
Practical statistical network analysis (with **R** and igraph)

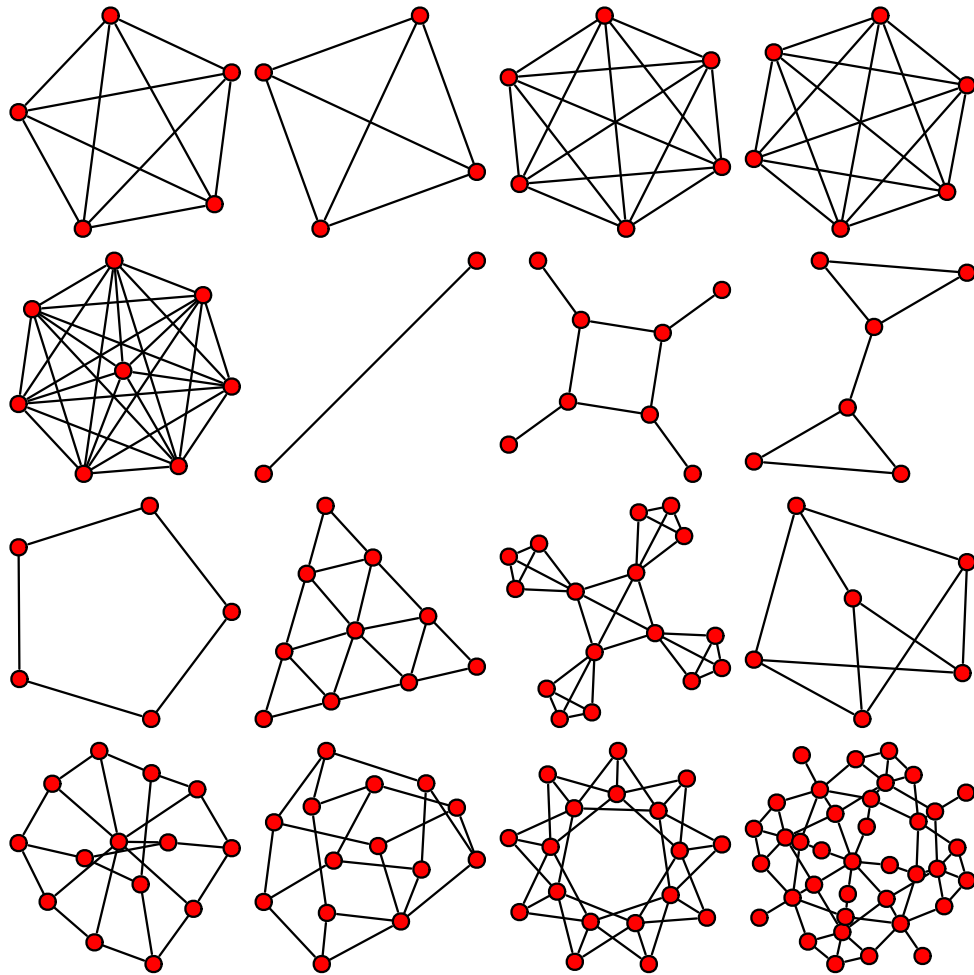
Gábor Csárdi

`csardi@rmki.kfki.hu`

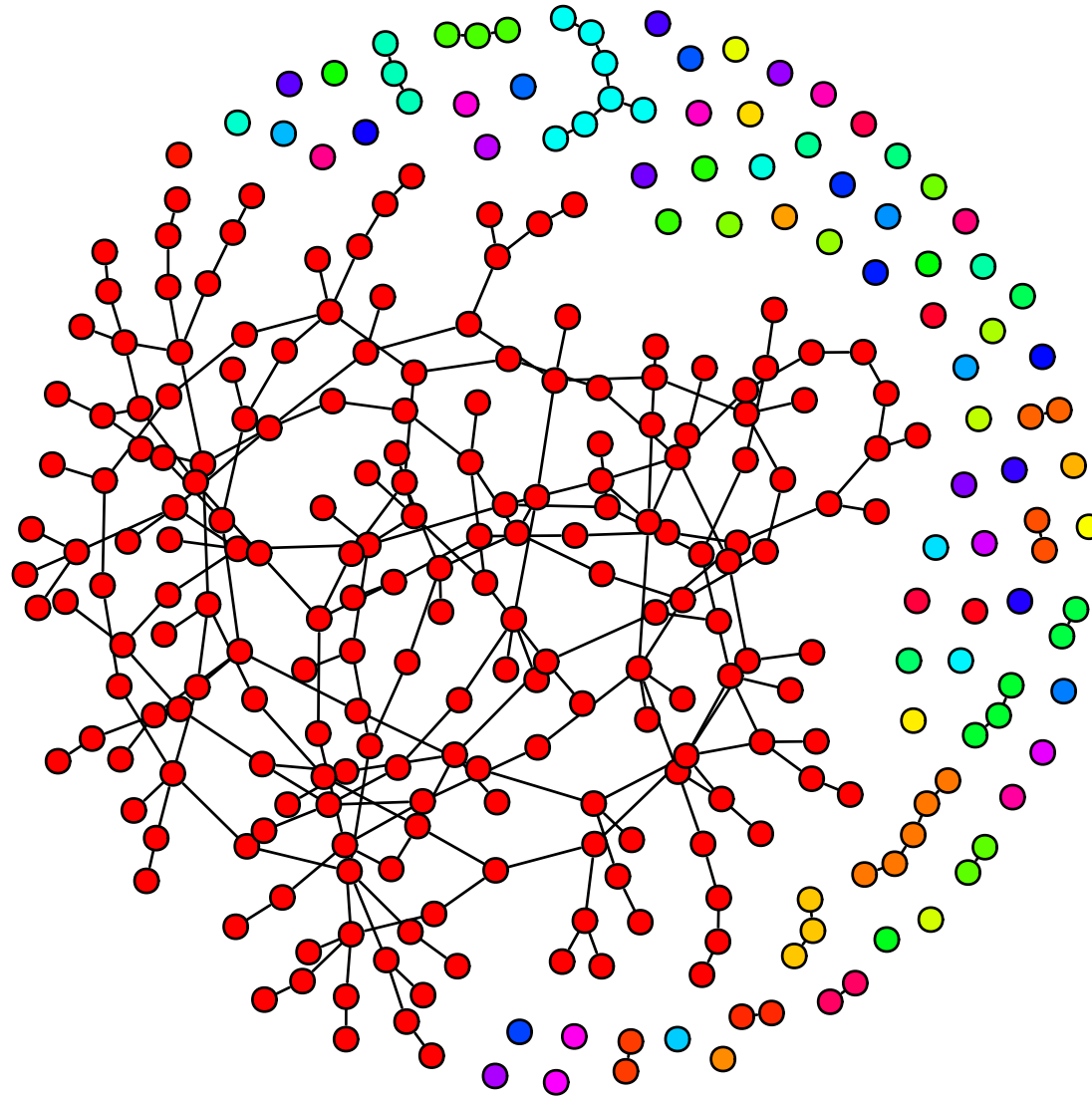
Department of Biophysics, KFKI Research Institute for Nuclear and Particle Physics of the
Hungarian Academy of Sciences, Budapest, Hungary

Currently at
Department of Medical Genetics,
University of Lausanne, Lausanne, Switzerland

