Chapter 4
Descriptive Analysis of Network Graph Characteristics

Statistical Analysis of Network Data, with R - Eric D. Kolaczyk

Thu Nguyen

24 June, 2019

Contents

Libraries

```r
library(igraph)
library(igraphdata)
library(sand)
library(ape)
```

1 Introduction

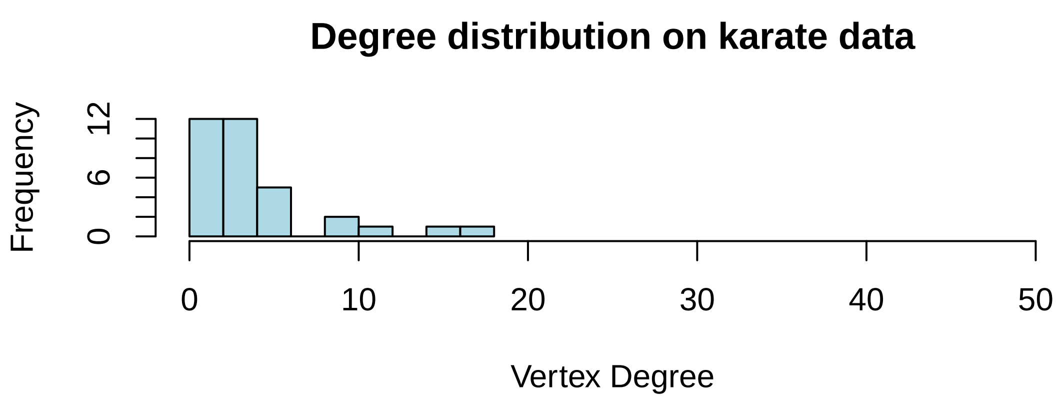
---

2 Vertex and Edge Characteristics

2.1 Vertex degree

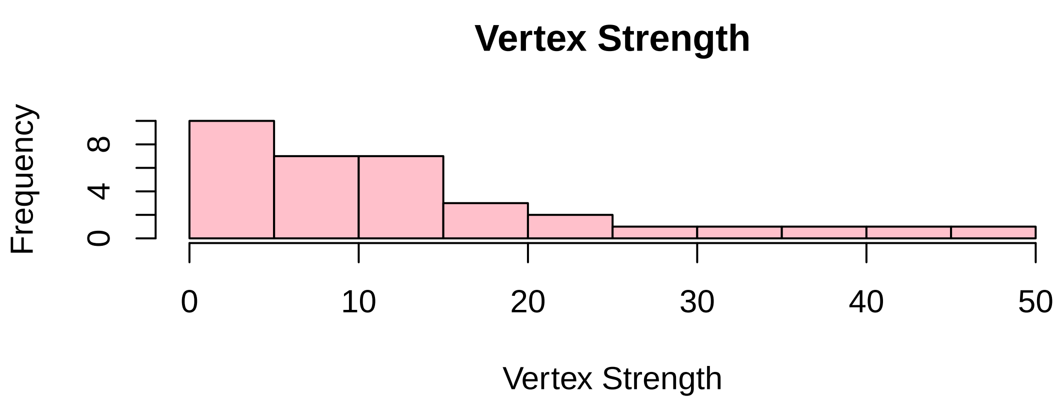
Given $G = (V, E)$, with degree d_v for vertex v, define f_d to be the fraction of vertices $v \in V$ with degree $d_v = d$. Then $\{f_d\}_{d \geq 0}$ is the **degree distribution** of G. For example:

```
data(karate)
hist(degree(karate), col = 'lightblue', xlim = c(0, 50),
     xlab = 'Vertex Degree', ylab = 'Frequency', main = 'Degree distribution on karate data')
```



For a *weighted networks*, **vertex strength** is the sum of weights of edges incident to a given vertex, by function `graph.strength()`. For example:

```
hist(graph.strength(karate), col = "pink",
     xlab = "Vertex Strength", ylab = "Frequency", main = 'Vertex Strength')
```

