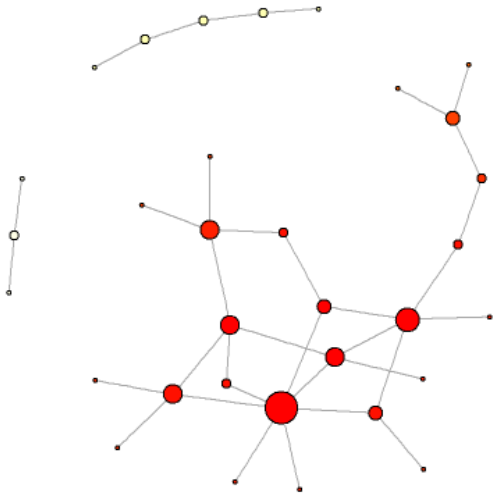
Closeness & color scheme

- Calculate closeness centrality.
- Rescale to create a color scheme to visualize the relative differences.

```
clo <- closeness(g)
# rescale values to match the elements of a color vector
clo.score <- round( (clo - min(clo)) * length(clo) / max(clo) ) + 1
# create color vector, use rev to make red "hot"
clo.colors <- rev(heat.colors(max(clo.score)))
V(g)$color <- clo.colors[ clo.score ]
plot(g)
```

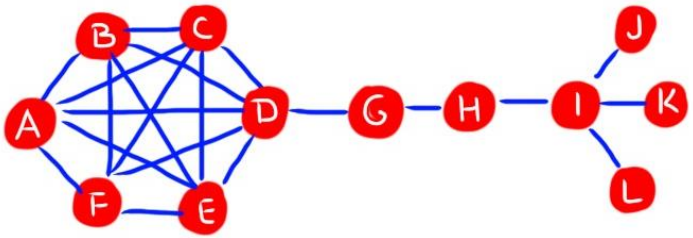
Closeness centrality

```
clo <- closeness(g)
# rescale values to match the elements of a color vector
clo.score <- round( (clo - min(clo)) * length(clo) / max(clo) ) + 1
# create color vector, use rev to make red "hot"
clo.colors <- rev(heat.colors(max(clo.score)))
V(g)$color <- clo.colors[ clo.score ]
plot(g)
```



Betweeness: definition

Intuition: how many pairs of individuals would have to go through you in order to reach one another in the minimum number of hops?

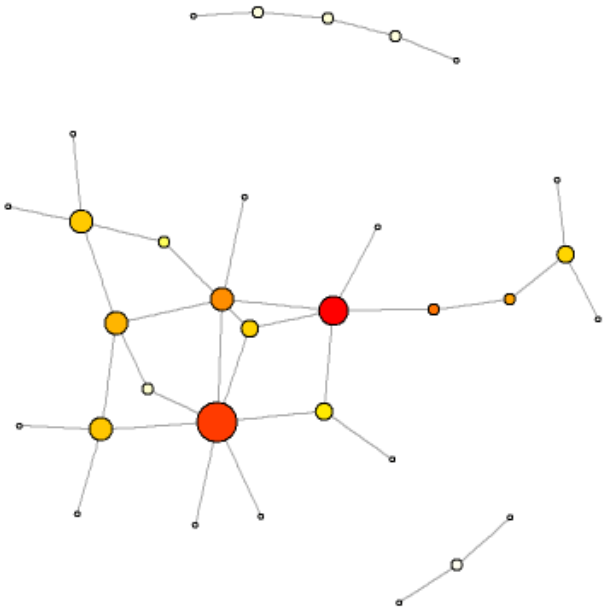


Betweeness Centrality:

