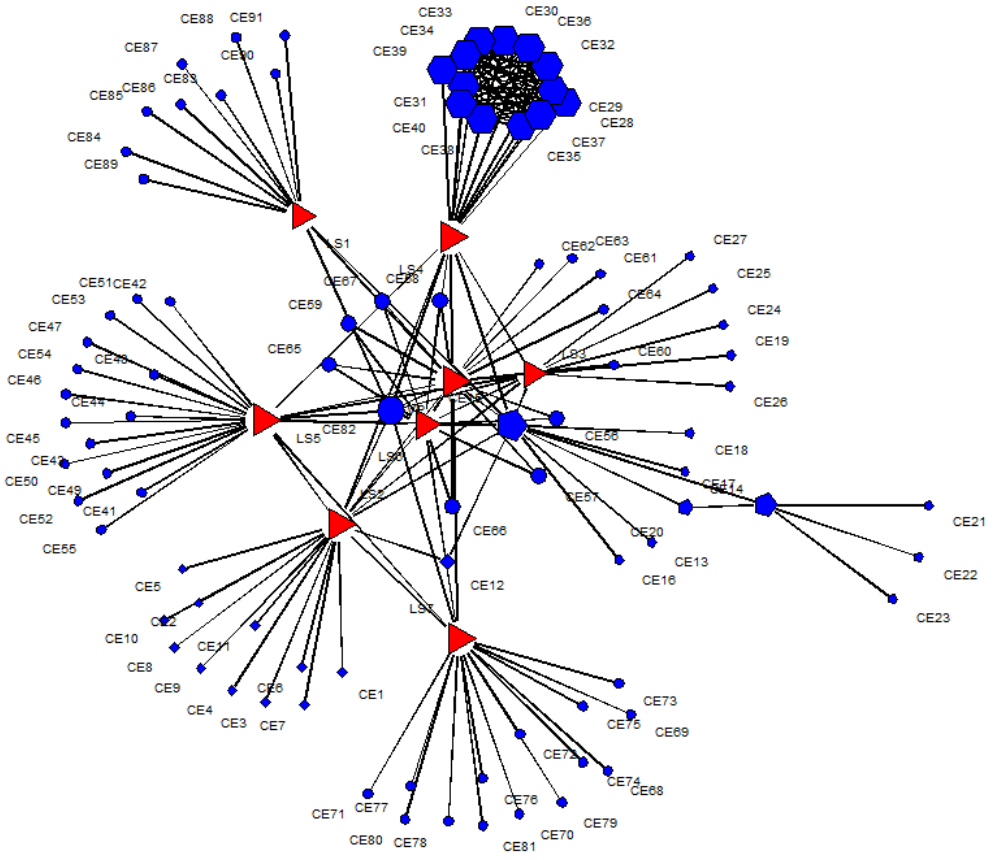
ORGANIZATIONAL NETWORK ANALYSIS
using R-studio



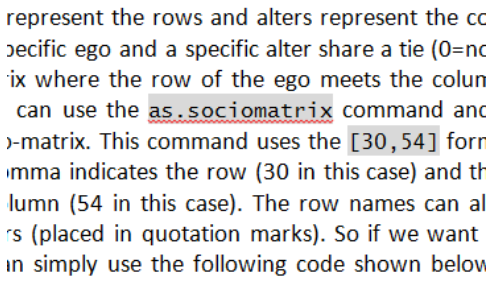
thom.de.vries@rug.nl
Faculty of Science & Engineering
Industrial & Engineering Management
University of Groningen
Groningen, the Netherlands
Version: 1.1.5, December 2019

CONTENTS

1 HOW TO USE THIS BOOK

Each section provides example programming codes for conducting organizational network analyses. Below, you find the explanation of how to use these codes.

To execute the code noted in some sections, you first need to execute the code given in previous sections. For example, in order to add vertex attributes to a loaded network data file (4.3.3), you first need to install/load relevant R packages (4.3.1), and load the socio-matrix (4.3.2).


represent the rows and alters represent the cc
ɔecific ego and a specific alter share a tie (0=nc
ix where the row of the ego meets the colun
can use the as.sociomatrix command anc
ɔ-matrix. This command uses the [30,54] forr
ımma indicates the row (30 in this case) and th
lumn (54 in this case). The row names can al
s (placed in quotation marks). So if we want
ın simply use the following code shown below

In-text highlights

Help you to spot programming code in the running text. The text surrounding the code provide an explanation of the use and purpose of the highlighted code.

```
network_data_sociomatrix
 Network attributes:
  vertices = 99
  directed = FALSE
  hyper = FALSE
  loops = FALSE
  multiple = FALSE
  bipartite = FALSE
  total edges= 210
    missing edges= 0
    non-missing edges= 210

 Vertex attribute names:
    vertex.names

 No edge attributes
```

Code boxes

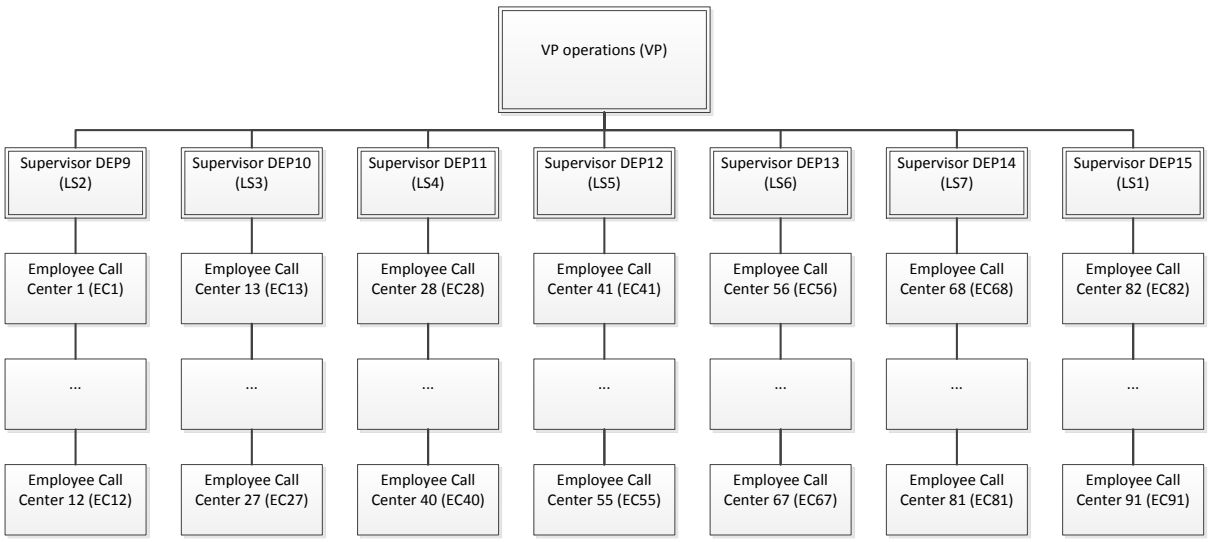
Contain actual output from organizational network analyses conducted within R-studio. The blue text shows the code that can be executed in R. You can copy-and-paste this to your R script and then execute it by selecting the line of code and pressing CTRL + ENTER. You should then get the same output as the black text shown in the highlighted code box.

2 INTRODUCING OUR CASE ORGANIZATION

To get hands-on experience in organizational network analysis (ONA), we will mimic a situation in which you work as a member of a consultancy team that is asked to advice a case organization. This case organization is experiencing problems related to its organizational network. *Throughout this book, it is our mission to make sense of the network data of this company and, ultimately, to help increase the effectiveness of the company.*

The case company is a medium-sized call center, employing exactly 99 employees and leaders, and it is located in Singapore. There are three hierarchical layers in the organization, namely (1) Vice President (VP), (2) first-line managers, and (3) call-center employees. Each first-line manager supervises between 10 and 15 call-center employees, and first-line managers are subsequently supervised by the VP. The company comprises 8 departments. The first department is the "Management team" and consists of the VP and the first-line supervisors. The remaining 7 departments are called DEP9-DEP15 and consist of call-center employees (CE1-CE91). The company relies on lateral communication within the management team to ensure integration between different departments. The organizational chart is shown in Figure 1. Please note that the LS1 has switched to DEP15.

Figure 1 - Chart of case organization



In the upcoming chapters, we will analyze the organizational network of this company. For this purpose, we have received the following data from the VP:

1) Network data on who calls whom in the call center to discuss work-related topics. This file is named "company_network_data.csv".
2) Information on organizational members' roles and department affiliations. This file supplements the network data and helps to identify who is who in the network data file. This file is called "company_attribute_data.csv".
3) Information on how closely individuals work together in the company. This file is called "company_edge_closeness.csv".

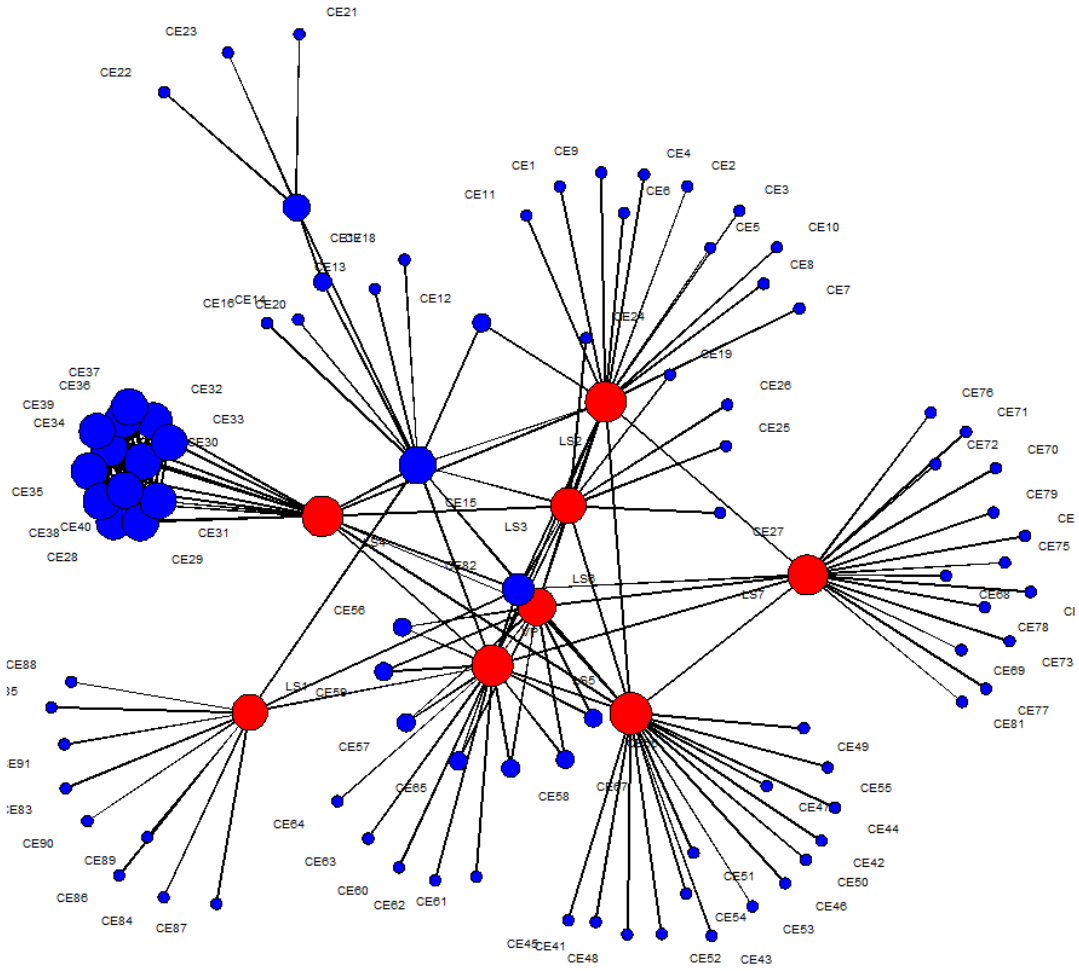
This data can be downloaded from Nestor.

3 ORGANIZATIONAL NETWORK ANALYSIS (ONA)

3.1 What is an organizational network (and why should we care)?

Employees in organizations typically do not work in isolation. Instead, they form relationships or 'ties' with their colleagues for a variety of reasons. For example, sometimes employees connect to other employees, because they want to get their advice, coordinate work, or exchange best-practices among each other. At other times, employees connect because they share a field of interest and like interacting with each other about this subject. An organizational network captures these interpersonal ties between employees and, thus, indicates who is connected to whom in the organization. Such organizational networks can be visualized as a network 'plot' or 'network graph', which displays all the employees that work in the organization as dots and connects the employees that share a tie with a line. The dots are called 'nodes' or 'vertices' and the lines are the 'ties' or 'edges'. Oftentimes, different colors or shapes are used to help distinguish between nodes from different departments, hierarchical layers, or work fields. By looking at the network graph, we thus get a good overview of how nodes are connected within and across important groups in the organization. Figure 2 provides an example network plot.

Figure 2 - Network graph of our case organization



Organizational networks can make or break an organization. Many organizations have been successful because their employees have built broad networks that allow cross-fertilization of ideas and company-wide coordination among different groups. In these organizations, networks have enabled innovation and efficiency. In other organizations, organizational networks are a source of inefficiency and ineffectiveness. What sometimes happens in these organizations is that employees that should be working with each other, in fact, do not work with each other. These employees will then fail to coordinate their work, align work schedules or exchange information on their work progress. As a result, work will be delayed, deadlines will be missed, and costly rework due coordination problems are inevitable.

3.2 What is ONA and how can we use it?

To properly understand the "inner social working" of an organization, we thus need to assess the organizational network that employees and supervisors use to complete work, exchange information, etc. This is, however, not an easy task. The organizational network is often hidden, as there is no formal document or chart that reliably specifies who works with whom and who is connected with whom inside the company. Hence, the organizational network needs to be uncovered through diligent research. One effective way to uncover an organizational network of a company is to conduct an "organizational network analysis", ONA. With ONA, we rely on data that provides information on who interacts with whom in the organization to discover how the overall organizational network looks like. Based on, for example, data logs of email traffic or data from a network questionnaire, we can get an accurate picture of who are the key employees in the organization and what kind of organizational network is in place. Armed with this information, we can diagnose problems in the organizational network (if any) and, if needed, devise appropriate solutions.

3.2.1 Step 1: Collecting data

ONA works as following. First, network data is collected on who works with whom in the organization. This information can come from organizational systems, such as email servers, agenda logs, telephone records, as well as from questionnaires in which we ask employees with whom they work. Objective data (i.e., email- or telephone logs) are preferable, because such data are unaffected by biases that may affect responses to questionnaires (e.g., employees may forget about their connections, they may not want to report certain connections, or simply lack the time to complete the survey).[1] If objective data are missing, questionnaires can be used. When questionnaires are used, employees of the organization typically receive an on-line survey with a request to select all employees with whom they frequently work or interact, followed by a list of all employees in the organization.[2] Data that results from such a questionnaire approach are generally reliably, but employees tend to forget about their weak ties (e.g., colleagues they only talk to once every month).

In addition to collecting data on who works with whom, we also need to collect background information on the individuals in the organization, such as their department affiliation, functional roles, etc. Later on, we can then use such information to diagnose, for example, connectivity within and across departments. Also, we sometimes collect information on the nature of specific ties between individuals. If a questionnaire approach is used, we can use

[1] For an example study using e-mail data to analyze organizational networks, see Kleinbaum (2013).
[2] For an example study using a questionnaire approach, see de Vries et al. (2014).

follow-up questions and ask individuals to rate the closeness or effectiveness of each of their ties in the organization on a scale ranging from 1 (not at all) to 5 (to very high extent). It is important to follow privacy and ethical rules when network data is collected or used. Following these rules will, in almost all cases, require researchers to ask employees for consent, consult an institutional research board (IRB), and aggregate/anonymize data.

3.2.2 Step 2: Structuring the data

3.2.2.1 Socio-matrix

In the second step, we need to structure data in a way that it can be analyzed with ONA. Specifically, the collected information on who interacts with whom needs to be structured as a big matrix, with the rows and the columns representing the individuals in the organization. This matrix is called an **adjacency** matrix or a "**socio-matrix"**. The rows in a socio-matrix indicate the "**egos**" in the network (i.e., the employees in the organization) and the columns indicate the "**alters**" (i.e., the potential partners of the egos). If an ego has a tie with an alter, this is indicated by placing a "1" in the cell of the matrix where the row of the ego intersects the column of the alter. If an ego and an alter do not have a tie, this is indicated by a "0". A socio-matrix should be "square", meaning that the number of egos (rows) should match the number of alters (columns). Also, the list of alters and egos should be equal in length, should include the exact same individuals, and should be ordered in the same way (e.g., alphabetically, based on team affiliation, etc.) to avoid running into errors when conducting ONAs. An example socio-matrix is given in Table 1. In this matrix, the names of the employees (John, Sally, Sue, and Rob) are displayed as both the row and the column headers. The 1s in the table indicate the presence of a tie between an ego and an alter, so you can tell that Sally has ties with John and Sue. The 0s indicate the absence of a tie, as is the case between Sally, on the one hand, and Rob, on the other hand. The diagonal cells in a socio-matrix always display 0s, as it is impossible for individuals to have ties with themselves.

Table 1 - Socio-matrix

	John	Sally	Sue	Rob
John	0	1	1	1
Sally	1	0	1	0
Sue	1	1	0	1
Rob	1	0	1	0

3.2.2.2 Vertex attribute information

Third, we need to structure the supplementary data file that provides background information on egos in the organization. Egos' background information needs to converted in a so-called "vertex attributes list". The rows in this list represent the egos and the columns provide information on their attributes. Column headers specify the attribute names.

The first column should be called "vertex.names" and provide the labels or names of the egos in the organization. Subsequent columns provide information on additional background characteristics, such as egos' functions or hierarchical levels (see Table 2). Importantly, the attribute list and socio-matrix should be ordered in the same way. For example, an individual in row 56 (as an ego) of the socio-matrix, should also be in row 56 of the attribute list.

