Large-scale network analysis

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

Outline

1. The igraph R package
2. What can you do with large graphs?
3. Some unique igraph features
4. Rapid prototyping

The **igraph** software package

- R package, Python extension and C library.

The **igraph** software package

- R package, Python extension and C library.
- Under active development.

The **igraph** software package

- R package, Python extension and C library.
- Under active development.
- Free for academic and commercial use (GPL). “Standing on the shoulder of giants.”

The **igraph** software package

- R package, Python extension and C library.
- Under active development.
- Free for academic and commercial use (GPL). "Standing on the shoulder of giants."
- State of the art data structures and algorithms, works well with large graphs.

How **LARGE**?

- Well, it depends what you want to calculate.

How **LARGE**?

- Well, it depends what you want to calculate.
- Just to create and manipulate it, it is enough if it fits into the memory.

How **LARGE**?

- Well, it depends what you want to calculate.
- Just to create and manipulate it, it is enough if it fits into the memory.
- How do I know that it fits into the memory?

How **LARGE**?

- Well, it depends what you want to calculate.
- Just to create and manipulate it, it is enough if it fits into the memory.
- How do I know that it fits into the memory?
- igraph (typically) needs 32 bytes per edge and 16 bytes per vertex.

How **LARGE**?

- Well, it depends what you want to calculate.
- Just to create and manipulate it, it is enough if it fits into the memory.
- How do I know that 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

```
install.packages("igraph")
```

It is really that simple. Isn't it?

Installation

```r
install.packages("igraph")
```

It is really that simple. Isn't it?

You might also need

```r
install.packages("digest")
install.packages("rgl")
```

How to follow this “lecture”?

1. Go to `http://cneurocvs.rmki.kfki.hu/igraph/NIPS2008.html` and copy & paste everything into your R session. You can skip any example if you wish to.

How to follow this “lecture”?

1. Go to `http://cneurocvs.rmki.kfki.hu/igraph/NIPS2008.html` and copy & paste everything into your R session. You can skip any example if you wish to.

2. You type in everything I type in.

How to follow this “lecture”?

1. Go to http://cneurocvs.rmki.kfki.hu/igraph/NIPS2008.html and copy & paste everything into your R session. You can skip any example if you wish to.

2. You type in everything I type in.

3. Sit back and watch. You can download the slides/code anyway.

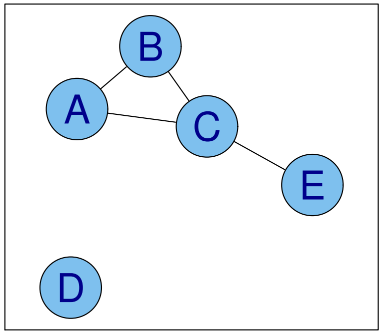
The igraph data model

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

The igraph data model

- Binary relation (=**edges**) between elements of a set (=**vertices**).
- If the pairs are unordered, then the graph is undirected:

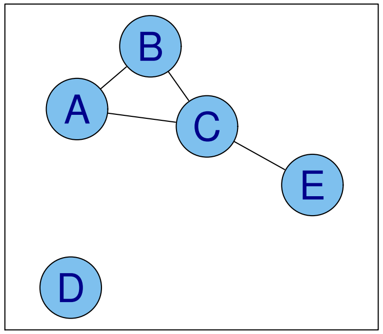
$$\text{vertices} = \{A, B, C, D, E\}$$
$$\text{edges} = (\{A, B\}, \{A, C\}, \{B, C\}, \{C, E\}).$$



The igraph data model

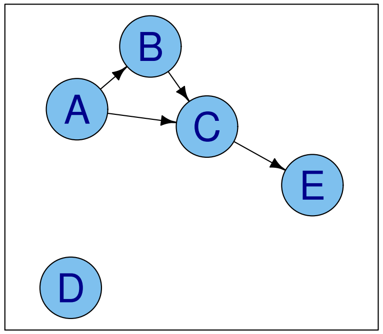
- Binary relation (=**edges**) between elements of a set (=**vertices**).
- If the pairs are unordered, then the graph is undirected:

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



- Otherwise it is directed:

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



Vertex and edge ids

- Vertices are always numbered from 0.
- Numbering is continual, form 0 to $n-1$.

Vertex and edge ids

- Vertices are always numbered from 0.
- Numbering is continual, form 0 to $n-1$.

- 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 0.
- Numbering is continual, form 0 to $n-1$.

- We have to translate vertex names to ids:

$$V = \{A, B, C, D, E\}$$
$$E = ((A, B), (A, C), (B, C), (C, E)).$$
$$A = 0, B = 1, C = 2, D = 3, E = 4.$$

```
library(igraph)
g <- graph( c(0,1, 0,2, 1,2, 2,4), n=5 )
g
g2 <- graph( c(0,1, 0,2, 1,2, 2,4),
             n=5, dir=FALSE )
g2
```

Working with igraph graphs

```
## How to decide what kind of object a variable refers to
class(g2)
class(1)
class("foobar")

## Is this object an igraph graph?
is.igraph(g)
is.igraph(1:10)

## Summary, number of vertices, edges
summary(g)
vcount(g)
ecount(g)

## Is the graph directed?
is.directed(g)
is.directed(g2)
is.directed(1:10)
```

Working with igraph graphs

```
## Convert from directed to undirected
as.undirected(g)

## And back
as.directed(as.undirected(g))
```

The igraph data model, multiple edges

- `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
```

The igraph data model, loop edges

- `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
```

The igraph data model, what cannot be represented

- "Mixed" graphs, with undirected and directed edges.
- Hypergraphs.
- No direct support for bipartite (two-mode) graphs.

Naming vertices

```
g <- graph.ring(10)
V(g)$name <- letters[1:10]
V(g)$name
g
print(g, v=T)
```

Creating graphs, the formula interface

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

Creating graphs, the formula interface

```
# 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 graphs, the formula interface

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

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

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

Creating graphs, the formula interface

```
# 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 graphs, the formula interface

```
# 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 )
```

Creating graphs, from edge lists and adjacency matrices

```
## From edge lists
el <- cbind( c(0, 0, 1, 2),
             c(1, 2, 2, 4) )
g <- graph.edgelist(el)
g

## Symbolic edge lists
el <- cbind( c("Alice", "Alice", "Bob",   "Cecil"),
             c("Bob",   "Cecil", "Cecil", "Ed") )
g <- graph.edgelist(el)
g
summary(g)

## Adjacency matrices
A <- matrix(sample(0:1, 100, rep=TRUE), 10, 10)
g <- graph.adjacency(A)
```

Creating graphs, from data frames

```
source("http://cneurocvs.rmki.kfki.hu/igraph/plus.R")
vertices <- read.csv("judicial.csv")
edges <- read.table("allcites.txt")
jg <- graph.data.frame(edges, vertices=vertices, dir=TRUE)
```

Visualizing graphs

- `plot` Uses traditional R graphics, non-interactive, 2d. Publication quality plots in all formats R supports.

Visualizing graphs

- `plot` Uses traditional R graphics, non-interactive, 2d. Publication quality plots in all formats R supports.

```
g <- barabasi.game(100, m=1)
g <- simplify(g)
igraph.par("plot.layout",
           layout.fruchterman.reingold)
plot(g, vertex.size=3, vertex.label=NA,
     edge.arrow.size=0.6)
```

Visualizing graphs

- `tkplot` Uses Tcl/Tk via the `tcltk` package, interactive, 2d.

Visualizing graphs

- `tkplot` Uses Tcl/Tk via the `tcltk` package, interactive, 2d.

```
id <- tkplot(g, vertex.size=3, vertex.label=NA,
      edge.arrow.size=0.6)
coords <- tkplot.getcoords(id)
```

Visualizing graphs

- `tkplot` Uses Tcl/Tk via the `tcltk` package, interactive, 2d.

```
id <- tkplot(g, vertex.size=3, vertex.label=NA,
      edge.arrow.size=0.6)
coords <- tkplot.getcoords(id)
```

- `rglplot` Needs the `rgl` package.

Visualizing graphs

- `tkplot` Uses Tcl/Tk via the `tcltk` package, interactive, 2d.

```
id <- tkplot(g, vertex.size=3, vertex.label=NA,
      edge.arrow.size=0.6)
coords <- tkplot.getcoords(id)
```

- `rglplot` Needs the `rgl` package.

```
rglplot(g, vertex.size=3, vertex.label=NA)

coords <- layout.kamada.kawai(g, dim=3)
rglplot(g, vertex.size=3, vertex.label=NA,
        layout=coords)
```

Visualizing graphs

- `tkplot` Uses Tcl/Tk via the `tcltk` package, interactive, 2d.

```
id <- tkplot(g, vertex.size=3, vertex.label=NA,
      edge.arrow.size=0.6)
coords <- tkplot.getcoords(id)
```

- `rglplot` Needs the `rgl` package.

```
rglplot(g, vertex.size=3, vertex.label=NA)

coords <- layout.kamada.kawai(g, dim=3)
rglplot(g, vertex.size=3, vertex.label=NA,
        layout=coords)
```

- (Almost) identical interfaces.

Graph/vertex/edge attributes

- Assigning/querying attributes:

  `set.graph.attribute`, `get.graph.attribute`,

  `set.vertex.attribute`, `get.vertex.attribute`,

  `set.edge.attribute`, `get.edge.attribute`,

  `list.graph.attributes`,

  `list.vertex.attributes`,

  `list.edge.attributes`.

Graph/vertex/edge attributes

- Assigning/querying attributes:
  `set.graph.attribute`, `get.graph.attribute`,
  `set.vertex.attribute`, `get.vertex.attribute`,
  `set.edge.attribute`, `get.edge.attribute`,
  `list.graph.attributes`,
  `list.vertex.attributes`,
  `list.edge.attributes`.
- `V(g)` and `E(g)`.

Graph/vertex/edge attributes

- Assigning/querying attributes:
  `set.graph.attribute`, `get.graph.attribute`,
  `set.vertex.attribute`, `get.vertex.attribute`,
  `set.edge.attribute`, `get.edge.attribute`,
  `list.graph.attributes`,
  `list.vertex.attributes`,
  `list.edge.attributes`.
- `V(g)` and `E(g)`.

