Further Topics in Social Network Analysis

Ego-networks and local neighborhoods

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

ICS RUG / UU / RU

UNIWERSYTET WARSZAWSKI

EUROPEAN UNION EUROPEAN SOCIAL FUND

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

1 Introduction

The primary focus of social network analysis are ties (relations, connections, links) between different categories of objects (vertices, nodes, actors). The ties form networks with various structural and relational properties. Depending on the research problem, some of those properties are used to describe the peculiarity of networks identified through research. Interpersonal relations, online communication, international trade, business cooperation and competition etc. have certain properties being in the interest of network researchers.

There are multiple approaches to studying social networks. In this chapter we will focus on basic properties of ego-networks extensively used in many social studies. Ego-network data are a bit more easy to collect as compared to complete network data and can be used to make inferences for large populations (S. Borgatti and Everett 2005). Ego-networks have been used in studies of: social support, knowledge sharing, or disease spread. Examples of ego-network measures can now be encountered on social networking websites, e.g. LinkedIn.

Even if one has complete network data it may be still of interest to "disassemble" the complete network into a set of ego-networks, or neighborhoods (S. Borgatti and Everett 2005). Such an exercise might facilitate understanding the variation across actors embedded in their "local" environments.

Let us start with defining two basic concepts:

a) *Ego* is an individual node identified in the network structure. Depending on research goals, egos can be persons, groups, societies, organizations, firms, states etc. Ego-network is a "set of alters who have ties to ego, and measurements on the ties among these alters" (Wasserman and Faust 1994: 42).

b) *Neighborhood* of a node v of order o is a set of vertices which are not further away from v than o steps. In other words, a neighborhood of order 0 is the node v, or ego, alone. Neighborhood of order 1 is v and its immediate neighbors. Neighborhood of order 2 is the neighborhood of order 1 and all the immediate neighbors of nodes in the neighborhood of order 1, and so on.

The definition of neighborhood is a bit more complicated in case of directed graph. We could distinguish "in" and "out" and other kinds of neighborhoods. An "out" neighborhood would include all the actors to whom ties are directed from ego. Similarly an "in" neighborhood would include all the actors who sent ties directly to ego. We might also want to define a neighborhood of only those actors to whom ego had reciprocated ties. Choice of neighborhood type should depend on research question.

The following picture illustrates the idea of ego-networks and neighborhoods in undirected graph.

```
# Loading necessary packages
library(igraph)
```

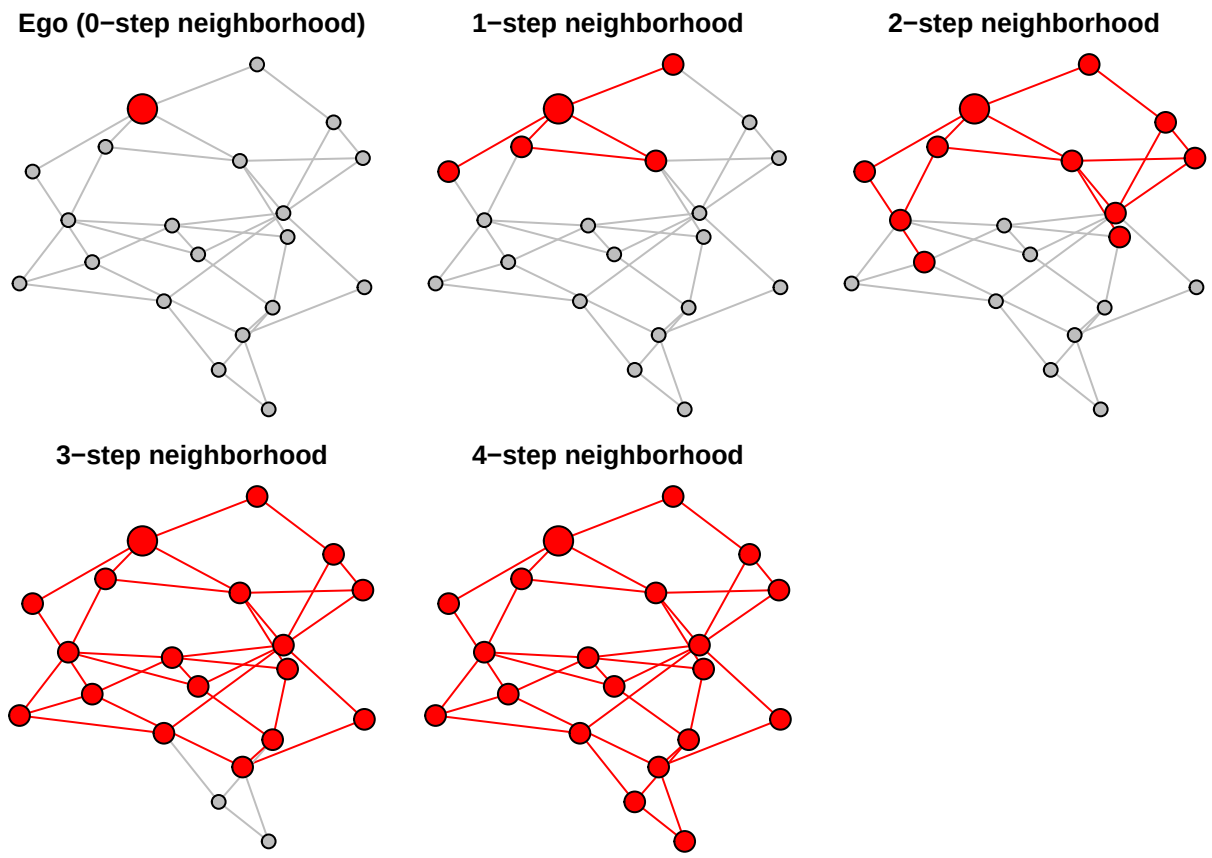
```
## Loading required package: methods
```

```
##
## Attaching package: 'igraph'
```

```
## The following objects are masked from 'package:stats':
##
##     decompose, spectrum
```