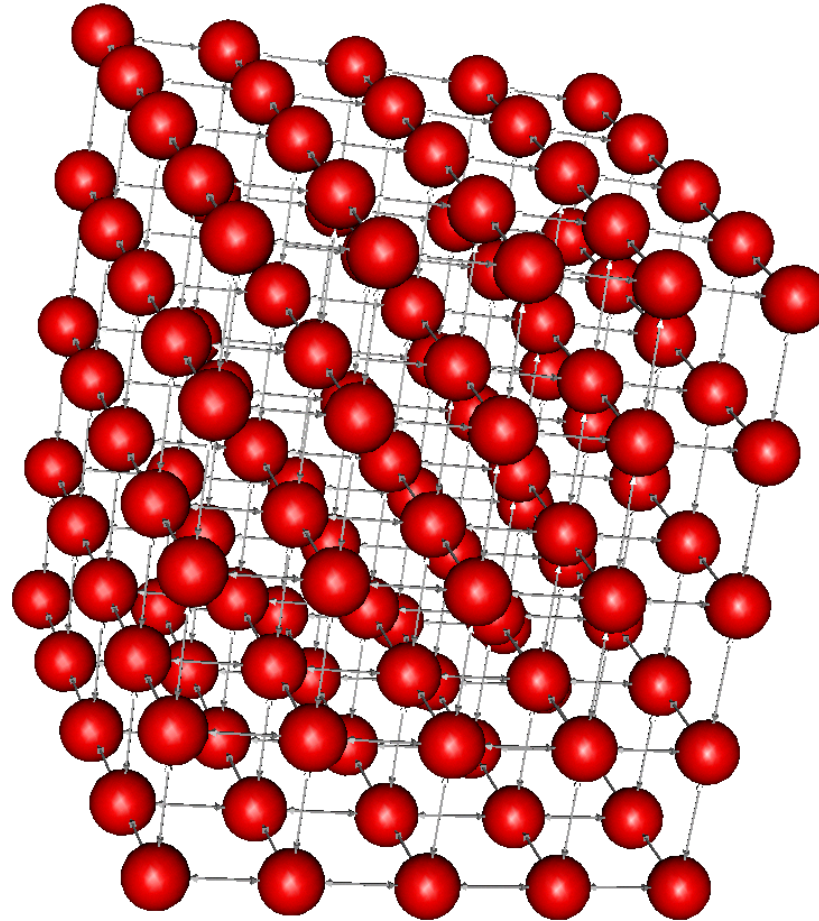
What is a network (or graph)?



What is a network (or graph)?



What is a network (or graph)?



What is a graph?

- Binary relation (=edges) between elements of a set (=vertices).

What is a graph?

- Binary relation (=edges) between elements of a set (=vertices).
- E.g.

vertices = $\{A, B, C, D, E\}$

edges = $(\{A, B\}, \{A, C\}, \{B, C\}, \{C, E\})$.

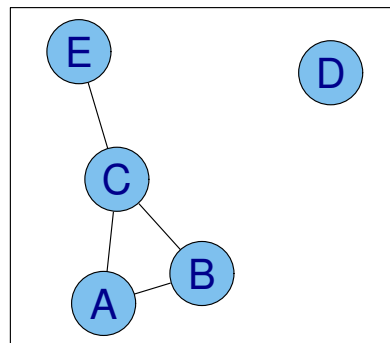
What is a graph?

- Binary relation (=edges) between elements of a set (=vertices).
- E.g.

vertices = $\{A, B, C, D, E\}$

edges = $(\{A, B\}, \{A, C\}, \{B, C\}, \{C, E\})$.

- It is “better” to draw it:



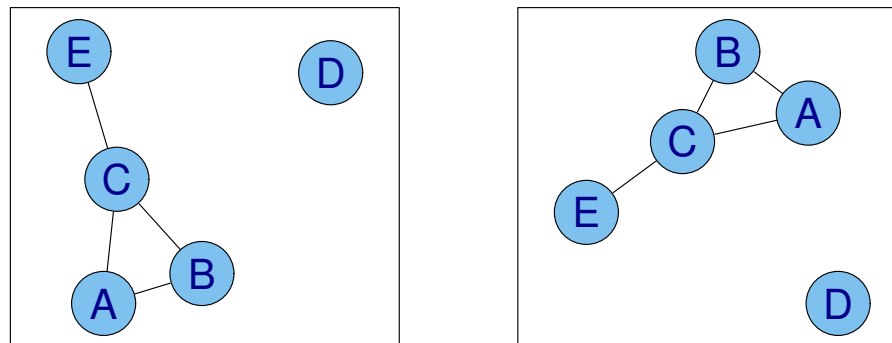
What is a graph?

- Binary relation (=edges) between elements of a set (=vertices).
- E.g.

vertices = $\{A, B, C, D, E\}$

edges = $(\{A, B\}, \{A, C\}, \{B, C\}, \{C, E\})$.

- It is “better” to draw it:

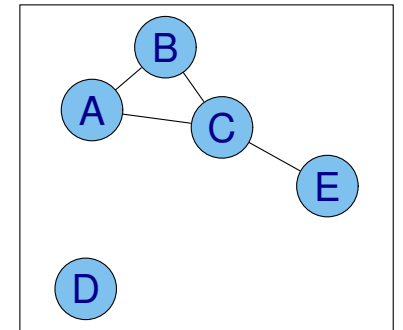


Undirected and directed graphs

- If the pairs are unordered, then the graph is undirected:

vertices = $\{A, B, C, D, E\}$

edges = $(\{A, B\}, \{A, C\}, \{B, C\}, \{C, E\})$.

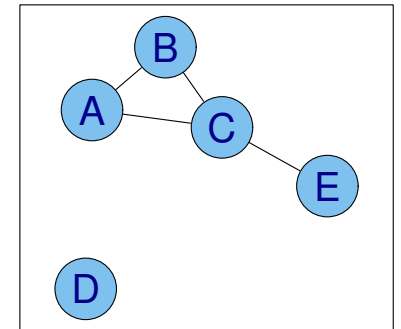


Undirected and directed graphs

- If the pairs are unordered, then the graph is undirected:

vertices = $\{A, B, C, D, E\}$

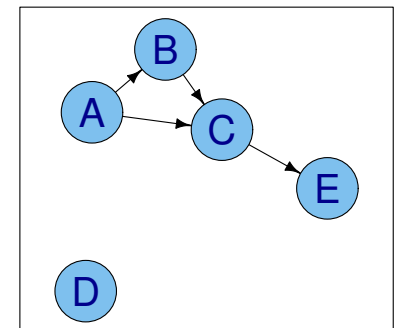
edges = $(\{A, B\}, \{A, C\}, \{B, C\}, \{C, E\})$.



