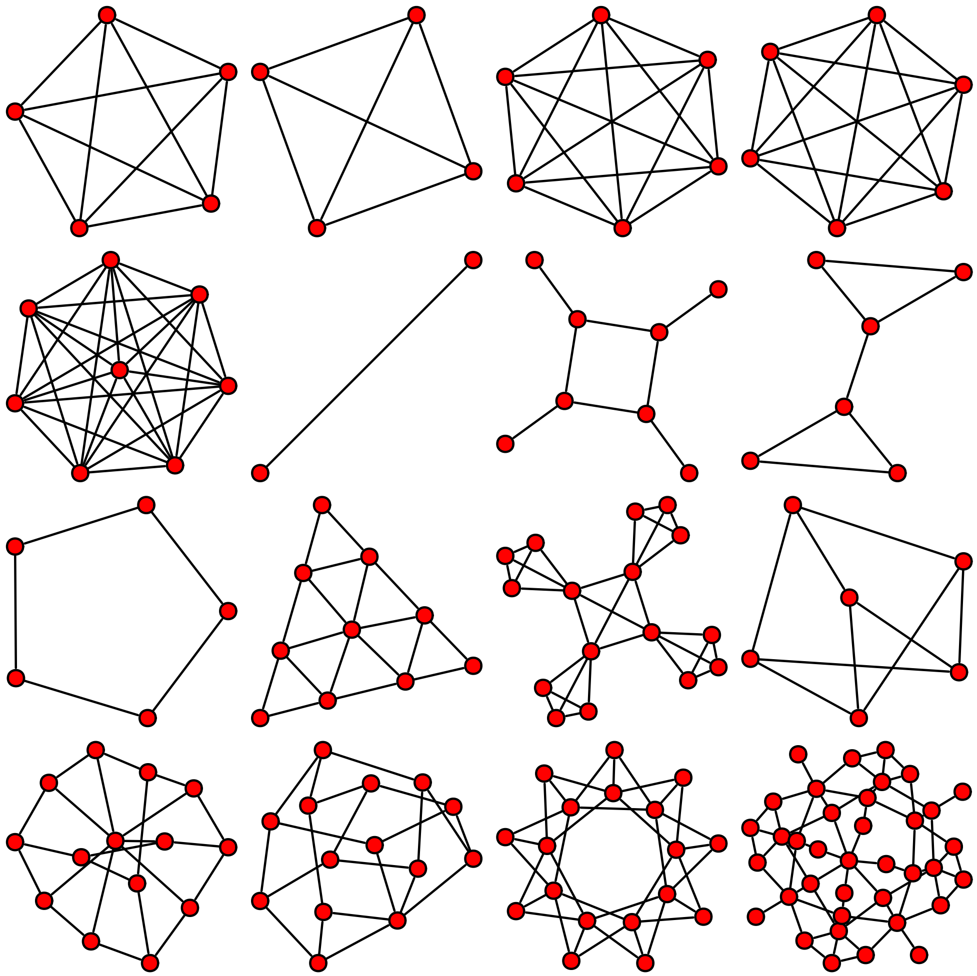
Practical statistical network analysis (with R and `igraph`)

Gábor Csárdi
csardi@rmki.kfki.hu

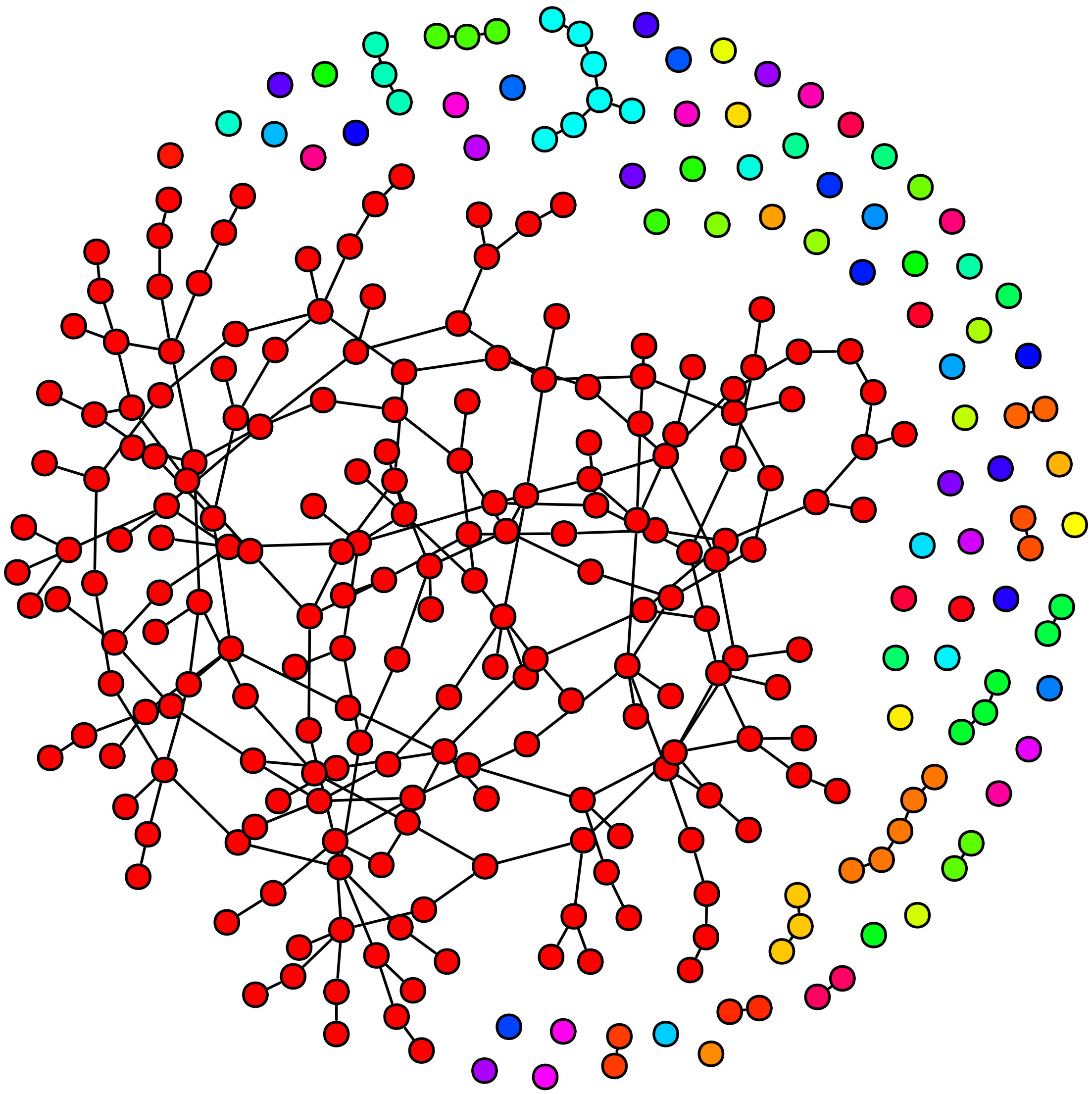
Department of Biophysics, KFKI Research Institute for Nuclear and Particle Physics of the Hungarian Academy of Sciences, Budapest, Hungary