Table 2 – Vertex attribute list

Vertex.names	Department	Hierarchical_level	Function
John	1	1	1
Sally	1	1	2
Sue	2	1	3
Rob	2	2	4

3.2.2.3 Edge attribute information

Fourth, we need to structure the edge attribute information. Let's first discuss what edge attributes represent. Sometimes we not only know that two employees have a tie (as captured in the socio-matrix), but we also have additional information on the nature of their tie. For example, in a questionnaire, we often ask individuals follow-up questions and ask them how close they are with the individuals they have ties with in the organization on a 5-point scale (1 = not close at all ~ 5 = very close). Alternatively, we could ask employees how much they benefited from the specific ties they have with other employees or how much time they invested in their ties. Moreover, we could use external files to gain more information on ties between employees. We could, for example, use personnel records to determine the team affiliation of the two employees who share a tie. Subsequently, we can determine if the tie between two employees represents a within-team tie (i.e., both employees work in the same team) or a between-team tie (i.e., employees work in different teams). In all cases, we have detailed information on ties (e.g., tie closeness, effectiveness, etc.) that we later on need to add to the socio-matrix as 'edge attributes' (see 4.3.4).

Edge attributes information needs to be structured as a matrix (see Table 3), similar to the socio-matrix shown in Table 1. Whereas the socio-matrix provides information on the *presence* of ties between employees, the edge-attributes matrix provides details on the attributes of these ties. Table 1 indicates that John and Sally have a tie with each other, and Table 3 indicates that this ties is of average closeness (i.e., a score of 3 on a 1-5 scale).

Table 3 – Matrix with edge attribute "closeness"

	John	Sally	Sue	Rob
John	0	3	4	1
Sally	3	0	2	0
Sue	4	5	0	2
Rob	1	0	2	0

3.2.3 Step 3: Running the analyses

After the data has been structured correctly, the next step is to load and integrate files into a program that can be used for ONA. One program that is often used is R-studio, which is an open source platform that can be used to run all sorts of statistical analyses, including ONA. What kind of ONA is executed, depends on the research question. We will discuss how to conduct ONA in R-studio in the subsequent sections.

4 GETTING STARTED WITH ONA IN R-STUDIO

4.1 Why use R-studio for ONA?

There are several software packages that can be used to conduct ONA, including Pajek, Gephi, UCINET, NodeXL, and R-studio. Here, we rely on R-studio for the following reasons. First of all, R-studio is a general statistical programming language. Mastering R-studio will therefore not only allow you to conduct ONA, it will also enable you to learn other types of analyses that you might want to conduct in the future (e.g., structural equation modeling, multilevel regressions, etc.). Second, R-studio is open-source software, which has enabled many software developers to contribute to R-studio by developing dedicated modules or 'packages' for conducting, among other things, sophisticated ONA. Among these packages are "statnet" and "sna", which we will primarily rely on in this book. We will use these packages in combination with more general statistical or data management packages included in R-studio, thereby benefiting from R-studio's versatility. Thirdly, because the popularity of R-studio, there are a lot of on-line resources to help you get started.

4.2 Installing and running R-studio

To conduct ONA, we first need to install R-studio and, subsequently, download the relevant packages for conducting ONA. If you are using a University of Groningen computer, R-studio is standardly available in the Windows Start menu. To install R-studio on your personal computer, please follow the steps listed below:

1) Download the latest version of R from: https://cran.r-project.org/
2) Download the latest "Rstudio desktop" from: https://rstudio.com/products/rstudio/download/

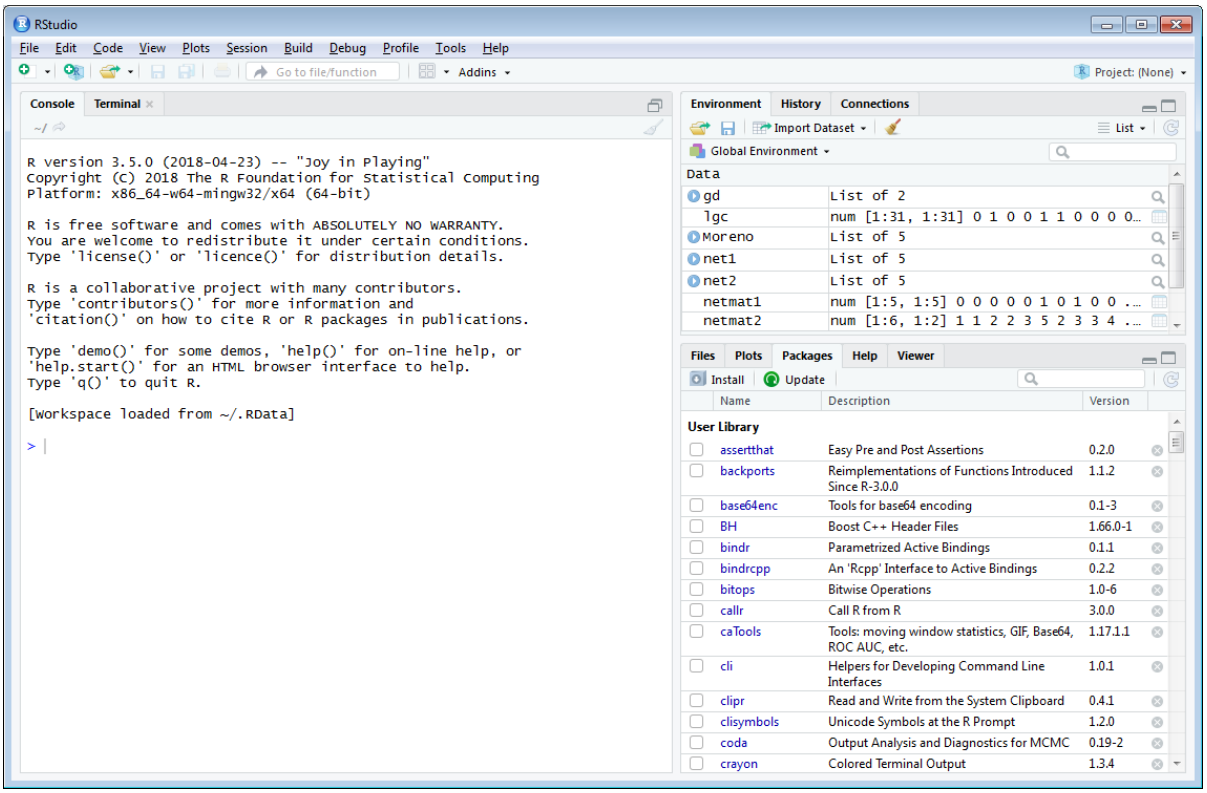
After installing these files, run R-studio and you should get an opening screen resembling Figure 3. In this screen, you see the "console," which is used to enter code into R-studio, as well as tabs on the right for "Environment" and "Packages". The environment tab displays the data objects that are loaded into R-studio. Later on you will load a socio-matrix into R-studio. That data will then be displayed in the environment as a data object under the label you have assigned to it. The packages tab displays the software modules that have been installed in R-studio. Packages that have been ticked are loaded and can be used.

4.3 Loading network data into R-studio

Before we can conduct ONA in R-studio, we need to load the network data (the socio-matrix specifying who interacted with whom in the case organization). There are, again, several steps to complete. Here's an overview of these steps:

1) First, we have to **load the packages** that enable R-studio to recognize the network data. To be exact, we need to load the packages "statnet" and "sna".
2) Second, we need to **load the socio-matrix** in R-studio.

Figure 3 - R-studio opening screen



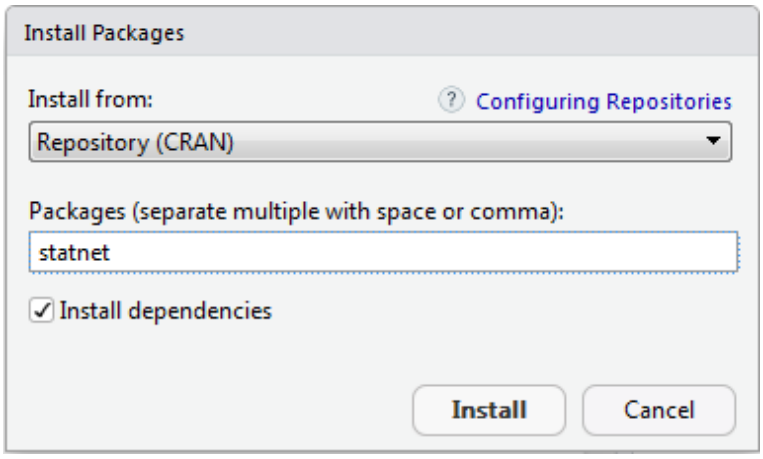
3) Third, we need to tell R-studio which row and column in the socio-matrix (step 2) belongs to which ego and alter. In other words, we have to **load the vertex attribute list** (i.e., characteristics of the employees in the network, such as their names, function, team affiliation, etc.) and attach it to the socio-matrix.
4) Fourthly, for some analyses, we need to specify the nature of the tie between two nodes. We could, for example, add information on how closely two individuals are connected to each other on a 1 to 5 scale (1 = not close all ~ 5 = very close). To add this information, we need to **load the edge attribute information**.

We will discuss these steps, and associated R-studio code, in depth below.

4.3.1 Loading ONA packages

Once R-studio is successfully installed, the next step is to install and load the relevant package for conducting ONA, namely “statnet” and “sna”. There are two ways to install and load packages: (1) via the menu structure, and (2) via typing in code into the R-studio console. For the menu option, go to the “packages” tab in the lower right box of R-studio, then click on the button “install” and enter the name of the package you are looking to install in the search box. In our case that would be “statnet” (and later “sna”), see Figure 4. Search for the package and press “install”. Make sure the box for “install dependencies” is ticked. Once you press install, the package is installed and loaded. Please note that R-studio might display some errors, which have to do with other required packages that need to be installed next to statnet. As long as the “install dependencies” box is checked, these errors are resolved automatically and statnet will work normally.

Figure 4 - Installing packages via the menu structure



For the console option, it is good practice to start with opening a new **"script" file**. A script file allows you to save all the code that you entered in a R-studio session. This will enable you to replicate your actions at a later time and rerun analyses, data manipulations, etc. To create a new script, click on File > New file > R Script (or simply press CTRL + Shift + N). The new script file is then automatically opened and the cursor is placed in line 1 of the script. Here you can start entering code. Please save your script file at regular intervals. To install the package statnet and sna, please enter the following code to your R script:

```
install.packages("statnet")
install.packages("sna")
```

Then locate your cursor in this code line and press CTRL + ENTER to execute the code. Once this code is executed, the package is installed, added to the library, and automatically loaded. If you exit R-studio and open it again, the package will still be in your R library, but it is not loaded automatically. Hence, you have to load all relevant packages each time you start up R-studio. To load a pre-installed package from the library, add the following code to your script file and again press CTRL + ENTER to run it:

```
library("statnet")
library("sna")
```

4.3.2 Loading the socio-matrix

Before we can load the socio-matrix, we need to identify the working directory that is set in R-studio. All data files should be located in this working directory. Enter:

```
getwd ()
```

in the script file, followed by CTRL + Enter, to discover the current working directory. If you are satisfied with the current working directory's location, simply copy your data files to this directory. If not, you can change the working directory by executing the following code:

```
setwd("X:/ONA")
```

This will change the working directory to the directory specified in the quotation marks. After changing the directory, copy the data to the new working directory (this is not done

automatically). Note that MAC OS or Linux computers require backward slashes (i.e., "\") instead of forward slashes (i.e., "/") to specify the location of the working directory.

Once the data has been copied into the working directory, we can load the data. Usually, network data is saved in the comma-delimited format (.csv). The socio-matrix network data of our case organization is, for example, stored in the file "company_network_data.csv". Please do not work on this file in Microsoft Excel, as this may change the format of the data file and make it incompatible with R-studio. We can load that data as a data object in the R-studio environment by adding the following code to the script:

```
network_data <- read.csv("company_network_data.csv", header=F)
```

Executing this code will load the "company_network_data.csv" file and store it in the R-studio environment as a data object that is labeled "network_data". We can use whatever label we like. As is common for network data, the.csv file does not contain row or column names and R-studio needs to be informed about that in the code. This is done by adding `header=F`. This tells R-studio that the data file only contains tie information and excludes any attribute information about the nodes in the network (such as names).

Next, we need to specify that the "network_data" object represents a network, as this will allow us to use this data in subsequent organizational network analyses. To do so, we inform R-studio that the data stored under the label network_data is, in fact, a socio-matrix (rather than a 'normal' matrix). We can do this by using `network()`:

```
network_data_sociomatrix <- network(network_data, directed=FALSE)
```

This will create an object in the R-environment, labeled "network_data_sociomatrix," that is the same as "network_data" but now correctly classified as a socio-matrix. It is important to specify here whether the network data is "directed" or "undirected". Directed network data means that when Sue indicates to have a tie with John, this does NOT automatically mean that John also has a tie with Sue. This is often the case when we look at advice networks and ask individuals to specify from whom they get advice. In this situation, Sue might obtain advice from John, but John may not necessarily also obtain advice from Sue, giving rise to asymmetrical, directed networks. Undirected networks, on the other hand, capture ties that are reciprocal by nature, such as friendship networks. When we look at such networks, Sue's indication that she is friends with John automatically means that John is also friends with Sue. When left unspecified, network data is automatically loaded as directed. To specify an undirected network, we have to add `directed=FALSE`, as we did above. Failure to specify the nature of network data (directed or undirected) will result in inaccurate ONA output.

To check whether R-studio now correctly recognizes our network data, we can simply enter the name of the network data object and press CTRL + ENTER:

```
network_data_sociomatrix
```

The R-studio output will then give you the summary statistics of the network data that was loaded, as displayed below. The output will list the number of employees in the network (i.e., number of vertices; 99 in our case), the number of ties between these employees (i.e.,

number of edges; 210 in our case), and whether the network is directed or not. It is good practice to carefully review this information before you proceed further.

```
network_data_sociomatrix
 Network attributes:
  vertices = 99
  directed = FALSE
  hyper = FALSE
  loops = FALSE
  multiple = FALSE
  bipartite = FALSE
  total edges= 210
    missing edges= 0
    non-missing edges= 210

 Vertex attribute names:
    vertex.names

 No edge attributes
```

After completing these steps, we now have successfully loaded the socio-matrix in R-studio. In the next steps, we will add the vertex and edge attributes to this socio-matrix.

4.3.3 Loading vertex attributes

Vertices are individuals in the organization and vertex attributes, correspondingly, specify the characteristics of those individuals in the organizational network. We can load vertex attributes from external files (e.g., personnel records, questionnaires), but we can also use ONA to calculate individuals' network positions (i.e., how many ties they have with colleagues) and add that information to the socio-matrix as a vertex attribute. Typical vertex attributes from external files include (a) name or abbreviation of the individual, (b) team or departmental affiliation, (c) hierarchical level, and (d) function or work field. A typical vertex attribute calculated by ONA includes how many ties an individual has in the organization.

We will start with discussing how we can load vertex attributes from an external file. This file is located on Nestor and is called "company_attribute_data.csv". This file has 99 rows (plus one row for the column header), with one row for each individual within the network we loaded in the previous step. Moreover, there are two columns in the file. The first column specifies an anonymized name (called "vertex.names") of each employee, and the second column specifies the department affiliation of employees (with a number, called "dep"). We first need to load the .csv file as a data object and assign a label to it (we will use "attributes"), as we did for the socio-matrix:

```
attributes <- read.csv("company_attribute_data.csv")
```

Then we need to attach the information in the data object "attributes" to the socio-matrix that we labeled "network_data_sociomatrix" using `set.vertex.attribute` as following:

```
set.vertex.attribute([label of the sociomatrix,names([label of the
attributes file]), [label of the attributes file])
```

Translated to our data, this results in:

```
set.vertex.attribute(network_data_sociomatrix,names(attributes),
attributes)
```

We can check if the attribute was added correctly by executing the code:

```
list.vertex.attributes(network_data_sociomatrix)
[1] "dep"          "na"           "vertex.names"
```

This will list the labels of the attributes that we added to the socio-matrix, as taken from the column headers of the attributes file. In our case, two attributes were added, namely "vertex.names" and "dep". Sometimes an empty attribute called "na" vertex attribute is also mentioned in the list. This "na" vertex attribute can be ignored. The "vertex.names" attribute is standardly created when you load a socio-matrix into R-studio. However, until we load the vertex attributes, this attribute is not populated (i.e., it is empty).

We can then display vertex attribute values by using `get.vertex.attribute` followed by the name of the attribute, as shown below. R-studio displays the vertex attributes "vertex.names" and "dep" in the wide format (from left to right). The number placed in brackets indicates the position of the first number in the subsequent row. For example, `[52]` next to the name label `CE44` indicates that CE44 is in the 52th position in the socio-matrix. CE45 is in the 53th position, CE46 in the 54th position, etc.

```
get.vertex.attribute(network_data_sociomatrix,"vertex.names")
 [1] VP   LS1  LS2  LS3  LS4  LS5  LS6  LS7  CE1  CE2  CE3  CE4  CE5  CE6  CE7  CE8  CE9
[18] CE10 CE11 CE12 CE13 CE14 CE15 CE16 CE17 CE18 CE19 CE20 CE21 CE22 CE23 CE24 CE25 CE26
[35] CE27 CE28 CE29 CE30 CE31 CE32 CE33 CE34 CE35 CE36 CE37 CE38 CE39 CE40 CE41 CE42 CE43
[52] CE44 CE45 CE46 CE47 CE48 CE49 CE50 CE51 CE52 CE53 CE54 CE55 CE56 CE57 CE58 CE59 CE60
[69] CE61 CE62 CE63 CE64 CE65 CE66 CE67 CE68 CE69 CE70 CE71 CE72 CE73 CE74 CE75 CE76 CE77
[86] CE78 CE79 CE80 CE81 CE82 CE83 CE84 CE85 CE86 CE87 CE88 CE89 CE90 CE91
99 Levels: CE1 CE10 CE11 CE12 CE13 CE14 CE15 CE16 CE17 CE18 CE19 CE2 CE20 CE21 CE22 ... VP

get.vertex.attribute(network_data_sociomatrix,"dep")
 [1]  1  2  3  4  5  6  7  8  9  9  9  9  9  9  9  9  9  9  9 10 10 10 10 10 10 10 10 10
[30] 10 10 10 10 10 10 11 11 11 11 11 11 11 11 11 11 11 11 11 12 12 12 12 12 12 12 12 12
[59] 12 12 12 12 12 13 13 13 13 13 13 13 13 13 13 13 13 14 14 14 14 14 14 14 14 14 14 14
[88] 14 14 15 15 15 15 15 15 15 15 15 15
```

4.3.4 Loading edge attributes

Next, we will load the edge properties. In our case organization, we used a follow-up questionnaire and asked individuals how close they are with the individuals they have ties with on a 5-point scale (1 = not close at all ~ 5 = very close). As discussed in 3.2.2.3, such edge attribute information is typically stored in a matrix (see Table 3), and then saved in a .csv file format. Indeed, for our case organization, we have a matrix file containing the closeness scores for all the ties. This file is called "company_edge_closeness.csv" and it does not contain row or column names.