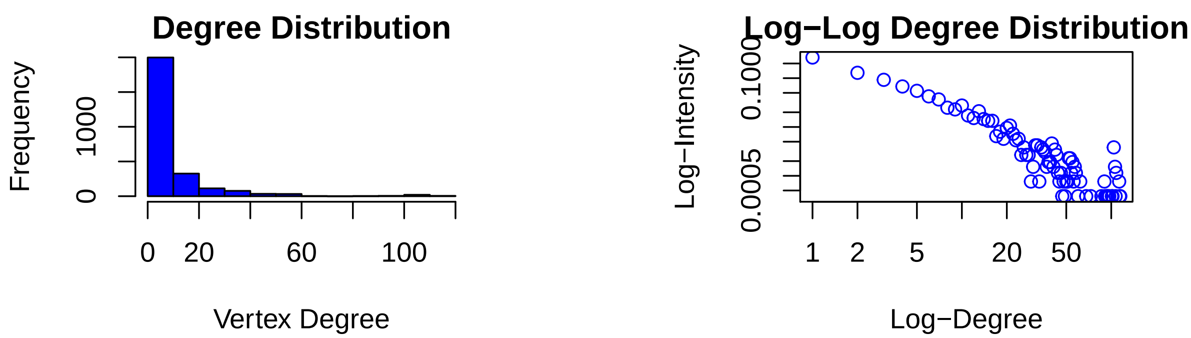


Another dataset, `yeast` from package `igraphdata`:

```
data(yeast)
nv <- vcount(yeast); ne <- ecount(yeast)
```

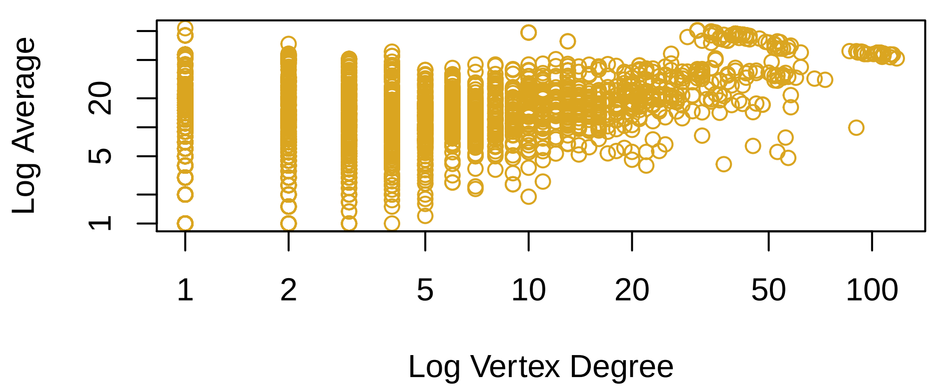
where there are 2617 vertices and 11855 edges, and the distributions of *degree* and log(*degree*), which gives a fairly linear relationship:

```
par(mfrow = c(1,2)); par(mar=c(4,5,1.5,5))
d.yeast <- degree(yeast)
hist(d.yeast, col = 'blue', xlab = 'Vertex Degree', ylab = 'Frequency',
     main = 'Degree Distribution')
dd.yeast <- degree.distribution(yeast)
d <- 1:max(d.yeast) - 1
ind <- (dd.yeast != 0)
plot(d[ind], dd.yeast[ind], log = 'xy', col = 'blue',
     xlab = 'Log-Degree', ylab = 'Log-Intensity', main = 'Log-Log Degree Distribution')
```



Alternatively, we can look at the average degree of the neighbors of a given vertex, by `graph.knn()`:

```
par(mar=c(4,5,.5,5))
a.nn.deg.yeast <- graph.knn(yeast,V(yeast))$knn
plot(d.yeast, a.nn.deg.yeast, log = "xy", col = "goldenrod",
     xlab = 'Log Vertex Degree', ylab ='Log Average')
```



