Network analysis with the igraph package

R-Lunch Geneva

Yannick Rochat, UNIL
02/04/2019

Me

- EPFL, MSc Mathematics, 2007
- UNIL SSP, PhD Applied Mathematics, 2014
  - At first, social network analysis (centrality)
  - Then, in digital humanities (character network analysis)

How I discovered and used igraph

- My PhD thesis in social network analysis and a colleague's PhD thesis in evolutionary game theory
- Through teaching
- Gábor Csárdi was working in Lausanne
- igraph is how I discovered R
- (didn't want to use Ucinet)

The origins of igraph

- Statistical simulation (Gábor Csárdi’s and Tamász Nepusz’s research)

For example…

- `aging.ba.game`
- `aging.barabasi.game`
- `aging.prefatt.game`
- etc.

1.1.1 Why another network analysis package?

The igraph library was developed because of the lack of network analysis software which (1) can handle large graphs efficiently, (2) can be embedded into a higher level program or programming language (like Python, Perl or GNU R) and (3) can be used both interactively and non-interactively.

The capability of handling large graphs was important because the authors were confronted with graphs with millions of vertices and edges.

Embedding igraph into Python or GNU R creates a very productive research environment, well suited for rapid development. All the expressing power of GNU R (or other higher level languange) is readily available in a convenient integrated environment for generating, manipulating and measuring graphs, and evaluating these measurements.

Interactive means of software usage is nowadays considered as superior to non-interactive interfaces, which is very true for most cases. Dealing with large graphs can be different though – if it takes three months to calculate the diameter of a graph, nobody wants that to be interactive.

Source: Csárdi, Gábor, and Tamás Nepusz. *The igraph software package for complex network research.* InterJournal, Complex Systems 1695.5 (2006): 1-9.

This paper has 4425 citations as of April 2019.

What is igraph

- Awesome community
- Availability and reactivity from Gábor Csárdi and Tamás Nepusz and others
- It's old (2005) and it's always been maintained (on a daily basis!)
- Today: integrated to the tidyverse and linked to ggplot2.

Websites

- Main website

https://igraph.org/r/

- Mailing list

https://lists.nongnu.org/mailman/listinfo/igraph-help

GitHub repositories

- C

https://github.com/igraph/igraph

- R

https://github.com/igraph/rigraph

- Python

https://github.com/igraph/python-igraph

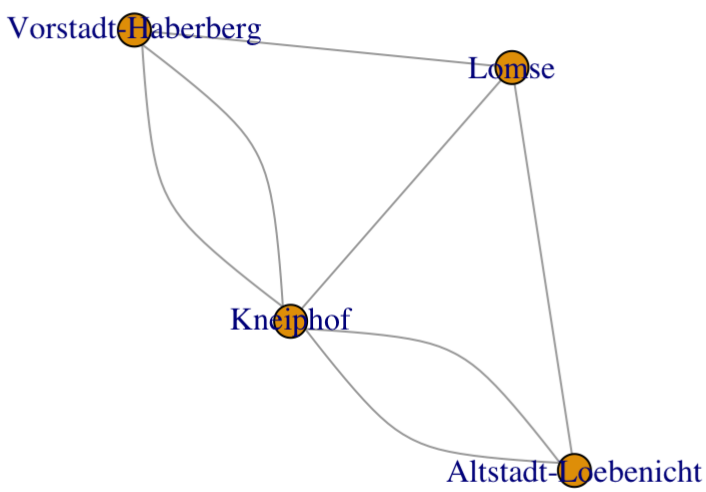
igraphdata package

https://github.com/igraph/igraphdata

igraphdata-package	The igraphdata package
enron	Enron Email Network
foodwebs	A collection of food webs
igraphdata	The igraphdata package
immuno	Immunoglobulin interaction network
karate	Zachary's karate club network
kite	Krackhardt's kite
Koenigsberg	Bridges of Koenigsberg from Euler's times
macaque	Visuotactile brain areas and connections
rfid	Hospital encounter network data
UKfaculty	Friendship network of a UK university faculty
USairports	US airport network, 2010 December
yeast	Yeast protein interaction network

Graph Theory and Network Analysis

```
library(igraphdata)
data("Koenigsberg")
Koenigsberg %>% plot
```



To name but a few examples, ‘network analysis’ is carried out in areas such as project planning, complex systems, electrical circuits, social networks, transporta- tion systems, communication networks, epidemiology, bioinformatics, hypertext systems, text analysis, bibliometrics, organization theory, genealogical research and event analysis.