There are two ways to load edge attribute data, method 1 and method 2. In method 1, we use the edge-attribute file to directly load *both* the socio-matrix and edge attributes. In method 2, we first load the socio-matrix (from a separate file) and then add the edge-attributes to that socio-matrix. Which method we need depends on the number of edge attributes we want to add. If there is only one edge attribute (e.g., only closeness), we can use method 1. Method 1, however, does not work when multiple edge attributes need to be

added (e.g., in case we want to add tie closeness *and* tie effectiveness as edge attributes). If we would use method 1 in this case, edge-attributes will be overwritten. Hence, if we need to add multiple edge attributes, we should use method 2. Using method 2, we can add as many edge attributes as we like. Please note that it is also possible to start with method 1 to load the socio-matrix and the first edge attribute (e.g., tie closeness) and then use method 2 to add additional edge attributes (e.g., tie effectiveness). Below, we discuss how we can use method 1 and 2 to load a single edge attribute, called closeness.

Method 1 involves loading the edge attribute file directly as a socio-matrix. In this case, we have to tell R-studio that we want to load a network socio-matrix from a matrix file that contains valued (i.e., in our case, values range from 0 to 5) rather than binary data (i.e., 0). In addition, we have to tell R-studio what to do with the valued data. In our case, the values denote closeness scores and, thus, should be stored as an edge attribute called "closeness". By specifying `ignore.eval=FALSE` and `names.eval="closeness"`, R-studio will do exactly that and retain the values (rather than converting them into binary data) and save the values as an edge attribute labeled "closeness". As before, we need to specify that the network data is undirected by adding `directed=FALSE` to the code.

```
edge <- as.matrix(read.csv("company_edge_closeness.csv", header=F))

network_data_sociomatrix<-
as.network(edge,directed=FALSE,ignore.eval=FALSE,names.eval="closeness")
```

In method 2, we attach the edge attribute information to an already loaded socio-matrix (for example, a socio-matrix loaded by using method 1). This requires us to complete the following steps. We first need to load the edge attribute .csv file and assign a label to it (we will use the label "edge") using `read.csv` with `header=F`. Then, we need to attach the information from the edge file to the socio-matrix we already loaded and have labeled "network_data_sociomatrix". We can use the `set.edge.value` for this. We label the edge attribute we want to add "closeness":

```
edge <- as.matrix(read.csv("company_edge_closeness.csv", header=F))

network_data_sociomatrix<-set.edge.value(network_data_sociomatrix,
'closeness',edge)
```

Afterwards, we should check whether the edge closeness information was attached correctly to the socio-matrix by requesting some statistics (average, minimum, maximum score) using `summary`, as shown below. In the summary code, we indicate the label of the socio-matrix, followed by `%e%` and the name of the edge attribute we want to inspect. The `%e%` tells R-studio that we are referring to an edge attribute embedded within "network_data_sociomatrix" (for vertex attributes, you can use `%v%`).

```
summary(network_data_sociomatrix %e% "closeness")
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    1.0     2.0     3.0     3.2     5.0     5.0
```

In addition, we can display the raw closeness scores in the socio-matrix by using `get.edge.attribute`. R-studio will then display the edge attribute scores for all existing

ties. In our case, there are 210 edges in the network, so R-studio correspondingly displays a list comprising 210 edge attribute scores.

```
get.edge.attribute(network_data_sociomatrix,"closeness")
  [1] 2 4 2 4 1 2 2 5 4 5 4 3 2 1 4 5 2 5 1 5 5 2 3 5 4 5 4 2 5 3 5 4 5 1 5 4 4 3 2 1 2 4 4 5 4 3 2
 [48] 3 3 2 4 3 1 5 1 2 3 2 1 3 1 3 2 4 2 2 3 4 1 5 5 4 4 4 1 1 4 3 4 3 1 4 1 3 2 3 5 3 5 1 5 2 2 3
 [95] 4 5 5 5 2 1 2 4 2 3 5 2 2 3 4 4 2 5 3 5 5 2 5 2 5 5 5 4 3 3 5 5 1 4 5 1 1 5 3 3 1 2 3 5 5 3 1
[142] 5 2 2 1 1 4 2 3 2 3 2 3 5 4 5 4 5 4 5 1 2 1 2 1 3 3 1 4 5 5 5 1 3 1 3 4 4 5 2 5 1 3 2 3 5 5 4
[189] 4 3 2 2 5 4 2 5 2 1 5 1 1 5 5 5 5 3 4 2 3 5
```

Finally, we can request the summary statistic for the socio-matrix (as we did in 4.3.2) to check if closeness is now added as an edge attribute to the network. As shown below, the summary output now lists “closeness” as an edge attribute, as well as the vertex attributes that we added in 4.3.3.

```
network_data_sociomatrix
 Network attributes:
  vertices = 99
  directed = FALSE
  hyper = FALSE
  loops = FALSE
  multiple = FALSE
  bipartite = FALSE
  total edges= 210
    missing edges= 0
    non-missing edges= 210

 Vertex attribute names:
    dep vertex.names

 Edge attribute names:
    closeness
```

5 NETWORK DATA MANIPULATION IN R-STUDIO

5.1 Accessing network data in R-studio

Before we start analyzing the network data, let us first shortly explore how we can access network data that includes vertex and edge attribute information.

5.1.1 Accessing vertex and edge attribute information

As discussed in 4.3.3 and 4.3.4, we can display the names of the attributes of our network data by using `list.vertex.attributes` and `list.edge.attributes`. By executing these codes, we learn that there are two vertex attributes attached to our network data object (i.e., " dep" and " vertex.names") and one edge attributes ("closeness"). There might also be a "na" listed as an attribute; this can be ignored.

```
list.vertex.attributes(network_data_sociomatrix)
[1] "dep"      "vertex.names"

list.edge.attributes(network_data_sociomatrix)
[1] "closeness"
```

Once we know the names of the vertex and edge attributes, we can display the raw values of these attributes by using `get.vertex.attributes` and `get.edge.attributes`, followed by the label of the network data object and attribute, as shown below for the attribute "dep". This will provide a list of the attribute scores of the egos in the network. Again, the order of the attribute values corresponds to the order of egos in the socio-matrix, such that the person who is in position 1 in the attribute list is also in position 1 of the socio-matrix.

```
get.vertex.attribute(network_data_sociomatrix,"dep")
 [1]  1  2  3  4  5  6  7  8  9  9  9  9  9  9  9  9  9  9  9 10 10 10 10 10 10 10 10 10
[30] 10 10 10 10 10 10 11 11 11 11 11 11 11 11 11 11 11 11 11 12 12 12 12 12 12 12 12 12
[59] 12 12 12 12 12 13 13 13 13 13 13 13 13 13 13 13 13 14 14 14 14 14 14 14 14 14 14 14
[88] 14 14 15 15 15 15 15 15 15 15 15 15
```

5.1.2 Accessing tie information

Sometimes we want to know if two specific individuals share a tie with each other. Here is how to extract that information from a socio-matrix. Remember that in a socio-matrix, rows represent egos and columns represent alters. Moreover, information on whether a specific ego and a specific alter share a tie (0=no, 1=yes) is located in the cell of the socio-matrix where the ego's row meets the alter's column. To obtain this information, we can use `as.sociomatrix` and display values of specific cells within the socio-matrix. As.sociomatrix uses the `[30,54]` format to refer to locate specific cells, where the number left from the comma refers to the row of the cell (30 in this case) and the value right from the comma refers to the column of the cell (54 in this case). The row and column numbers can also be replaced by the row and column headers (placed in quotation marks). So if we want to know if ego "CE64" has a tie with alter "LS4", we can simply use the code shown below. This returns the value "0", indicating that this ego and alter do not share a tie with each other.

```
as.sociomatrix(network_data_sociomatrix)["CE64","LS4"]
[1] 0
```

If you want to refer to row and column numbers (instead of using the row and column headers, as we did above), we need to complete a couple of additional steps. First, we need discover which individual is in which row (as an ego). For this purpose, we can retrieve information on the vertex.names values and positions in the network file, as shown below.

```
get.vertex.attribute(network_data_sociomatrix,"vertex.names")
 [1] VP   LS1  LS2  LS3  LS4  LS5  LS6  LS7  CE1  CE2  CE3  CE4  CE5  CE6  CE7  CE8  CE9
[18] CE10 CE11 CE12 CE13 CE14 CE15 CE16 CE17 CE18 CE19 CE20 CE21 CE22 CE23 CE24 CE25 CE26
[35] CE27 CE28 CE29 CE30 CE31 CE32 CE33 CE34 CE35 CE36 CE37 CE38 CE39 CE40 CE41 CE42 CE43
[52] CE44 CE45 CE46 CE47 CE48 CE49 CE50 CE51 CE52 CE53 CE54 CE55 CE56 CE57 CE58 CE59 CE60
[69] CE61 CE62 CE63 CE64 CE65 CE66 CE67 CE68 CE69 CE70 CE71 CE72 CE73 CE74 CE75 CE76 CE77
[86] CE78 CE79 CE80 CE81 CE82 CE83 CE84 CE85 CE86 CE87 CE88 CE89 CE90 CE91
```

Based on this output, we know that "CE64" is in position 72 of the socio-matrix. This means that this person, correspondingly, is in row 72 as an ego and column 72 as an alter (remember that rows and columns of a socio-matrix have equal lengths and are ordered in the same way). Now, suppose that we want to know if "CE64" has a tie with "LS4". From inspecting the vertex.names values and positions, we know that "LS4" is in position 5 in the socio-matrix. We already knew that "CE64" is in position 72. Combined, this tells us that the information on whether or not "CE64" and "LS4" have a tie is located in the cell of the socio-matrix where row 72 meets column 5. To access this information, we can use the code shown below. We find that the cell on the intersection of row 72 and column 5 indicates a "0", which means that there is no tie between "CE64" and "LS4".

```
as.sociomatrix(network_data_sociomatrix)[72,5]
[1] 0
```

Sometimes, we want to know whether an individual has ties with *multiple* alters from a larger group, such as his or her team or department. For example, we might want to know if the VP of the case organization has ties with other members of the management team (i.e., LS1 – LS7). To get this information, we should retrieve the values of the cells where the row of the VP (row 1) intersects with the columns of the other leaders (columns 2:8). To do this, we can refer to the row number and the *range* of columns in the socio-matrix:

```
as.sociomatrix(network_data_sociomatrix)[1,2:8]
LS1 LS2 LS3 LS4 LS5 LS6 LS7
  1   1   1   1   1   1   1
```

Alternatively, we can access this information by referring to the row and column *names*, using the code below. In this code, we need to use c, followed by a list of the column names in quotation marks and separated by commas, to refer to multiple columns simultaneously. It is not possible to directly refer to a range of column names (i.e., LS2:LS7); this is only possible when using row or column *numbers*.

```
as.sociomatrix(network_data_sociomatrix)["VP",c("LS1","LS2","LS3","LS4","LS5","LS6"
,"LS7")]
LS1 LS2 LS3 LS4 LS5 LS6 LS7
  1   1   1   1   1   1   1
```

Table 4 shows additional examples of how to obtain information on ties by referring to specific (ranges) of rows and columns using `as.sociomatrix`. Here "x" denotes the label of the network data object.

Table 4 - As.sociomatrix examples

Code	Displays information on
`as.sociomatrix(x)[,]`	All cells
`as.sociomatrix(x)[,5:10]`	Cells on the intersection between all rows and columns 5-10
`as.sociomatrix(x)[4:18,]`	Cells on the intersection between rows 4-18 and all columns
`as.sociomatrix(x)[c(4:18,99),]`	Cells on the intersection between rows 4-18 & 99 and all columns

It is also possible to calculate summary statistics on egos in the network by combining `as.sociomatrix` with `rowSums`. The rowSums code calculates the sum of all cells belonging to rows within the matrix. In case of a socio-matrix, the rowSums code calculates the total number of ties of egos, because each row represents a different ego. To calculate the total number of ties for all employees (as egos), we do not specify a range of rows or columns in the code (i.e., we refer to the complete matrix as `[,]`):

```
rowSums((as.sociomatrix(network_data_sociomatrix)[,]))
  VP  LS1  LS2  LS3  LS4  LS5  LS6  LS7  CE1  CE2  CE3  CE4  CE5  CE6  CE7  CE8  CE9 CE10
  21   12   20   12   20   23   15   19    1    1    1    1    1    1    1    1    1    1
CE11 CE12 CE13 CE14 CE15 CE16 CE17 CE18 CE19 CE20 CE21 CE22 CE23 CE24 CE25 CE26 CE27 CE28
   1    2    1    1   14    1    5    1    1    2    1    1    1    1    1    1    1   12
CE29 CE30 CE31 CE32 CE33 CE34 CE35 CE36 CE37 CE38 CE39 CE40 CE41 CE42 CE43 CE44 CE45 CE46
  13   13   13   13   13   13   13   13   13   13   12   13    1    1    1    1    1    1
CE47 CE48 CE49 CE50 CE51 CE52 CE53 CE54 CE55 CE56 CE57 CE58 CE59 CE60 CE61 CE62 CE63 CE64
   1    1    1    1    1    1    1    1    1    2    2    2    2    1    1    1    1    1
CE65 CE66 CE67 CE68 CE69 CE70 CE71 CE72 CE73 CE74 CE75 CE76 CE77 CE78 CE79 CE80 CE81 CE82
   2    2    2    1    1    1    1    1    1    1    1    1    1    1    1    1    1    8
CE83 CE84 CE85 CE86 CE87 CE88 CE89 CE90 CE91
   1    1    1    1    1    1    1    1    1
```

If we want to calculate the number of ties for certain individuals, we need to refer to the rows of these individuals in the socio-matrix. For example, we might be interested in how many ties certain managers have with all other members of the company. We know that managers are in positions 1:8 in the network. Hence, we can use the following code to calculate the total number of ties of each manager:

```
rowSums((as.sociomatrix(network_data_sociomatrix)[1:8,]))
 VP LS1 LS2 LS3 LS4 LS5 LS6 LS7
 21  12  20  12  20  23  15  19
```

We can also calculate how many ties certain individuals have with (a group of) certain other individuals. For example, we can calculate how many ties each manager has with alters located in DEP9 (i.e., CE1 – CE12). In that case, we need to refer to the rows of the egos and the columns of alters in the socio-matrix. Managers are located in rows 1-8 (as egos), while CE1 – C12 are located in columns 9-20 (as alters). Hence, to calculate managers' ties with

members from DEP9, we can use the code specified below. Based on the output, we can conclude that only manager "LS2" has direct ties with members of DEP9. This makes sense, considering that "LS2" is the supervisor of DEP9 (see Figure 1).

```
rowSums((as.sociomatrix(network_data_sociomatrix)[1:8,9:20]))
 VP LS1 LS2 LS3 LS4 LS5 LS6 LS7
  0   0  12   0   0   0   0   0
```

5.1.3 Accessing attribute information

To access vertex attributes, we can use the `get.vertex.attribute` explained in 4.3.3. In addition, we can obtain individuals' attributes from the root attributes data object that we labeled "attributes" (see 4.3.3). This can be useful, as this root data object is formatted as a matrix. Thus, by referring to this data object, we can use matrix computations that are not available (or much more tedious) when relying on the socio-matrix for attribute information.

Similar to the socio-matrix, a row in the attributes data object corresponds to a certain individual. For example, the first row of the attributes file provides information on the VP. Furthermore, the order of individuals in the attribute file is consistent with that of the socio-matrix, such that the VP should also be in the first row of socio-matrix (as an ego). Unlike the socio-matrix, however, the columns represent variables (rather than alters, as in the socio-matrix). These variables represent individuals' attributes. To learn more about the variables included in the attributes data object, we can use `head` followed by the label of the attributes data object. This will display the variable names (i.e., vertex attributes in our case) and the values for the first five egos. When applied to the attributes data object "attributes", R-studio returns the names and first 5 values of the attributes "vertex.names" and "dep":

```
head(attributes)
  vertex.names dep
1           VP   1
2          LS1   2
3          LS2   3
4          LS3   4
5          LS4   5
6          LS5   6
```

To display the attributes of only certain individuals, we can refer to the rows of these individuals within the matrix of the attributes data object. For example, to display the attributes of the DEP9 members, we can refer to rows 9:20 (see code below). We can also display a selection of attributes by refer to the column numbers or labels of these attributes. The second example (shown below) details the code for displaying only the attribute "dep" for members of DEP9. Note that we do not need to convert the attributes file to a matrix (i.e., by using `as.matrix`), because this file is already formatted as a matrix.

```
attributes[9:20,]
   vertex.names dep
9           CE1   9
10          CE2   9
11          CE3   9
12          CE4   9
13          CE5   9
14          CE6   9
15          CE7   9
```

```
16          CE8   9
17          CE9   9
18         CE10   9
19         CE11   9
20         CE12   9

attributes[9:20,"dep"]
 [1] 9 9 9 9 9 9 9 9 9 9 9 9
```

Finally, we can display all values of an attribute (or any variable within any data object) by referring to the label of the data object followed by `$` and the label of the variable. To display the attribute "dep", for example, we can use the following code.

```
attributes$dep
 [1]  1  2  3  4  5  6  7  8  9  9  9  9  9  9  9  9  9  9  9 10 10 10 10 10 10 10 10 10
[30] 10 10 10 10 10 10 11 11 11 11 11 11 11 11 11 11 11 11 11 12 12 12 12 12 12 12 12 12
[59] 12 12 12 12 12 13 13 13 13 13 13 13 13 13 13 13 13 14 14 14 14 14 14 14 14 14 14 14
[88] 14 14 15 15 15 15 15 15 15 15 15 15
```

5.2 Excluding certain persons from the network

In some cases we are interested in the ties of some employees, but not all employees. For example, let's say that there have been problems on the work floor in the case organization. In that case, we might want to limit our ONA to the workers and exclude management from our analyses. To do so, we need to create an alternative socio-matrix, one in which the managers are filtered out and only workers remain. We can achieve this by using `get.inducedSubgraph` in combination with `which`. By combining these commands, we can create a filtered socio-matrix that only includes vertices that meet our inclusion criteria (e.g., workers in our case). The `get.inducedSubgraph` allows us to create the filtered socio-matrix and `which` allows us to specify the criteria of the vertices that should be included.

