Semantic Network Analysis (SemNA): A Tutorial on Preprocessing, Estimating, and Analyzing Semantic Networks

PsyArXiv Preprint

Alexander P. Christensen
Department of Neurology
University of Pennsylvania
Philadelphia, PA, 19104
alexpaulchristensen@gmail.com

Yoed N. Kenett
Faculty of Industrial Engineering & Management
Technion Israel Institute of Technology
Haifa, 3200003, Israel
yoedk@technion.ac.il

October 5, 2020

Abstract

To date, the application of semantic network methodologies to study cognitive processes in psychological phenomena has been limited in scope. One barrier to broader application is the lack of resources for researchers unfamiliar with the approach. Another barrier, for both the unfamiliar and knowledgeable researcher, is the tedious and laborious preprocessing of semantic data. In this article, we aim to minimize these barriers by offering a comprehensive semantic network analysis pipeline (preprocessing, estimating, and analyzing networks), and an associated R tutorial that uses a suite of R packages to accommodate this pipeline. Two of these packages, *SemNetDictionaries* and *SemNetCleaner*, promote an efficient, reproducible, and transparent approach to preprocessing verbal fluency data. The third package, *SemNeT*, provides methods and measures for analyzing and statistically comparing semantic networks via a point-and-click graphical user interface. Using real-world data, we present a start-to-finish pipeline from raw data to semantic network analysis results. This article aims to provide resources for researchers, both the unfamiliar and knowledgeable, that reduce some of the barriers for conducting semantic network analysis.

***K*eywords** semantic networks · verbal fluency · semantic memory · network science

Introduction

In recent years, network science has become an increasingly popular approach to study psychological and cognitive constructs, such as psychopathology, language, memory, creativity, personality traits, and learning (Baronchelli, Ferrer-i-Cancho, Pastor-Satorras, Chater, & Christiansen, 2013; Borge-Holthoefer & Arenas, 2010; Borsboom & Cramer, 2013; Cramer et al., 2012; Fried & Cramer, 2017; Karuza, Thompson-Schill, & Bassett, 2016; Siew et al., 2019). Network science is based on mathematical graph theory, which provides quantitative methods to represent and investigate complex systems as networks (Baronchelli et al., 2013; Siew et al., 2019).

A network is comprised of nodes (circles) that represent the basic units of a system, and edges (lines) that represent the relations between the units. Network science methodologies provide a powerful quantitative approach for modeling cognitive structures such as semantic memory (i.e., memory of word meanings, categorizations of concepts and facts, and knowledge about the world; Jones, Willits, Dennis, & Jones, 2015; Steyvers & Tenenbaum, 2005) and mental lexicon (i.e., word meaning, pronunciation, and syntactic characteristics; Stella, Beckage, Brede, & De Domenico, 2018; Wulff et al., 2019; for a review, see Siew et al., 2019). In these semantic network models, nodes represent concepts in memory and edges represent the

similarity, co-occurrence, or strength of the associations between them (Collins & Loftus, 1975). The nodes of a semantic network depend on the task; for example, they can be category exemplars (verbal fluency), associations to cue words (free association), or cue words whose similarities are rated (similarity judgments; De Deyne, Kenett, Anaki, Faust, & Navarro, 2016; Siew et al., 2019). The present paper will focus exclusively on semantic networks estimated from verbal fluency tasks.

Despite the rise of network analysis in psychology and cognitive science, there are a couple barriers that stand in the way of psychologists applying network analysis to semantic data. One barrier is that there is a general lack of resources for how researchers can get started with conducting their own semantic network analysis (SemNA). Another barrier is that preprocessing semantic data (e.g., spell-checking, removing inappropriate responses, making exemplars homogeneous, formatting the data for SemNA) is tedious and time-consuming.

The goal of this paper is to minimize these barriers by introducing a suite of three R packages—*SemNetDictionaries*, *SemNetCleaner*, and *SemNeT*—that are designed to facilitate a comprehensive pipeline for preprocessing, estimating, and analyzing semantic networks (Figure 1). While we focus only on semantic networks estimated from verbal fluency data (see below), our R packages are flexible and provide a generalizable approach for processing data from other types of tasks. This paper is organized by integrating a detailed discussion on each step of the SemNA pipeline with an associated R tutorial. Our tutorial demonstrates how these R packages can be integrated into a single seamless pipeline; however, we emphasize that the pipeline can be modular such that researchers can perform one step (e.g., preprocessing) following our tutorial but then export their data to other software to perform other steps (e.g., network estimation, statistical analyses). As a whole, the tutorial represents a start-to-finish pipeline of SemNA (i.e., from raw data to statistical analyses on networks).

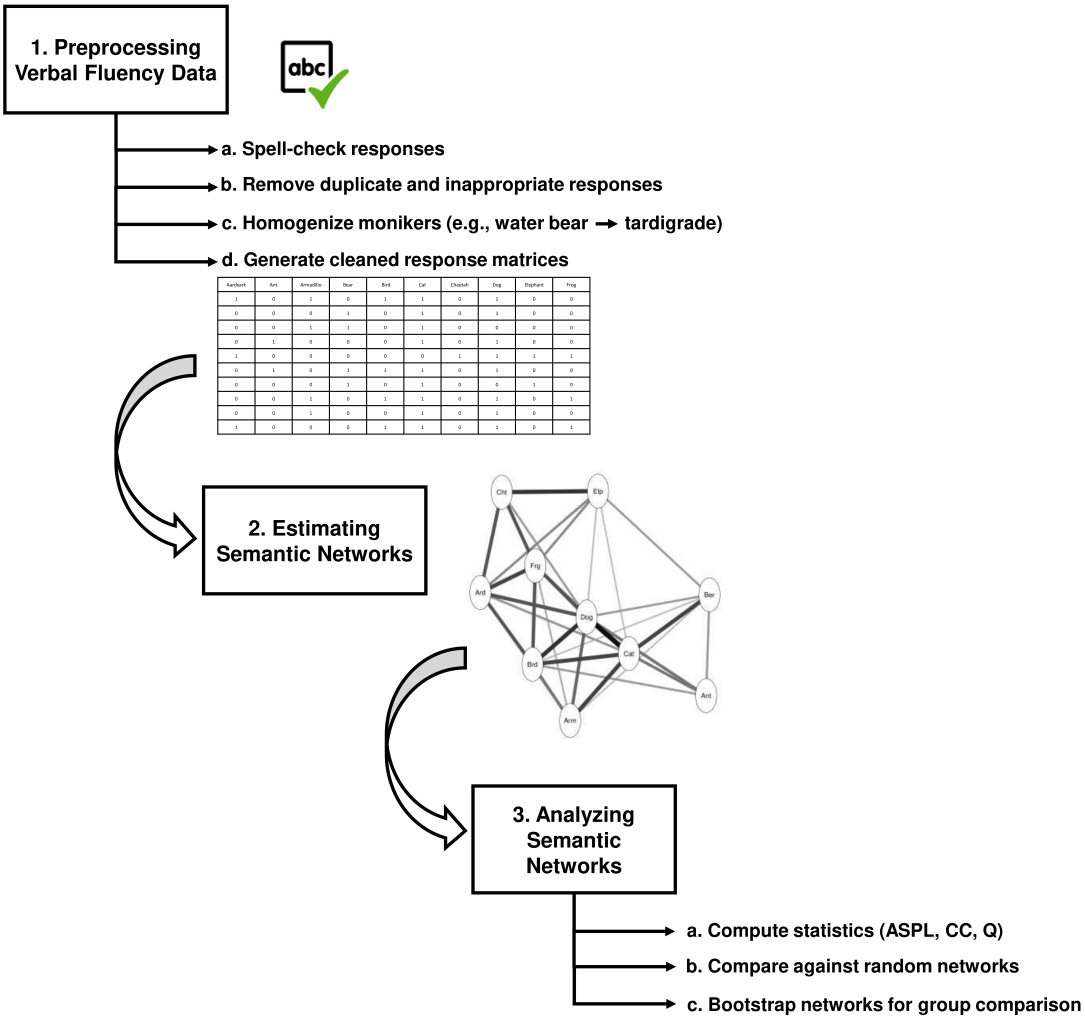

Figure 1: A step-by-step depiction of the semantic network analysis (SemNA) pipeline

Semantic Networks Estimated from Verbal Fluency Data

One popular method to estimate semantic networks is to use verbal fluency data (Siew et al., 2019; Zemla & Austerweil, 2018). Verbal fluency is a classic neuropsychiatric measure (Ardila, Ostrosky-Solís, & Bernal, 2006) where participants are asked to generate, in a short amount of time (e.g., 60 seconds), category members of a specific category. These categories can be semantic (e.g., animals) or phonological (e.g., words that start with 's'; Bousfield & Sedgewick, 1944). One strength of using verbal fluency tasks to study semantic memory is that they are quick to administer. In addition, some verbal fluency categories, such as animals, have a well-defined taxonomy that shows minor differences across different languages and cultures (e.g., animal kingdom; Ardila et al., 2006; Goñi et al., 2011).

To get from the raw verbal fluency data that participants have provided (usually typed) to statistical analyses on semantic networks, three main steps are required: preprocessing the raw data, estimating networks from the preprocessed data, and statistically analyzing the networks (Figure 1). Preprocessing the raw data includes spell-checking responses, removing duplicate and inappropriate responses (perseverations and intrusions, respectively), and homogenizing the naming scheme of exemplars into a common response (e.g.,

water bear, *moss piglets*, and *tardigrade* into *tardigrade*). After, networks are estimated from the preprocessed data using one of several network estimation approaches (Zemla & Austerweil, 2018). Finally, these estimated networks are statistically analyzed and compared using different statistical approaches (e.g., difference from random networks, bootstrap, random walks, spreading activation). Below, we discuss each step of the pipeline and provide associated R code. Our tutorial will follow the pipeline using a one specific network estimation method and a couple statistical approaches that are available in the *SemNeT* package.

Getting Started with the SemNA Pipeline in R

We used R version 4.0.2 and RStudio version 1.3.1073 to complete the pipeline. There are several R packages that need to be installed to setup the pipeline, which require R version 3.6.0 or higher to be installed. The main packages are provided below (see SI 1 for R session information):

```
# Install packages
install.packages(c("SemNetDictionaries", "SemNetCleaner",
                   "SemNeT"), dependencies = c("Imports", "Suggests"))

# Load packages
library(SemNetDictionaries)
library(SemNetCleaner)
library(SemNeT)
```

Additional packages are also necessary for using the point-and-click graphical user interface used in this tutorial (see SI 2). Moreover, for Linux users, additional installations are required for setting up Shiny server, which is necessary for the point-and-click graphical user interface (see https://github.com/rstudio/shiny-server/wiki/Building-Shiny-Server-from-Source). The reader can also follow along with the tutorial using the supplemental tutorial R script, which contains all the code chunks in this tutorial (SI 3). An additional template R script is also provided for readers interested in using this tutorial on their own data (SI 4). Both the tutorial and template R scripts are available for download from the Open Science Framework (`https://osf.io/hqxtc/`).

The tutorial will walk through the SemNA pipeline and its accompanying R packages, providing a step-by-step analysis of a verbal fluency dataset from one of our previously published papers (Christensen et al., 2018b). In this paper, we examined a personality trait, openness to experience, and its relationship to semantic network structure. This dataset includes 516 participants who completed the animals verbal fluency task (1 min.) and two measures of openness to experience (Big Five Aspects Scale; DeYoung, Quilty, & Peterson, 2007, and NEO–FFI–3; McCrae & Costa, 2007). The participants were evenly split into two groups based on low and high scores of openness to experience (both N's = 258). The raw verbal fluency data is stored in the *SemNetCleaner* package and can be accessed using the following R code:

```
# Raw verbal fluency data from 'SemNetCleaner'
data("open.animals")

# Structure of the data
head(open.animals)
```

The dataset is called `open.animals`, which is a data frame object that has 516 rows, corresponding to each participant and 38 columns corresponding to variables. The first column is the group membership variable that denotes which group each participant belongs (`1` = low openness to experience and `2` = high openness to experience). The second column contains each participant's openness to experience score. The third column is the participant's unique identifier variable (ID). The rest of the columns (4–38) contain each participant's responses or the animal names the participants typed. Demographic and procedure details can be found in Christensen et al. (2018b). This verbal fluency dataset will be used as our example for the tutorial.

1. Preprocessing Verbal Fluency Data

Regardless of how verbal fluency data is acquired, whether experimenter recorded or participant typed, several coding errors are likely to exist and need to be resolved before subsequent processing. It's necessary, for example, to determine whether responses are perseverations (given by a participant more than once),

intrusions (inappropriate exemplars for the category; e.g., animals category: *tasselled wobbegong*, *eucalyptus platypus*, *sugar bear*; a shark, small tree, and fictional character, respectively), or monikers (e.g., colloquial names or common misspellings) of a category exemplar (e.g., *water bear*, *moss piglet*, *tardigrade*). Other potential errors include spelling errors, compound responses (i.e., responses where a space is missing between responses; e.g., *guineapig*), and continuous strings (i.e., multiple responses entered as a single response). After resolving these errors, the data still need to be processed into a format that can be used for the estimation of networks. In addition, commonly used behavioral measures such as perseverations and instrusions should be counted and extracted from the processed data.

These issues motivated the creation of two R packages called, *SemNetDictionaries* and *SemNetCleaner*. The main purpose of these two packages is to automate and streamline the preprocessing of verbal fluency data while also minimizing the potential for human error. The *SemNetDictionaries* package assists *SemNetCleaner* by housing exemplar dictionary and moniker glossaries for verbal fluency categories. The *SemNetCleaner* package is the main package that focuses on the preprocessing of semantic data. We first introduce the *SemNetDictionaries* package because it has several functions that are integrated into the *SemNetCleaner* package.

1.1. *SemNetDictionaries* Package

The *SemNetDictionaries* package contains several functions that facilitate the management and loading of dictionaries into *SemNetCleaner*. These dictionaries are used to spell-check and auto-correct the raw verbal fluency data in the preprocessing stage. In addition, this package includes a function that allows the user to create their own custom dictionaries, enabling them to save their own dictionaries for future use.

1.1.1. Predefined dictionaries

The predefined dictionaries contained in the package currently include verbal fluency categories of *animals*, *fruits*, *vegetables*, and *jobs* as well as synonym categories of *hot* and *good* (Table 1). A *general* dictionary also accommodates phonological fluency tasks with the use of single letters (e.g., words that start with 'f') as well as the potential for more general text cleaning needs such as free association tasks (e.g., Nelson, McEvoy, & Schreiber, 2004) that are common in the semantic network literature. For each category and synonym dictionary, there is an accompanying moniker glossary that is used to automatically convert monikers (e.g., *bear cat*) and common misspellings (e.g., *binterong*) into a homogeneous response (e.g., *binturong*). These categories were included because the authors had data for them; however, we note that there are other possible verbal fluency categories.