- Otherwise it is directed:

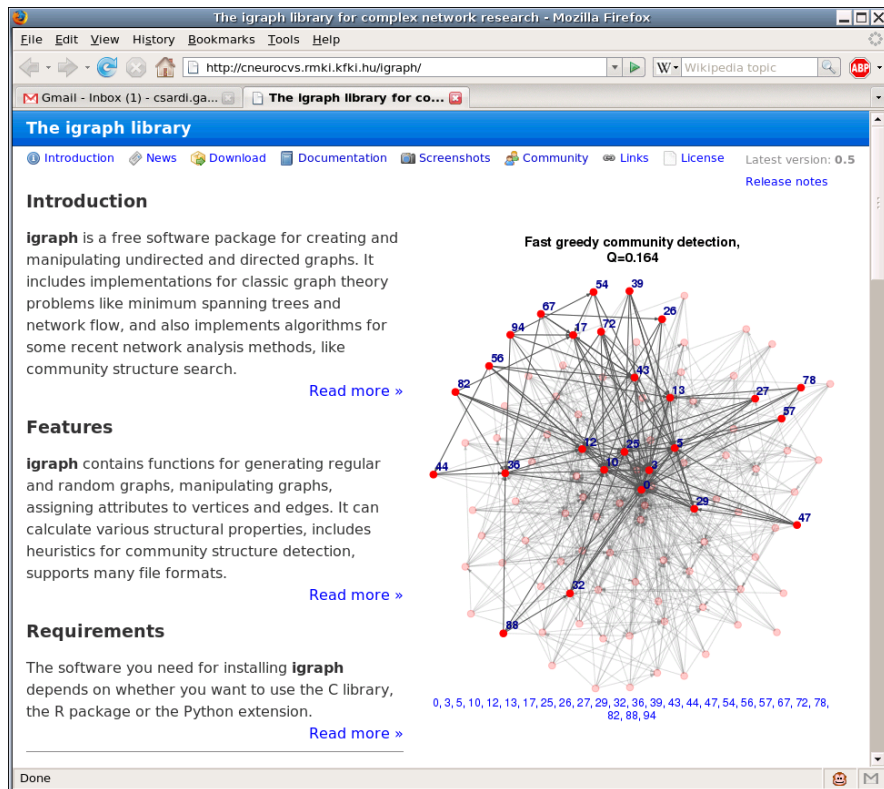
vertices = $\{A, B, C, D, E\}$

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



The igraph “package”

- For classic graph theory and network science.
- Core functionality is implemented as a C library.
- High level interfaces from **R** and **Python**.
- GNU GPL.
- <http://igraph.sf.net>



The screenshot shows a Mozilla Firefox browser window displaying the igraph website. The address bar shows the URL <http://cneurocvs.mki.kfki.hu/igraph/>. The page title is "The igraph library for complex network research - Mozilla Firefox". The website has a blue header with the title "The igraph library" and a navigation menu with links: Introduction, News, Download, Documentation, Screenshots, Community, Links, License, and Latest version: 0.5. Below the header, there is an "Introduction" section with a paragraph about igraph being a free software package for creating and manipulating undirected and directed graphs. It includes implementations for classic graph theory problems like minimum spanning trees and network flow, and also implements algorithms for some recent network analysis methods, like community structure search. There is a "Read more »" link. To the right of the text is a network graph visualization with red nodes and black edges, labeled "Fast greedy community detection, Q=0.164". Below the graph is a list of node IDs: 0, 3, 5, 10, 12, 13, 17, 25, 26, 27, 29, 32, 36, 39, 43, 44, 47, 54, 56, 57, 67, 72, 78, 82, 88, 94. There is also a "Features" section with a paragraph about igraph containing functions for generating regular and random graphs, manipulating graphs, assigning attributes to vertices and edges. It can calculate various structural properties, includes heuristics for community structure detection, supports many file formats. There is a "Read more »" link. Below the features section is a "Requirements" section with a paragraph about the software needed for installing igraph depending on whether you want to use the C library, the R package or the Python extension. There is a "Read more »" link.

Vertex and edge ids

- Vertices are always numbered from zero (!).
- Numbering is continual, from 0 to $|V| - 1$.

Vertex and edge ids

- Vertices are always numbered from zero (!).
- Numbering is continual, from 0 to $|V| - 1$.
- We have to “translate” vertex names to ids:

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

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

$$A = 0, B = 1, C = 2, D = 3, E = 4.$$

Vertex and edge ids

- Vertices are always numbered from zero (!).
- Numbering is continual, from 0 to $|V| - 1$.
- We have to “translate” vertex names to ids:

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

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

$$A = 0, B = 1, C = 2, D = 3, E = 4.$$

```
1 > g <- graph( c(0,1, 0,2, 1,2, 2,4), n=5 )
```

Creating igraph graphs

- igraph objects

Creating igraph graphs

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

Creating igraph graphs

- igraph objects
 - `print()`, `summary()`, `is.igraph()`
 - `is.directed()`, `vcount()`, `ecount()`
-

```
1 > g <- graph( c(0,1, 0,2, 1,2, 2,4), n=5 )
2 > g
3 Vertices: 5
4 Edges: 4
5 Directed: TRUE
6 Edges:
7
8 [0] 0 -> 1
9 [1] 0 -> 2
10 [2] 1 -> 2
11 [3] 2 -> 4
```

Visualization

```
1 > g <- graph.tree(40, 4)
2 > plot(g)
3 > plot(g, layout=layout.circle)
```

Visualization

```
1 > g <- graph.tree(40, 4)
2 > plot(g)
3 > plot(g, layout=layout.circle)
```

```
1 # Force directed layouts
2 > plot(g, layout=layout.fruchterman.reingold)
3 > plot(g, layout=layout.graphopt)
4 > plot(g, layout=layout.kamada.kawai)
```

Visualization

```
1 > g <- graph.tree(40, 4)
2 > plot(g)
3 > plot(g, layout=layout.circle)
```

```
1 # Force directed layouts
2 > plot(g, layout=layout.fruchterman.reingold)
3 > plot(g, layout=layout.graphopt)
4 > plot(g, layout=layout.kamada.kawai)
```

```
1 # Interactive
2 > tkplot(g, layout=layout.kamada.kawai)
3 > l <- layout=layout.kamada.kawai(g)
```

Visualization

```
1 > g <- graph.tree(40, 4)
2 > plot(g)
3 > plot(g, layout=layout.circle)
```

```
1 # Force directed layouts
2 > plot(g, layout=layout.fruchterman.reingold)
3 > plot(g, layout=layout.graphopt)
4 > plot(g, layout=layout.kamada.kawai)
```

```
1 # Interactive
2 > tkplot(g, layout=layout.kamada.kawai)
3 > l <- layout=layout.kamada.kawai(g)
```

```
1 # 3D
2 > rglplot(g, layout=l)
```

Visualization

```
1 > g <- graph.tree(40, 4)
2 > plot(g)
3 > plot(g, layout=layout.circle)
```

```
1 # Force directed layouts
2 > plot(g, layout=layout.fruchterman.reingold)
3 > plot(g, layout=layout.graphopt)
4 > plot(g, layout=layout.kamada.kawai)
```

```
1 # Interactive
2 > tkplot(g, layout=layout.kamada.kawai)
3 > l <- layout=layout.kamada.kawai(g)
```

```
1 # 3D
2 > rglplot(g, layout=l)
```

```
1 # Visual properties
2 > plot(g, layout=l, vertex.color="cyan")
```

Simple graphs

- igraph can handle multi-graphs:

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

$$E = ((AB), (AB), (AC), (BC), (CE)).$$

```
1 > g <- graph( c(0,1,0,1, 0,2, 1,2, 3,4), n=5 )
2 > g
3 Vertices: 5
4 Edges: 5
5 Directed: TRUE
6 Edges:
7
8 [0] 0 -> 1
9 [1] 0 -> 1
10 [2] 0 -> 2
11 [3] 1 -> 2
12 [4] 3 -> 4
```

Simple graphs

- igraph can handle loop-edges:

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

$$E = ((AA), (AB), (AC), (BC), (CE)).$$

```
1 > g <- graph( c(0,0,0,1, 0,2, 1,2, 3,4), n=5 )
2 > g
3 Vertices: 5
4 Edges: 5
5 Directed: TRUE
6 Edges:
7
8 [0] 0 -> 0
9 [1] 0 -> 1
10 [2] 0 -> 2
11 [3] 1 -> 2
12 [4] 3 -> 4
```

