Discourse Network Analyzer

Philip Leifeld

Last update: DNA 2.0 beta 20 with rDNA 2.0.1 on 2017-04-10.

This documents describes the software `Discourse Network Analyzer` (`DNA`) and its companion `R` package `rDNA` (Leifeld 2017). Section 1 is a concise and fairly technical description of the types of networks `DNA` can export. Section 2 is an introductory tutorial on using the `rDNA` package. The contents of both sections are meant to be extended in the future, and additional sections may be added. `DNA` and `rDNA` can be downloaded from `GitHub`. Questions and bug reports can be posted in the issue tracker on `GitHub`.

1 `DNA` algorithms

This section summarizes the main algorithms implemented in `DNA` in a technical way.

X is a three-dimensional array representing statement counts. x_{ijk} is a specific count value in this array, with the first index i denoting an instance of the first variable (e.g., organization i), the second index j denoting an instance of the second variable (e.g., concept j), and the third index k denoting a level on the qualifier variable (e.g., agreement $= 1$). For example, $x_{ijk} = 5$ could mean that organization i mentions concept j with intensity k five times.

Where the qualifier variable is binary, *false* values are represented as 0 and *true* values as 1 on the k index, i.e., $K^{\text{binary}} = \{0; 1\}$. Where the qualifier variable is integer, the respective integer value is used as the level. This implies that k can take positive or negative values or 0, i.e, $K^{\text{integer}} \subseteq \mathbb{Z}$. Note that all k levels of the scale are included in K, not just those values that are empirically observed.

Indices with a prime denote a second instance of an element, e.g., i' may denote another organization. Y denotes the output matrix to be obtained by applying a transformation to X. The following transformations are possible:

1.1 Congruence

In a congruence network, the edge weight between nodes i and i' represents the number of times they co-support or co-reject second-variable nodes (if a binary qualifier is used) or the cumulative similarity between i and i' over their assessments of second-variable nodes (in the case of an integer qualifier variable).

In the integer case:

$$y_{ii'}^{\text{congruence}} = \Phi_{ii'} \left(\sum_{j=1}^{n} \sum_{k} \sum_{k'} x_{ijk} x_{i'jk'} \left(1 - \frac{|k - k'|}{|K| - 1} \right) \right) \tag{1}$$

where $\Phi_{ii'}(\cdot)$ denotes a normalization function (to be specified below).

In the binary case, i.e., $|K| = 2$, this reduces to

$$y_{ii'}^{\text{congruence binary}} = \Phi_{ii'} \left(\sum_{j=1}^{n} \sum_{k} x_{ijk} x_{i'jk} + (1 - x_{ijk})(1 - x_{i'jk}) \right). \tag{2}$$

1.2 Conflict

Binary case:

$$y_{ii'}^{\text{conflict binary}} = \Phi_{ii'} \left(\sum_{j=1}^{n} \sum_{k} (1 - x_{ijk}) x_{i'jk} + x_{ijk}(1 - x_{i'jk}) \right). \tag{3}$$

More generally, in the integer case:

$$y_{ii'}^{\text{conflict}} = \Phi_{ii'} \left(\sum_{j=1}^{n} \sum_{k} \sum_{k'} x_{ijk} x_{i'jk'} \left(\frac{|k - k'|}{|K| - 1} \right) \right) \tag{4}$$

1.3 Subtract

$$y_{ii'}^{\text{subtract}} = y_{ii'}^{\text{congruence}} - y_{ii'}^{\text{conflict}} \tag{5}$$

1.4 Ignore

$$y_{ii'}^{\text{ignore}} = \Phi_{ii'} \left(\sum_{j=1}^{n} \left(\left(\sum_{k} x_{ijk} \right) \left(\sum_{k} x_{i'jk} \right) \right) \right) \tag{6}$$

1.5 Normalization

In the simplest case, normalization can be switched off, in which case $\Phi_{ii'}^{\text{no}}(\omega) = \omega$.

