**Example data, nodes & edges**

We will use the following packages that we need to load at first:

library(dplyr)

library(purrr)

library(igraph)

library(ggplot2)

library(ggraph)

library(RColorBrewer)

Let’s create some exemplary data. Let’s say we have 4 groups *a, b, c, d* and 40 nodes with the node IDs 1 to 40. Each node can belong to several groups but it must not belong to any group. An example would be the following data:

group\_a <- 1:5 # nodes 1 to 5 in group a

group\_b <- 1:10 # nodes 1 to 10 in group b

group\_c <- c(1:3, 7:18) # nodes 1 to 3 and 7 to 18 in c

group\_d <- c(1:4, 15:25) # nodes 1 to 4 and 15 to 25 in d

members <- data\_frame(id = c(group\_a, group\_b, group\_c, group\_d, 26:40),

group = c(rep('a', length(group\_a)),

rep('b', length(group\_b)),

rep('c', length(group\_c)),

rep('d', length(group\_d)),

rep(NA, 15))) # nodes 26 to 40 do not

# belong to any group

An excerpt of the data:

> members

id group

1 a

2 a

[...]

5 a

1 b

2 b

[...]

38 NA

39 NA

40 NA

Now we can create the edges of the graph, i.e. the connections between the nodes. All nodes within a group are connected to each other. Additionally, all nodes are connected with one “super node” (as mentioned in the introduction). In our example data, we pick node ID 1 to be this special node. Let’s start to create our edges by connecting all nodes to node 1:

edges <- data\_frame(from = 1, to = 2:max(members$id), group = NA)

We also denote here, that these edges are not part of any group memberships. We’ll handle these group memberships now:

within\_group\_edges <- members %>%

split(.$group) %>%

map\_dfr(function (grp) {

id2id <- combn(grp$id, 2)

data\_frame(from = id2id[1,],

to = id2id[2,],

group = unique(grp$group))

})

edges <- bind\_rows(edges, within\_group\_edges)

At first, we split the members data by their group which produces a list of data frames. We then use map\_dfr from the purrr package to handle each of these data frames that are passed as grp argument. grp$id contains the node IDs of the members of this group and we use combn to create the pair-wise combinations of these IDs. This will create a matrix id2id, where the columns represent the node ID pairs. We return a data frame with the from-to ID pairs and a group column that denotes the group to which these edges belong. These “within-group edges” are appended to the already created edges using bind\_rows.

> edges

from to group

1 2 NA

1 3 NA

1 4 NA

[...]

23 24 d

23 25 d

24 25 d

**Plotting with *ggraph***

We have our edges, so now we can create the graph with igraph and plot it using the ggraph package:

Library(igraph)

Library(ggraph)

g <- graph\_from\_data\_frame(edges, directed = FALSE)

ggraph(g) +

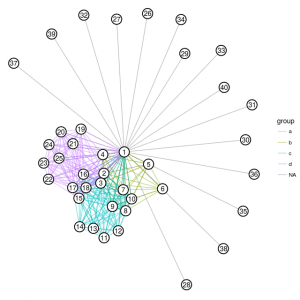
geom\_edge\_link(aes(color = group), alpha = 0.5) + # different edge color per group

geom\_node\_point(size = 7, shape = 21, stroke = 1,

fill = 'white', color = 'black') +

geom\_node\_text(aes(label = name)) + # "name" is automatically generated from the node IDs in the edges

theme\_void()

[](https://i2.wp.com/datascience.blog.wzb.eu/wp-content/uploads/10/2018/05/grouped_net_ggraph1.png?ssl=1)

Not bad for the first try, but the layout is a bit unfortunate, giving too much space to nodes that don’t belong to any group.

We can tell igraph’s layout algorithm to tighten the non-group connections (the gray lines in the above figure) by giving them a higher weight than the within-group edges:

# give weight 10 to non-group edges

edges <- data\_frame(from = 1, to = 2:40,

weight = 10, group = NA)

within\_group\_edges <- members %>%

split(.$group) %>%

map\_dfr(function (grp) {

id2id <- combn(grp$id, 2)

# weight 1 for within-group edges

data\_frame(from = id2id[1,],

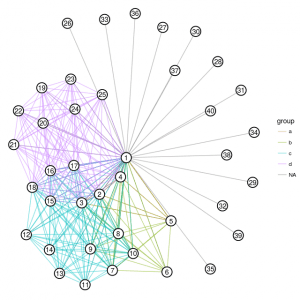
to = id2id[2,],

weight = 1,

group = unique(grp$group))

})

We reconstruct the graph g and plot it using the same commands as before and get the following:

[](https://i2.wp.com/datascience.blog.wzb.eu/wp-content/uploads/10/2018/05/grouped_net_ggraph2.png?ssl=1)

The nodes within groups are now much less cluttered and the layout is more balanced.

