# Supplemental Material

This material shows sample code for the analyses described in the main text. The code loads in ego network tie and attribute data, turns them into ego networks, then shows plotting options and calculation of network statistics.

The up-to-date code and sample data are stored at a Github repository, https://github.com/ATraxLab/sample-ego-NWs.

```
library(tidyverse)
library(igraph)
library(ggraph)
library(egor)
library(purrr) # not essential for networks, used in plotting
```

Start by reading in the data:

```
ties_support <- read_csv("ties_support_sample.csv")
node_attr <- read_csv("node_attr_sample.csv")
alter_attr <- read_csv("alter_attr_sample.csv")
ego_attr <- read_csv("ego_attr_sample.csv")
```

These objects hold:

- *ego_attr*: interview number, name, and work sector (three true/false columns) for six interview participants, the **egos**.
- *alter_attr*: interview number, name, group status (true/false), and number of mentions for each alter, followed by true/false columns for relationship categories (partners, friends, etc.), support types, and gaps in support. The final columns give Dissonance, Proximity, Proximity2 and Mentions2 (values adjusted to prevent plotting errors), career stage, and overall relationship category as discussed in the paper. The last column is the participant (ego) name for each **alter**.
- *node_attr*: combines the information in the above two objects. Fields that are not meaningful for a given row are left blank.
- *ties_support*: data for each **tie** with interview number, two name columns (for the "source" and "destination" of a tie), total number of support and gaps in support mentioned, and whether the tie was for support, gaps in support, no support (neutral), or between two alters.

```
ego_attr
```

```
## # A tibble: 6 x 5
##   interview Name          Academia Industry Government
##       <dbl> <chr>         <lgl>    <lgl>    <lgl>
## 1        32 Participant A TRUE     FALSE    TRUE
## 2        36 Participant B TRUE     FALSE    FALSE
## 3        46 Participant C TRUE     FALSE    FALSE
## 4        47 Participant D TRUE     FALSE    TRUE
## 5        69 Participant E FALSE    TRUE     FALSE
## 6       101 Participant F FALSE    TRUE     TRUE
```

```r
ties_support
```

```
## # A tibble: 443 x 6
##    interview Name1   Name2   support gaps_support color
##        <dbl> <chr>   <chr>     <dbl>        <dbl> <chr>
##  1        32 Alter 2 Alter 9       0            0 alters
##  2        32 Alter 5 Alter 9       0            0 alters
##  3        32 Alter 6 Alter 9       0            0 alters
##  4        32 Alter 7 Alter 9       0            0 alters
##  5        32 Alter 8 Alter 9       0            0 alters
##  6        32 Alter 9 Alter 2       0            0 alters
##  7        32 Alter 9 Alter 5       0            0 alters
##  8        32 Alter 9 Alter 6       0            0 alters
##  9        32 Alter 9 Alter 7       0            0 alters
## 10        32 Alter 9 Alter 8       0            0 alters
## # i 433 more rows
```

For each interview, we make an igraph object for subsequent plotting and analysis:

```r
# Set up empty list
n_interview <- length(unique(ties_support$interview))
gr_list <- vector(mode = 'list', length = n_interview)

# Make list of igraph objects, one per interview
for (i in seq(n_interview)) {
  # Map loop counter (i) to interview number (index); allows for skipped/omitted
  #  interviews, but they should be in ascending order
  index <- unique(ties_support$interview)[i]

  # Retrieve tie/edge and node attribute data for just that interview
  df_ties <- ties_support %>%
    filter(interview == index) %>%
    select(-interview)
  df_attr <- node_attr %>%
    filter(interview == index) %>%
    select(-interview)

  # Make igraph object
  gr_list[[i]] <- graph_from_data_frame(df_ties, vertices = df_attr,
                                        directed = FALSE)
}

summary(gr_list[[6]])
```

```
## IGRAPH 23b3060 UN-- 15 115 --
## + attr: name (v/c), Academia (v/l), Industry (v/l), Government (v/l),
## | Groups (v/l), Mentions (v/n), partners (v/l), friends (v/l), family
## | (v/l), other (v/l), subfield (v/l), groups.listservs (v/l),
## | conferences (v/l), affinity.group (v/l), undergraduate.program (v/l),
## | graduate.program (v/l), postdoc (v/l), past.job (v/l), current.job
## | (v/l), future.job (v/l), bosses.supervisors (v/l),
## | mentors..professional. (v/l), mentees.direct.reports (v/l),
```