Table 1: Dictionaries in SemNetDictionaries

Dictionary	Entries	Type
animals	1,210	Category
fruits	488	Category
general	370,103	All-purpose
good	284	Synonym
hot	281	Synonym
jobs	1,471	Category
vegetables	284	Category

The development of these category and synonym dictionaries included searching *Google* for lists of category exemplars and Thesaurus.com for synonyms. Their development was also aided by responses from several hundred participants, who generated verbal fluency responses for some of these categories (Christensen et al., 2018b) as well as responses from other published (Zemla & Austerweil, 2018) and unpublished data. Finally, the general dictionary was added for phonological fluency tasks and general spell-checking purposes (e.g., free association tasks). This dictionary was retrieved from the *dwyl* Github repository for English words:

https://github.com/dwyl/english-words. Examples for how to load some of these dictionaries is provided below:

```
# Check for available dictionaries
dictionaries()
# Load 'animals' dictionary
load.dictionaries("animals")
# Load all words starting with 'f'
load.dictionaries("f")
# Load multiple dictionaries
load.dictionaries("fruits", "vegetables")
```

The `load.dictionaries` function loads as many dictionaries as are entered. The function alphabetically sorts and removes any duplicates found between the dictionaries. Thus, it returns an alphabetized vector the length of the unique words in the specified dictionaries.

1.1.2. Custom dictionaries

A notable feature of the *SemNetDictionaries* package is that users can define their own dictionaries using the `append.dictionary` function. With this function, users can create their own custom dictionaries or append predefined dictionaries so that they can be used for future data cleaning and preprocessing. To create your own dictionary, the following code can be used:

```
# Create a custom dictionary
append.dictionary("your", "words", "here",
                  "in", "quotations", "and",
                  "separated", "by", "commas",
                  dictionary.name = "example",
                  save.location = "choose")
```

All words that are entered in quotations and separated by commas will be input into a new custom dictionary. The name of the dictionary can be defined with the `dictionary.name` argument (e.g., `dictionary.name = "example"`). All dictionaries that are saved using this function have a specific file suffix (`*.dictionary.rds`), which allows the dictionaries to be found by the function `find.dictionaries()`. A user can also append a predefined dictionary (e.g., `animals.dictionary`) by including the dictionary name in the function:

```
# Append a pre-defined dictionary
append.dictionary(animals.dictionary,
                  "tasselled wobbegong",
                  dictionary.name = "new.animals",
                  save.location = "choose")
```

Appending a pre-defined dictionary does not overwrite it; instead, the user must save this dictionary somewhere on their computer (e.g., `save.location = "choose"`). Dictionaries from the *SemNetDictionaries* package and dictionaries stored using the `append.dictionary` function can be integrated into the *SemNetCleaner* package for preprocessing (we describe how this is done in the next section).

1.1.3. Moniker glossaries

To view the moniker glossaries, the user must directly access the data file included in *SemNetDictionaries*. This can be done using the following code:

```
# Load 'animals' monikers
animals.moniker
```

Similar to the dictionary files, all monikers included in the package will have a `*.moniker` suffix. Unlike the dictionaries, users are unable to save and load their monikers for future use in the *SemNetCleaner* package. Instead, users are encouraged to submit their spelling corrections via our GitHub page: https://github.com/AlexChristensen/SemNetDictionaries/issues/new/choose. In the next section, we explain the necessary output that should be submitted.

1.2. *SemNetCleaner* Package

The *SemNetCleaner* package houses several functions for the preprocessing of semantic data. The purpose of this package is to facilitate efficient and reproducible preprocessing of semantic data. Notably, other R packages perform similar functions (e.g., spell-checking, text mining) such as *hunspell* (Ooms, 2018), *qdap* (Rinker, 2020), and *tm* (Feinerer, Hornik, & Meyer, 2008). However, the *SemNetCleaner* package sets itself apart from these other packages by focusing specifically on commonly used tasks for SemNA (e.g., verbal fluency), which facilitates automated preprocessing.

The *SemNetCleaner* package applies several steps to preprocess raw verbal fluency data so that it is ready to be used for estimating semantic networks. These steps include spell-checking, verifying the accuracy of the spell-check, homogenizing responses, and obtaining a cleaned response matrix for network estimation. To initialize this process, the following code must be run:

```
# Run 'textcleaner'
clean <- textcleaner(data = open.animals[,-c(1:2)], miss = 99,
                     partBY = "row", dictionary = "animals")
```

In the above code, we are removing the first two columns (group and score variable): `[,-c(1,2)]` because they are not relevant for our analysis. The only variables that should be entered into `textcleaner` are an optional column for participant's IDs and their respective verbal fluency data.

`textcleaner` is the main function that handles the data preprocessing in *SemNetCleaner* (for argument descriptions, see Table 2). For input into `data`, it's strongly recommended that the user input the full verbal fluency dataset and not data already separated into groups. If verbal fluency responses are already separated, then they will need to be inputted and preprocessed separately. Therefore, it's preferable to separate the preprocessed data into groups at a later stage of the SemNA pipeline.

Table 2: `textcleaner` Arguments

Argument	Description
`data`	A matrix or data frame object that contains the participants' IDs and semantic data
`miss`	A number or character that corresponds to the symbol used for missing data. The default is set to `99`
`partBY`	Specifies whether participants are across the rows (`"row"`) or down the columns (`"col"`)
`dictionary`	Specifies which dictionaries from *SemNetDictionaries* should be used (more than one is possible). If no dictionary is chosen, then the `"general"` dictionary is used
`add.path`	The path to a directory where the user's saved dictionaries are stored. Allows for an expedited search for user created dictionaries. Defaults to `NULL`
`continue`	A list object from a previous session of `textcleaner`. Allows progress to be continued after encountering an error or stopping the preprocessing step. Defaults to `NULL`

Two arguments in the `textcleaner` function pertain to importing custom dictionaries created using the `append.dictionary` function in *SemNetDictionaries*. The `dictionary` argument allows the user to type the name of one or more dictionaries to be used for spell-checking including the names of dictionaries the user has created. Dictionary names entered into the `dictionary` argument that are in the *SemNetDictionaries* package will be found immediately while dictionaries that were created using the `append.dictionary` function are searched for on the user's computer. This search finds any files with the `*.dictionary` suffix in R's environment, temporary files, and current working directory as well as the user's desktop and downloads folders. Users can also specify a path to a directory where dictionaries are saved using the `add.path` argument.

This latter argument is most useful when storing custom dictionaries in a single directory. Users can use `add.path = "choose"` to open an interactive directory explorer to navigate to a folder where there are saved dictionaries.

1.2.1. Spell-check

By running the above code, `textcleaner` will start preprocessing the data immediately. The first step of `textcleaner` is to automatically spell-check and attempt to correct all responses. The processes of the automated spell-check are printed to the user (Figure 2). `textcleaner` first attempts to detect whether there is a unique identifier variable (ID) that was included in the data. The first message to print in Figure 2, `IDs refer to variable: 'ID'`, tells the user what variable the IDs in the output will refer to. In our example data, there was a variable named `ID`. If an ID variable is not provided, then the IDs in the output will refer to the row numbers (and the message will notify the user that this is the case).

```
IDs refer to variable: 'ID'

Identifying correctly spelled responses...done.
(472 of 829 unique responses still need to be corrected)

Singularizing responses...done.
(399 unique responses still need to be corrected)

Auto-correcting common misspellings and monikers...done.

Attempting to auto-correct the remaining 270 responses individually...done.
(69 unique responses still need to be corrected)

Parsing multi-word responses...done

Automated spell-checking complete.
About 38 responses still need to be manually spell-checked

Press ENTER to start manual spell-check...
```

Figure 2: Automated spell-check and correction processes in textcleaner

The next five messages inform the user about the automated spell-checking processes. The first process separates the correctly spelled responses that were identified in the dictionary from responses not found in the dictionary. The second process checks for responses that are correctly spelled but are plural forms (e.g., *cats*) and converts them to their singular forms (e.g., *cat*). The third process checks for common misspellings and monikers and converts them into homogeneous responses. The fourth process attempts to automatically correct each remaining response (270 responses in our example) by individually evaluating the similarity between each response and responses in the dictionary using two edit distances (or measures of (dis)similarity): Damerau-Levenshtein distance (DL) (Damerau, 1964; Levenshtein, 1966) and QWERTY distance. Finally, the fifth process attempts to parse responses that are more than one word (e.g., *guinea pig*). This last process may also separate responses that were accidentally typed as a single response by the participant as we show next (Figure 3).

By pressing `ENTER`, the user can proceed to the manual spell-check where responses that were not able to be automatically corrected are checked by the user (in our example, only 38 need to be checked). The manual spell-check begins with more complicated cases first: responses where participants did not separate their responses—that is, when participants were typing their responses and did not hit a key (e.g., `ENTER`) to separate their responses, producing a string of multiple responses as if it were one response. After these so-called "continuous string" responses are handled, then individual responses are checked. The manual spell-check uses an interactive menu where the user has complete control over how to correct responses. The first example of this menu is shown in Figure 3.

Figure 3: Example of manual spell-check interface

At first, the interface may be overwhelming; however, breaking down the menu will make clear that every option has its purpose and that the user is in total control over their spelling corrections. Moreover, detailed help can be provided at any time by pressing `H`.

Starting from the top, `Original response` refers to the response that the participant provided. In the example, the participant did not separate their responses by pressing `ENTER` when typing during the task and therefore the response appears as a string of all of their responses (Figure 3).

`Auto-corrected response` shows the participant's original response after it's been passed through the automated spell-checking process. Two things are worth noting in our example: (1) all the responses were correctly separated into individual responses and (2) *squiral* and *giraff* were automatically corrected to *squirrel* and *giraffe*, respectively.

`Response to manually spell-check` displays the *target* response that needs to be manually spell-checked. Note that this response is highlighted to draw attention to what `textcleaner` needs the user to manually check.

`Word options` are options that change *only* the target response (i.e., *catdog* in our example). `SKIP WORD` will keep the target response "as is" and move on to the next response, `ADD WORD TO DICTIONARY` will keep the target response "as is" and add it to a dictionary that can later be saved, `TYPE MY OWN WORD` allows the user to type their own spelling correction, `GOOGLE WORD` will open the user's default internet browser and "Google" the word, and `BAD WORD` will mark the target response as an inappropriate response.

In contrast, `String options` are options that change the entire auto-corrected string (i.e., `auto-corrected response`). `KEEP ORIGINAL` and `KEEP AUTO-CORRECT` will change the response back the original response or keep the auto-corrected response "as is" (respectively), `TYPE MY OWN STRING` allows the user to type response(s) that will replace the entire string (rather than just *catdog*), `GOOGLE STRING` will "Google" the entire string, and `BAD STRING` will mark all responses as inappropriate responses.

Next, `Response options` are ten responses from the dictionary that were most similarly spelled (based on DL distance) and operate as spell correction suggestions. Below these options are additional features that allow the user to "go back" to the previous response by pressing `B`, get detailed help for the response options that we just described by pressing `H`, and saving the user's progress and exiting the function by pressing `X`.

In Figure 3, we've gone ahead and made a selection from these options where we decided to type our own response to correct the target response. In this response, the participant did not hit the `SPACE` key to separate *cat* and *dog*. Multiple words can be entered when typing your own response by using a comma to separate responses. After entering our correction, a message appears to let us know how the response was changed: `'cat' 'dog'`.

Finally, it's important to reiterate that the interface depicted in Figure 3 represents the most complicated case: a response where the participant did not enter their responses separately. In most cases, the target response will not be a string of responses but rather a single response. In these cases, the interface will display the target response (i.e., `Response to manually spell-check`), `Word options`, and `Response options` only. The reader is encouraged to progress through the manual spell-check on their own. We have provided a list of our spell correction changes in our Supplementary Information (SI 5), so that the reader can reproduce the decisions that we've made.

As a final step of the spell-check process, the user will be provided with instructions to verify the automated spell-check. Verification takes place in an Excel-like spreadsheet in a window that opens up after the instructions. At the end of the `textcleaner` process, a message will print letting the user know what output they can obtain to submit their dictionary (if you saved one) and moniker glossaries to our GitHub page: `https://github.com/AlexChristensen/SemNetDictionaries/issues/new/choose`.

1.2.2. `textcleaner` Output

With the preprocessing step complete, there are several output objects of interest for basic verbal fluency analyses and the next step of network estimation (see Table 3). These objects are stored in a list object, which we designated in our example as `clean`.

Table 3: `textcleaner` Output Objects

Object	Nested Object	Description
`responses`	`original`	Original response matrix where uppercase letters were made to lowercase and white spaces before and after responses were removed
	`clean`	Spell-corrected response matrix where the ordering of the original responses are preserved. Inappropriate and duplicate responses have been removed
	`binary`	Binary response matrix where rows are participants and columns are responses. 1's are responses given by a participant and 0's are responses not given by a participant
`spellcheck`	`correspondence`	A matrix of all unique original responses provided by the participants ("from" column) and what these responses were changed to in the `textcleaner` process ("to" columns)
	`automated`	A matrix of only the unique original responses provided by participants ("from" column) that were changed ("to" columns) during the automated spell-check processes
	`manual`	A matrix of only the unique original responses provided by participants ("from" column) that were changed ("to" column) during the manual spell-check process
	`verified`	A list of changes during the verification process
`behavioral`	—	A data frame containing the number of perseverations, intrusions, and appropriate responses for each participant

These objects can be accessed using a dollar sign (e.g., `clean$responses`) and nested objects can be accessed within their parent object (e.g., `clean$responses$original`). For basic verbal fluency analyses, the object `clean$behavioral` contains a data frame with common behavioral measures of verbal fluency: perseverations (number of repeated responses), intrusions (number of inappropriate category exemplars), and appropriate responses. For network estimation, the objects `clean$responses$clean` and `clean$responses$binary` are of interest. The former is the cleaned verbal fluency responses in the order they were typed by the participant whereas the latter is a binary response matrix where participants are the rows and responses are the columns with 1's corresponding to responses a participant provided and 0's to responses they did not.

1.3. Exporting Data

Before exporting these matrices, it's necessary to separate our data into groups. To so, we'll first access the grouping variable in the original dataset but also change the values to better reflect what the group values represent (i.e., `1` = low openness to experience and `2` = high openness to experience):

```
# Accessing and changing 'Group' variable
group <- ifelse(open.animals$Group == 1, "Low", "High")

# Separate group data
low <- clean$responses$clean[which(group == "Low"),]
high <- clean$responses$clean[which(group == "High"),]
```

To export these matrices and data frames so that they can be used in other software, the `write.csv` function can be used by specifying the object and path with a file name:

```
# Save cleaned responses
write.csv(low, file = "low_cleaned_responses.csv", row.names = TRUE)
write.csv(high, file = "high_cleaned_responses.csv", row.names = TRUE)
```

In the above code, we created a file called "low_cleaned.responses.csv" and "high_cleaned.responses.csv," which saved the clean response matrix to .csv files. This file can then be exported into other software to estimate and analyze semantic networks. For transparency purposes, users may also export the `clean$spellcheck$manual` object for colleagues and reviewers to evaluate decisions made during the preprocessing step. As we noted before, we've provided this output in our Supplementary Information (SI 5).