**Plotting with *igraph***

A problem with this type of plot is that connections within smaller groups are sometimes hardly visible (for example group *a* in the above figure). The plotting functions of igraph allow an additional method of highlighting groups in graphs: Using the parameter mark.groups will construct convex hulls around nodes that belong to a group. These hulls can then be highlighted with respective colors.

Examples

## Not run:

# plotting a simple ring graph, all default parameters, except the layout

g <- make\_ring(10)

g$layout <- layout\_in\_circle

plot(g)

tkplot(g)

rglplot(g)

# plotting a random graph, set the parameters in the command arguments

g <- barabasi.game(100)

plot(g, layout=layout\_with\_fr, vertex.size=4,

vertex.label.dist=0.5, vertex.color="red", edge.arrow.size=0.5)

# plot a random graph, different color for each component

g <- sample\_gnp(100, 1/100)

comps <- components(g)$membership

colbar <- rainbow(max(comps)+1)

V(g)$color <- colbar[comps+1]

plot(g, layout=layout\_with\_fr, vertex.size=5, vertex.label=NA)

# plot communities in a graph

g <- make\_full\_graph(5) %du% make\_full\_graph(5) %du% make\_full\_graph(5)

g <- add\_edges(g, c(1,6, 1,11, 6,11))

com <- cluster\_spinglass(g, spins=5)

V(g)$color <- com$membership+1

g <- set\_graph\_attr(g, "layout", layout\_with\_kk(g))

plot(g, vertex.label.dist=1.5)

# draw a bunch of trees, fix layout

igraph\_options(plot.layout=layout\_as\_tree)

plot(make\_tree(20, 2))

plot(make\_tree(50, 3), vertex.size=3, vertex.label=NA)

tkplot(make\_tree(50, 2, mode="undirected"), vertex.size=10,

vertex.color="green")

## End(Not run)

At first, we need to create a list that maps each group to a vector of the node IDs that belong to that group:

group\_ids <- lapply(members %>% split(.$group), function(grp) { grp$id })

> group\_ids

$a

[1] 1 2 3 4 5

$b

[1] 1 2 3 4 5 6 7 8 9 10

[...]

Now we can create a color for each group using RColorBrewer:

Library(RcolorBrewer)

group\_color <- brewer.pal(length(group\_ids), 'Set1')

# the fill gets an additional alpha value for transparency:

group\_color\_fill <- paste0(group\_color, '20')

We plot it by using the graph object g that was generated before with graph\_from\_data\_frame:

par(mar = rep(0.1, 4)) # reduce margins

plot(g, vertex.color = 'white', vertex.size = 9,

edge.color = rgb(0.5, 0.5, 0.5, 0.2),

mark.groups = group\_ids,

mark.col = group\_color\_fill,

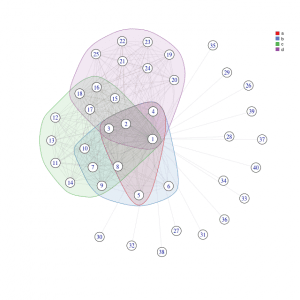
mark.border = group\_color)

legend('topright', legend = names(group\_ids),

col = group\_color,

pch = 15, bty = "n", pt.cex = 1.5, cex = 0.8,

text.col = "black", horiz = FALSE)

[](https://i1.wp.com/datascience.blog.wzb.eu/wp-content/uploads/10/2018/05/grouped_net_igraph1-1.png?ssl=1)

This option usually works well when you have groups that are more or less well separated, i.e. do not overlap too much. However, in our case there is quite some overlap and we can see that the shapes that encompass the groups also sometimes include nodes that do not actually belong to that group (for example node 8 in the above figure that is encompassed by group *a* although it does not belong to that group).

We can use a trick that leads the layout algorithm to bundle the groups more closely in a different manner: For each group, we introduce a “virtual node” (which will not be drawn during plotting) to which all the normal nodes in the group are tied with more weight than to each other. Nodes that only belong to a single group will be placed farther away from the center than those that belong to several groups, which will reduce clutter and wrongly overlapping group hulls. Furthermore, a virtual group node for nodes that do not belong to any group will make sure that these nodes will be placed more closely to each other.