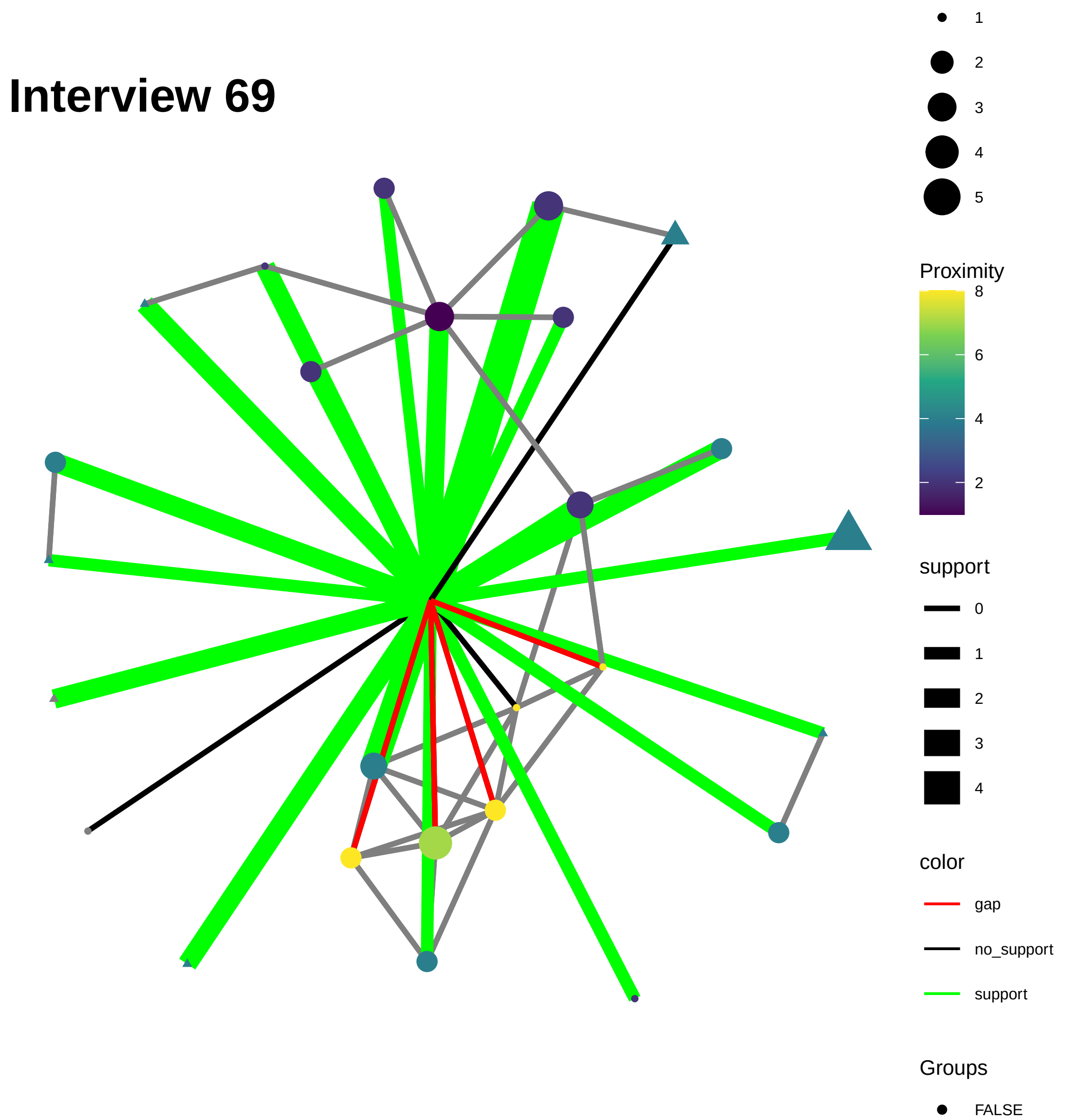
```
## | coworkers.peers (v/l), identity.based.support (v/l),
## | gap.identity.based.support (v/l), physical.closeness (v/l),
## | gap.physical.closeness (v/l), networking.support (v/l),
## | gap.networking.support (v/l), community.building.support (v/l),
## | gap.community.building.support (v/l), career.advice.support (v/l),
## | gap.career.advice.support (v/l), emotional.support (v/l),
## | gap.emotional.support (v/l), material.support (v/l),
## | gap.material.support (v/l), instrumental.support (v/l),
## | gap.instrumental.support (v/l), DEI.initiatives.policies (v/l),
## | gap.DEI.initiatives.policies (v/l), Dissonance (v/l), Proximity
## | (v/n), Proximity2 (v/n), Mentions2 (v/n), Participant.Name (v/c),
## | support (e/n), gaps_support (e/n), color (e/c)
```

### Plot with weighted support ties

The ggraph package can be used to display several kinds of the encoded network information at once. The loop below sets up a ggraph object for each ego network, using the count of support types and tie category to set line width and color, sizing nodes by Mentions, coloring by Proximity, and using a different shape for group nodes.