Finally, the `convert2snafu` function will save the data in a format that is compatible with the SNAFU (Semantic Network and Fluency Utility) library (Zemla, Cao, Mueller, & Austerweil, 2020) in Python:

```
# Save SNAFU formatted data
convert2snafu(low, high, category = "animals")
```

This function accepts any number of groups from the same dataset and requires the `category` argument to be set to the verbal fluency category used. Currently, datasets must be converted one at a time so if multiple categories are preprocessed, then separate files should be created and merged to be used in SNAFU. This function will open an interactive directory explorer so the user can navigate to the desired save location as well as allow the user to input a file name. The function will handle the rest by formatting the data and saving the file as a .csv. Thus, this function allows users to seamlessly transition from this pipeline into the SNAFU graphical user interface, facilitating cross-software compatibility for network estimation methods and analyses.

1.4. Summary

In this section we described and demonstrated how the packages *SemNetDictionaries* and *SemNetCleaner* are used to facilitate efficient and reproducible preprocessing of verbal fluency data. In this process, the raw data have been spell-checked, duplicate and inappropriate responses have been removed, monikers have been converged into one response, and cleaned response matrices have been generated. In addition, we described how to export the cleaned response matrix so that other software can be used to estimate and analyze semantic networks.

2. Estimating Semantic Networks

Continuing with our SemNA pipeline, the next step is to estimate semantic networks with the *SemNeT* package. The *SemNeT* package offers two options for performing this step: R console code or a point-and-click graphical user interface using RStudio's Shiny platform. For most users, the Shiny application will be the most intuitive and straightforward to use. For more experienced R users, code to perform these analyses are provided in a script in our Supplementary Information (SI 6). We will continue our tutorial with the Shiny application.

2.1. Shiny Application

To open the application, the following code can be run:

```
# SemNeT Shiny Application
SemNeTShiny()
```

Running the above code will open a separate window pictured in Figure 4. If a `textcleaner` object and a `group` object (like the objects we created in the last section; `clean` and `group`, respectively) are in R's environment, then a drop-down menu and radio button to select these objects will appear (Figure 4, a and b, respectively). Other options that will always appear are "Browse..." buttons to import a response matrix and group variable from the user's computer (c. and d. in Figure 4, respectively). If no group variable is entered, then the Shiny application will assume that the data belong to one group. Buttons to see examples of the data structures are also available. The Shiny application will give preference to the imported response matrix and group variable if data are input into a. through d. Finally, the "Load Data" button (Figure 4) will load the data into the application. A message window will pop-up letting the user know that their data was loaded successfully.

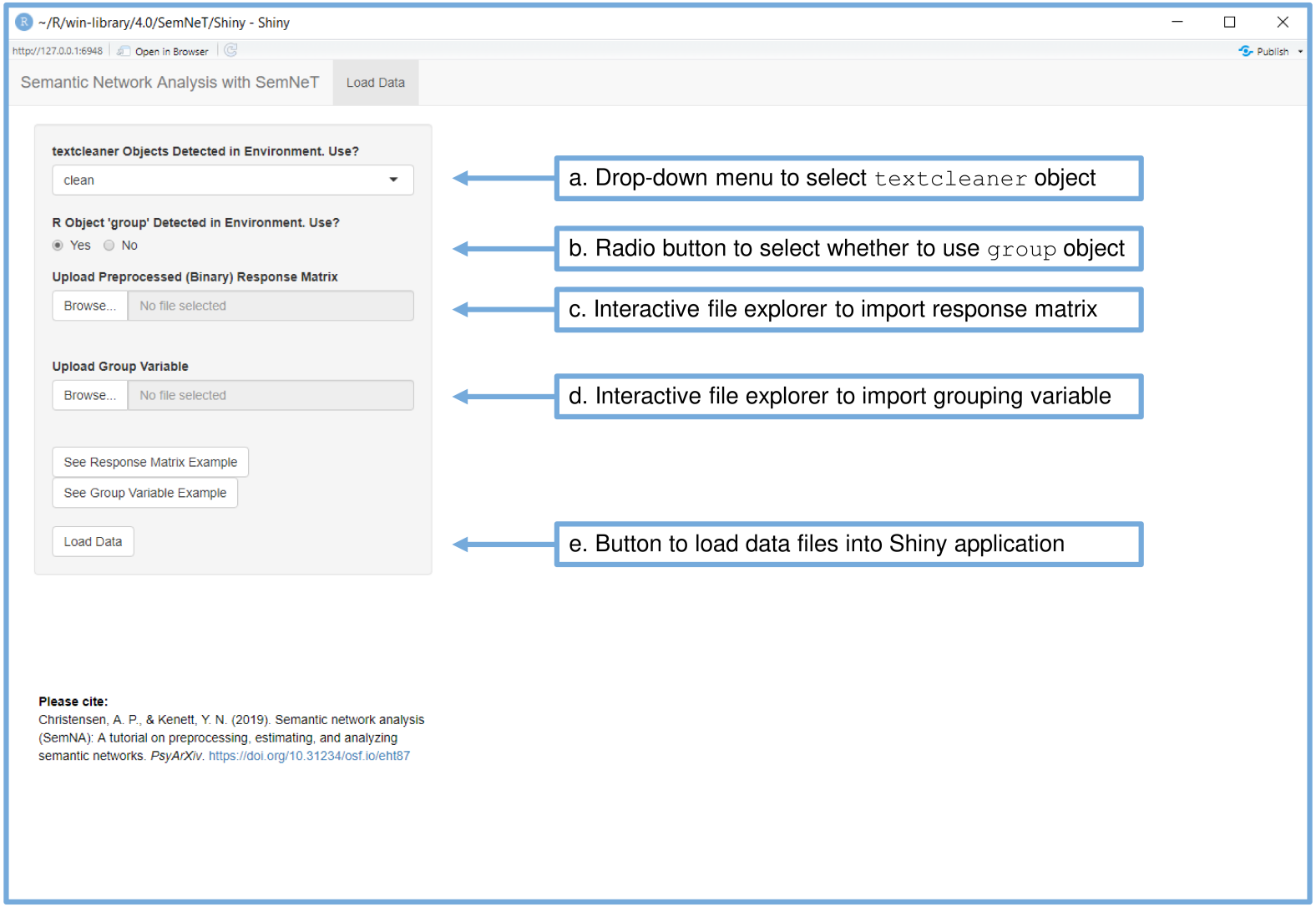

Figure 4: SemNeT Shiny application graphical user interface.

After the data are loaded, a new tab called "Network Estimation" will open (Figure 5), which we turn to next.

2.2. Network Estimation Methods

There are four network estimation methods currently available in *SemNeT*: Community Network (Goñi et al., 2011), Naïve Random Walk (Lerner, Ogrocki, & Thomas, 2009), Pathfinder (Paulsen et al., 1996; Quirin, Cordón, Guerrero-Bote, Vargas-Quesada, & Moya-Anegón, 2008; Schvaneveldt, 1990), and Correlation-based networks (Kenett et al., 2013). We provide brief descriptions of each of these approaches here along with the parameters that can be manipulated in the Shiny application and *SemNeT* package (for more detailed descriptions of these methods, see Zemla & Austerweil, 2018).

2.2.1. Community Network

The Community Network method estimates a semantic network using a co-occurrence matrix where two responses (e.g., *cat* and *dog*) have occurred within some *window* or distance from each word (Goñi et al., 2011). The size of the window is a parameter that can be manipulated and defaults to 2 in *SemNeT*. To provide an example, we consider the response pattern: *cat*, *fish*, *dog*, *aardvark*, *hippopotamus*, and *tardigrade*. The distance between *dog* and *cat* would be two, which would be within the window size of 2 and counted as a co-occurrence. The distance between *dog* and *tardigrade*, however, is three and therefore would not be counted as co-occurring.

The co-occurrence counts are then used to infer whether they are likely to occur by random chance using confidence intervals from a binomial distribution. These confidence intervals are a second parameter that can be adjusted using a significance level, which defaults to .05 (or a 95% confidence interval). Goñi et al. (2011) also implemented a clustering approach to "enrich" the network by connecting all nodes that are in the same cluster. This network enrichment has been implemented in *SemNeT* as a parameter that can be changed via a drop-down menu. The drop-down menu includes FALSE and TRUE for whether the network should be enriched; the default is to not enrich the network (FALSE). The Community Network method has demonstrated validity by finding differences in the semantic structure of multiple sclerosis patients (Abad et al., 2015) as well as the lexicon of younger and older adults (Wulff, Hills, & Mata, 2018). Moreover, Goñi and colleagues (2011) have demonstrated that the community structure (i.e., clusters of animal verbal fluency responses) largely correspond with human rated animal similarity.

2.2.2. Naïve Random Walk

The Naïve Random Walk (Lerner et al., 2009) method estimates a semantic network based on the assumption that the data are generated from an uncensored *random walk* or a stochastic process of taking "steps" from one node in the network to another. This process summarizes to estimating an edge between each adjacent pair of responses in the fluency response matrix. Thus, the Naïve Random Walk method operates with the assumption that adjacent responses are more likely to be related to one another. A common implementation (as it is in *SemNeT*) is to count the co-occurrences of adjacent pairs across the fluency response matrix and then apply a threshold so that only adjacent pairs that co-occur at least as many times as the threshold are estimated to have an edge in the network. This threshold is the only parameter in the *SemNeT* package's implementation of the Naïve Random Walk method and it defaults to 3. The Naïve Random Walk has demonstrated validity by finding exponential growth in the evolution of verbal fluency networks (Shreestha et al., 2015) as well as differences in semantic structure of people with mild cognitive impairments (Lerner et al., 2009).

2.2.3. Pathfinder Network

The Pathfinder Network method, as implemented in *SemNeT* estimates a semantic network using a proximity measure (e.g., Chebyshev distance) between every pair of responses in the binary response matrix (Paulsen et al., 1996). The method starts by computing the proximity matrix and retains only the path with the shortest distance between every pair of nodes. Following Zemla and colleagues' (2018) implementation, we estimate the Pathfinder Network using the union of all *minimum spanning trees* or a set of edges that links all nodes in the network that minimizes the total distance of all edges in the network (Quirin et al., 2008). The Pathfinder Network does not have any parameters to manipulate in *SemNeT*. The Pathfinder Network method has demonstrated validity in a number of studies focused on schizophrenia impairment (Paulsen et al., 1996) as well as Alzheimer's disease (Chan et al., 1995). The Pathfinder Network has also often been used

as a clustering method in non-semantic network studies such as suicide schemas (Pratt, Gooding, Johnson, Taylor, & Tarrier, 2010) and co-authorship in scienctific publications (Pisanski, Pisanski, & Pisanski, 2020).

2.2.4. Correlation-based Networks

Correlation-based Network methods, like the Community Network and Pathfinder Network methods, estimate a semantic network based on co-occurrences of responses in the response matrix. Using the binary response matrix, these methods compute an association measure between every pair of responses, resulting in an association matrix. Most often, these association matrices are estimated using Pearson's correlation (Kenett et al., 2013); however, any association measure could be used. In the original study that this tutorial is based on, for example, we used the cosine similarity to avoid negative assocations between nodes in the network (Christensen et al., 2018b). The association measure is therefore a parameter that can be manipulated with Correlation-based Networks; however, Pearson's correlation and cosine similarity are recommended.

For Correlation-based Networks, a family of network filtering methods known as *Information Filtering Networks* (Barfuss, Massara, Di Matteo, & Aste, 2016) are used to estimate semantic networks (Christensen et al., 2018b; Kenett et al., 2013). This family of filtering methods applies various geometric constraints on the association matrix to identify the most relevant information between nodes while ensuring every node is connected in the network. The two filtering methods that have been most widely applied in the literature are the *Planar Maximally Filtered Graph* (PMFG; Kenett et al., 2013; Tumminello, Aste, Di Matteo, & Mantegna, 2005) and *Triangulated Maximally Filtered Graph* (TMFG; Christensen et al., 2018b; Massara, Di Matteo, & Aste, 2016). We note that only the TMFG is implemented in the *SemNeT* package due to the time-intensiveness of the PMFG in R.[1]

Although the filters slightly differ (see Tumminello et al., 2005 and Massara et al., 2016 for PMFG and TMFG, respectively), their goals are similar: connect nodes in the network by maximizing the strength of their association (e.g., correlation) to other nodes while keeping the network *planar* or so that the network *could* be depicted on a two-dimensional surface without any edges crossing (e.g., Tumminello et al., 2005). This constraint retains $3n - 6$ edges in the network where n is the number of nodes in the network. The results from the PMFG and TMFG generally align with one another.

When estimating group-level networks, a common approach for Correlation-based Networks is to retain responses that at least two participants in each group has provided (e.g., Borodkin, Kenett, Faust, & Mashal, 2016). The purpose of this constraint has been to ensure that Pearson's correlations can be computed for all response pairs and to minimize spurious associations driven by idiosyncrasies in the sample. The minimum number of responses to retain is another parameter that can be adjusted when using the Correlation-based Network method. Finally, responses are "equated" or matched such that each group only retains the responses that are given by all other groups (Kenett et al., 2013). Equating responses ensures that all groups have the same number of nodes and therefore the same number of edges when estimating semantic networks with the Correlation-based Network methods, reducing their confounding effects when making comparisons between networks (Borodkin et al., 2016; van Wijk, Stam, & Daffertshofer, 2010).

Correlation-based networks have demonstrated validity by demonstrating semantic network differences between low and high creative people (Kenett, Anaki, & Faust, 2014; Kenett & Faust, 2019; but see Lange, Hopman, Zemla, & Austerweil, 2020) as well as people with high functioning autism (Asperger Syndrome) compared to matched controls (Kenett et al., 2016b). These networks have also shown validity for identifying key components of psychosis proneness in self-report psychopathological data (Christensen et al., 2018a), estimating factors in psychometric networks (Golino, Shi, et al., 2020), and identifying ecologically valid topics in Twitter data (Golino et al., 2020).

2.3. Comparison of Network Estimation Methods

To our knowledge, there is only one study to systematically compare these and other network estimation methods for group-level semantic networks estimated from verbal fluency data (Zemla & Austerweil, 2018). In their study, they simulated animals verbal fluency data from the University of South Florida free association norms and generated networks using censored random walks (Nelson et al., 2004). In addition, they obtained human-rated semantic similarity for animals pairs and evaluated the average similarity of these ratings for edges that were present and absent in the networks estimated with these methods.

