

Complex Networks

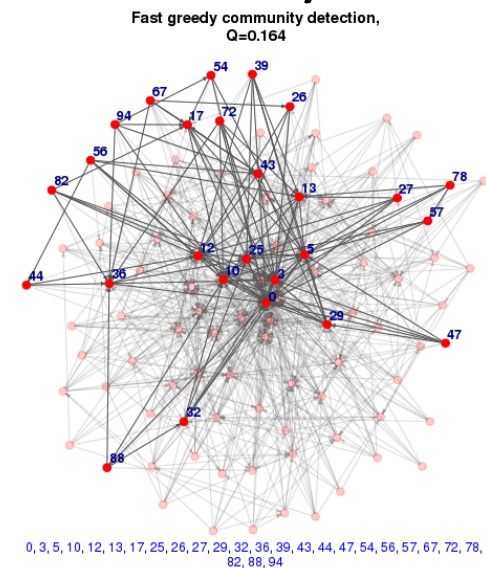
tools for analyzing networks (igraph)

2015.10.05(Mon)

igraph

- igraph is 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.
- The following three are available:
 - igraph R package
 - python-igraph
 - igraph C library

<http://igraph.org>



tutorials

- tutorial of python-igraph
 - <http://igraph.org/python/doc/tutorial/tutorial.html>
- tutorials of R
 - <http://cran.r-project.org/other-docs.html> (many tutorials in English and other languages)

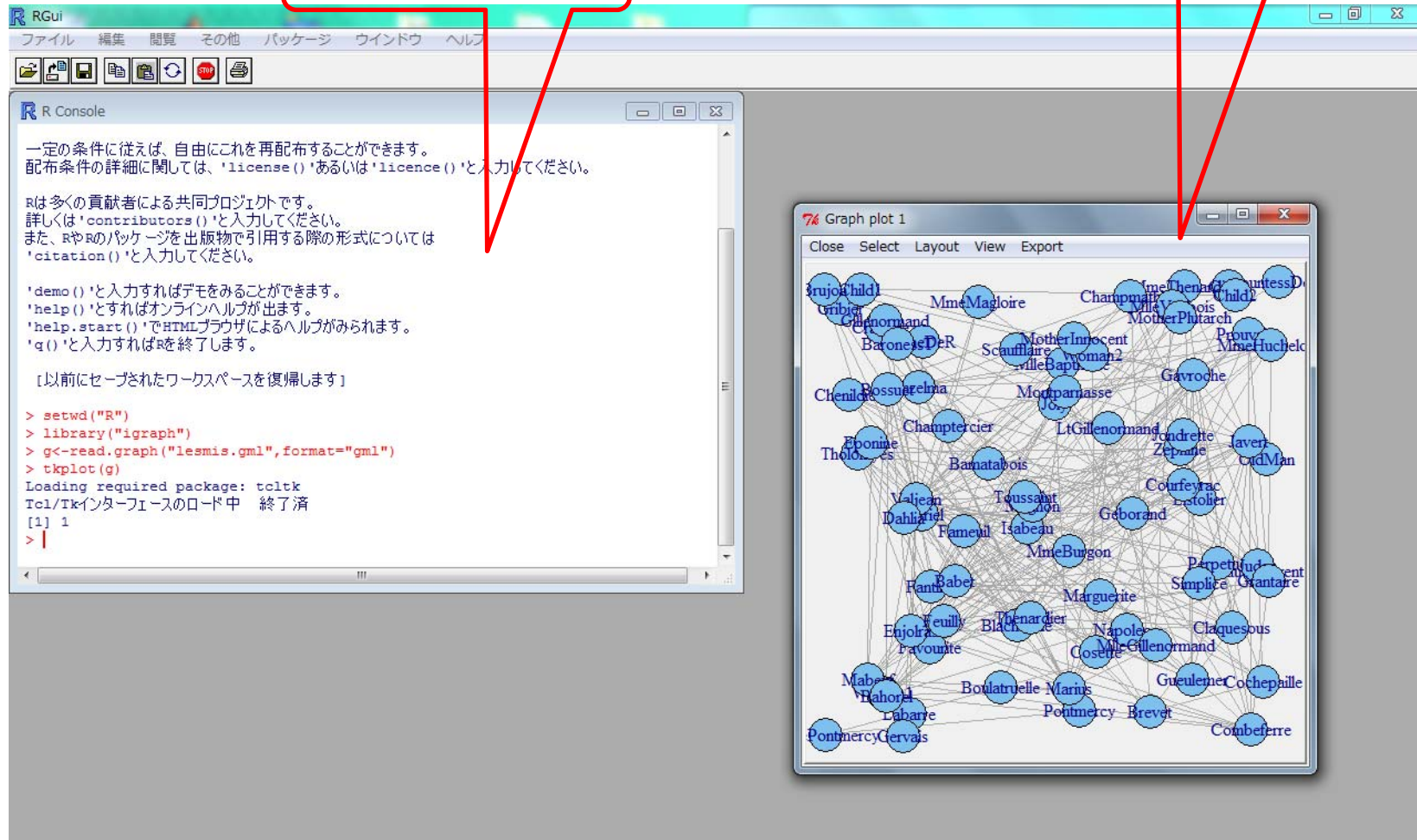
procedure for analyzing network

1. create graph object
2. layout the network
3. ranking
4. metrics
5. community detection
6. export