Source: Brandes and Erlebach (2005, p. 1)

Graphs (and networks) are about relational data

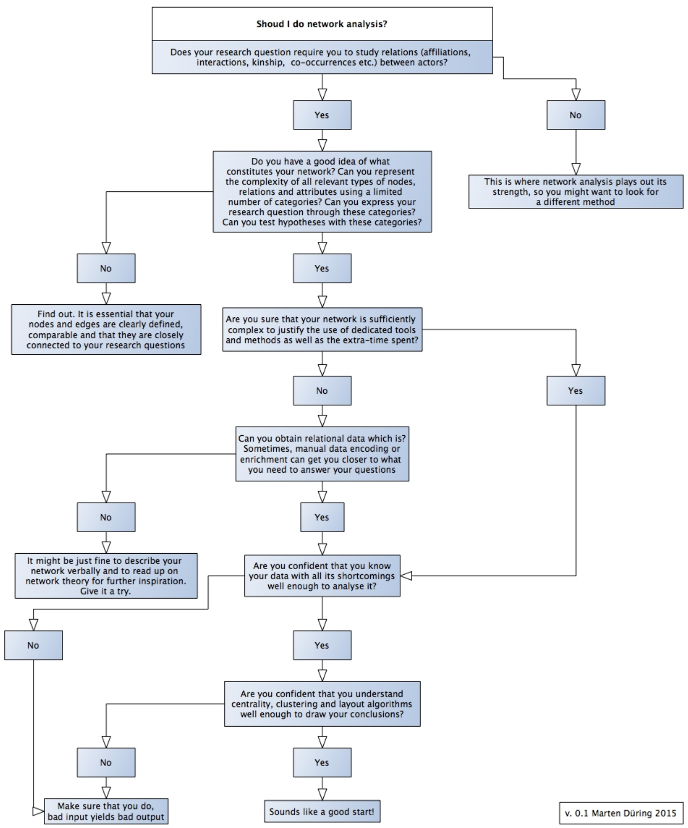
2.5 Summary

If you are planning research, in which social relations appear to be relevant, you must first answer the question concerning the purpose served by the network analysis. If you would like to make new phenomena visible, the description of network connections is sufficient in many cases. If a network is used to explain social phenomena, you must decide whether the network is the dependent or independent variable (see Section 2.2).

After this question has been clarified, you must then consider which data are necessary to answer the research question. Before collecting any data, you must choose the relevant unit of analysis, the relevant relationship (form and content), and the level of data analysis.

Source: Hennig, Marina, et al. *Studying social networks: A guide to empirical research.* Campus Verlag, 2012.

Should I do network analysis?



Source: https://cvcedhlab.hypotheses.org/125

The igraph objects

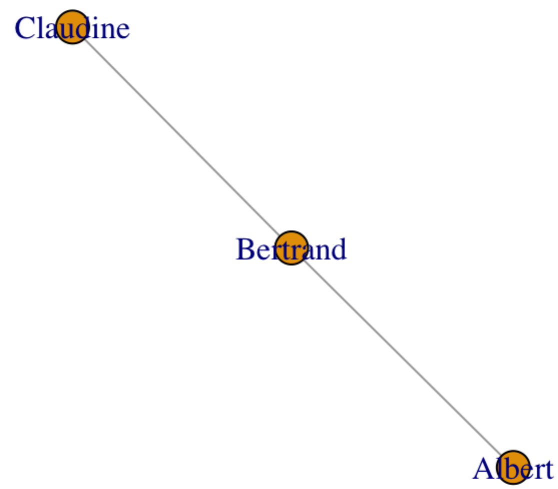
```
g0 <- graph_from_literal("Albert" -- "Bertrand" -- "Claudine")
g0
```

```
## IGRAPH e08b065 UN-- 3 2 --
## + attr: name (v/c)
## + edges from e08b065 (vertex names):
## [1] Albert  --Bertrand Bertrand--Claudine
```

Graph ids are used to check that a vertex or edge sequence belongs to a graph. If you create a new graph by changing the structure of a graph, the new graph will have a new id. Changing the attributes will not change the id.

