Statistics, Visualization and More Using R

R-package igraph

Nina Großegesse, Christopher Höhn, Marlene Holzleitner

06.05.2019

Universität Salzburg

Table of contents

Introduction

About igraph

- Developers: Gábor Csárdi and Tamás Nepusz
- collection of R functions to create, manipulate and visualize graphs
- Usage: network analysis

Examples:

social networks, airline networks, road networks, brain networks, email networks

- goals:
  - optimized for running time
  - easy use
- igraphdata R package: to be used with igraph R package

Short graph theory

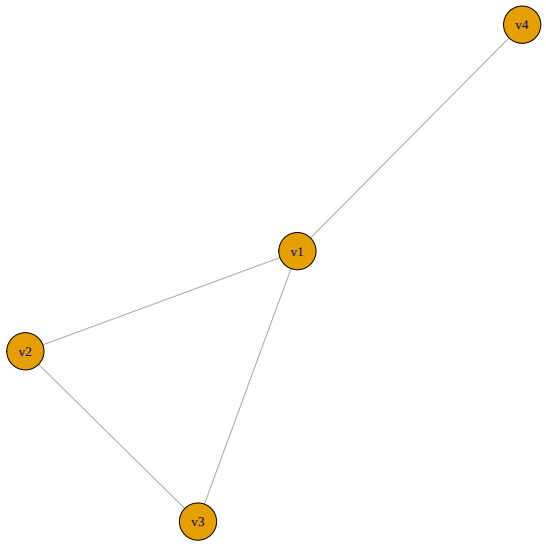
Undirected graph

Definition

An undirected graph is an ordered pair $G = (V, E)$ comprising

- V a set of vertices
- $E \subseteq \{\{x, y\} | (x, y) \in V^2\}$ a set of edges, which are unordered pairs of vertices

Example



$$V = \{v_1, v_2, v_3, v_4\}$$

$$E = \{\{v_1, v_2\}, \{v_1, v_3\}, \{v_1, v_4\}, \{v_2, v_3\}\}$$

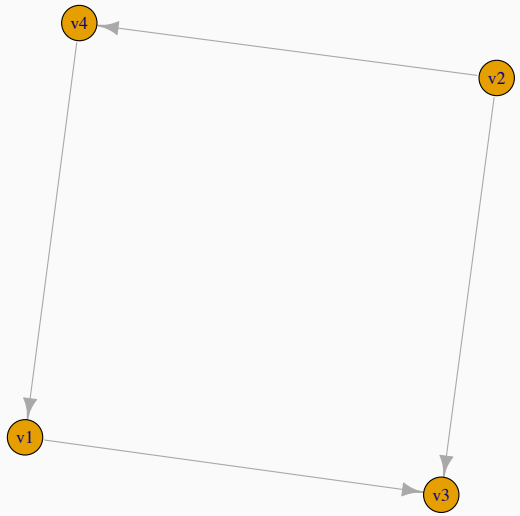
Directed graph

Definition

A directed graph is an ordered pair $G = (V, E)$ comprising

- V a set of vertices
- $E \subseteq \{(x, y) \in V^2\} = V^2$ a set of edges, which are ordered pairs of vertices

Example



$$V = \{v_1, v_2, v_3, v_4\}$$

$$E = \{(v_1, v_3), (v_2, v_3), (v_2, v_4), (v_4, v_1)\}$$

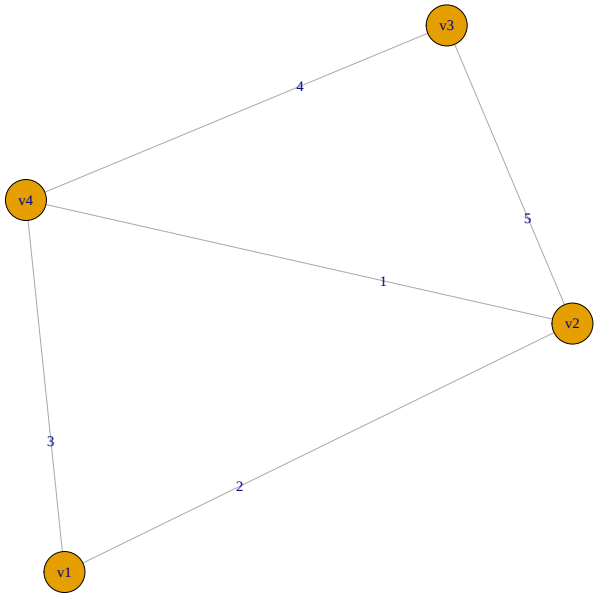
Weighted graph

Definition

Let $G = (V, E)$ be a graph. We can assign a weight to the edges:

$$w : E \to \mathbb{R}^+$$

Example



$$V = \{v_1, v_2, v_3, v_4\}$$

$$E = \{\{v_1, v_2\}, \{v_1, v_4\}, \{v_2, v_4\}, \{v_2, v_3\}, \{v_3, v_4\}\}$$

$$w(\{v_1, v_2\}) = 2,\ w(\{v_1, v_4\}) = 3,$$
$$w(\{v_2, v_4\}) = 1,\ w(\{v_2, v_3\}) = 5,$$
$$w(\{v_3, v_4\}) = 4$$

Definitions

Definition (order of a graph)

The order of a graph $G = (V, E)$ is the number of vertices. It is denoted by $|V|$.

Definitions

Definition (order of a graph)

The order of a graph $G = (V, E)$ is the number of vertices. It is denoted by $|V|$.

Definition (size of a graph)

The size of a graph $G = (V, E)$ is the number of edges. It is denoted by $|E|$.

Definitions

Definition (order of a graph)

The order of a graph $G = (V, E)$ is the number of vertices. It is denoted by $|V|$.

Definition (size of a graph)

The size of a graph $G = (V, E)$ is the number of edges. It is denoted by $|E|$.

Definition (adjacent vertices)

Two vertices v_1 and v_2 of an undirected graph $G = (V, E)$ are called adjacent or neighbored, if $\{v_1, v_2\} \in E$.

Definitions

Definition (order of a graph)

The order of a graph $G = (V, E)$ is the number of vertices. It is denoted by $|V|$.

Definition (size of a graph)

The size of a graph $G = (V, E)$ is the number of edges. It is denoted by $|E|$.

Definition (adjacent vertices)

Two vertices v_1 and v_2 of an undirected graph $G = (V, E)$ are called adjacent or neighbored, if $\{v_1, v_2\} \in E$.