After the loop, a sample plot is printed, corresponding to Figure 7 in the paper.

```r
# Set up empty list
ggr_list <- vector(mode = 'list', length = n_interview)

for (i in seq(n_interview)) {
  int_num <- unique(ties_support$interview)[i]

  g <- ggraph(gr_list[[i]], layout = 'fr') +
    geom_edge_link2(aes(edge_width = support, edge_colour = color)) +
    scale_edge_color_manual(values = c("alter" = "blue", "gap" = "red",
                                       "support" = "green",
                                       "no_support" = "black")) +
    geom_node_point(aes(size = Mentions, colour = Proximity,
                        shape = Groups)) +
    ggtitle(paste("Interview", int_num)) +
    theme_graph(base_family = "", base_size = 8) +
    scale_color_continuous(type = "viridis") #colorblind-friendly, NA gray
  ggr_list[[i]] <- g
}

print(ggr_list[[5]])
```




## Plots using egor package

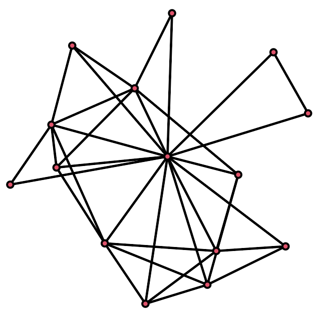
The egor package offers several plotting options tailored to personal networks. The first of these is through an interface to the networks package which has more streamlined defaults for visually comparing several ego networks at once.

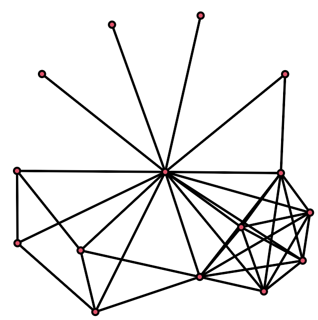
First, the code below needs to call the `egor()` function, which takes three inputs:

- a data frame of alter attributes, for which we use `alter_attr`,
- a data frame of ego attributes (`ego_attr`),
- and a data frame of alter-alter ties (`alter_connects`).

It returns an egor object, which is a list of three data frames holding essentially the input frames (with key ID columns renamed to standard titles).

```r
# Filter the support ties frame to just alter-alter ties, then drop color column
alter_connects <- ties_support %>%
  filter(color == "alters") %>%
  select(-color)

# Make egor objects
egr <- egor(alters = alter_attr, egos = ego_attr,
            aaties = alter_connects,
            ID.vars = list(ego = "interview", alter = "Name",
                           source = "Name1", target = "Name2"))

plot_indices <- c(1:2)
purrr::walk(as_network(egr[plot_indices], directed = FALSE,
                       include.ego = TRUE, ), plot)
```





Even though the egr object is a list of three data frames, accessing individual elements by `egr[1]` will pull up the network info for a single ID: an ego frame (now a 1-row tibble), an alter frame (only their alters), and an aatie frame (only their alter-alter ties).

```r
egr[1]
```