Creating (more) igraph graphs

```
1  el <- scan("lesmis.txt")
2  el <- matrix(el, byrow=TRUE, nc=2)
3  gmis <- graph.edgelist(el, dir=FALSE)
4  summary(gmis)
```

Naming vertices

```
1 V(gmis)$name  
2 g <- graph.ring(10)  
3 V(g)$name <- sample(letters, vcount(g))
```

Creating (more) igraph graphs

```
1 # A simple undirected graph
2 > g <- graph.formula( Alice-Bob-Cecil-Alice,
3   Daniel-Cecil-Eugene, Cecil-Gordon )
```

Creating (more) igraph graphs

```
1 # A simple undirected graph
2 > g <- graph.formula( Alice-Bob-Cecil-Alice,
3   Daniel-Cecil-Eugene, Cecil-Gordon )
```

```
1 # Another undirected graph, ":" notation
2 > g2 <- graph.formula( Alice-Bob:Cecil:Daniel,
3   Cecil:Daniel-Eugene:Gordon )
```

Creating (more) igraph graphs

```
1 # A simple undirected graph
2 > g <- graph.formula( Alice-Bob-Cecil-Alice,
3   Daniel-Cecil-Eugene, Cecil-Gordon )
```

```
1 # Another undirected graph, ":" notation
2 > g2 <- graph.formula( Alice-Bob:Cecil:Daniel,
3   Cecil:Daniel-Eugene:Gordon )
```

```
1 # A directed graph
2 > g3 <- graph.formula( Alice +--+ Bob --+ Cecil
3   +-- Daniel, Eugene --+ Gordon:Helen )
```

Creating (more) igraph graphs

```
1 # A simple undirected graph
2 > g <- graph.formula( Alice-Bob-Cecil-Alice,
3   Daniel-Cecil-Eugene, Cecil-Gordon )
```

```
1 # Another undirected graph, ":" notation
2 > g2 <- graph.formula( Alice-Bob:Cecil:Daniel,
3   Cecil:Daniel-Eugene:Gordon )
```

```
1 # A directed graph
2 > g3 <- graph.formula( Alice +--+ Bob --+ Cecil
3   +-- Daniel, Eugene --+ Gordon:Helen )
```

```
1 # A graph with isolate vertices
2 > g4 <- graph.formula( Alice -- Bob -- Daniel,
3   Cecil:Gordon, Helen )
```

Creating (more) igraph graphs

```
1 # A simple undirected graph
2 > g <- graph.formula( Alice-Bob-Cecil-Alice,
3   Daniel-Cecil-Eugene, Cecil-Gordon )
```

```
1 # Another undirected graph, ":" notation
2 > g2 <- graph.formula( Alice-Bob:Cecil:Daniel,
3   Cecil:Daniel-Eugene:Gordon )
```

```
1 # A directed graph
2 > g3 <- graph.formula( Alice +--+ Bob --+ Cecil
3   +-- Daniel, Eugene --+ Gordon:Helen )
```

```
1 # A graph with isolate vertices
2 > g4 <- graph.formula( Alice -- Bob -- Daniel,
3   Cecil:Gordon, Helen )
```

```
1 # "Arrows" can be arbitrarily long
2 > g5 <- graph.formula( Alice +-----+ Bob )
```

Vertex/Edge sets, attributes

- Assigning attributes:
`set/get.graph/vertex/edge.attribute.`

Vertex/Edge sets, attributes

- Assigning attributes:
set/get.graph/vertex/edge.attribute.
- $V(g)$ and $E(g)$.

Vertex/Edge sets, attributes

- Assigning attributes:
`set/get.graph/vertex/edge.attribute.`
- $V(g)$ and $E(g)$.
- Smart indexing, e.g.
`V(g)[color=="white"]`

Vertex/Edge sets, attributes

- Assigning attributes:
set/get.graph/vertex/edge.attribute.
- $V(g)$ and $E(g)$.
- Smart indexing, e.g.
 $V(g)[\text{color} == \text{"white"}]$
- Easy access of attributes:

```
1 > g <- erdos.renyi.game(100, 1/100)
2 > V(g)$color <- sample( c("red", "black"),
3                          vcount(g), rep=TRUE)
4 > E(g)$color <- "grey"
5 > red <- V(g)[ color == "red" ]
6 > bl <- V(g)[ color == "black" ]
7 > E(g)[ red %--% red ]$color <- "red"
8 > E(g)[ bl  %--% bl ]$color <- "black"
9 > plot(g, vertex.size=5, layout=
10      layout.fruchterman.reingold)
```

Creating (even) more graphs

- E.g. from .csv files.

```
1 > traits <- read.csv("traits.csv", head=F)
2 > relations <- read.csv("relations.csv", head=F)
3 > orgnet <- graph.data.frame(relations)
4
5 > traits[,1] <- sapply(strsplit(as.character
6   (traits[,1]), split=" "), "[", 1)
7 > idx <- match(V(orgnet)$name, traits[,1])
8 > V(orgnet)$gender <- as.character(traits[,3][idx])
9 > V(orgnet)$age <- traits[,2][idx]
10
11 > igraph.par("print.vertex.attributes", TRUE)
12 > orgnet
```

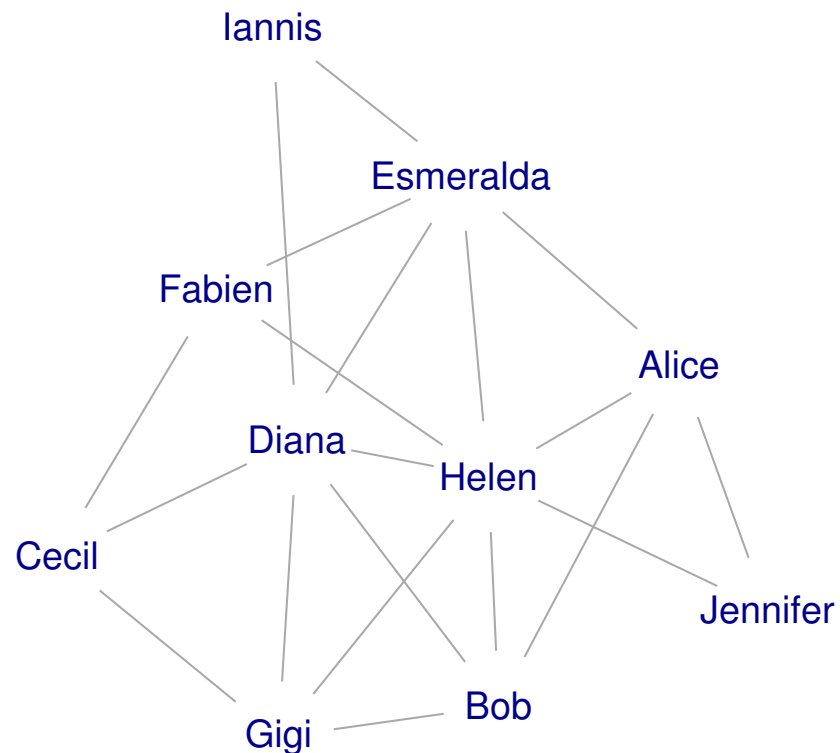
Creating (even) more graphs

- From the web, e.g. Pajek files.

```
1 > karate <- read.graph("http://cneurocv.s.rmki.kfki.hu/igraph/karate.net",
2                       format="pajek")
3 > summary(karate)
4 Vertices: 34
5 Edges: 78
6 Directed: FALSE
7 No graph attributes.
8 No vertex attributes.
9 No edge attributes.
```

Graph representation

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



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

- Edge list. Not really good for anything.

