discourseGT: An R package to analyze discourse networks in educational contexts

1 discourseGT Workflow

1.1 General Workflow

The functions of **discourseGT** were designed to be as modular as possible, making it possible to only run analyses of interest. Figure 1 represents the general workflow of **discourseGT**, and Table 1 describes explicit function names organized by their general uses.

Figure 1: General workflow of **discourseGT**. The raw data can either be converted to an **igraph** object for further analysis or directly passed for NGT analysis. All console output can be permanently stored to the user's local disk. Green represents the start of the workflow. Purple represents steps necessary to generate an **igraph** object. Blue represents the potential downstream uses of an **igraph** object. Orange represents NGT analysis. Red signals the end of the workflow.

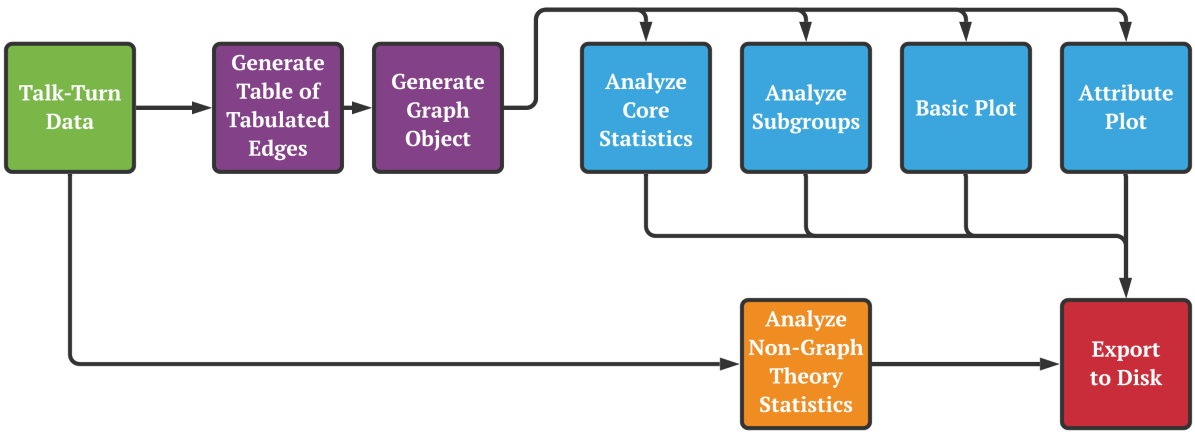


Table 1: List of all **discourseGT** functions

Phase	Function Name	Parameter(s)	Description
Purple	`tabulate_edges()`	`input` – `data.frame` or `string`. Points to `.csv` file with talk-turn data in the question-and-response format. `silentNodes` – `integer`. The number of nodes that do not interact with others.	Calculates the weighted edge list from the input data and number of silent nodes not captured in the data.
Purple	`prepareGraphs()`	`raw_data_input` – `list`. Output of `tabulate_edges()`. `project_title` – `string`. Sets the title of the project. `weightedGraph` - `boolean`. `TRUE` if downstream analysis should account for weighted edges. Else `FALSE`.	Prepares the **igraph** object from the weighted edge list. This is utilized by several downstream analytical functions.
Blue	`coreNetAnalysis()`	`ginp` – `list`. Output of `prepareGraphs()`.	Analyzes the input **igraph** object and returns basic network statistics, as reasoned in Chai et al. 2019.

Phase	Function Name	Parameter(s)	Description
Blue	`subgroupsNetAnalysis()`	`ginp` – `list`. Output of `prepareGraphs()`. `raw_input` – `data.frame`. Points to the original talk-turn data in the question-and-response format. `normalized` – `boolean`. Whether or not to normalize the betweenness centrality values relative to the graph.	Analyzes the input **igraph** object for potential subgroups.
Blue	`summaryNet()`	`netintconfigData` – `list`. Output of `prepareGraphs()`. `coreNetAnalysisData` – `list`. Output of `coreNetAnalysis()`. `subgroupsNetAnalysisData` – `list`. Output of `subgroupsNetanalysis()`. `display` – `boolean`. Whether or not to print output to console.	Summarizes the analytical output from several other functions into a single output.