$$C_B(i) = \sum_{j<k} g_{jk}(i) / g_{jk}$$

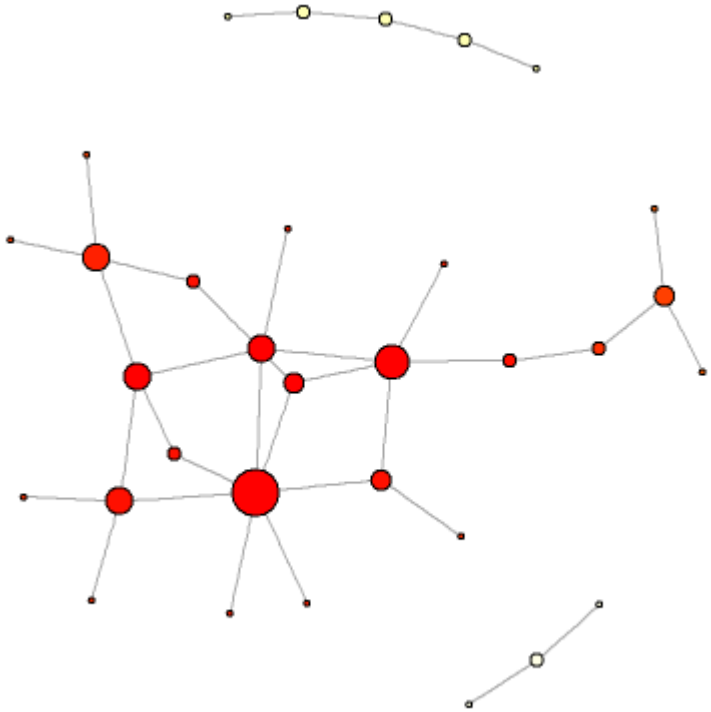
Betweeness centrality

```
btw <- betweenness(g)
btw.score <- round(btw) + 1
btw.colors <- rev(heat.colors(max(btw.score)))
V(g)$color <- btw.colors[ btw.score ]
plot(g)
```