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

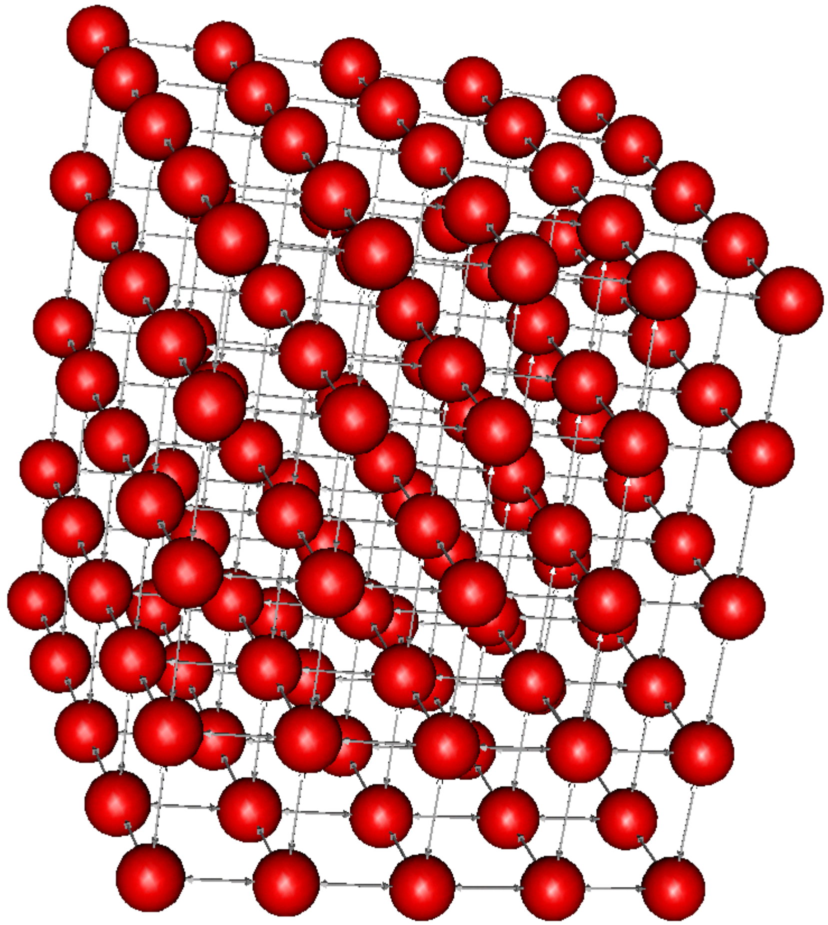
What is a network (or graph)?



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.

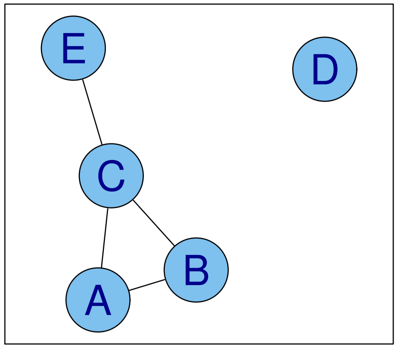
$$\text{vertices} = \{A, B, C, D, E\}$$
$$\text{edges} = (\{A, B\}, \{A, C\}, \{B, C\}, \{C, E\}).$$

What is a graph?

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

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

- It is "better" to draw it:

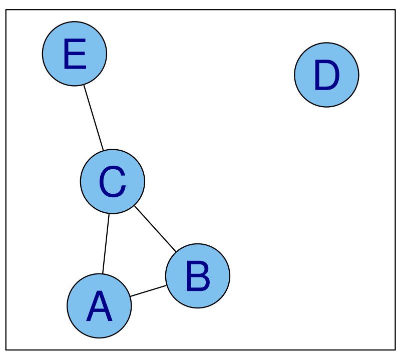


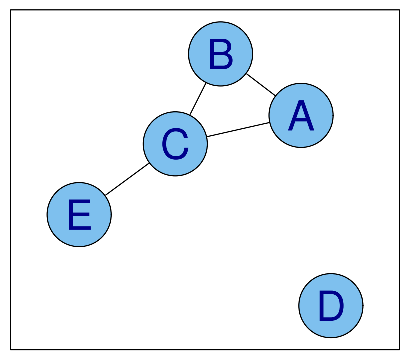
What is a graph?

