Network analysis offers a perspective of the data that broadens and enriches any investigation. Many times we deal with data in which the elements are related, but we have them in a tabulated format that is difficult to import into network analysis tools.

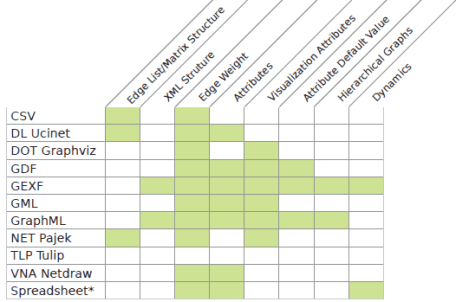
Relationship data require a definition of nodes and connections. Both parts have different structures and it is not possible to structure them in a single table, at least two would be needed. Data analysis tools define different input formats, one of them is GDF, which is characterized by its simplicity and versatility.

In this session, we will see how we can extract the relationships between elements of a file in CSV format to generate a file in GDF format to work with Gephi.

**Introduction**

One of the most used network analysis tools is Gephi. It is a desktop application that allows us to analyse a network in depth and visualize it. It provides a set of functions to filter nodes and edges or calculate network parameters or apply different layouts or give size and colour to nodes depending on different attributes.

Gephi supports a set of formats with which it is possible to perform more or less functions. The format with more features is GEXF but it has an XML structure that generates large files. When we work with very large graphs, size matters. The GDF Format is simple, compact and versatile. It allows us to work with attributes that are a complement that can be combined with the analysis of the topology of the network.



As Gephi said:

GDF is built like a database table or a coma separated file (CSV). It supports attributes to both nodes and edges. A standard file is divided in two sections, one for nodes and one for edges. Each section has a header line, which basically is the column title. Each element (i.e. node or edge) is on a line and values are separated by coma. The GDF format is therefore very easy to read and can be easily converted from CSV.

This is a basic example of a GDF file, in a first part it defines the nodes and in a second the connections:

nodedef>name VARCHAR,label VARCHAR

s1,Site number 1

s2,Site number 2

s3,Site number 3

edgedef>node1 VARCHAR,node2 VARCHAR

s1,s2

s2,s3

s3,s2

s3,s1

The steps to extract relationships from a CSV file and generate a file in GDF format are:

1. Define the parameters for the extraction of related data.
2. Define the data structures for the transformation.
3. Import the data from a file in CSV format and store the data in the structures.
4. Sort data according to connections.
5. Prepare the data for the GDF format.
6. Generate the file in GDF format.

**Define the parameters for the extraction of related data**

A set of parameters are defined so this converter can be adapted to different cases. We must specify what information will be taken from the file in CSV format, where we will leave the result and the graph type.

library("readr")

library("hash")

library("stringr")

source <- 3 # Column number of source entity

target <- 9 # Column number of target entity

source\_attribs <- c(5, 6, 11, 12, 13) # Column numbers of attribs

null\_attrib <- rep(NA, 5) # Default value for entities without attributes

name\_attribs <- c("app", "location", "hastag", "lang", "create at") # Attrib names

name\_file\_csv <- "rstudio\_RTs.csv" # Name of the input file in CSV format

name\_file\_gdf <- "rstudio\_RTs.gdf" # Name of the output file in GDF format

directed <- TRUE # Indicates whether the graph is directed or not

The sample CSV file can be found [here](https://codingclubuc3m.rbind.io/post/2019-06-18_files/rstudio_RTs.csv).

**Define the data structures for the transformation**

In order to store the data, dynamic structures are needed to allow data to be added as they appear. The hash tables were chosen because they are the most appropriate for this case.  
Hash tables will be used to store nodes and connections.

hash\_nodes <- hash()

hash\_links <- hash()

hash\_links\_in <- hash()

hash\_links\_out <- hash()

hash\_connections <- hash()

hash\_connections\_attrib <- hash()

num\_attribs <- length(source\_attribs)

**Import the data from a file in CSV format and store the data in the structures**

Import data reading the CSV file and run it row by row to store the nodes and connections in the hash tables.

Related entities can appear multiple times, as a source or as a target. When an entity appears for the first time, it is stored in the hash\_nodes table. Attributes are associated to source entities and null attributes to target entities. It is a criterion that assumes this algorithm, but there could be others. If an entity appears the first time as a target, it will be assigned the null attributes, but if it appears later as a source, the null attributes will be replaced by theirs.

For each entity, the number of total links (hash\_links), the number of inbound links (hash\_links\_in) and the number of outbound links (hash\_links) are counted. This is done to allow ordering the nodes from greater to lesser degree when generating the file in GDF format.