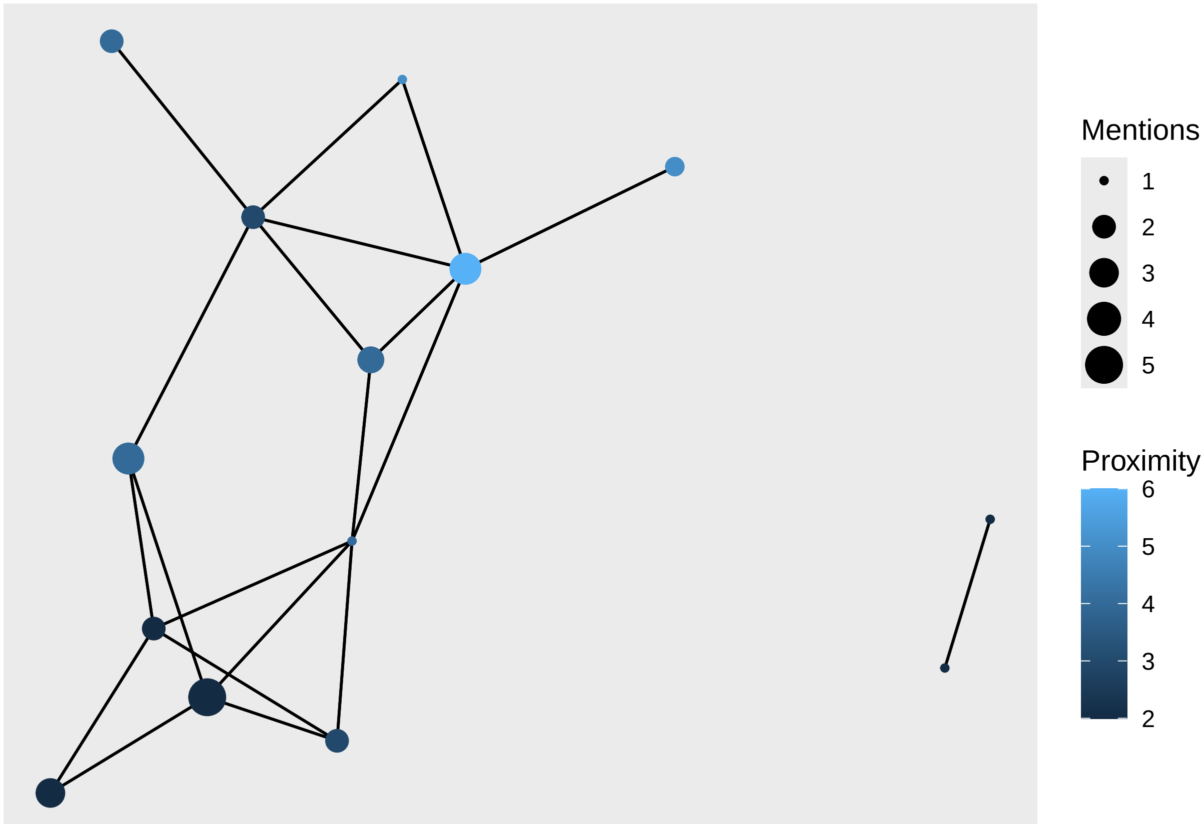
```
## # EGO data (active): 1 x 5
##   .egoID Name          Academia Industry Government
## * <chr>  <chr>         <lgl>    <lgl>    <lgl>
## 1 32     Participant A TRUE     FALSE    TRUE
```

```
## # ALTER data: 14 x 47
##   .altID  .egoID Groups Mentions partners friends family other subfield
## * <chr>   <chr>   <dbl>    <dbl> <lgl>    <lgl>   <lgl>  <lgl> <lgl>
## 1 Alter 1 32         NA        2 FALSE    FALSE   FALSE  FALSE FALSE
## 2 Alter 2 32         NA        1 FALSE    FALSE   FALSE  FALSE FALSE
## 3 Alter 3 32         NA        1 FALSE    TRUE    FALSE  FALSE FALSE
## # i 11 more rows
## # i 38 more variables: groups.listservs <lgl>, conferences <lgl>,
## #   affinity.group <lgl>, undergraduate.program <lgl>, graduate.program <lgl>,
## #   postdoc <lgl>, past.job <lgl>, current.job <lgl>, future.job <lgl>,
## #   bosses.supervisors <lgl>, mentors..professional. <lgl>,
## #   mentees.direct.reports <lgl>, coworkers.peers <lgl>,
## #   identity.based.support <lgl>, gap.identity.based.support <lgl>, ...
## # AATIE data: 40 x 5
##   .egoID .srcID  .tgtID  support gaps_support
## * <chr>  <chr>   <chr>     <dbl>        <dbl>
## 1 32     Alter 2 Alter 9       0            0
## 2 32     Alter 5 Alter 9       0            0
## 3 32     Alter 6 Alter 9       0            0
## # i 37 more rows
```

### Network statistics

To calculate network statistics, we first remove the egos. This can be done straightforwardly by the egor package, where the default behavior of the `as_igraph` function is to remove the ego.

```
gr_noego <- as_igraph(egr)

#summary(gr_noego[[1]])

g <- (ggraph(gr_noego[[1]], layout = 'fr') +
        geom_edge_link() +
        geom_node_point(aes(size = Mentions, colour = Proximity)))
print(g)
```