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

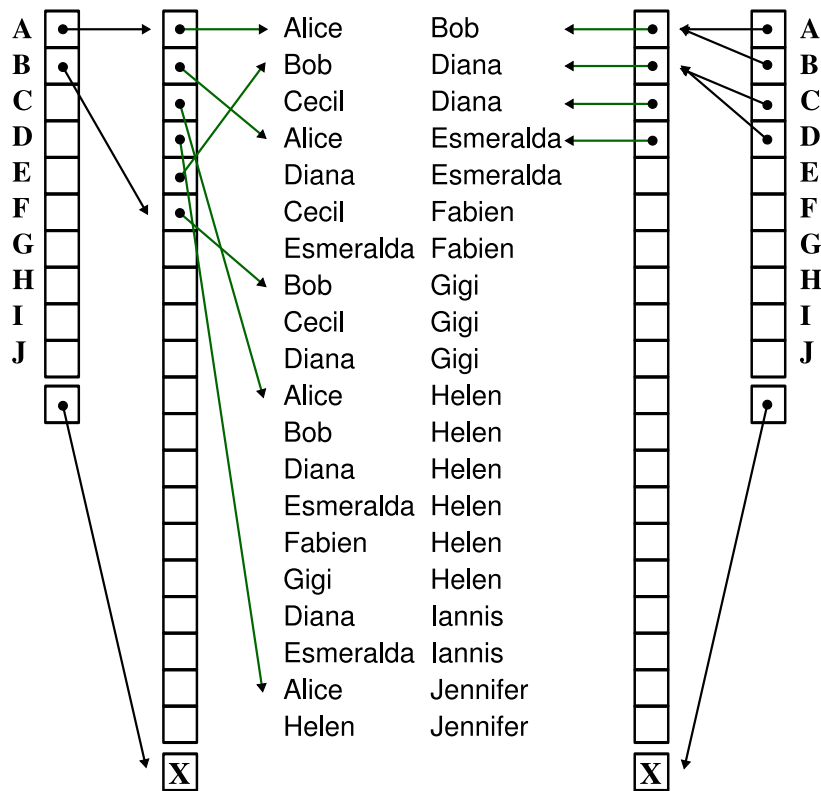
Graph representation

- Adjacency lists. GQ: who are the neighbors of 'Alice' ?

