Further Topics in Social Network Analysis

Cohesion in Social Networks

Dominik Batorski Michał Bojanowski Bartosz Chroł Kamil Filipek

Interdisciplinary Centre for Mathematical and Computational Modelling, University of Warsaw

WARSAW SCHOOL of DATA ANALYSIS

Contents

HUMAN CAPITAL NATIONAL COHESION STRATEGY

UNIVERSITY OF WARSAW

ICS RUG / UU / RU

EUROPEAN UNION EUROPEAN SOCIAL FUND

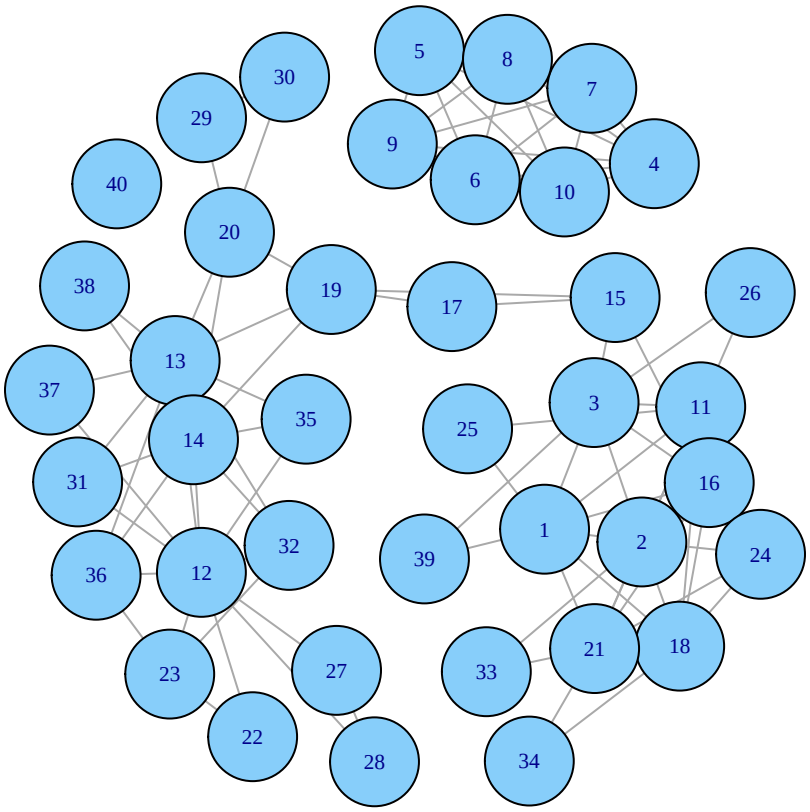
Project co-financed by the European Union under the European Social Fund

1 Introduction

The idea of cohesion has been widely studied in contemporary social science. Researchers from diverse backgrounds are trying to identify factors encouraging individuals to keep dense and tight relations embedded in larger structures e.g. firms, local communities, societies. In this chapter we focus on the idea of cohesive subgroup developed in social network analysis. We begin with general definition of cohesion and than introduce clique and community measures of cohesion.

In our analysis we will use network of judges – there is an edge between two judges if they were judging at least one case together.

```
library("igraph")
library("isnar")
data(judge_net)
graph <- judge_net
par(mar = rep(0,4))
plot(graph)
```



Cohesion is one of few elementary properties of social networks. Cohesiveness is a property of tightly connected actors within the larger network structure. Actors with strong, intense, frequent, direct or positive relations may compose cohesive subgroups. Therefore, in social network analysis the definition of cohesion is based on certain properties of the ties among the actors. In social science literature, definition of cohesion often overlaps with definitions of social group (Wasserman and Faust 1994). There are numerous ways to conceptualize or define subgroups in social science. To avoid theoretical pitfalls, Wasserman and Faust identified four general properties of cohesive subgroups. They assumed that these properties allow to develop methods that focus on social network features (Wasserman and Faust 1994). Those properties are:

1. The mutuality of ties.
2. The closeness or reachability of subgroup members.

3. The frequency of ties among members.
4. The relative frequency of ties among subgroup members compared to non-members [Ibidem: 251-252].

Methods of detection of cohesive groups in social networks vary accord to: a) type of network (one-mode vs. two-mode), b) type of relation (directed vs. undirected).

2 The idea of cliques

Clique is a basic concept to study cohesive subgroups in social networks [Ibidem: 254]. Its root can be traced back to the research initiated by W. Lloyd Warner and Elton Mayo during the 1930s and 1940s [Scott 2001: 16-26]. Generally, clique is a sub-structure of a network made by actors that are more intensely and closely tied to each other. In real life, cliques are often formed on a basis of gender, age, race, ethnicity, religion, knowledge etc.

In social network analysis, clique is understood as maximal complete subgraph of three or more nodes. All members of a clique are adjacent to each other. Dyads are not considered as a clique structure. Furthermore, cliques identified in a network may overlap. Node or set of nodes often belong to many cliques [Wasserman & Faust 1994: 254]. It is not possible that one clique can fully cover another clique. In such a case we have just one clique.

2.1 Cliques

The most common method to detect cliques in the network data is to find complete subgraphs. The algorithm searching for complete subgraphs in a larger structure is one of simplest method to trace cliques in social networks.

An `igraph` package provides a few useful functions for detecting clique. The main function is `cliques` which returns a list (as a vector of vertex id's) of all cliques found in the graph. We could set `min` argument to 3 to remove vertices and edges.

```
cliques <- cliques(graph, min = 3)
length(cliques)
```

```
## [1] 199
```

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

```
##
##  3  4  5  6  7
## 98 63 29  8  1
```

We could see that there are a lot of cliques even in rather small network. However we must remember that those cliques could overlap. In particular, a clique of size N includes $N-1$ cliques of size $N-1$, $\binom{N}{2}$ cliques of size $N-2$ and so on. Usually we are more interested in cliques, that are not part of some bigger ones. To find them we may use function `maximal.clique`.

```
maximal.cliques.count(graph, min = 3)
```

```
## [1] 23
```

```
max_cliques <- maximal.cliques(graph, min = 3)
table(sapply(max_cliques, length))
```

```
##
##  3  4  5  6  7
## 14  5  2  1  1
```

Now there are only 23 cliques. Some of them may overlap, but not completely. We could also directly find the largest cliques in the graph. Function `clique.number(graph)` gives us a size of the largest clique.

```
clique.number(graph)
```