2.2 Vertex Centrality

Closeness centrality measures attempts to measure that a vertex is 'central' if it is 'close' to many other vertices. Let $dist(v, u)$ be the geodesic distance, the standard approach is:

$$c_{Cl}\ (v) = \frac{1}{\sum_{u \in V} dist(v, u)}$$

Betweenness centrality measures attempts to summarize the extent to which a vertex is located 'between' other pairs of vertices. The perspective is that 'importance' relates to where a vertex is located relative to network' paths. Let $\sigma(s, t|v)$ be the total number of shortest paths between s and t passing through v, and $sigma(s, t)$ be the total number of shortest paths anywhere, the standard approach is:

$$c_B\ (v) = \sum_{s \neq t \neq t \in V} \frac{\sigma(s, t|v)}{\sigma(s, t)}$$

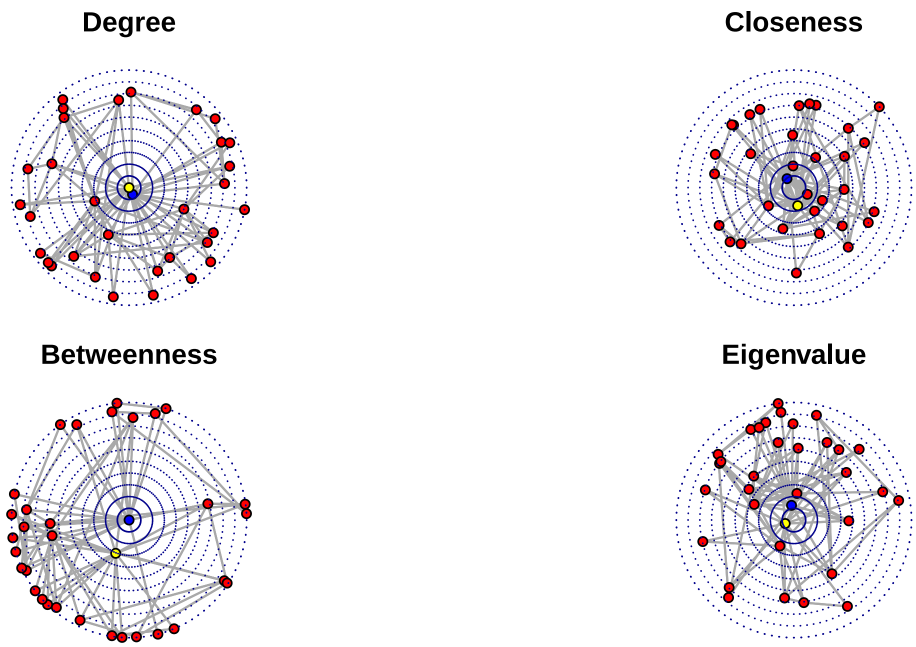
Status/Prestige/Rank: the more central a vertex's neighbors are, the more central that vertex itself is. This is typically expressed as eigenvector solutions of linear systems of equations, called **eigenvector centrality measures**. Let $\mathbf{c}_{Ei} = (c_{Ei}(1), \ldots, c_{Ei}(N_v))^T$ be the solution to $\mathbf{A}\mathbf{c}_{Ei} = \alpha^{-1}\mathbf{c}_{Ei}$, where $\mathbf{A}$ is the adjacency matrix:

$$c_{Ei}\ (v) = \alpha \sum_{\{u,v\} \in E} c_{Ei}\ (u)$$

To display vertex centrality, use `gplot.target()` from package `sna`, to get adjacency matrix: `get.adjacency()`:

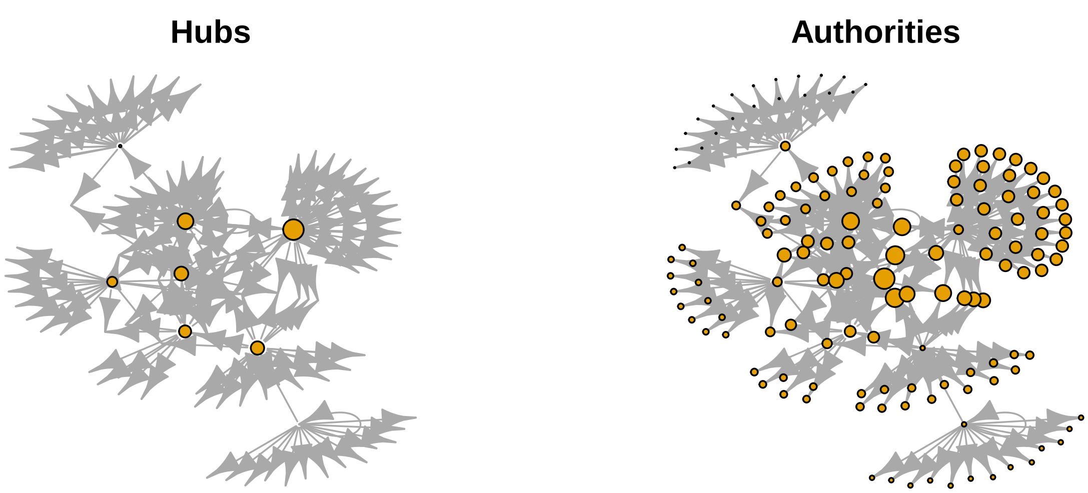
Similarly, instead of `degree(g)`, we can plot with attributes `closeness(g), betweenness(g), evcent(g)$vector` for the 3 measures above respectively.

```
library(network); library(sna)
par(mfrow = c(2,2)); par(mar=c(0,0,1,0))
A <- get.adjacency(karate, sparse = FALSE)
g <- network::as.network.matrix(A)
sna::gplot.target(g, degree(g), main = 'Degree',
                  circ.lab = FALSE, circ.col = 'darkblue', usearrows = FALSE,
                  vertex.col = c('blue', rep('red', 32), 'yellow'), edge.col = 'darkgray')
sna::gplot.target(g, closeness(g), main = 'Closeness',
                  circ.lab = FALSE, circ.col = 'darkblue', usearrows = FALSE,
                  vertex.col = c('blue', rep('red', 32), 'yellow'), edge.col = 'darkgray')
sna::gplot.target(g, betweenness(g), main = 'Betweenness',
                  circ.lab = FALSE, circ.col = 'darkblue', usearrows = FALSE,
                  vertex.col = c('blue', rep('red', 32), 'yellow'), edge.col = 'darkgray')
sna::gplot.target(g, evcent(g), main = 'Eigenvalue',
                  circ.lab = FALSE, circ.col = 'darkblue', usearrows = FALSE,
                  vertex.col = c('blue', rep('red', 32), 'yellow'), edge.col = 'darkgray')
```



Extension from *undirected* to *directed* graphs are straightforward. For example: new dataset AIDS blog network:

```
l <- layout.kamada.kawai(aidsblog)
par(mfrow=c(1,2)); par(mar=c(0,0,1,0))
plot(aidsblog, layout = l, main = 'Hubs', vertex.label = '',
     vertex.size = 10*sqrt(hub.score(aidsblog)$vector))
plot(aidsblog, layout = l, main = 'Authorities', vertex.label = '',
     vertex.size = 10*sqrt(authority.score(aidsblog)$vector))
```



2.3 Characterizing Edges

```
eb <- edge.betweenness(karate)
E(karate)[order(eb, decreasing = T)[1:3]]
```

```
## + 3/78 edges from 4b458a1 (vertex names):
## [1] Actor 20--John A   Mr Hi   --Actor 20 Mr Hi   --Actor 32
```

3 Characterizing Network Cohesion

Questions to consider regarding **network cohesion**:

- Do friends of a given person in a social network tend to be friends of another as well?
- Does the structure of WWW pages tend to separate with respect to distinct types of content?