Source: `?graph_id`

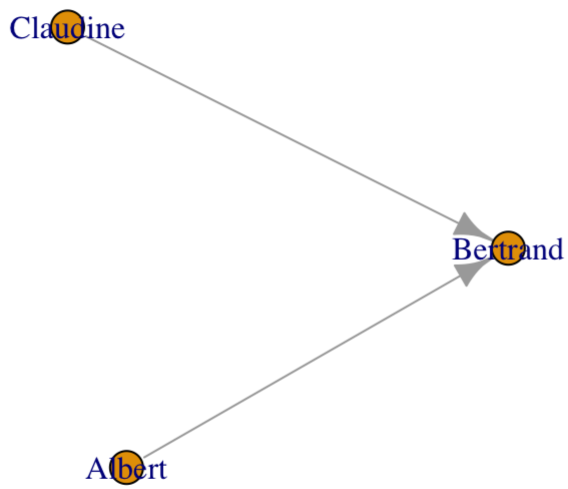
```
plot(g0)
```



```r
g1 <- graph_from_literal("Albert" -+ "Bertrand" +- "Claudine")
g1
```

```
## IGRAPH 3ee4e0a DN-- 3 2 --
## + attr: name (v/c)
## + edges from 3ee4e0a (vertex names):
## [1] Albert  ->Bertrand Claudine->Bertrand
```

```
plot(g1)
```



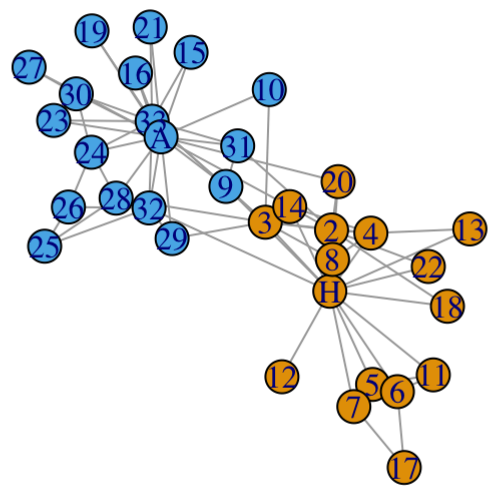
the first is ‘U’ for undirected and ‘D’ for directed graphs. The second is ‘N’ for named graph (i.e. if the graph has the ‘name’ vertex attribute set). The third is ‘W’ for weighted graphs (i.e. if the ‘weight’ edge attribute is set). The fourth is ‘B’ for bipartite graphs (i.e. if the ‘type’ vertex attribute is set).

Source: `?igraph`

```r
library(igraphdata)
data(karate)
karate
```

```
## IGRAPH 4b458a1 UNW- 34 78 -- Zachary's karate club network
## + attr: name (g/c), Citation (g/c), Author (g/c), Faction (v/n),
## | name (v/c), label (v/c), color (v/n), weight (e/n)
## + edges from 4b458a1 (vertex names):
##  [1] Mr Hi  --Actor 2  Mr Hi  --Actor 3  Mr Hi  --Actor 4
##  [4] Mr Hi  --Actor 5  Mr Hi  --Actor 6  Mr Hi  --Actor 7
##  [7] Mr Hi  --Actor 8  Mr Hi  --Actor 9  Mr Hi  --Actor 11
## [10] Mr Hi  --Actor 12 Mr Hi  --Actor 13 Mr Hi  --Actor 14
## [13] Mr Hi  --Actor 18 Mr Hi  --Actor 20 Mr Hi  --Actor 22
## [16] Mr Hi  --Actor 32 Actor 2--Actor 3  Actor 2--Actor 4
## [19] Actor 2--Actor 8  Actor 2--Actor 14 Actor 2--Actor 18
## + ... omitted several edges
```

```
plot(karate)
```