Graph/vertex/edge attributes

```
## Load the jurisdiction network
load("judicial.Rdata.gz")

## If we don't have it then create it again
if (!exists("jg")) {
  source("http://cneurocvs.rmki.kfki.hu/igraph/plus.R")
  vertices <- read.csv("http://cneurocvs.rmki.kfki.hu/igraph/judicial.csv")
  edges <- read.table("http://cneurocvs.rmki.kfki.hu/igraph/allcites.txt")
  jg <- graph.data.frame(edges, vertices=vertices, dir=TRUE)
}
```

Graph/vertex/edge attributes

```
## What do we have?
summary(jg)
V(jg)$year
V(jg)$parties

## Select vertices based on attributes
V(jg) [ year >= 1990 ]
V(jg) [ overruled!=0 ]

## Group network measures based on attributes
deg.per.year <- tapply(degree(jg, mode="out"),
                 V(jg)$year, mean)

## Plot it
plot( names(deg.per.year), deg.per.year )
```

Smart indexing

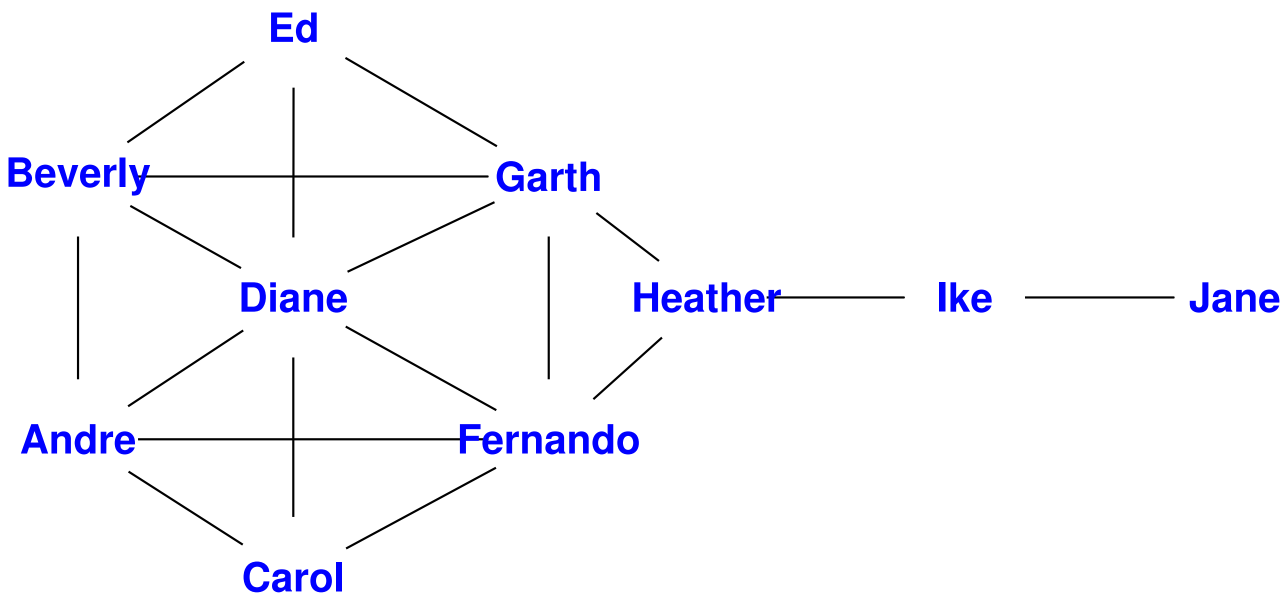
- Easy access of attributes:

```
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)
```

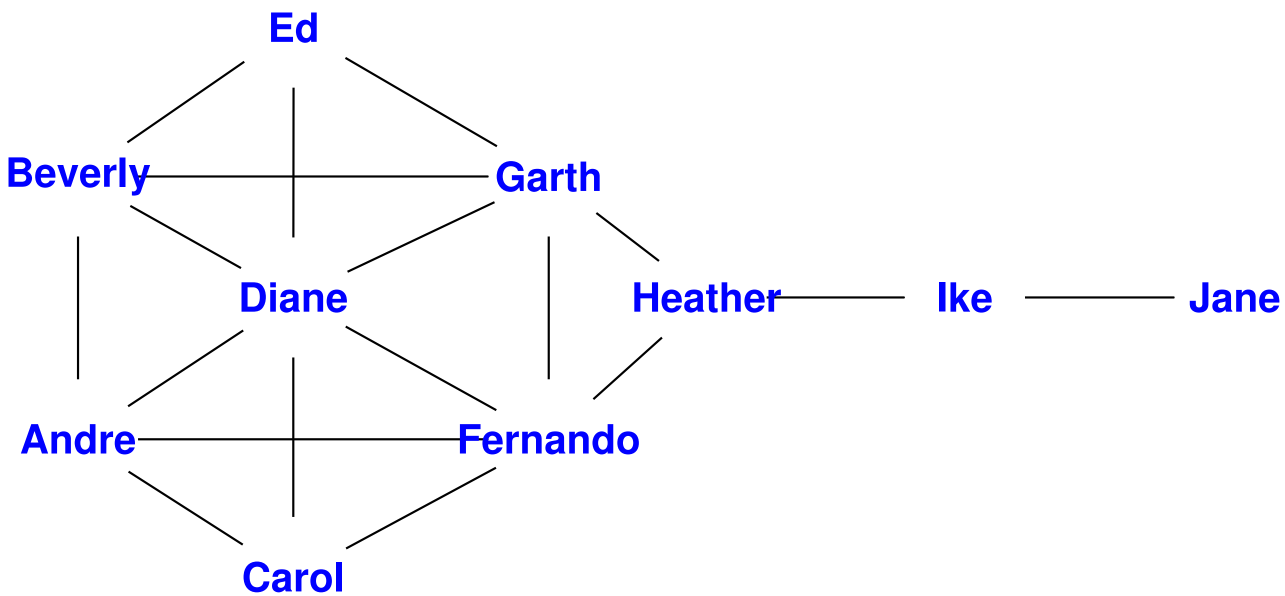
Centrality, the network

```
g <- graph.formula( Andre----Beverly:Diane:Fernando:Carol,
                    Beverly--Andre:Diane:Garth:Ed,
                    Carol----Andre:Diane:Fernando,
                    Diane----Andre:Carol:Fernando:Garth:Ed:Beverly,
                    Ed-------Beverly:Diane:Garth,
                    Fernando-Carol:Andre:Diane:Garth:Heather,
                    Garth----Ed:Beverly:Diane:Fernando:Heather,
                    Heather--Fernando:Garth:Ike,
                    Ike------Heather:Jane,
                    Jane-----Ike )
g <- simplify(g)
coords <- c(5,5,119,256,119,256,120,340,478,
    622,116,330,231,116,5,330,451,231,231,231)
coords <- matrix(coords, nc=2)
V(g)$label <- V(g)$name
g$layout <- coords # $
plot(g, asp=FALSE, vertex.label.color="blue", vertex.label.cex=1.5,
     vertex.label.font=2, vertex.size=20, vertex.color="white",
     vertex.frame.color="white", edge.color="black")
```