Phase	Function Name	Parameter(s)	Description
Blue	`basicPlot()`	`ginp` – `list`. Output of `prepareGraphs()`. `graph_selection_input` – `integer`. Numerical value from 0 to 2, inclusive, which selects the graphing algorithm used. 0 = Fruchterman Reingold, 1 = Kamada Kawai, and 2 = Reingold Tilford. `curvedEdgeLines` – `boolean`. Whether or not to curve graph edges. `arrowSizeMultiplier` – `numeric`. Scales arrow sizes based on input factor. `logscale` – `boolean`. If `TRUE`, scale graph edges logarithmically. Else do not. `logBase` – `integer`. Logarithmic base to scale graph edges.	Plots a basic network graph utilizing the default R visualization backend.

Phase	Function Name	Parameter(s)	Description
Blue	`plot1Att()`	`data` – `list`. Output of `prepareGraphs()`. `prop` – `integer`. Rescales the graph edge sizes. `graphmode` – `string`. Specifies the graphing algorithm used. Refer to `gplot.layout` for more options. `attribute` – `list`. Mapping to the attribute information. `attribute.label` – `string`. Name of attribute to display in the graph. `attribute.node.labels` – `list`. Mapping to the node labels. `attribute.nodesize` – `integer` or `list`. Mapping to universal or individualized node sizes, respectively.	Plots a network graph with a single input attribute. Utilizes the **ggplot2** [@R-ggplot2] backend.

Phase	Function Name	Parameter(s)	Description
Blue	`plot2Att()`	`data` – `list`. Output of `prepareGraphs()`. `prop` – `integer`. Rescales the graph edge sizes. `graphmode` – `string`. Specifies the graphing algorithm used. Refer to `gplot.layout` for more options. `attribute1` – `list`. Mapping to the first attribute information. `attribute2` – `list`. Mapping to the second attribute information. `attribute1.label` – `string`. Name of the first attribute to display in the graph. `attribute2.label` – `string`. Name of the second attribute to display in the graph. `attribute.node.labels` – `list`. Mapping to the node labels. `attribute.nodesize` – `integer` or `list`. Mapping to universal or individualized node sizes, respectively.	Plots a network graph with two input attributes. Utilizes the **ggplot2** [@R-ggplot2] backend.

Phase	Function Name	Parameter(s)	Description
Orange	`plotNGTData()`	`data` – `data.frame` or `string`. Points to `.csv` file with talk-turn data in the question-and-response format. `convoMinutes` – `integer`. Length of conversation, in minutes. `silentNode` – `integer`. The number of nodes that do not interact with others.	Analyzes non-graph theory statistics and visualizes them in three plots. These are elaborated on in Chai et al. 2019.
Red	`writeData()`	`project_name` – `string`. Sets the title of the project. `objectfile` – `list`. The object to be exported to disk. `dirpath` – `string`. The location on disk where the exported file will be written.	Writes any data object file as an appropriate format to a specified user directory. Images are saved with a resolution of 300dpi.

1.2 Data Structure

Collecting and formatting data for analysis by **discourseGT** is based on episodes and talk-turns [Chai et al., 2019]. Talk-turn data should be recorded as participants speak sequentially, which can be done with life observations in real time [Chai et al., 2019] or analysis of video or audio transcripts [Liyanage et al., 2021]. Be prepared to record the duration of the discussion (in minutes), which is required to determine the number of episode starts and episode continuations per unit of time. Talk-turn data are collected in a two-column table that tracks episode starts (`ep_start`) and episode continuations (`ep_cont`) and with each participant in the group assigned a unique identifier, such as a number (Table 2). Each row should only have a single participant's identifier entered once either in the `ep_start` or `ep_cont` column. An entry in the `ep_start` column denotes the beginning of a new episode. The boundaries of an episode are defined by the researcher and the research question, although these definitions should be set consistently within a study. It is vital that the column

names in the data are explicitly labeled as `ep_start` and `ep_cont`, respectively. Raw data may be prepared using most spreadsheet software or text editors, but it should ultimately be saved as a comma-separated file (`.csv`).

Table 2: Formatted talk-turn data ready for **discourseGT** analysis. In this example, an episode is defined arbitrarily as a topic (not shown) — that is, each episode is a relevant discussion on a single topic. There are two episodes. The first episode is three talk-turns long, with Participant 1 initiating the episode. Participant 3 then spoke, followed by Participant 2. The second episode has two talk-turns, with Participant 4 starting a new episode and Participant 2 speaking next to complete the overall discussion. It is important to note that the duration of the conversation (in minutes) is not a part of the table. Rather, it should be recorded elsewhere for use in NGT analysis.

ep_start	**ep_cont**
1	NA
NA	3
NA	2
4	NA
NA	2

2 Worked Case Example

The **discourseGT** software package comes equipped with example data. Here, we will utilize these data to demonstrate its utility in examining discourse networks.