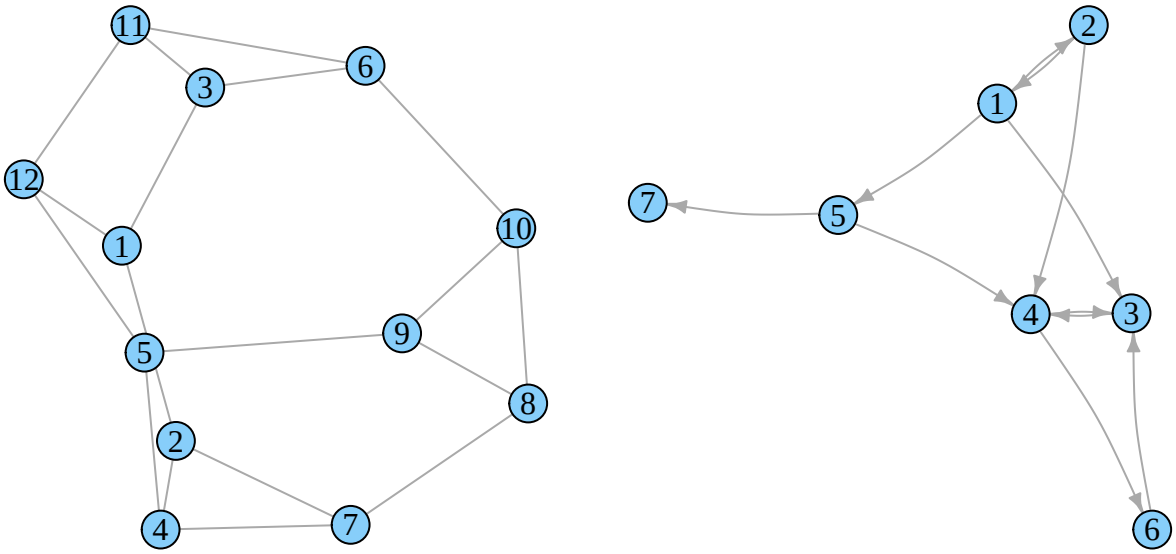
```
## The following object is masked from 'package:base':
##
##     union
```

```
library(isnar)
```



To find neighborhoods in `igraph`, both for directed and undirected graphs, you can use a variety of functions. We will demonstrate them using two networks – directed and undirected, both shown below.

```
g <- graph.famous("Frucht")
g_d <- graph(c(1,2, 1,3, 2,1, 2,4, 3,4, 4,3, 1,5, 5,4, 6,3, 4,6, 5,7))
```



The simplest function is `neighbors`, which returns neighbors of a given vertex (without that vertex) as a vertex sequence object. For a directed graph we could specify what type of neighborhood we want. Note that in last neighborhoods vertex 3 occurs twice, because the edge between 3 and 4 is reciprocated.

```
neighbors(g, 5)
```

```
## + 3/12 vertices:
## [1]  4  9 12
```

```
neighbors(g_d, 4, mode = "in")
```

```
## + 3/7 vertices:
## [1] 2 3 5
```

```
neighbors(g_d, 4, mode = "out")
```

```
## + 2/7 vertices:
## [1] 3 6
```

```
neighbors(g_d, 4, mode = "all")
```

```
## + 5/7 vertices:
## [1] 2 3 3 5 6
```

Function `neighborhood` is a bit more advanced, as it allows to chose the order of the neighborhood to be returned. It returns a list of neighborhoods (IDs) for all given vertices. Again, we could specify type of the neighborhood. Note that, differently than `neighbors`, `neighborhood` contains also the ego vertex.

```
neighborhood(g, order = 0, nodes = 5)
```

```
## [[1]]
## + 1/12 vertex:
## [1] 5
```

```
neighborhood(g, order = 1, nodes = 5)
```

```
## [[1]]
## + 4/12 vertices:
## [1]  5  4  9 12
```

```
neighborhood(g, order = 2, nodes = 5)
```

```
## [[1]]
## + 10/12 vertices:
##  [1]  5  4  9 12  2  7  8 10  1 11
```

```
neighborhood(g, order = 1, nodes = 1:5)
```

```
## [[1]]
## + 4/12 vertices:
## [1]  1  2  3 12
##
## [[2]]
## + 4/12 vertices:
```

```
## [1] 2 1 4 7
##
## [[3]]
## + 4/12 vertices:
## [1]  3  1  6 11
##
## [[4]]
## + 4/12 vertices:
## [1] 4 2 5 7
##
## [[5]]
## + 4/12 vertices:
## [1]  5  4  9 12
```

```
neighborhood(g_d, order = 1, nodes = 4, mode = "out")
```

```
## [[1]]
## + 3/7 vertices:
## [1] 4 3 6
```

```
neighborhood(g_d, order = 2, nodes = 4, mode = "in")
```

```
## [[1]]
## + 6/7 vertices:
## [1] 4 2 3 5 1 6
```

If you want to know only the size of the neighborhood instead of all nodes in it, you could use `neighborhood.size` function.

```
neighborhood.size(g, order = 2, nodes = 5)
```

```
## [1] 10
```

```
neighborhood.size(g_d, order = 2, nodes = 4, mode = "in")
```

```
## [1] 6
```

Another possibility is to generate neighborhoods through adjacency list. There is a function `get.adjlist` that returns list of neighbors for all vertices. Again, ego node is skipped.

```
get.adjlist(g_d, mode = "in")
```

```
## [[1]]
## + 1/7 vertex:
## [1] 2
##
## [[2]]
## + 1/7 vertex:
## [1] 1
##
## [[3]]
## + 3/7 vertices:
```

```
## [1] 1 4 6
##
## [[4]]
## + 3/7 vertices:
## [1] 2 3 5
##
## [[5]]
## + 1/7 vertex:
## [1] 1
##
## [[6]]
## + 1/7 vertex:
## [1] 4
##
## [[7]]
## + 1/7 vertex:
## [1] 5
```