Alternatively, edge weights can be divided by the average activity of nodes i and i':

$$\Phi_{ii'}^{\text{avg}}(\omega) = \frac{\omega}{\frac{1}{2}\left(\sum_{j=1}^{n} \sum_{k} x_{ijk} + \sum_{j=1}^{n} \sum_{k} x_{i'jk} \right)}. \tag{7}$$

With Jaccard normalization, we don't just count i's and i''s activity and sum them up independently, but we add up both their independent activities and their joint activity, i.e., both matches and non-matches:

$$\Phi_{ii'}^{\text{Jaccard}}(\omega) = \frac{\omega}{\sum_{j=1}^{n} \sum_{k} x_{ijk}[x_{i'jk} = 0] + \sum_{j=1}^{n} \sum_{k} x_{i'jk}[x_{ijk} = 0] + \sum_{j=1}^{n} \sum_{k} x_{ijk} x_{i'jk}}. \tag{8}$$

With cosine normalization, we take the product in the denominator:

$$\Phi_{ii'}^{\text{cosine}}(\omega) = \frac{\omega}{\sqrt{(\sum_{j=1}^{n} \sum_{k} x_{ijk})^2} \sqrt{(\sum_{j=1}^{n} \sum_{k} x_{i'jk})^2}}. \tag{9}$$

1.6 Affiliation networks

Ignoring the qualifier variable:

$$y_{ij}^{\text{affiliation ignore}} = \Phi_{ij}\left(\sum_k x_{ijk}\right) \tag{10}$$

Subtracting negative from positive ties (integer case):

$$y_{ij}^{\text{affiliation subtract binary}} = \Phi_{ij}\left(\sum_k k \cdot x_{ijk}\right) \tag{11}$$

Subtracting negative from positive ties (binary case):

$$y_{ij}^{\text{affiliation subtract binary}} = \Phi_{ij}\left(\sum_k (k \cdot x_{ijk} - (1-k) \cdot x_{ijk})\right) \tag{12}$$

Note that the binary case is *not* merely a special case of the weighted affiliation network in this case.

1.7 Normalization for affiliation networks

With *activity* normalization, ties from active nodes receive lower weights:

$$\Phi_{ij}^{\text{activity}}(\omega) = \frac{\omega}{\sum_{j=1}^{n}\sum_k x_{ijk}} \tag{13}$$

With *prominence* normalization, ties to prominent nodes receive lower weights:

$$\Phi_{ij}^{\text{prominence}}(\omega) = \frac{\omega}{\sum_{i=1}^{m}\sum_k x_{ijk}} \tag{14}$$

2 rDNA: Using DNA from R

DNA can be connected to the statistical computing environment R (R Core Team 2014) through the rDNA package (Leifeld 2017). There are two advantages to working with R on DNA data.

The first advantage is replicability. The network export function of DNA has many options. Remembering what options were used in an analysis can be difficult. If the analysis is executed in R, commands—rather than mouse clicks—are used to extract networks or attributes from DNA. These commands are saved in an R script file. This increases replicability because the script can be re-used many times. For example, after discovering a wrong code somewhere in the DNA database, it is sufficient to fix this problem in the DNA file and then re-run the R script instead of manually setting all the options again. This reduces the probability of making errors and increases replicability.

The second advantage is the immense flexibility of R in terms of statistical modeling. Analyzing DNA data in R permits many forms of data analysis beyond simple visualization of the resulting networks. Examples include cluster analysis or community detection, scaling and application of data reduction techniques, centrality analysis, and even statistical modeling of network data. R is also flexible in terms of combining and matching the data from DNA with other data sources.