To get started, install the software package through the Comprehensive R Archive Network (CRAN). Load it using:

```
library(discourseGT)
```

2.1 Importing Data

Raw data can be imported using the `read.csv()` function. For the sake of utilizing the example data, however, it is useful to duplicate it by assigning its values to a new variable. Once it has been duplicated, view the head of the data to ensure that it has been properly imported:

```
data <- sampleData1
head(data)
```

```
##   ep_start ep_cont
## 1        1      NA
## 2       NA       3
## 3       NA       4
## 4       NA       1
```

```
## 5       NA       2
## 6       NA       1
```

2.2 Preparing the igraph Object

Prior to generating the **igraph** object, a weighted edge list needs to be generated from the imported raw data. By default, the weight of an edge is defined as the number of times an edge has occurred between two nodes. Weights can be redefined based on other available criteria, but this must be done manually.

```
# Calculate the weighted edge list
tabEdge <- tabulate_edges(data, silentNodes = 0)
# Check the weighted edge list
head(tabEdge$master)
```

```
##   source target weight
## 1      1      1      8
## 2      2      1     25
## 3      3      1     49
## 4      4      1     75
## 5      1      2     28
## 6      3      2     11
```

Recall that an **igraph** object is the core input to many of the modular analytical functions offered in **discourseGT**. To generate an **igraph** object, the following information is required:

- The variable that stores the weighted edge list
- The title of the project. Default: `null`
- Is the graph weighted? Default: `TRUE`

```
prepNet <- prepareGraphs(tabEdge,
                         project_title = "Sample Data 1",
                         weightedGraph = TRUE)
```

The graph settings specified by this function will influence the analytical output of downstream functions.

2.3 Running Graph Theory Analysis

discourseGT offers graph theory-based analytics via two separate functions. The first, `coreNetAnalysis()`, will perform core operations that produce the parameters. It will count the number of nodes, and edges, calculate edge weights, average graph degree, modularity, centrality, and related graph theory parameters. To run the function and store it in a variable:

```
coreNet <- coreNetAnalysis(prepNet)
```

The second, `subgroupsNetAnalysis()`, utilizes the Girvan-Newman algorithm to detect subgroups within the overall network [Girvan and Newman, 2002], such that:

```
subNet <- subgroupsNetAnalysis(prepNet, raw_input = data,
                               normalized = TRUE)
```

2.4 Generating Summaries

While it is possible to display the generated **igraph** object, core network statistics, and subgroup statistics as separate outputs, it can be helpful to view them as an overall summary of a network's graph theory analytics. Furthermore, combining all of these outputs into a single variable is a necessary step in exporting them as a single text file. The `summaryNet()` function will combine the outputs from `prepareGraphs()`, `coreNetAnalysis()`, and `subgroupsNetAnalysis()` as such:

```
summaryData <- summaryNet(netintconfigData = prepNet,
                          coreNetAnalysisData = coreNet,
                          subgroupsNetAnalysisData = subNet,
                          display = TRUE)
```

```
## ================== BEGIN SUMMARY ==================
## discourseGT R Package - Production
## Package Version: [1] '1.1.8.9000'
## Graph Results - Project Summary
##
## ---------------PROJECT DETAILS---------------
## Name of Project:  Sample Data 1
## Summary Results Generated On: [1] "2023-07-07 19:15:37 PDT"
##
## ---------------GRAPH CONFIGURATION---------------
## Weighted Graph:  TRUE
##
## ---------------CORE PARAMETERS ANALYSIS---------------
## Number of Edges:  12
## Number of Nodes:  4
## Weighted Edges:  465
## Graph Adjacency Matrix:
## 4 x 4 sparse Matrix of class "dgCMatrix"
##    1  2  3  4
## 1  . 28 47 74
## 2 25  . 13 14
## 3 49 11  . 52
## 4 75 13 52  .
##
## Network Density: 1
## Average Degree:  6
```

```
## Strong/Weak Interactions:
## 1 2 3 4
## 1 1 1 1
##
## Unrestricted Modularity:
##
## ---------------GRAPH CENTRALITY---------------
## Degree Centrality:
## $res
## [1] 6 6 6 6
##
## $centralization
## [1] 0
##
## $theoretical_max
## [1] 12
##
##
## Articulation Points List:
## + 0/4 vertices, named, from 91abd58:
##
## Reciprocity:  1
##
## ---------------SUBGROUPS AND MODULARITY---------------
## Girvan-Newman Subgroups Detection:
## IGRAPH clustering edge betweenness, groups: 1, mod: 0
## + groups:
##   $`1`
##   [1] "1" "2" "3" "4"
##
##
## Betweeness:
## 1 2 3 4
## 0 1 0 0
##
## Normalized Betweeness:  TRUE
##
## Group Core Members:
## 1 2 3 4
## 6 6 6 6
##
## Graph Symmetry of Members:
## $mut
## [1] 6
##
```

```
## $asym
## [1] 0
##
## $null
## [1] 0
##
##
## Graph Connectedness Census:
##
## 4
## 1
##
## Neighborhood List for Each Adjacent Node:
## [[1]]
## + 4/4 vertices, named, from 91abd58:
## [1] 1 2 3 4
##
## [[2]]
## + 4/4 vertices, named, from 91abd58:
## [1] 2 1 3 4
##
## [[3]]
## + 4/4 vertices, named, from 91abd58:
## [1] 3 1 2 4
##
## [[4]]
## + 4/4 vertices, named, from 91abd58:
## [1] 4 1 2 3
##
##
## Transitivity/Clustering Coefficients:
## Local Transitivity values:
## [1] 1 1 1 1
## Global Transitivity values:
## [1] 1
##
##
## ================== END SUMMARY ==================
```