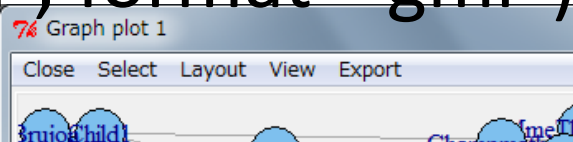
0. starting igraph

main

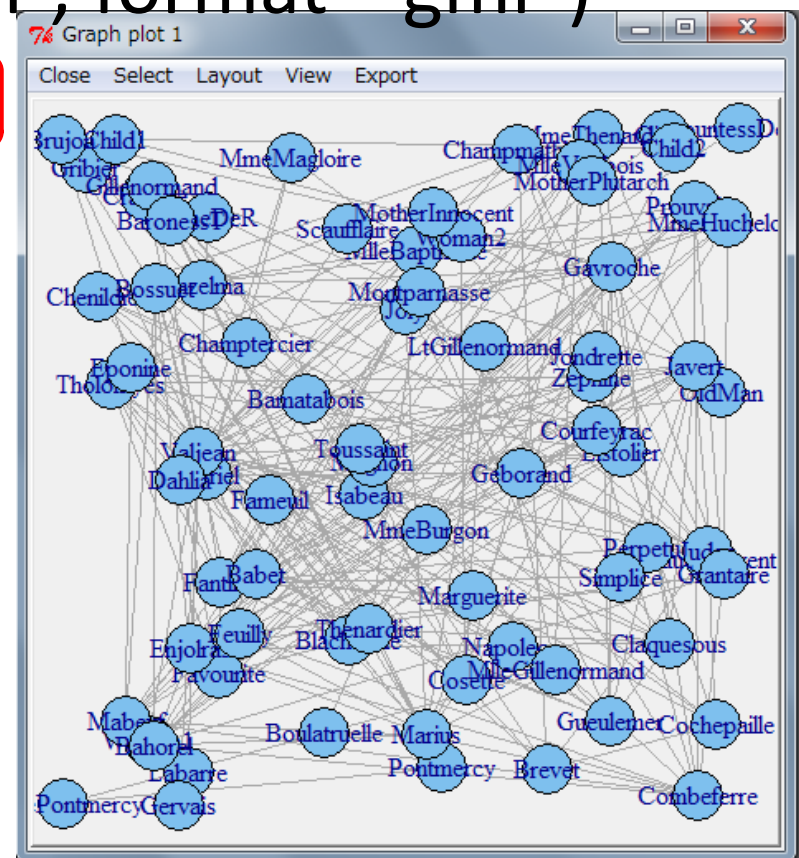
graph layout



1. create graph object

- `library("igraph")`  start igraph
 - `setwd("R")`  set directory
 - `g<-read.graph("lesmis.gml", format="gml")`  create graph
 - `tkplot(g)`  open graph window
- 

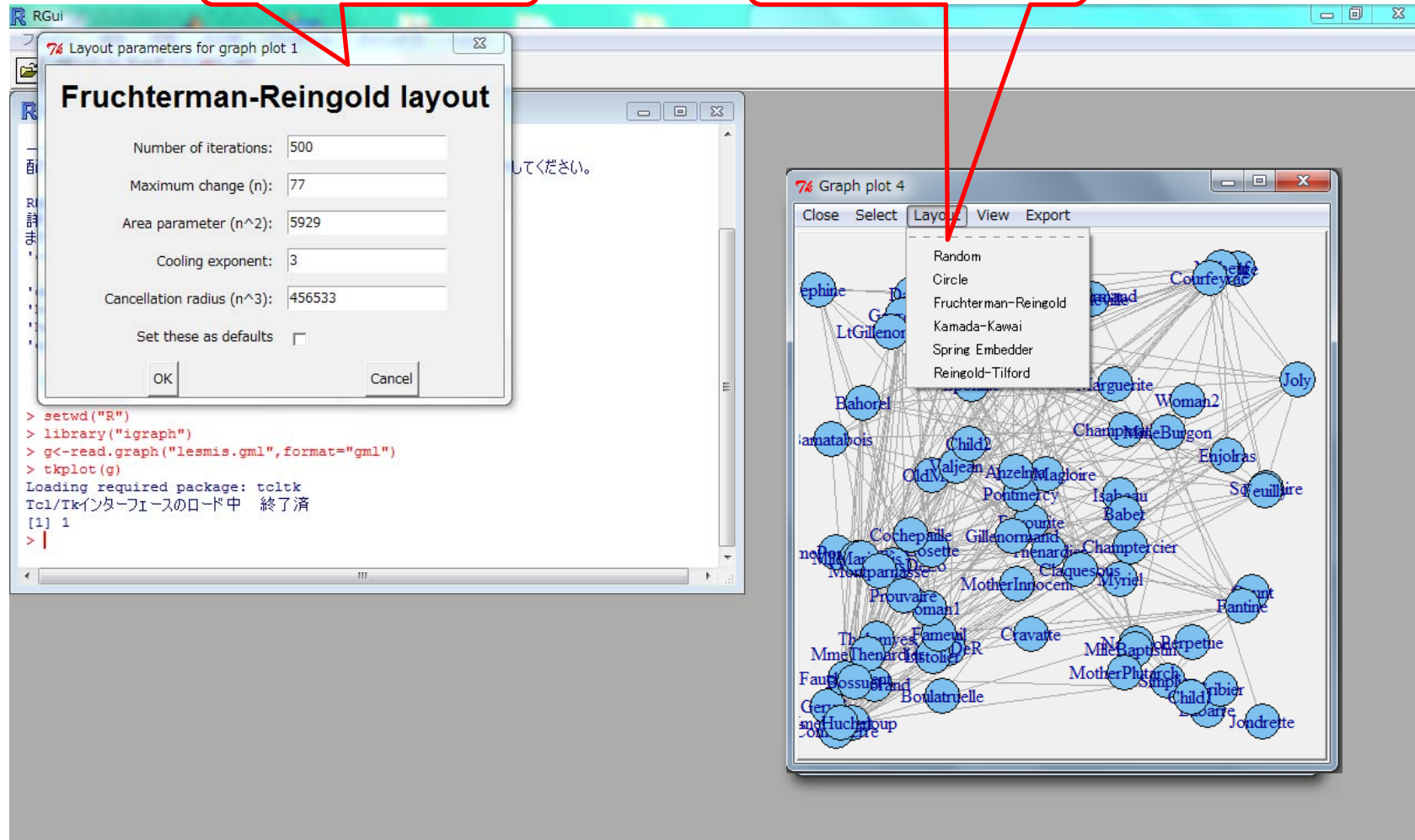
lesmis.gml is available at
Mark Newman's Website