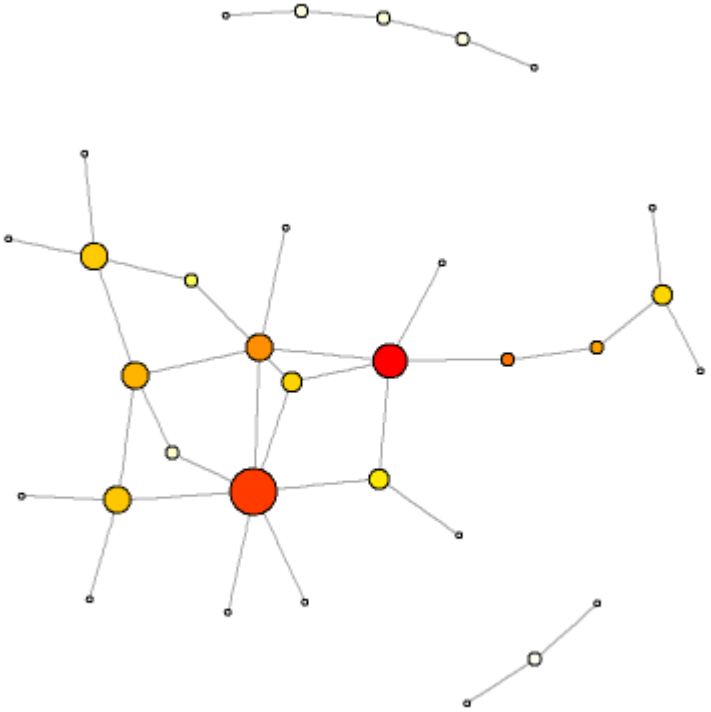


Closeness vs. betweeness

Closeness

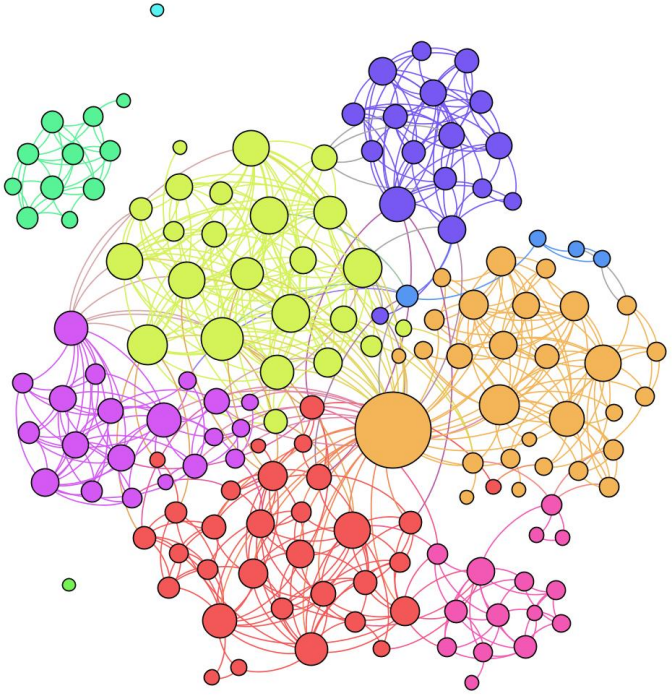


Betweeness



Community structure in networks

- Social networks tend to be highly clustered.
- There are several algorithms that uncover these clusters.



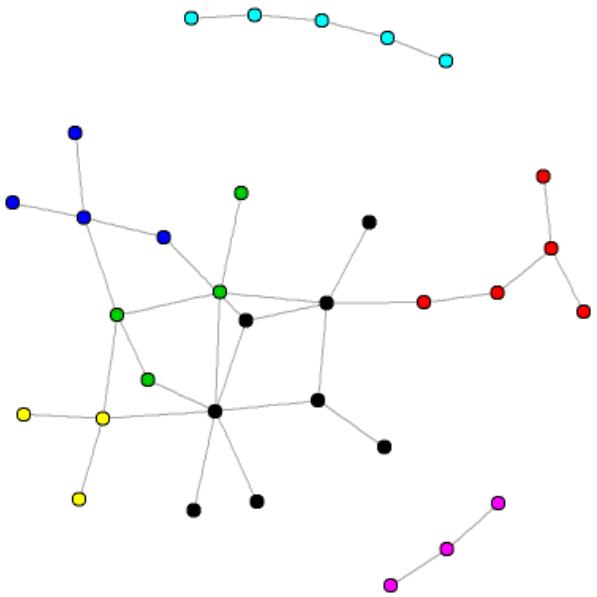
Community detection

- Here we use the implementation of the Girvan-Newman algorithm to detect the underlying community structure of the graph.

```
gnc <- edge.betweenness.community(g, directed=FALSE)
V(g)$color <- gnc$membership
V(g)$size <- 5 # Set same size to all nodes
plot(g)
```

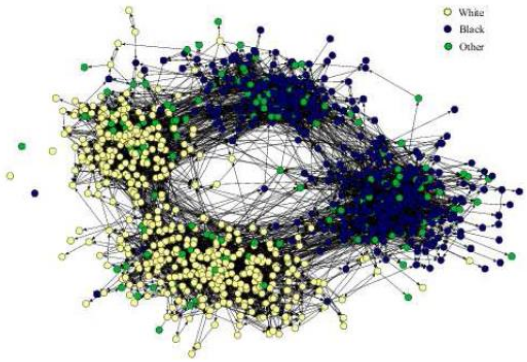
Girvan-Newman clustering

```
gnc <- edge.betweenness.community(g, directed=FALSE)
V(g)$color <- gnc$membership
V(g)$size <- 5 # Set same size to all nodes
plot(g)
```



Network elements may have attributes

- Example for node attributes:
  - geographical location
  - Ethnicity
  - musical tastes…



- Example for edge attributes:
  - weight (e.g. frequency of communication)
  - ranking (best friend, second best friend…)
  - type (friend, relative, co-worker)