19
21
15
27
16
10
30
23
33
A
31
24
20
9
26
28
32
14
3
2
4
13
25
29
8
22
H
18
12
11
5
6
7
17

```
karate$Citation
```

```
## [1] "Wayne W. Zachary. An Information Flow Model for Conflict and Fission in Smal
```

```
karate$name
```

```
## [1] "Zachary's karate club network"
```

```
V(karate)$name
```

```
##  [1] "Mr Hi"     "Actor 2"  "Actor 3"  "Actor 4"  "Actor 5"  "Actor 6"
##  [7] "Actor 7"  "Actor 8"  "Actor 9"  "Actor 10" "Actor 11" "Actor 12"
## [13] "Actor 13" "Actor 14" "Actor 15" "Actor 16" "Actor 17" "Actor 18"
## [19] "Actor 19" "Actor 20" "Actor 21" "Actor 22" "Actor 23" "Actor 24"
## [25] "Actor 25" "Actor 26" "Actor 27" "Actor 28" "Actor 29" "Actor 30"
## [31] "Actor 31" "Actor 32" "Actor 33" "John A"
```

```
V(karate)
```

```
## + 34/34 vertices, named, from 4b458a1:
##  [1] Mr Hi    Actor 2  Actor 3  Actor 4  Actor 5  Actor 6  Actor 7
##  [8] Actor 8  Actor 9  Actor 10 Actor 11 Actor 12 Actor 13 Actor 14
## [15] Actor 15 Actor 16 Actor 17 Actor 18 Actor 19 Actor 20 Actor 21
## [22] Actor 22 Actor 23 Actor 24 Actor 25 Actor 26 Actor 27 Actor 28
## [29] Actor 29 Actor 30 Actor 31 Actor 32 Actor 33 John A
```

```
E(karate)$weight
```

```
##  [1] 4 5 3 3 3 3 2 2 2 3 1 3 2 2 2 2 6 3 4 5 1 2 2 2 3 4 5 1 3 2 2 2 3 3 3
## [36] 2 3 5 3 3 3 3 3 4 2 3 3 2 3 4 1 2 1 3 1 2 3 5 4 3 5 4 2 3 2 7 4 2 4 2
## [71] 2 4 2 3 3 4 4 5
```

Creating an igraph object

- `graph_from_literal`
- `graph` (takes numeric vertex ids directly)
- `graph.atlas`
- `make_`

make_	Make a new graph
make_bipartite_graph	Create a bipartite graph
make_chordal_ring	Create an extended chordal ring graph
make_clusters	Creates a communities object.
make_de_bruijn_graph	De Bruijn graphs
make_directed_graph	Create an igraph graph from a list of edges, or a notable graph
make_ego_graph	Neighborhood of graph vertices
make_empty_graph	A graph with no edges
make_full_bipartite_graph	Create a full bipartite graph
make_full_citation_graph	Create a complete (full) citation graph
make_full_graph	Create a full graph
make_graph	Create an igraph graph from a list of edges, or a notable graph
make_kautz_graph	Kautz graphs
make_lattice	Create a lattice graph
make_line_graph	Line graph of a graph
make_ring	Create a ring graph
make_star	Create a star graph, a tree with n vertices and n - 1 leaves
make_tree	Create tree graphs
make_undirected_graph	Create an igraph graph from a list of edges, or a notable graph

Importing an igraph object

- `graph_from_edgelist`
- `graph_from_data_frame`
- `graph_from_adjacency_matrix`
- `graph_from_`

And `read_graph`

Exporting an igraph object

Attribute values can be set to any R object, but note that storing the graph in some file formats might result the loss of complex attribute values. All attribute values are preserved if you use `save` and `load` to store/retrieve your graphs.

And `write_graph`

Manipulating an igraph object

```
g <- karate
```

```
vcount(g) ## Number of vertices
```

```
## [1] 34
```

```
ecount(g) ## Number of edges
```

```
## [1] 78
```

```r
diameter(g)      ## The length of the longest shortest path
```

```
## [1] 13
```

```r
graph.density(g) ## Number of edges per edges possible
```

```
## [1] 0.1390374
```

```r
degree(g) ## Number of connection
```

```
##    Mr Hi  Actor 2  Actor 3  Actor 4  Actor 5  Actor 6  Actor 7  Actor 8
##       16        9       10        6        3        4        4        4
##  Actor 9 Actor 10 Actor 11 Actor 12 Actor 13 Actor 14 Actor 15 Actor 16
##        5        2        3        1        2        5        2        2
## Actor 17 Actor 18 Actor 19 Actor 20 Actor 21 Actor 22 Actor 23 Actor 24
##        2        2        2        3        2        2        2        5
## Actor 25 Actor 26 Actor 27 Actor 28 Actor 29 Actor 30 Actor 31 Actor 32
##        3        3        2        4        3        4        4        6
## Actor 33   John A
##       12       17
```

```
degree.distribution(g)
```

```
##  [1] 0.00000000 0.02941176 0.32352941 0.17647059 0.17647059 0.08823529
##  [7] 0.05882353 0.00000000 0.00000000 0.02941176 0.02941176 0.00000000
## [13] 0.02941176 0.00000000 0.00000000 0.00000000 0.02941176 0.02941176
```

```r
is.connected(g) ## Is the network connected?
```