Definition (incident edges)

A vertex v and an edge $\{v_1, v_2\}$ (or (v_1, v_2), resp.) of a graph $G = (V, E)$ are called incident, if $v \in \{v_1, v_2\}$.

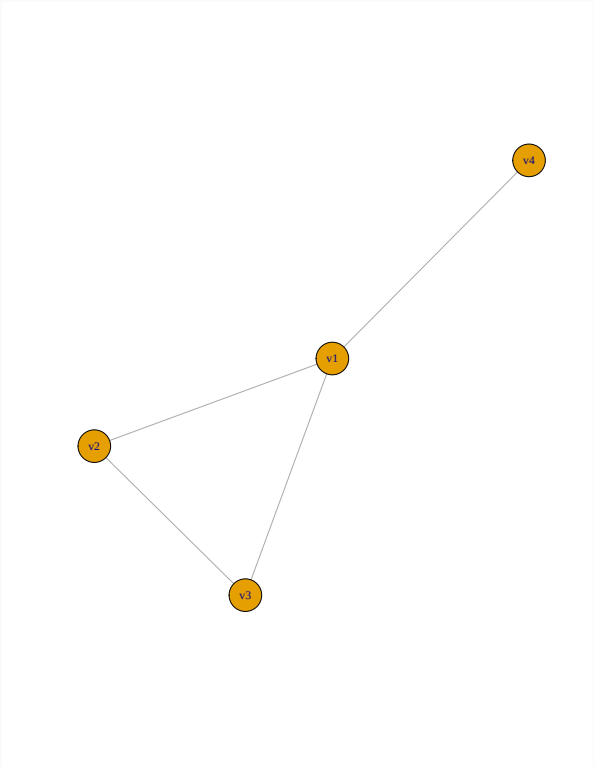
Basics

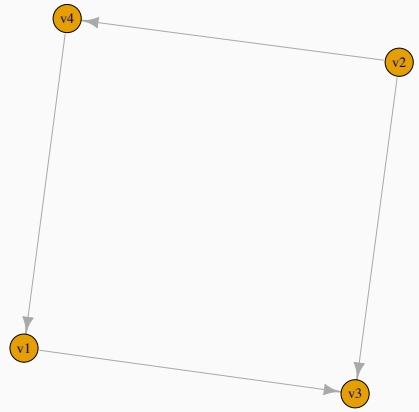
Generating igraph objects

- useful for small graphs: `graph_from_literal(...)`

Examples:

```
graph_from_literal(v1--v2:v3:v4, v2--v3)
graph_from_literal(v4--+v1--+v3, v3+--v2--+v4)
```



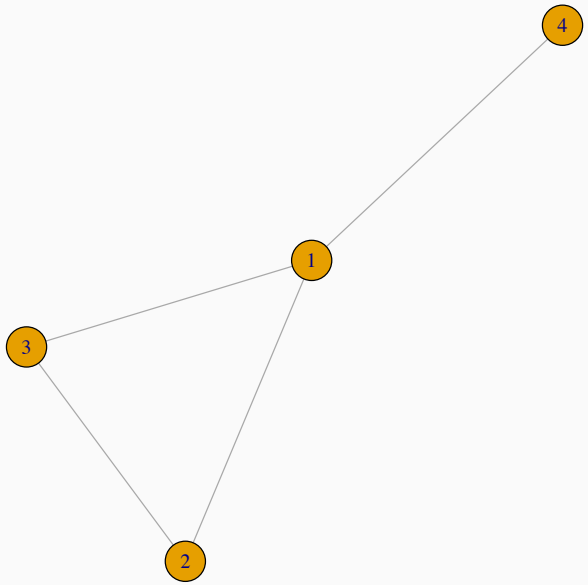


Generating igraph objects

- for modifications on the graph:
  `graph_from_adj_list(adjlist, mode = c("out", "in", "all")`

Example:

```
adjlist <- list(c(2,3,4), c(1,3), c(2, 1), c(1))
g <- graph_from_adj_list(adjlist, mode = "all")
```

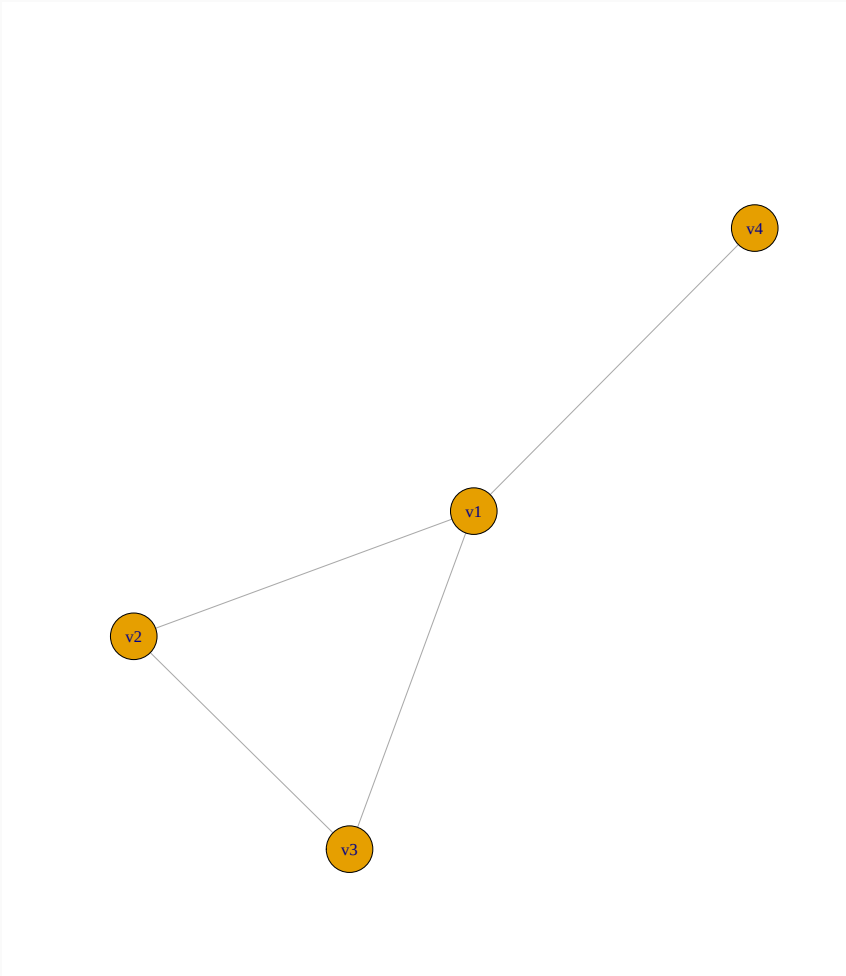


Generating igraph objects

- `make_graph(edges, directed = TRUE)`

Example:

```
g <- make_graph(c("v1","v2", "v2", "v3", "v3", "v1",
"v1", "v4"), directed = FALSE))
```