2.1 Getting started with rDNA

The first step is to install R. Installing additional R packages for network analysis and clustering, such as statnet (Goodreau et al. 2008; Handcock et al. 2008, 2016), xergm (Leifeld et al. 2017a,b), igraph (Csardi and Nepusz 2006), and cluster (Maechler et al. 2017), is recommended. Moreover, it is necessary to install the rJava package (Urbanek 2016), on which the rDNA package depends, and the devtools package (Wickham and Chang 2016), which permits installing R packages from GitHub.

```
install.packages("statnet")
install.packages("xergm")
install.packages("igraph")
install.packages("cluster")
install.packages("rJava")
install.packages("devtools")
```

After installing these supplementary packages, the rDNA package can be installed from GitHub. The devtools package contains a function that permits easy installation of R packages from GitHub and can be used as follows to install rDNA:

```
library("devtools")
install_github("leifeld/dna/rDNA")
```

Once installed, the rDNA package must be attached to the workspace:

```
library("rDNA")
```

To ensure that the following results can be reproduced exactly, we should set the random seed in R:

```
set.seed(12345)
```

Now we are able to use the package. The first step is to initialize DNA. Out of the box, rDNA does not know where the DNA .jar file is located. We need to register DNA with rDNA to use them together. To do that, one needs to save the DNA .jar file to the working directory of the current R session and then initialize DNA as follows (with dna-2.0-beta19.jar in this example):

```
dna_init("dna-2.0-beta19.jar")
```

After initializing DNA, we can open the DNA graphical user interface from the R command line:

```
dna_gui()
```

Alternatively, we can provide the file name of a local DNA database as an argument, and the database will be opened in DNA. For example, we could open the sample.dna database that is provided for download on GitHub under Releases:

```
dna_gui("sample.dna")
```

For this to work, the database file has to be saved in the working directory of the R session, or the path needs to be provided along with the file name.

In addition to opening the GUI, we will want to retrieve networks and attributes from DNA. For this to happen, a connection with a DNA database must first be established using the dna_connection function:

```
conn <- dna_connection("sample.dna")

## Error in .jnew("dna.export/Exporter", "sqlite", infile, "", "", verbose):  java.lang.ClassNotFound
```

The dna_connection function accepts a file name of the database including full or relative path (or, alternatively, a connection string to a remote mySQL database) and optionally the login and password for the database (in case a remote mySQL database is used). Details about the connection can be printed by calling the resulting object called conn.

After initializing DNA and establishing a connection to a database, we can now retrieve data from DNA. We will start with a simple example of a two-mode network from the sample database. To compute the network matrix, the connection that we just established must be supplied to the dna_network function:

```
nw <- dna_network(conn)

## Error in inherits(obj, "jobjRef"):  object 'conn' not found
```

The resulting matrix can be plotted using visualization functions from the statnet suite of packages:

```
library("statnet")
gplot(nw)

## Error in as.edgelist.sna(dat, force.bipartite = (gmode == "twomode")):  object 'nw'
not found
```

It is also easily possible to retrieve the attributes of a variable, for example the colors and types of organizations, using the dna_attributes function:

```
at <- dna_attributes(conn)

## Error in inherits(obj, "jobjRef"):  object 'conn' not found
```

The result is a data frame with organizations in the rows and one column per organizational attribute. The next section will provide usage examples of both the dna_network and the dna_attributes functions.

2.2 Retrieving networks and attributes

This section will explore the dna_network function and facilities for retrieving attributes in more detail. The dna_network function has a number of arguments, which resemble the export options in the DNA export window. The help page for the dna_network function provides details on these arguments. It can be opened using the command

```
help("dna_network")
```

We will start with a simple example: a one-mode congruence network of organizations in a policy debate. The `sample.dna` database is a small excerpt from a larger empirical research project that tries to map the ideological debates around American climate politics in the U.S. Congress over time. Details about the dataset from which this excerpt is taken are provided by Fisher et al. (2013a,b). Here, it suffice to say that the `sample.dna` file contains speeches from hearings in the U.S. Congress in which interest groups and legislators make statements about their views on climate politics. Accordingly, one should expect to find a polarized debate with environmental groups on one side and industrial interest groups on the other side. To compute a one-mode congruence network, the following code can be used:

```
congruence <- dna_network(conn,
                          networkType = "onemode",
                          statementType = "DNA Statement",
                          variable1 = "organization",
                          variable2 = "concept",
                          qualifier = "agreement",
                          qualifierAggregation = "congruence",
                          duplicates = "document")

## Error in inherits(obj, "jobjRef"):  object 'conn' not found
```