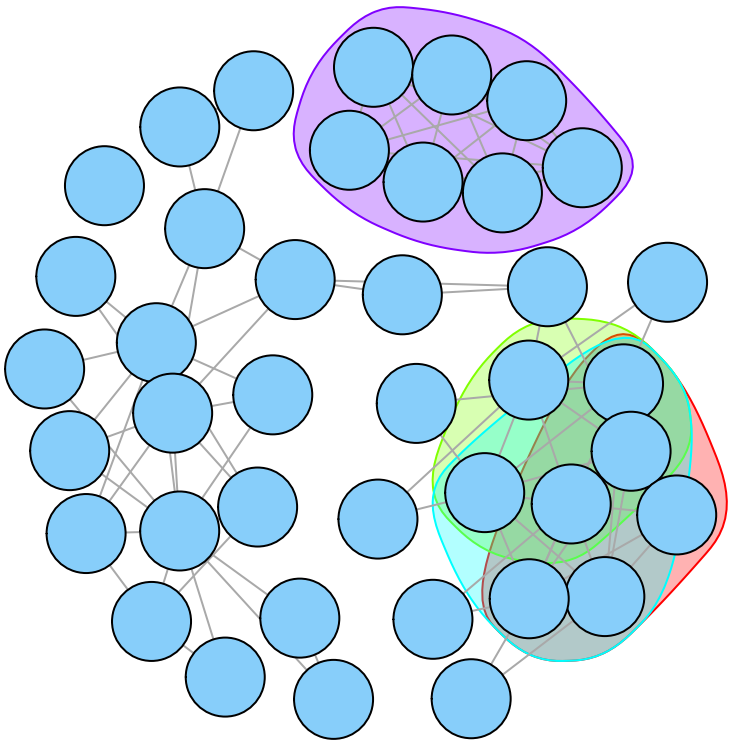
```
## [1] 7
```

```
largest.cliques(graph)
```

```
## [[1]]
## + 7/40 vertices, named:
## [1] 10 4  5  6  7  8  9
```

On the figure below maximal cliques of size at least 5 are highlighted.

```
groups <- maximal.cliques(graph, min = 5)
par(mar = rep(0,4))
plot(graph, mark.groups = groups, vertex.label = NA, margin = 0)
```



Two cliques of size 5 (yellow and red) are partially included in cliques of size 6 (blue), while the largest clique (violet) forms isolated group.

In package `network` there is a function `clique.census`, which finds all maximal cliques and additionally returns clique count and co-membership matrix for vertices. It also performs symmetrization step for undirected graphs.

2.2 N-cliques

As we mentioned above, the geodesic distances among members of cohesive groups should be small. To make the idea of clique more flexible it is possible to specify the cutoff value n as the maximum length of geodesic distance linking pairs of actors within the cohesive subgroup [Wasserman & Faust 1994: 254]. Thus, the n-clique is a subgraph with paths of n length as maximum value allowed between pairs of actors. Formally, n-clique is made of geodesics $d(i,j) =< n$ equal or lesser than n value. When $n = 1$ identified subgroup is a complete subgraph.

2.3 N-clans

The idea of n-cliques has its limitations that led to the idea of n-clans. Two problems might arise when we deal with n-cliques: 1. It may happen that the diameter of n-clique is greater than n value. 2. It may also happen that n-clique is disconnected as n-path connecting two nodes in analyzed n-clique goes through nodes outside of that n-clique. The solution to the first problem send us to the idea of n-clans. A n-clan is a n-clique in which the geodesic distance between all nodes is no greater than n [Ibidem: 260]. All n-clans in a subgraph are n-cliques, but not all n-cliques are n-clans.

2.4 K-plexes

Cohesive subgroups are often identified on adjacency of subgroup members. Such a subgroup is made of actors being adjacent to relatively numerous other members of the subgroup. In other words, all subgroup members must be adjacent to some minimum `k` number of other members belonging to that subgroup. If `k=N-1`, where `N` is the size of the subgraph, the subgraph is a clique. As `k` is getting smaller, each node is allowed to have more missing ties within the subgroup.

3 Communities

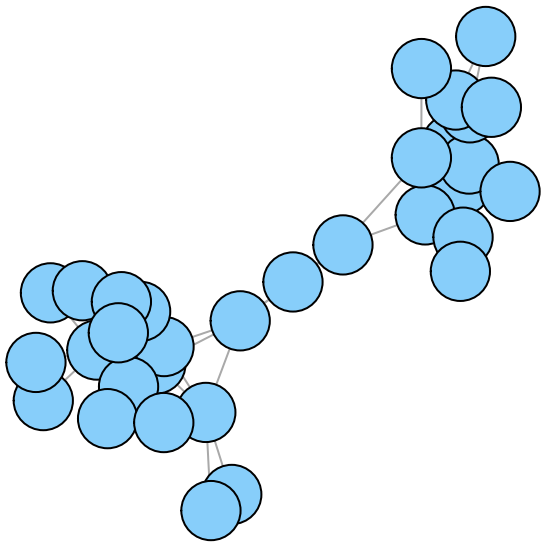
The idea of cliques is based on rigid formal assumptions (see above). It is quite complicated to detect cliques in real life social networks. Thus, in more recent works the idea of cliques has been replaced by more flexible but still very formal concept of 'community'. So, what is the community structure? Newman and Girvan spoke of "property that seems to be common to many networks is community structure, the division of network nodes into groups within which the network connections are dense, but between which they are sparser" (Newman and Girvan 2004). There are many algorithms availale to detect communities in social networks. They can be grouped into one of three types of algorithms: a) divisive algorithms, b) agglomerative algorithms, c) optimization methods. Some of them are briefly presented below.

It makes sense to study communities on connected components of a graph – most algorithms will not create community that connects nodes from two components. Therefore we will restrict following analyses to the largest connected component in our graph.

```
cl <- clusters(graph)
graph_c <- induced.subgraph(graph, cl$membership == which.max(cl$csize))
graph_c <- set.graph.attribute(graph_c, "layout",
                               layout.fruchterman.reingold(graph_c, params = list(repulserad = vcount(graph_c
```

```
## Warning in layout_with_fr(structure(list(32, FALSE, c(1, 2, 3, 8, 10, 13, :
## Argument `repulserad' is deprecated and has no effect
```

```
par(mar = rep(0, 4))
plot(graph_c, vertex.label = NA, margin = 0)
```



In `igraph` there are a few community–detection algorithms implemented. Every function returns an object of class `communities`, which offers some useful methods. Amongst others you could easily plot communities structure on a given graph, calculate modularity score or plot steps of algorithm as dendrogram.