Generating igraph objects

- a two column matrix (numeric or character):
  `graph_from_edgelist(edgelist, directed = TRUE)`
  or `graph.edgelist()`

Example:

```
elist <- rbind(c("A", "B"),
               c("A", "C"),
               c("B", "C"))
g <- graph.edgelist(elist, directed = F)
```

Generating igraph objects

- creating graph object using data frame:
  `graph_from_data_frame(df, directed = TRUE)`

Example:

```
df <- data.frame(from = c("Alex", "Chris", "Bob"),
                 to = c("Chris", "Bob", "Alex"),
                 friendship=c(3,4,1))
g <- graph_from_data_frame(df, directed = T)
```

Generating igraph objects

- creating graph object using square adjacency matrix:

  ```
  graph_from_adjacency_matrix(adjm,
      mode = "undirected", weighted = NULL)
  ```

  or `graph.adjacency()`

Example:

```
adjm <- matrix(c(0, 1, 1,
                 1, 0, 1,
                 1, 1, 0), ncol=3)
g <- graph.adjacency(adjm, mode= "undirected")
```

igraph object

- Igraph graphs are of class “igraph” and have a special format for printing on screen

Example: igraph object “friends”

```
IGRAPH   fd17f23   UN--   7  7   ---
+ attr:  name (v/c)
+  edges  from  fd17f23  (vertex  names):
  [1] Alex--Bruno  Alex--Christian Alex--Dana
  [4] Alex--Erik   Alex--Fabi      Erik--Fabi
  [7] Fabi--Gina
```

Properties of graphs

- `is_directed(g)` or `is.directed(g)`
- `is_simple(g)` or `is.simple(g)`
- `is_weighted(g)` or `is.weighted(g)`
- `is_named(g)` or `is.named(g)`

Manipulating graphs

- add vertices: `add_vertices(g, number,...)`
  or `add.vertices(g, number,...)`
- add edges: `add_edges(g, edges,...)`
  or `add.edges(g, edges,...)`
- delete vertices: `delete_vertices(g, vertices)`
  or `delete.vertices(g, vertices)`
- delete edges: `delete_edges(g, edges)`
  or `delete.edges(g, edges)`

Vertices and edges

- vertex and edge ids: can be identified and specified via their numeric ids
  - vertices and edges are always numbered consecutively
  - when number of vertices changes (e.g. subgraph), then vertices are renumbered to satisfy criteria
  - when following vertices along a number of graph operations it is better to assign attributes to vertices
- To get all vertices of the graph object use `V(g)`.
- For all edges of the graph object use `E(g)`.

Vertices and edges

- Looking at specific edges of graphs (with all edge attributes of the selected attribute):
  `E(g)[[name of attribute == value]]`

Example:

`E(g)[[friendship == 2]]`

- View the first n vertices (with all attributes of the vertices in the sequence 1 to n) in the graph object: `V(g)[[1:n]]`
- number of vertices $|V|$: `gorder(g)`
- number of edges $|E|$: `gsize(g)`

Attributes of vertices and edges

- Adding an attribute to vertices of graph g:
  `g <- set_vertex_attr(g, "name", value = c(a, b, c))`
  Query all vertex attributes: `vertex_attr(g)`
- Adding an attribute to edges of graph g:
  `g <- set_edge_attr(g, "name", value = c(a, b, c))`
  Query all edge attributes: `edge_attr(g)`
- basic plotting: `plot(g)`

Exercises part 1 – Basics

Exercise

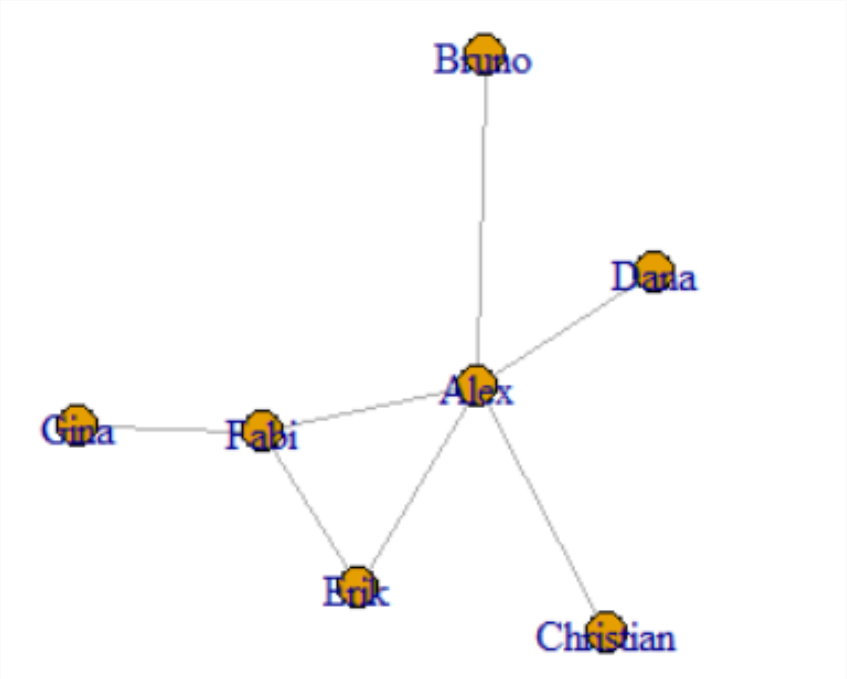
Do exercises 1 to 4.

Network Visualization

Graph Plotting

Plotting igraph objects:

- To visualize the created networks, the igraph objects can be plotted. For instance by using the `plot()` function.



Plotting Parameters:

- The igraph objects have many parameters for network plotting: Vertex-Parameters, Edge-Parameters and general Parameters.