2.5 Basic Visualization

discourseGT offers several methods to visualize networks. For a basic network graph, `basicPlot()` should be used, and its parameters should be modified to suit the needs of the user. These options include modifications to the plotting algorithm, edge curvature, arrow size, and edge weight scaling.

Its default plotting algorithm is Fruchterman Reingold, denoted by `0` [Fruchterman and Reingold, 1991]. This is typically the best option to use because it attempts to minimize edge intersections in the final plot, improving readabiliy. Other projections include Kamada Kawai [Kamada and Kawai, 1989] and Reingold Tilford [Reingold and Tilford, 1981], denoted by `1` and `2`, respectively.

Edge curvature defaults to `TRUE` so that differences in talk-turn taking between nodes can be distinguished. Consider two participants, represented as Node A and Node B. It is entirely possible for Node A to talk after Node B more than Node B talks after Node A. Consequently, the two edges that point in each direction will have different weights, and these can only be visually seen if they are curved instead of overlapping. On the other hand, graphs without curved edges may improve clarity. This can be especially favorable when plotting an unweighted graph.

To modify arrow sizes, a multiplier can be passed to `arrowSizeMultiplier`. The default value is `1`. Any values <1.0 will shrink the arrow, and vice versa. Again, this feature is added to improve readability in specific cases.

Lastly, edge weight scaling is best used for improved visualization of larger, weighted datasets. Due to the increase in raw edges, default plotting may yield unreadable results. We implemented Equation 1 to do so according to a linear scale. This method allows for users to visually compare talk-turn frequencies within a graph, which is not as intuitive with other forms of scaling.

$$\mathrm{y} = \frac{(\texttt{scaledMax} - \texttt{scaledMin}) \cdot (\mathrm{eachEdgeWeight} - \mathrm{rawMin})}{\mathrm{rawMax} - \mathrm{rawMin}} + \texttt{scaledMin} \qquad (1)$$