Centrality, the network

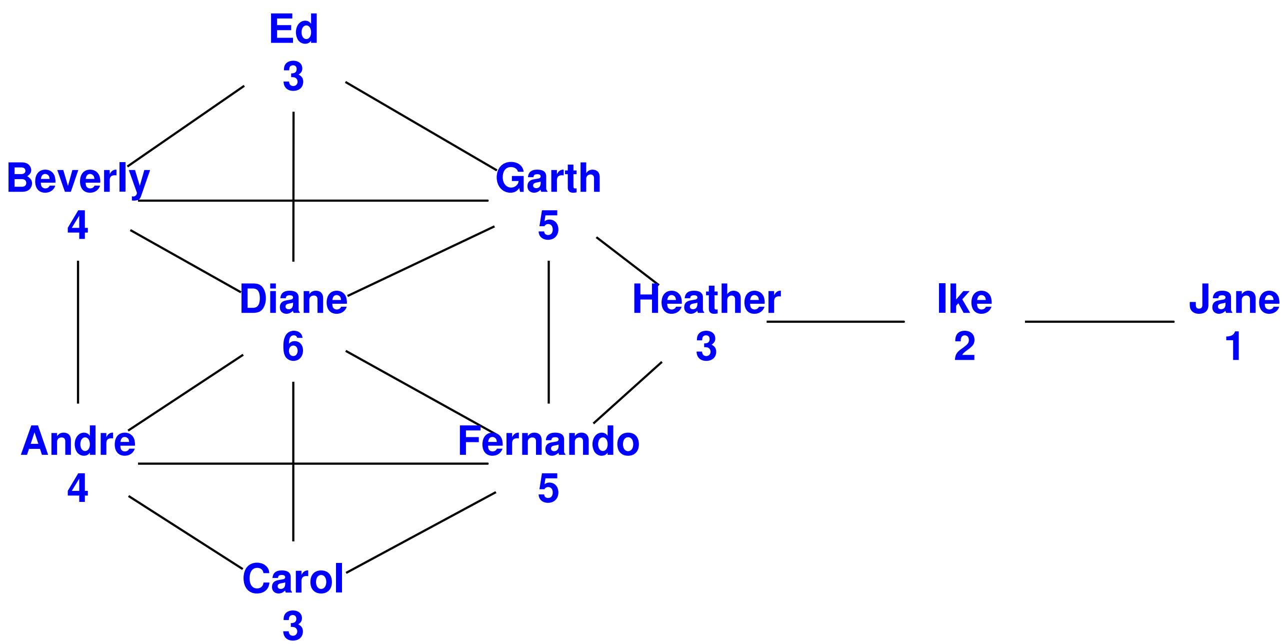


Centrality, degree



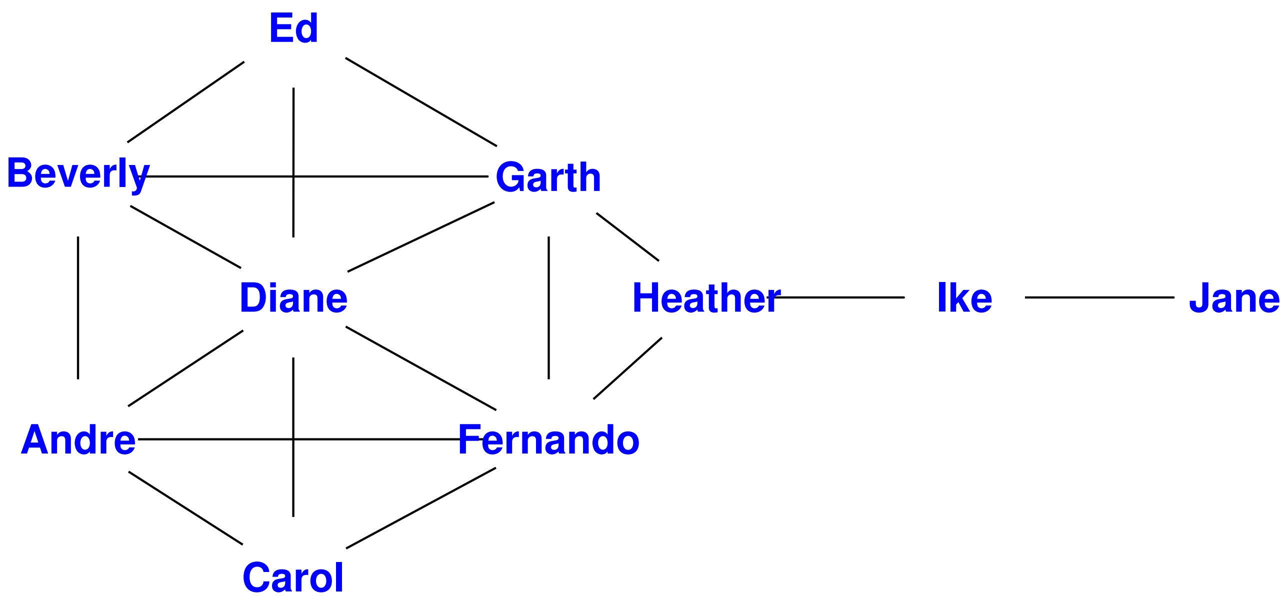
Number of adjacent edges.

Centrality, degree



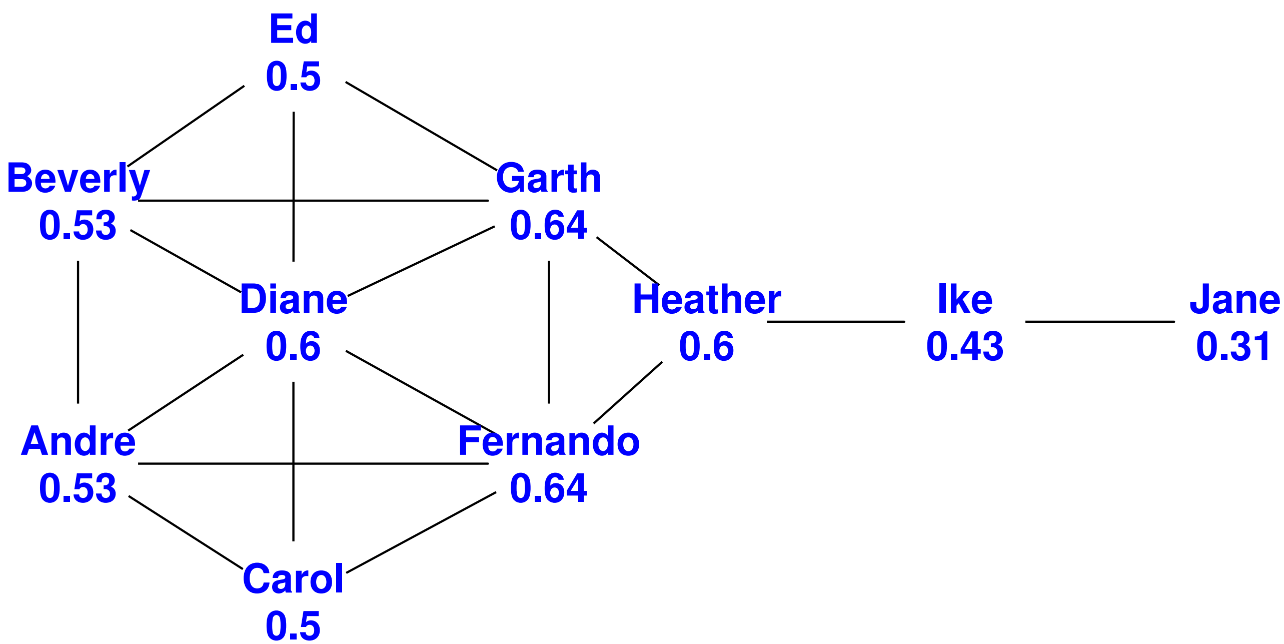
Number of adjacent edges.

Centrality, closeness



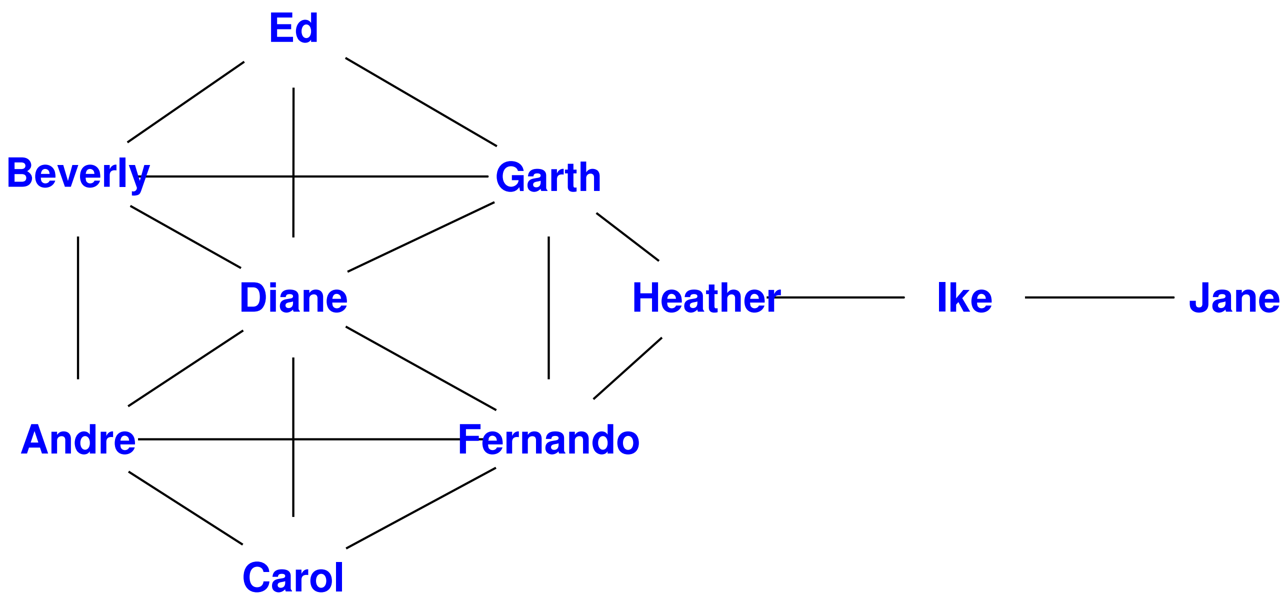
Reciproc of the average distance to other vertices.

Centrality, closeness



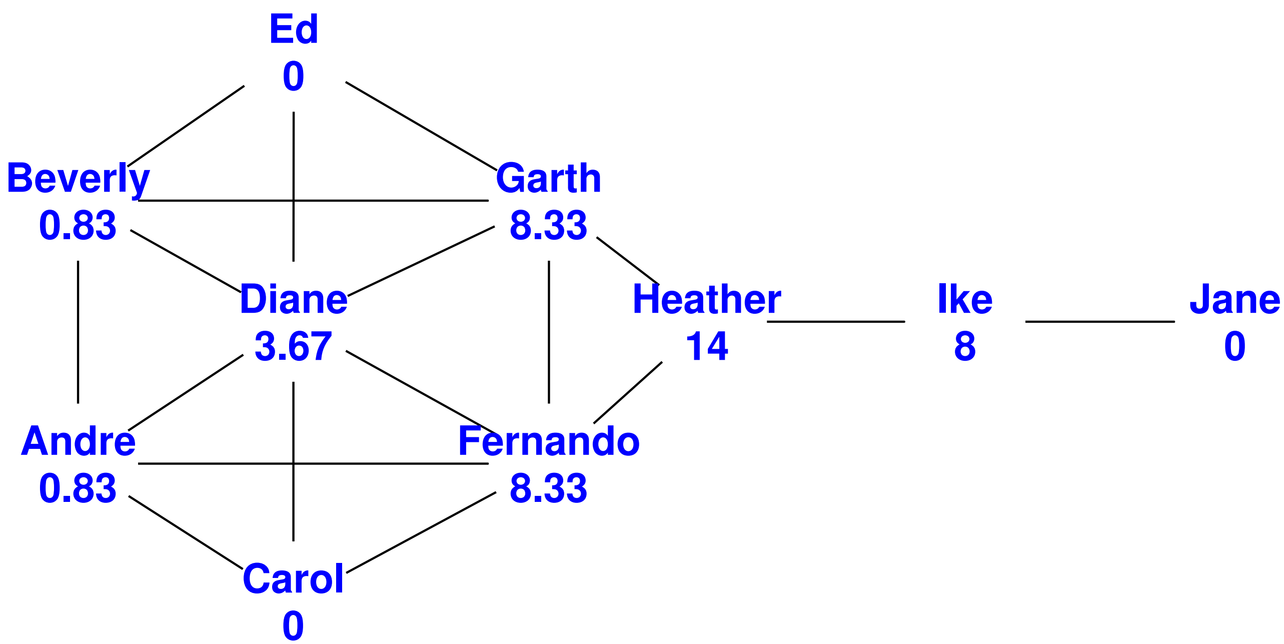
Reciproc of the average distance to other vertices.