In our dataset, the supervisors have the vertex.names: "VP" and "LS1-LS7". To create a new socio-matrix that excludes individuals that have these vertex.names values, we correspondingly use the following code:

```
Network_data_workers <- get.inducedSubgraph(network_data_sociomatrix,which(
   network_data_sociomatrix %v% "vertex.names" != "VP"
   & network_data_sociomatrix %v% "vertex.names" != "LS1"
   & network_data_sociomatrix %v% "vertex.names" != "LS2"
   & network_data_sociomatrix %v% "vertex.names" != "LS3"
   & network_data_sociomatrix %v% "vertex.names" != "LS4"
   & network_data_sociomatrix %v% "vertex.names" != "LS5"
   & network_data_sociomatrix %v% "vertex.names" != "LS6"
   & network_data_sociomatrix %v% "vertex.names" != "LS7"))
```

As noted above, `which` allows us to select certain cases. In order to work properly, `which` needs to be followed by the name of the variable on which we want to filter (`network_data_sociomatrix %v% "vertex.names"` in our case) followed by the value that should be included or excluded (here, cases should be excluded if their vertex.names value is "VP" or "LS1"-"LS7"). In specifying the value for filtering, `!=` means "does not equal", while `==` means "equals". Moreover, `which` is flexible to multiple criteria. We can add additional criteria by using the `&` or `|` signs. The `|` sign means "OR", while `&` means "AND". After executing this code, we can request the summary for the new, filtered socio-matrix that we

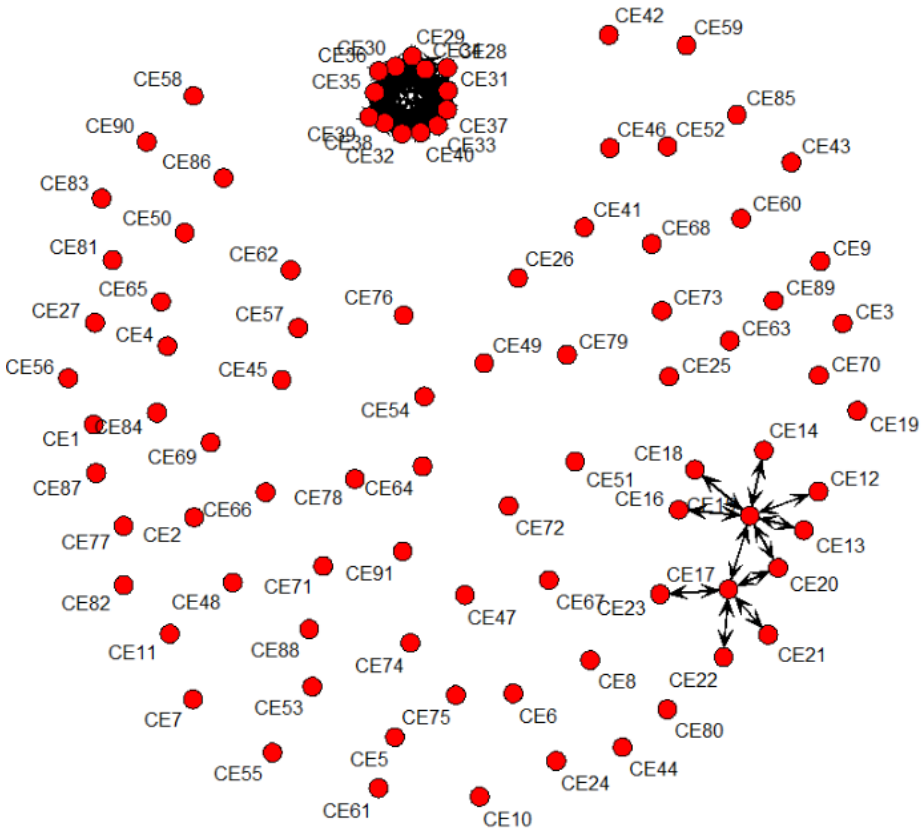
labeled “network_data_workers”. As shown below, 91 vertices remain in the new socio-matrix (compared to 99 in the original socio-matrix). Hence, we have successfully omitted the 8 supervisors from the socio-matrix.

```
Network_data_workers
 Network attributes:
  vertices = 91
  directed = FALSE
  hyper = FALSE
  loops = FALSE
  multiple = FALSE
  bipartite = FALSE
  total edges= 88
    missing edges= 0
    non-missing edges= 88

 Vertex attribute names:
    dep vertex.names

 Edge attribute names:
    closeness
```

Alternatively, we could plot the filtered socio-matrix and inspect if supervisors have been omitted. For this purpose, we can use `gplot`. This results in Figure 5.

```
gplot(Network_data_workers,displaylabels=TRUE)
```

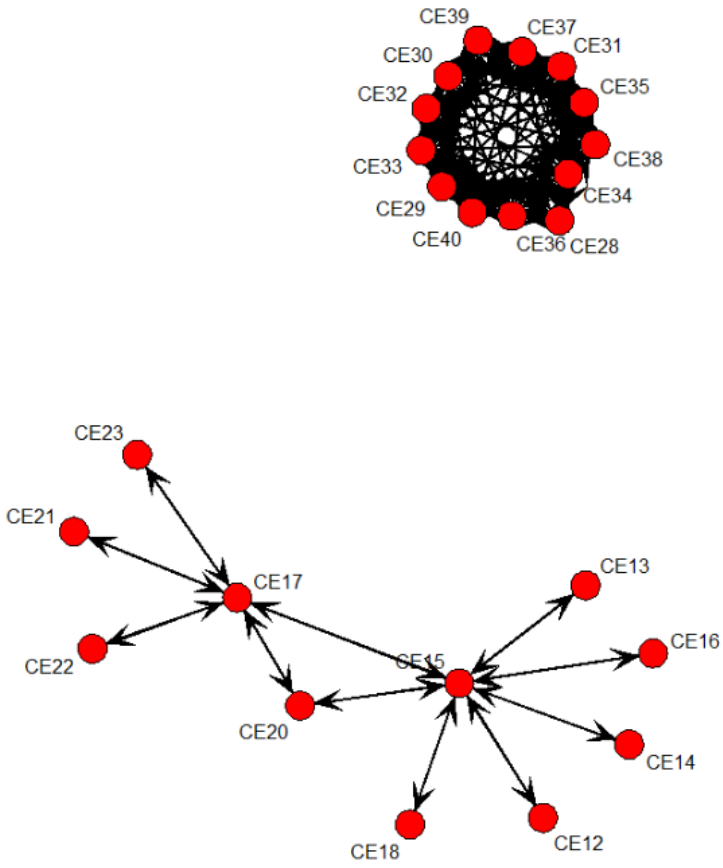
Figure 5 - Plot of filtered socio-matrix



As expected, the supervisors are omitted in Figure 5. Also, there are now a lot of isolates in the organizational network (employees that have no ties). For improving the visualization of the network, we can choose to omit these isolates and then plot the network again. We can use `isolates` to identify isolates and, subsequently, `delete.vertices` to remove the identified isolates from the filtered socio-matrix (see Figure 6):

```
isolates(Network_data_workers)

delete.vertices(Network_data_workers,isolates(Network_data_workers))

gplot(Network_data_workers,displaylabels=TRUE)
```

Figure 6 - Plot of filtered socio-matrix with isolates removed



5.3 Excluding certain types of ties from the network

In other situations, we might be interested in organizational networks that only include ties that meet a certain threshold. For example, we might want to focus on relatively close or strong ties in the company and filter out ties that have values below 3 on the edge attribute "closeness". Such filtering on edge attributes is a little bit more labor intensive, compared to the filtering on vertex attributes. It takes the following steps:

1) We need to go back and reload the edge-attribute file (company_edge_closeness.csv) and replace all values below 3 (<3) with a 0.

2) We then need to create a new socio-matrix from the filtered edge-attribute file. The new socio-matrix file should assign a "1" to ties that score 3 or higher on the edge attribute "closeness". Closeness information should be available for all included ties.
3) We need to reinsert the vertex attributes to the new socio-matrix created in step 2.

For step 1, we can use the code shown below. Do not forget to include `header=F`, because R-studio will otherwise mistakenly add column headers to the edge attribute matrix, which will cause the subsequent analyses to fail. After we loaded the .csv file, we need to replace all values <3 with a 0 in "edge_closeness_filtered" using the code shown below:

```
edge_closeness_filtered<-read.csv("company_edge_closeness.csv", header=F)

edge_closeness_filtered[edge_closeness_filtered<3]<-0
```

In the next step, we load the filtered data as a network using the code we discussed in 4.3.4. Because we now load valued rather than binary network data (i.e., values range from 0 to 5), we have to tell R-studio what to do with the valued data. In our case, the values denote closeness scores and, thus, should be stored as an edge attribute. By specifying `ignore.eval=FALSE` and `names.eval="closeness"`, R-studio will do exactly that and retain the values (rather than converting them into binary data) and save the values as an edge attribute labeled "closeness". As before, we need to specify that the network data is undirected by adding `directed=FALSE` to the code.

```
network_data_close<-
as.network(edge_closeness_filtered,directed=FALSE,ignore.eval=FALSE,names.e
val="closeness")
```

We can check the new network data by requesting summary statistics, simply by typing in the label of the new socio-matrix and pressing CTRL+ENTER. This results in the information shown below. You will notice that the number of vertices has remained the same (99), which is correct, since we did not filter out any vertices. However, the number of edges between these vertices has reduced from 210 to 180 (see below). Apparently, 30 ties were filtered out because their closeness scores were below the threshold of 3.

```
network_data_close
 Network attributes:
  vertices = 99
  directed = FALSE
  hyper = FALSE
  loops = FALSE
  multiple = FALSE
  bipartite = FALSE
  total edges= 180
    missing edges= 0
    non-missing edges= 180

 Vertex attribute names:
    vertex.names

 Edge attribute names:
    closeness
```

5.3.1 Focusing on ego-networks of individual employees

Until now, we focused on the whole organization. In many cases, however, we are not interested in the organizational network as a whole, but want to focus on the network of specific employees (or specific teams, as we will discuss in 5.3.2). To focus on a specific individual, we will need to extract the edges from the larger network that are relevant for understanding the individual under study (i.e., the focal individual or 'node'). We thus need to create a so-called "ego network" for the focal node. An ego network "consists of a focal node ("ego") and the nodes to whom ego has ties with (these are called "alters") plus the ties, if any, among the alters."[3]

To extract an ego network, we need to omit all vertices from the organizational network that have no direct ties with the focal node. Extracting an ego network therefore requires us to do more than simply filtering out certain vertices or edges (as we did in 5.2 and 5.3). To create an ego network, we need to find out which individuals have ties with the focal person and then create a network that only includes these vertices, as well as the focal node.

Let's suppose we want to zoom in on the VP's ego network in our case organization. As a first step, we need to add a vertex attribute to the socio-matrix that indicates which individuals have a tie with the VP. Later on, we can then exclude vertices that have no ties with the VP, and thus have no place in the VP's ego network. Here's the procedure for doing this in R-studio. First, we need to create a new row in the attributes file that indicates if an individual has a tie with the VP or not. Specifically, we need to create a new (empty) row called "ties_with_VP" in the data object "attributes" by entering the code: `attributes$ties_with_VP`. We subsequently need to tell R-studio which values (from which data source) we want to use to populate this new, empty row. In our case, the new row should indicate whether the individual to which the row belongs (please note that each row refers to a different individual in the attributes file) has a tie with the VP (1=yes, 0=no). Information on whether individuals have ties with the VP is captured in the socio-matrix that we labeled "network_data_sociomatrix". To access this data, we need to the refer to the column in the socio-matrix that specifies whether individuals have a tie with the alter "VP". Importantly, that information is located in the column with the name "VP". To refer to this location in the socio-matrix, we use the `[x,y]` format, where the x refer to the rows in the matrix and y refers to columns in the matrix (please see 5.1.2 for details). In our case, we specify `network_data_sociomatrix[,"VP"]` and use the data that is located in this column to populate our new column "ties_with_VP" using the code shown below:

```
attributes$ties_with_VP<-network_data_sociomatrix[,"VP"]
```

After inspecting this new column (by executing the code: `attributes$ties_with_VP`), we learn that the VP has a value of 0 in the new column. This means that the VP will be excluded from his or her own ego network when we would filter based on the "ties_with_VP" attribute. To fix that, we can assign a value of 1 to the VP's cell in the "ties_with_VP" attribute within the attributes data object:

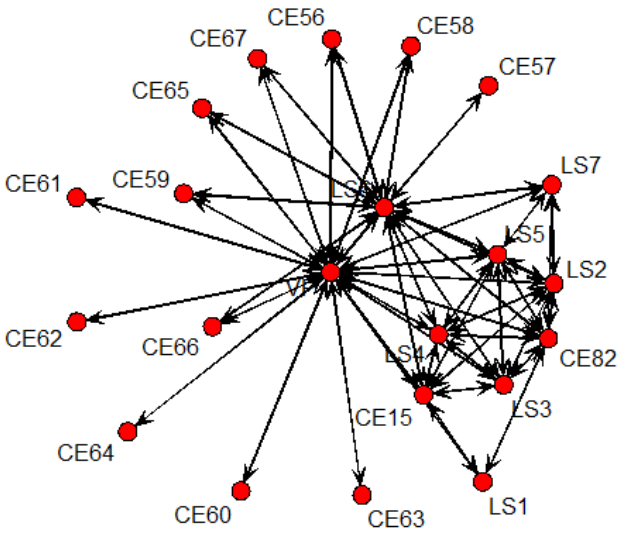
```
attributes$ties_with_VP[attributes$vertex.names=="VP"]<-1
```

[3] http://www.analytictech.com/networks/egonet.htm

Once we added the "ties_to_VP" attribute to the attributes data object, we use `set.vertex.attributes` to attach the updated attributes data object to the socio-matrix (see 4.3.3). Afterwards, we can use `get.inducedSubgraph` to create a new socio-matrix that excludes vertices that have no ties with the VP (see below). We label this socio-matrix "ego_network_VP". In the final step, we plot this ego network using `gplot`. We should then end up with Figure 7.

```
set.vertex.attribute(network_data_sociomatrix,names(attributes),
attributes)

ego_network_VP=get.inducedSubgraph(network_data_sociomatrix,which
(network_data_sociomatrix %v% "ties_with_VP" == "1"))

gplot(ego_network_VP,displaylabels=TRUE)
```

Figure 7 - VP ego network



5.3.2 Focusing on ego-networks of organizational teams

It is also common to examine the network of a group of employees, such as an organizational team or department. In this case, we examine whether team members have ties with each other, as well as how the team members have organized their external ties with other teams in the organization. This allows us to spot problems because members, for example, miss vital ties inside or outside their team. The basic procedure for extracting a team ego network from the organizational network is as following. We first add a new attribute to the attributes data object that lists which individuals (inside or outside the team) have ties with (any) members of the focal team. Second, we add this attribute to the network of the whole organization. Third, we create an "induced subgraph", i.e., a smaller sub network that only includes members that have ties with the focal team. In the final step, we plot and diagnose the team's network by looking at its internal and external connectivity.

Let's practice with calculating the team ego network for "department 9" (i.e., DEP9) in the case organization that we have been analyzing up to this point. First, we need to figure out who works in this department. We can do this be displaying the attribute data object we

loaded before. Execute the code `attributes` and you get the list shown below[4]. Looking at this output, it becomes clear that DEP9 has 12 members, CE1 to C12. In addition, from other company records, we know that DEP9 is supervised by LS2 (see Figure 1). Please note that these members are located in rows 3 & 9-20 of the attributes file. Importantly, the rows and columns of the socio-matrix mimic this order of individuals in the attributes list. Remember from 3.2 that the rows in the socio-matrix indicate the "ego" (i.e., the employees) and the columns of the socio-matrix specify their "alters" (i.e., their partners). An ego has a tie with an alter when the socio-matrix shows a 1 in the cell where the row of the alter meets the column of the alter. Translating this to our case organization, this means that an individual has a tie with DEP9 when he/she has at least one non-zero value in columns 3 & 9-20. These are the individuals we need to include in DEP9's ego network.

```
attributes
   vertex.names dep
1            VP   1
2           LS1   2
3           LS2   3
4           LS3   4
5           LS4   5
6           LS5   6
7           LS6   7
8           LS7   8
9           CE1   9
10          CE2   9
11          CE3   9
12          CE4   9
13          CE5   9
14          CE6   9
15          CE7   9
16          CE8   9
17          CE9   9
18         CE10   9
19         CE11   9
20         CE12   9
...
```

Now that we know how to locate partners (i.e., alters) of DEP9, we can calculate a new attribute that indicates whether an individual has a tie with DEP9. To create this new attribute, we follow the same procedure we discussed in 5.3.1 and first add an empty column to the attributes data object. We label this column "ties_with_team" and use the code `attributes$ties_with_team`. Next, we calculate this attribute as the number of ties an employee has with DEP9 members. To do so, we simply sum up the 1s that are listed in columns 3 & 9-20 for each employee using `rowSums`. The `c(3,9:20)` code allows us to refer to columns 3 and column 9:20 in a single code line (see 5.1.2). We then use that value from the rowSums function to define the new attribute we created earlier:

```
attributes$ties_with_team<-
rowSums((network_data_sociomatrix[,c(3,9:20)]),na.rm = F)
```

[4] See 4.3.3 for information on how to load the attributes file.