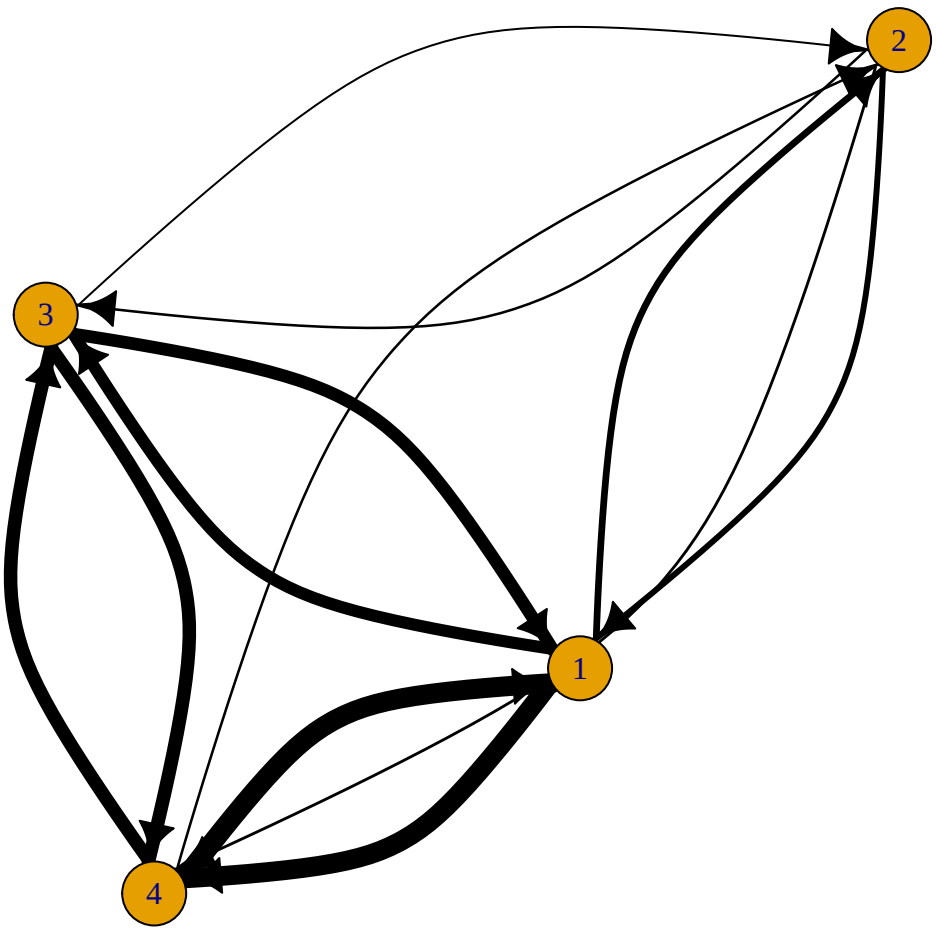
Here, each edge weight is individually scaled to a new value y. `scaledMax` and `scaledMin` are the user-defined boundaries of a new scale for all weighted edges. $rawMin$ and $rawMax$ are the minimum and maximum edge weights that are extracted from the raw data via the `prepareGraphs()` function. $eachEdgeWeight$ refers to the weight of each unique edge.

For users, `scaledMax` must be greater than or equal to `scaledMin`. These variables may also be set to equal, non-zero values to produce an unweighted version of the graph.

Note that while both `scaledMin` and `scaledMax` can theoretically be set to `0`, we advise against this because the resulting graph will appear to have no edges. Likewise, if `scaledMin` is set to `0` while `scaledMax` is a non-zero value, the resulting graph will appear to have no edges where the most infrequent talk-turns occurred. This may have some functionality depending on the user's use-case.

Below is an example of a graph that uses the Fruchterman Reingold projection, linearly scales the dataset to new weighted edge boundaries of `[1, 10]`, and applies a scale of 2 to the arrow sizes.

```
basicPlot(prepNet, graph_selection_input = 0, curvedEdgeLines = TRUE,
    arrowSizeMultiplier = 2, scaledEdgeLines = TRUE, scaledMin = 1,
    scaledMax = 10)
```



In this plot, it can be easily seen that the fewest number of talk-turns relative to the entire discourse network occurred between Nodes 2 and 3 as well as Nodes 2 and 4. Nodes 1 and 2 shared the next fewest number of talk-turns, followed by Nodes 1 and 3 and Nodes 3 and 4. Nodes 1 and 4 shared the greatest number of talk-turns between them. In each of these node pairs, the conversation appeared to travel equally between the nodes involved, as the edges of similar thickness indicate. Note that we cannot view any attribute data about the nodes here.

2.6 Attribute Visualization

To add attributes to a network graph, the `plot1Att()` and `plot2Att()` functions can be used. These functions utilize the `ggplot2` backend with `GGally` [Wickham et al., 2021, Schloerke et al. [2021]], giving them an appearance distinct from the previously discussed `basicPlot()` function.

Before starting, ensure that a properly formatted `data.frame` with attributes is in the working environment. Displayed below is an example attribute dataset included with **dis-**

courseGT:

```
attData <- attributeData
head(attData)
```

```
##   node gender        ethnicity current_gpa first_generation stem_major
## 1    1 female            white        3.56               no        yes
## 2    2   male            white        3.26              yes         no
## 3    3 female            asian        3.46               no        yes
## 4    4   male african_american        3.60              yes        yes
##                major course_reason class_level number_prior_ap residency
## 1    bioengineering         major      junior               0        CA
## 2 political_science            ge      senior               2        CA
## 3           biology         major   sophomore               3        CA
## 4         chemistry      elective      junior               4        WA
##   sat_score
## 1      1323
## 2      1449
## 3      1228
## 4      1494
```

Note that the first column, `node`, contains each node name that was included in the initial imported data. This is a crucial aspect to the attribute data because it identifies attributes associated with particular nodes for `plot1Att()` and/or `plot2Att()`.