3.1 Edge betweenness

One of the most popular algorithms detecting communities in a social networks has been given by Michelle Girvan and Mark Newman. The Girvan-Newman algorithm is a hierarchical method used in complex systems. Progressive removing of edges from the original network leads to the connected components of the remaining network which are the communities. This algorithm does not search for the most central edges of the whole structure. It focuses on the edge that is the most between and removes it. The most between edge is defined by the highest number of shortests paths running through the pair of nodes. The algorithm can be summarized by the following steps:

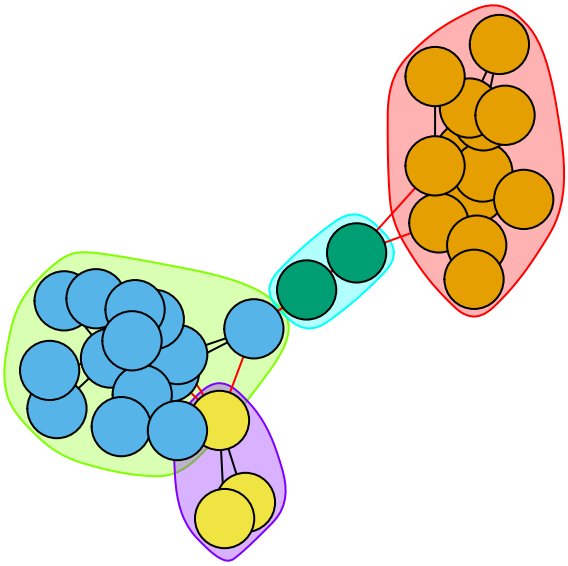
1. Betweenness of all edges is calculated as a first.
2. Edge with the highest betweenness level is removed.
3. Betweenness of all remaining edges is recalculated.
4. Steps 2 and 3 are repeated until no edges remain in a network.

The Girvan-Newman algorithm is based on the divisive approach in finding communities in network structures. In `igraph` package the function `edge.betweenness.community` is based on this algorithm.

```
edge_comm <- edge.betweenness.community(graph_c)
```

Algorithm is running until all edges are removed. However, we are interested in the best community structure obtained by the algorithm, which usually occurs in some intermediate step. Fortunately `igraph` finds the best division for us, so we could easily analyse properties of community structure or plot it. As for plotting, we need to provide our original graph as a second argument to appropiate `plot` method.

```
par(mar = rep(0, 4))
plot(edge_comm, graph_c, vertex.label = NA)
```



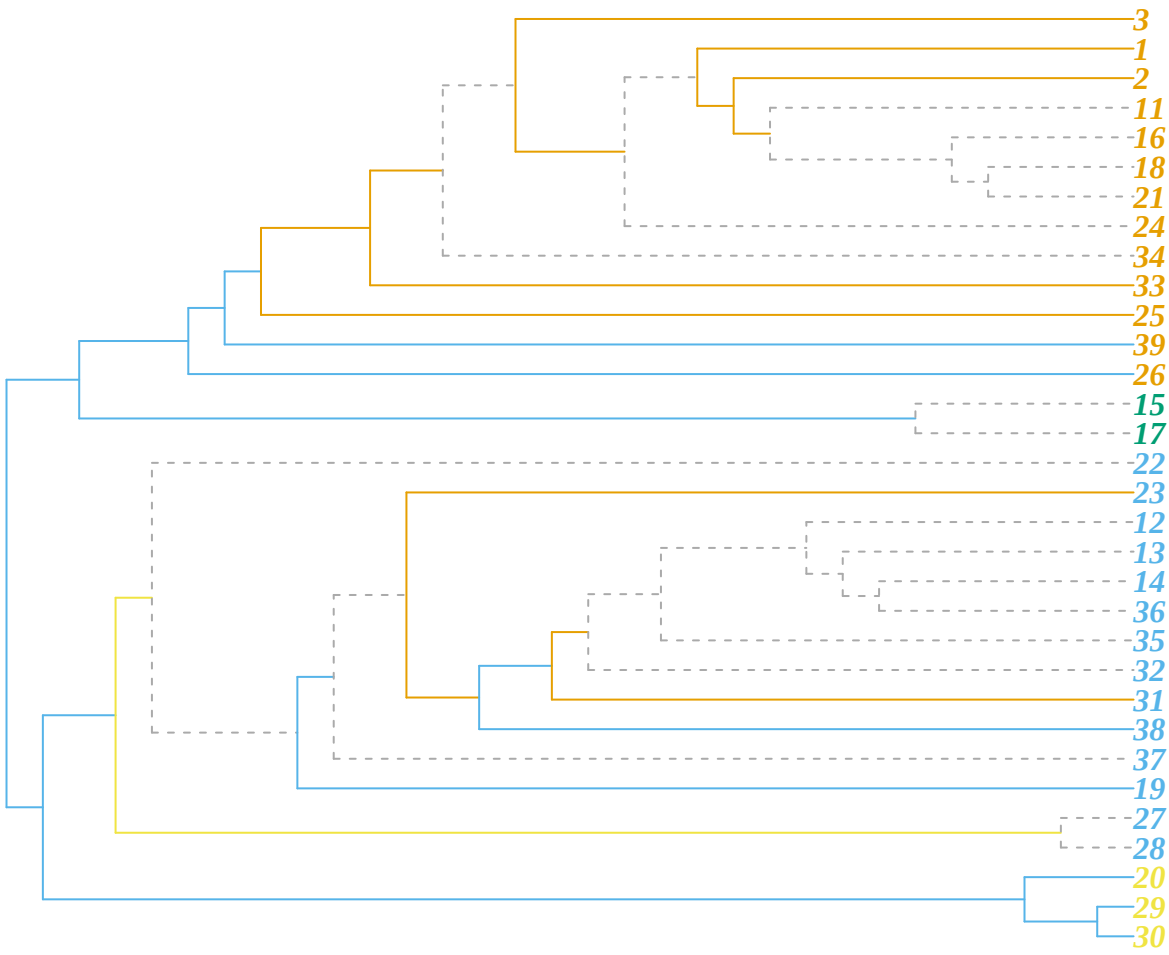
You could see how graph was divided at every step of the algorithm with `dendPlot` function. This function works for all communities that have a full hierarchical structure. We could check this with `is.hierarchical`. The coloured rectangles show the best community structure.

```
is.hierarchical(edge_comm)
```

```
## [1] TRUE
```

```
dendPlot(edge_comm)
```

HUMAN CAPITAL NATIONAL COHESION STRATEGY
UNIVERSITY OF WARSAW
ICS RUG / UU / RU
EUROPEAN UNION EUROPEAN SOCIAL FUND
Project co-financed by the European Union under the European Social Fund



3.2 Modularity

Modularitty can be refered to the quality of divisions that can be turned into communities within a network. Clauset, Newman and Moore explain that "modularity is a property of a network and a specific proposed division of that network into communities" (Clauset, Newman, and Moore 2004). It is defined as

$$Q = \frac{1}{2m} \sum_{ij} \left(A_{ij} - \frac{k_i k_j}{2m} \right) \delta(ci, cj),$$

where m is the number of edges, A is the adjacency matrix, k_i is the degree of node i, c_i is the type (or component) of node i and $\delta(x, y) = 1$ if $x = y$ and 0 otherwise. When modularity score is positive it means that there are more edges within communities than expected in random network with the same degree sequence.