The result is an organization × organization matrix, where the cells represent on how many concepts any two actors (i.e., the row organization and the column organization) had the same issue stance (by values of the qualifier variable `agreement`).

The arguments of the `dna_network` function resemble the options in the DNA export window. Details on the various arguments of the function can be obtained by displaying the help page (`?dna_network`).

`statementType = "DNA Statement"` indicates which statement type should be used for the network export. In this case, the statement type `DNA Statement` contains the variables `organization`, `concept`, and `agreement`. The argument `qualifierAggregation = "congruence"` causes `rDNA` to count how often the unique elements of `variable1` have an identical value on the `qualifier` variable (here: `agreement`) when they refer to a concept (`variable2`).

If the algorithm finds duplicate statements within documents—i. e., statements containing the same organization, concept, and agreement pattern—, only one of them is retained for the analysis (`duplicates = "document"`).

The resulting matrix can be converted to a network object and plotted as follows:

```
nw <- network(congruence)

## Error in as.network(x, directed = directed, hyper = hyper, loops = loops, :  object
'congruence' not found

plot(nw,
     edge.lwd = congruence^2,
     displaylabels = TRUE,
     label.cex = 0.5,
     usearrows = FALSE,
     edge.col = "gray"
     )
```

```
## Error in plot(nw, edge.lwd = congruence^2, displaylabels = TRUE, label.cex = 0.5, :
object 'nw' not found
```

Here, we used the `edge.lwd` argument of the `plot.network` function to make the line width proportional to the strength of congruence between actors. We used squared edge weights to emphasize the difference between low and high edge weights. We also displayed the labels of the nodes at half the normal size, suppressed arrow heads, and changed the color of the edges to gray. More information about the visualization capabilities of the `network` and `sna` packages are provided by Butts (2008a,b, 2015).

The network is not particularly polarized. We can suspect that some of the concepts are not very contested. If they are supported by all actors, this may mask the extent of polarization with regard to the other concepts. From our experience with the dataset, we can tell in this particular case that the concept "There should be legislation to regulate emissions." is in fact very consensual. If everybody agrees to this concept, it obfuscates the real structure of the network. Therefore we should exclude it from the congruence network. To do that, we need to export and plot the congruence network again and use the `excludeValues` argument this time:

```
congruence <- dna_network(conn,
                          networkType = "onemode",
                          statementType = "DNA Statement",
                          variable1 = "organization",
                          variable2 = "concept",
                          qualifier = "agreement",
                          qualifierAggregation = "congruence",
                          duplicates = "document",
                          excludeValues = list("concept" =
                              "There should be legislation to regulate emissions."))

## Error in inherits(obj, "jobjRef"):  object 'conn' not found

nw <- network(congruence)

## Error in as.network(x, directed = directed, hyper = hyper, loops = loops, :  object
'congruence' not found

plot(nw,
     edge.lwd = congruence^2,
     displaylabels = TRUE,
     label.cex = 0.5,
     usearrows = FALSE,
     edge.col = "gray"
     )

## Error in plot(nw, edge.lwd = congruence^2, displaylabels = TRUE, label.cex = 0.5, :
object 'nw' not found
```

This reveals the structure of the actor congruence network. There are two camps revolving around environmental groups on the right and industrial interest groups and state actors on the left, with `Energy and Environmental Analysis, Inc.` taking a bridging position. The strongest belief congruence ties can be found within, rather than between, the coalitions.