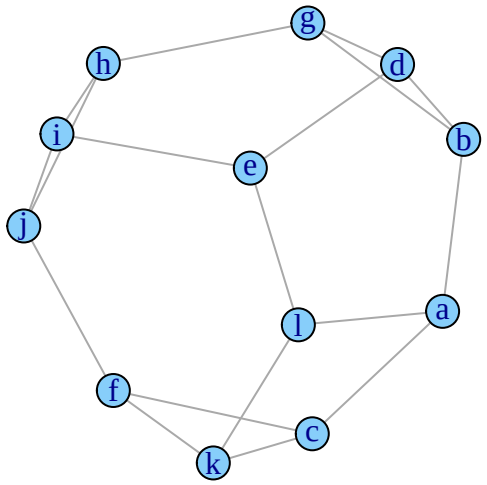
```
get.adjlist(g_d, mode = "out")
```

```
## [[1]]
## + 3/7 vertices:
## [1] 2 3 5
##
## [[2]]
## + 2/7 vertices:
## [1] 1 4
##
## [[3]]
## + 1/7 vertex:
## [1] 4
##
## [[4]]
## + 2/7 vertices:
## [1] 3 6
##
## [[5]]
## + 2/7 vertices:
## [1] 4 7
##
## [[6]]
## + 1/7 vertex:
## [1] 3
##
## [[7]]
## + 0/7 vertices:
```

You could also directly create ego-network based on the `graph.neighborhood` function. We set letters as vertex names so as it is clear which vertices are in which ego-network.

```
V(g)$name <- letters[1:vcount(g)]
plot(g, main="Whole network", vertex.color="lightskyblue")
```

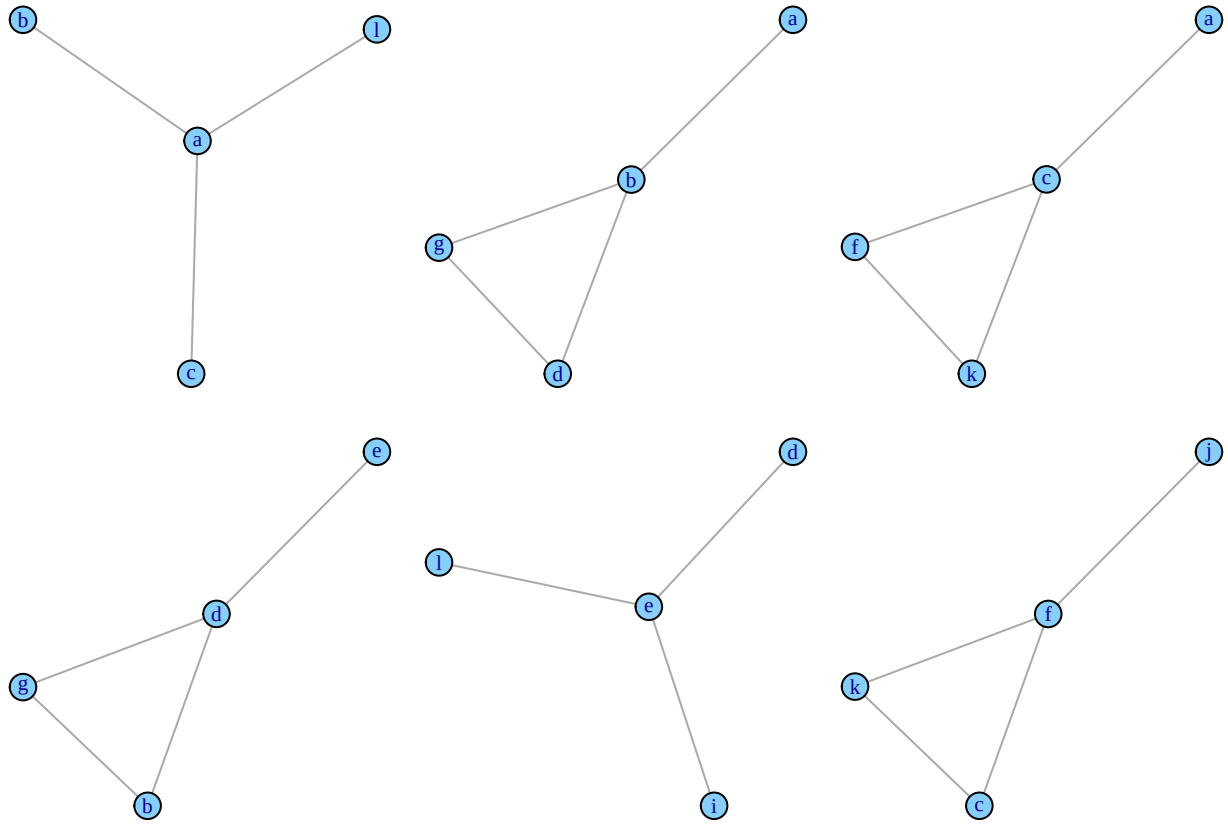
Whole network




```
# compute neighborhood graphs
ego_networks <- graph.neighborhood(g, order = 1, nodes = 1:6)
```

Plotting neighborhood graphs of vertices a, b, c, d, e, f:

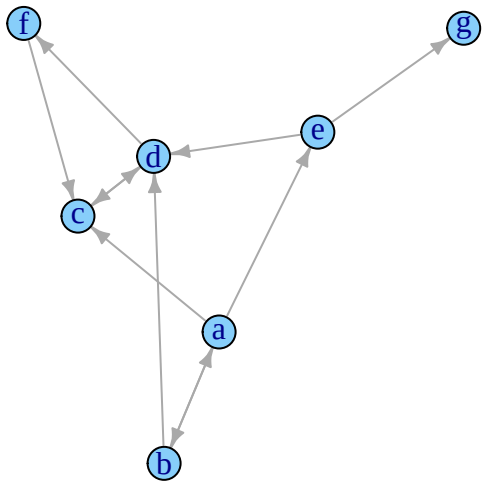
```
layout(matrix(1:6, 2, 3, byrow=TRUE))
op <- par(mar = rep(0.1, 4))
for(graph in ego_networks) {
  plot(graph, vertex.color="lightskyblue")
}
```



```
par(op)
```