Centrality, betweenness



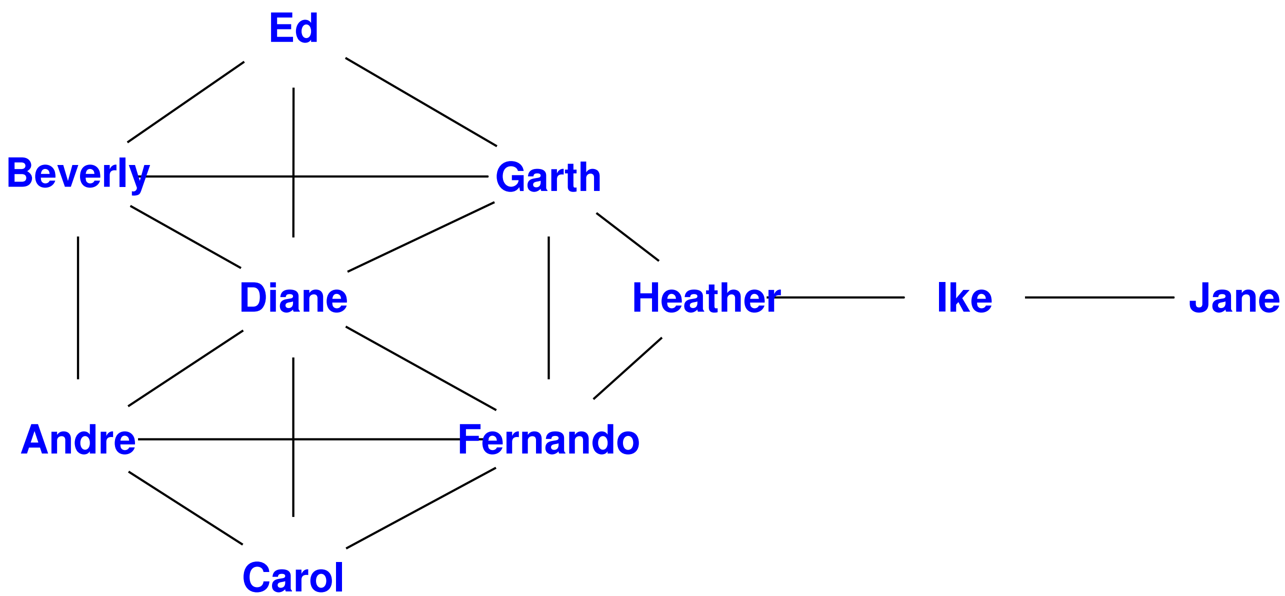
Number of shortest paths going through a vertex.

Centrality, betweenness



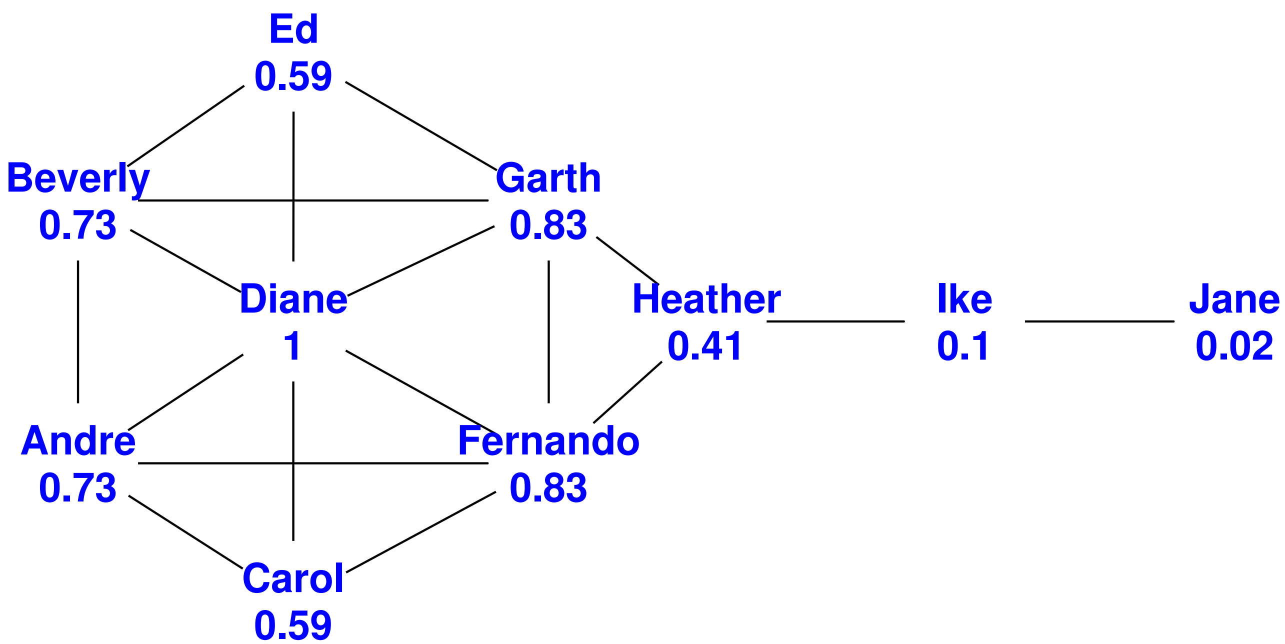
Number of shortest paths going through a vertex.

Centrality, eigenvector centrality



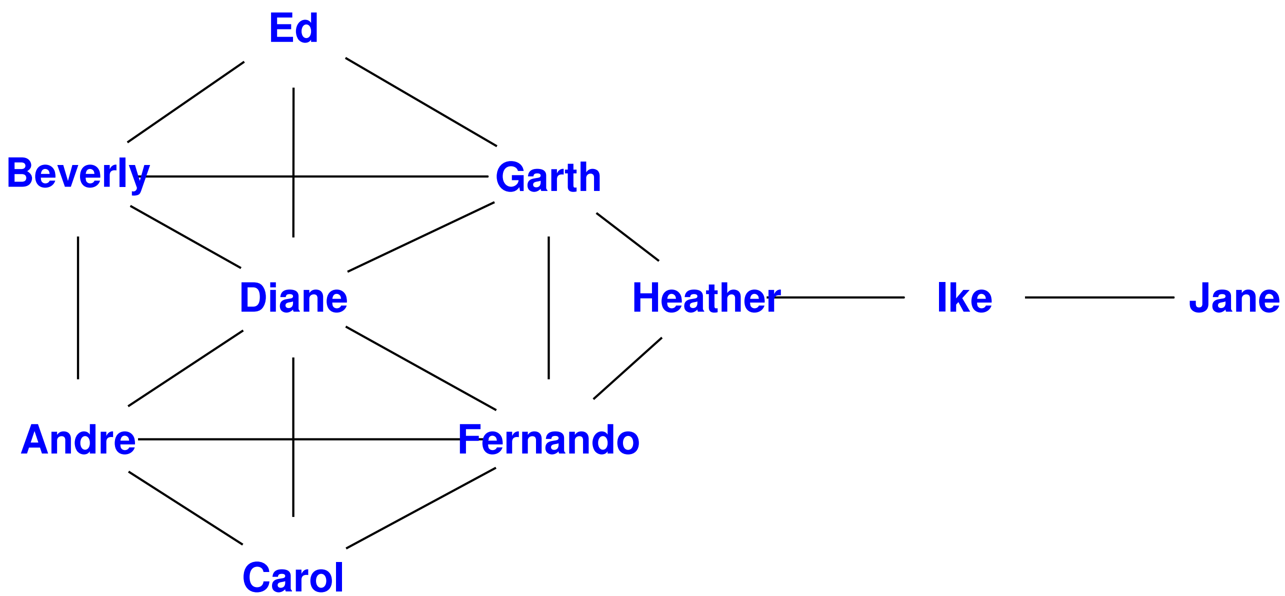
Number of adjacent edges, weighted by their “goodness”.

Centrality, eigenvector centrality



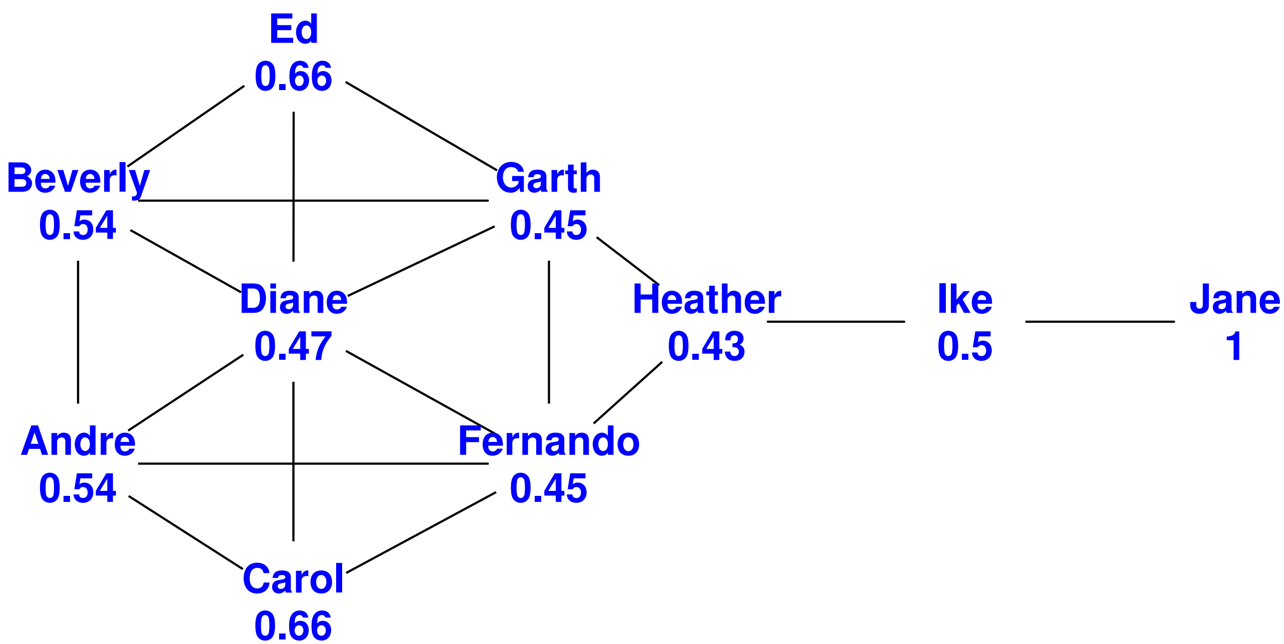
Number of adjacent edges, weighted by their “goodness”.