Next, we assign a 1 to the new "ties_with_team" attribute for all the team members of DEP9, because we want to make sure that they will be included in the ego network (even when they did not interact with their fellow team members):

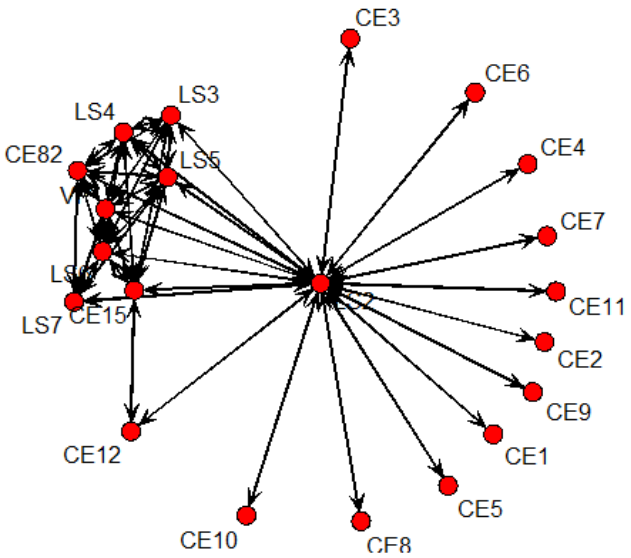
```
attributes$ties_with_team[attributes$vertex.names=="CE1"]<-1
attributes$ties_with_team[attributes$vertex.names=="CE2"]<-1
attributes$ties_with_team[attributes$vertex.names=="CE3"]<-1
attributes$ties_with_team[attributes$vertex.names=="CE4"]<-1
attributes$ties_with_team[attributes$vertex.names=="CE5"]<-1
attributes$ties_with_team[attributes$vertex.names=="CE6"]<-1
attributes$ties_with_team[attributes$vertex.names=="CE7"]<-1
attributes$ties_with_team[attributes$vertex.names=="CE8"]<-1
attributes$ties_with_team[attributes$vertex.names=="CE9"]<-1
attributes$link_with_team[attributes$vertex.names=="CE10"]<-1
attributes$link_with_team[attributes$vertex.names=="CE11"]<-1
attributes$link_with_team[attributes$vertex.names=="CE12"]<-1
```

Subsequently, we attach the updated attributes data object to the socio-matrix (see code below). We can then create the DEP9 ego network by defining a new network that includes only vertices from the organizational network that have a score of at least 1 (>= 1) on the "ties_with_team" attribute. As before, we use `get.inducedSubgraph` for this purpose.

```
set.vertex.attribute(network_data_sociomatrix,names(attributes),
attributes)

ego_network_team<-get.inducedSubgraph(network_data_sociomatrix,which
(network_data_sociomatrix %v% "ties_with_team" >= "1"))

gplot(ego_network_team,displaylabels=TRUE)
```

Finally, we can plot the ego network of DEP9 using `gplot`. The result should look like Figure 8. Based on this figure, we can conclude that DEP9 has a highly centralized network, as all ties are mediated by the supervisor (LS2). Also, all external team ties are mediated by LS2.

Figure 8 - Team ego network plot



6 CALCULATING NETWORK STATISTICS

6.1 Organization-level discriptives

It is good practice to get a 'feel' for the general properties of an organization's overall network, once we have successfully loaded the data in R-studio. There are five network properties that give a good overview of a network: size, density, centralization, fragmentation, and degree of separation. The following sections will explain what these properties represent on a conceptual level and how to calculate these network statistics using R-studio.

6.1.1 Network size

Network size is straightforward; it captures the number of vertices in the socio-matrix. In other words, it indicates how many employees are present within the overall organizational network. To calculate network size, we can use `network.size`, followed by the label of the socio-matrix. This will return the number of vertices in the network. In our case, we find that the network we labeled "network_data_sociomatrix" comprises 99 vertices:

```
network.size(network_data_sociomatrix)
[1] 99
```

Alternatively, we can type in the name of the socio-matrix and execute the code. This will give us overall statistics on the organization's network, including the number of vertices, number of edges, and whether the network is directed or not. Using this approach, we again find that the network labeled "network_data_sociomatrix" comprises of 99 vertices:

```
network_data_sociomatrix
 Network attributes:
  vertices = 99
  directed = FALSE
  hyper = FALSE
  loops = FALSE
  multiple = FALSE
  bipartite = FALSE
  total edges= 210
    missing edges= 0
    non-missing edges= 210

 Vertex attribute names:
    dep ties_with_team ties_with_VP vertex.names

No edge attributes
```

Finally, using the `summary(label of network, print.adh=false)`, we get information on network properties such as size, as well as summary statistics on vertex and edge attributes (if there are any). In our case, we get details on the vertex attributes "dep" (i.e., team affiliation), "ties_with_VP" (whether the person has a tie with the VP), and "ties_with_team" (the number of ties a person has with members of DEP9). The code `print.adj=FALSE` suppresses irrelevant information.

```
summary(network_data_sociomatrix,print.adj = FALSE)
Network attributes:
  vertices = 99
```

```
  directed = FALSE
  hyper = FALSE
  loops = FALSE
  multiple = FALSE
  bipartite = FALSE
 total edges = 210
   missing edges = 0
   non-missing edges = 210
 density = 0.04329004

Vertex attributes:

 dep:
   integer valued attribute
   99 values

 ties_with_team:
   numeric valued attribute
   attribute summary:
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.0000  0.0000  0.0000  0.3333  0.0000 12.0000

 ties_with_VP:
   numeric valued attribute
   attribute summary:
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.0000  0.0000  0.0000  0.2222  0.0000  1.0000
  vertex.names:
   character valued attribute
   99 valid vertex names

Edge attributes:

 closeness:
   integer valued attribute
   210values
```

6.1.2 Network density

Network density indicates the overall connectivity in the organizational network. It is calculated by determining the total number of actual ties in the organizational network and dividing that value by the total number of all possible ties (i.e., if everyone has a tie with everyone). Network density thus reports the percentage of the possible ties that are actually activated within the organization. Network density is reported in the output of `summary` we used above, but it can also be requested using `gden`, followed by the label of the network:

```
gden(network_data_sociomatrix)
[1] 0.04329004
```

We obtain a network density score of .0433, which means that 4.33% of all possible network ties are actually activated in the organization. Based on this information, it is possible to conclude that employees, in general, are not heavily connected with each other. There is a good indication that at least some parts of the organization maintain little to no contact with each other. This provides a good reason for conducting follow-up analyses (e.g., by looking at the ego networks of teams or individuals) to determine which parts that might be.

6.1.3 Network centralization

Network centralization indicates whether there are central individuals in the organization that mediate ties between other individuals. In highly centralized networks, employees do not have many direct ties with each other. Instead, employees connect with each other through a central intermediate person. In highly *de*centralized networks, on the other hand, employees connect directly with each other without going through an intermediary person.

Centralization has advantages and disadvantages. High centralization can enable efficiency and oversight, as employees do not have to spend much of their time on interacting with other individuals (they can just talk to one or a few central persons). Moreover, the central person is in a unique position to develop a 'big picture' of what is going on in the organization and, subsequently, to distribute that knowledge throughout the organization. After all, this person is connected to a large number of employees. On the negative side, the central persons may become overloaded and employees may miss the opportunity to engage in open problem solving and debate with fellow colleagues. Hence, innovation and creativity may decrease in centralized organizations. Whether the benefits outweigh the detriments of centralization depends on the organization's strategy and goals.

We can use `centralization` to calculate the degree to which ties between employees are mediated by (a) central person(s). This will return a value ranging between 0 (the network is highly decentralized; everyone has direct ties with everyone else in the organization) to a theoretical maximum of 1 (all ties are mediated by one single central person).

```
centralization(network_data_sociomatrix,degree)
[1] 0.1953503
```

6.1.4 Network fragmentation

Network fragmentation indicates whether there are isolated subgroups inside the organization. Subgroup members are connected to each, either directly or indirectly (e.g., through a friend of a friend). However, there are no ties between *different* subgroups. An organization with many subgroups is, thus, highly fragmented, as it is comprises several groups of employees that have no ties with each other. This could result in a number of issues, as important individuals and departments might not share information and they might not coordinate important tasks. In addition, the chances of cross-pollination of creative ideas across different groups are slim when there is much fragmentation. High fragmentation, thus, indicates a need for a more in-depth investigation of the organization.

We can calculate fragmentation as the number components by using `components`. This returns the number of subgroups that are present in the organization ('components' is another word for subgroups). In our case organization, there is one component (see below). This means that all employees are directly or indirectly connected with each other.

```
components(network_data_sociomatrix)
[1] 1
```

6.1.5 Degree of separation in the network

The "degree of separation" indicates how many ties employees are apart from each other in the organization. This relates to the famous "six degrees of separation" rule, which suggests that any individual is, on average, only six social ties away from any other individual in the

world. We can compute both the maximum and average degree of separation between individuals in the organization. The maximum degree of separation indicates the shortest path between the two most distal colleagues. For example, a maximum degree of separation score of 1 indicates that all employees are directly connected with each other, as they are separated by one tie at the most (their direct tie) to any other individual. A maximum degree of separation of 10, on the other hand, indicates that at least some individuals need to go through 9 other individuals (connecting to friends of friends of friends) to reach some of their colleagues. Besides the maximum, we can also calculate the average degree of separation. This indicates how many intermediate colleagues an employee, on average, needs to go through in order to connect to a specific other employee in the organization.

There could be problems in the organization if we find a very low or a very high average and maximum degree of separation. When the degree of separation is very low, employees may have so many direct contacts that they may suffer from "collaboration overload". When that happens, individuals spend so much time collaborating with each other that they are left with little time to finish their actual work in the organization. By contrast, when the degree of separation is very high, it could take very long before important information and know-how reaches all individuals in the organization, because such information and know-how has to go through many intermediary contacts. Again, we may need to execute follow-up organizational network analyses to get to the bottom of this.

When the organization network comprises one component (i.e., there is no fragmentation, see 6.1.4), we can directly calculate the degree of separation. To do so, we first create a new matrix that indicates the degree of separation between all possible dyads in the overall organization. This can be done by using `geodist`, followed by the label of the organization's socio-matrix. This will create a column called "gdist" in the data object we defined (here: "degree_of_separation"). Next, we can calculate the maximum and average degree of separation using `mean` and `max`:

```
degree_of_separation<-geodist(network_data_sociomatrix)

mean(degree_of_separation$gdist)
[1] 2.913988

max(degree_of_separation$gdist)
[1] 5
```

When the overall network comprises several components, it is not possible to directly calculate the degree of separation. This is because subgroups are not connected and, thus, impossible for employees to reach all other employees in the organization. Calculating the overall degree of separation then does not make sense. We can, however, calculate the degree of separation within the largest component in the organization:

```
network_largest_component<-
component.largest(network_data_sociomatrix,result="graph")

degree_of_separation_component<-geodist(network_largest_component)

max(degree_of_separation_component$gdist)
[1] 5
```

```
mean(degree_of_separation_component$gdist)
[1] 2.913988
```

Here, the degree of separation of the largest component equals the degree of separation in the overall network, because there is only one component in the overall network.

6.2 Individual-level descriptives

We can also zoom in and calculate network statistics for specific individuals or employees in the overall organization. In the subsequent sections we will discuss how we can calculate meaningful statistics on employees' relative position in the organizational network, the breadth of their networks, and the quality of the their ties.

6.2.1 An individual's network position

Individuals differ in their position in the organizational network, and such differences can have important implications for individuals' career development, promotion chances, and personal effectiveness. To determine an employee's network position, we can calculate 'actor centrality' measures. We will discuss three centrality measures here, namely measures for 'degree, 'closeness', and 'betweenness' centrality. In addition, we will discuss how to determine if individuals occupy "cutpoint" positions in the organizational network.

6.2.1.1 Degree centrality

Degree centrality is the simplest measure for gauging a person's network position; it represents a simple count of the number of ties an individual has within the organization. We can compute degree centrality by using `degree`. To correctly execute this code, we need to specify if the network is directed or undirected (see 3.2). To specify an undirected network, we add `gmode="graph"` to the code. If we specify an undirected network as directed, degree centrality scores are mistakenly doubled (e.g., individuals with 2 ties get degree centrality scores of 4).

The code for calculating the degree centrality scores of individuals in the network labeled "network_data_sociomatrix' is shown below. Based on the outcome from this analysis, we can conclude that the first individual listed in the network has the highest degree centrality (i.e., 21 ties). This makes sense, considering that this is the VP of the organization. The following 7 individuals also have relatively high degree centrality scores. This is also to be expected, because these individuals are the department supervisors. Unexpectedly, however, there are also a number of operational employees with a high degree centrality (e.g., the persons in positions 36-49) with centrality scores of 12 to 13 ties.

```
degree(network_data_sociomatrix,gmode="graph")
[1] 21 12 20 12 20 23 15 19  1  1  1  1  1  1  1  1  1  1  2  1  1 14  1  5  1  1  2  1  1  1  1  1
[34]  1  1 12 13 13 13 13 13 13 13 13 13 13 12 13  1  1  1  1  1  1  1  1  1  1  1  1  1  2  2  2
[67]  2  1  1  1  1  1  2  2  2  1  1  1  1  1  1  1  1  1  1  1  1  1  1  8  1  1  1  1  1  1  1  1
```

6.2.1.2 Closeness centrality

Closeness centrality differs from degree centrality in that it also takes the degree of separation into account. Specifically, with closeness centrality we calculate how close an individual is to all other individuals by looking at the ties of the focal individual, but also by looking at the ties of his or her alters. An individual with ties to 5 highly connected alters may thus still achieve a higher closeness centrality than a person who has ties with 10 alters,

in case all of these alters have marginal and overlapping network positions in the company. We can use `closeness` to calculate and display individuals' scores on this type of centrality:

```
closeness(network_data_sociomatrix,gmode="graph")
 [1] 0.5505618 0.4000000 0.5212766 0.4666667 0.4851485 0.5297297 0.5077720 0.4537037 0.3438596 0.3438596
[11] 0.3438596 0.3438596 0.3438596 0.3438596 0.3438596 0.3438596 0.3438596 0.3438596 0.3438596 0.3698113
[21] 0.3344710 0.3344710 0.5000000 0.3344710 0.3426573 0.3344710 0.3192182 0.3391003 0.2558747 0.2558747
[31] 0.2558747 0.3192182 0.3192182 0.3192182 0.3192182 0.3402778 0.3414634 0.3414634 0.3414634 0.3414634
[41] 0.3414634 0.3414634 0.3414634 0.3414634 0.3414634 0.3414634 0.3402778 0.3414634 0.3475177 0.3475177
[51] 0.3475177 0.3475177 0.3475177 0.3475177 0.3475177 0.3475177 0.3475177 0.3475177 0.3475177 0.3475177
[61] 0.3475177 0.3475177 0.3475177 0.3576642 0.3576642 0.3576642 0.3576642 0.3563636 0.3563636 0.3563636
[71] 0.3563636 0.3563636 0.3576642 0.3576642 0.3576642 0.3130990 0.3130990 0.3130990 0.3130990 0.3130990
[81] 0.3130990 0.3130990 0.3130990 0.3130990 0.3130990 0.3130990 0.3130990 0.3130990 0.3130990 0.4900000
[91] 0.2865497 0.2865497 0.2865497 0.2865497 0.2865497 0.2865497 0.2865497 0.2865497 0.2865497
```

Looking at the results, we see that the first person listed in the network (the VP) has the highest closeness centrality score (.55), followed by the individual in position 3 (.52).

6.2.1.3 Betweenness centrality

Betweenness centrality captures the degree to which an individual mediates ties between other employees. Hence, betweenness centrality indicates which individuals controls the flow of information in the organization, as a high betweenness centrality suggests that a lot of other employees need to go through or pass by this individual to reach other individuals in the organization. We can calculate this using `betweenness`:

```
betweenness(network_data_sociomatrix,gmode="graph")
 [1] 1.155262e+03 8.384286e+02 1.151929e+03 4.764286e+02 1.106512e+03 1.448929e+03 3.719286e+02 1.267000e+03 0.000000e+00
[10] 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
[19] 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 1.025333e+03 0.000000e+00 2.880000e+02 0.000000e+00 0.000000e+00
[28] 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
[37] 8.333333e-02 8.333333e-02 8.333333e-02 8.333333e-02 8.333333e-02 8.333333e-02 8.333333e-02 8.333333e-02 8.333333e-02
[46] 8.333333e-02 0.000000e+00 8.333333e-02 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
[55] 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
[64] 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
[73] 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
[82] 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 2.983333e+02
[91] 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
```

It appears that the individual in position 2 (LS5) has the highest betweenness centrality score (i.e., a score of 1448), even surpassing the VP who had the highest degree and closeness centrality scores.

6.2.1.4 Network cut points

As a final check of a person's network position, we can check what would happen to fragmentation in the overall organization should the focal individual be removed from the network (e.g., because he or she accepts a job elsewhere or calls in sick). To do so, we can use `cutpoints`. Running this code will return a list of IDs of the individuals in the organizations that are identified as cutpoints. These individuals maintain ties between otherwise unconnected groups in the organizations. Correspondingly, should any of these individuals be removed from the network, then the fragmentation would increase in the network (as shown in an increase in the number of components). In our case organization, 9 cutpoints were detected, namely the persons in position 1, 2, 3, 4, 5, 6, 8, 23, and 25.

```
cutpoints(network_data_sociomatrix, mode="graph")
```

```
[1]  1  2  3  4  5  6  8 23 25
```

To check whether the removal of one of these cutpoints indeed results in more fragmentation in the organization, we can use the following procedure. First, we calculate an new subnetwork (using `induced.Subgraph`) that excludes the identified cutpoint. In this case, we will remove the person in position 1 (the VP). Next, we assess the number of components in the organizational including the VP and compare it to the number of cutpoints in the subnetwork. The results indicate that the number of components indeed increase when the VP is excluded from the network. As shown below, the number of components increases from 1 to 6 when the VP is dropped from the organizational network.

```
Network_excluding_VP = get.inducedSubgraph(network_data_sociomatrix,
which(network_data_sociomatrix %v% "vertex.names" != "VP"))

components(network_data_sociomatrix)
[1] 1

components(Network_excluding_VP)
[1] 6
```

6.2.2 An individual's network breadth

The sheer number of ties does not tell the complete story; we also need to consider the diversity of individuals' ties in the company. Individuals with broad networks have networks that span different departments, hierarchical layers, locations, functional work domains, etc.

6.2.2.1 Number of internal and external team ties

To determine network breadth, we can calculate the number of ties an individual has, for example, inside and outside his or her own department. As a first step, we need to calculate how many ties each individual has within each of the organization's departments. Our case organization includes the following departments: TMT (i.e., Top Management Team, which included the VP and all supervisors), DEP9, DEP10, DEP11, DEP12, DEP13, DEP14, and DEP15. We now want to know how many ties each individual in the organization has with members from the TMT, DEP 9, DEP10, DEP11, etc.