Next, we build the table of network statistics.

```
# Many of the stats come from summarizing node_attr columns, so do that first
node_summaries <- node_attr %>%
  mutate(Proximity = as.numeric(Proximity)) %>%
  group_by(interview) %>%
  summarise(total_groups = sum(Groups, na.rm = TRUE),
            total_dissonance = sum(Dissonance, na.rm = TRUE),
            avg_proximity = mean(Proximity, na.rm = TRUE),
            std_proximity = sd(Proximity, na.rm = TRUE),
            avg_mentions = mean(Mentions, na.rm = TRUE),
            std_mentions = sd(Mentions, na.rm = TRUE),
            total_support = sum(identity.based.support,
                                physical.closeness,
                                networking.support,
                                community.building.support,
                                career.advice.support,
                                emotional.support,
                                material.support,
                                instrumental.support,
                                DEI.initiatives.policies,
                                na.rm = TRUE),
            total_gaps_support = sum(gap.identity.based.support,
                                     gap.physical.closeness,
                                     gap.networking.support,
                                     gap.community.building.support,
```

```
                                    gap.career.advice.support,
                                    gap.emotional.support,
                                    gap.material.support,
                                    gap.instrumental.support,
                                    gap.DEI.initiatives.policies,
                                    na.rm = TRUE))

# Put together node_attr summary stats with network calculations and compute
#  new columns
nw_stats <- tibble(interview = unique(ties_support$interview),
                   total_nodes = map_int(gr_noego, vcount)) %>%
  left_join(node_summaries) %>%
  mutate(percent_groups = total_groups / total_nodes,
         percent_dissonance = total_dissonance / total_nodes,
         support_to_nodes_ratio = total_support / total_nodes,
         gaps_support_to_nodes_ratio = total_gaps_support / total_nodes,
         density = map_dbl(gr_noego, function(g) edge_density(simplify(g))),
         diameter = map_dbl(gr_noego, diameter),
         components = map_int(gr_noego, count_components),
         largest_clique = map_int(gr_noego, clique_num),
         cliques = map_int(gr_noego,
                           function(g) length(max_cliques(g, min = 3))),
         transitivity = map_dbl(gr_noego, function(g) transitivity(g)))

# Move a couple of columns to put percentages by total values
nw_stats <- nw_stats %>%
  relocate(percent_groups, .after = total_groups) %>%
  relocate(percent_dissonance, .after = total_dissonance)

nw_stats
```

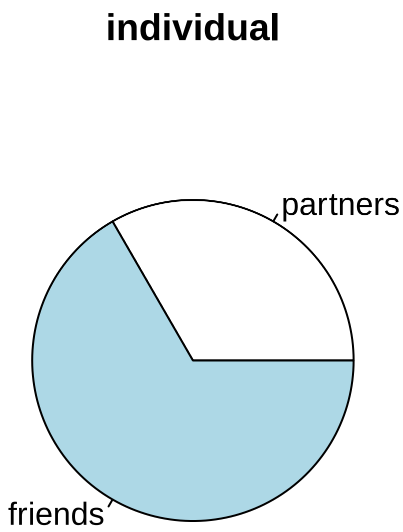
```
## # A tibble: 6 x 20
##   interview total_nodes total_groups percent_groups total_dissonance
##       <dbl>       <int>        <int>          <dbl>            <int>
## 1        32          14            4         0.286                 0
## 2        36          14            2         0.143                 0
## 3        46          17            6         0.353                 0
## 4        47          16            3         0.188                 0
## 5        69          26            7         0.269                 5
## 6       101          14            1         0.0714                0
## # i 15 more variables: percent_dissonance <dbl>, avg_proximity <dbl>,
## #   std_proximity <dbl>, avg_mentions <dbl>, std_mentions <dbl>,
## #   total_support <int>, total_gaps_support <int>,
## #   support_to_nodes_ratio <dbl>, gaps_support_to_nodes_ratio <dbl>,
## #   density <dbl>, diameter <dbl>, components <int>, largest_clique <int>,
## #   cliques <int>, transitivity <dbl>
```

### Alter composition plots

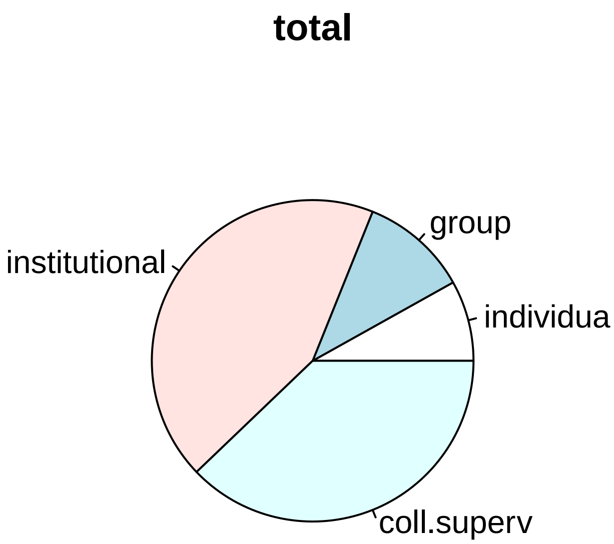
Below are examples of plotting the composition of the alters by a single interview. The code also works if a subset of alters (e.g., by workforce sector) is selected instead of a single ego.