Similarly to the `basicPlot()` function, the attribute plotting functions include options to modify the overall projection, albeit less granular. These include edge scaling, node sizes, and plotting algorithm.

Edge weight scaling can be modified by changing the value of `prop`, and node sizes can be modified by changing the value of `attribute.nodesize`. Each of these have a default value of `20`, although this is arbitrary. The user should find the best settings that suit their use case.

The default plotting algorithm is again Fruchterman Reingold for its readability [Fruchterman and Reingold, 1991]. Here, however, this option is indicated by passing `fruchtermanreingold` into the function. Other projections can be found with `gplot.layout`.

Lastly, it is important to note that only 1 or 2 attributes can be plotted at once. These cases should utilize the `plot1Att()` and `plot2Att()` functions, respectively.

Below is an example of an attribute graph with larger-than-default edge sizes and smaller-than-default node sizes. It utilizes the Fruchterman Reingold projection.

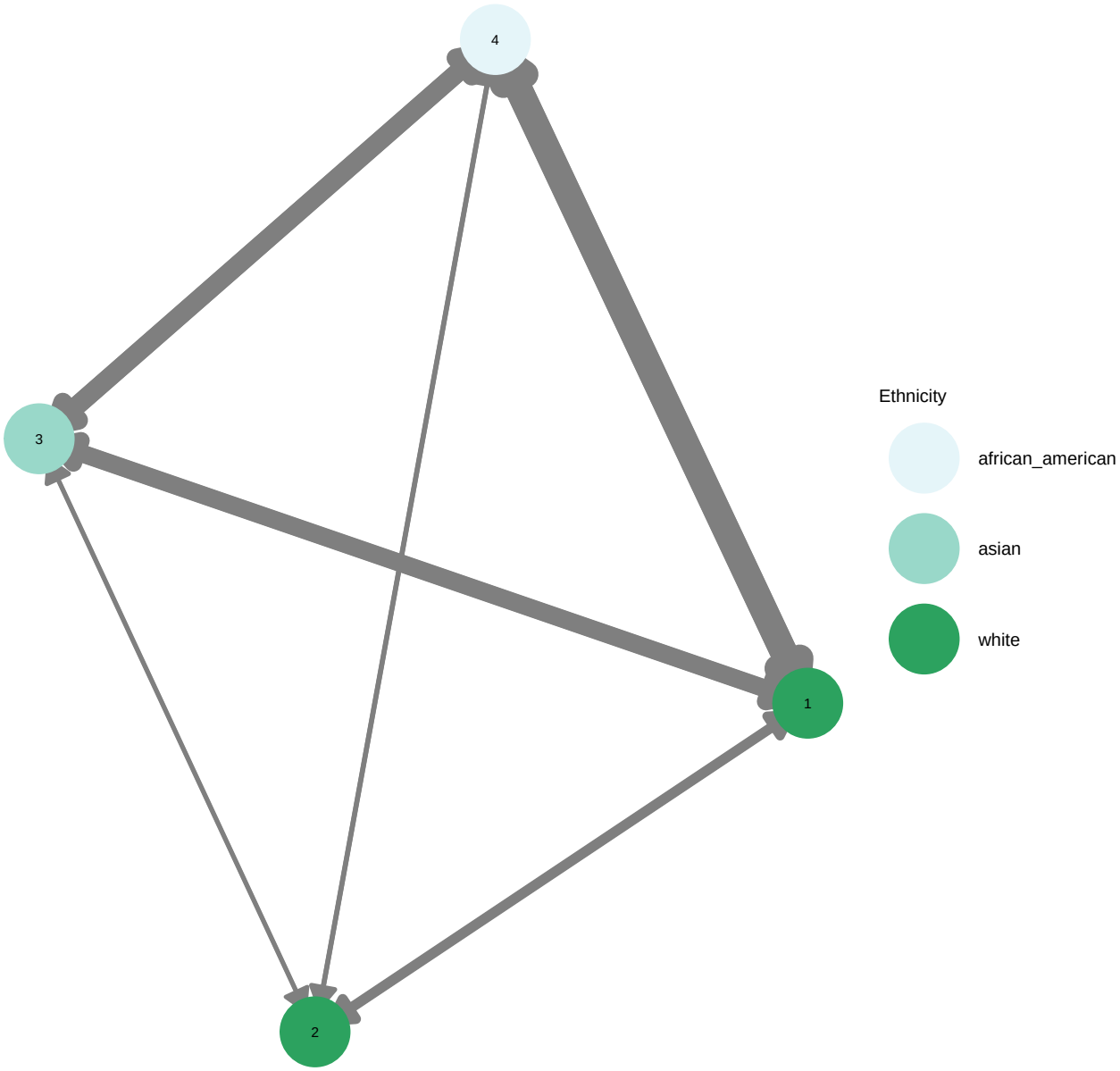
```
plot1Att(prepNet,
         prop = 40,
         graphmode = "fruchtermanreingold",
         attribute = attData$ethnicity,
```

```
        attribute.label = "Ethnicity",
        attribute.node.labels = attData$node,
        attribute.nodesize = 16)
```

```
## Registered S3 method overwritten by 'GGally':
##   method from
##   +.gg   ggplot2
```

```
## $g2plot
```