[1] The PMFG method is available in Python (SNAFU) as well as in R (`https://github.com/AlexChristensen/PMFG`) and Matlab (`https://www.mathworks.com/matlabcentral/fileexchange/38689-pmfg?s_tid=prof_contriblnk`). The R implementation is substantially slower than the Python and Matlab implementations.

Based on their results, they recommended the Community Network method for groups where the network may not be fully connected and the goal is to minimize non-edge (absent edge) similarities; the Pathfinder Network method was recommended for groups where the network is fully connected and the goal is to maximize edge similarities whereas the Naïve Random Walk method was recommended when the goal is to minimize non-edge similarities. Finally, two methods that are not available in *SemNeT* but are available in SNAFU (Zemla et al., 2020), U-INVITE and Hierarchical U-INVITE, were recommended for groups when the network is fully connected and the goal is to minimize non-edge similarities and when the network may not be fully connecting and the goal is to maximize edge similarity, respectively. Correlation-based methods were not recommended for any of these conditions.

This systematic comparison is an important first step for evaluating these methods but more studies are needed. Comparisons of network estimation methods are beyond the scope of this paper, so we provide users flexibility in the estimation method they prefer. However, we provide three recommendations for future comparisons. First, verbal fluency data should be generated based on verbal fluency data. Although the USF network is a well-validated network that "accurately reflects mental associations between items in the network" (Zemla & Austerweil, 2018, p. 47), it would be surprising if the cognitive processes (e.g., strategic search, spreading activation) of free association norms were identical to the processes of verbal fluency. Some similar processes are likely to be present, but there are different task constraints and people are likely to use different strategies in their search processes (e.g., Unsworth, Spillers, & Brewer, 2011). Moreover, task features are likely to promote different semantic structures such as higher clustering and shorter paths lengths in verbal fluency networks.

Second, the verbal fluency data were generated using a censored random walk, which may be a reasonable data generating mechanism, but it matches the way in which the U-INVITE method estimates networks. This potentially confounds the results to the extent in which the true data generating process matches the way the data is generated. Therefore, other data generating mechanisms should be used to test the generality of the approaches and how approaches are affected should the true data generating model not correspond to a censored random walk.

Finally, the parameters of the network estimation methods should be varied to determine what parameters are most optimal for different conditions. It's unclear, for example, if a window size of two for the Community Network method is optimal or whether thresholding co-occurrences of three in the Naive Random Walk method or two in the Correlation-based Network methods produce consistent results. Moreover, for Correlation-based Networks, what similarity measure works best and under what conditions? Evaluating the effects of these parameters can help researchers make more informed decisions about what method and parameters might work best with their data.

2.4. Estimating Networks in the *SemNeT* Shiny Application

In continuing with our tutorial, we implemented the TMFG method, which we implemented in our original published paper (Christensen et al., 2018b). For completeness, we estimated semantic networks from this data using the other estimation methods and achieved similar results (see SI 7 and 8). In Figure 5, we demonstrate how to implement the TMFG method in the Shiny application.

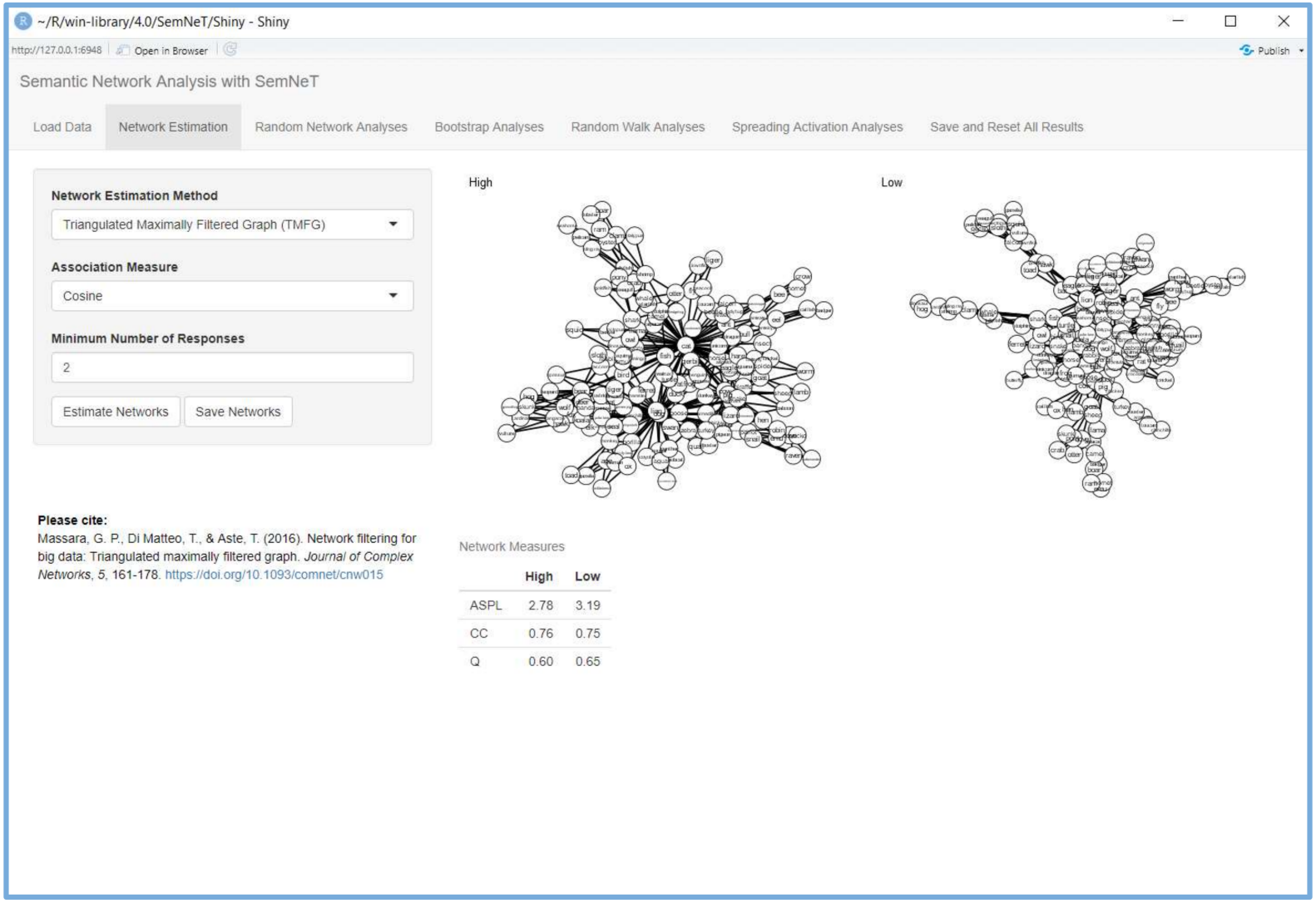

Figure 5: TMFG estimation method applied in Shiny application.

When selecting a network estimation method from the drop-down menu, the parameter options will change. In Figure 5, there are two parameters: Association Measure and Minimum Number of Responses. We chose the defaults of cosine and two, respectively, which correspond with how we estimated the networks in the original paper. After setting the parameters, the "Estimate Networks" button can be pressed.

Once the networks are estimated, a figure will appear with the networks side-by-side along with some global network measures (which we will discuss in the next section). Of note, on the left-hand side of the screen below the parameters, citations for the methods will appear. Citations will appear with each type of analysis performed in the *SemNeT* Shiny application to provide researchers with greater detail about the methods, but also to provide quick APA-style references for writing the Methods section of a manuscript. Hyperlinked DOIs are also provided where available.

2.5. Exporting Networks

These networks can be extracted and used in other software by pressing the "Save Networks" button, which will produce a message that allows the user to input a name for the object to be saved (e.g., "my_networks"). Upon closing out of the Shiny application, an object in R's environment called `my_networks` will appear. These networks will be labeled with their group name. In the example code below, we show how to extract these networks and demonstrate how to convert them to the *igraph* package's (Csardi & Nepusz, 2006) format:

```
# Extract group network results
# from Network Estimation section
low <- my_networks$network$Low
high <- my_networks$network$High


# Convert network's to igraph's format
low.igraph <- convert2igraph(low)
high.igraph <- convert2igraph(high)

# Save networks for other software
write.csv(low, file = "low_network.csv", row.names = FALSE)
write.csv(high, file = "high_network.csv", row.names = FALSE)
```

The conversion of the networks to the *igraph* package's format is another demonstration of how this pipeline can be modular and facilitate cross-software compatibility. Once converted, these networks can be used with the popular *igraph* package in R. Moreover, similar to the preprocessed data export, the network objects `low` and `high` can also be saved as .csv files and exported to other software.

2.6. Summary

In this section, we briefly reviewed the network estimation methods available in the *SemNeT* package and applied one approach for estimating group-based semantic networks. For networks estimated in the *SemNeT* Shiny application, we demostrated how they can be converted to use with the popular *igraph* package in R as well as exported for other software.

We will continue the tutorial by applying statistical analyses to the TMFG networks we estimated in this section. We note, however, that these same analyses can be found for each network estimation method in the Supplementary Materials (SI 7 and 8). All statistical analyses can be completed within the *SemNeT* Shiny application and are seamlessly connected to the networks estimated in this section—that is, networks for all subsequent analyses will be estimated with the same parameters used to estimate the group networks.

3. Analyzing Semantic Networks

To continue with our SemNA pipeline, the user does not need to exit the Shiny application. Instead, there are several statistical analyses available in the *SemNeT* package to apply to the networks, which appear as tabs in the Shiny application (see Figure 5). These analyses include comparisons against random networks (Random Network Analyses), bootstrapping the networks with group comparison (Bootstrap Analyses), random walks on the networks with group comparison (Random Walk Analyses), and spreading activation on individual

networks provided by the *spreadr* package (Spreading Activation Analyses; Siew, 2019).[2] For brevity, we discuss and perform a comparison against random networks and bootstrap network group comparisons (see Kenett & Austerweil, 2016 and Siew, 2019, for explanations on the Random Walk and Spreading Activation Analyses).

3.1. Global Network Measures

Returning to the global network measures computed in Figure 5, we begin with an explanation of these measures. Common SemNA metrics are global (or macroscopic) network measures. These measures focus on the structure of the entire network, emphasizing how the nodes are connected as a cohesive whole (Siew, 2019). In contrast, local (or microscopic) network measures focus on the role of individual nodes in the network (Bringmann et al., 2019). Below, we briefly define and describe a few global network measures—average shortest path length, clustering coefficient, and modularity—that are commonly used in SemNA. A more extensive review of these and other measures can be found in Siew (2019).

A common network structure that tends to emerge across many different systems (including semantic memory) is a small-world structure (Watts & Strogatz, 1998). Small-world networks are characterized by having a moderate average shortest path length (ASPL) and large clustering coefficient (CC). The ASPL refers to the average shortest number of steps (i.e., edges) that is needed to get between any pair of nodes in the network. That is, it's the average of the minimum number of steps from one node to all other nodes. In cognitive models, ASPL may affect the activation of associations between concepts (known as *spreading activation*; Anderson, 1983; Siew, 2019) such that a lower ASPL would increase the likelihood of reaching a greater number associations. Several studies of creative ability and semantic networks have linked lower ASPL to greater creative ability (Benedek et al., 2017; Kenett et al., 2014; Kenett & Faust, 2019; but see, Lange et al., 2020). These studies have argued that the lower ASPL in the semantic network of people with higher creative ability (compared to less creative people) may have allowed them to reach more remote associations, which in turn could be combined into novel and useful associations (Kenett & Faust, 2019).

The CC refers to the extent that two neighbors of a node will be neighbors themselves, on average, in the network. A network with a higher CC, for example, suggests that nodes that are near-neighbors to each other tend to co-occur and be connected. Wulff et al. (2018) examined younger and older adults semantic networks that were estimated using verbal fluency data and found that the older adults had a smaller CC compared to the younger adults. This structure was interpreted as having a role in the cognitive slowing observed in older adults.

Finally, modularity measures how well a network compartmentalizes (or partitions) into sub-networks (i.e., smaller networks within the overall network; Fortunato, 2010; Newman, 2006). The maximum modularity coefficient (Q) estimates the extent to which the network has dense connections between nodes within a sub-network and sparse (or few) connections between nodes in different sub-networks. In this implementation, Q refers to the maximum modularity possible given all possible partition organizations. These partitions are estimated using the Louvain algorithm (Blondel, Guillaume, Lambiotte, & Lefebvre, 2008) in the *igraph* package. Larger Q values suggest that these sub-networks are more well-defined than lower Q values, suggesting that the network may be more readily segmented into different parts. A few studies have demonstrated the significance of modularity in cognitive networks (Kenett et al., 2016b; Siew, 2013). For example, Kenett et al. (2016b) found that people with high functioning autism (Asperger Syndrome) had a more modular semantic network relative to matched controls. They suggest that this "hyper" modularity might be related to rigidity of thought that often characterizes people with Asperger Syndrome.

3.2. Statistical Tests

In order to statistically test for differences in these measures across networks, other approaches must be used. Here, we present two procedures that are available in the *SemNeT* Shiny application that statistically test for differences between networks: comparisons against random networks and bootstrap network group comparisons (e.g., Kenett et al., 2013, 2016a).

[2] We are grateful to Cynthia Siew for her approval to include the *spreadr* package in the Shiny application and we thank her for the helpful feedback on the implementation.

3.2.1. Random Network Analyses

Comparisons against random networks can be performed to determine whether the network measures observed in the groups are different from what would be expected from a random network with the same number of nodes and edges (Beckage, Smith, & Hills, 2011; Steyvers & Tenenbaum, 2005). Further, these random networks are generated such that their *degree sequence* or the number of connections to each node is preserved, maintaining the general structure of the network.

These random networks are generated using Viger and Latapy's (2016) Markov Chain Monte Carlo (MCMC) algorithm on a random network with a specified degree sequence. Specifically, the approach starts with a random network with the same number of nodes and edges as the original network. The edges are than randomized using the MCMC algorithm to ensure that each node in the network preserves the number of connections it had in the original network. This approach is implemented using the *igraph* package's `sample_degseq` function.

The test against random networks approach works as follows: (1) for each group's network, generate X number of random networks (e.g., 1000), (2) then compute global network measures (i.e., ASPL, CC, and Q) for each group's random networks, resulting in a sampling distribution of these measures, (3) finally, compute a p-value for the original group's network measures based on the sampling distribution (e.g., normal) of the random network's network measures (Kenett et al., 2013). To implement this procedure, the user should click on the "Random Analyses" tab (Figure 6).