**By alter count**

```
# Subset method 1: single interview
int <- 36
int_attr <- alter_attr %>% filter(interview == int)

# Count alters of each type in these relational categories, also total
individual <- c(sum(int_attr$partners), sum(int_attr$friends),
                sum(int_attr$family), sum(int_attr$other))

group <- c(sum(int_attr$subfield), sum(int_attr$groups.listservs),
           sum(int_attr$conferences), sum(int_attr$affinity.group))

institutional <- c(sum(int_attr$undergraduate.program),
                   sum(int_attr$graduate.program), sum(int_attr$postdoc),
                   sum(int_attr$past.job), sum(int_attr$current.job),
                   sum(int_attr$future.job))

coll.superv <- c(sum(int_attr$bosses.supervisors),
                 sum(int_attr$mentors..professional.),
                 sum(int_attr$mentees.direct.reports),
                 sum(int_attr$coworkers.peers))

total <- c(sum(individual), sum(group), sum(institutional), sum(coll.superv))

# Vectors with labels for each type and for total
indiv_lab <- c("partners", "friends", "family", "other")
group_lab <- c("subfield", "groups.listservs", "conferences", "affinity")
inst_lab <- c("undergrad", "grad", "postdoc", "past.job", "current.job",
              "future.job")
coll_lab <- c("boss.super", "mentor", "mentee", "coworker")
total_lab <- c("individual", "group", "institutional", "coll.superv")


# Plot pie charts
pie(individual[individual > 0], labels = indiv_lab, main = "individual")
```




```r
#pie(group[group > 0], labels = group_lab, main = "group")
#pie(institutional[institutional > 0], labels = inst_lab,
#    main = "institutional")
#pie(coll.superv[coll.superv > 0], labels = coll_lab,
#    main = "collegial.supervisorial")
pie(total, labels = total_lab, main = "total")
```

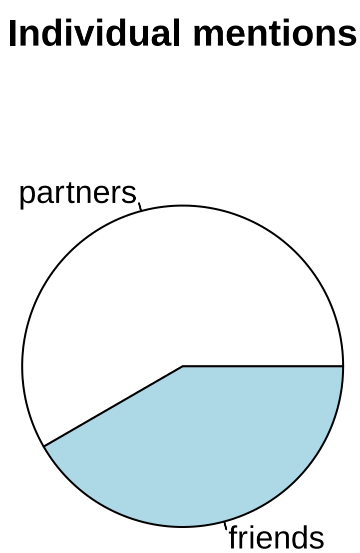



```r
par(mfrow = c(1, 1)) #how many plots to show per page
```

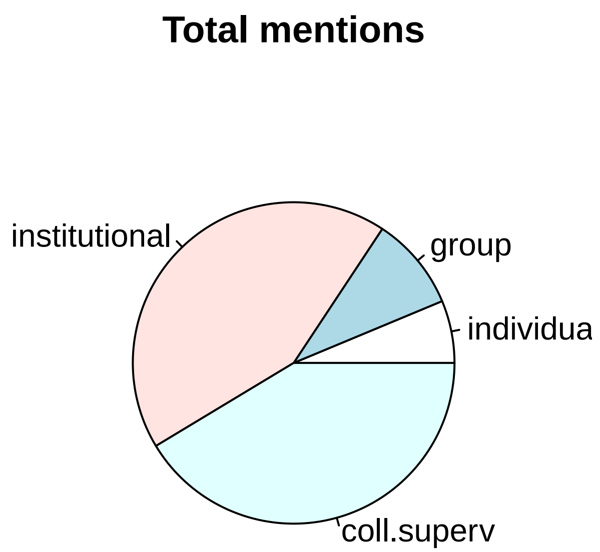
**By alter mention**

```r
# Subset method 1: single interview
int <- 36
int_attr <- alter_attr %>% filter(interview == int)

# Vectors with labels for each type and for total
# Must match column names exactly to make the code below easier
indiv_lab <- c("partners", "friends", "family", "other")
group_lab <- c("subfield", "groups.listservs", "conferences", "affinity.group")
inst_lab <- c("undergraduate.program", "graduate.program", "postdoc",
              "past.job", "current.job", "future.job")
coll_lab <- c("bosses.supervisors", "mentors..professional.",
              "mentees.direct.reports", "coworkers.peers")
total_lab <- c("individual", "group", "institutional", "coll.superv")

# Counts of mentions by alter category type
individual_men <- int_attr %>%
  select(Mentions, all_of(indiv_lab)) %>%
  pivot_longer(cols = all_of(indiv_lab)) %>%
  mutate(Mentions = Mentions * value) %>%   # only count if TRUE for this node
  summarise(n = sum(Mentions), .by = name) %>%
```