Sample Data 1



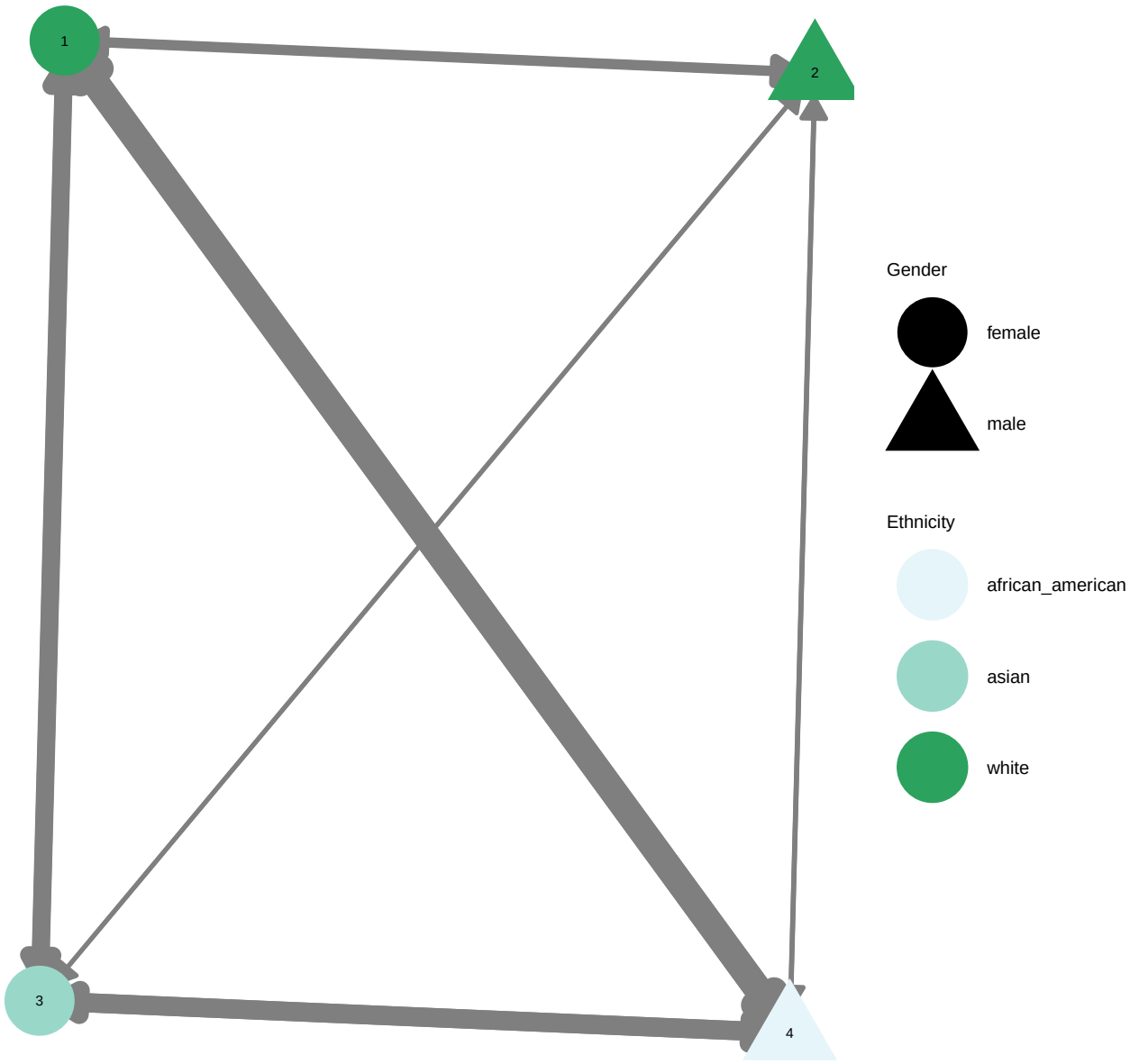

```
##
## $saveDataVar
```

```
## [1] 1
```