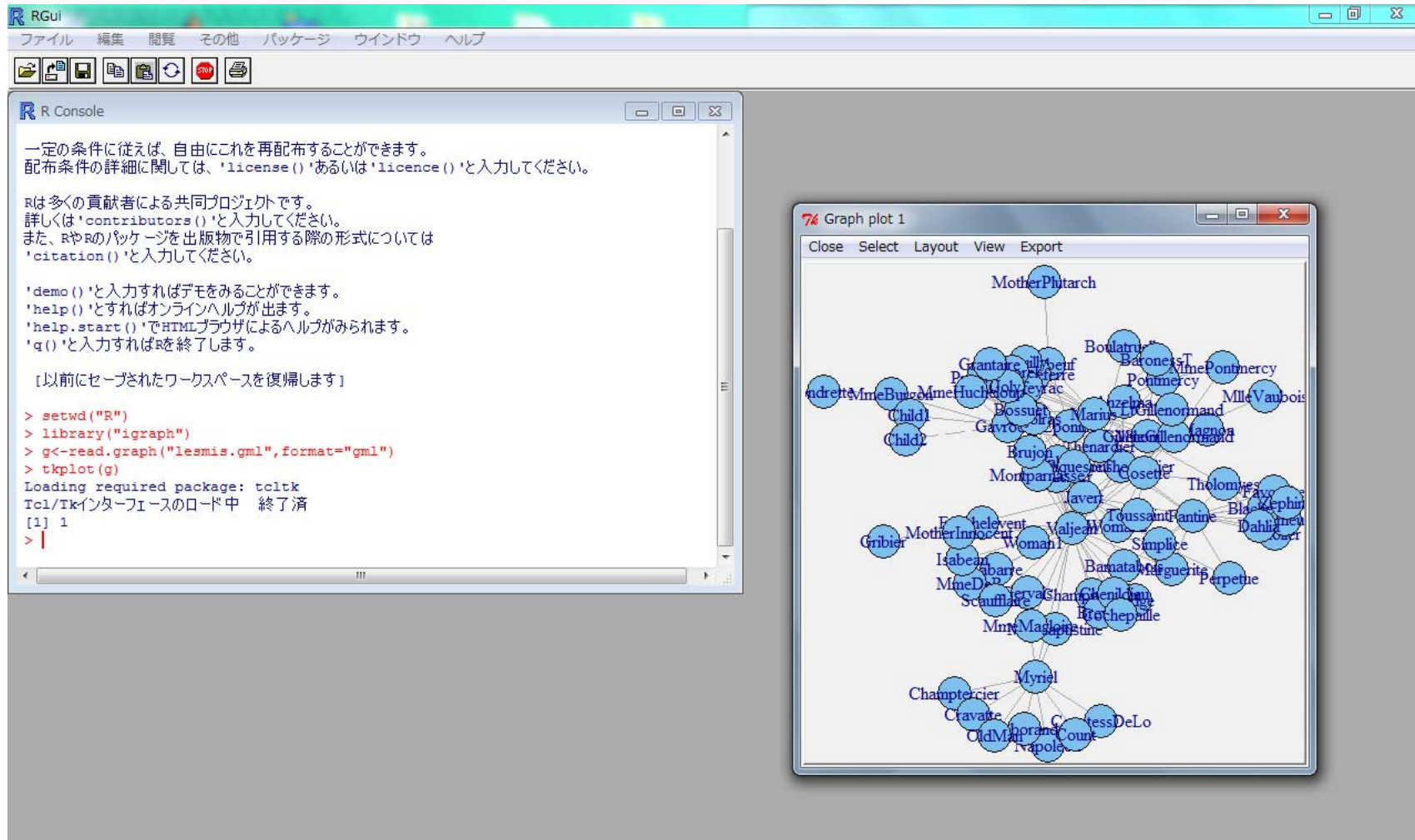
2. layout the network

set parameters

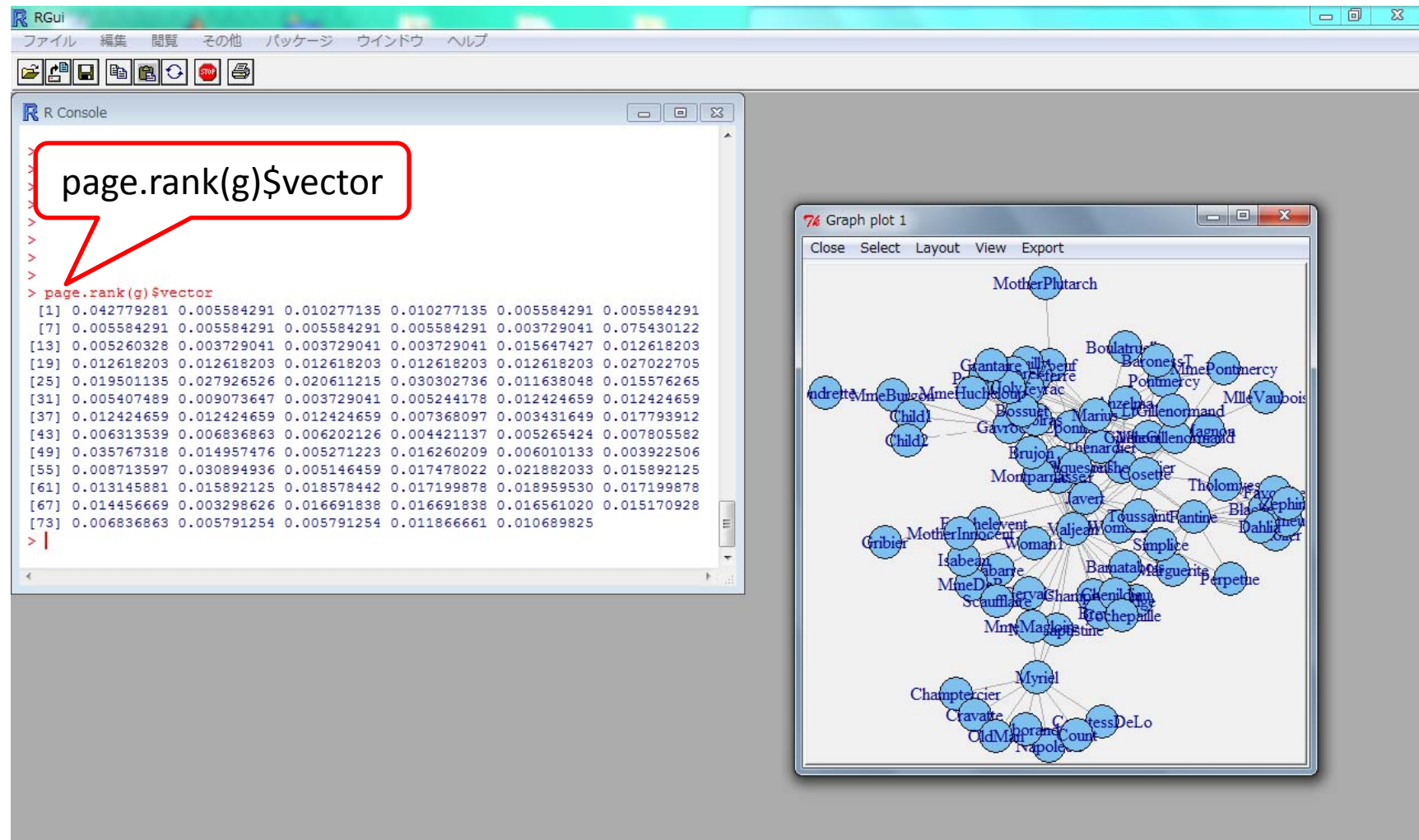
choose layout



2. layout the network



3. ranking




- original gml file contain labels

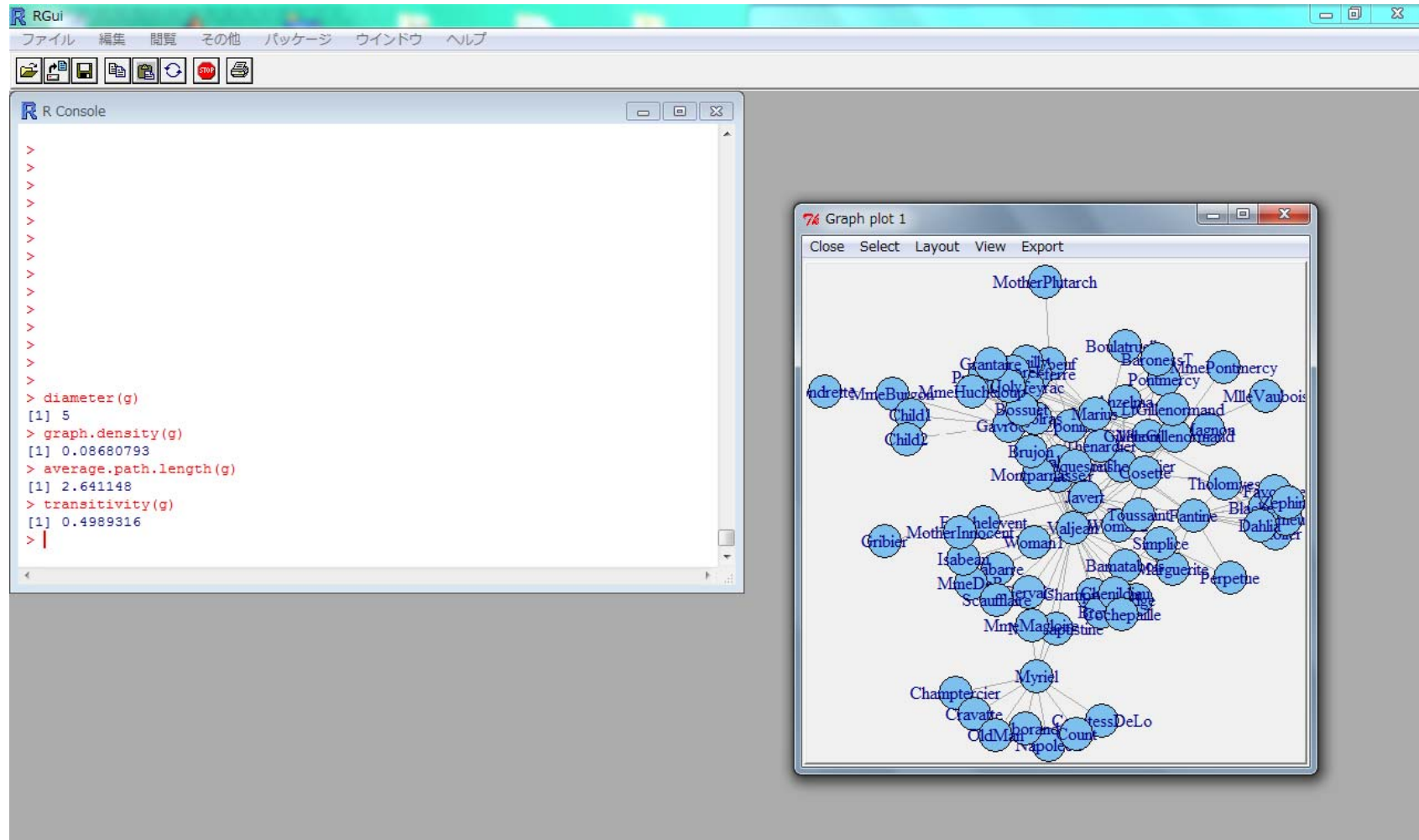
- original gml file contain labels



4. metrics

- `diameter(g)`
- `graph.density(g)`
- `average.path.length(g)`