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

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

- It is "better" to draw it:

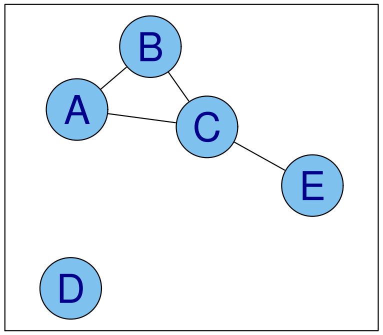




Undirected and directed graphs

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

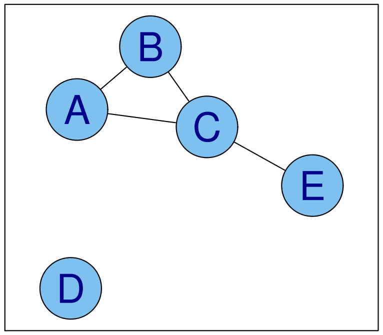
$$\text{vertices} = \{A, B, C, D, E\}$$
$$\text{edges} = (\{A, B\}, \{A, C\}, \{B, C\}, \{C, E\}).$$



Undirected and directed graphs

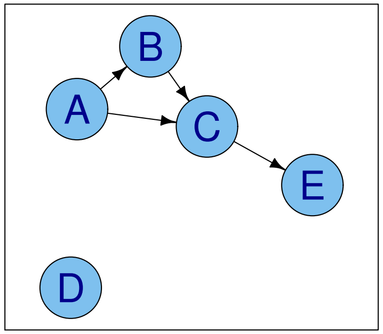
- If the pairs are unordered, then the graph is undirected:

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



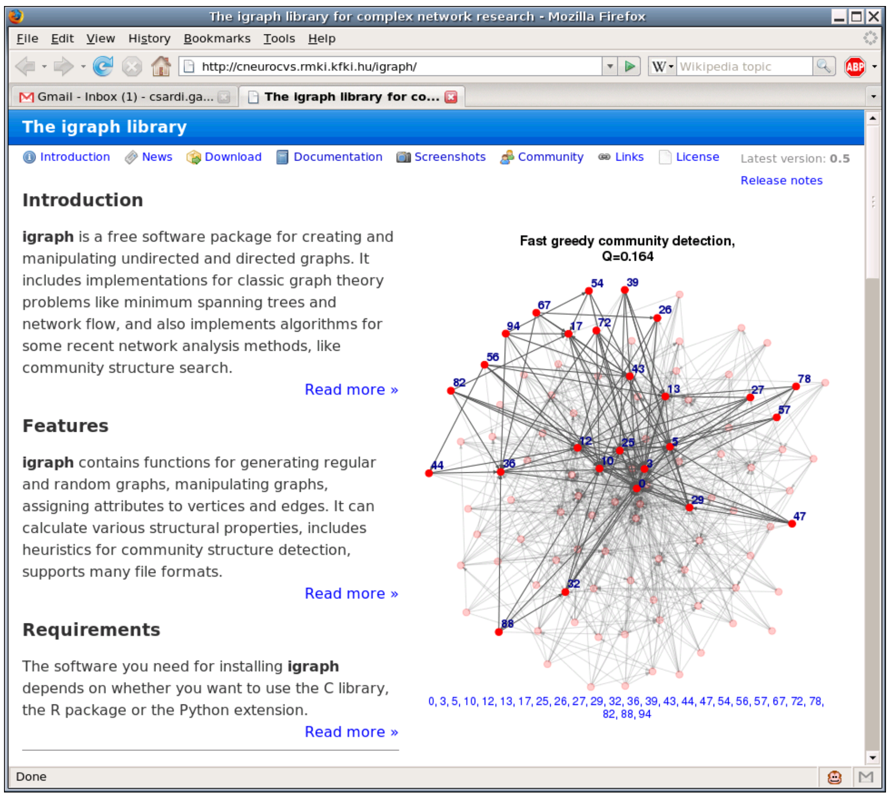
- Otherwise it is directed:

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



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`



Vertex and edge ids

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

Vertex and edge ids

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

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

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

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

```
> 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()`

```
> g <- graph( c(0,1, 0,2, 1,2, 2,4), n=5 )
> g
Vertices: 5
Edges: 4
Directed: TRUE
Edges:

[0] 0 -> 1
[1] 0 -> 2
[2] 1 -> 2
[3] 2 -> 4
```

Visualization

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

Visualization

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

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

Visualization

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

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

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

Visualization

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

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

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

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

Visualization

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

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

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

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

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

Simple graphs

- `igraph` can handle multi-graphs:

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

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

```
> g <- graph( c(0,1,0,1, 0,2, 1,2, 3,4), n=5 )
> g
Vertices: 5
Edges: 5
Directed: TRUE
Edges:

[0] 0 -> 1
[1] 0 -> 1
[2] 0 -> 2
[3] 1 -> 2
[4] 3 -> 4
```

Simple graphs

- `igraph` can handle loop-edges:

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

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

```
> g <- graph( c(0,0,0,1, 0,2, 1,2, 3,4), n=5 )
> g
Vertices: 5
Edges: 5
Directed: TRUE
Edges:

[0] 0 -> 0
[1] 0 -> 1
[2] 0 -> 2
[3] 1 -> 2
[4] 3 -> 4
```

Creating (more) igraph graphs

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

Naming vertices

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

Creating (more) igraph graphs

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

Creating (more) igraph graphs

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

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

Creating (more) igraph graphs

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

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

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

Creating (more) igraph graphs

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

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

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

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

Creating (more) igraph graphs

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

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

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

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