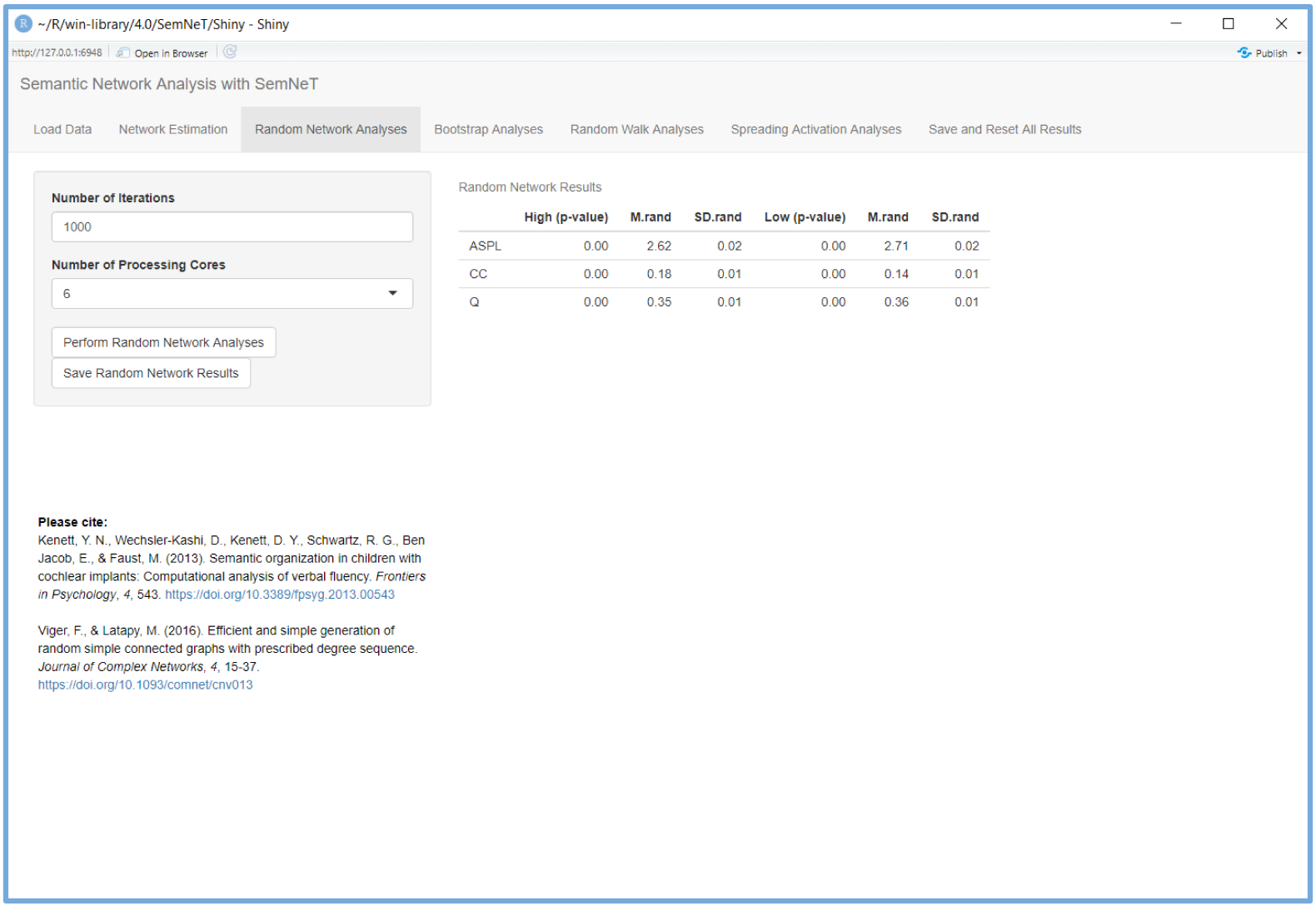

Figure 6: Random Network Analyses in Shiny application.

There is one analysis and one computation option that can be changed. For the analysis option, the number of iterations or random networks generated can be adjusted. The default and standard in the literature has been to generate 1,000 random networks per group network (e.g., Kenett et al., 2013). For the computation option, the number of processors for parallel computing can be selected. The default is for half the number of processors on the user's machine. The more processors, the faster the computation time. On the computer used to compute these analyses—Windows 10, Intel Core i7-9750H 2.60GHz processor, and 32GB of RAM using 6 processing cores—the timing is around fifteen minutes. After clicking the "Perform Random Analyses" button, a message will appear letting the user know the analysis has started and that progress can be tracked in R's console.

The output from this function is a table that reports the p-values for each group's network compared to the random network's network measure values (e.g., High (p-value) and Low (p-value)) as well as the means (e.g, M.rand) and standard deviations (e.g., SD.rand) for the random network's distribution of global network measures (Figure 6).

In our example, there were two groups, so the result was based on random networks computed for each group's network structure. As shown in Figure 6, all global network measures were significantly different from random for both openness to experience groups. This suggests that both networks have significantly different structures than a random network with the same number of nodes, edges, and degree sequence.[3]

3.2.2. Bootstrap Analyses

The second analysis to statistically compare semantic networks applies a bootstrap method (Efron, 1979). There are two bootstrap approaches that can be applied in the SemNA pipeline: case-wise and node-wise bootstrap. The case-wise approach samples N participants with replacement from the respective group, meaning that some participants may be included more than once while others may not be included at all. For each replicate sample, a network estimation method is applied (e.g., TMFG) and then the global network measures—ASPL, CC, and Q—are computed. This process repeats iteratively (usually 1,000 times).

The node-wise approach is only available for the TMFG method because the common approach is to equate the responses between groups. The node-wise approach starts by selecting a subset of nodes in the network (e.g., 50%), then estimating the networks for each group using this subset of nodes, and finally, computing the network measures for each network (Kenett et al., 2014). This method is known as *without replacement* because each node can only be selected once (Bertail, 1997; Politis & Romano, 1994; Shao, 2003). With these nodes removed from the data, partial networks are estimated from each group's data. The network measures are then computed for these partial networks. This process repeats iteratively (usually 1,000 times). The rationale for the node-wise approach is twofold: (1) if the full networks differ from each other, then any partial network consisting of the same nodes should also be different, and thus (2) the generation of many partial networks allows for a direct statistical comparison between the full networks (Kenett et al., 2016b).

For both bootstrapping approaches, these replicate networks form sampling distributions of the global network measures, but solely based on the empirical data. These sampling distributions are then statistically compared using an analysis of covariance (ANCOVA) with the number of nodes and edges used as covariates for estimating whether the global network measures are different between each group's networks. These covariates statistically control for confounds that affect comparing network measures between groups: ASPL, for example, will often be smaller for networks with a greater ratio of edges to nodes (van Wijk et al., 2010). Adjusted means and effect sizes that account for these confounds are then estimated.

Sticking with the analyses of the original paper that is our example (Christensen et al., 2018b), we implemented the node-wise approach, retaining a gradation of the proportion of nodes (.50, .60, .70, .80, and .90; Christensen et al., 2018b). As a comparison, we applied the case-wise approach but only report the results alongside the node-wise approach to demonstrate that they reach similar conclusions. To continue with the tutotial, the user can click on the "Bootstrap Analyses" tab and select the "Node" option for the "Type of Bootstrap" parameter (Figure 7).

[3] All network estimation methods found that both group networks were significantly different from random networks (SI 7).

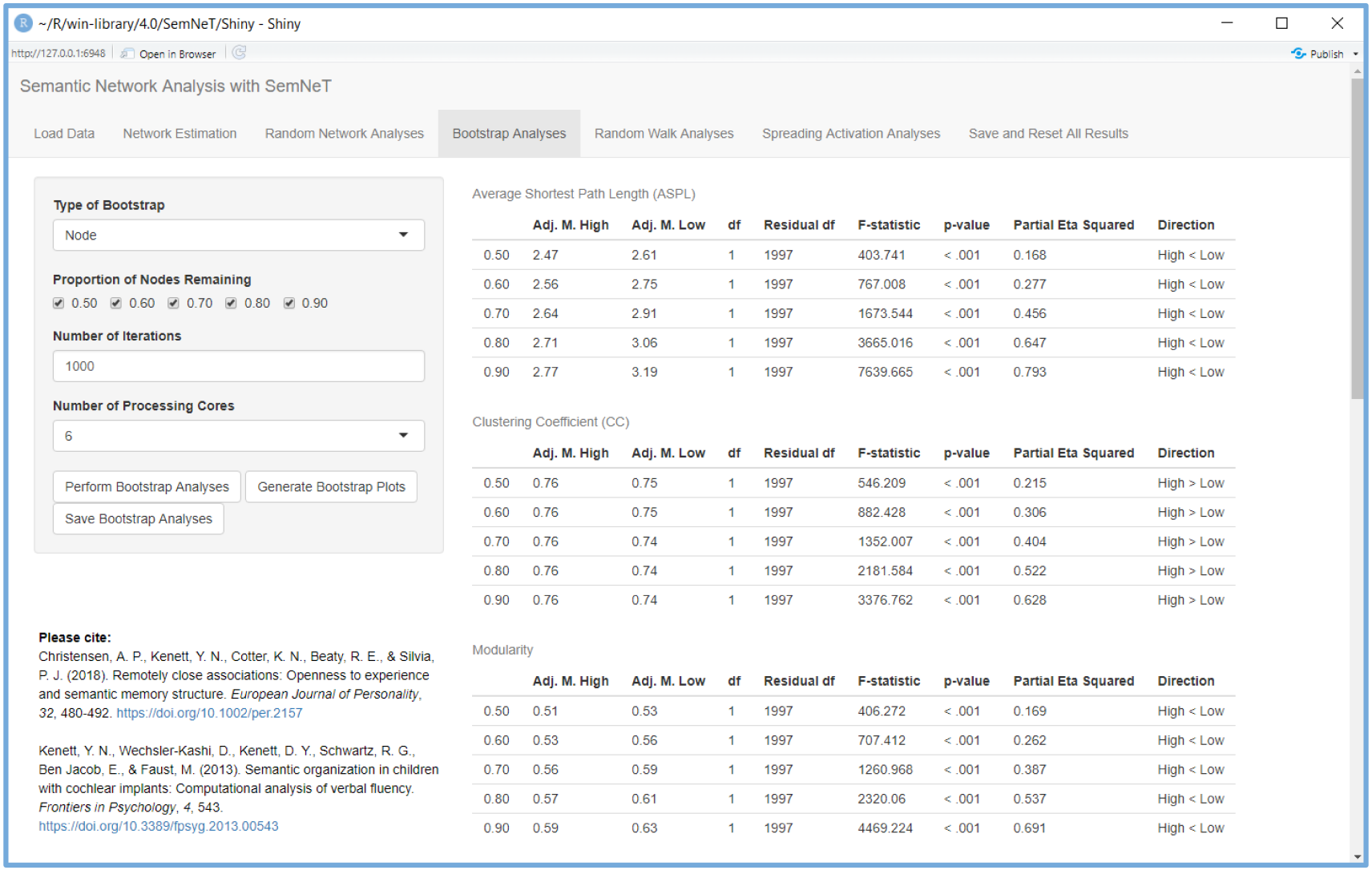

Figure 7: Bootstrap Analyses in Shiny application.

The default for the node-wise approach is to compute all proportions of nodes remaining (i.e., .50, .60, .70, .80, and .90). To perform the analyses with only a certain set of proportions, the boxes can be unchecked (Figure 7). After the analyses are complete (around 15 minutes on the abovementioned computer), a table with the results will appear. In Figure 7, we show that across the proportions of nodes retained, the results are consistent: The high openness to experience network had significantly lower ASPL and Q, and higher CC.[4]

A button labeled "Generate Plots" will appear after the analyses is finished. Pressing this button will generate plots for the distribution of the ASPL, CC, and Q for each group. The Shiny application will offer a preview of these plots but for publication purposes larger or individual plots are often necessary. To demonstrate how to generate and access individual plots, we'll turn to the output of the Shiny application.

3.3. Shiny Results Output

Output from the Shiny application will automatically be generated upon closing the application (e.g., clicking on the "X" in the top right corner of the window). This output will be stored in a list object called `resultShiny` in R's environment. Below is a table describing the possible output:

Table 4: SemNeT Shiny Application Output Objects

Object	Description
`data`	Data imported into the Shiny application
`group`	Grouping variable imported into the Shiny application
`network`	The last networks generated during the session. These networks will be labeled using the grouping variable
`measures`	Network measures ASPL, CC, and Q for the each group's networks
`comparePlot`	Visualization of each group's networks
`randomTest`	Statistical results from the Random Network Analyses tab
`bootstrap`	The bootstrapped samples and measures from the Bootstrap Analyses tab
`bootstrapTest`	Statistical results table from the Bootstrap Analyses tab
`bootstrapPlot`	Plots of the statistical results from the Bootstrap Analyses tab
`randomWalk`	Results from the Random Walk Analyses tab
`spreadingActivation`	Results from the Spreading Activation Analyses tab
`spreadingActivationPlot`	Plots of the analyses in the Spreading Activation Analyses tab

Objects in the respective analysis section can be saved individually by using the "Save..." buttons within each analysis tab of the Shiny application (e.g., see Figures 5, 6, and 7). Moreover, there is a tab called "Save and Reset All Results," which allows the user to do just as it says: save all results or reset the results to start over. These buttons are useful for saving different results (e.g., different network estimation method and analyses) across the Shiny session and then resetting the results to perform analyses using a different network estimation method. These results will be available in R's environment upon exiting the Shiny application. It's important to note that the results reported in the `resultShiny` object upon closing the application will only save the last analyses performed in the application. Therefore, it's optimal to save results either individually or using the "Save All Results" button to ensure all desired results are saved.

One object worth pointing out is the bootstrap plots output, which appears in the Shiny application when clicking the "Generate Plots" button in the Bootstrap Analyses tab. The plots in the application are useful

[4]These results were replicated by the case-wise bootstrap approach and largely reproduced by the other network estimation methods with the Pathfinder Network method finding the same pattern of significant differences, Community Network method finding the same pattern for ASPL and Q but the opposite direction for CC, and Naïve Random Walk method finding the opposite pattern of the TMFG and Pathfinder methods (SI 8).

for viewing; however, they are not optimal for publication purposes. Instead, these plots can be regenerated from R's console by using the following code:

```
# Plot bootstrap results
## ASPL
plot(resultShiny$bootstrapPlot$aspl)
## CC
plot(resultShiny$bootstrapPlot$cc)
## Q
plot(resultShiny$bootstrapPlot$q)
```

Each network measure is output as an individual plot, which allows the user to manipulate each plot into the desired size for publication (e.g., making sure plot features are large enough to be easily discernable). Moreover, each plot is generated using the *ggplot2* package (Wickham, 2016), which means they can be manipulated to use different colors (e.g., distributions, boxplots, dots) or adjust titles, axis labels, and legend keys (see Wickham, 2016 for getting started).