3.1 Subgraphs and Censuses

Cliques are complete subgraphs, and thus subsets of fully cohesive vertices, like $K_1, K_2, K_3, \ldots$, by `cliques()`. Example from `karate` where there are 34 isolated vertices (K_1), 78 pairs/K_2, 45 triangles/K_3, and so on:

```
table(sapply(cliques(karate), length))
```

```
##
##  1  2  3  4  5
## 34 78 45 11  2
```

The 2 biggest cliques of size 5, both having the head instructor, are:

```
cliques(karate)[sapply(cliques(karate), length) == 5]
```

```
## [[1]]
## + 5/34 vertices, named, from 4b458a1:
## [1] Mr Hi    Actor 2  Actor 3  Actor 4  Actor 14
##
## [[2]]
## + 5/34 vertices, named, from 4b458a1:
## [1] Mr Hi   Actor 2 Actor 3 Actor 4 Actor 8
```

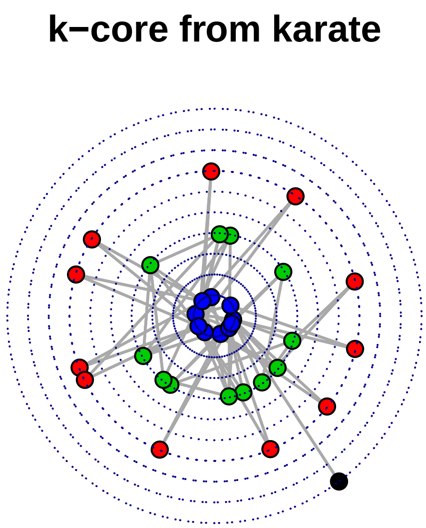
Maximal cliques: cliques that are not subsets of a larger clique, `maximal.cliques()`

```
table(sapply(maximal.cliques(karate), length))
```

```
##
##  2  3  4  5
## 11 21  2  2
```

k**-core** of G is *maximal* subgraph for which all vertex degrees are $\geq k$. The notion of cores is popular in visualization in helping decomposing a network into 'layers', `graph.coreness()`:

```
library(sna)
cores <- graph.coreness(karate)
par(mar=c(0,0,1,0))
sna::gplot.target(g, cores, circ.lab = FALSE, circ.col = 'darkblue', usearrows = FALSE,
                  vertex.col = cores, edge.col = 'darkgray', main = 'k-core from karate')
```




```
detach('package:sna'); detach('package:network')
```

3.2 Density and Related Notions of Relative Frequency

Density: frequency of realized edges relative to potential edges. For a *simple, undirected* G, with n vertices and m edges, and subgraph H:

$$den(H) = \frac{m}{\frac{n(n-2)}{2}}$$

For a *simple, directed* G:

$$den(H) = \frac{m}{n(n-1)}$$

If $H = G$, $den(H) = den(G)$. If $H = H_v : V(H_v) = N(v)$ (set of neighbors of v), $den(H)$ measures the density in the immediate neighborhood of v. To get $N(v)$: `neighborhood()`, subgraph: `induced.subgraph()`, and density: `graph.density()`. For example:

```
# Getting subgraph of neighbors of the instructor, vertex 1, and admin, vertex 34
ego.instr <- induced.subgraph(karate, neighborhood(karate, 1, 1)[[1]])
ego.admin <- induced.subgraph(karate, neighborhood(karate, 1, 34)[[1]])
den.k <- graph.density(karate);
den.i <- graph.density(ego.instr); den.a <- graph.density(ego.admin)
```

Whereas the whole network density is low, at 0.1390374, densities at both the instructor = 0.25 and the admin = 0.2091503 are quite higher, which is consistent with the disparity in the number of within-versus between-fraction edges.