```
# "Arrows" can be arbitrarily long
> 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)[color=="white"]`
- Easy access of attributes:

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

Creating (even) more graphs

- E.g. from `.csv` files.

```
> traits <- read.csv("traits.csv", head=F)
> relations <- read.csv("relations.csv", head=F)
> orgnet <- graph.data.frame(relations)

> traits[,1] <- sapply(strsplit(as.character
  (traits[,1]), split=" "), "[[", 1)
> idx <- match(V(orgnet)$name, traits[,1])
> V(orgnet)$gender <- as.character(traits[,3][idx])
> V(orgnet)$age <- traits[,2][idx]

> igraph.par("print.vertex.attributes", TRUE)
> orgnet
```

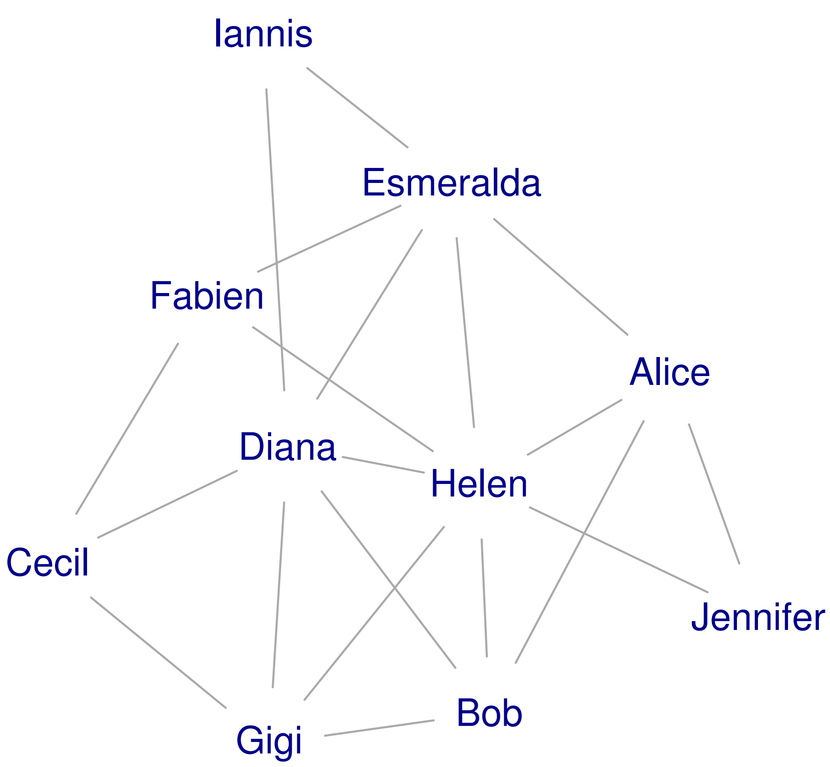