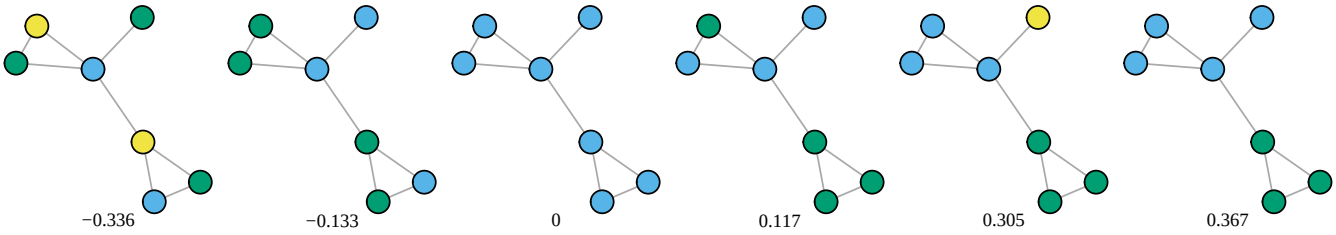
One problem with this measure is a lack of fixed range. Maximum (and minimum too) values depend on a structure of particular network, though they will always lie between -1 and 1. Values over 0.3 seem to be a good indicator of significant community structure.

Modularity could ba calculated in `igraph` with `modularity` function. It has methods for objects of class `igraph` or `communities`. In former case you need to provide also a vector of class membership, in the latter the maximum value of modularity obtained by community detection algorithm is returned.

On the following figure you could see how modularity score changes across different division of nodes.

```
g <- graph(c(1,2, 1,3, 2,3, 1,4, 1,5, 5,6, 6,7, 5,7), directed = FALSE)
g$layout <- layout.auto(g)
par(mfrow=c(1, 6), mar = rep(0, 4))
```

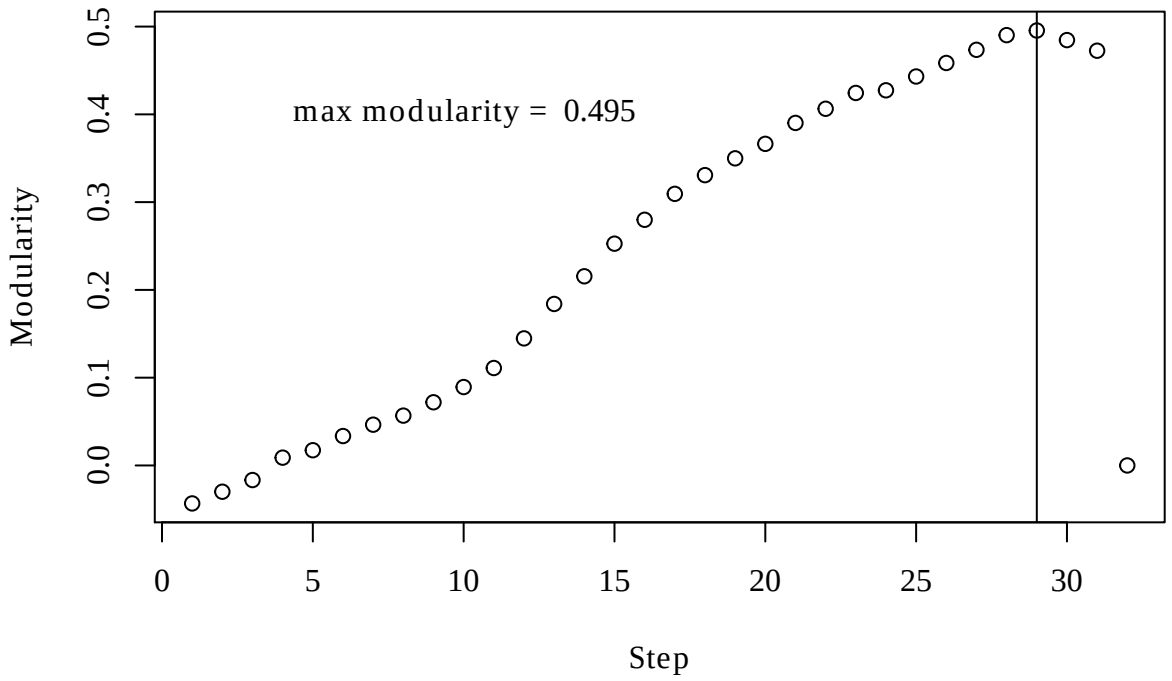
```
plot(g, vertex.color = c(1, 2, 3, 2, 3, 2, 1) + 1, vertex.label = NA)
text(0, -1.2, round(modularity(g, c(1, 2, 3, 2, 3, 2, 1)), 3))

plot(g, vertex.color = c(1, 2, 2, 1, 2, 1, 2) + 1, vertex.label = NA)
text(0, -1.2, round(modularity(g, c(1, 2, 2, 1, 2, 1, 2)), 3))

plot(g, vertex.color = c(1, 1, 1, 1, 1, 1, 1) + 1, vertex.label = NA)
text(0, -1.2, round(modularity(g, c(1, 1, 1, 1, 1, 1, 1)), 3))

plot(g, vertex.color = c(1, 1, 2, 1, 2, 2, 2) + 1, vertex.label = NA)
text(0, -1.2, round(modularity(g, c(1, 1, 2, 1, 2, 2, 2)), 3))

plot(g, vertex.color = c(1, 1, 1, 3, 2, 2, 2) + 1, vertex.label = NA)
text(0, -1.2, round(modularity(g, c(1, 1, 1, 3, 2, 2, 2)), 3))

plot(g, vertex.color = c(1, 1, 1, 1, 2, 2, 2) + 1, vertex.label = NA)
text(0, -1.2, round(modularity(g, c(1, 1, 1, 1, 2, 2, 2)), 3))
```



We could see that the modularity score is the lowest if there is no edge within any community. On the other hand the highest modularity is obtained for reasonable division, with only one edge between communities. However, when there is only one community, modularity is always 0.

We could see how modularity score was changing throughout the Girvan-Newman algorithm applied to our network of judges.

```
plot(edge_comm$modularity, xlab = "Step", ylab = "Modularity")
abline(v = which.max(edge_comm$modularity))
text(10, 0.4, paste("max modularity = ", round(modularity(edge_comm), 3)))
```



In general it is difficult to find the best (according to modularity score) division of nodes, but there are many approximative algorithms aimed at maximising modularity. Further in this chapter we will describe greedy optimization, random walks, statistical mechanics and Louvain method.