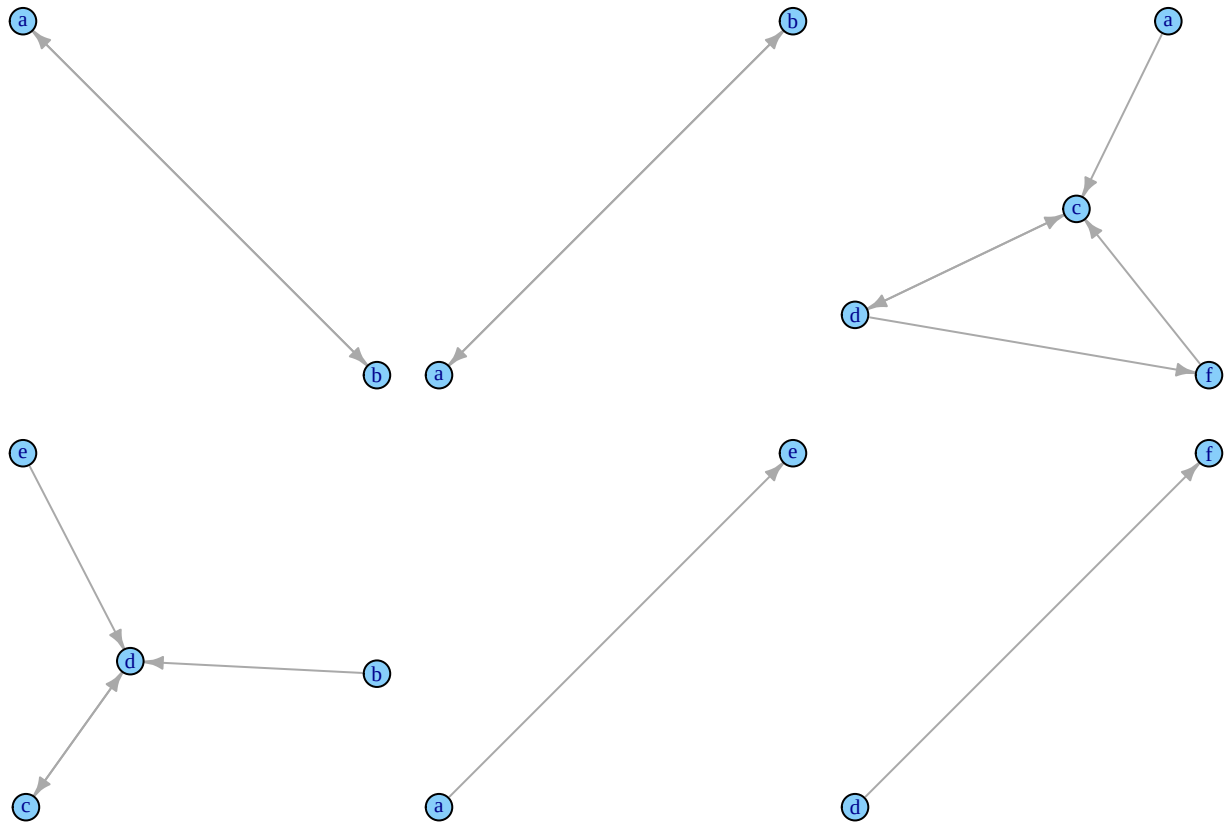
For a directed network we may choose to compute only neighbors on incoming ties using argument `mode="in"`:

```
V(g_d)$name <- letters[1:vcount(g_d)]
plot(g_d, vertex.color="lightskyblue", main="Whole directed network",
     edge.arrow.size=0.5)
```




```
# compute neighborhoods
ego_networks <- graph.neighborhood(g_d, order = 1, mode = "in", nodes=1:6)
```

```
layout(matrix(1:6, 2, 3, byrow=TRUE))
op <- par(mar=rep(0.1, 4))
for (graph in ego_networks) {
  plot(graph, vertex.color="lightskyblue",
       edge.arrow.size=0.5)
}
```



For objects of class `network` from `statnet` you could use `get.neighborhood` function.

2 Ego-network properties

There are multiple measures in social network analysis applied to identify the properties of ego-networks. Those measures can be grouped into: a) compositional measures, b) structural measures. For example, John have a five workmates he added to his profile at the LinkedIn. It means that his vertex degree is 5. Degree of a vertex represents the structural property. John workmates from LinkedIn are computer geeks and he often ask them for computer help. It means that John have a good access to computer knowledge. Knowledge resources are example of the compositional property. Further in this chapter we will focus on both compositional and structural properties of ego-networks.

2.1 Compositional properties

As we mentioned above, ego-network is a set of nodes who have ties to ego. Thus, ego-networks can be characterized through attributes of ego and its neighbors. Variability within and between ego and neighbors contributes to different network compositions. Below, we will consider some basic properties shaping network composition.

2.1.1 Ego characteristic

Depending on type of research, subjectivity of ego may vary. Individual, group, organization, corporation, state etc. may be the focal point of ego-network. If we assume that ego is a person it can be characterized by socio-demographic attributes e.g. sex, race, occupation, ethnicity. It is simple to recognize and deal with these attributes when one-mode networks are analyzed. When two-mode networks (see further in this course) are considered it is necessary to choose type of actor around which ego-network is build. Article by Paul Nieuwbeerta and Henk Falp shows how ego characteristics can be used (Nieuwbeerta and Flap 2000)

2.1.2 Resources

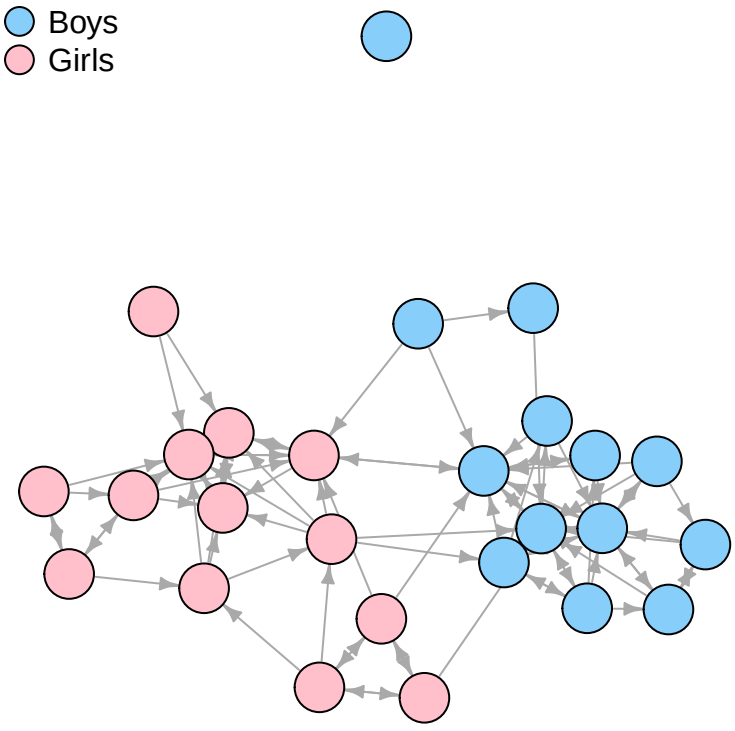
Resources owned by ego's neighbors have an impact on ego's success and opportunities. Thus, composition of the ego-network is shaped by resources that can be accessed and mobilized by individual in purposive actions. For example, individuals with greater amount of mobilizable resources in their ego-networks have better opportunities to find a new or better job. To collect data on resources embedded in ego-networks (social capital), the Resource Generator instrument is often used by network researchers (Van Der Gaag and Snijders 2005). To get some more details on the Resource Generator instrument see (Webber and Huxley 2007).