Gender attribute - example

- Nodes csv file

```
ga.vrtx <- read.csv('ga_actors.csv', header=TRUE, stringsAsFactors=FALSE)
head(ga.vrtx)
```

```
##       name gender
## 1  addison      F
## 2    adele      F
## 3   altman      F
## 4  arizona      F
## 5    avery      M
## 6   bailey      F
```

Gender attribute - example

```
ga.data <- read.csv('ga_edgelist.csv', header=TRUE, stringsAsFactors=FALSE)
ga.vrtx <- read.csv('ga_actors.csv', header=TRUE, stringsAsFactors=FALSE)
g <- graph.data.frame(ga.data, vertices=ga.vrtx, directed=FALSE)
g
```

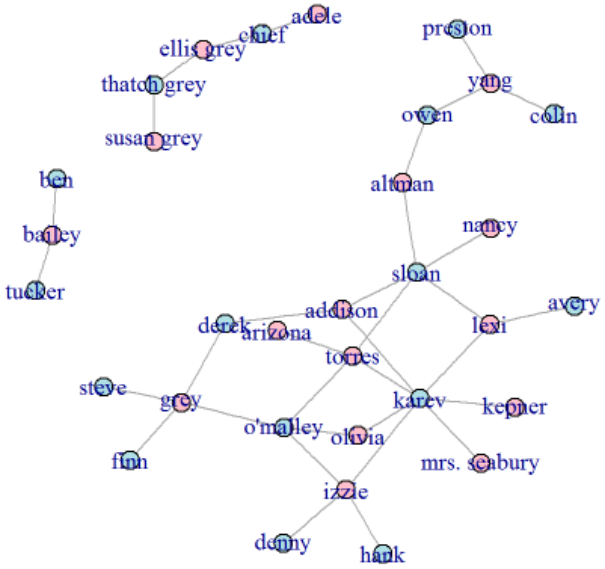
```
## IGRAPH UN-- 32 34 --
## + attr: name (v/c), gender (v/c)
```

```
V(g)$gender
```

```
##  [1] "F" "F" "F" "F" "M" "F" "M" "M" "M" "M" "M" "F" "M" "F" "M" "F" "M"
## [18] "F" "F" "F" "F" "F" "M" "M" "M" "M" "M" "F" "M" "F" "M" "F"
```

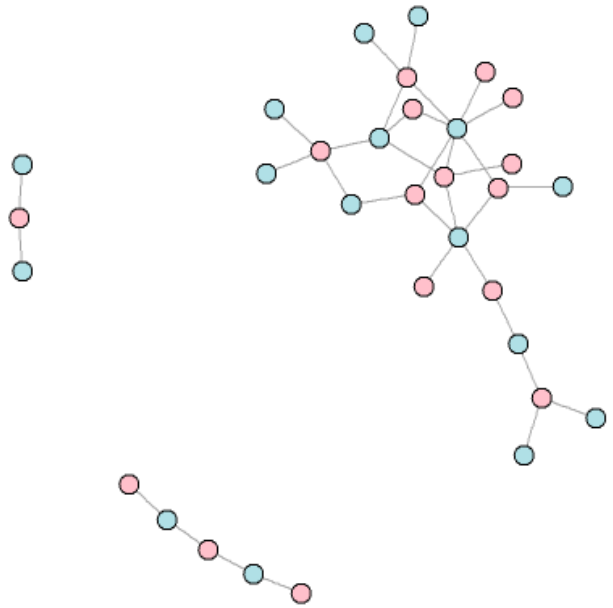
Color nodes by gender

```
V(g)$size <- 7 # Set size to all nodes
V(g)$color <- "powderblue"
females <- which(V(g)$gender == "F")
V(g)$color[females] <- "pink"
plot(g)
```