Creating (even) more graphs

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

```
> karate <- read.graph("http://cneurocvs.rmki.kfki.hu/igraph/karate.net",
                       format="pajek")
> summary(karate)
Vertices: 34
Edges: 78
Directed: FALSE
No graph attributes.
No vertex attributes.
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

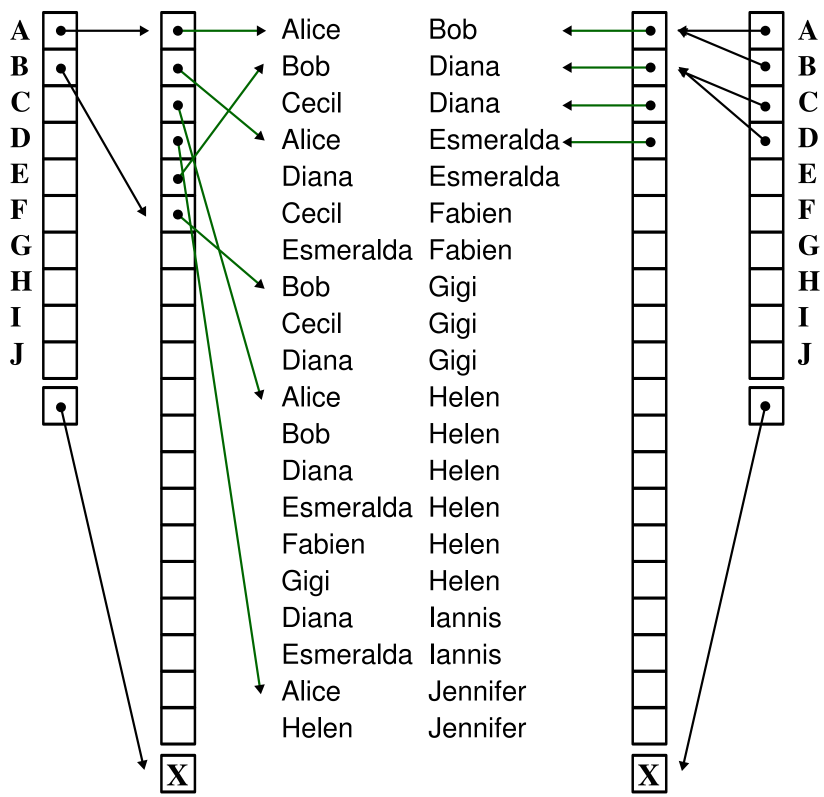
Graph representation

- 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

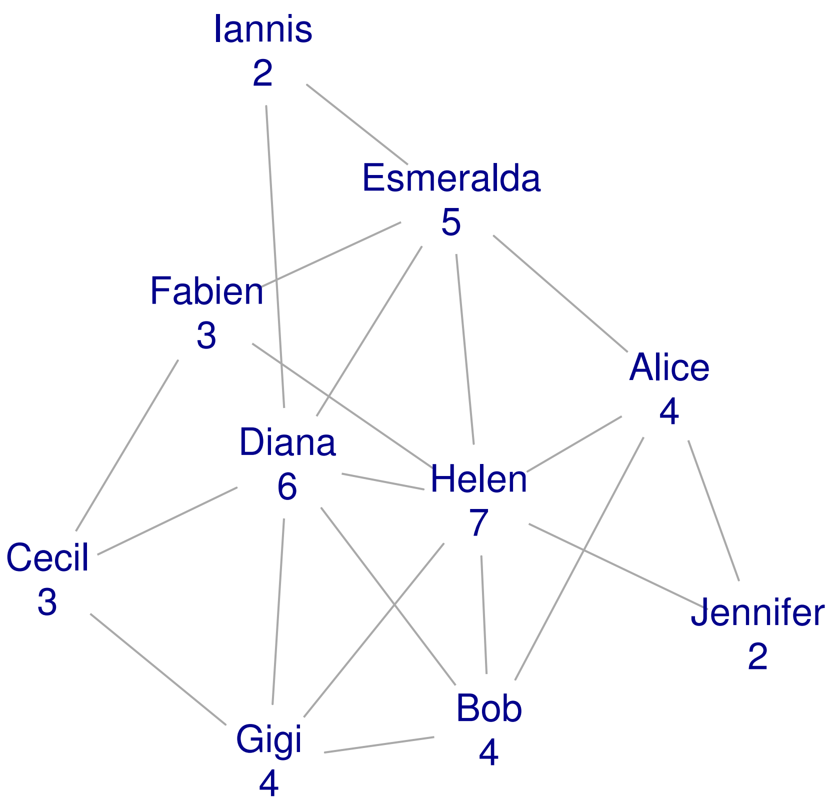
Graph representation

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



Centrality in networks

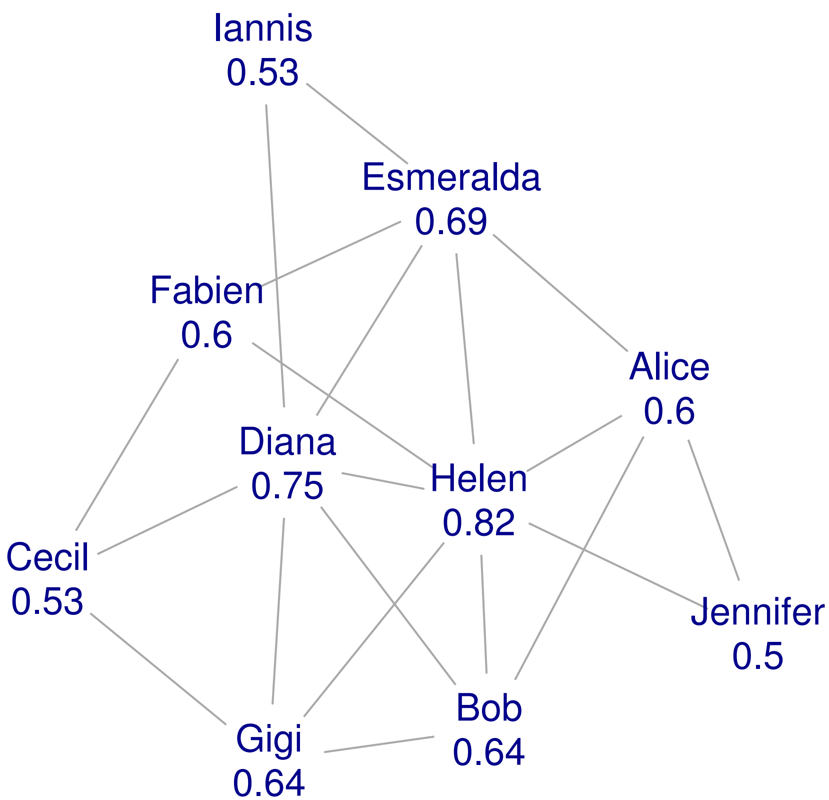