Centrality, Burt's constraint



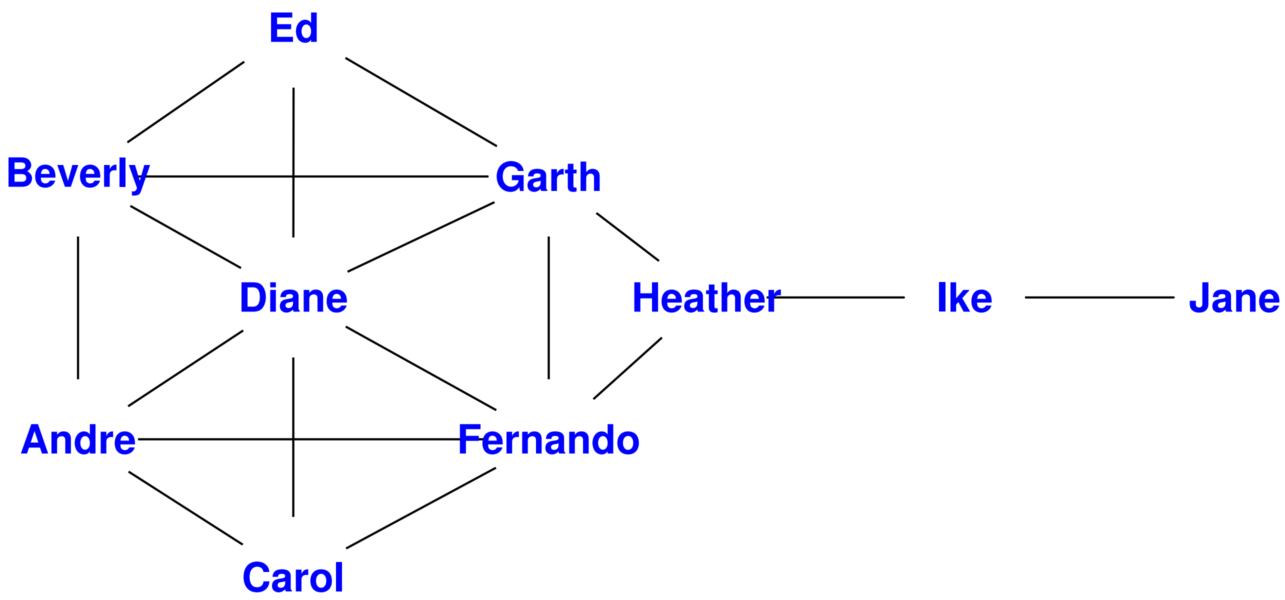
Benefit of brokering between other actors.

Centrality, Burt's constraint



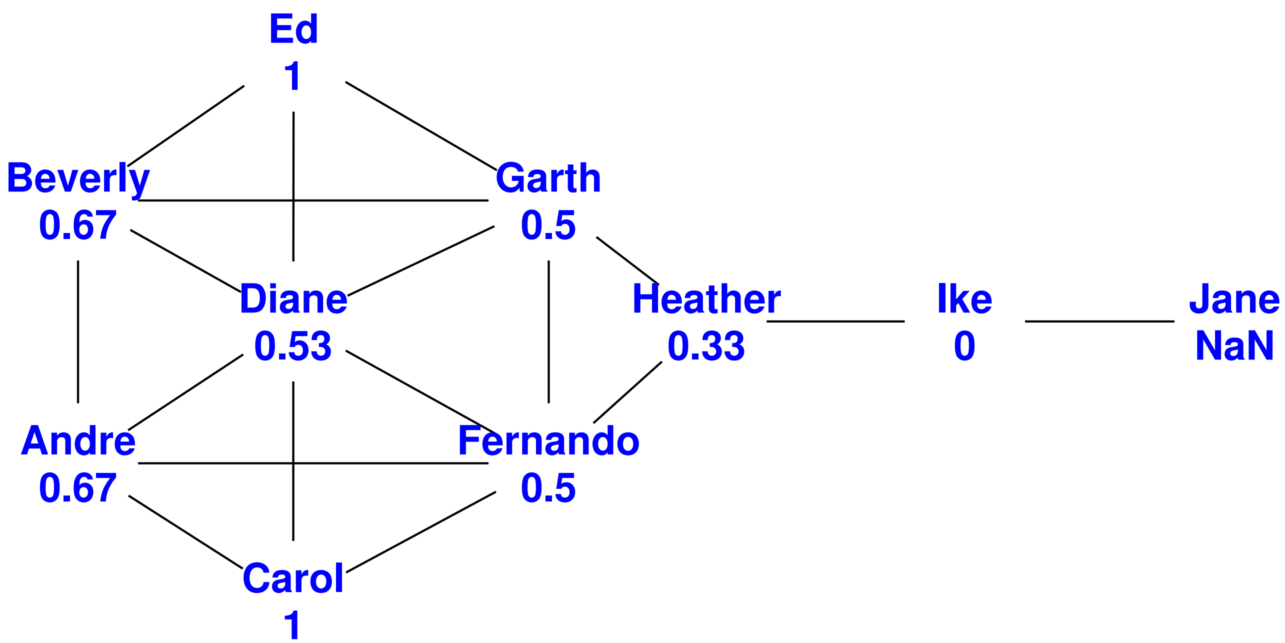
Benefit of brokering between other actors.

Centrality, transitivity



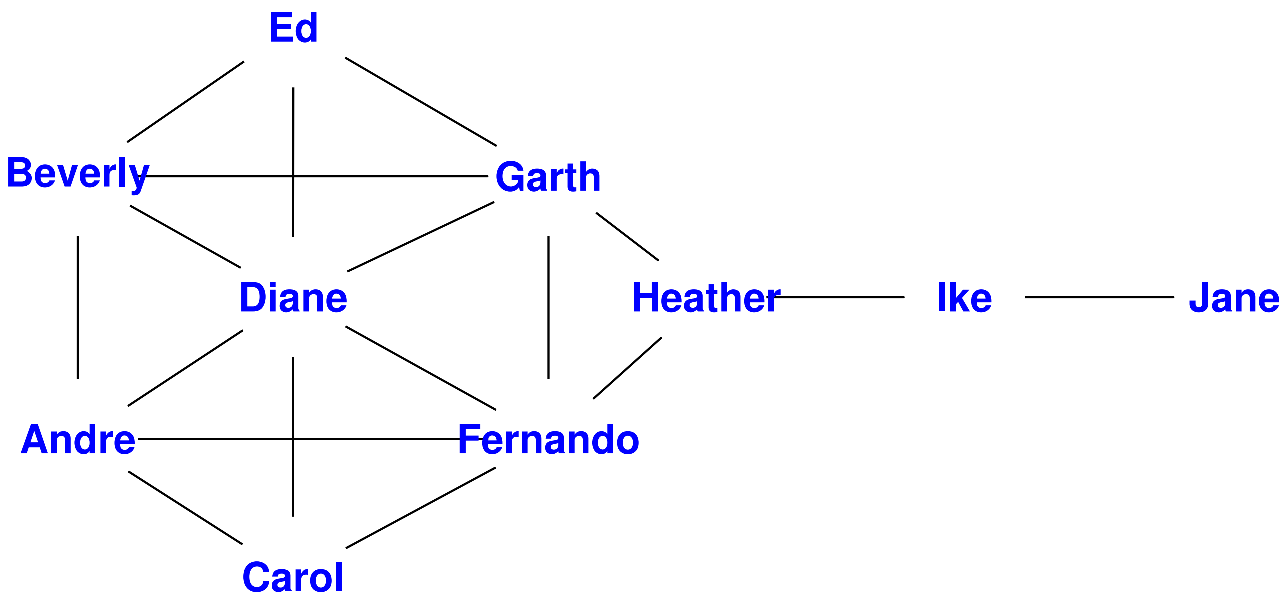
Are my friends also friends of each other?

Centrality, transitivity



Are my friends also friends of each other?