Clustering coefficient, given $\tau_\Delta(G)$: the number of triangles in G, and $\tau_3(G)$: the number of connected triples (or 2-star):

$$cl_T(G) = \frac{3\ \tau_\Delta(G)}{\tau_3(G)}$$

$c|_T$ also **transitivity**, or 'fraction of transitive triples', measuring global clustering for the entire network, `transitivity()`:

```
transitivity(karate)
```

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

Locally, let $\tau_\Delta(v)$: the number of triangles in G containing v, and $\tau_3(v) = \binom{d_v}{2}$: the number of connected triples for which both 2 edges are incident to v. Then for v such that $\tau_3(v) > 0$, the **local clustering coefficient** is:

$$cl(v) = \frac{\tau_\Delta(v)}{\tau_3(v)}$$

```
transitivity(karate, 'local', vids = c(1,34))
```

```
## [1] 0.1500000 0.1102941
```

Unique to *directed* graph is **reciprocity**. There are 2 approaches: through dyads or directed edges:

```
def <- reciprocity(aidsblog, mode = 'default'); rat <- reciprocity(aidsblog, mode = 'ratio')
print(cat('Dyads: ', def, '\n', 'Directed edges: ', rat, '\n'))
```

```
## Dyads:  0.03243243
##  Directed edges:  0.01648352
## NULL
```

3.3 Connectivity, Cuts, and Flows

Recall *components*, by `decompse.graph()`, and getting vertex count by `vcount()`. From `yeast` dataset, there is *giant component* of 2375 vertices or 90% of all vertices, 0 isolated vertices, 63 pairs, ...

```
comps <- decompose.graph(yeast)
table(sapply(comps, vcount))
```

```
##
##    2    3    4    5    6    7 2375
##   63   13    5    6    1    3    1
```

```
yeast.gc <- decompose.graph(yeast)[[1]]  # giant component
ave.path <- round(average.path.length(yeast.gc),2)
diam <- diameter(yeast.gc)
tran <- round(transitivity(yeast.gc),2)
```

The giant component have a few characteristics of a ***small world*** model such as small *shortest-path distance between pairs of vertices* of 5.1, and small *longest paths* of 15, and high *clustering* of 0.47, indicating that close to 50% of connected triples form triangles.

Recall from Graph Theory of Math 154, k**-vertex-connected** and k**-connected**, and that $\kappa(u,v) \leq \lambda(u,v) \leq \delta(G)$. To see connectivity: `vertex.connectivity()`, and `edge.connectivity()`:

```
v.con <- vertex.connectivity(yeast.gc)
e.con <- edge.connectivity(yeast.gc)
print(paste('Vertex connectivity: ', v.con, '; Edge connectivity: ', e.con))
```

```
## [1] "Vertex connectivity:  1 ; Edge connectivity:  1"
```

Vertex-cut (edge-cut) is a set vertices (edges) that disconnect G. Also, if only of size 1, *cut vertex* or *articulation point*, by `articulation.points()`:

```
yeast.cut.vertices <- articulation.points(yeast.gc)
print(paste('The number of cut vertices:', length(yeast.cut.vertices)))
```

```
## [1] "The number of cut vertices: 350"
```

Menger's theorem vertex form:

$$\min\{ab - \text{separator}\} = \max\{\text{pariwise internally disjoint paths between } a \text{ and } b\}$$

`R`: `shortest.paths()`, `graph.maxflow()`, and `graph.mincut()`.

These concepts extend naturally to *directed* graphs:

```r
w.con <- is.connected(aidsblog, mode = 'weak')
s.con <- is.connected(aidsblog, mode = 'strong')
print(paste0('Weak: ', w.con, '; Strong: ', s.con))
```

```
## [1] "Weak: TRUE; Strong: FALSE"
```

Strongly connected components, from `clusters()`:

```r
aidsblog.scc <- clusters(aidsblog, mode = 'strong')
table(aidsblog.scc$csize)
```

```
##
##   1   4
## 142   1
```

---

4 Graph Partitioning

4.1 Hierarchical Clustering

There are 2 main approaches:

- **agglomerative**: successive coarsening of paritions through merging
- **divisive**: successive coarsening of paritions through splitting

To do clustering: `fastgreedy.community()`, `sizes()`, `membership()`:

```
kc <- fastgreedy.community(karate)
sizes(kc)
```