- degree



Centrality in networks

- closeness

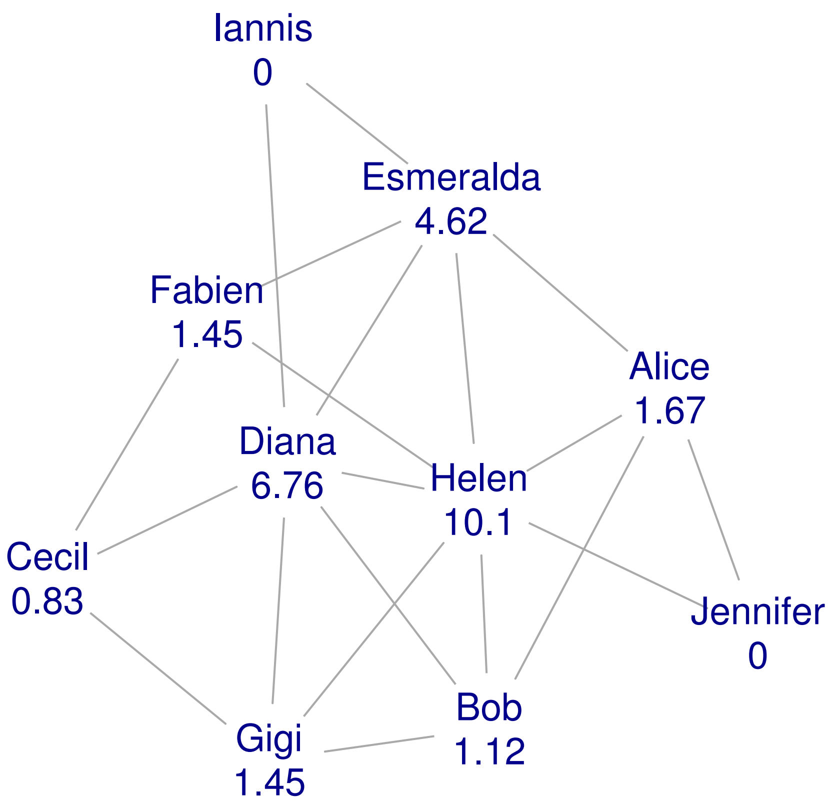
$$C_v = \frac{|V| - 1}{\sum_{i \neq v} d_{vi}}$$



Centrality in networks

- betweenness

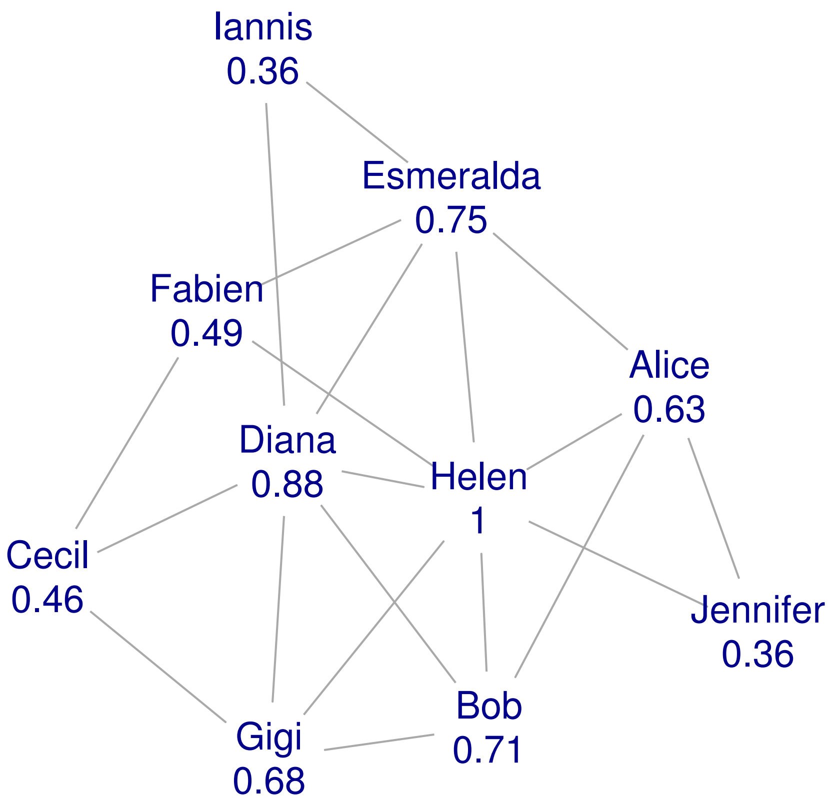
$$B_v = \sum_{i \neq j, i \neq v, j \neq v} g_{ivj}/g_{ij}$$



Centrality in networks

- eigenvector centrality

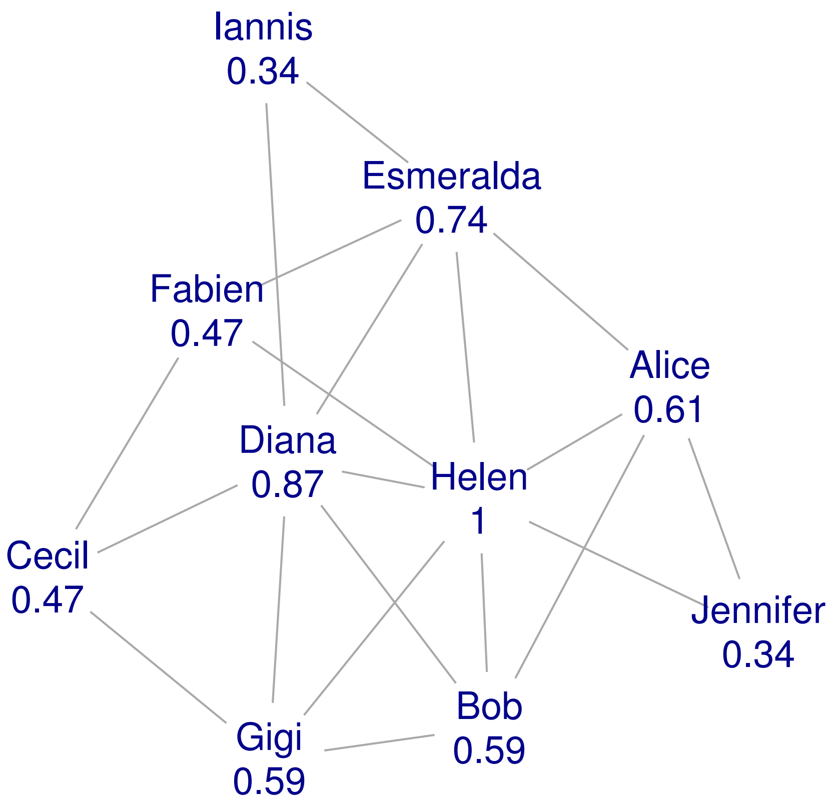
$$E_v = \frac{1}{\lambda} \sum_{i=1}^{|V|} A_{iv} E_i, \quad Ax = \lambda x$$



Centrality in networks

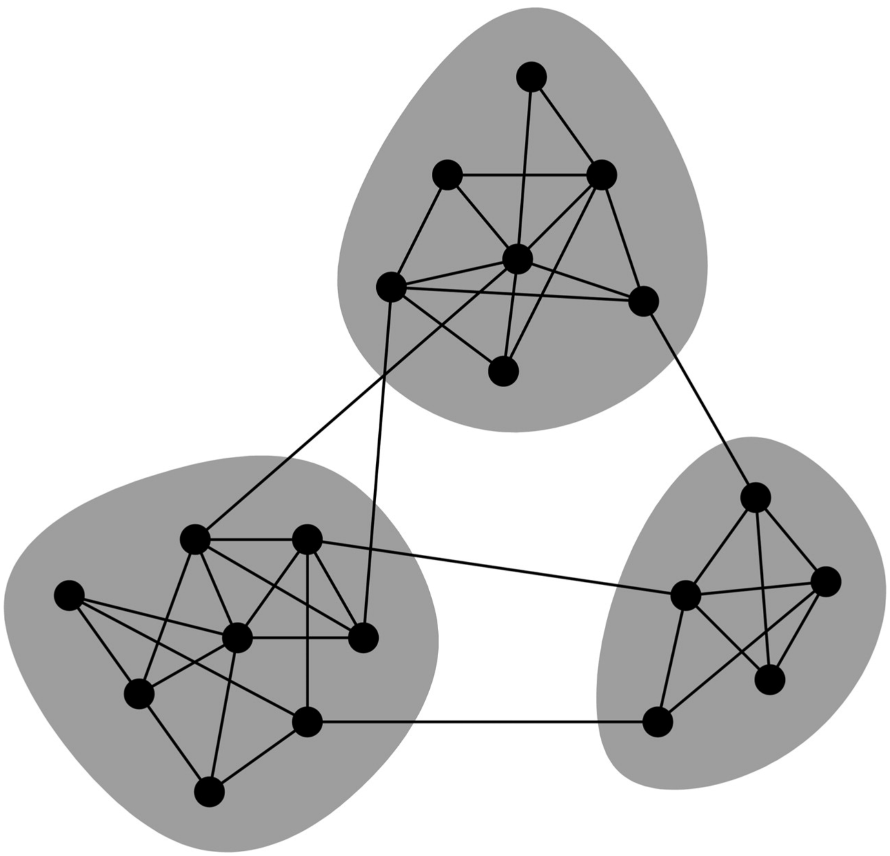
- page rank

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