3.4. Summary

In this section, we briefly reviewed two statistical analyses techniques, comparisons against random networks and bootstrap network group comparisons, available in the *SemNeT* package. We then demonstrated how to perform these analyses using the Shiny application and provided an overview of the output that can be saved. From this output, we demonstrated how to obtain individual plots from the bootstrap results that enables the user to generate publication ready figures.

General Discussion

In this paper, we put forward a SemNA pipeline for semantic networks based on verbal fluency data. This pipeline was accompanied by three R packages—*SemNetDictionaries*, *SemNetCleaner*, and *SemNeT*—that are designed together to facilitate the application of semantic network analysis. With these packages, we demonstrated how this pipeline could be applied to real-world data by re-analyzing verbal fluency data previously collected by Christensen et al. (2018b). To get to these results, this tutorial went step-by-step through preprocessing the verbal fluency data by checking for spelling errors and inappropriate and duplicate responses, and then loading the data into the SemNeT Shiny application to estimate and analyze semantic networks. This tutorial serves two roles not yet filled in the literature: first, for novice SemNA researchers, we've provided a step-by-step resource for how to execute SemNA with verbal fluency data; second, for experienced SemNA researchers, we've provided a standardized approach to efficiently preprocess verbal fluency data, while also increasing the transparency of this process. Because the pipeline is modular, novice and experienced SemNA researchers alike are encouraged to interchange parts of the pipeline with other software, maximizing the potential analyses available for their semantic data.

The standardization of the preprocessing stage offers a few advances for semantic network analysis and beyond. First, the preprocessing of semantic data becomes more consistent and transparent across researchers, encouraging open science practices (e.g., output can be shared in the peer-review process). Such an approach enables common practices across labs and fields to emerge. Moreover, with the rise of big data and crowd-sourcing (Mandera, Keuleers, & Brysbaert, in press), norms can be developed for each verbal fluency category (e.g., De Deyne, Navarro, Perfors, Brysbaert, & Storms, 2019), which would further facilitate the automation and accuracy of preprocessing verbal fluency data. The *SemNetDictionaries* package, for example, provides researchers with the capability to create their own dictionaries, which can be developed from these norms. Broadly, researchers are encouraged to share their dictionaries on the first author's GitHub page,[5] so that future versions can include them as pre-defined dictionaries in the package.

Second, the *SemNetCleaner* package enables researchers to efficiently preprocess their verbal fluency data into a format that can be used with any semantic network estimation method available in *SemNeT* and SNAFU (e.g., Zemla & Austerweil, 2018). The speed with which *SemNetCleaner* preprocesses verbal fluency data is magnitudes times faster than any manual method. Take, for instance, our example data: there were over 9,000 responses and over 800 unique responses. Although many of these responses were spelled correctly,

[5] https://github.com/AlexChristensen/SemNetDictionaries/issues/new/choose

the user was required to make fewer than 50 decisions. With *SemNetCleaner*, the entire preprocessing stage can be managed in under 20 minutes.[6]

Beyond SemNA, the automated preprocessing of verbal fluency data should achieve more reliable verbal fluency results because human error is largely removed from this process. This means that researchers who are not necessarily interested in performing SemNA can still use *SemNetCleaner* to ensure the reliability and validity of their verbal fluency results (e.g., total number of appropriate responses). This becomes especially important when using automated methods of scoring (e.g., clustering and switching, semantic and lexical diversity; Kim, Kim, Wolters, MacPherson, & Park, 2019; Pakhomov, Eberly, & Knopman, 2016; Zemla et al., 2020).

With the networks outputted from the network estimation stage, researchers can feasibly implement other network analysis packages in R (e.g., *igraph*) to compute other network measures not covered in this tutorial (e.g., centrality measures). One notable package is *spreadr* (Siew, 2019), which simulates spreading activation across a network. Spreading activation is an important concept in the semantic memory literature, which may provide insights into how information is retrieved from semantic memory. This pipeline can seamlessly be integrated with *spreadr* by using the *SemNeT* Shiny application for network estimation and statistical analyses (i.e., Spreading Activation Analyses tab).

Finally, the *SemNeT* package offers researchers a couple methods for the analysis of semantic network data—comparisons against random networks and bootstrap network group comparisons. *SemNeT* also includes some methods not covered in this article such as random walk analysis (Kenett & Austerweil, 2016). In sum, this suite of open-source R packages provides researchers with an efficient and feasible approach to produce reliable and reproducible SemNA results.

Importantly, these packages minimize the barriers for researchers to apply SemNA to their own data, opening doors to applications of SemNA in new domains. One promising avenue for future research is to understand how the structure and processes of semantic memory are associated with psychological disorders (Elvevåg et al., 2017; Holmlund, Cheng, Foltz, Cohen, & Elvevåg, 2019; Kenett & Faust, 2020; Kenett et al., 2016b) and personality traits (Christensen et al., 2018b). To date, there have been only a handful of studies that have quantitatively examined the cognitive processes that underlie these phenomena in psychology.

With this pipeline in hand, researchers can readily apply SemNA to their own data. The strength of analyzing verbal fluency data is that it is quick and easy to administer within any experimental paradigm. For applied researchers, this makes verbal fluency data an attractive option for studying cognitive processes associated with behavioral and psychopathological phenomena. By quantitatively analyzing semantic networks, cognitive theory can be extended and integrated into psychological theories, bridging these sometimes disparate yet parallel lines of research.

[6] A. P. Christensen initially cleaned this data by hand for his Master's thesis. It took nearly 8 hours to manually input the 1's and 0's in the binary response matrix alone—the timing of which did not include any spell-checking or verification of category exemplars.

References

Abad, E., Sepulcre, J., Martinez-Lapiscina, E. H., Zubizarreta, I., Garcia-Ojalvo, J., & Villoslada, P. (2015). The analysis of semantic networks in multiple sclerosis identifies preferential damage of long-range connectivity. *Multiple Sclerosis and Related Disorders*, *4*, 387–394. https://doi.org/10.1016/j.msard.2015.07.002

Anderson, J. R. (1983). A spreading activation theory of memory. *Journal of Verbal Learning and Verbal Behavior*, *22*, 261–295. https://doi.org/10.1016/S0022-5371(83)90201-3

Ardila, A., Ostrosky-Solís, F., & Bernal, B. (2006). Cognitive testing toward the future: The example of semantic verbal fluency (ANIMALS). *International Journal of Psychology*, *41*, 324–332. https://doi.org/10.1080/00207590500345542

Barfuss, W., Massara, G. P., Di Matteo, T., & Aste, T. (2016). Parsimonious modeling with information filtering networks. *Physical Review E*, *94*, 062306. https://doi.org/10.1103/PhysRevE.94.062306

Baronchelli, A., Ferrer-i-Cancho, R., Pastor-Satorras, R., Chater, N., & Christiansen, M. H. (2013). Networks in cognitive science. *Trends in Cognitive Sciences*, *17*, 348–360. https://doi.org/10.1016/j.tics.2013.04.010

Beckage, N., Smith, L., & Hills, T. (2011). Small worlds and semantic network growth in typical and late talkers. *PloS ONE*, *6*, e19348. https://doi.org/10.1371/journal.pone.0019348

Benedek, M., Kenett, Y. N., Umdasch, K., Anaki, D., Faust, M., & Neubauer, A. C. (2017). How semantic memory structure and intelligence contribute to creative thought: A network science approach. *Thinking & Reasoning*, *23*, 158–183. https://doi.org/10.1080/13546783.2016.1278034

Bertail, P. (1997). Second-order properties of an extrapolated bootstrap without replacement under weak assumptions. *Bernoulli*, *3*, 149–179. https://doi.org/10.2307/3318585

Blondel, V. D., Guillaume, J.-L., Lambiotte, R., & Lefebvre, E. (2008). Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, *2008*, P10008. https://doi.org/10.1088/1742-5468/2008/10/P10008

Borge-Holthoefer, J., & Arenas, A. (2010). Semantic networks: Structure and dynamics. *Entropy*, *12*, 1264–1302. https://doi.org/10.3390/e12051264

Borodkin, K., Kenett, Y. N., Faust, M., & Mashal, N. (2016). When pumpkin is closer to onion than to squash: The structure of the second language lexicon. *Cognition*, *156*, 60–70. https://doi.org/10.1016/j.cognition.2016.07.014

Borsboom, D., & Cramer, A. O. J. (2013). Network analysis: An integrative approach to the structure of psychopathology. *Annual Review of Clinical Psychology*, *9*, 91–121. https://doi.org/10.1146/annurev-clinpsy-050212-185608

Bousfield, W. A., & Sedgewick, C. H. W. (1944). An analysis of sequences of restricted associative responses. *The Journal of General Psychology*, *30*, 149–165. https://doi.org/10.1080/00221309.1944.10544467

Bringmann, L. F., Elmer, T., Epskamp, S., Krause, R. W., Schoch, D., Wichers, M., ... Snippe, E. (2019). What do centrality measures measure in psychology networks? *Journal of Abnormal Psychology*, *128*, 892–903. https://doi.org/10.1037/abn0000446

Chan, A. S., Butters, N., Salmon, D. P., Johnson, S. A., Paulsen, J. S., & Swenson, M. R. (1995). Comparison of the semantic networks in patients with dementia and amnesia. *Neuropsychology*, *9*, 177–186. https://doi.org/10.1037/0894-4105.9.2.177

Christensen, A. P., Kenett, Y. N., Aste, T., Silvia, P. J., & Kwapil, T. R. (2018a). Network structure of the Wisconsin Schizotypy Scales–Short Forms: Examining psychometric network filtering approaches. *Behavior Research Methods*, *50*, 2531–2550. https://doi.org/10.3758/s13428-018-1032-9

Christensen, A. P., Kenett, Y. N., Cotter, K. N., Beaty, R. E., & Silvia, P. J. (2018b). Remotely close associations: Openness to experience and semantic memory structure. *European Journal of Personality*, *32*, 480–492. https://doi.org/10.1002/per.2157

Collins, A. M., & Loftus, E. F. (1975). A spreading-activation theory of semantic processing. *Psychological Review*, *82*, 407–428. https://doi.org/10.1037/0033-295X.82.6.407

Cramer, A. O. J., van der Sluis, S., Noordhof, A., Wichers, M., Geschwind, N., Aggen, S. H., ... Borsboom, D. (2012). Dimensions of normal personality as networks in search of equilibrium: You can't like parties if you don't like people. *European Journal of Personality*, *26*, 414–431. https://doi.org/10.1002/per.1866

Csardi, G., & Nepusz, T. (2006). The igraph software package for complex network research. *InterJournal, Complex Systems*, *1695*, 1–9. Retrieved from https://pdfs.semanticscholar.org/1d27/44b83519657f5f2610698a8ddd177ced4f5c.pdf

Damerau, F. J. (1964). A technique for computer detection and correction of spelling errors. *Communications of the ACM*, *7*, 171–176. https://doi.org/10.1145/363958.363994

De Deyne, S., Kenett, Y. N., Anaki, D., Faust, M., & Navarro, D. J. (2016). Large-scale network representations of semantics in the mental lexicon. In M. N. Jones (Ed.), *Big data in cognitive science: From methods to insights* (pp. 174–202). New York, NY: Taylor & Francis. https://doi.org/10.4324/9781315413570

De Deyne, S., Navarro, D. J., Perfors, A., Brysbaert, M., & Storms, G. (2019). The "Small World of Words" English word association norms for over 12,000 cue words. *Behavior Research Methods*, *51*, 987–1006. https://doi.org/10.3758/s13428-018-1115-7

DeYoung, C. G., Quilty, L. C., & Peterson, J. B. (2007). Between facets and domains: 10 aspects of the Big Five. *Journal of Personality and Social Psychology*, *93*, 880–896. https://doi.org/10.1037/0022-3514.93.5.880

Efron, B. (1979). Bootstrap methods: Another look at the jackknife. *The Annals of Statistics*, 1–26. Retrieved from https://www.jstor.org/stable/2958830

Elvevåg, B., Foltz, P. W., Rosenstein, M., Ferrer-i-Cancho, R., De Deyne, S., Mizraji, E., & Cohen, A. (2017). Thoughts about disordered thinking: Measuring and quantifying the laws of order and disorder. *Schizophrenia Bulletin*, *43*, 509–513. https://doi.org/10.1093/schbul/sbx040

Feinerer, I., Hornik, K., & Meyer, D. (2008). Text mining infrastructure in R. *Journal of Statistical Software*, *25*, 1–54. Retrieved from http://www.jstatsoft.org/v25/i05/

Fortunato, S. (2010). Community detection in graphs. *Physics Reports*, *486*, 75–174. https://doi.org/10.1016/j.physrep.2009.11.002

Fried, E. I., & Cramer, A. O. J. (2017). Moving forward: Challenges and directions for psychopathological network theory and methodology. *Perspectives on Psychological Science*, *12*, 999–1020. https://doi.org/10.1177/1745691617705892

Golino, H., Christensen, A. P., Moulder, R., Kim, S., & Boker, S. M. (2020). Modeling latent topics in social media using Dynamic Exploratory Graph Analysis: The case of the right-wing and left-wing trolls in the 2016 US elections. *PsyArXiv*. https://doi.org/10.31234/osf.io/tfs7c

Golino, H., Shi, D., Christensen, A. P., Garrido, L. E., Nieto, M. D., Sadana, R., ... Martinez-Molina, A. (2020). Investigating the performance of Exploratory Graph Analysis and traditional techniques to identify the number of latent factors: A simulation and tutorial. *Psychological Methods*, *25*, 292–320. https://doi.org/10.1037/met0000255

Goñi, J., Arrondo, G., Sepulcre, J., Martincorena, I., de Mendizábal, N. V., Corominas-Murtra, B., ... Villoslada, P. (2011). The semantic organization of the animal category: Evidence from semantic verbal fluency and network theory. *Cognitive Processing*, *12*, 183–196. https://doi.org/10.1007/s10339-010-0372-x

Holmlund, T. B., Cheng, J., Foltz, P. W., Cohen, A. S., & Elvevåg, B. (2019). Updating verbal fluency analysis for the 21st century: Applications for psychiatry. *Psychiatry Research*, *273*, 767–769. https://doi.org/10.1016/j.psychres.2019.02.014

Jones, M. N., Willits, J., Dennis, S., & Jones, M. (2015). Models of semantic memory. In J. R. Busemeyer, Z. Wang, J. T. Townsend, & A. Eidels (Eds.), *The Oxford handbook of mathematical and computational psychology* (pp. 232–254). New York, NY: Oxford University Press. https://doi.org/10.1093/oxfordhb/9780199957996.013.11

Karuza, E. A., Thompson-Schill, S. L., & Bassett, D. S. (2016). Local patterns to global architectures: Influences of network topology on human learning. *Trends in Cognitive Sciences*, *20*, 629–640. https://doi.org/10.1016/j.tics.2016.06.003

Kenett, Y. N., Anaki, D., & Faust, M. (2014). Investigating the structure of semantic networks in low and high creative persons. *Frontiers in Human Neuroscience*, *8*, 407. https://doi.org/10.3389/fnhum.2014.00407

Kenett, Y. N., & Austerweil, J. L. (2016). Examining search processes in low and high creative individuals with random walks. In *Proceedings of the 38th annual meeting of the cognitive science society*. Austin, TX. Retrieved from https://cogsci.mindmodeling.org/2016/papers/0066/index.html

Kenett, Y. N., Beaty, R. E., Silvia, P. J., Anaki, D., & Faust, M. (2016a). Structure and flexibility: Investigating the relation between the structure of the mental lexicon, fluid intelligence, and creative achievement. *Psychology of Aesthetics, Creativity, and the Arts*, *10*, 377–388. https://doi.org/10.1037/aca0000056

Kenett, Y. N., & Faust, M. (2019). A semantic network cartography of the creative mind. *Trends in Cognitive Sciences*, *23*, 271–274. https://doi.org/10.1016/j.tics.2019.01.007

Kenett, Y. N., & Faust, M. (2020). Clinical cognitive networks: A graph theory approach. In M. S. Vitevitch (Ed.), *Network science in cognitive science*. New York, NY: Routledge.