- `transitivity(g)`  `clustering coefficient`
- `help`
 - `??rank`
 - `help("page.rank")`

4. metrics



5. community detection

modularity optimization

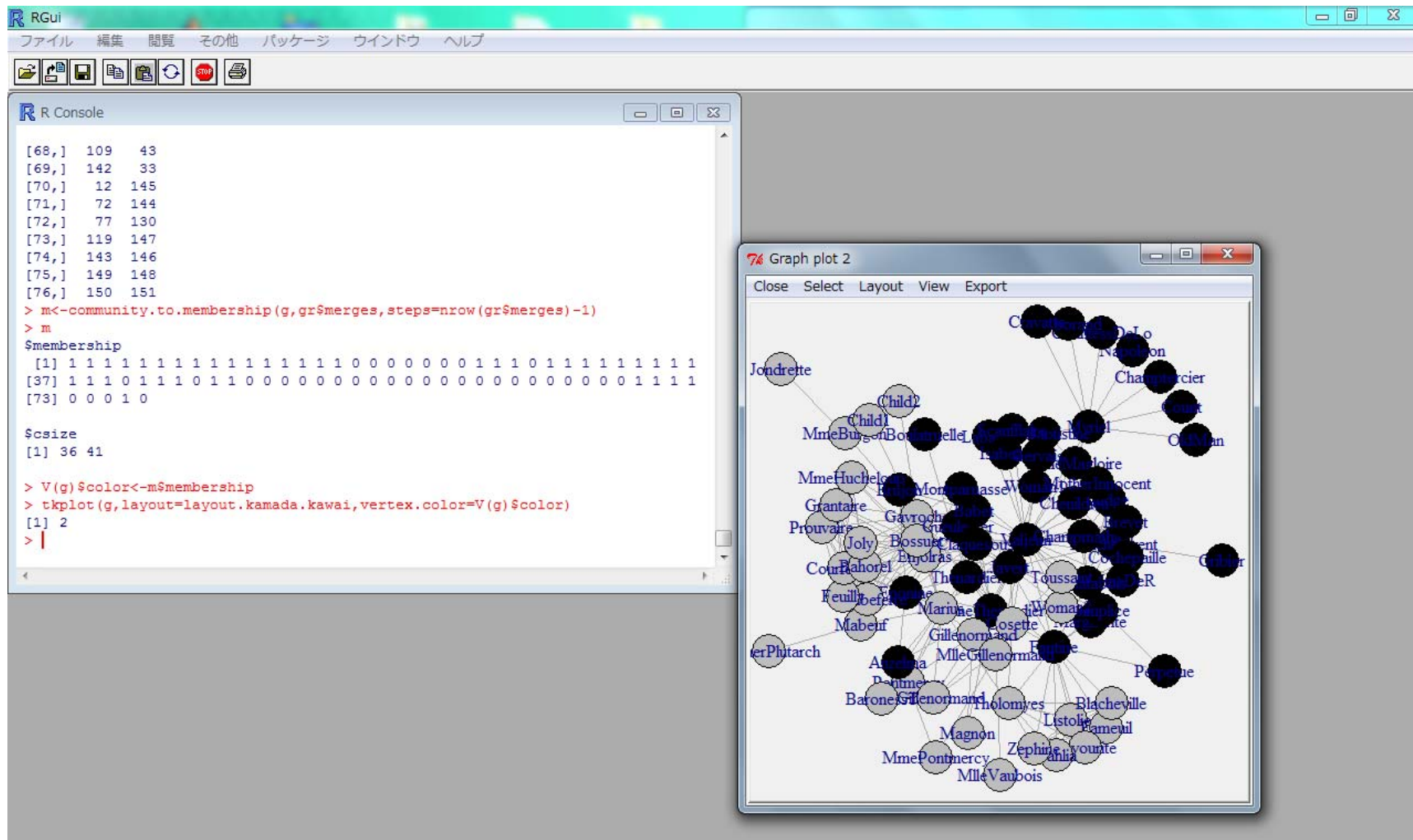
store membership and size

```
> gr<-fastgreedy.community(g)
> m<-community.to.membership(g,gr$merges,steps=nrow(gr$merges)-1)
> m
$membership
[1] 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 1 0 0 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1
$size
[1] 17 17
> V(g)$color<-m$membership
> V(g)$color
[1] 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 1 0 0 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1
> tkplot(g,layout=layout.kamada.kawai,vertex.color=V(g)$color)
[1] 2
>
```