Next, we should tweak the congruence network further by changing the appearance of the nodes. We can use the colors for the organization types saved in the database and apply them to the nodes in the network. We can also make the size of each node proportional to its activity. The `dna_attributes` function serves to retrieve these additional data from `DNA`. The result is a data frame with the relevant data for each organization in the `color` and `frequency` columns:

```
at <- dna_attributes(conn,
                     statementType = "DNA Statement",
                     variable = "organization")

## Error in inherits(obj, "jobjRef"):  object 'conn' not found

at

## Error in eval(expr, envir, enclos):  object 'at' not found
```

To use these data in the congruence network visualization, we can use the plotting facilities of the `plot.network` function:

```
plot(nw,
     edge.lwd = congruence^2,
     displaylabels = TRUE,
     label.cex = 0.5,
     usearrows = FALSE,
     edge.col = "gray",
     vertex.col = at$color,
     vertex.cex = at$frequency
     )

## Error in plot(nw, edge.lwd = congruence^2, displaylabels = TRUE, label.cex = 0.5, :
object 'nw' not found
```

This yields a clear visualization of the actor congruence network, with simultaneous display of the network structure includings its coalitions, the actors' activity in the debate, and actor types.

Another way to visualize a discourse network is a two-mode network visualization. To compute a two-mode network of organizations and concepts, the following code can be used:

```
affil <- dna_network(conn,
                     networkType = "twomode",
                     statementType = "DNA Statement",
                     variable1 = "organization",
                     variable2 = "concept",
                     qualifier = "agreement",
                     qualifierAggregation = "combine",
                     duplicates = "document")

## Error in inherits(obj, "jobjRef"):  object 'conn' not found
```

This creates a 7×6 matrix of organizations and their relations to concepts. The argument `networkType = "twomode"` is necessary because the rows and columns of the `affil` matrix should contain different variables. The arguments `variable1 = "organization"` and `variable2 = "concept"` define which variables should be used for the rows and columns, respectively. The arguments `qualifier = "agreement"` and `qualifierAggregation = "combine"` define how the cells of the matrix should be populated: `agreement` is a binary variable, and the `combine` option causes a cell to have a value of 0 if the organization never refers to the concept, 1 if the organization refers to the respective concept exclusively in a positive way, 2 if the organization refers to the concept exclusively in a negative way, and 3 if there are both positive and negative statements by the organization about the concept. `rDNA` reports on the `R` console what each combination means.

As in the previous example, the resulting network matrix can be converted to a `network` object (as defined in the `network` package). The colors of the edges can be stored as an edge attribute, and the resulting object can be plotted with different colors representing positive, negative, and ambivalent mentions.

```
nw <- network(affil, bipartite = TRUE)

## Error in as.network(x, directed = directed, hyper = hyper, loops = loops, :  object
'affil' not found

colors <- as.character(t(affil))

## Error in t(affil):  object 'affil' not found

colors[colors == "3"] <- "blue"

## Error in colors == "3":  comparison (1) is possible only for atomic and list types

colors[colors == "2"] <- "red"

## Error in colors == "2":  comparison (1) is possible only for atomic and list types

colors[colors == "1"] <- "green"

## Error in colors == "1":  comparison (1) is possible only for atomic and list types

colors <- colors[colors != "0"]

## Error in colors != "0":  comparison (2) is possible only for atomic and list types

set.edge.attribute(nw, "color", colors)

## Error in inherits(x, "network"):  object 'nw' not found

plot(nw,
     edge.col = get.edge.attribute(nw, "color"),
     vertex.col = c(rep("white", nrow(affil)),
                    rep("black", ncol(affil))),
     displaylabels = TRUE,
     label.cex = 0.5
     )

## Error in plot(nw, edge.col = get.edge.attribute(nw, "color"), vertex.col = c(rep("white",
:  object 'nw' not found
```

In this example, we first converted the two-mode matrix to a bipartite `network` object, then created a vector of colors for the edges (excluding zeros), and inserted this vector into the `network` object as an edge attribute. It was necessary to work with the transposed `affil` matrix (using the `t` function) because the `set.edge.attribute` function expects edge attributes in a row-wise order while the `as.character` function returns them in a column-wise order based on the `affil` matrix. Finally, we plotted the network object with edge colors and labels. In the visualization, we used white nodes for organizations and black nodes for concepts.