We start by generating IDs for the virtual nodes:

# 4 groups plus one "NA-group"

virt\_group\_nodes <- max(members$id) + 1:5

names(virt\_group\_nodes) <- c(letters[1:4], NA)

This will give us the following IDs:

> virt\_group\_nodes

a b c d NA

41 42 43 44 45

We start to create the edges again by connecting all nodes to the “super node” with ID 1:

edges\_virt <- data\_frame(from = 1, to = 2:40, weight = 5, group = NA)

Then, the edges within the groups will be generated again, but this time we add additional edges to each group’s virtual node:

within\_virt %>% split(.$group) %>% map\_dfr(function (grp) {

group\_name <- unique(grp$group)

virt\_from <- rep(virt\_group\_nodes[group\_name], length(grp$id))

id2id <- combn(grp$id, 2)

data\_frame(

from = c(id2id[1,], virt\_from),

to = c(id2id[2,], grp$id), # also connects from virtual\_from node to each group node

weight = c(rep(0.1, ncol(id2id)), # weight between group nodes

rep(50, length(grp$id))), # weight that 'ties together' the group (via the virtual group node)

group = group\_name

)

})

edges\_virt <- bind\_rows(edges\_virt, within\_virt)

We add edges from all nodes that don’t belong to a group to another virtual node:

virt\_group\_na <- virt\_group\_nodes[is.na(names(virt\_group\_nodes))]

non\_group\_nodes <- (members %>% filter(is.na(group)))$id

edges\_na\_group\_virt <- data\_frame(from = non\_group\_nodes,

to = rep(virt\_group\_na,

length(non\_group\_nodes)),

weight = 10,

group = NA)

edges\_virt <- bind\_rows(edges\_virt, edges\_na\_group\_virt)

This time, we also create a data frame for the nodes, because we want to add an additional property is\_virt to each node that denotes if that node is virtual:

nodes\_virt <- data\_frame(id = 1:max(virt\_group\_nodes),

is\_virt = c(rep(FALSE, max(members$id)),

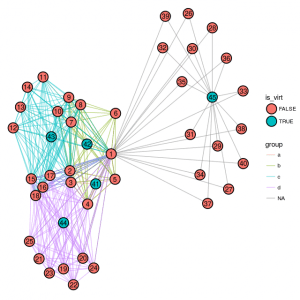
rep(TRUE, length(virt\_group\_nodes))))

We’re ready to create the graph now:

g\_virt <- graph\_from\_data\_frame(edges\_virt, directed = FALSE,

vertices = nodes\_virt)

To illustrate the effect of the virtual nodes, we can plot the graph directly and get a figure like this (virtual nodes highlighted in turquois):

[](https://i2.wp.com/datascience.blog.wzb.eu/wp-content/uploads/10/2018/05/grouped_net_ggraph_virt.png?ssl=1)

We now want to plot the graph without the virtual nodes, but the layout should nevertheless be calculated *with* the virtual nodes. We can achieve that by running the layout algorithm first and then removing the virtual nodes from both the graph and the generated layout matrix:

# use "auto layout"

lay <- layout\_nicely(g\_virt)

# remove virtual group nodes from graph

g\_virt <- g\_virt - vertices(virt\_group\_nodes)

# remove virtual group nodes' positions from the layout matrix

lay <- lay[-virt\_group\_nodes, ]

It’s important to pass the layout matrix now with the layout parameter to produce the final figure:

plot(g\_virt, layout = lay, vertex.color = 'white', vertex.size = 9,

edge.color = rgb(0.5, 0.5, 0.5, 0.2),

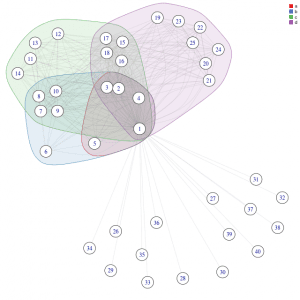
mark.groups = group\_ids, mark.col = group\_color\_fill,

mark.border = group\_color)

legend('topright', legend = names(group\_ids), col = group\_color,

pch = 15, bty = "n", pt.cex = 1.5, cex = 0.8,

text.col = "black", horiz = FALSE)

[](https://i1.wp.com/datascience.blog.wzb.eu/wp-content/uploads/10/2018/05/grouped_net_igraph_virt.png?ssl=1)

We can see that output is less cluttered and nodes that belong to the same groups are bundled nicely while nodes that do not share the same groups are well separated. Note that the respective edge weights were found empirically and you will probably need to adjust them to achieve a good graph layout for your data.