MEDWIN PUBLISHERS
Committed to Create Value for Researchers

Psychology & Psychological Research International Journal

Appendix A. Syntax and Instructions

- **Preprocessing Data**

The data are in the format of an edge per row. An edge is defined as a connection between nodes (residents) within the network. In this dataset, each edge is represented as a binary: 1 for having a connection and 0 for not having a connection. Each row includes the edge's source node, edge's target node, and the attributes the edge has. Every edge in a wave has a row dedicated to it. If an edge persists through multiple waves, it is considered a new edge in a new wave and has a row dedicated to it. If nodes left the network, a list of existing nodes in each wave is also included.

- **Package And Data Loading**

A variety of packages was utilized in the study. The key packages required for processing the data and animating the dynamic network include SNA and NDTV [12]. Other packages may be interchanged with preferred packages with similar functions. Data loading should be done to fit the proper file path required for a user's interface. Replace quotes "file path…" with your unique file paths depending on how the data is saved on your device.

```
    # Load necessary packages
packages <- c("data.table", "tidyverse", "keyring", "blastula", "dtplyr", "naniar", "net-work", "sna", "Matrix", "haven", "xtable", "ndtv", "networkDynamic")
needed_packages <- setdiff(packages, rownames(installed.packages()))
lapply(needed_packages, function(pkg) {if (!require(pkg, character.only = TRUE)) {in-stall.packages(pkg)}library(pkg, character.only = TRUE)})
```

Next, we filtered the edges into a binary format as mentioned above. In the directions below, we focus on friendship connections. Weight was assigned to give an edge if a participant rated another high enough to be considered a friend within the network (being rated a friend or close friend). The following steps are required for future checks and data assignments.

```
# Data Upload
e0 <- read.csv("file path to house 16 relationships")
House_16_Presence <- read.csv("file path to house 16 presence")
```

- **Creating Networkdynamic Object**

The following codes created an empty networkDynamic object. This object, part of the networkDynamic package [13], facilitates the querying, manipulation, and analysis of network data as it evolves, and is highly compatible with other network analysis tools in R. It supports dynamic visualization and efficient data import/export, making it an essential tool for analyzing time-varying network data. Before creating the object, a list of nodes that were to appear in the network was created, as well as the number of waves of which data were collected. The code created the networkDynamic object and all nodes were activated for all waves, nodes leaving the network were later deactivated in the code.

```
#Assigns weight to edges
e0$wt <- if_else(e0$frd < 3, 1, 0)
e1 <- e0[, 1:4]
#Helper function that finds a specific vertex
get_vertex_id_by_name <- function(net, name) {
        return (which(get.vertex.attribute(net, "vertex.names") == name))}
#Counts the number of waves
wave_num <- unique(e0$h)
#Creates a vector of wavetable names
wave_names <- paste0("w", 1:7)
# Uses mget to create a list of data frames
wave_tables <- mget(wave_names)
#Creates separate tables for each wave
for(wave in wave_num){
        wave_table_name <- paste("w", wave, sep = "")
          assign(wave_table_name, e1 %>% filter(h == wave))}
#Intialiaze a network and give each vertex a name
net_dynamic <- network.initialize(length(all_vertices), directed = TRUE)
set.vertex.attribute(net_dynamic, "vertex.names", all_vertices)
#Acctivate all vertices
activate.vertices(net_dynamic, onset = 0, terminus = length(wave_num) + 1)
#Get rid of the first 0-1 interval
deactivate.vertices(net_dynamic, onset = 0, length = 1)
```

- **Deactivating Vertices**

This section of code selects people who were not in the house for a specific wave from the list of nodes in that wave and generates a list of vertices that need to be deactivated. Then the code loops through each wave and deactivates them. When deactivating vertices, deactivate.edge needs to be set to false since the function would disable edges connected to

those vertices for all waves.

```
#Creates a separate list of people for those not present in each wave
for(wave in wave_num)
{cur_wave <- paste("w", wave, sep = "")
        cur_name <- paste("notIn_w", wave, sep = "")
        individual_ids <- House_16_Presence %>%
        filter(!!sym(cur_wave) == 0) %>%
        pull(ID)
        assign(cur_name, individual_ids)}

#Deactivate vertices who are not in the house at the current wave
for(wave in wave_num)
{in_name <- paste("notIn_w", wave, sep = "")
        cur_active <- get(in_name)

#Loops through the list of people not in the house
for(i in 1:length(cur_active))
{cur_id <- get_vertex_id_by_name(net_dynamic, cur_active[[i]])
        deactivate.vertices(net_dynamic, onset = wave, length = 1, v = cur_id, deacti-vate.edges =FALSE)}}
```