To calculate this, we add 8 new rows to the attributes data object, which we call "ties_with_TMT" to "ties_with_DEP15". We then populate these rows with the number of ties an individual has within the respective department. For example, "ties_with_TMT" should note how many ties a person has with TMT members. For this purpose, we will use `rowSums` and calculate for each row the sum of the columns that correspond to the alters within the different teams. Please remember from 3.2 that rows represent the 'egos' in a socio-matrix and the columns represent the 'alters', so by calculating the row sum for specific columns, we actually calculate egos' sum of ties with certain alters (i.e., that correspond to the specified column numbers). By inspecting the attributes data object, we know that the VP and LS1-7 (as alters) are positioned in columns 1:9. These columns, thus specify if an individual has a tie with TMT members or not (0=no, 1=yes). Correspondingly, by taking the sum of columns 1:9 for each row, we get a value that indicates how many ties a person has with TMT members. We then repeat this for the other departments as following:

```
attributes$ties_with_TMT<-rowSums((network_data_sociomatrix[,1:9]),na.rm = F)
attributes$ties_with_DEP9<-rowSums((network_data_sociomatrix[,9:20]),na.rm = F)
attributes$ties_with_DEP10<-rowSums((network_data_sociomatrix[,21:35]),na.rm = F)
```

```
attributes$ties_with_DEP11<-rowSums((network_data_sociomatrix[,36:48]),na.rm = F)
attributes$ties_with_DEP12<-rowSums((network_data_sociomatrix[,49:63]),na.rm = F)
attributes$ties_with_DEP13<-rowSums((network_data_sociomatrix[,64:75]),na.rm = F)
attributes$ties_with_DEP14<-rowSums((network_data_sociomatrix[,76:89]),na.rm = F)
attributes$ties_with_DEP15<-rowSums((network_data_sociomatrix[,90:99]),na.rm = F)
```

Based on the newly calculated attributes, we can then calculate each individual's total ties within the organization, again using `rowSums`:

```
attributes$total_ties<-rowSums(attributes[,c("ties_with_TMT",
                                             "ties_with_DEP9",
                                             "ties_with_DEP10",
                                             "ties_with_DEP11",
                                             "ties_with_DEP12",
                                             "ties_with_DEP13",
                                             "ties_with_DEP14",
                                             "ties_with_DEP15")])
```

Next, we calculate individuals' ties with direct colleagues inside their own department. We use `ifelse` for this purpose and specify that individuals' score on "attributes$internal_ties" should be derived from the row "attributes$DEP9" if the individual works in DEP9, from the row "attributes$DEP10" if the individual works in DEP10, etc.:

```
attributes$internal_ties<-ifelse(attributes$dep<9,attributes$ties_with_TMT,
                                 ifelse(attributes$dep == 9,attributes$ties_with_DEP9,
                                 ifelse(attributes$dep == 10,attributes$ties_with_DEP10,
                                 ifelse(attributes$dep == 11,attributes$ties_with_DEP11,
                                 ifelse(attributes$dep == 12,attributes$ties_with_DEP12,
                                 ifelse(attributes$dep == 13,attributes$ties_with_DEP13,
                                 ifelse(attributes$dep == 14,attributes$ties_with_DEP14,
                                 ifelse(attributes$dep == 15,attributes$ties_with_DEP15,
                               NA)))))))
```

Finally, we calculate the number of external_ties (i.e., ties between the individual and alters from other departments) as a function of the individual's total ties and internal ties:

```
attributes$external_ties<-(attributes$total_ties-attributes$internal_ties)
```

Once we complete the calculation of the internal, external, and total ties of all individuals in the network, we can print a list from the attributes data object with each individual's score:

```
attributes[,c("vertex.names","internal_ties","external_ties","total_ties")]
   vertex.names internal_ties external_ties total_ties
1            VP             7            14         21
2           LS1             1            11         12
3           LS2             7            14         21
4           LS3             5             7         12
5           LS4             5            15         20
6           LS5             6            17         23
7           LS6             6             9         15
8           LS7             4            15         19
9           CE1             0             1          1
10          CE2             0             1          1
...
```

In addition, we can calculate the minimum, maximum, and average scores across individuals on the new attribute variables. This allows us to compare a specific individual's score to the average and determine his or her network position against that of the average employee:

```
summary(attributes$total_ties)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.000   1.000   1.000   4.253   2.000  23.000

summary(attributes$internal_ties)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  0.000   0.000   0.000   2.172   1.000  12.000

summary(attributes$external_ties)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  0.000   1.000   1.000   2.081   1.000  17.000
```

6.2.2.2 Diversity of ties across different groups

Beyond calculating the raw number of ties inside and outside an individual's department (or whatever kind of group), we can also calculate how an individual's ties are dispersed across groups. We can do this with a diversity index, such as the Blau index. This diversity index calculates a value that ranges between 0 and 1. A value of 0 indicates that an individual's ties are all concentrated in one single group. When the value approaches the theoretical maximum of 1, an individual's ties are equally distributed across all groups that are present in the organization. The mathematical formula for Blau's diversity index is as following:

$$1 - \sum_{i=1}^{k} p_i^2$$

where p_i is the percentage of total ties with the ith group in the organization and k represents the total number of groups.

In our case organization, we can determine individuals' tie diversity by calculating whether their ties are equally distributed across all 8 departments in the organization. We can use the "total_ties" and "ties_with_TMT" to "ties_with_DEP15" rows from the attributes data object for this. When implementing Blau's diversity index in R-studio, we should end up with the code shown below. This code adds a new row to the attributes data object, called "tie_diversity".

```
attributes$tie_diversity=1-(
     (attributes$ties_with_TMT/attributes$total_ties)^2+
     (attributes$ties_with_DEP9/attributes$total_ties)^2+
     (attributes$ties_with_DEP10/attributes$total_ties)^2+
     (attributes$ties_with_DEP11/attributes$total_ties)^2+
     (attributes$ties_with_DEP12/attributes$total_ties)^2+
     (attributes$ties_with_DEP13/attributes$total_ties)^2+
     (attributes$ties_with_DEP14/attributes$total_ties)^2+
     (attributes$ties_with_DEP15/attributes$total_ties)^2)
```

When we inspect our newly calculated tie diversity attribute, we can see that a lot of individuals have a score of 0. This means that they only have ties within one department (probably their own department). A few individuals, on the other hand, do have high tie

diversity. The individual labeled “LS6” has the most diverse collection of ties, with a tie diversity score of .61 (see below).

```
attributes[,c("vertex.names", "tie_diversity")]
   vertex.names tie_diversity
1            VP     0.5578231
2           LS1     0.2916667
3           LS2     0.5578231
4           LS3     0.5694444
5           LS4     0.5100000
6           LS5     0.5028355
7           LS6     0.6133333
8           LS7     0.4099723
9           CE1     0.0000000
10          CE2     0.0000000
11          CE3     0.0000000
12          CE4     0.0000000
13          CE5     0.0000000
14          CE6     0.0000000
15          CE7     0.0000000
16          CE8     0.0000000
17          CE9     0.0000000
18         CE10     0.0000000
19         CE11     0.0000000
20         CE12     0.5000000
...
```

6.2.3 An individual’s network quality

Network quality indicates whether an individual has deep and meaningful ties with his or her alters, as opposed to shallow and superficial ties. To determine individuals’ network quality, we need information on the quality of ties in the organizational network. For example, we could obtain information on how effective, productive, or valuable individuals in the network perceive their ties with their alters in the organization. Subsequently, we can add this information to the socio-matrix as “edge attributes”. Based on this information, we can then calculate how an individual is evaluated by his or her alters in terms of quality.

In our case organization, we asked individual employees to indicate how closely they work with each of their alters on a scale ranging from 1 (not close at all) to 5 (very close). With this information we can calculate how close an individual is, on average, with his or her alters. A high average closeness value indicates that the individual has built deep and meaningful ties with his or her alters. The information on tie closeness is stored in the file “company_edge_closeness.csv”. To add this information as an edge attribute to the socio-matrix, we can use the code below. This will first load the tie closeness information as a data object called “edge”. Please note that we load this data using `as.matrix` with `header=F` to make sure that the matrix with edge information is completely square and that the number of rows and columns correspond to that of the socio-matrix. In the next line of code, we add the edge information to the socio-matrix “network_data_sociomatrix” as an edge attribute called “closeness”. For this purpose, we use `set.edge.value` (see 4.3.4).

```
edge <- as.matrix(read.csv("company_edge_closeness.csv", header=F))

network_data_sociomatrix<-
set.edge.value(network_data_sociomatrix,'closeness',edge)
```

Next, we calculate the sum of all closeness scores for each row, using `rowSums`. Remember, each row represents one 'ego' in the organizational network, so the row sum for one particular row represents the overall score of one employee. For example, the row sum of the first row corresponds to the overall closeness score of the VP of the organization. To get to an average closeness score, we divide the overall closeness score by the individual's total number of ties (which we calculated in 6.2.2.1 as "attributes$total_ties"). We add the result to our attributes data object as a new row labeled "avg_closeness":

```
attributes$avg_closeness<-
rowSums(as.sociomatrix(network_data_sociomatrix,attrname="closeness")[,])/a
ttributes$total_ties
```

We can then use the newly calculated avg_closeness row in the attributes data object to discover individuals' network quality. In addition, we can compare individuals' scores to the mean average closeness score within the organization using `summary`:

```
summary(attributes$avg_closeness)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.000   2.692   3.077   3.313   4.000   5.000

attributes[,c("vertex.names","avg_closeness")]
   vertex.names avg_closeness
1            VP      3.238095
2           LS1      3.666667
3           LS2      3.095238
4           LS3      2.416667
5           LS4      3.000000
6           LS5      3.086957
7           LS6      3.000000
8           LS7      3.842105
9           CE1      3.000000
10          CE2      2.000000
11          CE3      1.000000
12          CE4      2.000000
13          CE5      4.000000
14          CE6      4.000000
15          CE7      5.000000
16          CE8      4.000000
17          CE9      3.000000
18         CE10      2.000000
19         CE11      3.000000
20         CE12      3.000000
...
```

6.2.4 Reporting on individual-level information

6.2.4.1 Individual-level summary statistics

For evaluation purposes, it is good practice to assess a specific person's network statistics against the average scores within the overall organization. We can, for example, determine the average degree, closeness, and betweenness centrality in the organization, and then use those numbers to assess whether a specific individual's connectivity in the organization is above or below average. We use `summary` to calculate organizational level statistics on vertex attributes, as shown below. This will not only provide the organizational average, but also the percentile, minimum, maximum, and median scores.

```
summary(degree(network_data_sociomatrix,gmode="graph"))
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.000   1.000   1.000   4.242   2.000  23.000

summary(degree(network_data_sociomatrix,gmode="graph"))
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.000   1.000   1.000   4.242   2.000  23.000

summary(closeness(network_data_sociomatrix,gmode="graph"))
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.2559  0.3131  0.3427  0.3468  0.3475  0.5506

summary(betweenness(network_data_sociomatrix,gmode="graph"))
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   0.00    0.00    0.00   95.24    0.00 1448.93
```

6.2.4.2 Exporting Individual-level information

Once we calculated individuals' centrality measures, we can store this information alongside the other attributes that we already have on individuals (e.g., department affiliation). To do so, we add new columns to the attributes data object we conveniently labeled "attributes" (i.e., the data object that already lists information on employees' department affiliation, their code, etc.). Later on, we can export this data object as a .csv file for future reference. Alternatively, we can load the new information on individuals' centrality as a vertex attribute and use it filter the data based on individuals' degree centrality (see 5.2). Below is an example of how we can add new rows with information on individuals' degree, closeness, and betweenness centrality to the attributes data object "attributes". After executing this code, we can type in the label of the data object to see individuals' raw scores:

```
attributes$degree<-degree(network_data_sociomatrix,gmode="graph")

attributes$closeness<-closeness(network_data_sociomatrix,gmode="graph")

attributes$betweenness<-betweenness(network_data_sociomatrix,gmode="graph")

attributes
   vertex.names dep degree closeness  betweenness
1            VP   1     21 0.5505618 1.155262e+03
2           LS1   2     12 0.4000000 8.384286e+02
3           LS2   3     20 0.5212766 1.151929e+03
4           LS3   4     12 0.4666667 4.764286e+02
5           LS4   5     20 0.4851485 1.106512e+03
6           LS5   6     23 0.5297297 1.448929e+03
7           LS6   7     15 0.5077720 3.719286e+02
...
```

Once these attributes have been added to the attributes data object, we can add this information as vertex attributes, as we have done in 4.3.3. We can also export attributes as a .csv file using `write.csv`. This will save the .csv file in the working directory.

```
write.csv(attributes, file="Attributes_vertices.csv", row.names = TRUE)
```

Please note that the .csv format uses comma to delimitate different columns in the table. To access this data, we can import it in Microsoft Excel by going to the "Data" menu tab and then selecting "From text". Subsequently, select the.csv file on your computer and import it as a comma delimitated file.

6.3 Department-level/team-level descriptives

Finally, we might be interested in calculating networks statistics for teams or departments. Based on this information, we can diagnose the networks of departments or teams and, if necessary, develop interventions to enhance team or departmental effectiveness.

When evaluating departments or teams using ONA, we have to distinguish between the internal and external network of the department or team. The internal network comprises all connections between members of the respective department or team. Hence, the internal network indicates how employees of the *same* team or department connect with each other to complete work. The external network, on the other hand, comprises all connections between members of the respective department or team with members from *other* departments or teams. Hence, the external network reflects how the team or department works with other groups in the organization to complete organization-wide tasks and to exchange information across boundaries.

In the following sections, we will calculate department-level network statistics for the departments in our case organization. Before we do this, we will recode the department identifier in the attributes file so that all managers belong to team 1 (i.e., the top management team). We label the new identifier "dep_recoded" and add it to the attributes:

```
attributes$dep_recoded<-attributes$dep
attributes$dep_recoded[attributes$dep_recoded<9]<-1
```

6.3.1 Internal team and departmental networks

6.3.1.1 Density in departments' internal networks

Density of departments' internal network is conceptually the same as density within the overall organization (see 6.1.2), but focused on the connectivity *inside* the departments. The internal network density thus only looks considers ties that exist within the departments under study, thereby ignoring any between-department ties. There is no direct command in R-Studio to quickly calculate the internal network density. Instead we have to manually calculate the density of departments' internal networks by using the internal group density formula:

$$Internal\ group\ density = \frac{\sum_{i=1}^{g} C_{DM}(n_i)}{(g^s(g^s-1))},$$

where the numerator represents the sum of internal ties of all individual department members g (denoted as $C_{DM}(n_i)$), and the denominator represents the theoretical maximum number of ties that can emerge within the department (i.e., when all department members have ties with each other) in a department with a total of g^s members. Each department member can, in theory, maintain ties with anyone expect him or herself. Hence, we calculate the maximum number of ties of an individual department member as $g^s - 1$ (i.e., the department size, excluding the focal department member). If we then multiply this by the number of members that work in the department (i.e., $g^s(g^s-1)$), we end up with the theoretical maximum number of ties that can emerge in the department as a whole.

To implement this formula in R-Studio, we need the `dplyr` package. This package provides a number of data manipulation options and can be used summarize individual-level data by group membership. As such, we can use this package to calculate department-level statistics, such as group density. The first step is to install and load this package:

```
install.packages("dplyr")
library("dplyr")
```

In dplyr, we can use `group_by`, in combination with `summarize`, to specify that we want to calculate group-level statistics (or in our case, department-level statistics). We can calculate the average (`mean`), maximum value (`max`), minimum value (`min`), median value (`median`), standard deviation (`sd`), and variance (`var`) of the scores of employees that work in the same department, as identified by the grouping variable. In addition, we can count the number of observations (i.e., employees) that share the same identifier value using `n()`.

In the present case, we use `dplyr` to implement the internal group density formula and, subsequently, to calculate a new data table that is labelled "internal_group_density":

```
internal_group_density<-attributes %>%
  group_by(dep_recoded) %>%
  summarize(internal_group_density=(sum(internal_ties)/((n()*(n()-1)))))
```

This code produces a table that comprises 8 rows (i.e., one row per department) and 2 columns. The first column specifies the department number and lists the values of our grouping variable ("dep_recoded"). The second column is called "internal_group_density" and is calculated as the sum of all internal ties within a department, divided by theoretical maximum number of ties within the department. Once we calculated these values, we convert the table to a data frame using `as.data.frame` and then display it in R-Studio.

```
internal_group_density<-as.data.frame(internal_group_density)
internal_group_density
  dep_recoded internal_group_density
1           1              0.7321429
2           9              0.0000000
3          10              0.0952381
4          11              0.9871795
5          12              0.0000000
6          13              0.0000000
7          14              0.0000000
8          15              0.0000000
```

Looking at the table, it is apparent that DEP11 has by far the highest internal density score, followed by the TMT and DEP10. The remaining departments have an internal group density score of 0, which makes sense because there are no internal ties within these departments.

6.3.1.2 Centralization in departments' internal network

In the next step, we will calculate whether the ties inside the departments are mediated by some central department members (i.e., internal group centralization). This can be calculated using the "Group Degree Centralization" formula:

$$Internal\ group\ centralization = \frac{\sum_{i=1}^{g}[C_{DM}(n^*) - C_{DM}(n_i)]}{g^s(g^s - 1)},$$

where the numerator represents the sum of the difference between the number of internal ties of each team member g (denoted as $C_{DM}(n_i)$) and the number of internal ties of the department member who maintains the most ties within the department (denoted as $C_{DM}(n^*)$). The denominator reflects the theoretical maximum number of ties within the team, similar to what we used to calculate internal group density.

An internal group centralization score can range from 0 to 1. A score near the scale's maximum value of 1 suggests that a single department member has a disproportionately larger number of internal ties relative to other department members. This means that this central department member takes the lead in coordinating work, exchanging information, etc. within the department (i.e., high internal group centralization). Conversely, a score near the theoretical minimum of 0 indicates that all department members maintain an equal number of internal ties and, thus, hold equal roles in the coordination of tasks, exchange of information, etc. within the department (i.e., low internal group centralization).