Kenett, Y. N., Gold, R., & Faust, M. (2016b). The hyper-modular associative mind: A computational analysis of associative responses of persons with asperger syndrome. *Language and Speech*, *59*, 297–317. https://doi.org/10.1177/0023830915589397

Kenett, Y. N., Wechsler-Kashi, D., Kenett, D. Y., Schwartz, R. G., Ben Jacob, E., & Faust, M. (2013). Semantic organization in children with cochlear implants: Computational analysis of verbal fluency. *Frontiers in Psychology*, *4*. https://doi.org/10.3389/fpsyg.2013.00543

Kim, N., Kim, J.-H., Wolters, M. K., MacPherson, S. E., & Park, J. C. (2019). Automatic scoring of semantic fluency. *Frontiers in Psychology*, *10*, 1020. https://doi.org/10.3389/fpsyg.2019.01020

Lange, K. V., Hopman, E. W., Zemla, J. C., & Austerweil, J. L. (2020). Evidence against a relation between bilingualism and creativity. *PLoS ONE*, *15*, e0234928. https://doi.org/10.1371/journal.pone.0234928%0A

Lerner, A. J., Ogrocki, P. K., & Thomas, P. J. (2009). Network graph analysis of category fluency testing. *Cognitive and Behavioral Neurology*, *22*, 45–52. https://doi.org/10.1097/WNN.0b013e318192ccaf

Levenshtein, V. I. (1966). Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, *10*, 707–710.

Mandera, P., Keuleers, E., & Brysbaert, M. (in press). Recognition times for 62 thousand English words: Data from the English Crowdsourcing Project. *Behavior Research Methods*. https://doi.org/10.31234/osf.io/wm5gv

Massara, G. P., Di Matteo, T., & Aste, T. (2016). Network filtering for big data: Triangulated Maximally Filtered Graph. *Journal of Complex Networks*, *5*, 161–178. https://doi.org/10.1093/comnet/cnw015

McCrae, R. R., & Costa, P. T. (2007). Brief versions of the NEO PI-3. *Journal of Individual Differences*, *28*, 116–128. https://doi.org/10.1027/1614-0001.28.3.116

Nelson, D. L., McEvoy, C. L., & Schreiber, T. A. (2004). The University of South Florida free association, rhyme, and word fragment norms. *Behavior Research Methods, Instruments, & Computers*, *36*, 402–407. https://doi.org/10.3758/BF03195588

Newman, M. E. J. (2006). Modularity and community structure in networks. *Proceedings of the National Academy of Sciences*, *103*, 8577–8582. https://doi.org/10.1073/pnas.0601602103

Ooms, J. (2018). *hunspell: High-performance stemmer, tokenizer, and spell checker*. Retrieved from https://CRAN.R-project.org/package=hunspell

Pakhomov, S. V. S., Eberly, L., & Knopman, D. (2016). Characterizing cognitive performance in a large longitudinal study of aging with computerized semantic indices of verbal fluency. *Neuropsychologia*, *89*, 42–56. https://doi.org/10.1016/j.neuropsychologia.2016.05.031

Paulsen, J. S., Romero, R., Chan, A., Davis, A. V., Heaton, R. K., & Jeste, D. V. (1996). Impairment of the semantic network in schizophrenia. *Psychiatry Research*, *63*, 109–121. https://doi.org/10.1016/0165-1781(96)02901-0%20

Pisanski, T., Pisanski, M., & Pisanski, J. (2020). A novel method for determining research groups from co-authorship network and scientific fields of authors. *Informatica*, *44*. https://doi.org/10.31449/inf.v44i2.307

Politis, D. N., & Romano, J. P. (1994). Large sample confidence regions based on subsamples under minimal assumptions. *The Annals of Statistics*, 2031–2050. Retrieved from https://www.jstor.org/stable/2242497

Pratt, D., Gooding, P., Johnson, J., Taylor, P., & Tarrier, N. (2010). Suicide schemas in non-affective psychosis: An empirical investigation. *Behaviour Research and Therapy*, *48*, 1211–1220. https://doi.org/10.1016/j.brat.2010.08.005

Quirin, A., Cordón, O., Guerrero-Bote, V. P., Vargas-Quesada, B., & Moya-Anegón, F. (2008). A quick MST-based algorithm to obtain Pathfinder networks (∞, n- 1). *Journal of the American Society for Information Science and Technology*, *59*, 1912–1924. https://doi.org/10.1002/asi.20904

Rinker, T. W. (2020). *qdap: Quantitative discourse analysis package*. Buffalo, New York. Retrieved from http://github.com/trinker/qdap

Schvaneveldt, R. W. (1990). *Pathfinder associative networks: Studies in knowledge organization*. Norwood, NJ: Ablex Publishing.

Shao, J. (2003). Impact of the bootstrap on sample surveys. *Statistical Science*, *18*(2), 191–198. Retrieved from https://www-jstor-org.libproxy.uncg.edu/stable/3182849%20

Shreestha, R., Shaw, B., Woyczynski, W., Thomas, P. J., Fritsch, T., & Lerner, A. J. (2015). Growth and evolution of category fluency network graphs. *Journal of Systems and Integrative Neuroscience*, *1*, 6–13. https://doi.org/10.15761/JSIN.1000103

Siew, C. S. Q. (2013). Community structure in the phonological network. *Frontiers in Psychology*, *4*, 553. https://doi.org/10.3389/fpsyg.2013.00553

Siew, C. S. Q. (2019). spreadr: An R package to simulate spreading activation in a network. *Behavior Research Methods*, *51*, 910–929. https://doi.org/10.3758/s13428-018-1186-5

Siew, C. S. Q., Wulff, D. U., Beckage, N. M., & Kenett, Y. N. (2019). Cognitive network science: A review of research on cognition through the lens of network representations, processes, and dynamics. *Complexity*, *2019*. https://doi.org/10.1155/2019/2108423

Stella, M., Beckage, N. M., Brede, M., & De Domenico, M. (2018). Multiplex model of mental lexicon reveals explosive learning in humans. *Scientific Reports*, *8*, 2259. https://doi.org/10.1038/s41598-018-20730-5

Steyvers, M., & Tenenbaum, J. B. (2005). The large-scale structure of semantic networks: Statistical analyses and a model of semantic growth. *Cognitive Science*, *29*, 41–78. https://doi.org/10.1207/s15516709cog2901_3

Tumminello, M., Aste, T., Di Matteo, T., & Mantegna, R. N. (2005). A tool for filtering information in complex systems. *Proceedings of the National Academy of Sciences*, *102*, 10421–10426. https://doi.org/10.1073/pnas.0500298102

Unsworth, N., Spillers, G. J., & Brewer, G. A. (2011). Variation in verbal fluency: A latent variable analysis of clustering, switching, and overall performance. *Quarterly Journal of Experimental Psychology*, *64*, 447–466. https://doi.org/10.1080/17470218.2010.505292

van Wijk, B. C. M., Stam, C. J., & Daffertshofer, A. (2010). Comparing brain networks of different size and connectivity density using graph theory. *PloS ONE*, *5*, e13701. https://doi.org/10.1371/journal.pone.0013701

Viger, F., & Latapy, M. (2016). Efficient and simple generation of random simple connected graphs with prescribed degree sequence. *Journal of Complex Networks*, *4*, 15–37. https://doi.org/10.1093/comnet/cnv013

Watts, D. J., & Strogatz, S. H. (1998). Collective dynamics of "small-world" networks. *Nature*, *393*, 440–442. https://doi.org/10.1038/30918

Wickham, H. (2016). *ggplot2: Elegant graphics for data analysis*. Springer. Retrieved from https://ggplot2-book.org/

Wulff, D. U., De Deyne, S., Jones, M. N., Austerweil, J. L., Baayen, R. H., Balota, D. A., ... Mata, R. (2019). New perspectives on the aging lexicon. *Trends in Cognitive Sciences*, *23*, 686–698. https://doi.org/10.1016/j.tics.2019.05.003

Wulff, D. U., Hills, T., & Mata, R. (2018). Structural differences in the semantic networks of younger and older adults. https://doi.org/10.31234/osf.io/s73dp

Zemla, J. C., & Austerweil, J. L. (2018). Estimating semantic networks of groups and individuals from fluency data. *Computational Brain & Behavior*, *1*, 36–58. https://doi.org/10.1007/s42113-018-0003-7

Zemla, J. C., Cao, K., Mueller, K. D., & Austerweil, J. L. (2020). SNAFU: The semantic network and fluency utility. *Behavior Research Methods*, 1–19. https://doi.org/10.3758/s13428-019-01343-w

Supplementary Information

Semantic Network Analysis (SemNA): A Tutorial on Preprocessing, Estimating, and Analyzing Semantic Networks

SI 1. R Session Information

```
R version 4.0.2 (2020-06-22)
Platform: x86_64-w64-mingw32/x64 (64-bit)
Running under: Windows 10 x64 (build 19041)

Matrix products: default

locale:
[1] LC_COLLATE=English_United States.1252
[2] LC_CTYPE=English_United States.1252
[3] LC_MONETARY=English_United States.1252
[4] LC_NUMERIC=C
[5] LC_TIME=English_United States.1252

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods   base

other attached packages:
 [1] shinyBS_0.61             shinyMatrix_0.3.0        shinyalert_2.0.0
 [4] shinyjs_2.0.0            shiny_1.5.0              SemNeT_1.4.0
 [7] SemNetCleaner_1.3.0      SemNetDictionaries_0.1.7 kableExtra_1.2.1
[10] knitr_1.30               knitcitations_1.0.10     papaja_0.1.0.9997
```

```
loaded via a namespace (and not attached):
 [1] tidyselect_1.1.0  xfun_0.17         purrr_0.3.4       colorspace_1.4-1
 [5] generics_0.0.2    vctrs_0.3.4       htmltools_0.5.0   viridisLite_0.3.0
 [9] yaml_2.2.1        rlang_0.4.7       pillar_1.4.6      later_1.1.0.1
[13] glue_1.4.2        lifecycle_0.2.0   plyr_1.8.6        stringr_1.4.0
[17] munsell_0.5.0     gtable_0.3.0      rvest_0.3.6       evaluate_0.14
[21] fastmap_1.0.1     httpuv_1.5.4      Rcpp_1.0.5        xtable_1.8-4
[25] scales_1.1.1      promises_1.1.1    webshot_0.5.2     jsonlite_1.7.1
[29] mime_0.9          ggplot2_3.3.2     png_0.1-7         digest_0.6.25
[33] stringi_1.5.3     bookdown_0.20     dplyr_1.0.2       grid_4.0.2
[37] bibtex_0.4.2.3    tools_4.0.2       magrittr_1.5      tibble_3.0.3
[41] RefManageR_1.2.12 crayon_1.3.4.9000 pkgconfig_2.0.3   ellipsis_0.3.1
[45] xml2_1.3.2        lubridate_1.7.9   rmarkdown_2.3     httr_1.4.2
[49] rstudioapi_0.11   R6_2.4.1          compiler_4.0.2
```

SI 2. Additional R Packages for Shiny Application

```
# Shiny packages

install.packages(c("shiny", "shinyjs", "shinyalert",

                   "shinyMatrix", "shinyBS"))
```

SI 3. SemNA Manuscript R Script

```
##########################################################
#### Semantic Networks Estimated from Verbal Fluency ####
##########################################################

# Install packages (and their dependencies)
install.packages(c("SemNetDictionaries", "SemNetCleaner", "SemNeT"),
                 dependencies = c("Imports", "Suggests"))

# Load packages
library(SemNetDictionaries)
library(SemNetCleaner)
library(SemNeT)

################################################
#### 1. Preprocessing Verbal Fluency Data ####
################################################

#####################################
## 1.1. SemNetDictionaries Package ##
#####################################

#---------------------------------#
# 1.1.1. Pre-defined dictionaries #
#---------------------------------#
```

```
# Check for available dictionaries
dictionaries()

# Load 'animals' dictionary
load.dictionaries("animals")

# Load all words starting with 'f'
load.dictionaries("f")

# Load multiple dictionaries
load.dictionaries("fruits", "vegetables")

#---------------------------#
# 1.1.2. Custom dictionaries #
#---------------------------#

# Create a custom dictionary
append.dictionary("your", "words", "here", "in", "quotations",
                  "and", "separated", "by", "commas",
                  dictionary.name = "example",
                  save.location = "choose")

?append.dictionary

# Append a pre-defined dictionary
append.dictionary(animals.dictionary,
                  "tasselled wobbegong",
```

```
                  dictionary.name = "new.animals",
                  save.location = "choose")


################################
## 1.2. SemNetCleaner Package ##
################################

# Data from 'SemNetCleaner'
data("open.animals")

## Fluency data
fluency <- open.animals[,-c(1:2)]

# Run `textcleaner`
clean <- textcleaner(data = fluency, miss = 99,
                     partBY = "row", dictionary = "animals")

# Get groups
group <- ifelse(open.animals$Group == 1, "Low", "High")

#######################################################
#### Estimating and Analyzing Semantic Networks ####
#######################################################

# Run SemNeT Shiny application
SemNeTShiny()
```