2.1.3 Homophily

Homophily means that alters in ego-network are similar to the ego according to some node attribute, like gender, education, age or income. It should be evaluated depending on the type of attribute. For instance when comparing income you could use mean squared difference, but for nominal attribute it would be more meaningful to calculate fraction of alters which has the same attribute level as ego.

Consider the classroom network. We want to asses to what extent children prefer to play with colleagues of the same sex. As we are interested in ego's preference towards others and not the other way round, we will analyze out-neighborhood.

```
data(IBE121)
playnet <- delete.edges(IBE121, E(IBE121)[question != "play"])
V(playnet)$color <- ifelse(V(playnet)$female, "pink", "lightskyblue")
plot(playnet, vertex.label = NA, edge.arrow.size=0.5)
legend(
  "topleft",
  pch=21,
  pt.cex=2,
  bty="n",
  pt.bg=c("lightskyblue", "pink"),
  legend=c("Boys", "Girls")
)
```



```
par(op)
```

We create explicit vector with sex, for clarity.

```
gender <- ifelse(V(playnet)$female, "female", "male")
```

Now actual computation. We want to inspect alters of every node, so an adjacency list will be useful.

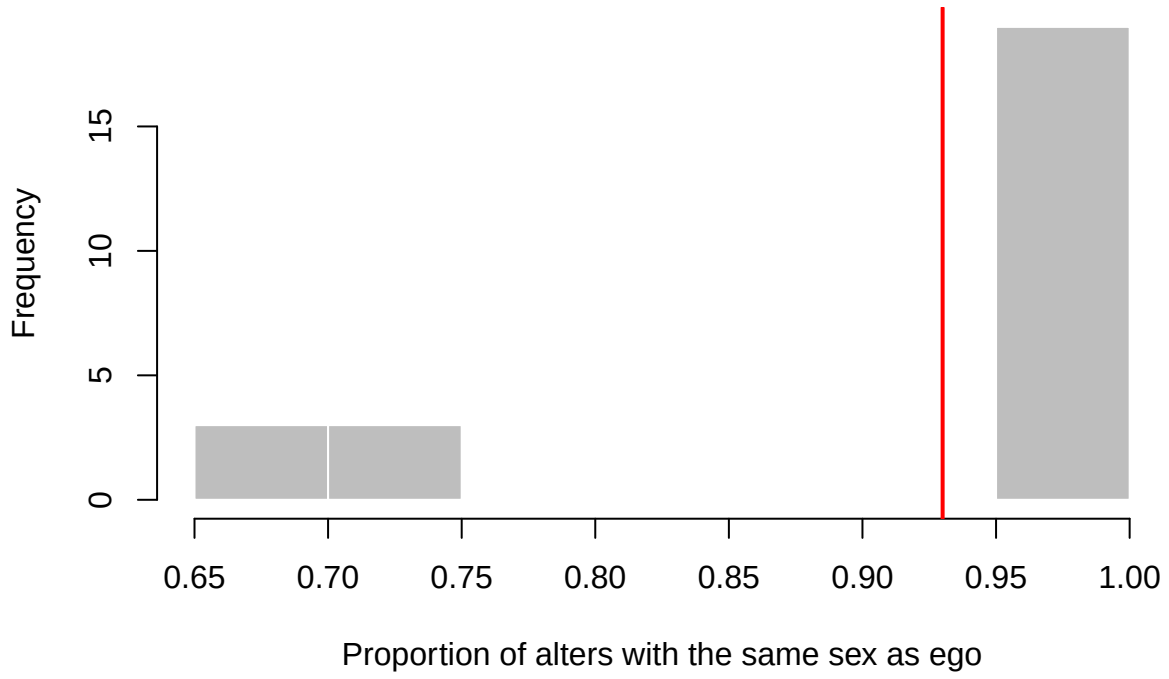
```
adj <- get.adjlist(playnet, mode = "out")
```

For each node, we calculate proportions of sex amongst its alters (we use factor to preserve genders with zero-share).

```
props <- lapply(adj, function(alters) {
  s <- factor(gender[alters], levels = c("male", "female"))
  prop.table(table(s))
})
```

Finally we compare our proportions to ego's sex.

```
frac <- sapply(seq_along(props), function(i) props[[i]][gender[i]])
hist(frac, xlab = "Proportion of alters with the same sex as ego", main = "",
     col="grey", border="white")
abline(v = mean(frac, na.rm = TRUE), col="red", lwd=2)
```



The vertical line indicates the average in whole network.

Our simple analysis suggests that children indeed prefer to play with same-sex colleagues. Almost all choose playmates ONLY from their own gender.