To determine internal group centralization, we first calculate the difference between each employee and the most-connected colleague within his or her department. To do that, we use `dplyr` to add a variable that is called "internal_ties_diff" to our vertex attributes file. We calculate this variable by subtracting the value listed in the column "internal_ties" from the maximum value observed within the respective individual's department. Afterwards, we convert the attributes to a data frame using `as.data.frame`. `Dplyr` automatically converts the file to a table format, which could cause problems later on when left unchanged.

```
attributes<-attributes %>% group_by(dep_recoded) %>%
  mutate(internal_ties_diff = max(internal_ties)-internal_ties)

attributes<-as.data.frame(attributes)
```

We can subsequently calculate internal group centralization by taking the group-level sum of the "internal_ties_diff" column and dividing this value by the theoretical maximum number of possible internal ties within the department:

```
internal_group_centralization<-attributes %>%
  group_by(dep_recoded) %>%

summarize(internal_group_centralization=sum(internal_ties_diff)/(n()*(n()-
1)))
```

In the final step, we convert the resulting internal group centralization scores to a data frame and then combine it with the data on group internal density using `join`. We can then display both the departments' density and centralization scores by referring to the label of the combined data frame (i.e., "dep_statistics_internal" in our case):

```
internal_group_centralization<-as.data.frame(internal_group_centralization)

dep_statistics_internal<-left_join(internal_group_density,
internal_group_centralization,by="dep_recoded")
```

```
dep_statistics_internal
  dep_recoded internal_group_density internal_group_centralization
1           1               0.7321429                     0.26785714
2           9               0.0000000                     0.00000000
3          10               0.0952381                     0.33333333
4          11               0.9871795                     0.01282051
5          12               0.0000000                     0.00000000
6          13               0.0000000                     0.00000000
7          14               0.0000000                     0.00000000
8          15               0.0000000                     0.00000000
```

Based on our results, we can conclude that both the TMT (dep_recoded = 1) and DEP10 have fairly centralized internal networks. That means that there are probably one or a few members with a lot of ties to other members in those departments, while other members in these departments do not maintain a lot of ties with each other. In other words, the flow of information is heavily controlled by a few central department members.

6.3.2 External team and departmental networks

6.3.2.1 Density in departments' external networks

Density in a department's external network indicates how well the department is connected to other departments in the organizations. The procedure for calculating such external group density is largely similar to that for internal group density, as we discussed in 6.3.1.1. The difference is that external group density is calculated based on *external ties*. For calculating the external group density, we calculate how many external ties the department's members maintain in total, using `sum`. Next, we divide this by the theoretical maximum number of external ties that department's members can maintained, given the size of the organization.

Importantly, the theoretical maximum is calculated differently for external group density, as compared with internal group density. For determining the maximum number of external ties, we take the size of the organization and subtract the size of the respective department. We end up with the number of potential external ties that the respective department's members, in theory, can maintain. Our case organization has 99 employees. Hence, each individual member of the TMT can, at the most, maintain (99-9 [i.e., the size of the TMT]) 90 external ties. We then multiply this number by the size of the respective department to arrive at the theoretical maximum number of external ties that can be maintained by the focal department as a whole (90*9 in our case). Note that we do not subtract 1 from this value (as we did for internal density). The code below illustrates how to do this in R-Studio:

```
external_group_density<-attributes %>%
   group_by(dep_recoded) %>%
   summarize(external_group_density=(sum(external_ties)/((n()*(99-n())))))
```

By executing the above stated code, we create a data table that lists the external group density score of each department in our case organization. We need to convert this table to a data frame, to avoid problems later on when we merge that data with other data. Subsequently, we can display departments' values by referring to the label of the data frame and pressing CTRL + ENTER. The results below indicate that the TMT has the highest external group density, compared to the other departments in the case organization.

```
external_group_density<-as.data.frame(external_group_density)
external_group_density
  dep_recoded external_group_density
1           1             0.14010989
2           9             0.01245211
3          10             0.01031746
4          11             0.01162791
5          12             0.01190476
6          13             0.01819923
7          14             0.01176471
8          15             0.01910112
```

6.3.2.2 Centralization in departments' external network

External group centralization indicates whether a department's external ties are held by one of multiple members of the respective department. A high centralization score indicates that all external connections are maintained by a single department member. Hence, this member is over-connected to other departments, while his or her colleagues inside the department are under-connected as they maintain no or only very few ties with outside members. A low external group centralization score, on the other hand, indicates that the external ties are equally distributed among members of the respective department.

We can calculate external group centralization by using a slightly modified version of the procedure we used for calculating internal group centralization (see 6.3.1.2). For calculating external group centralization, we need to modify the formula so that it calculates the difference between each department member's external ties and the member in the department that holds most external ties:

```
attributes<-attributes %>% group_by(dep_recoded) %>%
  mutate(external_ties_diff = max(external_ties)-external_ties)

attributes<-as.data.frame(attributes)
```

In the next step, we calculate for each department the sum of its members' difference scores and divide that by the theoretical maximum number of external ties that the respective department can maintain. The theoretical maximum is calculate as before (see 6.3.2.1):

```
external_group_centralization<-attributes %>%
   group_by(dep_recoded) %>%
   summarize(external_group_centralization=sum(external_ties_diff)/((n()*
   (99-n()))))
```

In the final step, we convert our result to a data frame and display the results. Based on the results, we can see that DEP10 scores highest in terms of external group centralization.

```
external_group_centralization<-as.data.frame(external_group_centralization)
external_group_centralization
  dep_recoded external_group_centralization
1           1                   0.046703297
2           9                   0.010536398
3          10                   0.084920635
4          11                   0.000000000
5          12                   0.000000000
6          13                   0.004789272
```

```
7           14                    0.000000000
8           15                    0.070786517
```

As a final step, we can merge the data frames with information on departments' internal and external networks using `join`. This file can come in handy when examining what kind of combinations of internal and external network structures departments use to execute work.

```
dep_statistics_internal<-left_join(internal_group_density,
internal_group_centralization,by="dep_recoded")

dep_statistics_external<-left_join(external_group_density,
external_group_centralization, by="dep_recoded")

dep_statistics<-left_join(dep_statistics_internal,dep_statistics_external,
by="dep_recoded")

dep_statistics
  dep_recoded internal_group_density internal_group_centralization
1           1              0.7321429                    0.26785714
2           9              0.0000000                    0.00000000
3          10              0.0952381                    0.33333333
4          11              0.9871795                    0.01282051
5          12              0.0000000                    0.00000000
6          13              0.0000000                    0.00000000
7          14              0.0000000                    0.00000000
8          15              0.0000000                    0.00000000
  external_group_density external_group_centralization
1             0.14010989                    0.046703297
2             0.01245211                    0.010536398
3             0.01031746                    0.084920635
4             0.01162791                    0.000000000
5             0.01190476                    0.000000000
6             0.01819923                    0.004789272
7             0.01176471                    0.000000000
8             0.01910112                    0.070786517
```

7 PLOTTING ORGANIZATIONAL NETWORKS

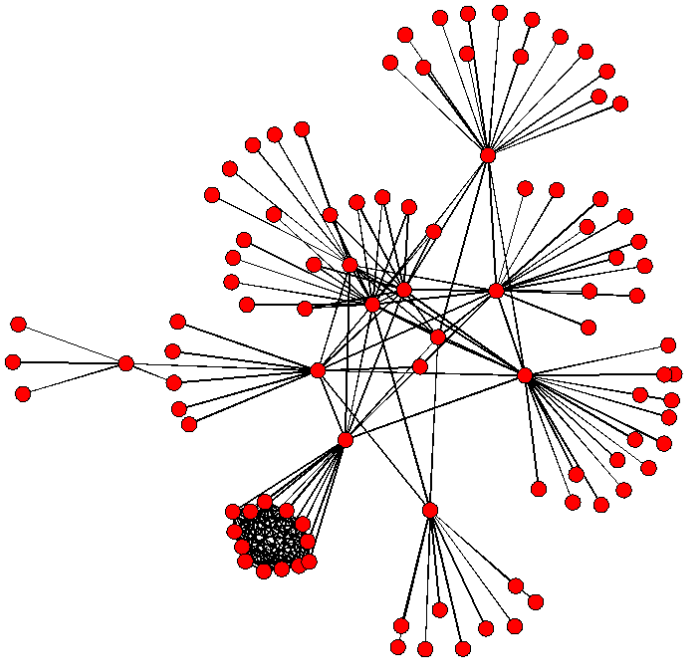
To visualize an organizational network, we can make a network plot. A network plot is collection of dots and lines that provide a schematic overview of who has ties with whom in the organization. The dots represent the nodes in the network, so organizational members in our organization, and the lines represent the presence of a ties between nodes. In case there is a line between two nodes, this indicates that the two nodes have a tie with each other. Below, we will discuss how to create network plots with varying layouts.

7.1 A basic network plot

Basic network plots can be created with the `gplot`.[5] The code for plotting a basic network using `gplot` is shown below. This code will produce the network plot that is shown in Figure 9. The `usearrows = FALSE` part of the code indicates that the network is undirected and, thus, that there is no need for drawing lines (i.e., ties) as arrows.

```
gplot(network_data_sociomatrix,usearrows=FALSE)
```

Figure 9 - Standard network plot



This standard plot does not give much information, except for a display of the overall structure of the organizational network. From the plot, we can already assess that the network is relative centralized with a few nodes controlling much of the information flow in the organization. However, we do not know who the central nodes are because the network plot excludes node labels. Hence, the next step is to add node labels.

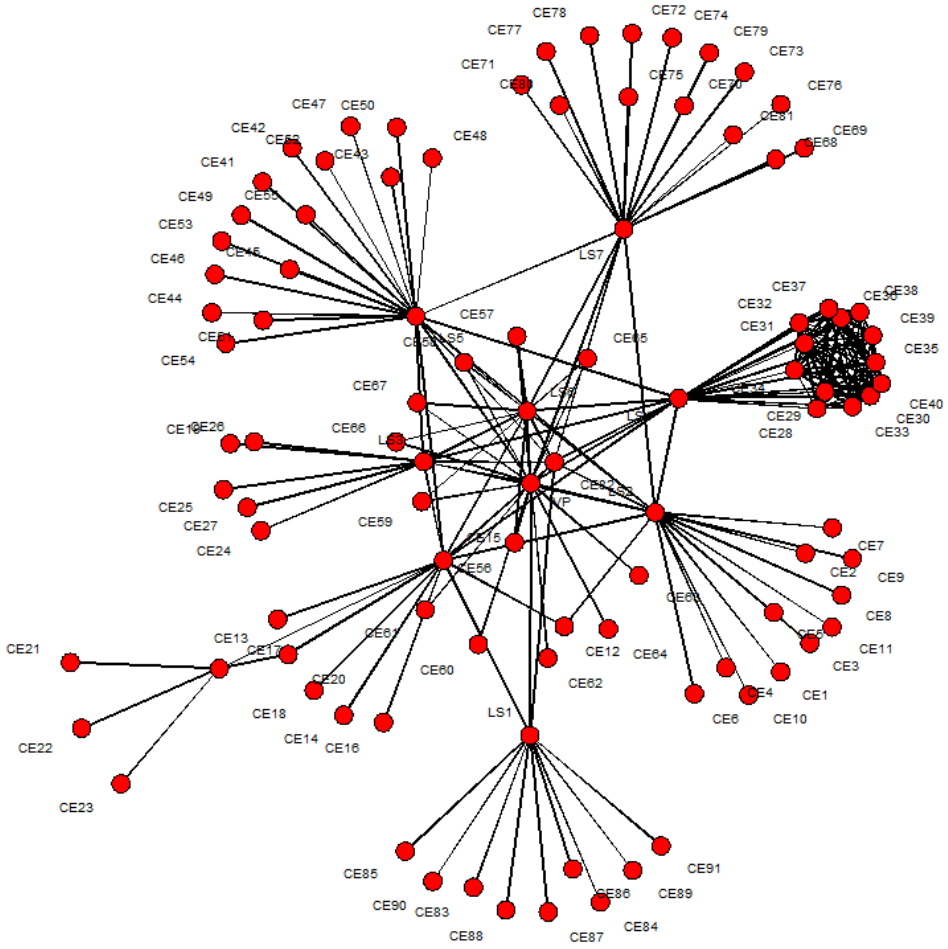
If labels are stored as a vertex attribute called "vertex.names", we can simply add `displayLabels=T` to the code and R-studio will use vertex.names as node labels, as shown

[5] Be sure to load both the "sna" and "statnet" package, as this might otherwise cause errors when using `gplot`.

in Figure 10. We can further change the size of the nodes by specifying `vertex.cex`, as well as the size of the labels by specifying `label.cex`. In the code shown below, we specify that the labels should be rescaled by a factor of 0.60, while nodes are rescaled by a factor of 0.90.

```
gplot(network_data_sociomatrix,displaylabels=T,
      label.cex=0.60,
      vertex.cex=0.90,
      usearrows=FALSE)
```

Figure 10 - Basic network plot with node labels



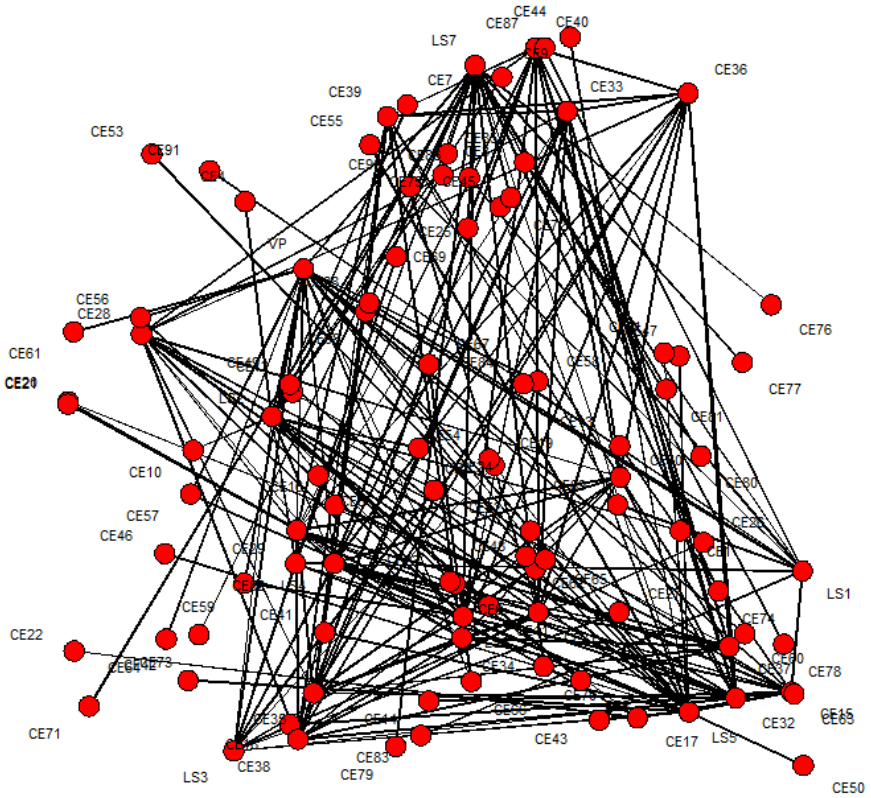
If we do not want to use vertex.names for node labels, we need to define a new data object that provides alternative node labels. Next, we need to add a piece of code to `gplot` to use this data object for the node labels. For example, if we want to display employees' department affiliation as node labels (instead of vertex.names), we first define a data object called "dep" that is derived from the vertex attribute "dep". Subsequently, we refer to this data object by adding `label=dep` to `gplot`. This will display nodes' department numbers as node labels.

```
dep <-get.vertex.attribute(network_data_sociomatrix,"dep")

gplot(network_data_sociomatrix,label=dep,displaylabels=T,label.cex=0.70,use
arrows=FALSE)
```

Besides adding node labels, we can also change the way in which nodes are positioned in the network plot. There are several options available for this in `gplot`. We can choose for a random layout in which nodes are placed randomly in the network plot. This is done by adding `mode= "random"` to the `gplot` code:

```
gplot(network_data_sociomatrix,displaylabels=T,label.cex=0.70,usearrows=FALSE,mode="random")
```

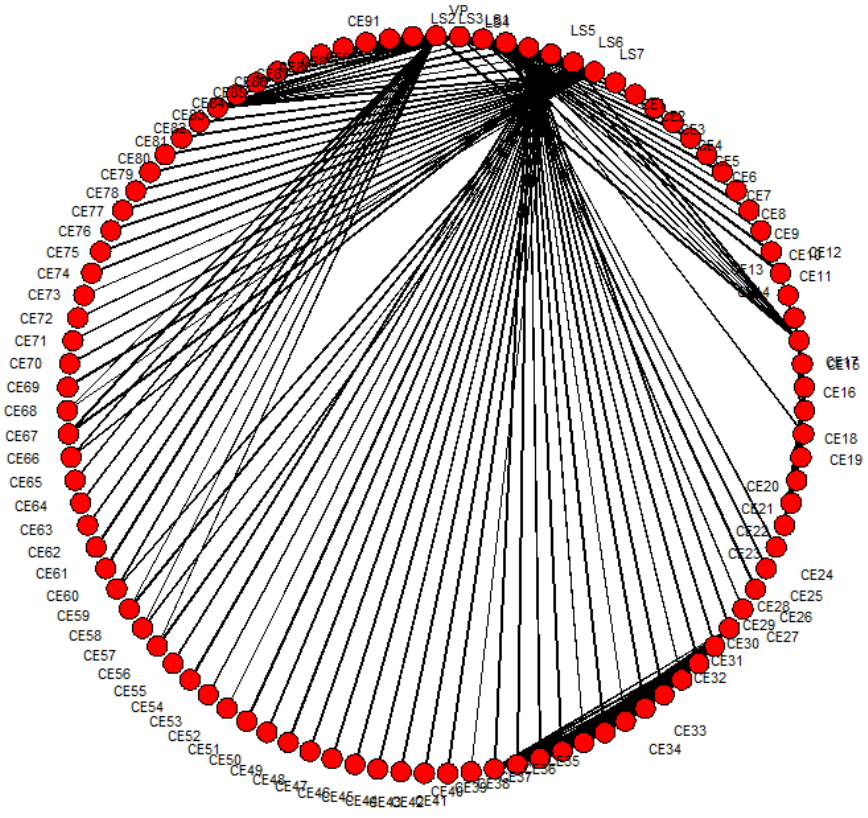
Figure 11 - Network plot with random positioning of nodes



For obvious reasons, the random positioning does not produce the most interpretable network plot (see Figure 11). Indeed, to maximize interpretability, we need to prevent that nodes are caught up in-between a lot of ties. When nodes are in the middle of ties, their labels will not be readable and the overall network plot may look sloppy. To resolve this, we can consider positioning nodes in a big circle. This will prevent that nodes are surrounded by the lines of ties (Figure 12). Here's how to implement such as circle-positioning design:

```
gplot(network_data_sociomatrix,displaylabels=T,label.cex=0.70,usearrows=FALSE,mode="circle")
```