Sensitivity of centrality measures

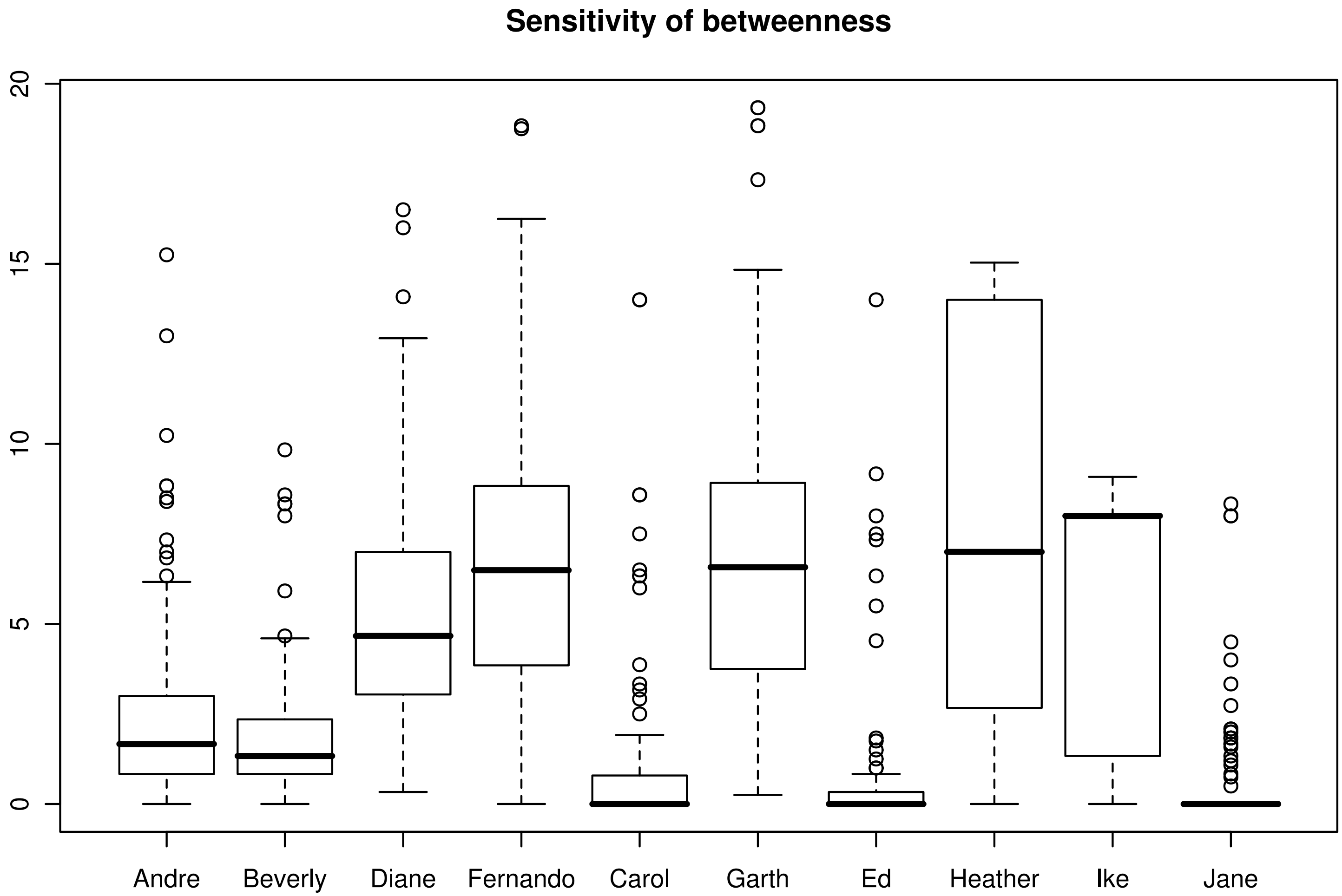


Remove two random edges, add two random edges.

Sensitivity of centrality measures

```
cl <- numeric()
for (i in 1:100) {
  g2 <- delete.edges(g, sample(ecount(g), 2)-1)
  g2 <- g2 %u%
     random.graph.game(vcount(g), 2, type="gnm")
  cl <- cbind(cl, betweenness(g2))
}
cl <- as.data.frame(t(cl))
colnames(cl) <- V(g)$name  # $
boxplot(cl, main="Sensitivity of betweenness")
```

Sensitivity of centrality measures



Working with a (moderately) large graph

```
library(igraph)

## Load the jurisdiction network
load("judicial.Rdata.gz")

## If we don't have it then create it again
if (!exists("jg")) {
  source("http://cneurocvs.rmki.kfki.hu/igraph/plus.R")
  vertices <- read.csv("http://cneurocvs.rmki.kfki.hu/igraph/judicial.csv")
  edges <- read.table("http://cneurocvs.rmki.kfki.hu/igraph/allcites.txt")
  jg <- graph.data.frame(edges, vertices=vertices, dir=TRUE)
}

## Basic data
summary(jg)

## Is it a simple graph?
is.simple(jg)
```

Working with a (moderately) large graph

```
## Is it connected?
is.connected(jg)

## How many components?
no.clusters(jg)

## How big are these?
table(clusters(jg)$csize)   #$

## In-degree distribution
plot(degree.distribution(jg, mode="in"), log="xy")

## Out-degree distribution
plot(degree.distribution(jg, mode="out"), log="xy")

## Largest in- and out-degree, total degree
max(degree(jg, mode="in"))
max(degree(jg, mode="out"))
max(degree(jg, mode="all"))
```

Working with a (moderately) large graph

```
## Density
graph.density(jg)

## Transitivity
transitivity(jg)

## Transitivity of a random graph of the same size
g <- erdos.renyi.game(vcount(jg), ecount(jg), type="gnm")
transitivity(g)

## Dyad census
dyad.census(jg)

## Triad census
triad.census(jg)
```

Working with a (moderately) large graph

```
## Authority and Hub scores
authority.score(jg)$vector
cor(authority.score(jg)$vector, V(jg)$auth)

hub.score(jg)$vector
cor(hub.score(jg)$vector, V(jg)$hub)
```

Big table of "what can be run"

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, evcent • 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 • bonpow • alpha centrality • (sub)graph isomorphism

Why is igraph sooooo slow?

cliques and independent vertex sets	Hard problem
cohesive blocks	Semi-hard problem
edge/vertex connectivity, maximum flows and minimum cuts	Semi-hard problem
Bonacich's power centrality, alpha centrality	Poor implementation
(sub)graph isomorphism	Hard problem
anything else	**contact us if you want to speed it up**

Connection to other network/graph software

- `sna` and `network` R packages. Currently throught adjacency matrices. Use namespaces!

Connection to other network/graph software

- `sna` and `network` R packages. Currently throught adjacency matrices. Use namespaces!
- Pajek. `.net` file format is supported.

Connection to other network/graph software

- `sna` and `network` R packages. Currently throught adjacency matrices. Use namespaces!
- Pajek. `.net` file format is supported.

```
g <- read.graph("http://vlado.fmf.uni-lj.si/pub/networks/data/",
     format="pajek")
```

Connection to other network/graph software

- `sna` and `network` R packages. Currently throught adjacency matrices. Use namespaces!

- Pajek. `.net` file format is supported.

```
g <- read.graph("http://vlado.fmf.uni-lj.si/pub/networks/data/",
     format="pajek")
```

- Visone. Use GraphML format.

Connection to other network/graph software

- `sna` and `network` R packages. Currently throught adjacency matrices. Use namespaces!
- Pajek. `.net` file format is supported.

```
g <- read.graph("http://vlado.fmf.uni-lj.si/pub/networks/data/",
     format="pajek")
```

- Visone. Use GraphML format.
- Cytoscape. Use GML format.

Connection to other network/graph software

- `sna` and `network` R packages. Currently throught adjacency matrices. Use namespaces!
- Pajek. `.net` file format is supported.

```
g <- read.graph("http://vlado.fmf.uni-lj.si/pub/networks/data/",
      format="pajek")
```

- Visone. Use GraphML format.
- Cytoscape. Use GML format.
- GraphViz. igraph can write `.dot` files.

Connection to other network/graph software

- `sna` and `network` R packages. Currently throught adjacency matrices. Use namespaces!

- Pajek. `.net` file format is supported.

```
g <- read.graph("http://vlado.fmf.uni-lj.si/pub/networks/data/",
     format="pajek")
```

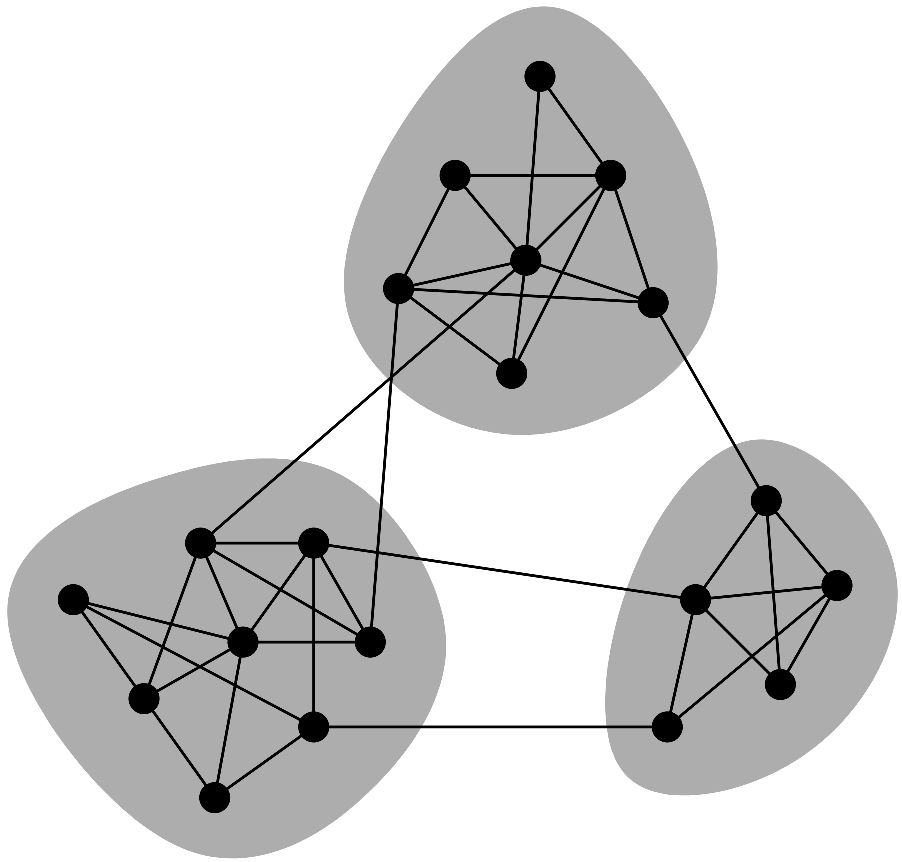
- Visone. Use GraphML format.

- Cytoscape. Use GML format.

- GraphViz. igraph can write `.dot` files.

- In general. The **GraphML** and **GML** file formats are fully supported, many programs can read/write these.