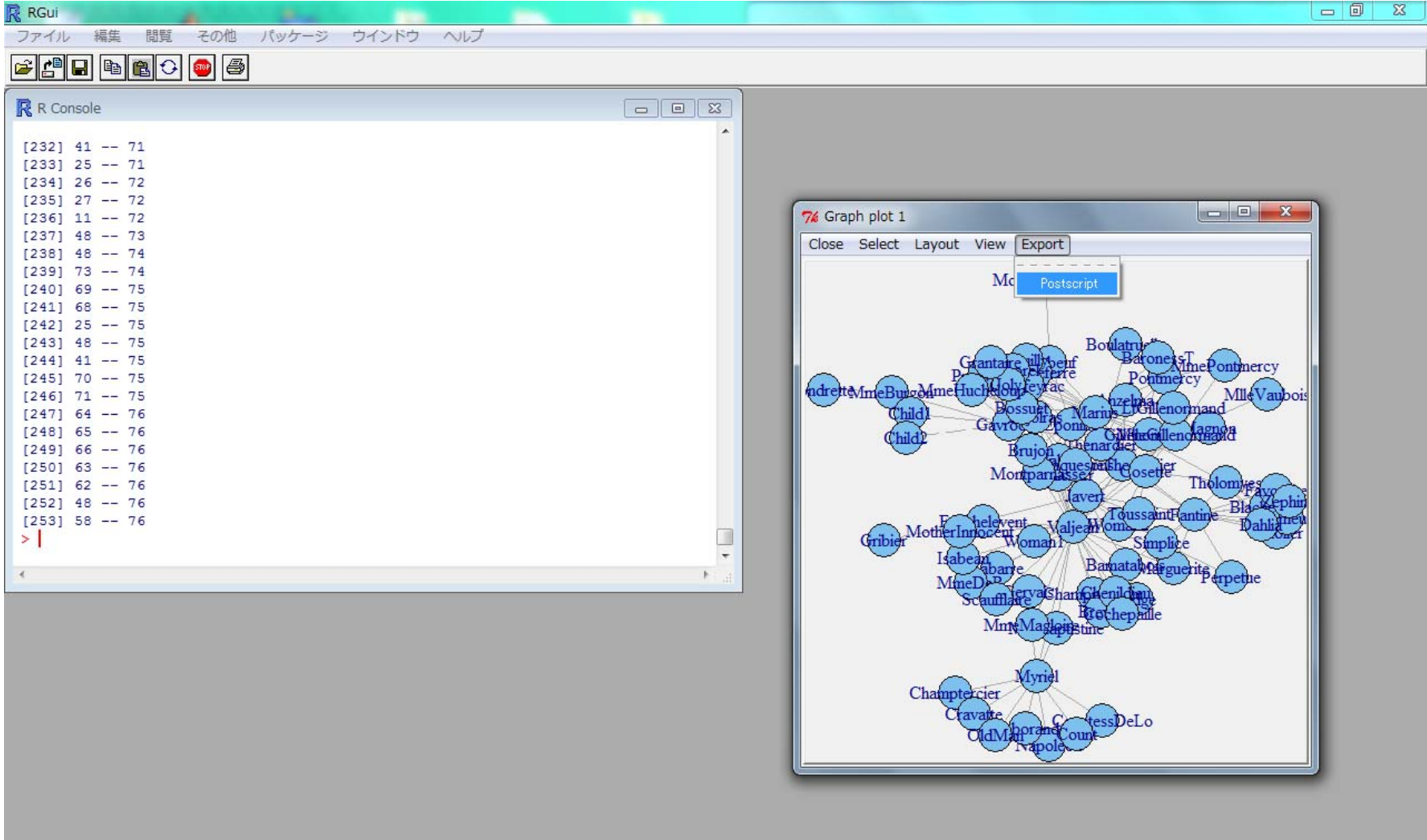
store membership

visualize network

5. community detection



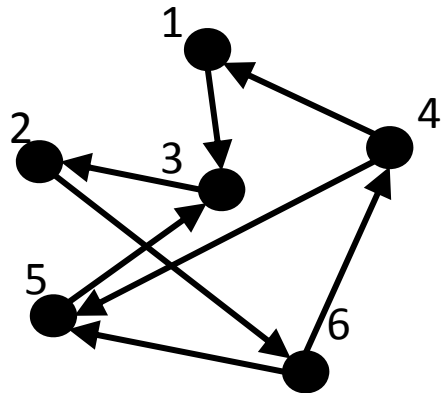
6. export



create from adjacency matrix(1)