Figure 12 - Network plot with circular positioning of nodes



The circular positioning improves the interpretation somewhat, but it is still not optimal. There are still a lot of line crossings and lines differ in length. To resolve this, we need to place nodes together that are densely connected to each other together, while positioning nodes that are not connected (directly or indirectly) further apart in the network plot. One way to do this, is by using the "Fruchterman-Reingold" algorithm. This algorithm works as a force field in which nodes gravitate towards each other to the degree to which they are directly and indirectly connected with each other. When applying this positioning algorithm to a network plot, we end up with a tailor-made layout that is specific to the organization under study. To use the Fruchterman-Reingold, we add `mode= "fruchtermanreingold"` to `gplot`. This is, however, already standardly done in statnet.

```
gplot(network_data_sociomatrix,displaylabels=T,label.cex=0.70,usearrows=FALSE,mode="fruchtermanreingold")
```

7.2 Modifying node and tie layout in a network plot

7.2.1 Varying nodes sizes in network plots

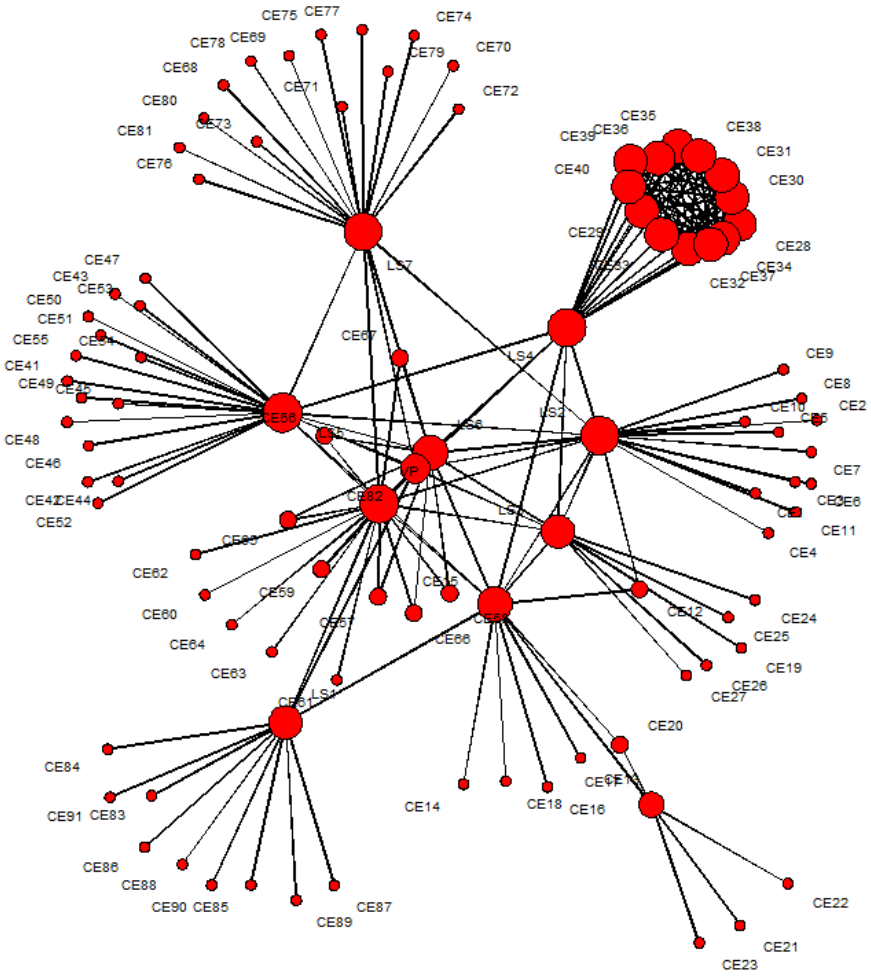
To further improve network plots, we can vary the size, shape, and/or colors of the nodes in. For example, we can set node size to vary according to how central the node is in the network. Central nodes then have larger dots, as compared with isolated nodes.

To vary node sizes according to node centrality, we first need to create a data object with information on employees' centralization scores. In this case, we are interested in the

degree centrality and we will calculate a data object that is called “degree” (for details, see 6.2.1.1). Next, we specify that the vertex size should depend on the individual’s centralization score, as specified in “degree”. This is done by adding `vertex.cex=degree` to `gplot`. Upon inspection, this code results in extremely large nodes for central members of the organization. The node sizes of employees with lower centralization scores, on the other hand, are fine. Hence, we need to apply a non-linear transformation to the node sizes; one that reduces the node sizes of central employees while leaving the node sizes of other members relatively unchanged. To do this, we can take the square root, natural logarithm, or common logarithm of node centrality. Here, we use the common logarithm, as this gives the best result (Figure 13). We add a 0.50 intercept to the node size value, to make sure that the minimum node size is 0.50:

```
degree<-degree(network_data_sociomatrix,gmode="graph")

gplot(network_data_sociomatrix,displaylabels=T,
      label.cex=0.60,
      vertex.cex=(log10(degree))+0.5,
      usearrows=FALSE)
```

Figure 13 - Network plot with varying node sizes



7.2.2 Varying nodes shapes in network plots

Node shapes can be used to display nodes' group affiliations. In our case organization, we can, for example, assign a 3-sided triangle to members of the management team, a 4-sided square to department DEP9, a 5-sided pentagon to DEP10, a 6-sided hexagon to DEP11, etc. This will help to visualize within and between-departmental ties in the organization.

To vary node shapes, we first need to recode the vertex.attribute "dep", such that all managers have a department number of "1" (in the original attributes file, managers had different department numbers). To do so, we duplicate the vertex.attribute "dep" as a new data object called "dep_nodes". Then, we recode the department values of the managers (dep 1-8) to "1" in the "dep_nodes" data object. We keep the department values of other employees unchanged. We subsequently convert this data object by using `as.factor`:

```
dep_nodes<-attributes$dep

dep_nodes[dep_nodes<9]<-1

dep_nodes<-as.factor(dep_nodes)
```

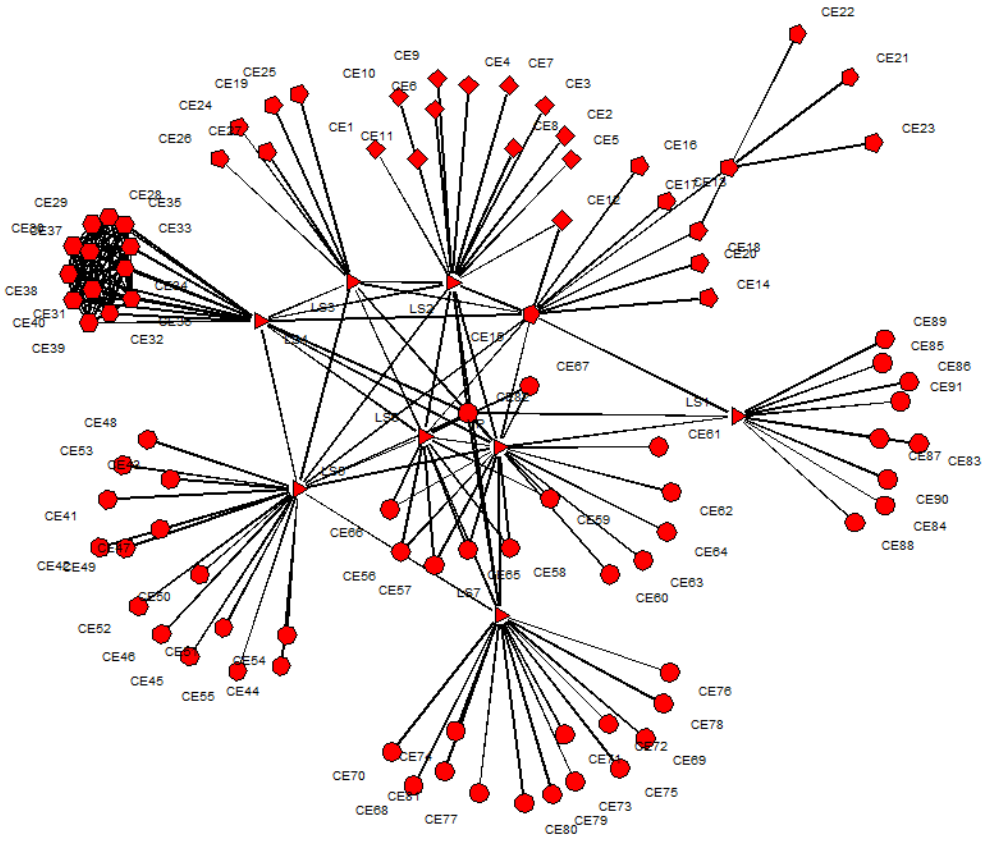
Next, we need to specify the shapes we want to apply in our network plot. We have 8 groups in our organization (one management team, and 7 departments). Hence, we need 8 different shapes to display department affiliation in the network plot. An enclosed shape has a minimum of 3 sides (i.e., a triangle). Every next shape adds one side. Considering that we need 8 shapes, our network plot thus needs to include triangles (3 sides), squares (4 sides), pentagons (5 sides), hexagons (6 sides), heptagons (7 sides), octagons (8 sides), nonagons (9 sides), and decagons (10 sides). In other words, our shapes need to vary between 3 and 10 sides. We inform R-studio of this by creating a new data object (called "sides") that includes information on the range of shapes we are looking to use in our network plot.

```
sides <- 3:10
```

Once we created the departments identifier ("dep_nodes") and specified range of sides of the accompanying shapes ("sides"), we can add this to the `gplot` code. We use `vertex.sides` to specify that the number of sides is specified in the data object "sides" and should be contingent on the department affiliation, as stored in "dep_nodes". The result is shown in Figure 14. As you will notice, it is hard to distinguish between, for example, nonagons and decagons. Hence, varying shapes is only useful when only a few groups needs to be represented.

```
gplot(network_data_sociomatrix,displaylabels=T,
      label.cex=0.60,
      vertex.sides=sides[dep_nodes],
      usearrows=FALSE)
```

Figure 14 - Network plot with varying node shapes



7.2.3 Varying nodes colors in network plots

Colors can also be useful to distinguish between organizational groups in a network plot. We can, for example, color the leader nodes red and the nodes of employees blue.

To implement this in R-studio, we first need to create a data object that specifies whether a node is a leader or an employee. Similar to what we did in the previous section, we create this data object by taking information from the attributes data object. We call this new data object "hierarchical_role". In our attributes list, leaders have department codes 1:8 and employees has department codes 9:15. To correctly code "hierarchical_role", we thus recode values 1:8 into 1 (=leader) and 9:15 into 2 (=employee):

```
hierarchical_role<-attributes$dep
hierarchical_role[hierarchical_role<9]<-1
hierarchical_role[hierarchical_role>8]<-2
hierarchical_role<-as.factor(hierarchical_role)
```

In the following step, we specify the color pallet that we want to use. In our case, we want to display leader nodes in red and employee nodes in blue. To specify this pallet, we create a data object that we call "color_pallet" that lists the color red and blue:

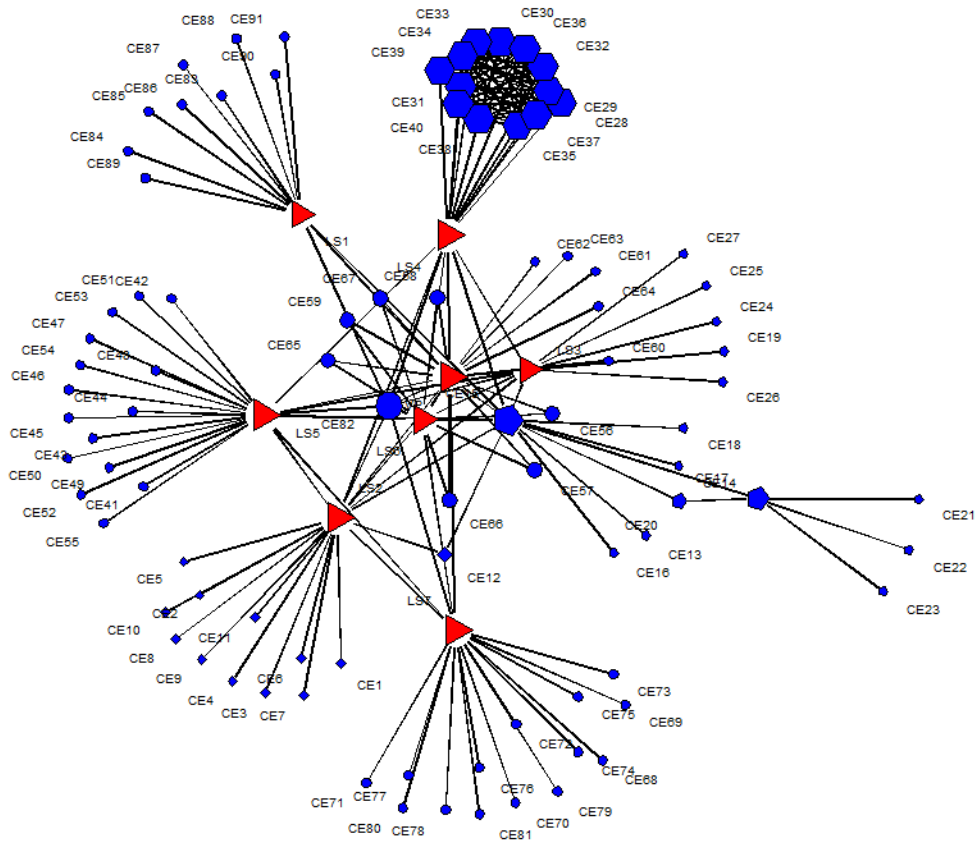
```
color_pallet <- c("red", "blue")
```

We then use `vertex.color` in gplot to specify that the node color that R picks from our pallet should depend on the node's "hierarchical_role". This will assign the same color to

node's who have the same "hierarchical_role" value and, of course, different colors to individuals who have different hierarchical roles (see Figure 15).

```
gplot(network_data_sociomatrix,
      usearrows=FALSE,
      displaylabels=T,
      label.cex=0.60,
      vertex.cex=(log10(degree))+0.5,
      vertex.sides=sides[dep_nodes],
      vertex.col=color_pallet[hierarchical_role])
```

Figure 15 - Network plot with varying node shapes and colors



7.2.4 Varying tie thickness in network plots

Finally, we can change the layout of the lines that represent ties between nodes. For example, we can vary line thickness according to tie closeness (see 4.3.4). Thick lines then represent close ties between nodes, while thin lines represent more distal ties.

To vary line thickness depending on an edge attribute, we first need to make sure that the edge attribute is attached to the socio-matrix. If not, we should load the .csv file that houses the edge information and load it using `set.edge.value` (see 4.3.4). Subsequently, we create a new data object called "edge" that lists the edge attribute "closeness":

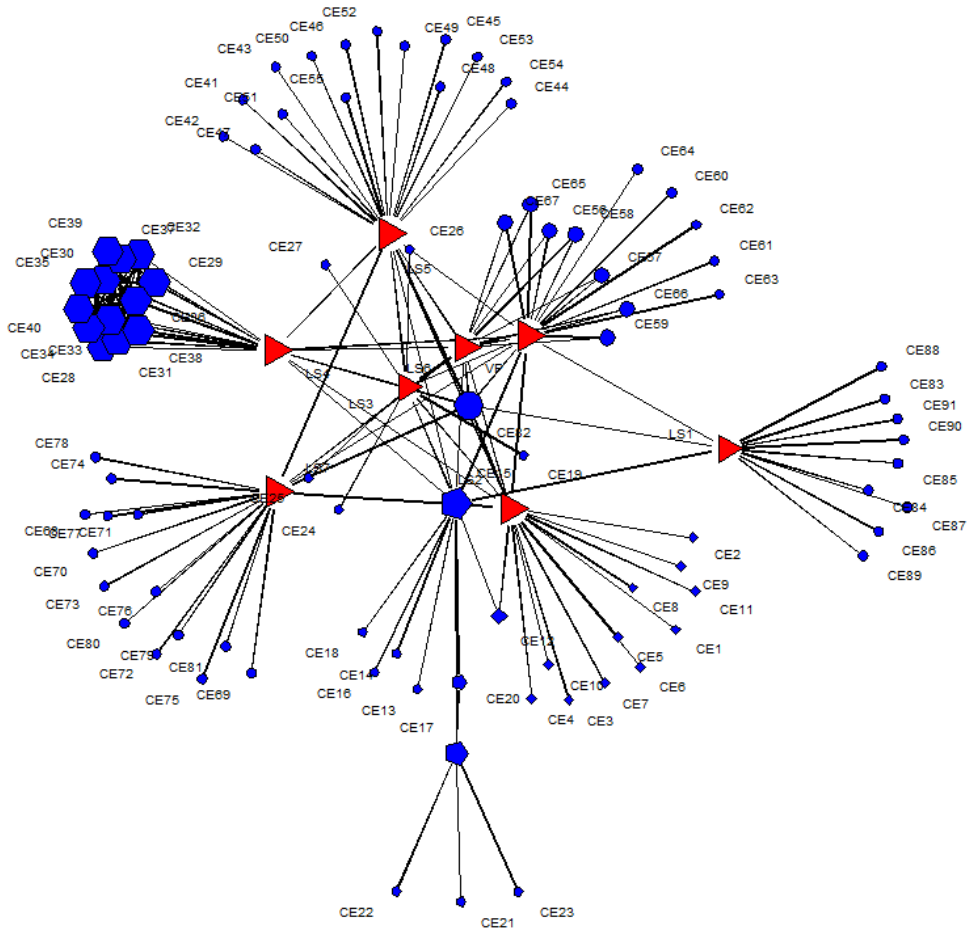
```
edge <- as.matrix(read.csv("company_edge_closeness.csv", header=F))

network_data_sociomatrix<-set.edge.value(network_data_sociomatrix,
'closeness',edge)
```

```
closeness<-network_data_sociomatrix %e% "closeness"
```

In the final step we specify with `edge.lwd` that the line thickness should depend on the closeness value of a tie. Here, we multiply the closeness score by a factor of 0.20 to avoid that lines in the network plot become overly thick. The result is shown in Figure 16.

```
gplot(network_data_sociomatrix,
      usearrows=FALSE,
      displaylabels=T,
      label.cex=0.60,
      vertex.cex=(log10(degree))+0.5,
      vertex.sides=sides[dep_nodes],
      vertex.col=color_pallet[hierarchical_role],
      edge.lwd = 0.2*closeness)
```

Figure 16 - Network plot with varying tie thickness



8 INDEX R-COMMANDS