Community structure detection



(Modularity and community structure in networks, by Mark Newman, 2006)

Modularity score

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

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

Modularity score

- 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:

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

Modularity score

- 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:

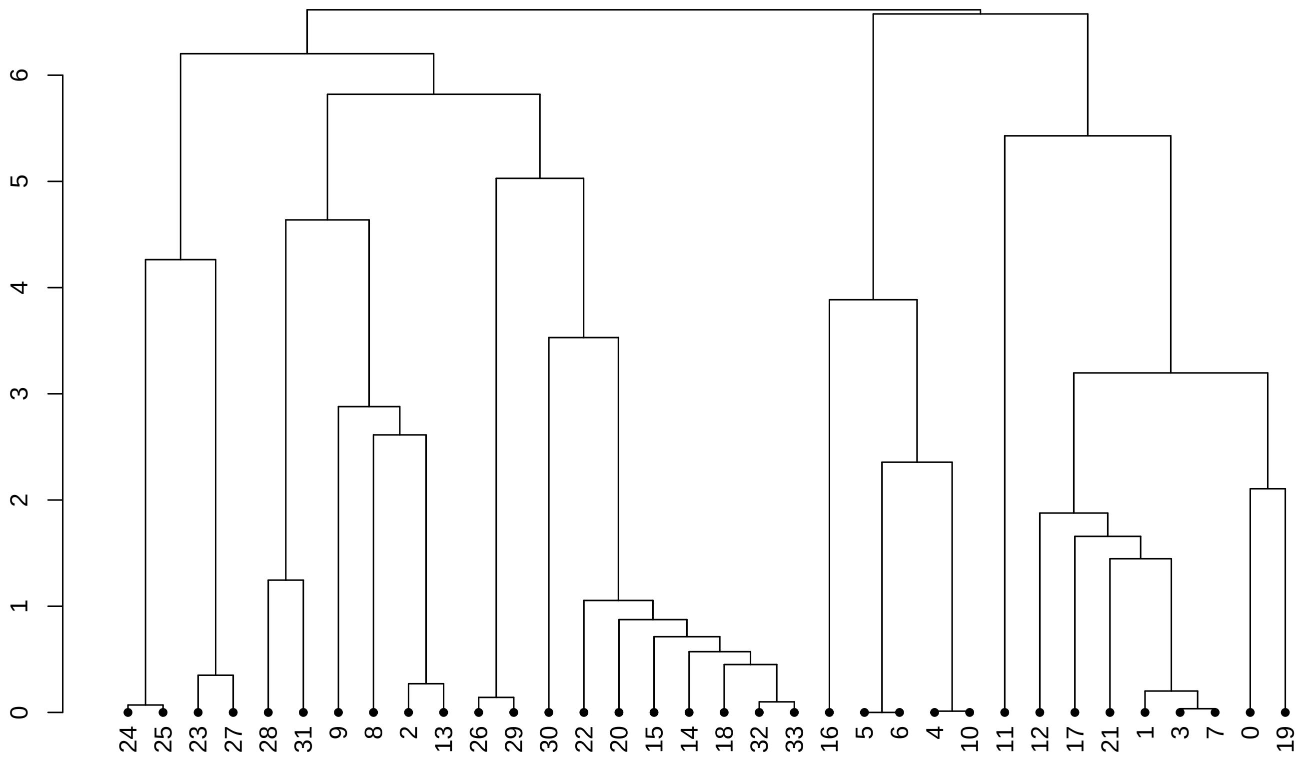
$$p_{vw} = p = \frac{1}{|V|(|V| - 1)}$$

- Degree sequence based null model:

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

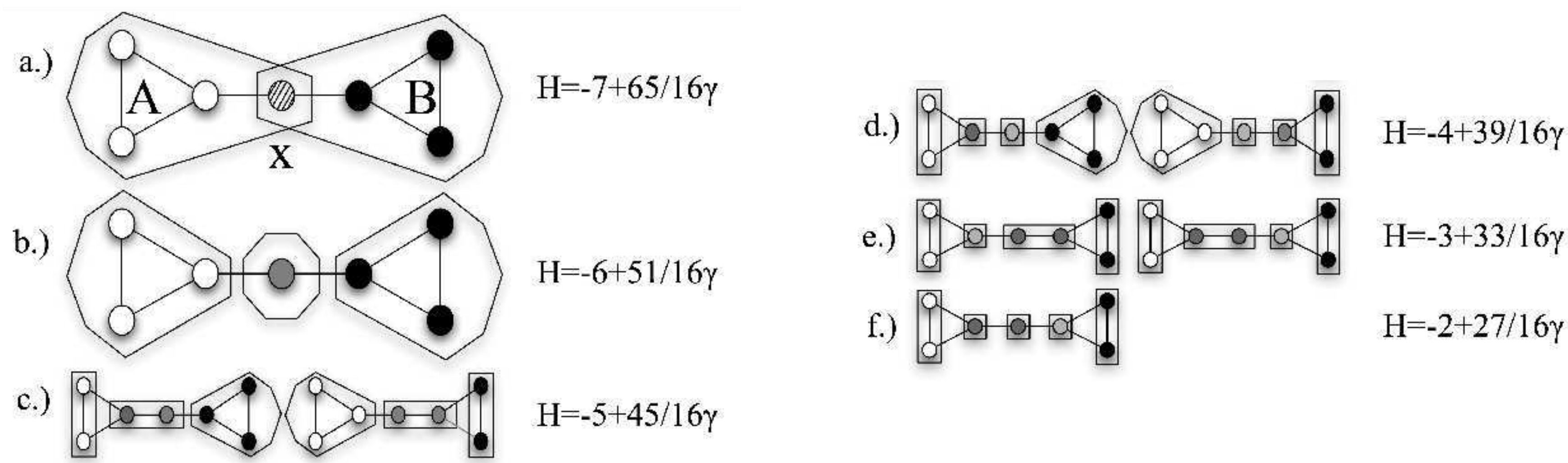
"Fast-greedy" algorithm

A Clauset, MEJ Newman, C Moore: Finding community structure in very large networks,
`http://www.arxiv.org/abs/cond-mat/0408187`



“Spinglass” algorithm

J. Reichardt and S. Bornholdt: Statistical Mechanics of Community Detection, Phys. Rev. E, 74, 016110 (2006), `http://arxiv.org/abs/cond-mat/0603718`



Cohesive blocks

- ‘Structural Cohesion and Embeddedness: a Hierarchical Concept of Social Groups’ by J.Moody and D.White, Americal Sociological Review, 68, 103–127, 2003

Cohesive blocks

- ‘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

- ‘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.

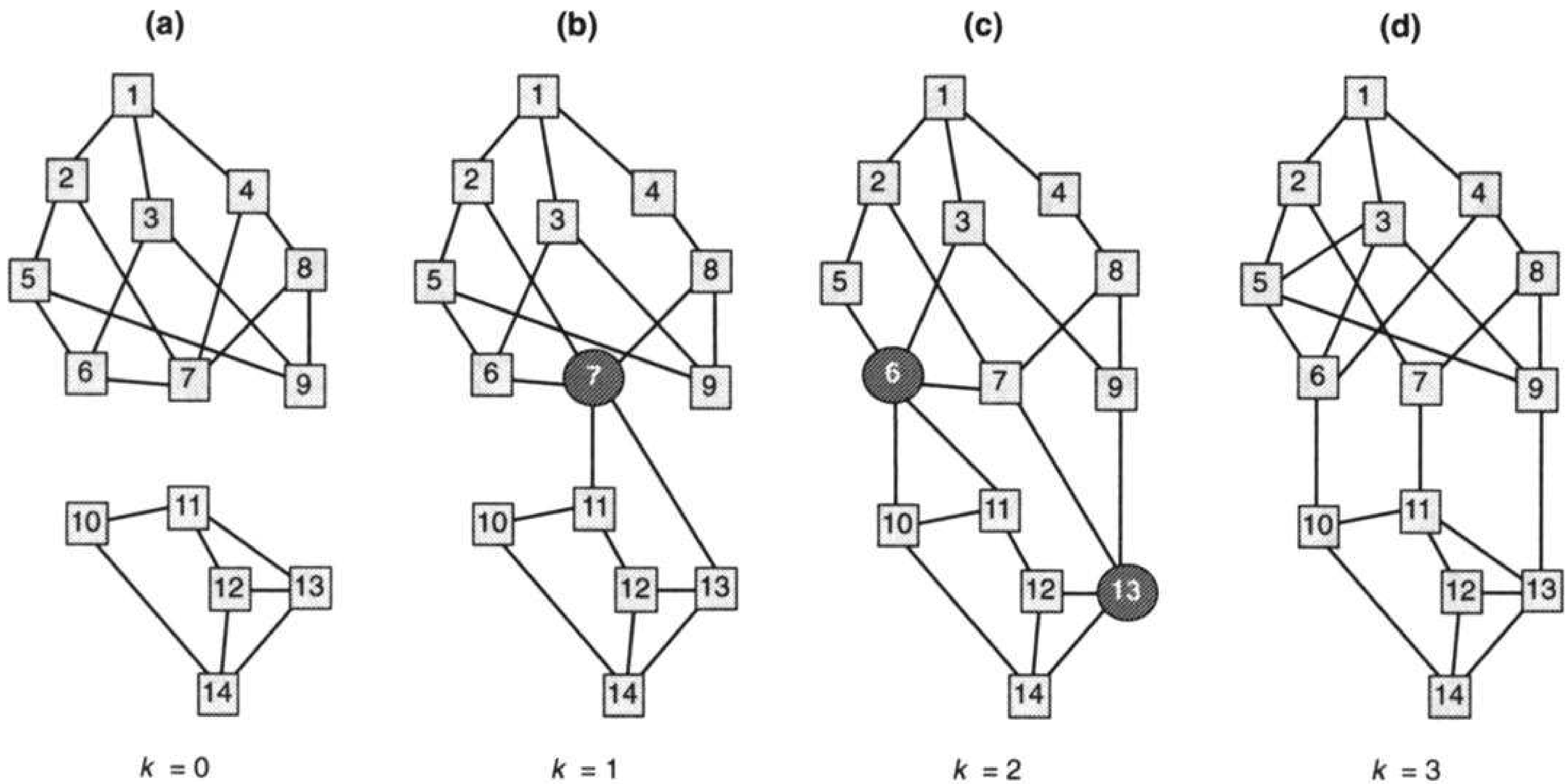
Cohesive blocks

- '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.