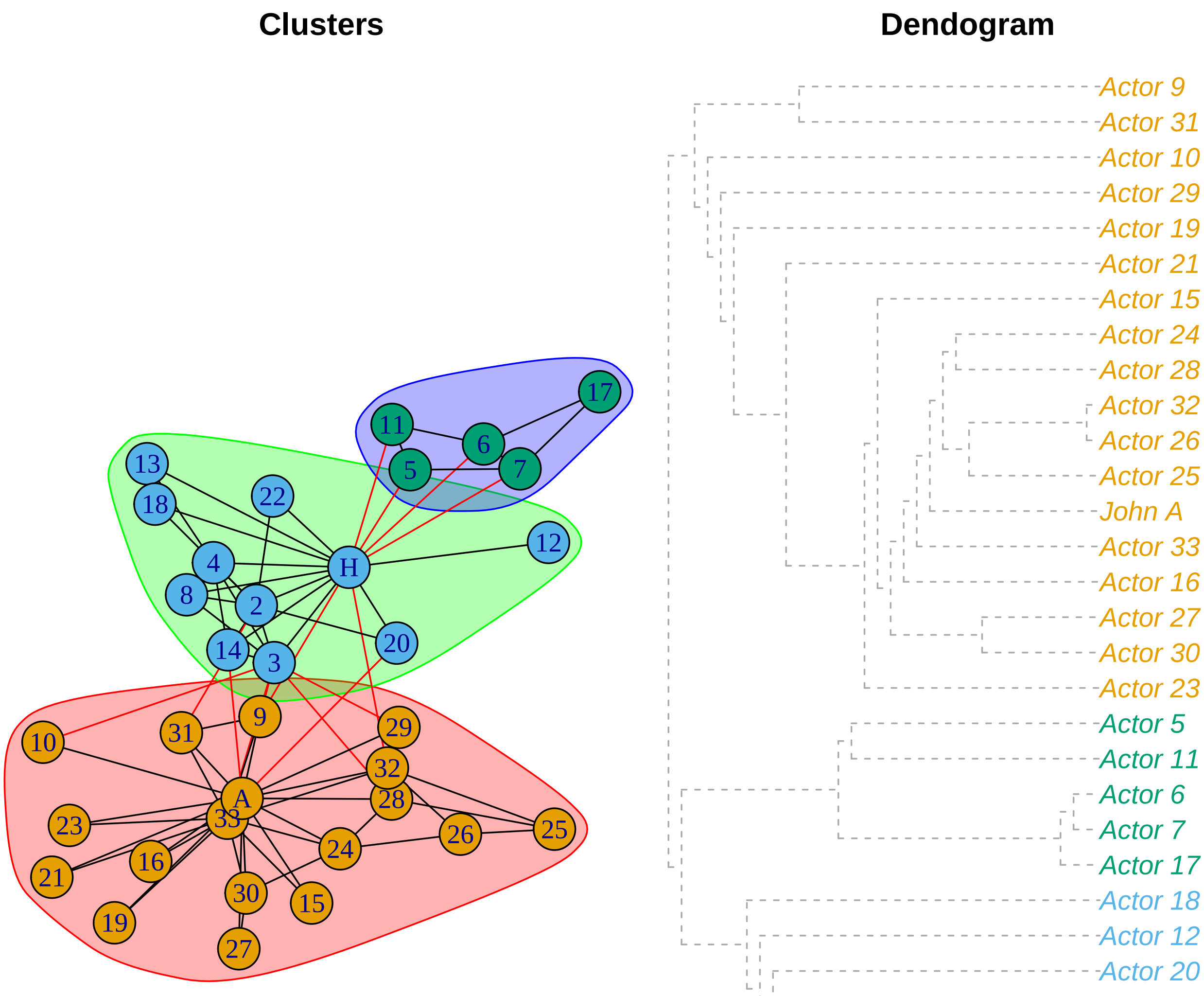
```
## Community sizes
##  1  2  3
## 18 11  5
```

```
membership(kc)
```