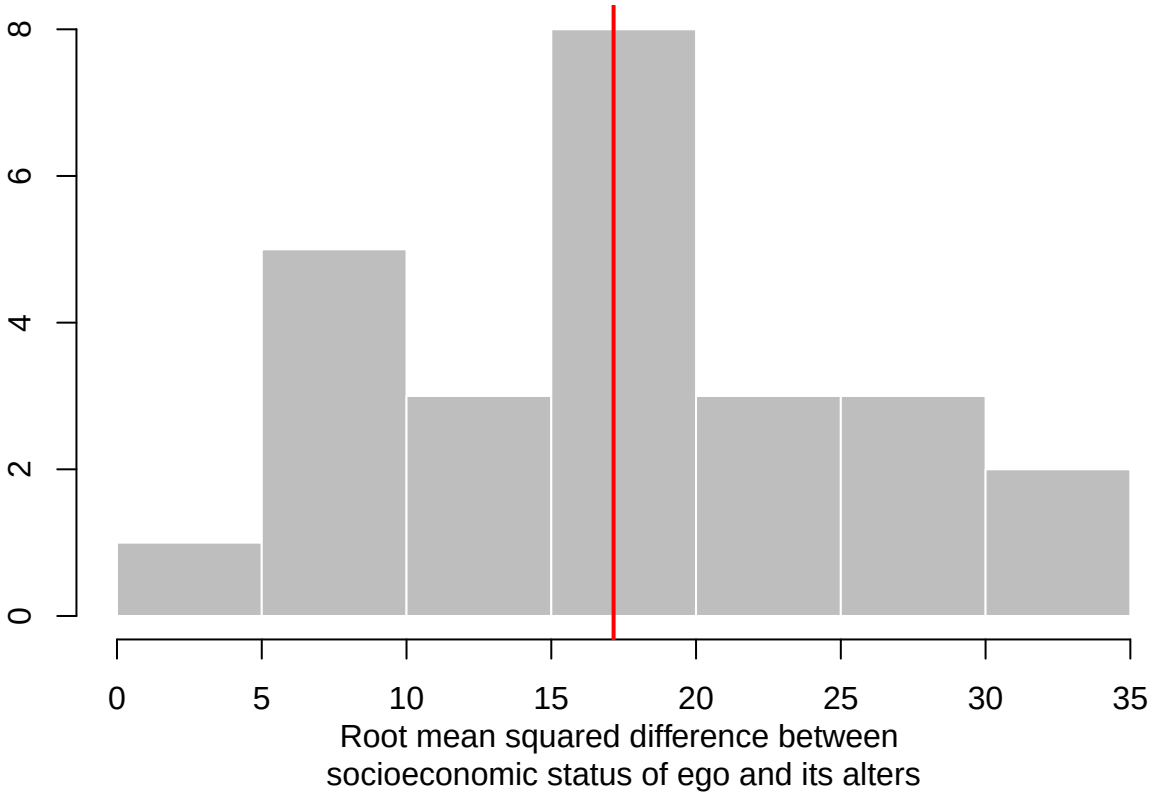
Another question we may ask is whether socioeconomic status of parents influence choice of playmates. We have information about status of every parent, so to obtain one value we'll calculate mean.

```
status <- matrix(c(V(playnet)$isei08_m, V(playnet)$isei08_f), ncol = 2)
status <- rowMeans(status, na.rm = TRUE)
```

Algorithm is almost the same as above, but this time we don't compare fractions but calculate root-mean-square difference.

```
alters_status <- lapply(adj, function(alters) s <- status[alters])
rmsd <- sapply(seq_along(alters_status), function(i){
  diff <- abs(alters_status[[i]] - status[i])
  sqrt(mean(diff^2, na.rm = TRUE))
})
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##   4.998  12.190  17.300  17.150  22.400  33.360       1
```



We see that it varies more than in case of gender. It could suggest that children don't choose their playmates according to their parents socioeconomic status, at least not in this class and not compared to sex criterion.

2.2 Structural properties

Variability of connections between ego and alters produce different structures of ego-networks. Those structures have certain properties (structural properties) that are measured in social network analysis. In this section, we will focus on some basic structural properties of ego-networks:

a) degree,
b) effective size,
c) efficiency,
d) constraint,
e) dyadic constraint.

2.2.1 Degree

In symmetric networks (undirected graphs) degree of a vertex is a number of vertices adjacent to that vertex. In non-symmetric networks (directed graphs), degree of a vertex is divided into in-degree and out-degree. In-degree of a vertex is a number of received ties, while out-degree is a number of ties sent by a vertex. Degree (in-degree, out-degree) may inform us about popularity, power or influence of the ego.

To calculate degree use function called `degree`. You could also use `neighborhood.size` to directly calculate size of ego-network.

```
degree(playnet, mode = "out")
```

```
## 1003 1006 1009 1012 1015 1018 1021 1024 1027 1030 1033 1036 1039 1042 1045
##    3    4    3    3    3    5    5    0    6    2    1    3    2    4    2
## 1048 1051 1054 1057 1060 1063 1066 1069 1072 1075 1078
##    5    5    5    2    4    4    4    3    3    4    3
```

```
degree(playnet, mode = "in")
```

```
## 1003 1006 1009 1012 1015 1018 1021 1024 1027 1030 1033 1036 1039 1042 1045
##    2    7    7    2    1    9    2    0    2    0    1    0    2    9    2
## 1048 1051 1054 1057 1060 1063 1066 1069 1072 1075 1078
##    4   10    2    6    1    2    2    2    3    2    8
```

```
degree(playnet, mode = "all")
```

```
## 1003 1006 1009 1012 1015 1018 1021 1024 1027 1030 1033 1036 1039 1042 1045
##    5   11   10    5    4   14    7    0    8    2    2    3    4   13    4
## 1048 1051 1054 1057 1060 1063 1066 1069 1072 1075 1078
##    9   15    7    8    5    6    6    5    6    6   11
```

```
neighborhood.size(playnet, 1)
```

```
##  [1]  5  9  8  4  4 10  7  1  9  3  3  4  4 10  5  7 12  6  7  5  5  5  4
## [24]  5  7  9
```

For networks of class `network` there is a function also called `degree` in package `sna`.

2.2.2 Effective size of the network

The effective size is the number of nodes ego has, minus the average number of ties that each node has, excluding tie to ego. Imagine that an ego is linked to three other nodes. At the same time, all of the nodes are connected to each other. Ego's network size is 3. However, the ties are redundant. Stephen Borgatti simply pointed that "The general meaning of redundancy is clear: a person's ego network has redundancy to the extent that her contacts are connected to each other as well" (S. P. Borgatti 1997: 35). The average degree of ego's alters is 2. So, the effective size of the network is 1 (3 (ego's network size) - 2 (average degree of ego's alters)). See more in: (Burt 1992; S. P. Borgatti 1997).