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

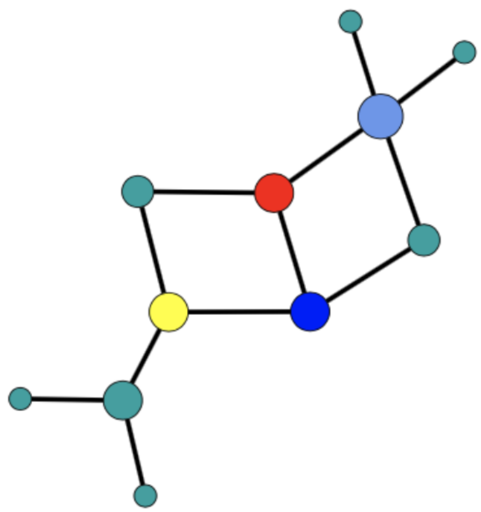
```r
no.clusters(g)  ## How many clusters?
```

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

Centrality

- Degree (cf. `degree` earlier)
- Closeness
- Betweenness
- Eigenvector (or Bonacich or Power)

A comparison of centrality measures



degree
eigenvector
closeness
betweenness

Brandes (2012)

```r
closeness(g) ## how close to others on average
```

```
##       Mr Hi    Actor 2    Actor 3    Actor 4    Actor 5    Actor 6 
## 0.007575758 0.005494505 0.005847953 0.005235602 0.004587156 0.004587156 
##     Actor 7    Actor 8    Actor 9   Actor 10   Actor 11   Actor 12 
## 0.004629630 0.005494505 0.005882353 0.005494505 0.005291005 0.004405286 
##    Actor 13   Actor 14   Actor 15   Actor 16   Actor 17   Actor 18 
## 0.006134969 0.005747126 0.005154639 0.004098361 0.003267974 0.005780347 
##    Actor 19   Actor 20   Actor 21   Actor 22   Actor 23   Actor 24 
## 0.005586592 0.007246377 0.006134969 0.005319149 0.004716981 0.004149378 
##    Actor 25   Actor 26   Actor 27   Actor 28   Actor 29   Actor 30 
## 0.004694836 0.003690037 0.005076142 0.004629630 0.006097561 0.005263158 
##    Actor 31   Actor 32   Actor 33     John A 
## 0.004878049 0.006097561 0.005952381 0.007575758
```

```
betweenness(g) ## how often in between other nodes (taking shortest paths) on averag
```

```
##      Mr Hi    Actor 2    Actor 3    Actor 4    Actor 5    Actor 6
## 250.150000  33.800000  36.650000   1.333333   0.500000  15.500000
##    Actor 7    Actor 8    Actor 9   Actor 10   Actor 11   Actor 12
##  15.500000   0.000000  13.100000   7.283333   0.500000   0.000000
##   Actor 13   Actor 14   Actor 15   Actor 16   Actor 17   Actor 18
##   0.000000   1.200000   0.000000   0.000000   0.000000  16.100000
##   Actor 19   Actor 20   Actor 21   Actor 22   Actor 23   Actor 24
##   3.000000 127.066667   0.000000   0.000000   0.000000   1.000000
##   Actor 25   Actor 26   Actor 27   Actor 28   Actor 29   Actor 30
##  33.833333   0.500000   0.000000   6.500000  10.100000   0.000000
##   Actor 31   Actor 32   Actor 33     John A
##   3.000000  66.333333  38.133333 209.500000
```

```r
eigen_centrality(g)$vector ## how central based on other's values of centrality
```

```
##      Mr Hi    Actor 2    Actor 3    Actor 4    Actor 5    Actor 6
## 0.85787944 0.82876616 0.99036448 0.54536909 0.15291191 0.18519270
##    Actor 7    Actor 8    Actor 9   Actor 10   Actor 11   Actor 12
## 0.18250148 0.49006831 0.67825515 0.13788382 0.12588193 0.11866884
##   Actor 13   Actor 14   Actor 15   Actor 16   Actor 17   Actor 18
## 0.11499616 0.66050150 0.21845188 0.31067062 0.05086244 0.11732645
##   Actor 19   Actor 20   Actor 21   Actor 22   Actor 23   Actor 24
## 0.13429645 0.20164970 0.17234251 0.15554033 0.22248354 0.59886179
##   Actor 25   Actor 26   Actor 27   Actor 28   Actor 29   Actor 30
## 0.14020695 0.33667492 0.16102652 0.40561477 0.23660019 0.37306834
##   Actor 31   Actor 32   Actor 33     John A
## 0.43481077 0.57527665 0.91256318 1.00000000
```