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, Americal 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, Americal 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, Americal 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

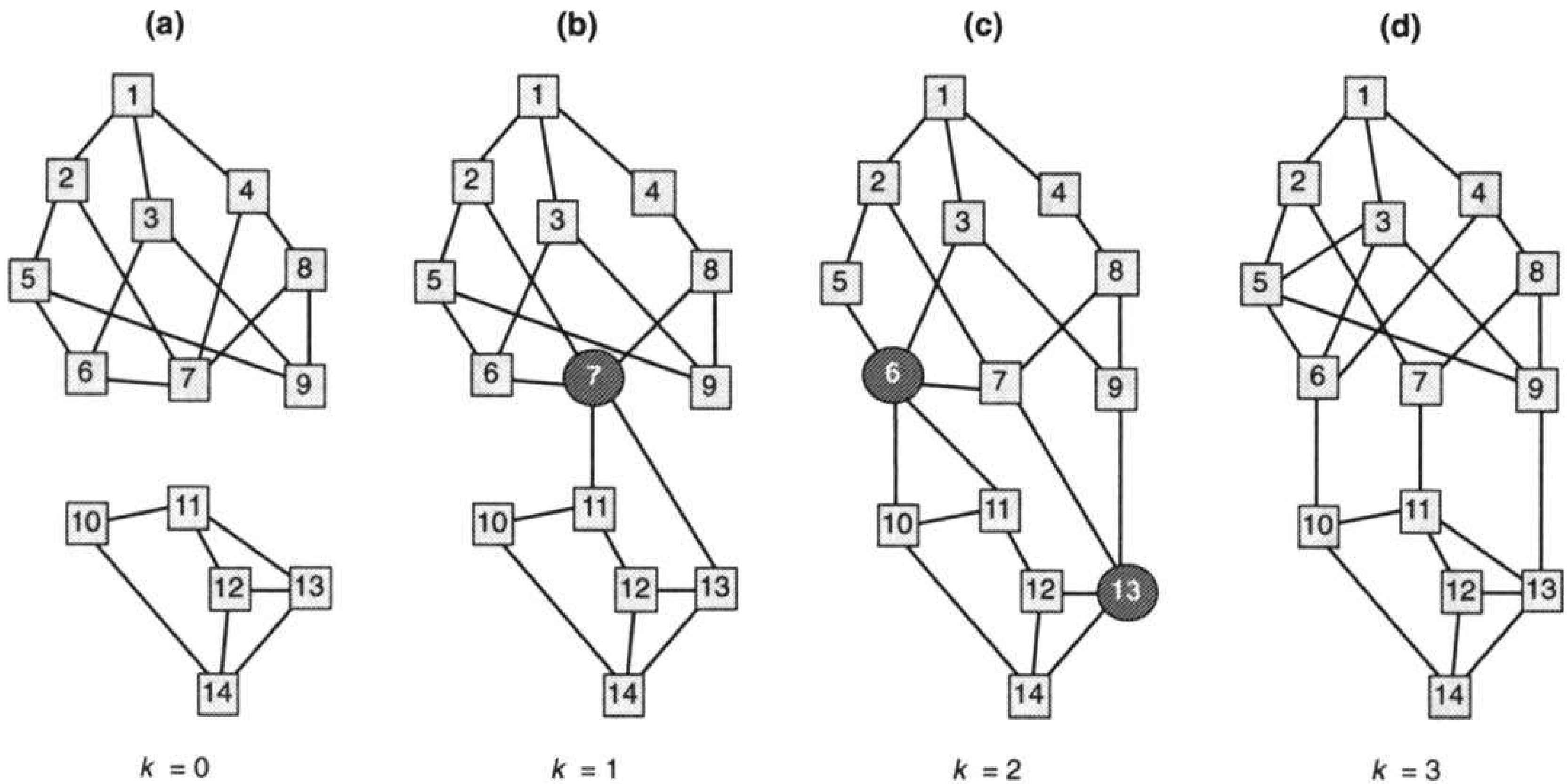
(Based on 'Structural Cohesion and Embeddedness: a Hierarchical Concept of Social Groups' by J.Moody and D.White, Americal Sociological Review, 68, 103–127, 2003)

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

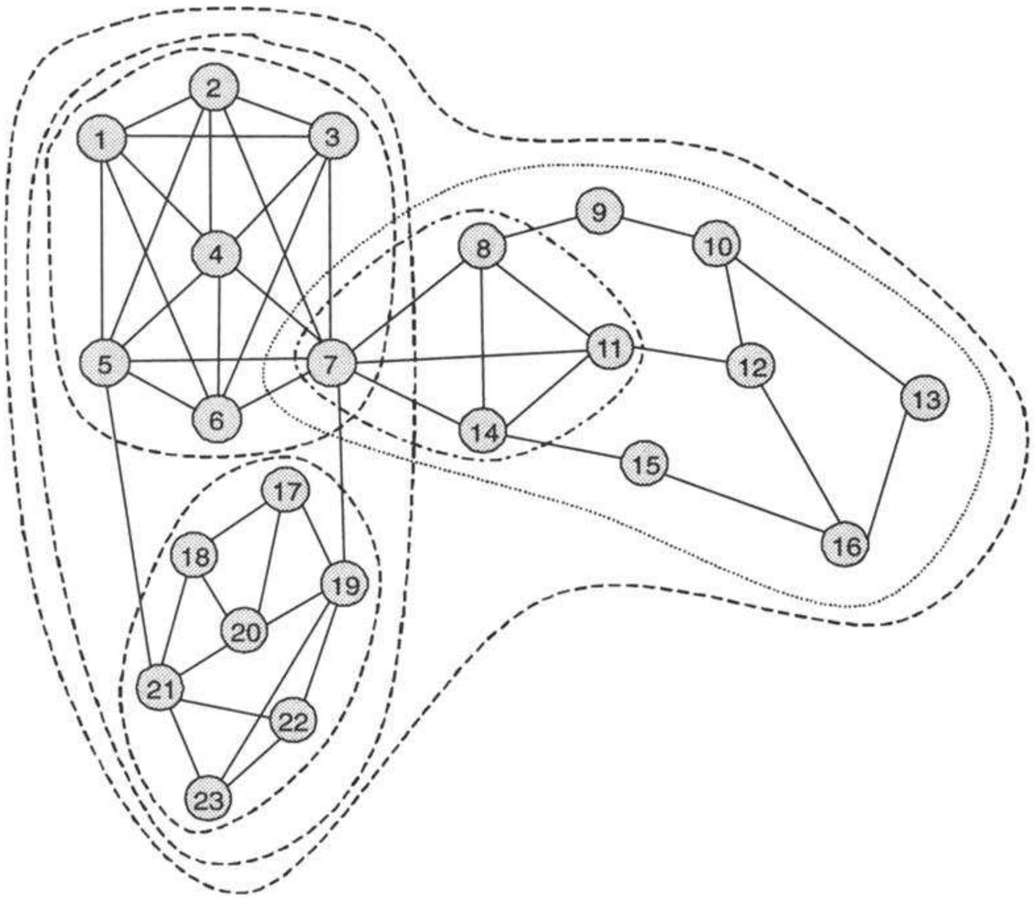
> Definition 2: A group is structurally cohesive to the extent that multiple independent relational paths among all pairs of members hold it together.

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

Cohesive blocks