We can now see the opinions of all actors on the various concepts. The blue edge indicates that `Energy and Environmental Analysis, Inc.` has both positive and negative things to say about the concept `"Emissions legislation should regulate CO2`. This is why this organization acts as a bridge between the two camps in the congruence network. Furthermore, we can now see more clearly that the concept we omitted in the congruence network, `"There should be legislation to regulate emissions"`, is viewed positively by four organizations but still receives a negative mention by one actor. The green edges span both camps, and this caused additional connections between the two groups. The negative tie is ignored in the construction of the congruence network because conflicts are not counted and there is no second red tie to that concept.

References

Butts, C. T. (2008a). Social network analysis with `sna`. *Journal of Statistical Software*, 24(6):1–51.

Butts, C. T. (2008b). `network`: A package for managing relational data in `R`. *Journal of Statistical Software*, 24(2):1–36.

Butts, C. T. (2015). *`network`: Classes for Relational Data*. The Statnet Project (http://statnet.org). `R` package version 1.13.0.

Csardi, G. and Nepusz, T. (2006). The `igraph` software package for complex network research. *InterJournal, Complex Systems*, 1695(5):1–9.

Fisher, D. R., Leifeld, P., and Iwaki, Y. (2013a). Mapping the ideological networks of American climate politics. *Climatic Change*, 116(3):523–545.

Fisher, D. R., Waggle, J., and Leifeld, P. (2013b). Where does political polarization come from? Locating polarization within the U.S. climate change debate. *American Behavioral Scientist*, 57(1):70–92.

Goodreau, S. M., Handcock, M. S., Hunter, D. R., Butts, C. T., and Morris, M. (2008). A `statnet` tutorial. *Journal of Statistical Software*, 24(9):1–26.

Handcock, M. S., Hunter, D. R., Butts, C. T., Goodreau, S. M., Krivitsky, P. N., Bender-deMoll, S., and Morris, M. (2016). *`statnet`: Software Tools for the Statistical Analysis of Network Data*. The Statnet Project (http://www.statnet.org). `R` package version 2016.9.

Handcock, M. S., Hunter, D. R., Butts, C. T., Goodreau, S. M., and Morris, M. (2008). `statnet`: Software tools for the representation, visualization, analysis and simulation of network data. *Journal of Statistical Software*, 24(1):1–11.

Leifeld, P. (2017). *`rDNA`: `R` bindings for the `Discourse Network Analyzer`*. `R` package version 2.0.1.

Leifeld, P., Cranmer, S. J., and Desmarais, B. A. (2017a). *Temporal Exponential Random Graph Models with `btergm`: Estimation and Bootstrap Confidence Intervals*. Forthcoming.

Leifeld, P., Cranmer, S. J., and Desmarais, B. A. (2017b). *`xergm`: Extensions of Exponential Random Graph Models*. `R` package version 1.8.2.

Maechler, M., Rousseeuw, P., Struyf, A., Hubert, M., and Hornik, K. (2017). *`cluster`: Cluster Analysis Basics and Extensions.* `R` package version 2.0.6.

`R` Core Team (2014). *`R`: A Language and Environment for Statistical Computing.* `R` Foundation for Statistical Computing, Vienna, Austria.

Urbanek, S. (2016). *`rJava`: Low-Level `R` to `Java` Interface.* R package version 0.9-8.

Wickham, H. and Chang, W. (2016). *`devtools`: Tools to Make Developing `R` Packages Easier.* `R` package version 1.12.0.