Plotting Parameters

Node Parameters:

- vertex.color - Node color
- vertex.shape - Node shape (e.g. ”circle”, ”square” or ”rectangle”)
- vertex.size - Node size
- vertex.label - Character vector used to label the nodes

Edge Parameters:

- edge.color - Edge color
- edge.width - Edge width (default: 1)
- edge.arrow.size - Arrow size for directed graphs (default: 1)
- edge.lty - Line type (e.g. ”solid”, ”dotted”, ”dashed”)

General Parameters:

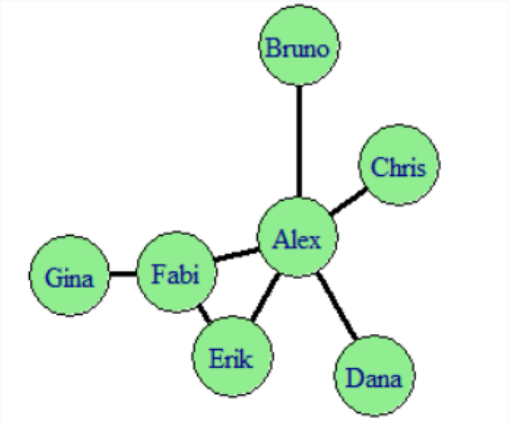
- margin - empty space around the plot (vector of length 4)
- frame - if TRUE, the plot will be framed
- main - if set, add a title to the plot

Style Adjustments

There are two main ways to set the plotting parameters:

Specifiy parameters in **`plot()`** function:

- `plot(g, edge.color = "black", vertex.color = "lightgreen", vertex.size = 50)`



Specifiy parameters by setting them in the igraph object:

- `V(g)$color = "red"`
- `E(g)$width = 3`
- `g$main = "Title"`

Network Layouts

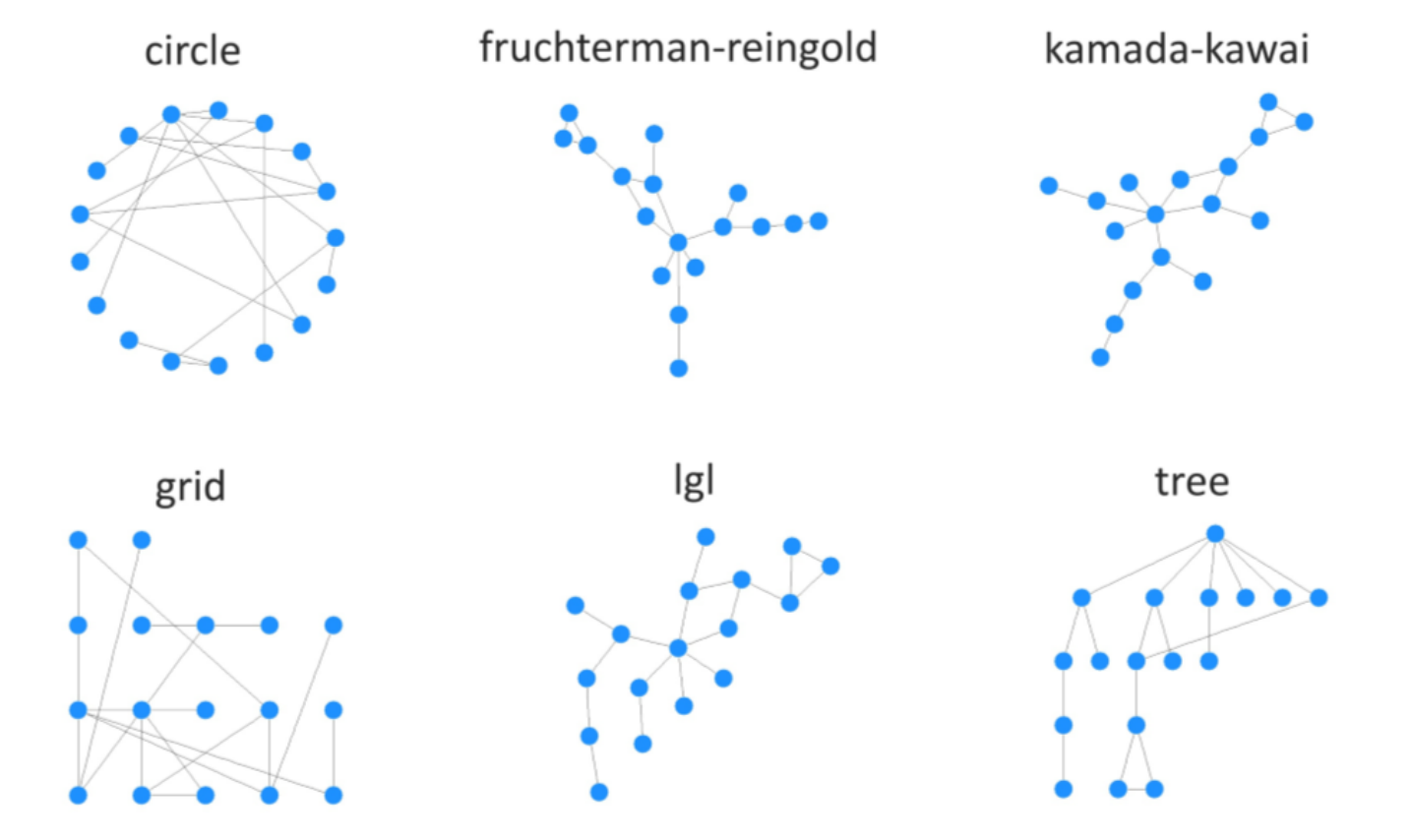
- Network layouts are algorithms that return coordinates for each node of the network and therefore determine the appearance of the plotted graph

- The layout can be set within the plot function:
  `plot(g, layout = layout.fruchterman.reingold(g))`

- There are many different layouts with the main goals:
  - Minimize edge crossing
  - No overlaps between vertices
  - Uniform edge lengths
  - Increase network symmetry

Network Layouts

Layout Types:



Highlighting Graph Aspects

For big networks it is not very helpful to plot just the whole network graph, because links and nodes are too dense.