3.2.1 Greedy optimization

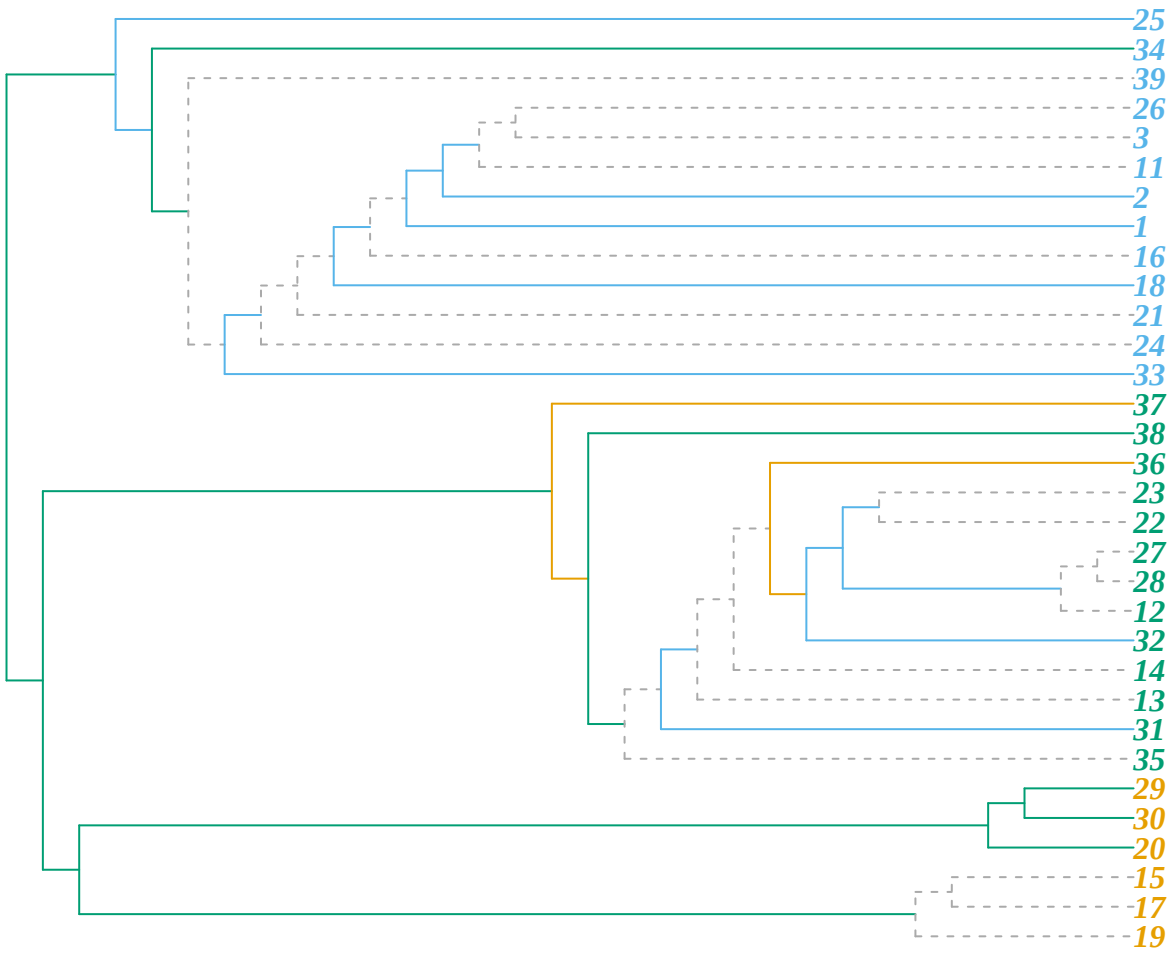
The greedy optimization algorithm, designed to detect communities in very large networks, has been proposed by (Clauset, Newman, and Moore 2004). The greedy algorithm selects the best solution at every step without regard to possible future. It starts with every node as a single community and then iteratively joins two communities whose connection provides the biggest increase in modularity. Procedure continues until all nodes are linked.

This algorithm has been implemented in `fastgreedy.community` function.

```
greed_comm <- fastgreedy.community(graph_c)
is.hierarchical(greed_comm)
```

```
## [1] TRUE
```

```
dendPlot(greed_comm)
```



3.2.2 Louvain method

The Louvain method is another algorithm based on the optimization of modularity. It's designed to find communities in large networks. In first phase, communities are assigned to each node in the network. Thus, there are as many communities as nodes in a graph. Then for each node we calculate a change in modularity for moving this node to the community of each of his neighbors. Finally node is moved to the community with the maximum gain of modularity. If gain is negative for all neighbors node is left where it was. The process is sequentially repeated until not further improvement can be achieved (when no individual move can improve modularity). In the second phase new network is built, where communities from the first phase become nodes. Weights of the links between new nodes are calculated as the sum of the weight of the links between nodes in the corresponding two communities. When second phase is finished, it is possible to apply first and then second phase again and again until maximum modularity is achieved (Blondel et al. 2008). This algorithm does not produce hierarchical clustering.

This algorithm is implemented under different name in `multilevel.community` function.

```
multim_comm <- multilevel.community(graph_c)
```

3.3 Walktrap community

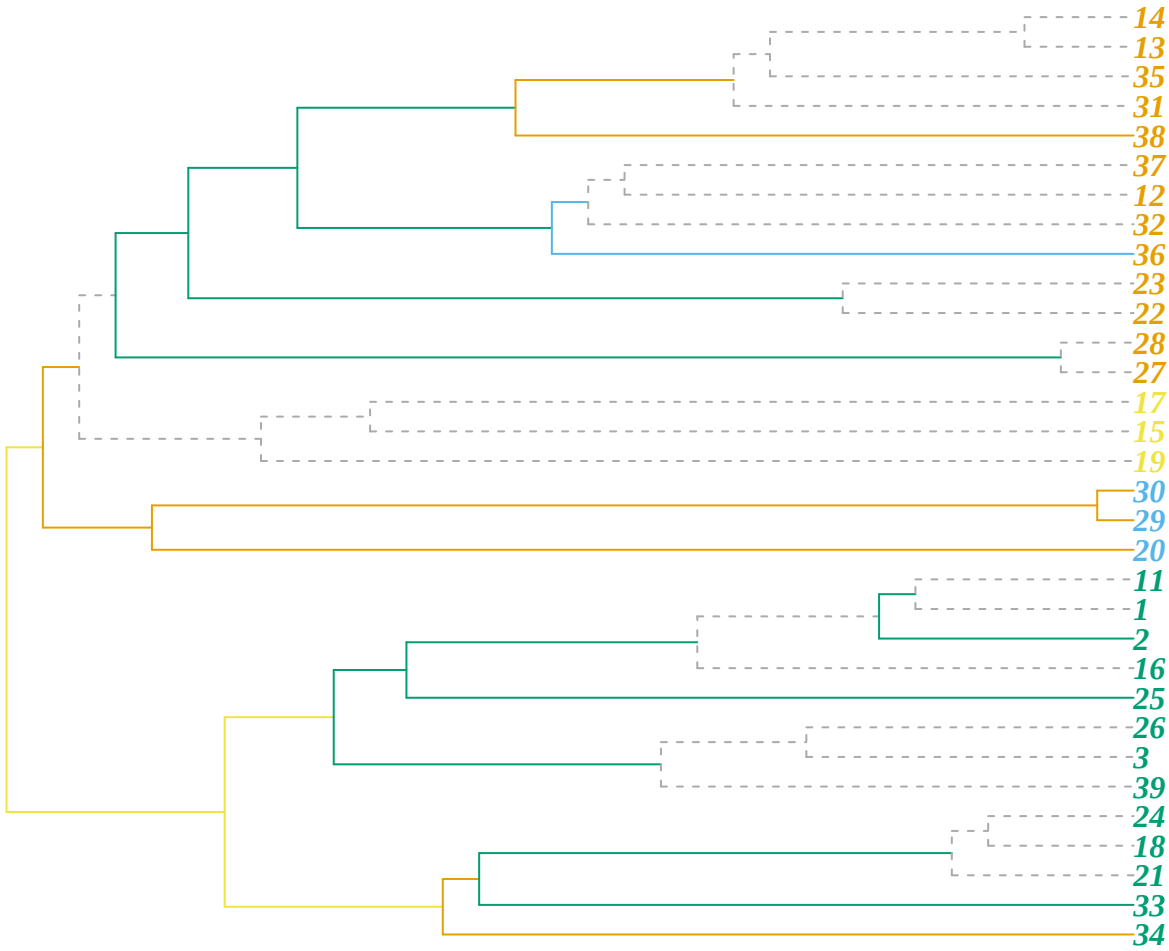
The idea behind this algorithm is that random walker on a graph tends to be trapped in a dense part of a network structure, which corresponds to some community, because there are few edges leading outside (Pons and Latapy 2005). It starts with every node as a single community, then iteratively calculates distances between communities by means of short random walks and connects the nearest communities.