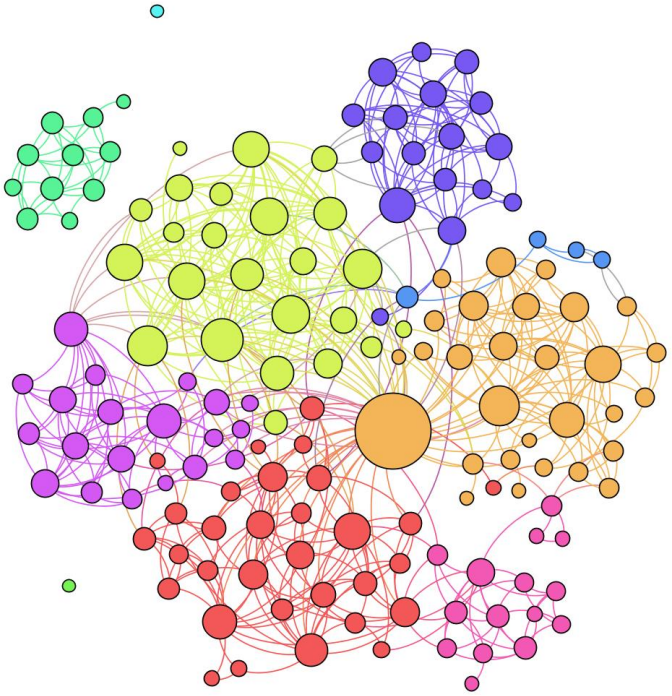
Remove labels & layout differently

```
V(g)$label <- NA # remove labels for now
g$layout <- layout.kamada.kawai(g)
plot(g)
```



Connected components

- Social networks tend to be connected graphs, such that almost every node is reachable from almost every other node.



Connected components

```
no.clusters(g)
```

```
## [1] 3
```

```
cl <- clusters(g)
which.max(cl$csize)
```

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

```
cl$membership == which.max(cl$csize)
```

```
##  [1]  TRUE FALSE  TRUE  TRUE  TRUE FALSE FALSE FALSE  TRUE  TRUE  TRUE
## [12] FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [23]  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE  TRUE FALSE  TRUE
```

```
cl$membership
```

```
##  [1] 1 2 1 1 1 3 3 2 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 1 3 1
```

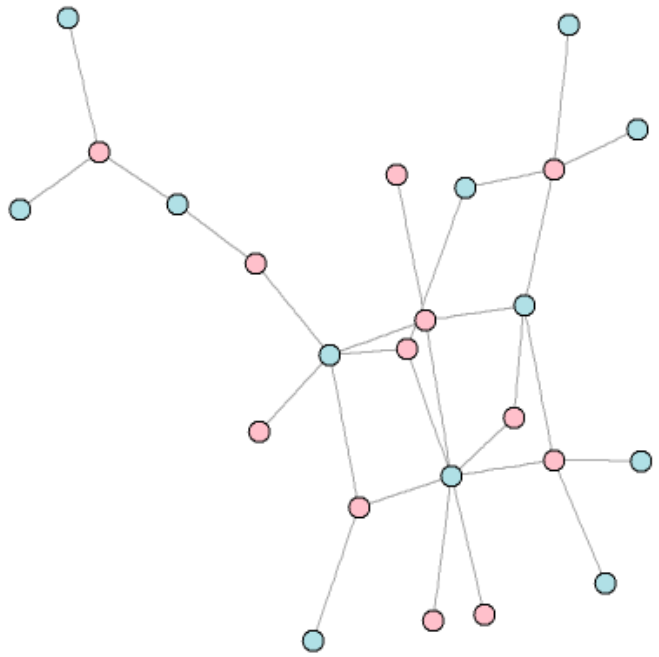
Keep only the Giant Component

```
cl <- clusters(g)
to.keep <- which(cl$membership == which.max(cl$csize))
g_gc <- induced.subgraph(g, to.keep)
summary(g_gc)
```

```
## IGRAPH UN-- 24 28 --
## attr: layout (g/n), name (v/c), gender (v/c), size (v/n), color
##   (v/c), label (v/l)
```

Print the giant component only

```
g_gc$layout <- layout.kamada.kawai(g_gc)
plot(g_gc)
```



Questions?


Thank you!