Transitivity

```
transitivity(g) ## average of clustering coefficients (closing the triangles)
```

```
## [1] 0.2556818
```

```
transitivity(g, type = "localundirected") ## clustering coefficient
```

```
##  [1] 0.1500000 0.3333333 0.2444444 0.6666667 0.6666667 0.5000000 0.5000000
##  [8] 1.0000000 0.5000000 0.0000000 0.6666667       NaN 1.0000000 0.6000000
## [15] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 0.3333333 1.0000000
## [22] 1.0000000 1.0000000 0.4000000 0.3333333 0.3333333 1.0000000 0.1666667
## [29] 0.3333333 0.6666667 0.5000000 0.2000000 0.1969697 0.1102941
```

`tidygraph`

A graph, while not “tidy” in itself, can be thought of as two tidy data frames describing node and edge data respectively. ‘tidygraph’ provides an approach to manipulate these two virtual data frames using the API defined in the ‘dplyr’ package, as well as provides tidy interfaces to a lot of common graph algorithms.

Source: `?tidygraph`

- https://www.data-imaginist.com/2017/introducing-tidygraph/
- https://www.data-imaginist.com/2018/tidygraph-1-1-a-tidy-hope/

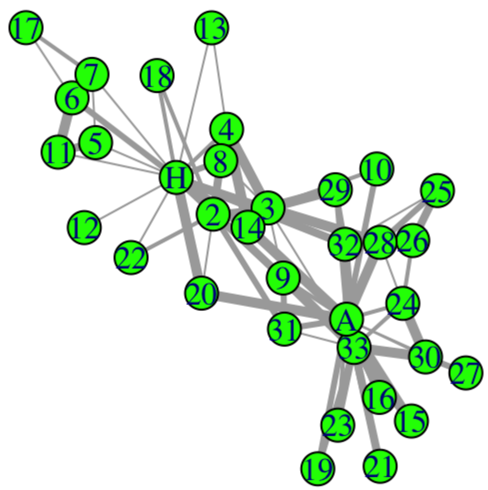
```r
library(tidygraph)

play_erdos_renyi(10, 0.5) %>%
  activate(nodes) %>%
  mutate(degree = centrality_degree()) %>%
  activate(edges) %>%
  mutate(centrality = centrality_edge_betweenness()) %>%
  arrange(centrality)
#> # A tbl_graph: 10 nodes and 37 edges
#> #
#> # A directed simple graph with 1 component
#> #
#> # Edge Data: 37 x 3 (active)
#>    from    to centrality
#>   <int> <int>      <dbl>
#> 1    10     3   1.500000
#> 2     5     6   1.500000
#> 3     2     7   1.500000
#> 4    10     9   1.500000
#> 5     8     7   1.833333
#> 6     5     8   1.833333
#> # ... with 31 more rows
#> #
#> # Node Data: 10 x 1
#>   degree
#>    <dbl>
#> 1      5
#> 2      3
#> 3      4
#> # ... with 7 more rows
```

Source: https://github.com/thomasp85/tidygraph

Network visualisations

```
plot(g, vertex.color = "green", edge.width = sample(5, ecount(g), replace = TRUE))
```