SI 4. SemNA Template R Script

```
############################################################
#### Semantic Networks Estimated from Verbal Fluency ####
############################################################

# Load packages
library(SemNetDictionaries)
library(SemNetCleaner)
library(SemNeT)
library(NetworkToolbox)

#-----------------#
# Loading in data #
#-----------------#

# Your own data
my.data <- read.data()

# `read.data` will automatically load in data
# from common file extensions such as
# Excel, R, Matlab, and SPSS
#
# An interactive menu will allow you to navigate
# to the file you'd like to load and will
# load the data in the correct format for
# the pipeline
```

```
#-----------------#
# Data management #
#-----------------#

# If your data has variables that are not
# subject IDs and verbal fluency responses,
# then you should remove these variables before
# running the preprocessing step
#
# To figure out which variables you need to remove
# you can use the `colnames` function to identify
# these columns
colnames(my.data)

# Identify the numbers of the columns that need to
# be removed
#
# For example:
prep.data <- my.data[,-c(1:4, 67, 89)]

# The function `c` is concatenate, which
# creates a vector with the numbers typed
# into between the parentheses.
#
# The `[1,1]` represents elements with
# the rows on the left of the comma and
```

```
# columns on the right of the comma. In the example,
# `[1,1]`, this would be element in row 1 and column 1.
# If there is no number input for either row or column
# (e.g., `[,1]`), then the entire row or column is selected
# (e.g., all of column `1` is selected).
#
# The `:` means all numbers in-between. So, in the
# above example, we've selected columns 1 *through*
# 4, 67, and 89.
#
# The `-` removes rows or columns from the matrix.
#
# The resulting matrix should consist ONLY of
# your subject IDS and raw verbal fluency responses.
#
# If you have a grouping variable in this data,
# it is OKAY to remove it. We will come back to this later.


################################################
#### 1. Preprocessing Verbal Fluency Data ####
################################################


#################################
## 1.2. SemNetCleaner Package ##
#################################


#--------------------#
```

```
# 1.2.1. Spell check #
#--------------------#

# Documentation for `textcleaner` function
?textcleaner

# Run 'textcleaner'
clean <- textcleaner(data = prep.data, miss = 99,
                     partBY = "row", dictionary = "animals")

#-----------------#
# `data` argument #
#-----------------#
##
## This is where you can input `prep.data`

#-----------------#
# `miss` argument #
#-----------------#
##
## This argument is for missing data.
## If you have missing data and use a different
## value that NA, NaN, or empty cells, then
## an additional value can be assigned.
##
## Note that NA, NaN, and empty cells are
## automatically identified as missing data
```

```
#-------------------#
# `partBY` argument #
#-------------------#
##
## If your data has participants' responses going
## across the rows (from left to right),
## then this argument should be "row"
##
## If your data has participants' responses going
## down the columns (from top to bottom),
## then this argument should be "col"


#-----------------------#
# `dictionary` argument #
#-----------------------#
##
## This argument specifies the dictionaries
## to be used for the automated cleaning
## and manual spelling suggestion for
## the data.
##
## To see which dictionaries are available,
## type and enter: dictionaries()
##
## For letter fluency tasks,
## single letters can be used
```

```
## For example: dictionary = c("f")
##
## If no dictionary represents the appropriate
## category for spell-check, then
## the general dictionary will be used.

# Get directory to save in
dir <- choose.dir()

# Save .csv of manual changes
write.csv(clean$spellcheck$manual,
          paste(dir, "manual_changes.csv", sep = "/"),
          row.names = TRUE)



# Verbal fluency response totals
totals <- clean$behavioral$Appropriate

#######################################################
#### Estimating and Analyzing Semantic Networks ####
#######################################################

# Run SemNeT Shiny application
SemNeTShiny()
```

SI 5. Manual Chages Made in `textcleaner`

	Option Selected	Target Response	Changed To
1	3	catdog	cat, dog
2	5	mose	NA
3	8	did	bear
4	5	creatures	NA
5	3	catefrog	cat, frog
6	5	criters	NA
7	5	mario	NA
8	Q	chiuaua	chihuahua
9	W	garafi	giraffe
10	5	snack	NA
11	5	jesus	NA
12	Q	squrill	squirrel
13	5	your mom	NA
14	Q	geaniu pig	guinea pig
15	Q	crocidle	crocodile
16	Q	dinasor	dinosaur
17	Q	buffel	buffalo
18	Q	doplin	dolphin
19	5	more cats	NA
20	5	lotus	NA
21	Q	ostrig	ostrich
22	Q	pingwin	penguin
23	Q	amardillo	armadillo
24	Q	merekat sp	meerkat
25	Q	heghog	hedgehog

```
26               5          lagoon         NA
27               Q       getcho sp      gecko
28               5           oh my         NA
29               5       bablefish         NA
30               W         analope   antelope
31               5         sporian         NA
32               I           women      human
33               Q          lizers     lizard
34               W            koal      koala
35               W         atalope   antelope
36               W     teranchilla  tarantula
37               5         manster         NA
38               Q        sprikbok  springbok
```

Changes made during `textcleaner` verification.

```
$house
          from    to
Previous  "house" "mouse"
Corrected "house" "NA"


$beasts
          from     to
Previous  "beasts" "yeast"
Corrected "beasts" "NA"


$god
          from  to
Previous  "god" "cod"
```

```
Corrected "god" "NA"


$liam
          from   to
Previous  "liam" "lemur"
Corrected "liam" "NA"


$chia
          from   to
Previous  "chia" "cheetah"
Corrected "chia" "NA"


$ig
          from to
Previous  "ig" "pig"
Corrected "ig" "NA"


$`sugar bear`
          from         to
Previous  "sugar bear" "sugar glider"
Corrected "sugar bear" "NA"


$at
          from to
Previous  "at" "gnat"
Corrected "at" "NA"
```

SI 6. R Console Code for Manuscript R Script

```
#########################################################
#### Semantic Networks Estimated from Verbal Fluency ####
#########################################################

# Install packages (and their dependencies)
install.packages(c("SemNetDictionaries", "SemNetCleaner", "SemNeT"),
                 dependencies = c("Imports", "Suggests"))

# Load packages
library(SemNetDictionaries)
library(SemNetCleaner)
library(SemNeT)

###############################################
#### 1. Preprocessing Verbal Fluency Data ####
###############################################

#####################################
## 1.1. SemNetDictionaries Package ##
#####################################

#---------------------------------#
# 1.1.1. Pre-defined dictionaries #
#---------------------------------#
```

```
# Check for available dictionaries
dictionaries()

# Load 'animals' dictionary
load.dictionaries("animals")

# Load all words starting with 'f'
load.dictionaries("f")

# Load multiple dictionaries
load.dictionaries("fruits", "vegetables")

#---------------------------#
# 1.1.2. Custom dictionaries #
#---------------------------#

# Create a custom dictionary
append.dictionary("your", "words", "here", "in", "quotations",
                  "and", "separated", "by", "commas",
                  dictionary.name = "example",
                  save.location = "choose")

?append.dictionary

# Append a pre-defined dictionary
append.dictionary(animals.dictionary,
                  "tasselled wobbegong",
```

```
                  dictionary.name = "new.animals",
                  save.location = "choose")


##################################
## 1.2. SemNetCleaner Package ##
##################################

# Data from 'SemNetCleaner'
data("open.animals")

## Fluency data
fluency <- open.animals[,-c(1:2)]

# Run `textcleaner`
clean <- textcleaner(data = fluency, miss = 99,
                     partBY = "row", dictionary = "animals")



# Get groups
group <- ifelse(open.animals$Group == 1, "Low", "High")

########################################
#### 2. Estimating Semantic Networks ####
########################################

##################
## 2.1. Process ##
```

```
##################


#-----------------------------------------#
# 2.1.1. Preparation for network estimation #
#-----------------------------------------#


# Attach 'Group' variable to the binary response matrix
behav <- cbind(open.animals$Group, clean$responses$binary)


# For Community and Pathfinder networks:
## behav <- cbind(open.animals$Group, clean$responses$clean)


# Create low and high openness to experience response matrices
low <- behav[which(behav[,1]==1),-1]
high <- behav[which(behav[,1]==2),-1]


# Save binary response matrices
write.csv(low, "low_BRM.csv", row.names = TRUE)
write.csv(high, "high_BRM.csv", row.names = TRUE)


# Finalize matrices so that each response
# has been given by at least two participants
final.low <- finalize(low, minCase = 2)
final.high <- finalize(high, minCase = 2)


# Equate the responses across the networks
eq <- equate(final.low, final.high)
```

```r
equate.low <- eq$final.low
equate.high <- eq$final.high

#---------------------------#
# 2.1.2. Network estimation #
#---------------------------#

# Compute cosine similarity for the 'low' and
# 'high' equated binary response matrices
cosine.low <- similarity(equate.low, method = "cosine")
cosine.high <- similarity(equate.high, method = "cosine")

# Estimate 'low' and 'high' openness to experience networks
net.low <- TMFG(cosine.low)
net.high <- TMFG(cosine.high)

# For other networks:
## Community Network
### net.low <- CN(low)
### net.high <- CN(high)
##
## Naive Random Walk
### net.low <- NRW(low)
### net.high <- NRW(high)
##
## Pathfinder Network
### net.low <- PF(low)
```

```
### net.high <- PF(high)


# Save the networks
write.csv(net.low, "low_network.csv", row.names = FALSE)
write.csv(net.high, "high_network.csv", row.names = FALSE)


########################################
#### 3. Analyzing Semantic Networks ####
########################################


##############################################
## 3.1. Visualization of Semantic Networks ##
##############################################


# Visually compare networks
compare_nets(net.low, net.high,
             title = list("Low Openness", "High Openness"),
             config = "spring", weighted = FALSE)


###################################
## 3.2. Global Network Measures ##
###################################


# Compute network measures
semnetmeas(net.low, meas = c("ASPL", "CC", "Q"), weighted = FALSE)
semnetmeas(net.high, meas = c("ASPL", "CC", "Q"), weighted = FALSE)
```

```
############################
## 3.3. Statistical Tests ##
############################

#-------------------------------------#
# 3.3.1. Tests against random networks #
#-------------------------------------#

# Compute tests against random networks
rand.test <- randnet.test(net.low, net.high, iter = 1000, cores = 4)

#---------------------------------------#
# 3.3.2. Bootstrapped node-wise networks #
#---------------------------------------#

# Compute partial bootstrap network analysis
## 50% of nodes remaining in network
boot.fifty <- bootSemNeT(low, high, method = "TMFG", type = "node",
                         percent = .50, iter = 1000,
                         sim = "cosine", cores = 4)
## 60% of nodes remaining in network
boot.sixty <- bootSemNeT(low, high, method = "TMFG", type = "node",
                         percent = .60, iter = 1000,
                         sim = "cosine", cores = 4)
## 70% of nodes remaining in network
boot.seventy <- bootSemNeT(low, high, method = "TMFG", type = "node",
                           percent = .70, iter = 1000,
```

```
                              sim = "cosine", cores = 4)
## 80% of nodes remaining in network
boot.eighty <- bootSemNeT(low, high, method = "TMFG", type = "node",
                          percent = .80, iter = 1000,
                          sim = "cosine", cores = 4)
## 90% of nodes remaining in network
boot.ninety <- bootSemNeT(low, high, method = "TMFG", type = "node",
                          percent = .90, iter = 1000,
                          sim = "cosine", cores = 4)


# Plot bootstrap results
plots <- plot(boot.fifty, boot.sixty, boot.seventy,
              boot.eighty, boot.ninety, groups = c("Low","High"),
              measures = c("ASPL", "CC", "Q"))


# To arrange plots so they are stacked
install.packages("gridExtra")
library(gridExtra)
grid.arrange(plots$aspl, plots$cc, plots$q)


# Perform t-tests on bootstrap results
tests <- test.bootSemNeT(boot.fifty, boot.sixty, boot.seventy,
                         boot.eighty, boot.ninety)


#--------------------------------------#
# 3.3.2. Bootstrapped case-wise networks #
#--------------------------------------#
```

```
# Compute bootstrap network analysis
boot <- bootSemNeT(low, high, method = "TMFG",
                   type = "case", iter = 1000,
                   sim = "cosine", cores = 4)

# See documentation: ?bootSemNeT

# For other methods:
## Community Network
### boot <- bootSemNeT(low, high, method = "CN",
###                    type = "case", iter = 1000,
###                    cores = 4)
##
## Naive Random Walk
### boot <- bootSemNeT(low, high, method = "NRW",
###                    type = "case", iter = 1000,
###                    cores = 4)
##
## Pathfinder Network
### boot <- bootSemNeT(low, high, method = "PF",
###                    type = "case", iter = 1000,
###                    cores = 4)
```

SI 7. Random Network Analyses Results for Network Estimation Methods

Community Network

	High (p-value)	M.rand	SD.rand	Low (p-value)	M.rand	SD.rand
ASPL	0	4.1988	0.0685	0	4.0561	0.0901
CC	0	0.0219	0.0095	0	0.0277	0.0127
Q	0	0.5651	0.0102	0	0.5610	0.0119

Naïve Random Walk

	High (p-value)	M.rand	SD.rand	Low (p-value)	M.rand	SD.rand
ASPL	0	2.1553	0.0072	0	2.1105	0.0076
CC	0	0.3929	0.0096	0	0.4222	0.0113
Q	0	0.1477	0.0039	0	0.1431	0.0040

Pathfinder Network

	High (p-value)	M.rand	SD.rand	Low (p-value)	M.rand	SD.rand
ASPL	0	2.0622	0.0036	0	2.1212	0.0039
CC	0	0.8041	0.0067	0	0.7360	0.0091
Q	0	0.0543	0.0016	0	0.0680	0.0018

SI 8. (Case-wise) Bootstrap Network Analyses Results for Network Estimation Methods

Correlation-based Network

	Adj. M. High	Adj. M. Low	df	Residual df	F-statistic	p-value
ASPL	3.129	3.259	1	1996	341.289	< .001
CC	0.736	0.731	1	1996	284.632	< .001
Q	0.636	0.65	1	1996	370.225	< .001

	Partial Eta Squared	Direction
ASPL	0.146	High < Low
CC	0.125	High > Low
Q	0.156	High < Low

Community Network

	Adj. M. High	Adj. M. Low	df	Residual df	F-statistic	p-value
ASPL	3.272	3.303	1	1996	282.771	< .001
CC	0.27	0.275	1	1996	45.629	< .001
Q	0.492	0.511	1	1996	1052.190	< .001

	Partial Eta Squared	Direction
ASPL	0.124	High < Low
CC	0.022	High < Low
Q	0.345	High < Low

Naïve Random Walk

	Adj. M. High	Adj. M. Low	df	Residual df	F-statistic	p-value
ASPL	2.271	2.209	1	1996	14988.423	< .001
CC	0.35	0.368	1	1996	1009.267	< .001
Q	0.199	0.186	1	1996	1514.573	< .001

	Partial Eta Squared	Direction
ASPL	0.882	High > Low

CC	0.336	High < Low
Q	0.431	High > Low

Pathfinder Network

	Adj. M. High	Adj. M. Low	df	Residual df	F-statistic	p-value
ASPL	4.245	4.341	1	1996	54.912	< .001
CC	0.539	0.522	1	1996	479.105	< .001
Q	0.135	0.15	1	1996	510.675	< .001

	Partial Eta Squared	Direction
ASPL	0.027	High < Low
CC	0.194	High > Low
Q	0.204	High < Low