- from adjacency matrix

```
> a <- matrix(c(0,0,0,1,0,0,  
                0,0,1,0,0,0,  
                1,0,0,0,1,0,  
                0,0,0,0,0,1,  
                0,0,0,1,0,1,  
                0,1,0,0,0,0),nrow=6,byrow=TRUE)
```



$$A = \begin{matrix} & \begin{matrix} & & & \textcolor{red}{j} & & \end{matrix} \\ \begin{matrix} \textcolor{red}{i} \\ \end{matrix} & \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix} \end{matrix}$$

create from adjacency matrix(2)

```
> ga <- graph.adjacency(t(a))
```

```
> ga
```

Vertices: 6

Edges: 8

Directed: TRUE

Edges:

[0] 0 -> 2

[1] 1 -> 5

[2] 2 -> 1

[3] 3 -> 0

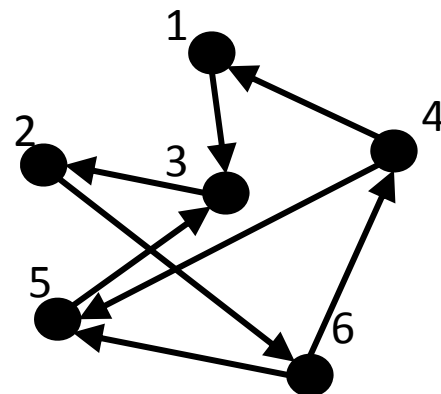
[4] 3 -> 4

[5] 4 -> 2

[6] 5 -> 3

[7] 5 -> 4

transposition



In igraph,

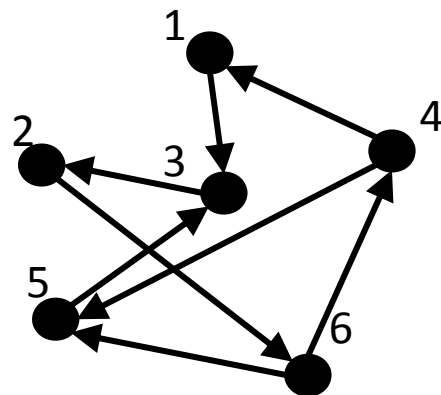
ID starts from 0 &

direction is from i to j

create from edge list

```
> el <-  
  matrix(c(0,2,1,5,2,1,3,0,3,4,4,2,5,  
          3,5,4),nc=2,byrow=TRUE)
```

```
> el  
  [,1] [,2]  
[1,]  0  2  
[2,]  1  5  
[3,]  2  1  
[4,]  3  0  
[5,]  4  2  
[6,]  5  3  
[7,]  5  4
```



```
> gb<-graph.edgelist(el)
```

```
> gb
```

Vertices: 6

Edges: 8

Directed: TRUE

Edges:

[0] 0 -> 2

[1] 1 -> 5

[2] 2 -> 1

[3] 3 -> 0

[4] 3 -> 4

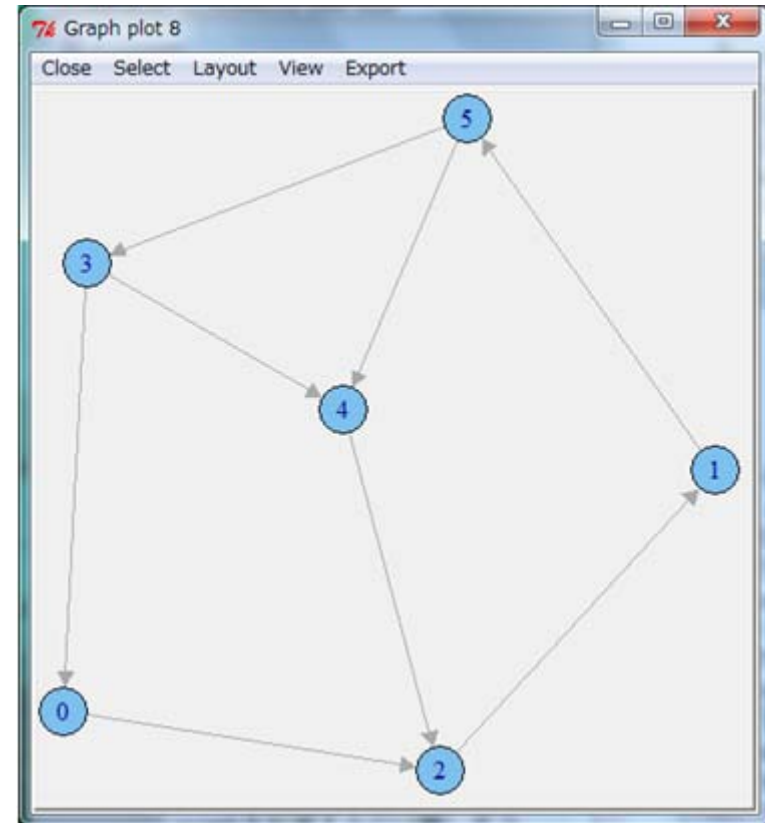
[5] 4 -> 2

[6] 5 -> 3

[7] 5 -> 4

layout the network

- > `tkplot(ga,layout=layout.kamada.kawai)`
 - choose layout (random, circle, Fruchterman-Reingold, Kamada-Kawai)
 - deform graph
 - export (Postscript)