- **Activating Edges**

In this section, the code looped through each wave and activates the valid edges for those waves. When trying to find an edge by its source and target, the vertices have their persis-tent ID (PID) which is a constant between waves. The PID is not the same as the vertex name, which is shown on the final animation and raw data, therefore the PID needs to first be located. Each edge has its PID as well, once located, the wave in which an edge is active can be edited.

```
# Activates valid edges at each wave
for(wave in 1:length(wave_num)){

#Finds the current wave being run
valid_edges_table_name <- paste("validEdge_w", wave, sep = "")
valid_edges <- get(valid_edges_table_name)

#Activates edges that are present during current wave
for(i in 1:nrow(valid_edges))
{source_vertex <- which(get.vertex.attribute(net_dynamic, "vertex.names") == val-id_edges[i, "source"])
  target_vertex <- which(get.vertex.attribute(net_dynamic, "vertex.names") == val-id_edges[i, "target"])

#Checks if the vertex exists
if(length(source_vertex) > 0 && length(target_vertex) > 0) {

#Checks if the edge already exists, adds new edge, and activates vertex if not
existing_edge_ids <- get.edgeIDs(net_dynamic, v = source_vertex, alter = tar-get_vertex)
if(length(existing_edge_ids) == 0){
add.edge(net_dynamic, tail = source_vertex, head = target_vertex, activate = FALSE)
activate.edges(net_dynamic, onset = wave, terminus = wave, e = get.edgeIDs(net_dynamic, v = source_vertex, alter = target_vertex))}

#Activates an existing edge
activate.edges(net_dynamic, onset = wave, terminus = wave, e = get.edgeIDs(net_dynamic, v = source_vertex, alter = target_vertex))}}}
```

- **Dynamic Attribute**

Continuing with the next section of code gives dynamic attributes to vertices, specifically color to a specific vertex. Even if there is only one vertex that has the attribute, every vertex needs a default value for the attribute because the rendering package needs it. In the case of this code, the default color is cyan.

```
vertex_colors <- rep("cyan", network.size(net_dynamic))
names(vertex_colors) <- get.vertex.attribute(net_dynamic, "vertex.names")
vertex_colors["1228"] <- "red"
set.vertex.attribute(net_dynamic, "vertex.color", vertex_colors)
```

- **Rendering**

The final section of code rendered the networkDynamic object using the NDTV package. Aesthetic attributes were changed, and if they were not included as part of the parameters, a default value was filled by the package itself.

```
render.d3movie(net_dynamic,
 displaylabels = TRUE,
 vertex.col = "vertex.color",
 edge.lwd = 4, #Value of visual edge size
 vertex.cex = 12, #Value of visual vertex size
 label = "vertex.names",
 output.mode = "html",
 filename = "dynamic_network_visualization.html")
```

Appendix B. Visualizations, Data, and Open Access Code

Visualization for Friendship: https://asikora3.github.io/Social-Network-Animation/dynamic_network_visualization_FRD.html

Visualization for Advice-Seeking: https://asikora3.github.io/Social-Network-Animation/dynamic_network_visualization_ADV.html

Visualization for Loaning: https://asikora3.github.io/Social-Network-Animation/dynamic_network_visualization_ln.html

Data-Relationships:https://github.com/Asikora3/Social-Network-Animation/blob/main/House%2016.csv

Data-Presence:https://github.com/Asikora3/Social-Network-Animation/blob/main/House%2016%20Presence.csv

GitHub for all data, code, and syntax: https://github.com/Asikora3/Social-Network-Animation