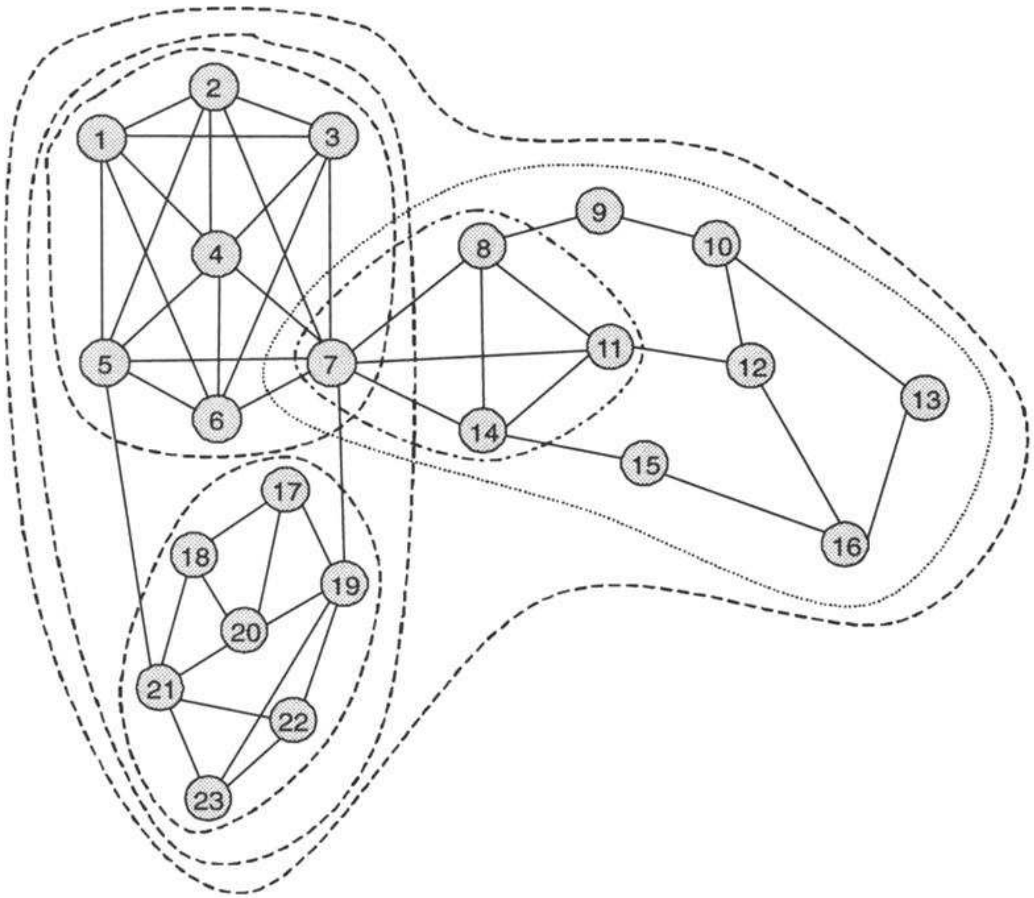
Cohesive blocks

- '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

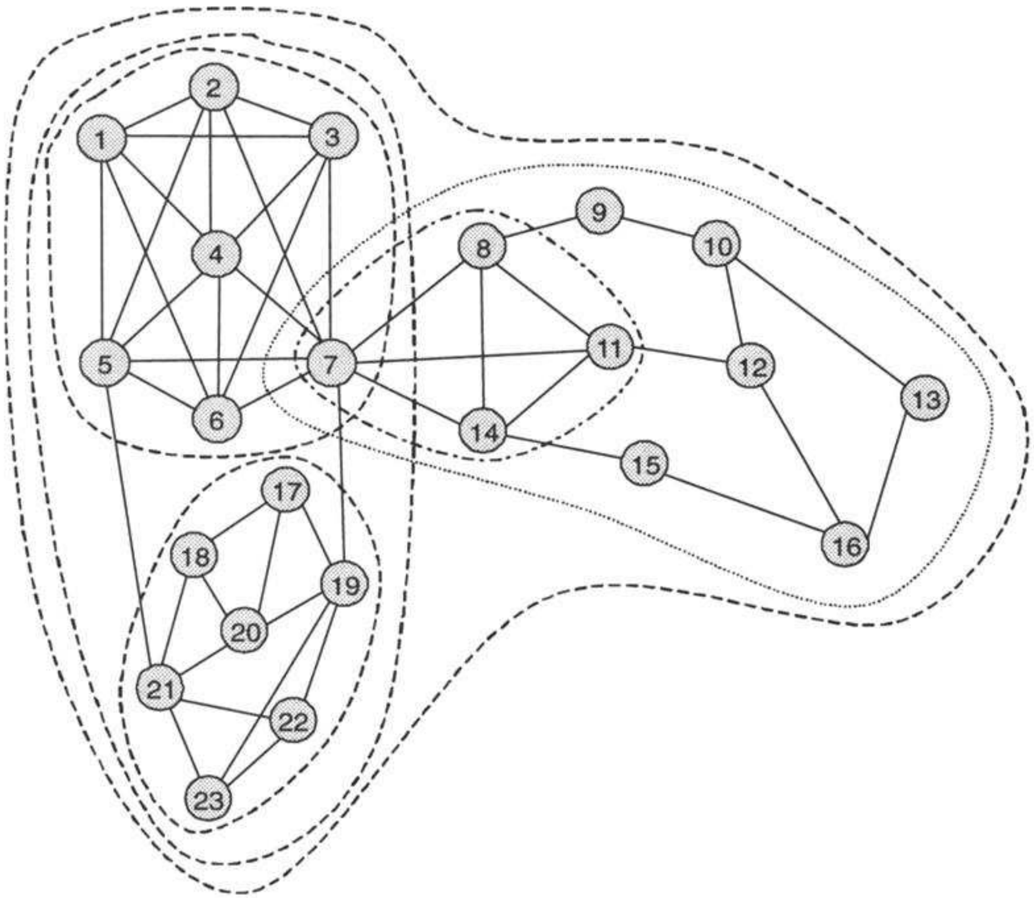


Cohesive blocks

```
cb <- graph( c(1,2,1,3,1,4,1,5,1,6,
               2,3,2,4,2,5,2,7,
               3,4,3,6,3,7,
               4,5,4,6,4,7,
               5,6,5,7,5,21,
               6,7,
               7,8,7,11,7,14,7,19,
               8,9,8,11,8,14,
               9,10,
               10,12,10,13,
               11,12,11,14,
               12,16, 13,16, 14,15, 15,16,
               17,18,17,19,17,20,
               18,20,18,21,
               19,20,19,22,19,23,
               20,21, 21,22,21,23,
               22,23)-1, dir=FALSE)

V(cb)$label <- seq(vcount(cb))  # $
```

Cohesive blocks



Cohesive blocks

```
blocks <- cohesive.blocks(cb)
blocks

summary(blocks)
blocks$blocks
lapply(blocks$blocks, "+", 1)
blocks$block.cohesion #$
plot(blocks, layout=layout.kamada.kawai,
     vertex.label.cex=2, vertex.size=15,
     vertex.label.color="black")
```

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

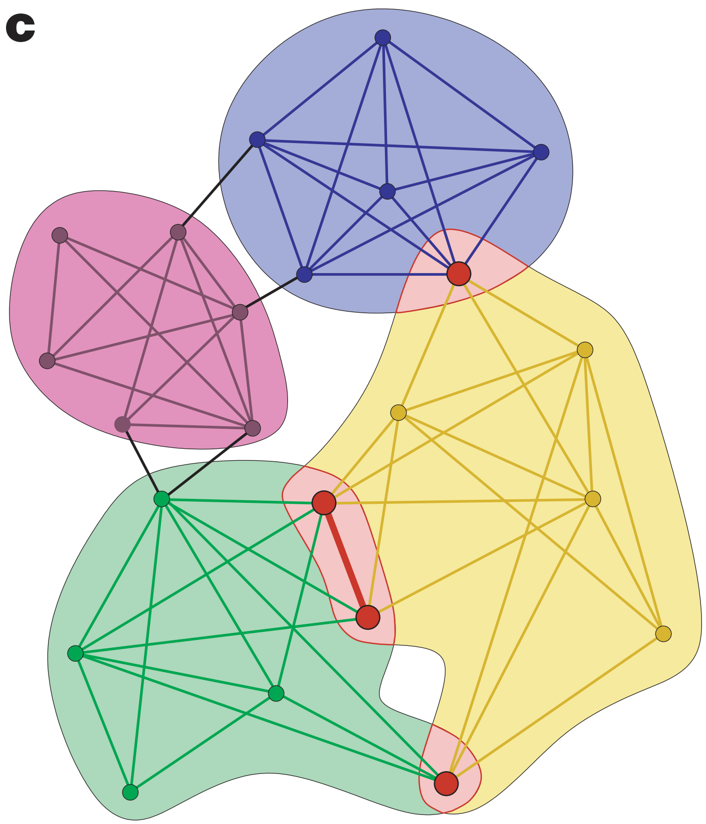
$$c(i) = \frac{\mathbf{A}^3_{ii}}{(\mathbf{A}\mathbf{1}\mathbf{A})_{ii}}$$

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

```
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 435 814–818, 2005



Rapid prototyping, clique percolation

```
clique.community <- function(graph, k) {
 clq <- cliques(graph, min=k, max=k)
 edges <- c()
 for (i in seq(along=clq)) {
   for (j in seq(along=clq)) {
     if ( length(unique(c(clq[[i]],
               clq[[j]]))) == k+1 ) {
       edges <- c(edges, c(i,j)-1)
     }
   }
 }
 clq.graph <- simplify(graph(edges))
 V(clq.graph)$name <-
    seq(length=vcount(clq.graph))
 comps <- decompose.graph(clq.graph)

 lapply(comps, function(x) {
   unique(unlist(clq[ V(x)$name ]))
 })
}
```

Acknowledgement

Tamás Nepusz

Peter McMahan, the BLISS, Walktrap and Spinglass projects

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

The R project

Hungarian Academy of Sciences

The OSS community in general

More information

`http://igraph.sf.net`

Please send your comments, questions, feature requests, code (!) to the `igraph-help` mailing list. (See *Community* on the homepage.)