All options of classic `plot` function: `?igraph.plotting`

Layout

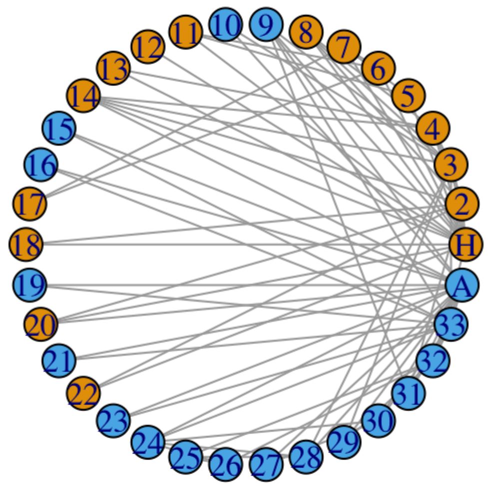
```
g$layout
```

```
## NULL
```

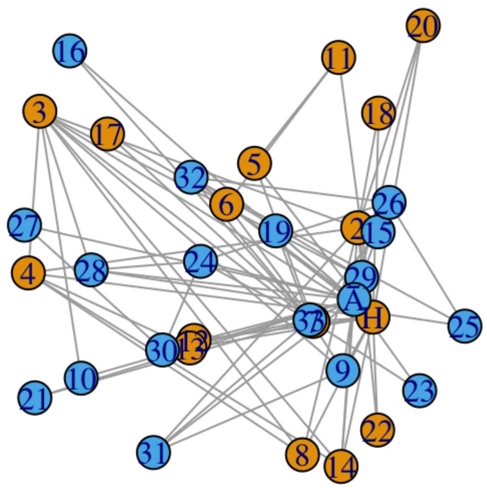
```
g$layout <- layout.circle(g)
head(g$layout)
```

```
##           [,1]      [,2]
## [1,] 1.0000000 0.0000000
## [2,] 0.9829731 0.1837495
## [3,] 0.9324722 0.3612417
## [4,] 0.8502171 0.5264322
## [5,] 0.7390089 0.6736956
## [6,] 0.6026346 0.7980172
```

```
plot(g)
```


10
9
8
7
6
5
4
3
2
H
A
33
32
31
30
29
28
27
26
25
24
23
22
21
20
19
18
17
16
15
14
13
12
11

```
g$layout <- layout_randomly(g)
plot(g)
```