For each origin-target entity pair, the number of times that the relation appears (hash\_connections) and the attributes (hash\_connections\_attrib) are stored. In the first case, we get the weight of the relationship and, in the second, we get the associated attributes.

table\_csv <- read\_csv2(name\_file\_csv)

num\_rows <- nrow(table\_csv)

num\_cols <- ncol(table\_csv)

for (i in 1:num\_rows)

{

node\_source <- table\_csv[[i, source]]

node\_target <- table\_csv[[i, target]]

print(i)

node\_source\_attribs <- null\_attrib

# The attributes are stored and the ',' character is changed by '-'. to avoid confict in the GDF format

for (j in 1:num\_attribs)

{

k <- source\_attribs[[j]]

raw\_attrib <- table\_csv[[i, k]]

cooked\_attrib <- str\_replace\_all(raw\_attrib, ",", "-")

node\_source\_attribs[[j]] <- cooked\_attrib

}

# If a source node appears for the first time, store with attributes

if (!(has.key(node\_source, hash\_nodes)))

{

hash\_nodes[[node\_source]] <- node\_source\_attribs

hash\_links[[node\_source]] <- 0

hash\_links\_in[[node\_source]] <- 0

hash\_links\_out[[node\_source]] <- 0

}

# If the source node exists and has null attributes, store its own

else

{

node\_source\_attribs\_old <- hash\_nodes[[node\_source]]

if (identical(node\_source\_attribs\_old, null\_attrib))

hash\_nodes[[node\_source]] <- node\_source\_attribs

}

# Check that target node exists

if (!(is.na(node\_target)))

{

# If a target node appears for the first time, store with null attributes

if (!(has.key(node\_target, hash\_nodes)))

{

hash\_nodes[[node\_target]] <- null\_attrib

hash\_links[[node\_target]] <- 0

hash\_links\_in[[node\_target]] <- 0

hash\_links\_out[[node\_target]] <- 0

}

par\_nodes <- paste(node\_source, node\_target)

# If a pair of nodes appears for the first time related, store relation

if (!(has.key(par\_nodes, hash\_connections)))

{

hash\_connections[[par\_nodes]] <- 0

hash\_connections\_attrib[[par\_nodes]] <- node\_source\_attribs

}

# In all cases, increase the number of connections

hash\_connections[[par\_nodes]] <- hash\_connections[[par\_nodes]] + 1

hash\_links[[node\_source]] <- hash\_links[[node\_source]] + 1

hash\_links\_out[[node\_source]] <- hash\_links\_out[[node\_source]] + 1

hash\_links[[node\_target]] <- hash\_links[[node\_target]] + 1

hash\_links\_in[[node\_target]] <- hash\_links\_in[[node\_target]] + 1

}

}

**Sort data according to connections**

The hash table object does not have the sort method, but has one to convert it into a list. Once we have converted the hash\_links and hash\_connections into a list, we sort them down by number of connections.

list\_links <- as.list.hash(hash\_links)

list\_link\_order <- list\_links[order(unlist(list\_links), decreasing=TRUE)]

list\_connections <- as.list.hash(hash\_connections)

list\_connections\_order <- list\_connections[order(unlist(list\_connections), decreasing=TRUE)]

**Prepare the data for the GDF format**

In this step we place in GDF format nodes and links in descending order by number of connections.

In the GDF format, the only data required for the definition of nodes is the name of the node, but attributes can be added. In this case, three fixed attributes are included, which are the total number of links, the number of inbound links and the number of outbound links. Since the GDF format is readable, these attributes allow getting an idea of the most relevant nodes even before importing them into Gephi. The attributes configured in the parameters are also added.

The information of the nodes is stored in a matrix sized in rows by the number of nodes and in columns by the number of attributes configured plus four.

For the definition of links only the source and target nodes are required, but we can also expand them with attributes. In this case we add the weight of the relation, a boolean variable to indicate if the graph is directed or not (by default it is not directed) and the attributes configured in the parameters.

The information of the links is stored in a matrix dimensioned in rows by the number of pairs of connections and in columns by the number of attributes configured plus four.