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

`dendPlot()` from package `ape` for dendogram:

```
library(ape)
par(mfrow=c(1,2)); par(mar=c(0,0,1,0))
plot(kc, karate, main = 'Clusters')
dendPlot(kc, mode = 'phylo', main = 'Dendogram')
```


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

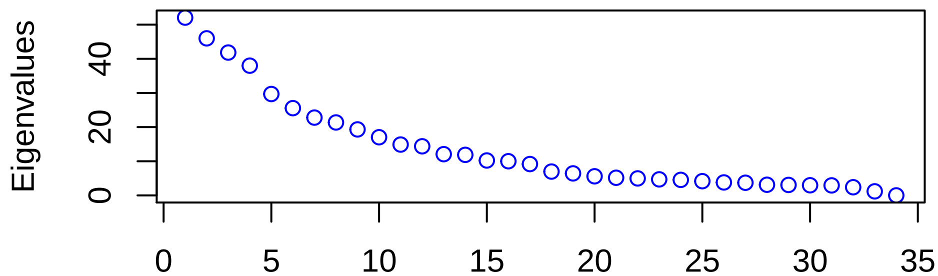
4.2 Spectral Partitioning

Spectral graph theory: connectivity of G is associated with the eigen-analysis of certain matrices. Define *Laplacian* L of G with adjacency matrix A, and diagonal degree matrix $D = diag[(d_v)]$:

$$L = D - A$$

Then the number of components in G is directly related to the number of non-zero eigenvalues of L. In `R`: `graph.laplacian()`, `eigen()`, `get.vertex.attribute()`:

```
par(mar=c(2,5,.5,5))
k.lap <- graph.laplacian(karate)
eig.anal <- eigen(k.lap)
plot(eig.anal$values, col = 'blue', ylab = 'Eigenvalues')
```

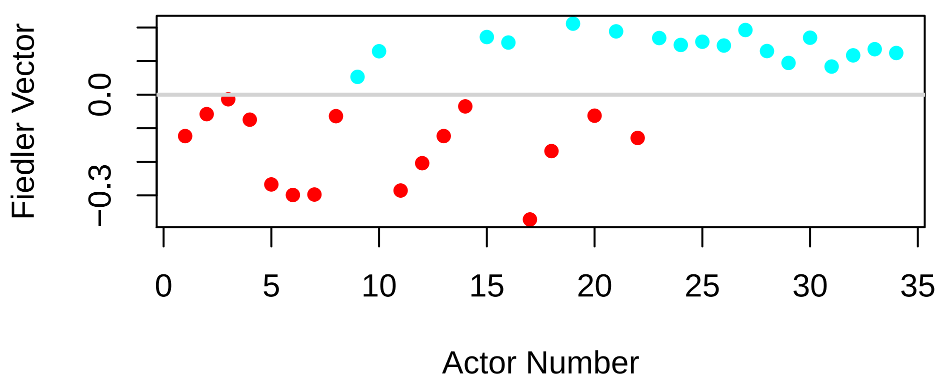


Extracting *Fiedler vector* $\mathbf{x}_2$ from:

$$S = \{v \in V : \mathbf{x}_2(v) \geq 0\}, \quad \text{and} \quad \bar{S} = \{v \in V : \mathbf{x}_2(v) < 0\}$$

which partition into 2 subsets of vertices, for example:

```
par(mar=c(4,5,.5,5))
f.vec <- eig.anal$vectors[, 33]
faction <- get.vertex.attribute(karate, 'Faction')
f.colors <- as.character(length(faction))
f.colors[faction == 1] <- 'red'; f.colors[faction == 2] <- 'cyan'
plot(f.vec, pch = 16, xlab = 'Actor Number', ylab = 'Fiedler Vector', col = f.colors)
abline(0, 0, lwd = 2, col = 'lightgray')
```



4.3 Validation of Graph Partitioning

---

5 Assortativity and Mixing

```r
assortativity.degree(yeast)
```

```
## [1] 0.4610798
```