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

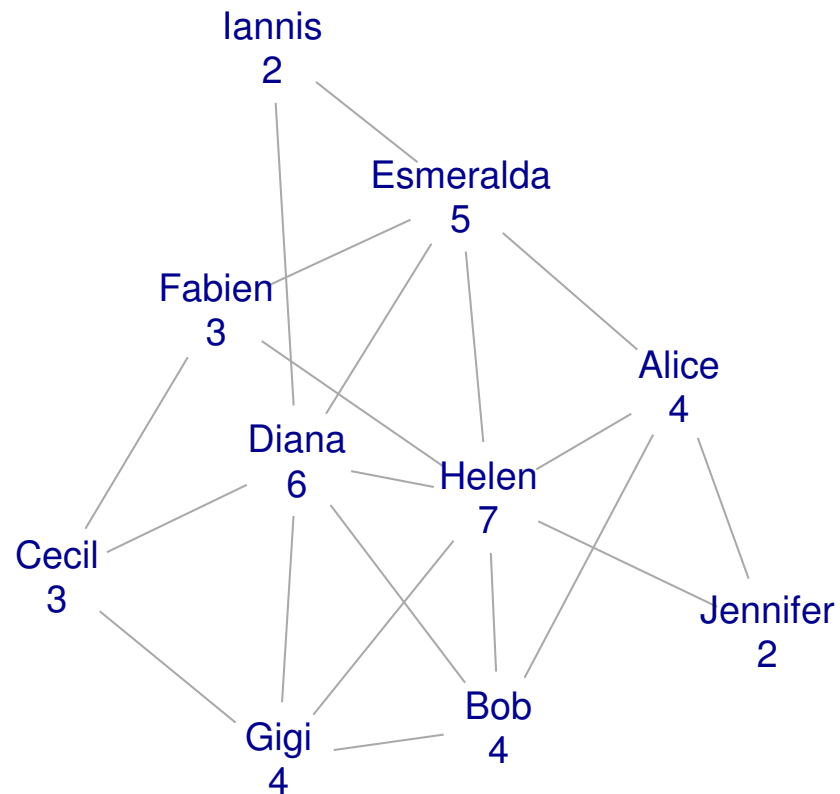
Graph representation

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



Centrality in networks

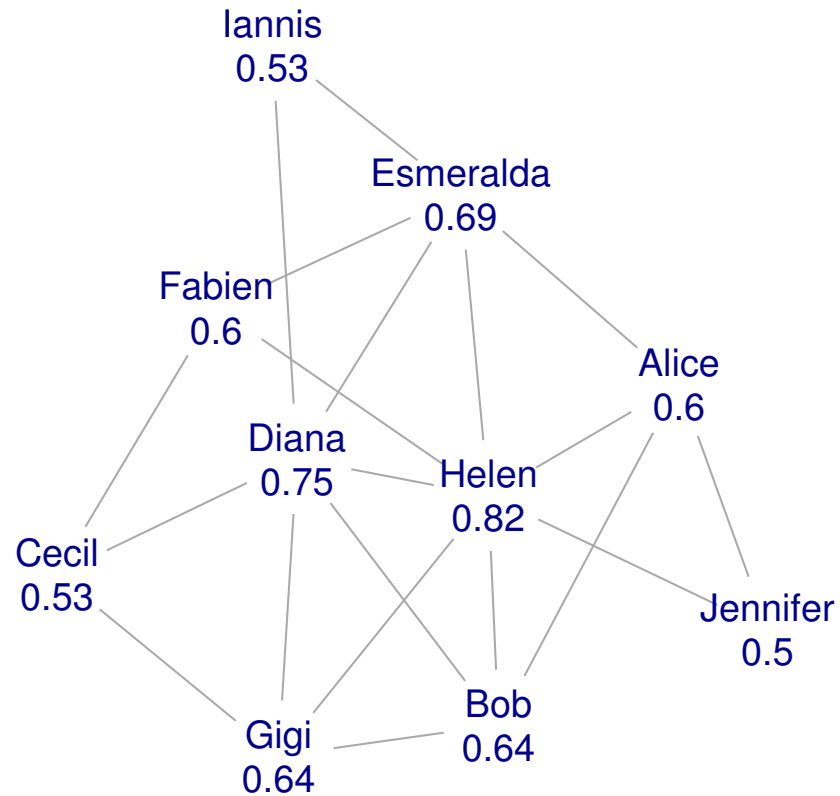
- degree



Centrality in networks

- closeness

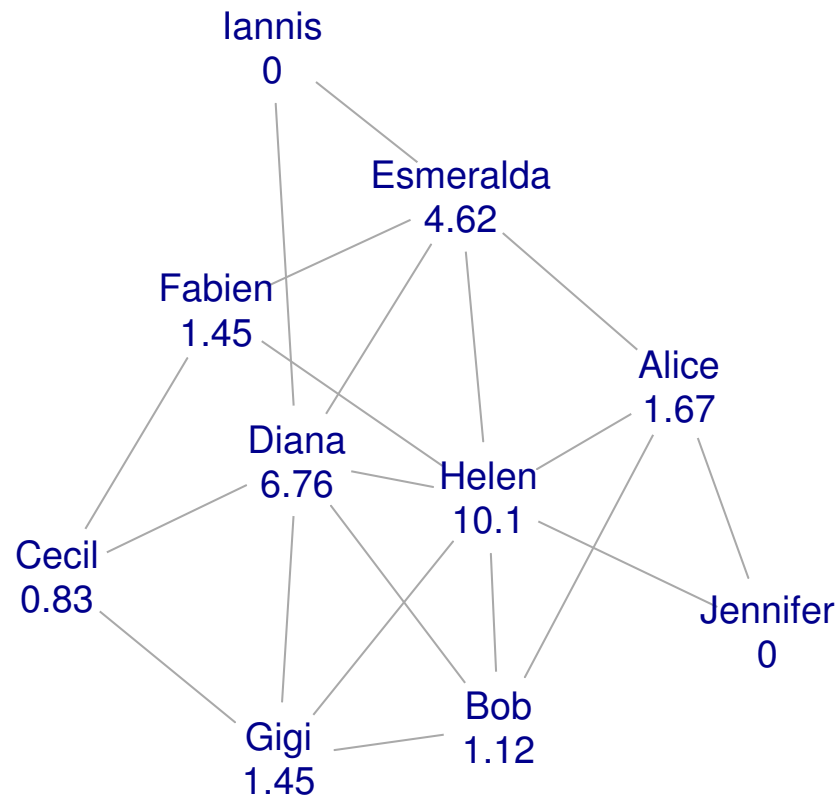
$$C_v = \frac{|V| - 1}{\sum_{i \neq v} d_{vi}}$$



Centrality in networks

- betweenness

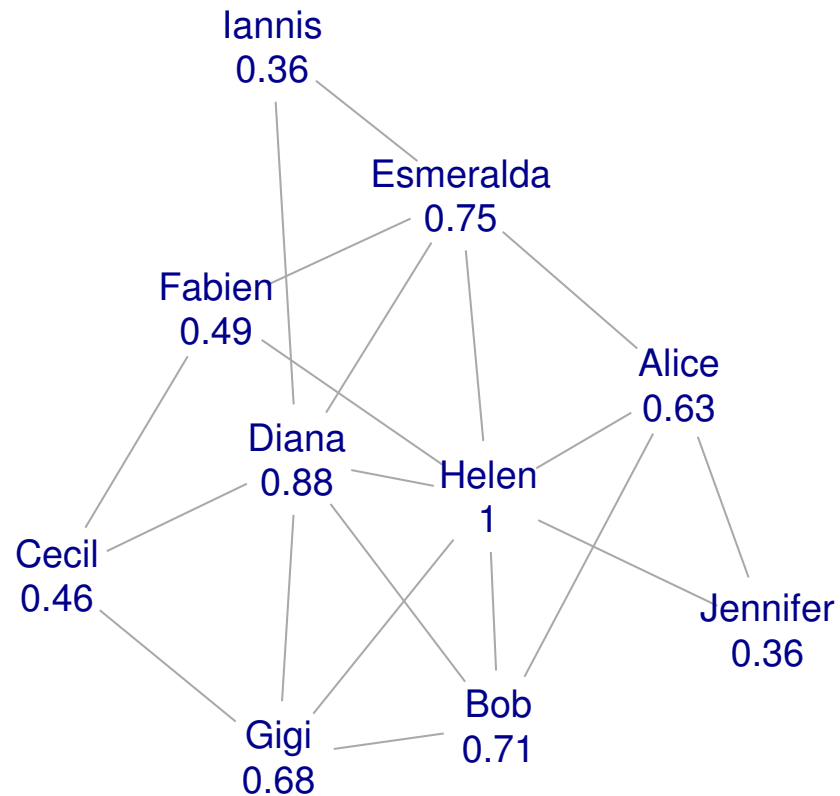
$$B_v = \sum_{i \neq j, i \neq v, j \neq v} g_{ivj} / g_{ij}$$



Centrality in networks

- eigenvector centrality

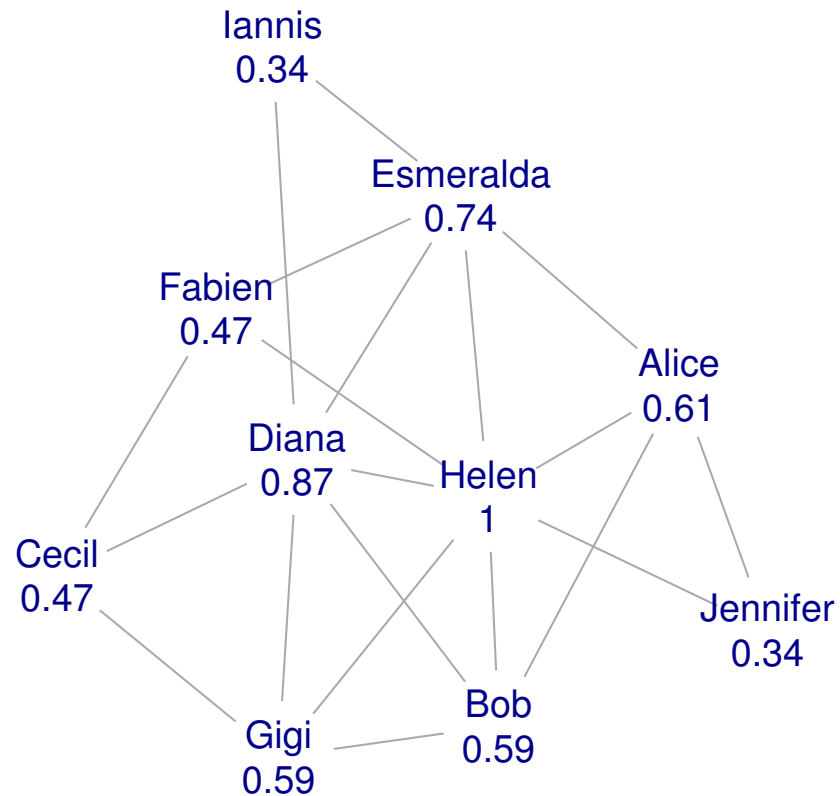
$$E_v = \frac{1}{\lambda} \sum_{i=1}^{|V|} A_{iv} E_i, \quad Ax = \lambda x$$