# Definition of nodes

num\_nodes <- length(list\_links)

table\_nodes <- matrix(nrow=num\_nodes, ncol=num\_attribs+4)

num\_nodes\_connected <- 0

for(i in 1:num\_nodes)

{

name\_node <- names(list\_link\_order)[i]

if (hash\_links[[name\_node]] > 0)

{

num\_nodes\_connected <- num\_nodes\_connected + 1

table\_nodes[i, 1] <- name\_node

table\_nodes[i, 2] <- hash\_links[[name\_node]]

table\_nodes[i, 3] <- hash\_links\_in[[name\_node]]

table\_nodes[i, 4] <- hash\_links\_out[[name\_node]]

node\_attrib <- hash\_nodes[[name\_node]]

for (j in 1:num\_attribs)

table\_nodes[i, 4+j] <- node\_attrib[[j]]

}

}

# Only connected nodes are considered

k <- num\_attribs + 4

table\_nodes <- table\_nodes[1:num\_nodes\_connected, 1:k]

# Definition of links

num\_connections <- length(list\_connections)

table\_connections <- matrix(nrow=num\_connections, ncol=num\_attribs+4)

for(i in 1:num\_connections)

{

name\_conexion <- names(list\_connections\_order)[i]

source\_target <- strsplit(name\_conexion, " ")

table\_connections[i, 1] <- source\_target[[1]][1]

table\_connections[i, 2] <- source\_target[[1]][2]

table\_connections[i, 3] <- list\_connections\_order[[i]]

table\_connections[i, 4] <- directed

connection\_attrib <- hash\_connections\_attrib[[name\_conexion]]

for (j in 1:num\_attribs)

table\_connections[i, 4+j] <- connection\_attrib[[j]]

}

**Generate the file in GDF format**

The last step is to write the file in GDF format. We will only have to add the headers before writing the information of the nodes and links.

# Definition of nodes

head\_nodes <- "nodedef>name VARCHAR,links INT,Links\_in INT,links\_out INT"

for (j in 1:num\_attribs)

{

attrib\_type <- paste(name\_attribs[[j]], "VARCHAR", sep = " ")

head\_nodes <- paste(head\_nodes, attrib\_type, sep = ",")

}

write.table(head\_nodes, file = name\_file\_gdf, append = FALSE, quote = FALSE, sep = ",",

eol = "\n", na = "NA", dec = ".", row.names = FALSE,

col.names = FALSE, qmethod = c("escape", "double"),

fileEncoding = "UTF-8")

write.table(table\_nodes, file = name\_file\_gdf, append = TRUE, quote = FALSE, sep = ",",

eol = "\n", na = "NA", dec = ".", row.names = FALSE,

col.names = FALSE, qmethod = c("escape", "double"),

fileEncoding = "UTF-8")

# Definition of links

head\_arcs <- "edgedef>node1 VARCHAR,node2 VARCHAR, weight INT, directed BOOLEAN"

for (j in 1:num\_attribs)

{

attrib\_type <- paste(name\_attribs[[j]], "VARCHAR", sep = " ")

head\_arcs <- paste(head\_arcs, attrib\_type, sep = ",")

}

write.table(head\_arcs, file = name\_file\_gdf, append = TRUE, quote = FALSE, sep = ",",

eol = "\n", na = "NA", dec = ".", row.names = FALSE,

col.names = FALSE, qmethod = c("escape", "double"),

fileEncoding = "UTF-8")

write.table(table\_connections, file = name\_file\_gdf, append = TRUE, quote = FALSE, sep = ",",

eol = "\n", na = "NA", dec = ".", row.names = FALSE,

col.names = FALSE, qmethod = c("escape", "double"),

fileEncoding = "UTF-8")

**The Complete R Code**

library("readr")

library("hash")

library("stringr")

################################

# parameters

# A set of parameters are defined so this converter can be adapted to different cases.

# We must specify what information will be taken from the file in CSV format,

# where we will leave the result and the graph type

#

################################

source <- 3 # Column number of the origin of the relationship

target <- 9 # Column number of the destination of the relationship

source\_attribs <- c(5,6,11,12,13) # Column numbers where to get the attributes

null\_attrib <- c(NA,NA,NA,NA,NA) # Default value for entities without attributes

name\_attribs <- c("app",

"location",

"hastag",

"lang",

"create at") # Names of the attributes

name\_file\_csv <- "rstudio\_RTs.csv" # Name of the input file in CSV format

name\_file\_gdf <- "rstudio\_RTs.gdf" # Name of the output file in GDF format

directed <- TRUE # Indicates whether the graph is directed or not

###############################

# Data

#

# In order to store the data, dynamic structures are needed to allow data to be

# added as they appear. The hash tables were chosen because they are the most appropriate

# for this case. Hash tables will be used to store nodes and connections

###############################

hash\_nodes <- hash()

hash\_links <- hash()

hash\_links\_in <- hash()

hash\_links\_out <- hash()

hash\_connections <- hash()

hash\_connections\_attrib <- hash()

num\_attribs <- length(source\_attribs)