The walktrap community could be calculated with `walktrap.community` function. The default length of random walks is 4 but you could change it with `steps` argument.

```
walktrap_comm <- walktrap.community(graph_c)
is.hierarchical(walktrap_comm)
```

```
## [1] TRUE
```

```
dendPlot(walktrap_comm)
```



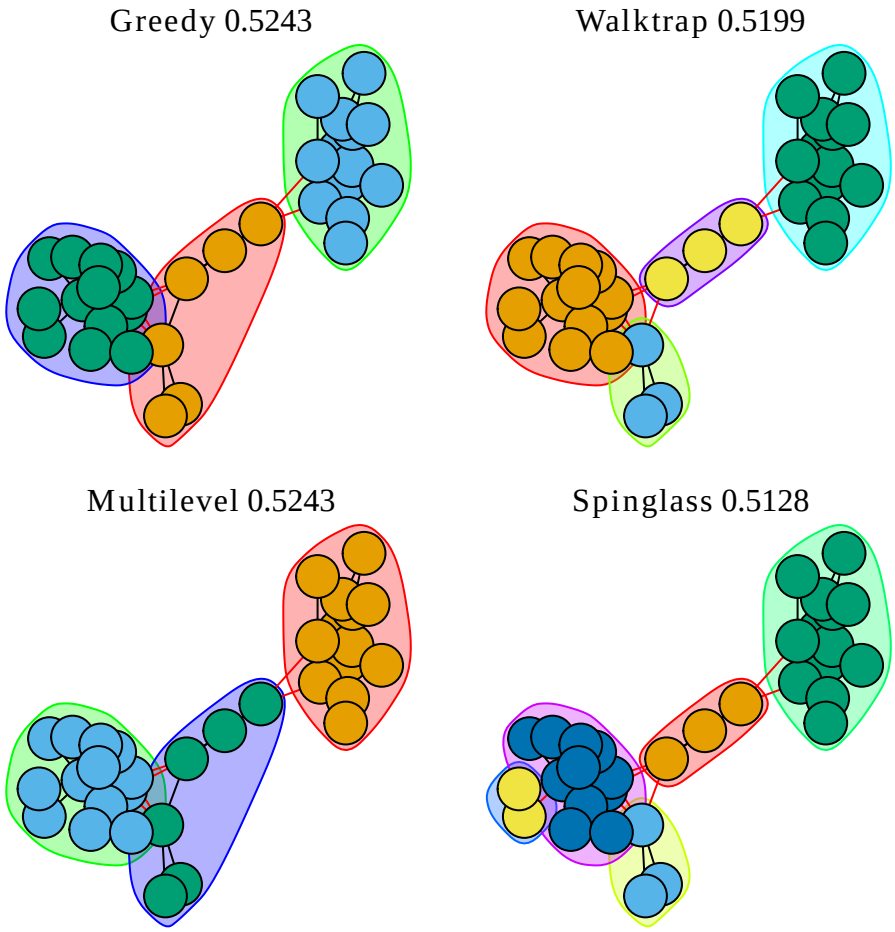
3.4 Spinglass community

Last algorithm we focus on can be applied to directed and weighted networks. In this model, each element of the network (node) can be in one of N spin states. Interactions between elements (the edges of the graph) define which pairs of nodes would stay in the same spin state and which have different spin states. The model is then simulated for a given number of steps. In the end the spin states of elements define the communities in a network. The number of communities is never greater than N. Some spin states might be empty, so the number of communities in a network is often lesser than N (Reichardt and Bornholdt 2006). `spinglass.community` is an implementation of this algorithm.

```
set.seed(12345)
spinglass_comm <- spinglass.community(graph_c)
```

3.5 Comparison of different algorithms.

```
par(mfrow = c(2,2), mar = c(1,0,1,0))
plot(greed_comm, graph_c, vertex.label = NA,
     main = paste("Greedy", round(modularity(greed_comm), dig = 4)))
plot(walktrap_comm, graph_c, vertex.label = NA,
     main = paste("Walktrap", round(modularity(walktrap_comm), dig = 4)))
plot(multim_comm, graph_c, vertex.label = NA,
     main = paste("Multilevel", round(modularity(multim_comm), dig = 4)))
plot(spinglass_comm, graph_c, vertex.label = NA,
     main = paste("Spinglass", round(modularity(spinglass_comm), dig = 4)))
```



We could see that all four algorithms return almost the same results. Spinglass and walktrap divided the smallest community (in the other two methods) into two triads and spinglass additionally separate two nodes from big community. Nevertheless, the highest modularity score is obtained for simple division with three communities.

4 Two-mode networks

Two-mode (two-dimensional) networks are specific types of networks made of two different sets of actors or set of actors and set of events (firms, states). They are often called affiliation networks (De Nooy, Mrvar, and Batagelj 2011). Consider directors sitting on corporate boards (interlocking directorate). Some of them are members of many boards. Thus, it's possible to reconstruct network of relations between comapnies and its directors. Directors are example of one type of actros, while companies belong to second type of actors. In terms of social network analysis,

in two-mode networks rows and columns are different sets of entities. In other words, two-mode networks appear when researchers collect data reflecting relations between different classes of actors.