Therefore one needs to highlight the important aspects of the graph. This can be realized by highlighting specific nodes or links with different colors or sizes:

Examples:

- Immediate neighbors
- Mark Group of nodes
- Shortest path visualization

Highlighting Graph Aspects

Highlighting specific nodes or links with colors or size:

- To focus the visualization on particular nodes or links, they can be highlighted using different colors or sizes

Example 1:

- **Immediate neighbors:**

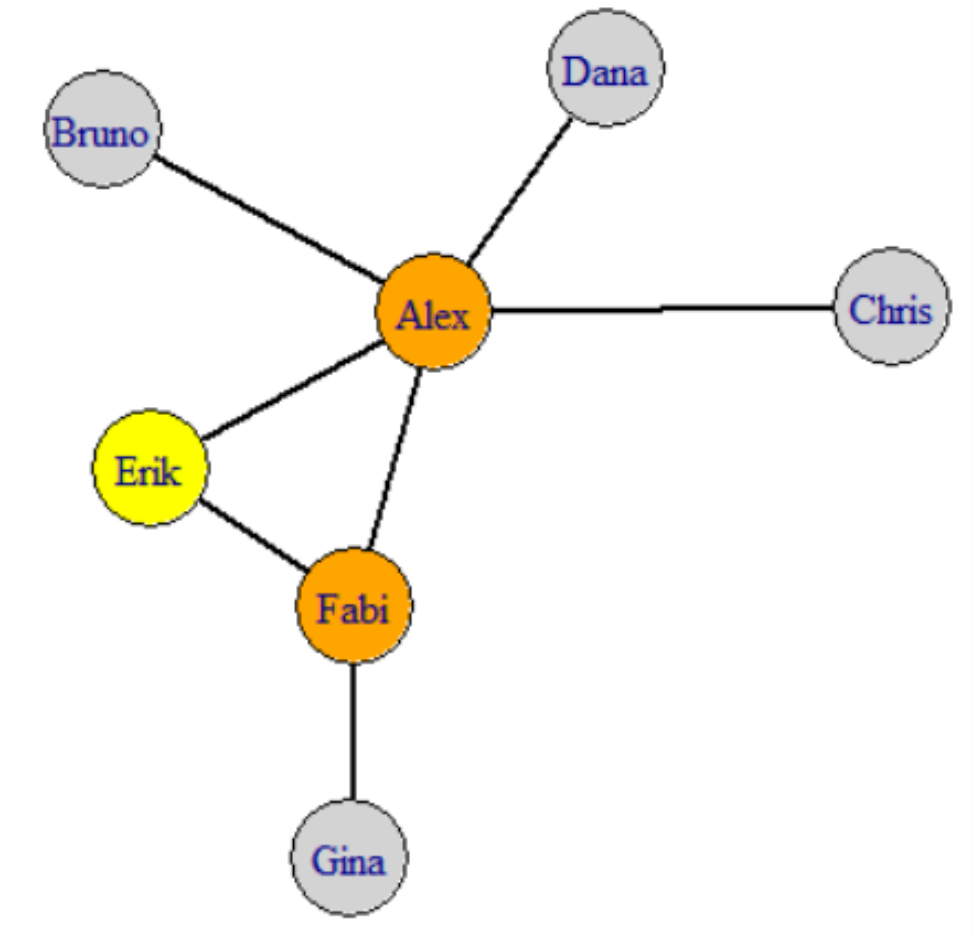
```
# calculate neighbors
neigh.nodes <- neighbors(g, V(g)["Eric"])

# set colors of vertices
vcol <- rep("lightgrey", vcount(g))
vcol[V(g)["Erik"] <- "yellow"
vcol[neigh.nodes] <- "orange

# plot the network
plot(g, vertex.color=vcol)
```

Highlighting Graph Aspects

Highlighting specific nodes or links with colors or size:

Example 1:

- Immediate neighbors:



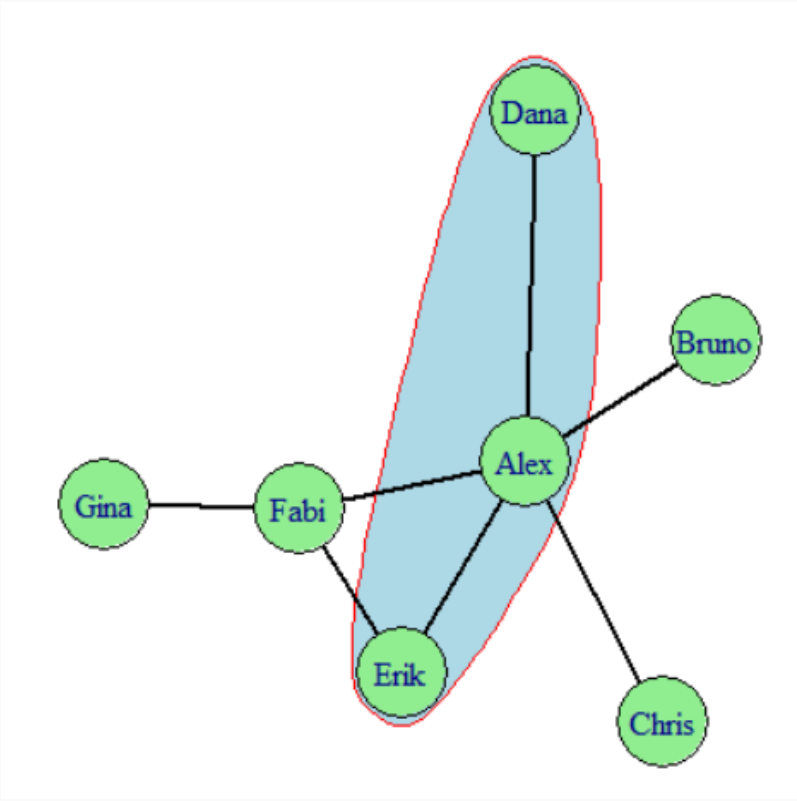
Highlighting Graph Aspects

Highlighting specific nodes or links with colors or size:

Example 2:

- Mark group of nodes:

```
plot(g, mark.groups=c(1,4,5), mark.col="lightblue")
```



Highlighting Graph Aspects

Highlighting specific nodes or links with colors or size:

Example 3:

- Shortest path visualization:

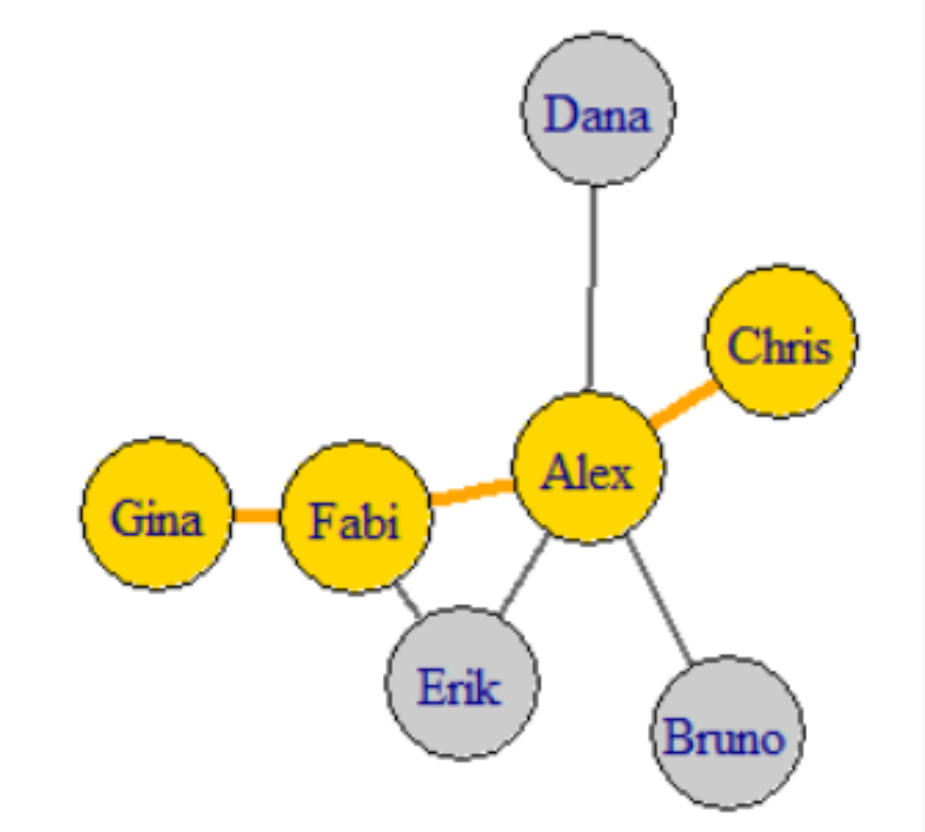
```
# calculate shortest path
sp <- shortest_paths(g, from=V(g)["Chris"],
to=V(g)["Gina"], output="both")

# set edge color
ecol <- rep("gray40", ecount(g))
ecol[unlist(sp$epath)] <- "orange"

# set vertex color
vcol <- rep("gray80", vcount(g))
vcol[unlist(sp$vpath)] <- "gold"

# plot the network
plot(g, vertex.color=vcol, edge.color=ecol)
```

Highlighting Graph Aspects

Highlighting specific nodes or links with colors or size:

Example 3:

- Shortest path visualization:



Interactive Visualization - tkplot

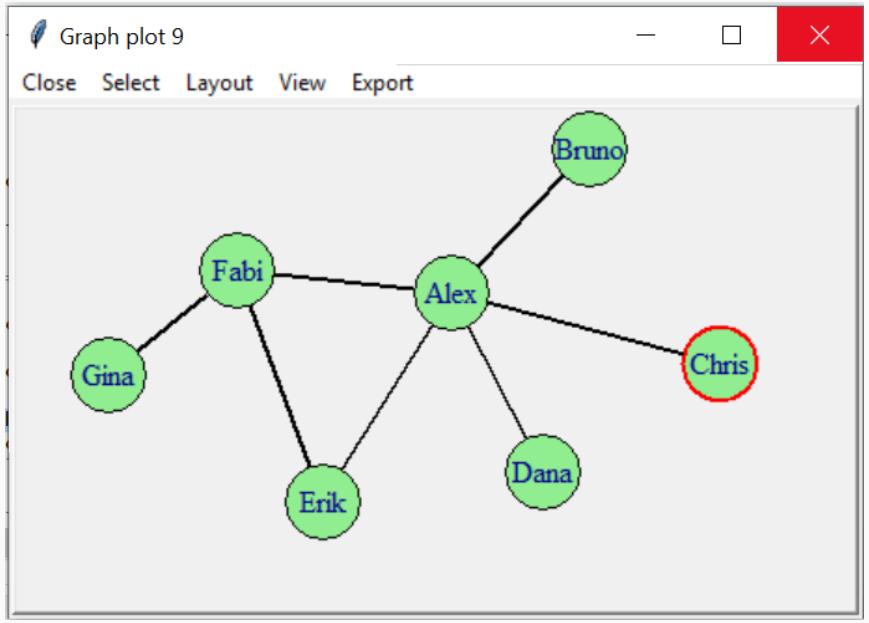
Manual adjustments with tkplot:

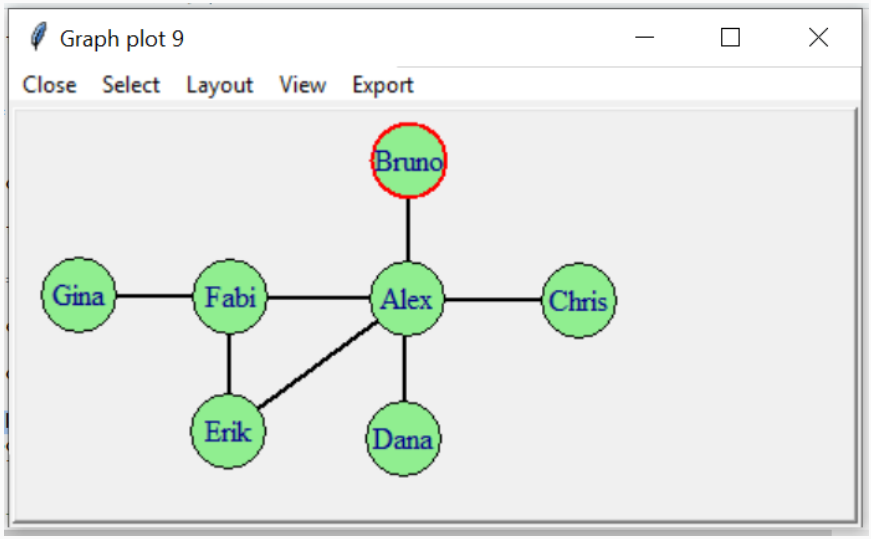
- R and igraph allow interactive plotting of networks
- Useful for small improvements of the network plots
- After adjusting the layout manually, you can get the coordinates of the nodes and use them for other plots

Example:

```
# open interactive plot and set id
tkid <-tkplot(g)

# get adjusted coordinates
layoutcoordinates <- tkplot.getcoords(tkid)

# plot the network with adjusted coordinates
plot(g, layout = layoutcoordinates)
```

Interactive Visualization - tkplot





Interactive Visualization - threejs

Creating interactive plots with threejs:

- Additional package for interactive network visualization
- With threejs it is possible to export networks from R to javascript, which can directly read igrpah objects
- The network can be saved in a HTML file and can be opened in a browser

Exercises part 2 – Visualization

Exercise

Do exercises 5 to 8.