There is no function in `igraph` to calculate effective size of the network, but it is easy to write it by ourselves.

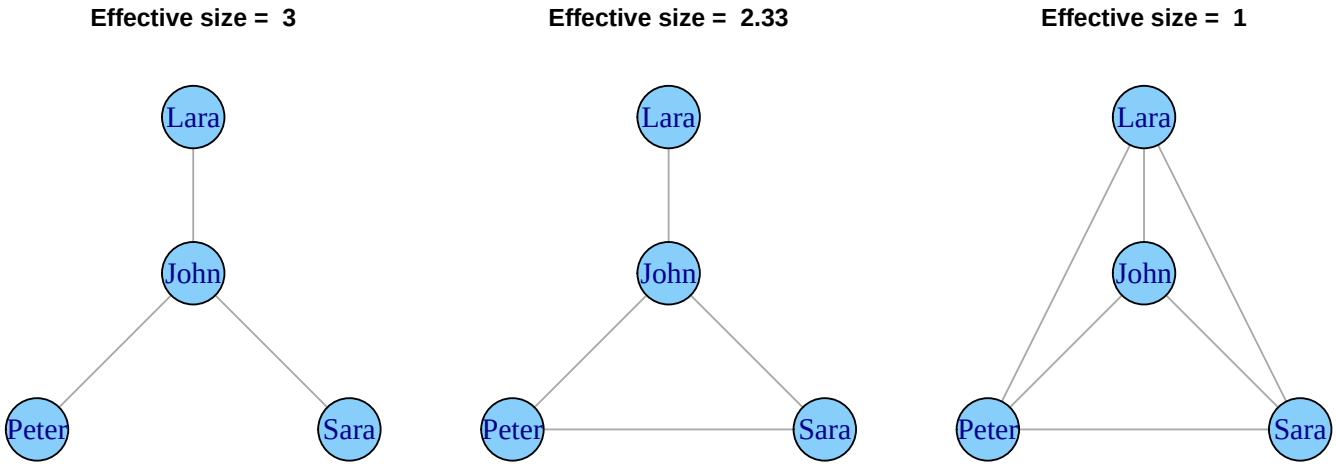
```
effective_size <- function(g, v) {
  degree(g, v) - mean(degree(delete.vertices(g, v)))
}
```

We will calculate it on small artificial network.

```
g <- graph.edgelist(matrix(c(1,2, 1,3, 1,4, 3,4), ncol = 2, byrow = TRUE),
                    directed = FALSE)
V(g)$name <- c("John", "Lara", "Peter", "Sara")
effective_size(g, "John")
```

```
##     John
## 2.333333
```

To feel how effective size works consider three versions of John's ego-network.



Effective size of the left network is 3, because alters couldn't communicate with each other at all. On the other hand, effective size of the right network is 1, although John still has three alters. However, this time alters could communicate with each other, therefore links from John are redundant - if we remove two of them, information from John could still diffuse on the whole network.

2.2.3 Efficiency

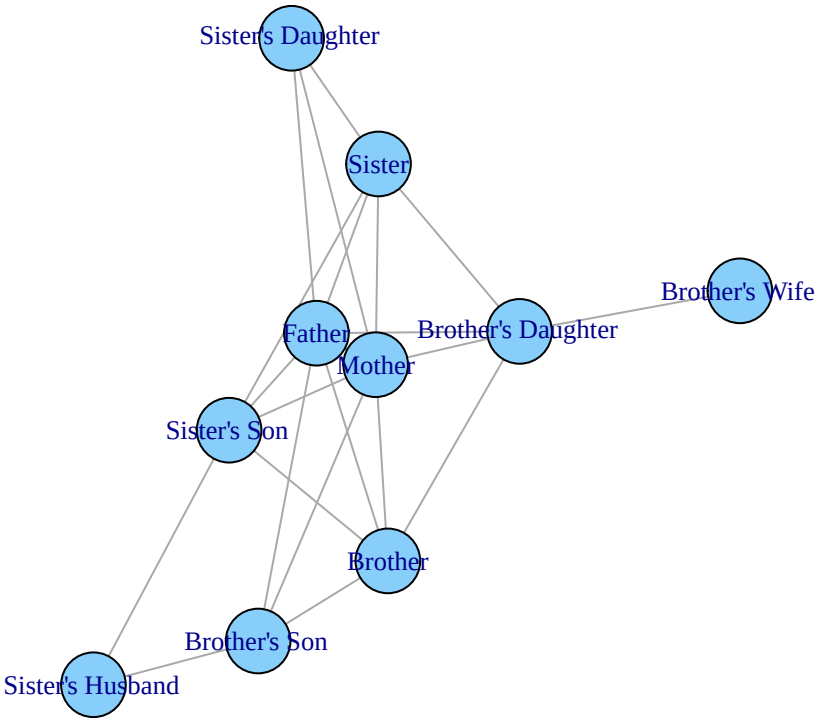
Efficiency weighs effective size of A's network with its actual size. Basic question on efficiency is what is the proportion of non-redundant ties between ego and his alters? As Hanneman and Riddle clearly stated "The effective size of ego's network may tell us something about ego's total impact; efficiency tells us how much impact ego is getting for each unit invested in using ties. An actor can be effective without being efficient; and actor can be efficient without being effective" (Hanneman and Riddle 2005: 138)

Again there is no explicit function to calculate efficiency, but we could write our own function.

```
efficiency <- function(g, v) {
  effective_size(g, v) / (vcount(g) - 1)
}
efficiency(g, "John")
```

```
##      John
## 0.7777778
```

See how efficiency and effective size are distributed in kinship network.