`igraph` recognizes bipartite graphs and provides some special functionalities. Bipartite graphs are stored simply as normal (one–mode) graphs, but with `type` argument indicating which nodes belong to which group. Unfortunately `igraph` wants type to be logical, or is coercing them to logical, which could be sometimes annoying as you couldn't explicitly use names of classes as types.

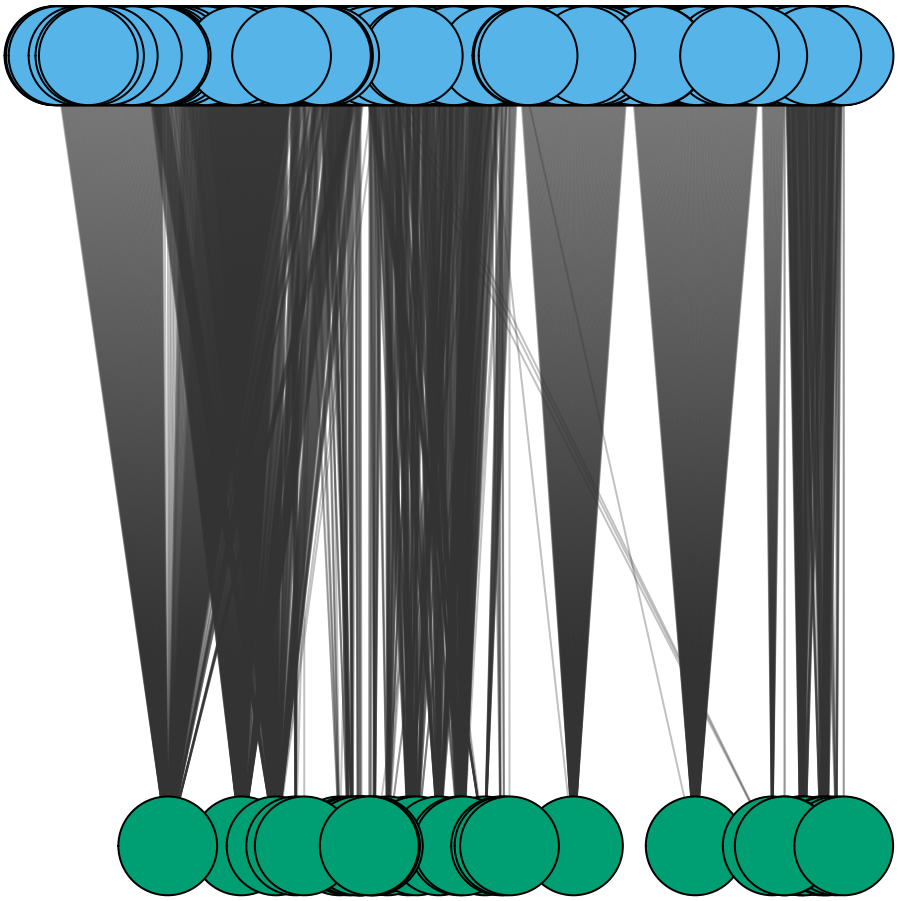
Below you could see a bipartite network, which was a source network for a graph used in previous chapters. There are two class of nodes – judges and judgments. Nodes are connected if a given judge was judging at a given case.

```
data(judge_net_bp)
graph_bip <- judge_net_bp
table(V(graph_bip)$type)
```

```
##
## FALSE  TRUE
##  1149    40
```

A "**B**" in "**UN-B**" indicates that we are dealing with bipartite graph. We could see that there are a lot more judgments (`FALSE`) than judges (`TRUE`). There is a special layout for plotting bipartite graphs, where classes are clearly separated.

```
par(mar = rep(0,4))
plot(graph_bip, layout = layout.bipartite,
     vertex.label = NA,
     vertex.color = V(graph_bip)$type + 2,
     edge.color = rgb(0.2,0.2,0.2,0.3),
     margin = 0)
```



Although our network is too big to be plotted in any sensible way, it could be seen that some judges ruled in a huge number of judgments. Quick look at degrees confirms that.

```
table(degree(graph_bip)[V(graph_bip)$type])
```

```
##
##   1   2   3   5   7  10  14  18  21  27  30  33  36  37  40  42  47  65
##   7   2   4   1   1   1   1   2   1   1   1   1   2   1   1   1   2   1
##  83  97 108 118 163 193 271 280 361
##   1   1   1   1   1   1   1   1   1
```

4.1 Cohesive subgroups in two-mode networks

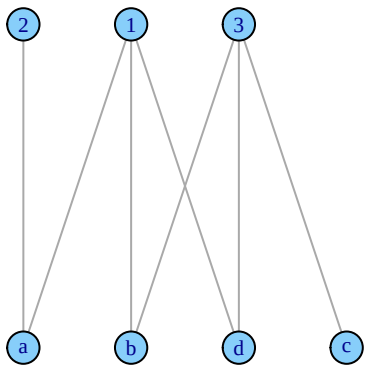
Detecting cohesive subgroups in two-mode networks is a bit more complicated than in ordinary networks. There are couple of methods to do that. The choice of method should mainly depend on properties of our data and on a target we want to achieve by finding communities. The easiest way is to convert two-mode data into one-mode. When data is converted methods designed for one-mode networks can be used to detect communities. Further in this chapter we will describe the projection procedure in detail.

4.2 Projection procedure

As we mentioned above, to conduct certain types of analysis researchers transform two-mode data into one-mode. One of few transformation techniques is called projection. In the projection procedure one type of nodes are removed from the two-mode network. Two nodes from the other class are connected if they were connected with the same node from the removed class.

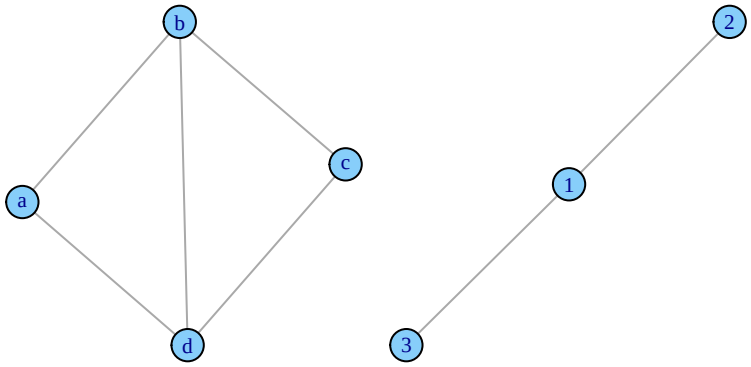
Let's say we have the network with two sets of nodes $A = \{a, b, c, d\}$ and $B = \{1, 2, 3\}$ and relations depicted below.

```
g <- graph.bipartite(c(TRUE, TRUE, TRUE, TRUE, FALSE, FALSE, FALSE),
                     c(1,5, 1,6, 2,5, 2,7, 3,7, 4,5, 4,7))
V(g)$name <- c("a", "b", "c", "d", "1", "2", "3")
par(mar = rep(0,4))
plot(g, layout = layout.bipartite, vertex.size = 20)
```