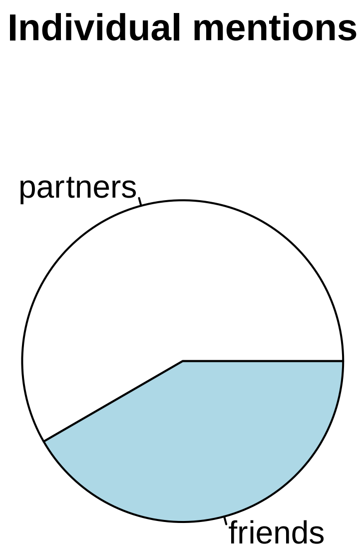
```r
  deframe() # turn into a named vector
individual_men <- individual_men[indiv_lab]  # restore original order

# That's a lot of lines, make a function
count_mentions <- function(df, labs) {
  mens <- df %>%
    select(Mentions, all_of(labs)) %>%
    pivot_longer(cols = all_of(labs)) %>%
    mutate(Mentions = Mentions * value) %>%   # only count if TRUE for this node
    summarise(n = sum(Mentions), .by = name) %>%
    deframe() # turn into a named vector
  mens <- mens[labs]  # restore original order
  return(mens)
}

group_men <- count_mentions(int_attr, group_lab)
institutional_men <- count_mentions(int_attr, inst_lab)
coll.superv_men <- count_mentions(int_attr, coll_lab)
total_men <- c(sum(individual_men), sum(group_men), sum(institutional_men),
               sum(coll.superv_men))
names(total_men) <- total_lab

pie(individual_men[individual_men > 0], main = "Individual mentions")
```



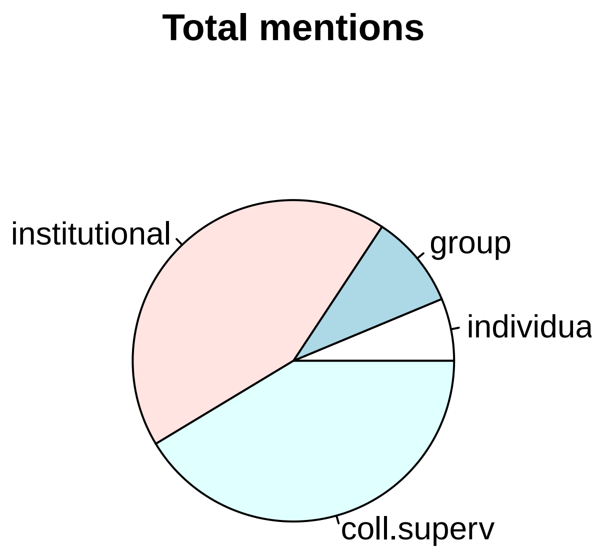
```r
#pie(group_men[group_men > 0], main = "Group mentions")
#pie(institutional_men[institutional_men > 0], main = "Institutional mentions")
#pie(coll.superv_men[coll.superv_men > 0],
#    main = "Collegial & supervisorial mentions")
pie(total_men[total_men > 0], main = "Total mentions")
```




```r
# Summation for subset of interviews

int2_attr <- alter_attr %>% filter(interview %in% c(34, 36))

# Alter mentions of each type by category, also total
individual_men <- count_mentions(int2_attr, indiv_lab)
group_men <- count_mentions(int2_attr, group_lab)
institutional_men <- count_mentions(int2_attr, inst_lab)
coll.superv_men <- count_mentions(int2_attr, coll_lab)
total_men <- c(sum(individual_men), sum(group_men), sum(institutional_men),
               sum(coll.superv_men))
names(total_men) <- total_lab

pie(individual_men[individual_men > 0], main = "Individual mentions")
```




```
#pie(group_men[group_men > 0], main = "Group mentions")
#pie(institutional_men[institutional_men > 0], main = "Institutional mentions")
#pie(coll.superv_men[coll.superv_men > 0],
#    main = "Collegial & supervisorial mentions")
pie(total_men[total_men > 0], main = "Total mentions")
```

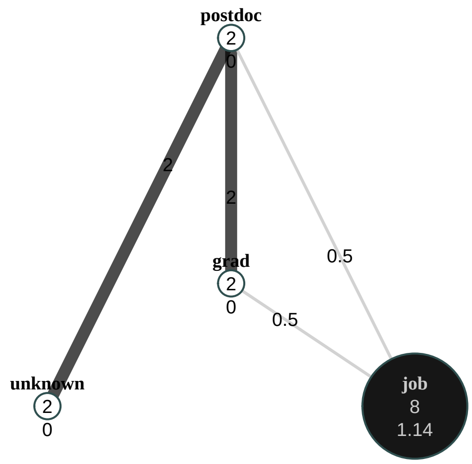



## Clustered graphs with the egor package

Clustering alters by career stage or relationship gives a condensed view that can facilitate visually comparing a larger number of networks.