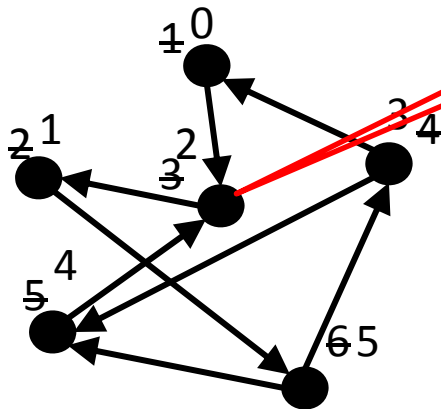


ranking

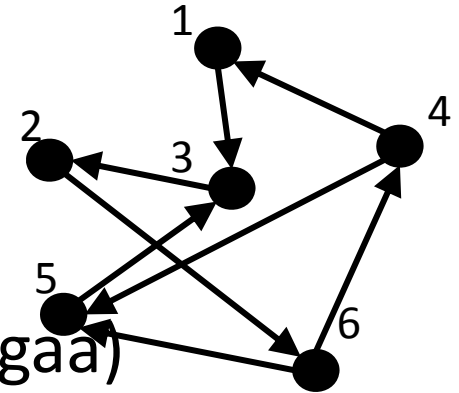
- PageRank: importance of vertices
 - the probability that a random walker will visit

```
> page.rank(ga)$vector
```

```
[1] 0.07337065 0.21643820 0.22522142  
0.11381330 0.16218395 0.20897247
```



metrics (1)



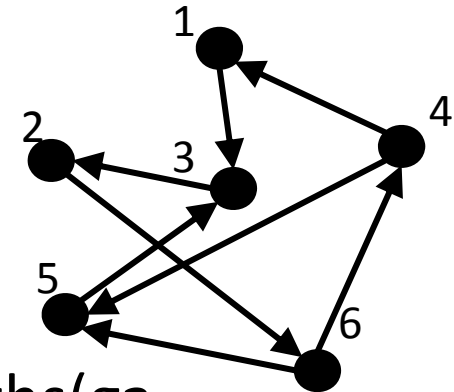
> cocitation(gaa)

	[,1]	[,2]	[,3]	[,4]	[,5]	[,6]
[1,]	0	0	0	0	1	0
[2,]	0	0	0	0	0	0
[3,]	0	0	0	0	0	0
[4,]	0	0	0	0	1	0
[5,]	1	0	0	1	0	0
[6,]	0	0	0	0	0	0

> bibcoupling(gaa)

	[,1]	[,2]	[,3]	[,4]	[,5]	[,6]
[1,]	0	0	0	0	1	0
[2,]	0	0	0	0	0	0
[3,]	0	0	0	0	0	0
[4,]	0	0	0	0	0	1
[5,]	1	0	0	0	0	0
[6,]	0	0	0	1	0	0

metrics (2)



- undirected

```
> shortest.paths(ga)
```

	[,1]	[,2]	[,3]	[,4]	[,5]	[,6]
[1,]	0	2	1	1	2	2
[2,]	2	0	1	2	2	1
[3,]	1	1	0	2	1	2
[4,]	1	2	2	0	1	1
[5,]	2	2	1	1	0	1
[6,]	2	1	2	1	1	0

- directed

```
> shortest.paths(ga,
  mode="out")
```

	[,1]	[,2]	[,3]	[,4]	[,5]	[,6]
[1,]	0	2	1	4	4	3
[2,]	3	0	3	2	2	1
[3,]	4	1	0	3	3	2
[4,]	1	3	2	0	1	4
[5,]	5	2	1	4	0	3
[6,]	2	3	2	1	1	0

metrics (3)

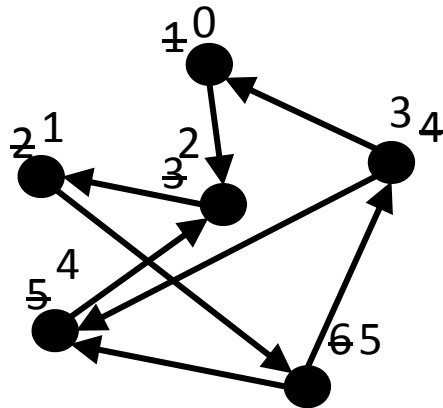
```
> average.path.length(ga)
```

```
[1] 2.433333
```

```
> average.path.length(ga,directed=FALSE)
```

```
[1] 1.466667
```

directed



```
> get.all.shortest.paths(ga,0)
```

```
[[1]]
```

```
[1] 0
```

```
[[2]]
```

```
[1] 0 2 1
```

```
[[3]]
```

```
[1] 0 2
```

```
[[4]]
```

```
[1] 0 3
```

```
[[5]]
```

```
[1] 0 3 4
```

```
[[6]]
```

```
[1] 0 2 4
```

```
[[7]]
```

```
[1] 0 3 5
```

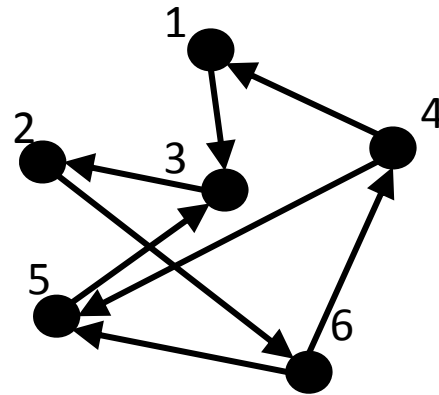
shortest
paths from 0

metrics (4)

```
> is.connected(ga)
[1] TRUE
```

```
> no.clusters(ga)
[1] 1
```

```
> clusters(ga)
$membership
[1] 0 0 0 0 0 0
$csizs
[1] 6
$no
[1] 1
```



metrics (5)

```
> graph.density(ga)
```

```
[1] 0.2666667
```

$$\rho = \frac{m}{n(n-1)} = \frac{8}{6 \cdot 5}$$