Centrality in networks

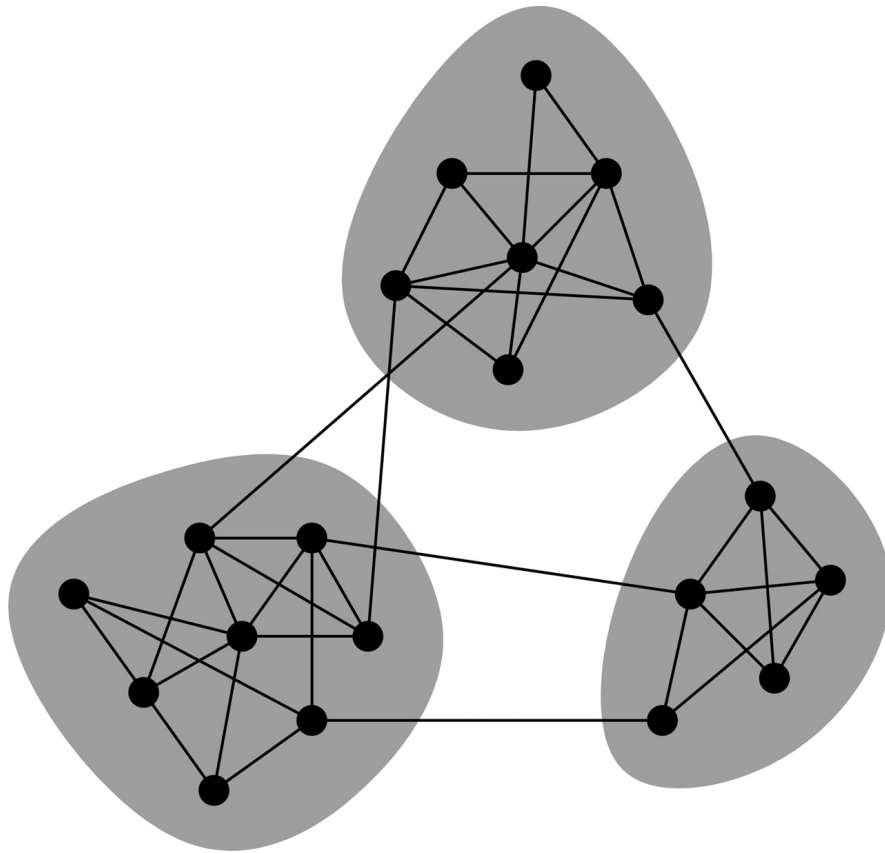
- page rank

$$E_v = \frac{1 - d}{|V|} + d \sum_{i=1}^{|V|} A_{iv} E_i$$



Community structure in networks

- Organizing things, clustering items to see the structure.



M. E. J. Newman, PNAS, 103, 8577–8582

Community structure in networks

- How to define what is modular?
Many proposed definitions, here is a popular one:

$$Q = \frac{1}{2|E|} \sum_{vw} [A_{vw} - p_{vw}] \delta(c_v, c_w).$$

Community structure in networks

- How to define what is modular?
Many proposed definitions, here is a popular one:

$$Q = \frac{1}{2|E|} \sum_{vw} [A_{vw} - p_{vw}] \delta(c_v, c_w).$$

- Random graph null model:

$$p_{vw} = p = \frac{1}{|V|(|V| - 1)}$$

Community structure in networks

- How to define what is modular?
Many proposed definitions, here is a popular one:

$$Q = \frac{1}{2|E|} \sum_{vw} [A_{vw} - p_{vw}] \delta(c_v, c_w).$$

- Random graph null model:

$$p_{vw} = p = \frac{1}{|V|(|V| - 1)}$$

- Degree sequence based null model:

$$p_{vw} = \frac{k_v k_w}{2|E|}$$

Cohesive blocks

(Based on 'Structural Cohesion and Embeddedness: a Hierarchical Concept of Social Groups' by J.Moody and D.White, American Sociological Review, 68, 103–127, 2003)

Definition 1: A collectivity is structurally cohesive to the extent that the social relations of its members hold it together.

Cohesive blocks

(Based on 'Structural Cohesion and Embeddedness: a Hierarchical Concept of Social Groups' by J.Moody and D.White, American Sociological Review, 68, 103–127, 2003)

Definition 1: A collectivity is structurally cohesive to the extent that the social relations of its members hold it together.

Definition 2: A group is structurally cohesive to the extent that multiple independent relational paths among all pairs of members hold it together.

Cohesive blocks

(Based on 'Structural Cohesion and Embeddedness: a Hierarchical Concept of Social Groups' by J.Moody and D.White, American Sociological Review, 68, 103–127, 2003)

Definition 1: A collectivity is structurally cohesive to the extent that the social relations of its members hold it together.

Definition 2: A group is structurally cohesive to the extent that multiple independent relational paths among all pairs of members hold it together.

- Vertex-independent paths and vertex connectivity.

Cohesive blocks

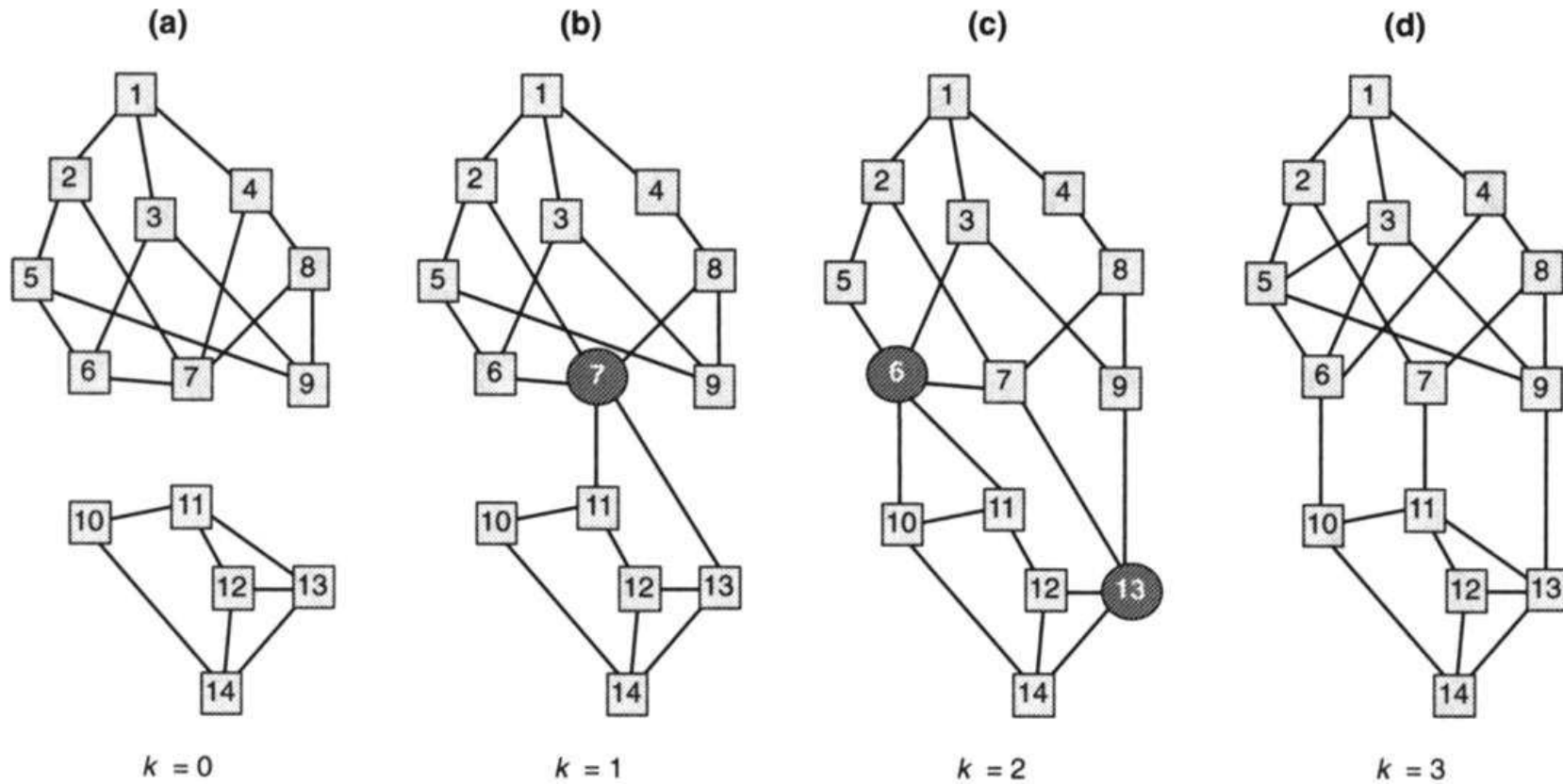
(Based on 'Structural Cohesion and Embeddedness: a Hierarchical Concept of Social Groups' by J.Moody and D.White, American Sociological Review, 68, 103–127, 2003)

Definition 1: A collectivity is structurally cohesive to the extent that the social relations of its members hold it together.