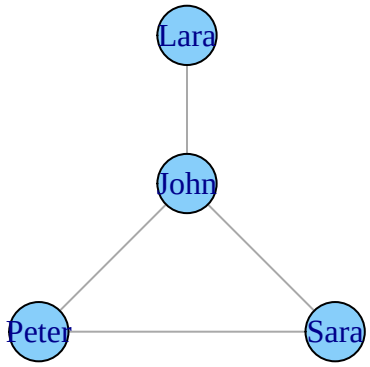
Size of ego network	Effective size Eff	iciency	
Mother	7	3.57	0.51
Sister	5	2.20	0.44
Brother's Wife	1	1.00	1.00
Sister's Daughter	3	1.00	0.33
Brother's Daughter	5	3.00	0.60
Father	7	3.57	0.51
Brother	5	2.20	0.44
Sister's Husband	2	2.00	1.00
Brother's Son	4	2.50	0.62
Sister's Son	5	3.00	0.60

Mother and father have the highest effective size of their ego-networks, but these networks are big, therefore their efficiency isn't the highest. On the other hand sister's husband has the highest possible efficiency, but she communicates only with two persons, so she is not effective.

2.2.4 Constraint (aggregate constraint)

Constraint is a measure that informs about the extent to which ego's connections are to nodes who are connected to each other. Suppose that A has connections to B and C, while B and C are connected to each other. In this case A is constrained. But if A's alters have no connections besides links to A, A is not constrained. The idea of constrain sends us to the important paradox. It happens that people who have many connections may lose autonomy of action.

In our simple example John is constrained by a link between Sara and Peter.



In `igraph` there is a function `constraint` that helps us to calculate aggregated constraint of the actors. The higher number, the more constrained the actor is in his action.

```
constraint(g)
```

```
##      John      Lara     Peter      Sara
## 0.6111111 1.0000000 1.0069444 1.0069444
```

In the subject literature there are many works where structural measures of ego-network were applied. To get some more details on presented measures it is worth to see (Roberts et al. 2009).

2.2.5 Dyadic constraint

Dyadic constraint highlights the extent of constraint between ego and each of his alters. In other words, the dyadic constraint between actors A and B shows the extent to which A has both more and stronger relations with nodes that are well connected to the actor B. Wider description of the constraint idea can be found in Burt's monographs (Burt 1992). Nice explanation could be found in (De Nooy, Mrvar, and Batagelj 2011).

There is no function for calculating dyadic constraint in `igraph`, so we have to implement our own.

```
dyadic_constraint <- function(g) {
  # proportional strength of a ties
  strength <- 1 / degree(g)

  A <- get.adjacency(g, sparse = FALSE)
  A2 <- A * strength %*% t(rep(1, vcount(g)))

  result <- A2 %*% A2 + A2
  result <- result^2
  # multiply by A to zero-out non-existent links
  result * A
}
```

Below we could see a matrix of dyadic constraints between actors in the kinship network.

```
##                        Mother     Sister Brother's Wife Sister's Daughter
## Mother             0.00000000 0.07183859           0.00        0.03680133
## Sister             0.14080363 0.00000000           0.00        0.06612245
## Brother's Wife     0.00000000 0.00000000           0.00        0.00000000
## Sister's Daughter  0.20036281 0.18367347           0.00        0.00000000
## Brother's Daughter 0.09521633 0.06612245           0.04        0.00000000
## Father             0.11592404 0.07183859           0.00        0.03680133
```

```
## Brother            0.12857347 0.00000000          0.00       0.00000000
## Sister's Husband   0.00000000 0.00000000          0.00       0.00000000
## Brother's Son      0.11270408 0.00000000          0.00       0.00000000
## Sister's Son       0.09521633 0.06612245          0.00       0.00000000
##                    Brother's Daughter     Father    Brother
## Mother                        0.04857976 0.11592404 0.06559871
## Sister                        0.06612245 0.14080363 0.00000000
## Brother's Wife                1.00000000 0.00000000 0.00000000
## Sister's Daughter             0.00000000 0.20036281 0.00000000
## Brother's Daughter            0.00000000 0.09521633 0.06612245
## Father                        0.04857976 0.00000000 0.06559871
## Brother                       0.06612245 0.12857347 0.00000000
## Sister's Husband              0.00000000 0.00000000 0.00000000
## Brother's Son                 0.00000000 0.11270408 0.10331633
## Sister's Son                  0.00000000 0.09521633 0.06612245
##                    Sister's Husband Brother's Son Sister's Son
## Mother                       0.0000    0.03680133   0.04857976
## Sister                       0.0000    0.00000000   0.06612245
## Brother's Wife               0.0000    0.00000000   0.00000000
## Sister's Daughter            0.0000    0.00000000   0.00000000
## Brother's Daughter           0.0000    0.00000000   0.00000000
## Father                       0.0000    0.03680133   0.04857976
## Brother                      0.0000    0.06612245   0.06612245
## Sister's Husband             0.0000    0.25000000   0.25000000
## Brother's Son                0.0625    0.00000000   0.00000000
## Sister's Son                 0.0400    0.00000000   0.00000000
```

We could see that sister is strongly constrained by father, because father is connected to all sister's alters and to other nodes as well.

References

Borgatti, Stephen P. 1997. "Structural Holes." *Analytictech. Com* 20 (1): 35–38.

Borgatti, Steven, and Martin Everett. 2005. "Ego-Network Betweeness." *Social Networks* 27 (1): 31–38.

Burt, Ronald S. 1992. *Structural Holes: The Social Structure of Competition*. Harvard: Harvard University Press.

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

Hanneman, Robert A, and Mark Riddle. 2005. "Introduction to Social Network Methods." University of California Riverside.

Nieuwbeerta, Paul, and Henk Flap. 2000. "Crosscutting Social Circles and Political Choice: Effects of Personal Network Composition on Voting Behavior in the Netherlands." *Social Networks* 22 (4). Elsevier: 313–35.

Roberts, Sam GB, Robin IM Dunbar, Thomas V Pollet, and Toon Kuppens. 2009. "Exploring Variation in Active Network Size: Constraints and Ego Characteristics." *Social Networks* 31 (2). Elsevier: 138–46.

Van Der Gaag, Martin, and Tom AB Snijders. 2005. "The Resource Generator: Social Capital Quantification with Concrete Items." *Social Networks* 27 (1). Elsevier: 1–29.

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

Webber, Martin P, and Peter J Huxley. 2007. "Measuring Access to Social Capital: The Validity and Reliability of the Resource Generator-UK and Its Association with Common Mental Disorder." *Social Science & Medicine* 65 (3). Elsevier: 481–92.