Important Characterizations

Vertex relations

Vertex Environment:

- `neighbors(g, v = "Vertex", mode = " ")`
- `neighborhood(g, order = #steps, nodes = "Vertex", mode = " ")`
- the mode argument can be set to ”in”, ”out” or ”all”

Vertex relations

Vertex Environment:

- `neighbors(g, v = "Vertex", mode = " ")`
- `neighborhood(g, order = #steps, nodes = "Vertex", mode = " ")`
- the mode argument can be set to ”in”, ”out” or ”all”

Bridging Vertices or Common Neighbors:

- `a <- neighbors(g, v = "Vertex", mode = " ")`
- `b <- neighbors(g, v = "Vertex", mode = " ")`
- `intersection(a,b)`

Vertex relations

Vertex Environment:

- `neighbors(g, v = "Vertex", mode = " ")`
- `neighborhood(g, order = #steps, nodes = "Vertex", mode = " ")`
- the mode argument can be set to ”in”, ”out” or ”all”

Bridging Vertices or Common Neighbors:

- `a <- neighbors(g, v = "Vertex", mode = " ")`
- `b <- neighbors(g, v = "Vertex", mode = " ")`
- `intersection(a,b)`

Distances (aka path-lengths):

- `farthest_vertices(g)` and `get_diameter(g)`
- `distances(g, v = "Vertex", to = "Vertex", mode = " ")`
- `mean_distance(g, directed = TRUE)`

Determining important vertices

Vertex Degrees

- `degree(g, v = "VertexID", mode = " ")`
- measures the amount of outgoing, ingoing or total edges of a given vertex

Betweenness

- `betweenness(g, v = "VertexID", directed = TRUE, normalized = FALSE)`
- measures how often a **vertex** is part of the shortest path between other vertices
- `edge_betweenness(g)` would do the same job for edges

Advice: It might be useful to save the results for further calculations or for distribution plotting

- e.g.: `hist(graphdegrees)` or `which.max(betweenness)`

Network characteristics

(Edge) Density

- `edge_density(g, loops = F)`
  calculates the ratio between the number of edges in your graph and the number of all possible edges in the graph
- use `any_multiple(g)` to check if there are any multiple edges

Advice: Use the `simplify(g)` function to get rid of all multiple edges (and/or loops) if you want to use the edge_density function!

Network characteristics

(Edge) Density

- `edge_density(g, loops = F)`
  calculates the ratio between the number of edges in your graph and the number of all possible edges in the graph
- use `any_multiple(g)` to check if there are any multiple edges

Advice: Use the `simplify(g)` function to get rid of all multiple edges (and/or loops) if you want to use the edge_density function!

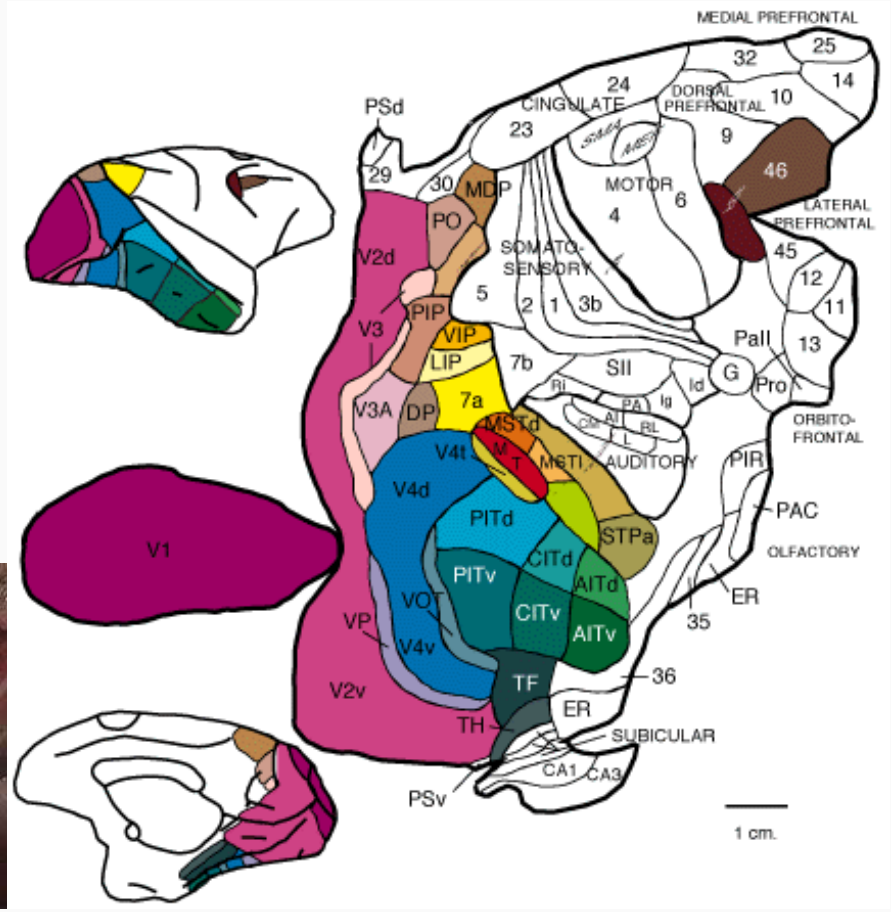
Eigenvector Centrality

- `eigen_centrality(g, directed = FALSE)$vector`
- vertices with high eigenvector centralities are connected to many other vertices which are again connected to many others (and so on...)
- note that the eigen_centrality function returns a list; therefore you need to specify the ”vector” variable for the specific eigen-values

Exercises - Analyzing a Brain-Network I

Exercises:

Do exercises 9 to 12



That’s what a Macaque and its rough brain structure look like (fyi)

Network (Sub-)Structures

Are vertices randomly or preferentially associated?

Assortativity

- **how likely is it for two vertices to be attached to each other if they have a given attribute in common?**
- attribute in question has to be converted into numeric values first (e.g. gender)
- values range from -1 to +1 and can be interpreted like correlation coefficients
- `assortativity_nominal(g, types = attr, directed = T)`

An example question could be:

Do females tend to connect more with other females than with men in a given social network?