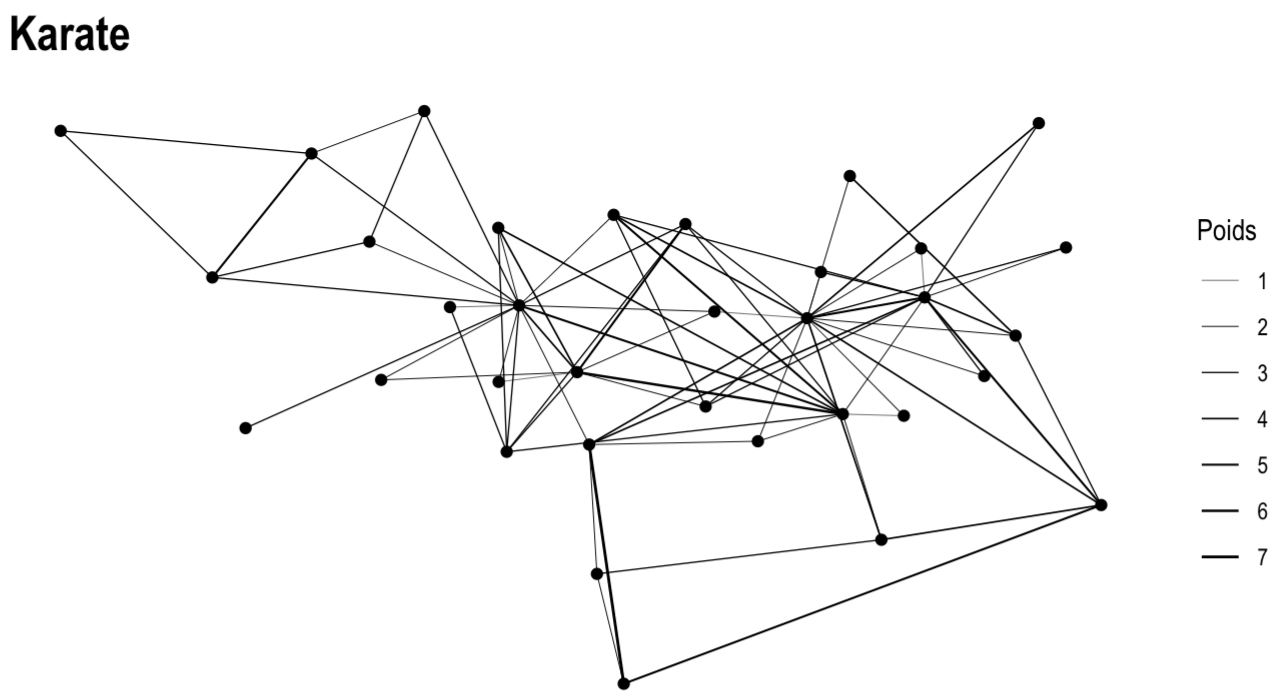
Beautiful network visualisations

ggraph

- https://www.data-imaginist.com/2017/announcing-ggraph/
- https://www.data-imaginist.com/2017/ggraph-introduction-nodes/
- https://www.data-imaginist.com/2017/ggraph-introduction-edges/
- https://www.data-imaginist.com/2017/ggraph-introduction-layouts/

```r
p <- ggraph(karate, layout = 'kk') +
    geom_edge_link(aes(edge_width = weight)) +
    scale_edge_width_continuous(range = c(.1, .5), "Poids") +
    geom_node_point() +
    ggtitle('Karate') +
    theme_graph()
```

```
p
```



Interactive graphs with networkD3

```
example("forceNetwork")
```

Community Detection

The different methods for finding communities, they all return a `communities` object: `cluster_edge_betweenness`, `cluster_fast_greedy`, `cluster_label_prop`, `cluster_leading_eigen`, `cluster_louvain`, `cluster_optimal`, `cluster_spinglass`, `cluster_walktrap`.

Community Detection with tidygraph

`group_components`: Group by connected compenents using [`igraph::components()`]

`group_edge_betweenness`: Group densely connected nodes using [`igraph::cluster_edge_betweenness()`]

`group_fast_greedy`: Group nodes by optimising modularity using [`igraph::cluster_fast_greedy()`]

`group_infomap`: Group nodes by minimizing description length using [`igraph::cluster_infomap()`]

`group_label_prop`: Group nodes by propagating labels using [`igraph::cluster_label_prop()`]

`group_leading_eigen`: Group nodes based on the leading eigenvector of the modularity matrix using [`igraph::cluster_leading_eigen()`]

`group_louvain`: Group nodes by multilevel optimisation of modularity using [`igraph::cluster_louvain()`]

`group_optimal`: Group nodes by optimising the moldularity score using [`igraph::cluster_optimal()`]

`group_spinglass`: Group nodes using simulated annealing with [`igraph::cluster_spinglass()`]

`group_walktrap`: Group nodes via short random walks using [`igraph::cluster_walktrap()`]

`group_biconnected_component`: Group edges by their membership of the maximal binconnected components using [`igraph::biconnected_components()`]

How fast (a slide from 2010)

Functionality, what can be calculated?

Fast (millions)	creating graphs (most of the time) • structural modification (add/delete edges/vertices) • subgraph • simplify • graph.decompose • degree • clusters • graph.density • is.simple, is.loop, is.multiple • articulation points and biconnected components • ARPACK stuff: page.rank, hub.score, authority.score, eigenvector centrality • transitivity • Burt's constraint • dyad & triad census, graph motifs • k-cores • MST • reciprocity • modularity • closeness and (edge) betweenness *estimation* • *shortest paths from one source* • *generating* $G_{n,p}$ *and* $G_{n,m}$ *graphs* • *generating PA graphs with various PA exponents* • *topological sort*
Slow (10000)	closeness • diameter • betweenness • all-pairs shortest paths, average path length • most layout generators •
Very slow (100)	cliques • cohesive blocks • edge/vertex connectivity • maximum flows and minimum cuts • power centrality • alpha centrality • (sub)graph isomorphism

Source: the igraph workshop of Gábor Csárdi at University of Lausanne, 2010.

Other ressources

- http://kateto.net/networks-r-igraph
- https://www.jessesadler.com/post/network-analysis-with-r/
- https://github.com/briatte/awesome-network-analysis
- Kolaczyk, Eric D., and Gábor Csárdi. *Statistical analysis of network data with R.* Vol. 65. New York: Springer, 2014.

Contact

yannick.rochat@gmail.com

https://yro.ch

@yrochat