Cohesive blocks



Rapid prototyping

Weighted transitivity

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

Rapid prototyping

Weighted transitivity

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

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

Rapid prototyping

Weighted transitivity

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

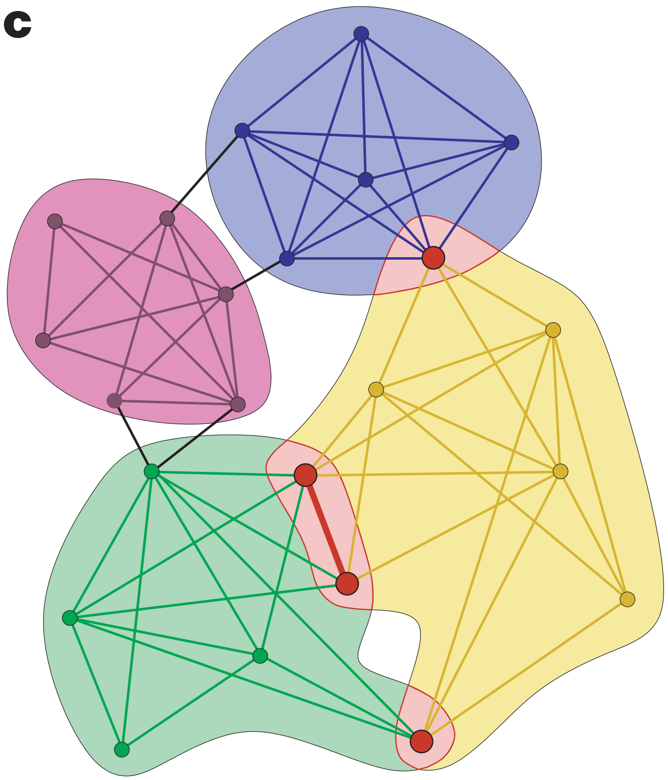
$$c_w(i) = \frac{\mathbf{W}^3_{ii}}{(\mathbf{W}\mathbf{W}_{\mathsf{max}}\mathbf{W})_{ii}}$$

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

Rapid prototyping

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

c



. . . 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