If we remove nodes B and apply the projection procedure, relations in A set of nodes will look like: a is connected to $\{b, d\}$, b is connected to $\{a, c, d\}$, c to $\{b, d\}$ and d to $\{a, b, c\}$. We could analogically remove nodes A and obtain second projection. Both are show below.

```
par(mar = rep(0,4), mfrow=c(1,2))
plot(graph.data.frame(data.frame(c("a","a","b","b","c"), c("b","d","c","d","d")), directed = FALSE),
     vertex.size = 20)
plot(graph(c(1,2, 1,3), directed = FALSE), vertex.size = 20)
```

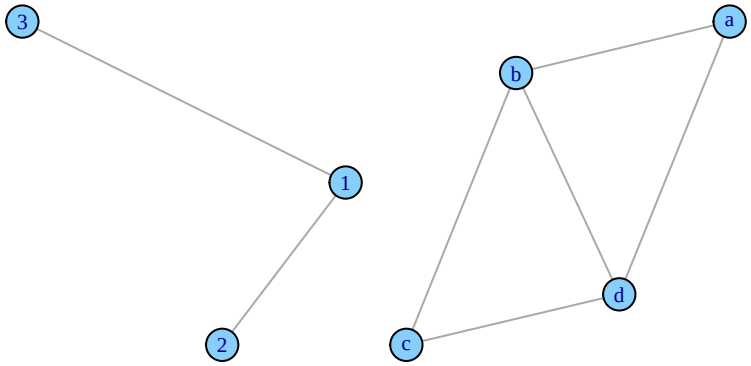


Projection procedure is fortunately implemented `igraph` in `bipartite.projection`. Firstly we could check how the result of projection will look without constructing it.

```
bipartite.projection.size(g)
```

```
## $vcount1
## [1] 3
##
## $ecount1
## [1] 2
##
## $vcount2
## [1] 4
##
## $ecount2
## [1] 5
```

We want projection to be binary graphs, so we set `multiplicity` argument to `FALSE` to simplify multiple edges.



We see that these networks are exactly the same as we've constructed by hand.

Now let's calculate projection of judges–judgments network. We are more interested in a network between judges, therefore we could restrict projection procedure to construct network only between `TRUE`s. We want binary graph, so we set `multiplicity` argument to `FALSE` to simplify multiple edges. The resulting network is exactly our co–judging network.

```
graph_proj <- bipartite.projection(graph_bip, multiplicity = FALSE, which = "true")
graph_proj
```

```
## IGRAPH UN-- 40 94 --
## + attr: name (v/c), JudgeSex (v/c), DivisionCode (v/c)
## + edges (vertex names):
##  [1] 1 --2  1 --3  1 --16 1 --21 1 --18 1 --11 1 --25 1 --39 2 --3  2 --11
## [11] 2 --16 2 --21 2 --18 2 --24 3 --11 3 --15 3 --16 3 --26 3 --39 4 --5
## [21] 4 --6  4 --9  4 --10 4 --8  4 --7  5 --6  5 --8  5 --7  5 --9  5 --10
## [31] 6 --8  6 --7  6 --10 6 --9  7 --8  7 --9  7 --10 8 --9  8 --10 9 --10
## [41] 11--16 11--18 11--21 11--25 11--26 11--24 12--13 12--14 12--22 12--23
## [51] 12--27 12--28 12--31 12--32 12--35 12--36 12--37 13--14 13--19 13--20
## [61] 13--35 13--31 13--36 13--37 13--38 13--32 14--19 14--31 14--32 14--20
## [71] 14--36 14--38 14--35 15--16 15--17 15--19 16--18 16--21 16--33 17--19
## + ... omitted several edges
```

```
graph
```

```
## IGRAPH UN-- 40 94 --
## + attr: layout (g/n), name (v/c), JudgeSex (v/c), DivisionCode
## | (v/x)
## + edges (vertex names):
##  [1] 1 --2  1 --3  1 --11 1 --16 1 --18 1 --21 1 --25 1 --39 2 --3  2 --11
## [11] 2 --16 2 --18 2 --21 2 --24 3 --11 3 --15 3 --16 3 --26 3 --39 4 --5
## [21] 4 --6  4 --7  4 --8  4 --9  4 --10 5 --6  5 --7  5 --8  5 --9  5 --10
## [31] 6 --7  6 --8  6 --9  6 --10 7 --8  7 --9  7 --10 8 --9  8 --10 9 --10
## [41] 11--16 11--18 11--21 11--24 11--25 11--26 12--13 12--14 12--22 12--23
## [51] 12--27 12--28 12--31 12--32 12--35 12--36 12--37 13--14 13--19 13--20
## [61] 13--31 13--32 13--35 13--36 13--37 13--38 14--19 14--20 14--31 14--32
## + ... omitted several edges
```

References

Blondel, Vincent D., Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. 2008. "Fast unfolding of communities in large networks." *Journal of Statistical Mechanics: Theory and Experiment* 2008 (10): 6. doi:10.1088/1742-5468/2008/10/P10008.

Clauset, Aaron, Mark E. J. Newman, and Cristopher Moore. 2004. "Finding Community Structure in Very Large Networks." *Phys. Rev. E* 70 (6). American Physical Society: 066111. doi:10.1103/PhysRevE.70.066111.

De Nooy, Wouter, Andrej Mrvar, and Vladimir Batagelj. 2011. *Exploratory Social Network Analysis with Pajek*. New York: Cambridge University Press.

Newman, Mark E. J., and M. Girvan. 2004. "Finding and Evaluating Community Structure in Networks." *Phys. Rev. E* 69 (2). American Physical Society: 026113. http://link.aps.org/doi/10.1103/PhysRevE.69.026113.

Pons, Pascal, and Matthieu Latapy. 2005. "Computing Communities in Large Networks Using Random Walks." In *Computer and Information Sciences - ISCIS 2005*, edited by pInar Yolum, Tunga Güngör, Fikret Gürgen, and Can Özturan, 3733:284–93. Lecture Notes in Computer Science. Springer Berlin Heidelberg. doi:10.1007/11569596_31.

Reichardt, Jorg, and Stefan Bornholdt. 2006. "Statistical Mechanics of Community Detection." *Physical Review E* 74 (1). APS: 016110.

Wasserman, Stanley, and Katherine Faust. 1994. *Social Network Analysis: Methods and Applications*. New York: Cambridge University Press.