###############################

# read source data

#

# Import data reading the CSV file and run it row by row to store the nodes and connections

# in the hash tables.

#

# Related entities can appear multiple times, as a source or as a target. When an entity appears

# for the first time, it is stored in the hash\_nodes table. Attributes are associated to source

# entities and null attributes to target entities. It is a criterion that assumes this algorithm,

# but there could be others. If an entity appears the first time as a target, it will be assigned

# the null attributes, but if it appears later as a source, the null attributes will be replaced

# by theirs

#

# For each entity, the number of total links (hash\_links), the number of inbound links (hash\_links\_in)

# and the number of outbound links (hash\_links) are counted. This is done to allow ordering the nodes

# from greater to lesser degree when generating the file in GDF format.

#

# For each origin-target entity pair, the number of times that the relation appears (hash\_connections)

# and the attributes (hash\_connections\_attrib) are stored. In the first case, we get the weight of

# the relationship and, in the second, we get the associated attributes.

###############################

table\_csv <- read\_csv2(name\_file\_csv)

num\_rows <- nrow(table\_csv)

num\_cols <- ncol(table\_csv)

for (i in 1:num\_rows)

{

node\_source <- table\_csv[[i,source]]

node\_target <- table\_csv[[i,target]]

print (i)

node\_source\_attribs<-null\_attrib

# The attributes are stored and the ',' character is changed by '-'. to avoid confict in the GDF format

for (j in 1:num\_attribs)

{

k <- source\_attribs[[j]]

raw\_attrib <- table\_csv[[i,k]]

cooked\_attrib <- str\_replace\_all(raw\_attrib,",","-")

node\_source\_attribs[[j]] <- cooked\_attrib

}

# If a source node appears for the first time, store with attributes

if (!(has.key(node\_source, hash\_nodes)))

{

hash\_nodes[[node\_source]] <- node\_source\_attribs

hash\_links[[node\_source]] <- 0

hash\_links\_in[[node\_source]] <- 0

hash\_links\_out[[node\_source]] <- 0

}

# If the source node exists and has null attributes, store its own

else

{

node\_source\_attribs\_old=hash\_nodes[[node\_source]]

if (identical(node\_source\_attribs\_old,null\_attrib))

{hash\_nodes[[node\_source]] <- node\_source\_attribs}

}

# Check that target node exists

if (!(is.na(node\_target)))

{

# If a target node appears for the first time, store with null attributes

if (!(has.key(node\_target, hash\_nodes)))

{

hash\_nodes[[node\_target]] <- null\_attrib

hash\_links[[node\_target]] <- 0

hash\_links\_in[[node\_target]] <- 0

hash\_links\_out[[node\_target]] <- 0

}

#Store connections

par\_nodes=paste(node\_source,node\_target)

# If a pair of nodes appears for the first time related, store relation

if (!(has.key(par\_nodes, hash\_connections)))

{

hash\_connections[[par\_nodes]] <- 0

hash\_connections\_attrib[[par\_nodes]] <- node\_source\_attribs

}

# In all cases, increase the number of connections

hash\_connections[[par\_nodes]] <- hash\_connections[[par\_nodes]] +1

hash\_links[[node\_source]] <- hash\_links[[node\_source]]+1

hash\_links\_out[[node\_source]] <- hash\_links\_out[[node\_source]] +1

hash\_links[[node\_target]] <- hash\_links[[node\_target]]+1

hash\_links\_in[[node\_target]] <- hash\_links\_in[[node\_target]] +1

}

}

##################################

# Order descending by connections

#

# The hash table object does not have the sort method, but has one to convert it into a list.

#

# Once we have converted the hash\_links and hash\_connections into a list, we sort them down

# by number of connections

##################################

list\_links <- as.list.hash(hash\_links )

list\_link\_order <- list\_links[order(unlist(list\_links), decreasing=TRUE)]

list\_connections <- as.list.hash(hash\_connections )

list\_connections\_order <- list\_connections[order(unlist(list\_connections), decreasing=TRUE)]