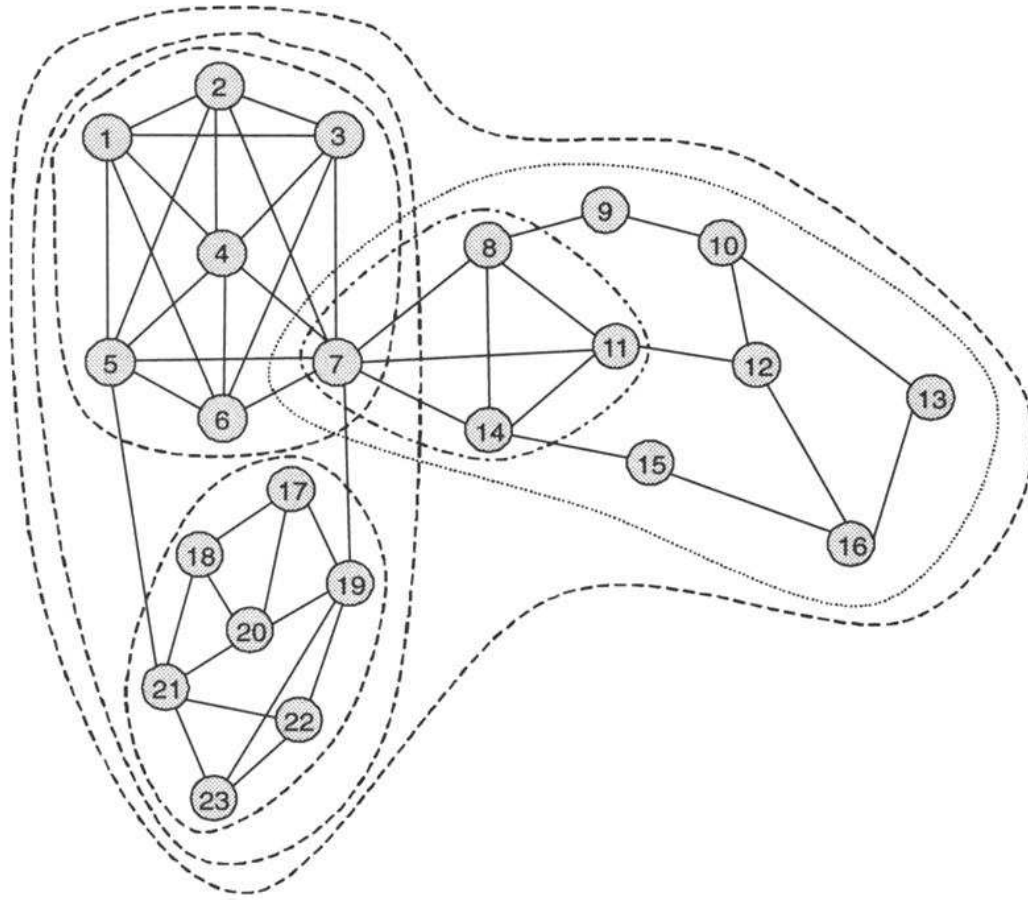
Definition 2: A group is structurally cohesive to the extent that multiple independent relational paths among all pairs of members hold it together.

- Vertex-independent paths and vertex connectivity.
- Vertex connectivity and network flows.

Cohesive blocks



Cohesive blocks



Rapid prototyping

Weighted transitivity

$$c(i) = \frac{\mathbf{A}_{ii}^3}{(\mathbf{A}\mathbf{1}\mathbf{A})_{ii}}$$

Rapid prototyping

Weighted transitivity

$$c(i) = \frac{\mathbf{A}_{ii}^3}{(\mathbf{A}\mathbf{1}\mathbf{A})_{ii}}$$

$$c_w(i) = \frac{\mathbf{W}_{ii}^3}{(\mathbf{W}\mathbf{W}_{\max}\mathbf{W})_{ii}}$$

Rapid prototyping

Weighted transitivity

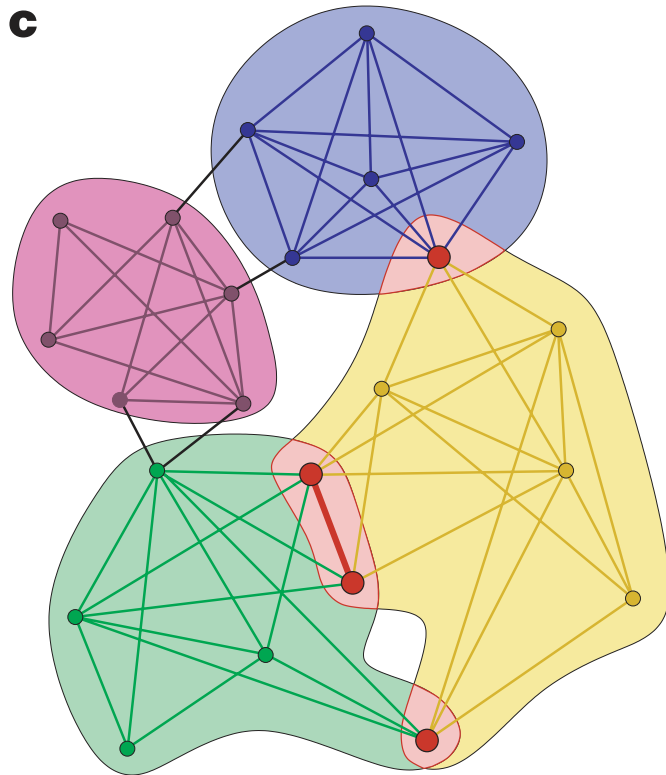
$$c(i) = \frac{\mathbf{A}_{ii}^3}{(\mathbf{A}\mathbf{1}\mathbf{A})_{ii}}$$

$$c_w(i) = \frac{\mathbf{W}_{ii}^3}{(\mathbf{W}\mathbf{W}_{\max}\mathbf{W})_{ii}}$$

```
1 wtrans <- function(g) {  
2   W <- get.adjacency(g, attr="weight")  
3   WM <- matrix(max(W), nrow(W), ncol(W))  
4   diag(WM) <- 0  
5   diag( W %*% W %*% W ) /  
6     diag( W %*% WM %*% W )  
7 }
```

Rapid prototyping

Clique percolation (Palla et al., Nature 435, 814, 2005)



... and the rest

- Cliques and independent vertex sets.
- Network flows.
- Motifs, i.e. dyad and triad census.
- Random graph generators.
- Graph isomorphism.
- Vertex similarity measures, topological sorting, spanning trees, graph components, K-cores, transitivity or clustering coefficient.
- etc.
- C-level: rich data type library.

Acknowledgement

Tamás Nepusz

All the people who contributed code, sent bug reports, suggestions

The R project

Hungarian Academy of Sciences

The OSS community in general