The `clustered_graphs()` function takes as its inputs an egor object and the name of a factor variable (in the alter frame) to define the groups. It returns a list of igraph objects, which can then be plotted using the `vis_clustered_graphs()` function.

```
clus_graphs <- clustered_graphs(egr, "stage")

vis_clustered_graphs(clus_graphs[2],
                     node.size.multiplier = 1,
                     edge.width.multiplier = 3,
                     label.size = 0.6,
                     labels = TRUE)
```

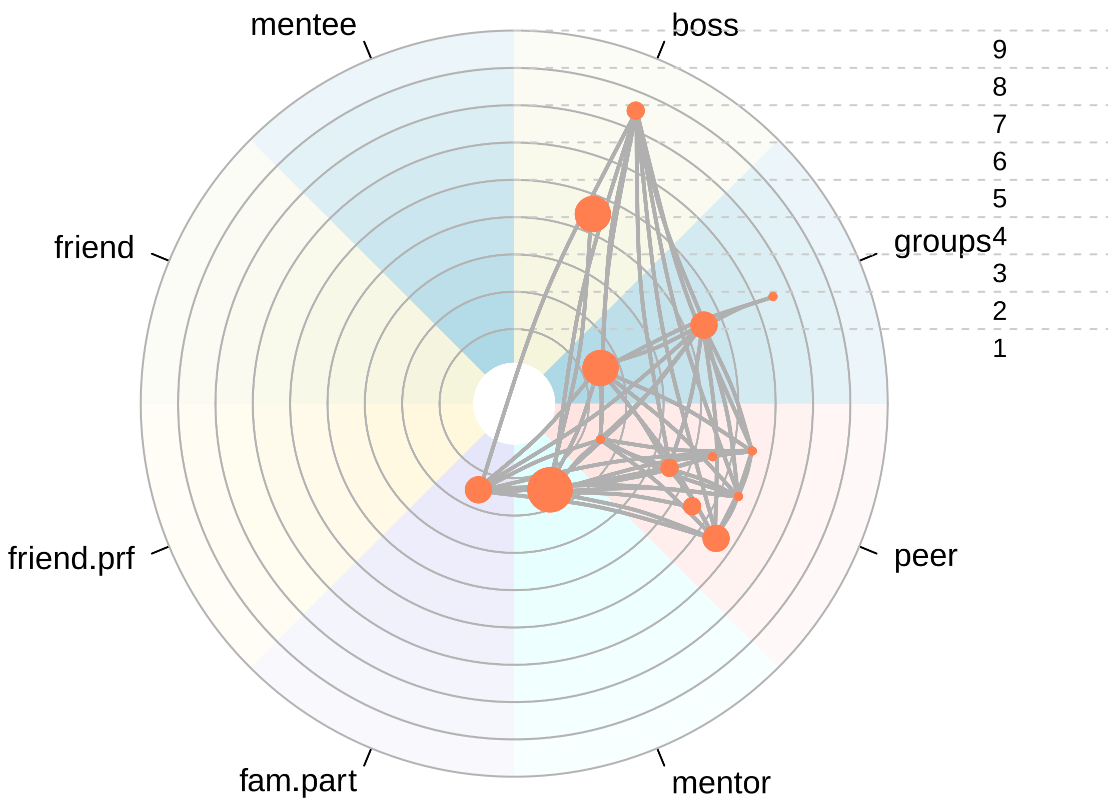



## Quadrant graphs with the egor package

Finally, networks can be plotted in sectors based on their category, and at distances scaled by some quantitative variable. The egor package provides the `plot_egograms` function, with relevant parameters for our sample case:

- `ego_no` is the ego number to plot.
- `venn_var` is the "rings" of the plot, here Proximity2 (Proximity with missing values coded to 9).
- `pie_var` is the "wedges" of the plot, here career stage.
- `ascending_inwards` is a true/false for the order of the Venn circles. In our example, it is `FALSE`, since higher values of Proximity mean farther away.
- `vertex_size_var` is the name of a column, or skip it to set all nodes to the same size.
- `vertex_label_var` is the name of a column, which defaults to "name" or can be set to NULL for no labels.

```
plot_egograms(egr,
              ego_no = 6, x_dim = 1, y_dim = 1,
              venn_var = "Proximity2", pie_var = "relationship",
              ascending_inwards = FALSE,
              vertex_size_var = "Mentions2", vertex_label_var = NULL)
```


mentee
boss
friend
groups
friend.prf
peer
fam.part
mentor
9
8
7
6
5
4
3
2
1