Patterns of vertex association I

Note that the following functions will treat all graphs as undirected

- `as.undirected(g, mode = "collapse")` could be used to set a directed graph to an undirected one

Triangles

- refer to full interconnections between three vertices
- `count_triangles(g, vids = "Vertex")`
  returns the amount of triangles each vertex is part of

Patterns of vertex association I

Note that the following functions will treat all graphs as undirected

- `as.undirected(g, mode = "collapse")` could be used to set a directed graph to an undirected one

Triangles

- refer to full interconnections between three vertices
- `count_triangles(g, vids = "Vertex")`
  returns the amount of triangles each vertex is part of

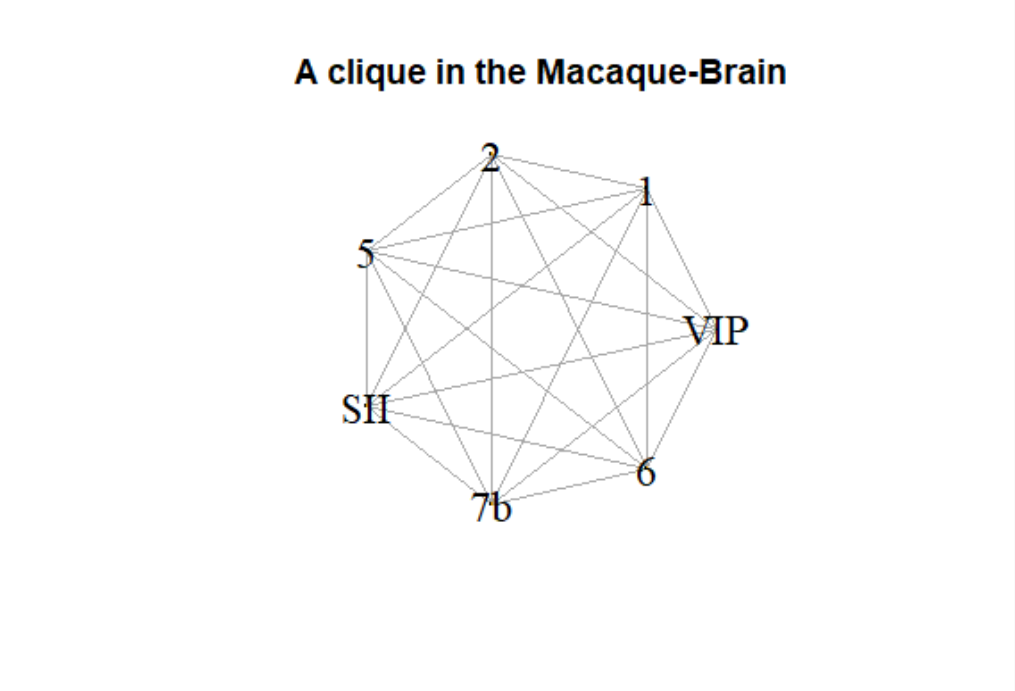
Transitivity

- calculates the proportion of existing triangles (”clustering coefficient”)
- default calculation method is ”global”, however you can specify the function to give values for each vertex
- `transitivity(g, type = "global", "local", ...)`

Patterns of vertex association II

Cliques

- are fully interconnected (sub-)graphs
- since cliques rarely occur as whole networks, they are usually analyzed as substructures
- `largest_cliques(g)` & `max_cliques(g, min = ?, max = ?)`



Community detection I

- communities are substructures of a network that have more dense connections between their members than between members and non-members
- community detection can help to identify functional sub-units of a network
- igraph has several built-in algorithms with different strengths and weaknesses
- some algorithms can only deal with undirected graphs, therefore you may have to transform your graph first: `as.undirected(g, mode = "collapse")`

Community detection II

Fast Greedy Method (modularity based, *only undirected*)

- `cluster_fast_greedy(g)`
- tries to build larger and larger communities as it's adding vertices step by step and assesses a modularity score with each step

Edge Betweenness Method (divisive)

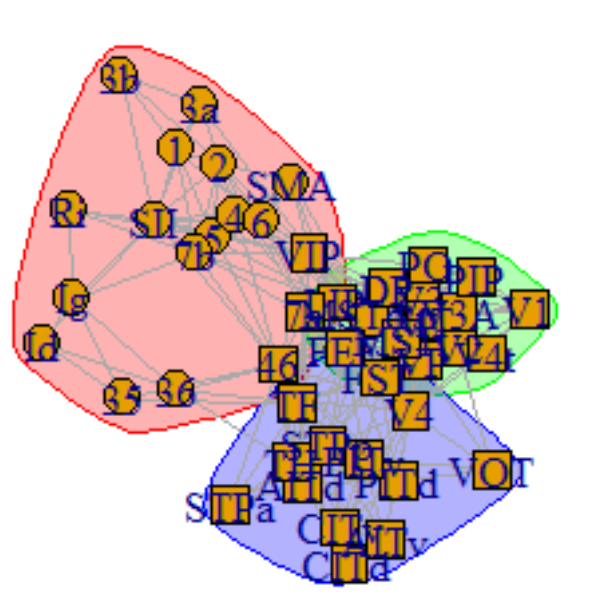
- `cluster_edge_betweenness(g)`
- divides the graph into smaller and smaller pieces until it finds edges that are assumed to be bridges between communities (having high betweenness scores)

note that these functions return *community-objects* which contain several pieces of information to enable nice and easy community-plotting!

Community Visualization

Example:

```
# calculate communities via the fast greedy method
comms <- cluster_fast_greedy(macaqueud)

# plot the graph and set mark.groups to the community
object
plot(macaqueud, mark.groups = comms)
```



One possible result of community detection in the macaque brain

Random Network Testing

- helps to infer if a given network metric is meaningful
- random graphs can be generated by algorithms and should be defined with the same amount of vertices and the same density as the original graph
- `erdos.renyi.game(n = size, p.or.m = edge_density, type = "gnp", directed = FALSE)`
  can be used as a function to create such random graphs
- you can create a lot (e.g. 1000) of these random graphs and compare their average metrics with the original graph

Exercises - Analyzing a Brain-Network II

Exercises:

Do exercises 13 to 16 which will be the final ones! :-)

Thank you for your kind attention.

References

R. Diestel.
Graphentheorie.
Springer Verlag, 4. Auflage, Heidelberg 2010.

L. Négyessy et al.
Prediction of the main cortical areas and connections involved in the tactile function of the visual cortex by network analysis.
European Journal of Neuroscience, 23(7):1919–1930, 2006.

K. Ognyanova.
Network Analysis and Visualization with R and igraph.
NetSciX 2016 School of Code Workshop, Wroclaw, Poland.

DataCamp Course ”Network Analysis in R” by James Curley.
www.datacamp.com/courses/network-analysis-in-r.
Some examples were adapted from this course.