##################################

# Prepare the data for the GDF format

#

# In this step we place in GDF format nodes and links in descending order by number

# of connections.

#

# In the GDF format, the only data required for the definition of nodes is the name of the node,

# but attributes can be added. In this case, three fixed attributes are included, which are the

# total number of links, the number of inbound links and the number of outbound links. Since the

# GDF format is readable, these attributes allow getting an idea of the most relevant nodes even

# before importing them into Gephi. The attributes configured in the parameters are also added.

# The information of the nodes is stored in a matrix sized in rows by the number of nodes and in columns

# by the number of attributes configured plus four.

#

# For the definition of links only the source and target nodes are required, but we can also expand

# them with attributes. In this case we add the weight of the relation, a boolean variable to indicate

# if the graph is directed or not (by default it is not directed) and the attributes configured

# in the parameters.

# The information of the links is stored in a matrix dimensioned in rows by the number of pairs of

# connections and in columns by the number of attributes configured plus four.

#################################

# Definition of nodes

num\_nodes=length(list\_links)

table\_nodes <- matrix(nrow=num\_nodes,ncol=num\_attribs+4)

num\_nodes\_connected <- 0

for(i in 1:num\_nodes)

{

name\_node=names(list\_link\_order)[i]

if (hash\_links[[name\_node]] > 0)

{

num\_nodes\_connected <- num\_nodes\_connected+1

table\_nodes[i,1] <- name\_node

table\_nodes[i,2] <- hash\_links[[name\_node]]

table\_nodes[i,3] <- hash\_links\_in[[name\_node]]

table\_nodes[i,4] <- hash\_links\_out[[name\_node]]

node\_attrib <- hash\_nodes[[name\_node]]

for (j in 1:num\_attribs)

{

table\_nodes[i,4+j] <- node\_attrib[[j]]

}

}

}

# Only connected nodes are considered

k <- num\_attribs+4

table\_nodes <- table\_nodes[1:num\_nodes\_connected, 1:k]

# Definition of links

num\_connections <- length(list\_connections)

table\_connections <- matrix(nrow=num\_connections,ncol=num\_attribs+4)

for(i in 1:num\_connections)

{

name\_conexion <- names(list\_connections\_order)[i]

source\_target <- strsplit(name\_conexion," ")

table\_connections[i,1] <- source\_target[[1]][1]

table\_connections[i,2] <- source\_target[[1]][2]

table\_connections[i,3] <- list\_connections\_order[[i]]

table\_connections[i,4] <- directed

connection\_attrib <- hash\_connections\_attrib[[name\_conexion]]

for (j in 1:num\_attribs)

{

table\_connections[i,4+j] <- connection\_attrib[[j]]

}

}

#################################

# Generate the file in GDF format

#

# The last step is to write the file in GDF format. We will only have to add the headers

# before writing the information of the nodes and links.

#################################

# Definition of nodes

head\_nodes <- "nodedef>name VARCHAR,links INT,Links\_in INT,links\_out INT"

for (j in 1:num\_attribs)

{

attrib\_type <- paste(name\_attribs[[j]],"VARCHAR",sep = " ")

head\_nodes <- paste(head\_nodes,attrib\_type,sep = ",")

}

write.table(head\_nodes, file = name\_file\_gdf, append = FALSE, quote = FALSE, sep = ",",

eol = "\n", na = "NA", dec = ".", row.names = FALSE,

col.names = FALSE, qmethod = c("escape", "double"),

fileEncoding = "UTF-8")

write.table(table\_nodes, file = name\_file\_gdf, append = TRUE, quote = FALSE, sep = ",",

eol = "\n", na = "NA", dec = ".", row.names = FALSE,

col.names = FALSE, qmethod = c("escape", "double"),

fileEncoding = "UTF-8")

# Definition of links

head\_arcs<-"edgedef>node1 VARCHAR,node2 VARCHAR, weight INT, directed BOOLEAN"

for (j in 1:num\_attribs)

{

attrib\_type <- paste(name\_attribs[[j]],"VARCHAR",sep = " ")

head\_arcs <- paste( head\_arcs,attrib\_type,sep = ",")

}

write.table(head\_arcs, file = name\_file\_gdf, append = TRUE, quote = FALSE, sep = ",",

eol = "\n", na = "NA", dec = ".", row.names = FALSE,

col.names = FALSE, qmethod = c("escape", "double"),

fileEncoding = "UTF-8")

write.table(table\_connections, file = name\_file\_gdf, append = TRUE, quote = FALSE, sep = ",",

eol = "\n", na = "NA", dec = ".", row.names = FALSE,

col.names = FALSE, qmethod = c("escape", "double"),

fileEncoding = "UTF-8")

**Example**

The data has been obtained with the tool [t-hoarder\_kit](https://github.com/congosto/t-hoarder_kit), which allows downloading data through the Twitter API. With it, a query of the tweets that mentioned the @rstudio profile has been made. From the data downloaded in CSV format, the most relevant columns have been selected to facilitate the visibility of the tables.

Once the file in CSV format has been converted to a file in GDF format with the RT relation, this is the visualization of the graph with Gephi.