To plot a second attribute to a network, utilize `plot2Att()` with the aforementioned notation. The following graph showcases the network with both ethnic and gender data:

```
plot2Att(prepNet,
         prop = 40,
         graphmode = "fruchtermanreingold",
         attribute1 = attData$ethnicity,
         attribute2 = attData$gender,
         attribute1.label = "Ethnicity",
         attribute2.label = "Gender",
         attribute.node.labels = attData$node,
         attribute.nodesize = 16)
```

```
## $g2plot
```




```
##
## $saveDataVar
## [1] 2
```

2.7 Customizable Visualization

Further graph customizability, such as node placements, can be achieved with **Cytoscape**, an open-source network plotting software [Shannon et al., 2003]. In order to utilize this method:

1. Download & install **Cytoscape**.
2. Install `RCy3` [Pico et al., 2021] using the `BiocManager` package [Morgan and Ramos, 2021].

3. Plot the **igraph** object and modify it in **Cytoscape**.

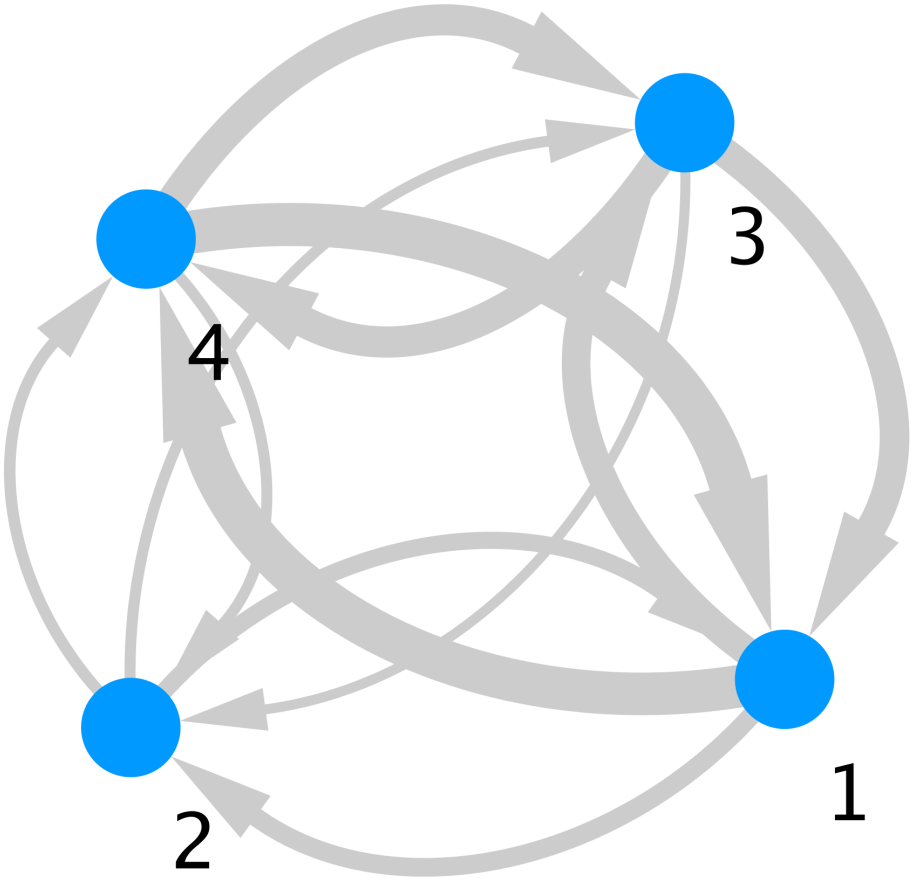
Assuming that **Cytoscape** is installed, install and load `RCy3` to properly link it to R. This can be done by:

```
install.packages("BiocManager")
BiocManager::install("RCy3")
library(RCy3)
```

To plot a graph, first ensure that a new **Cytoscape** session is loaded. Then, utilize the following command to send an **igraph** object to the GUI:

```
createNetworkFromIgraph(prepNet$graph)
```

The graph will now appear in **Cytoscape**, where further modifications can be made.



2.8 Running Non-Graph Theory Analysis

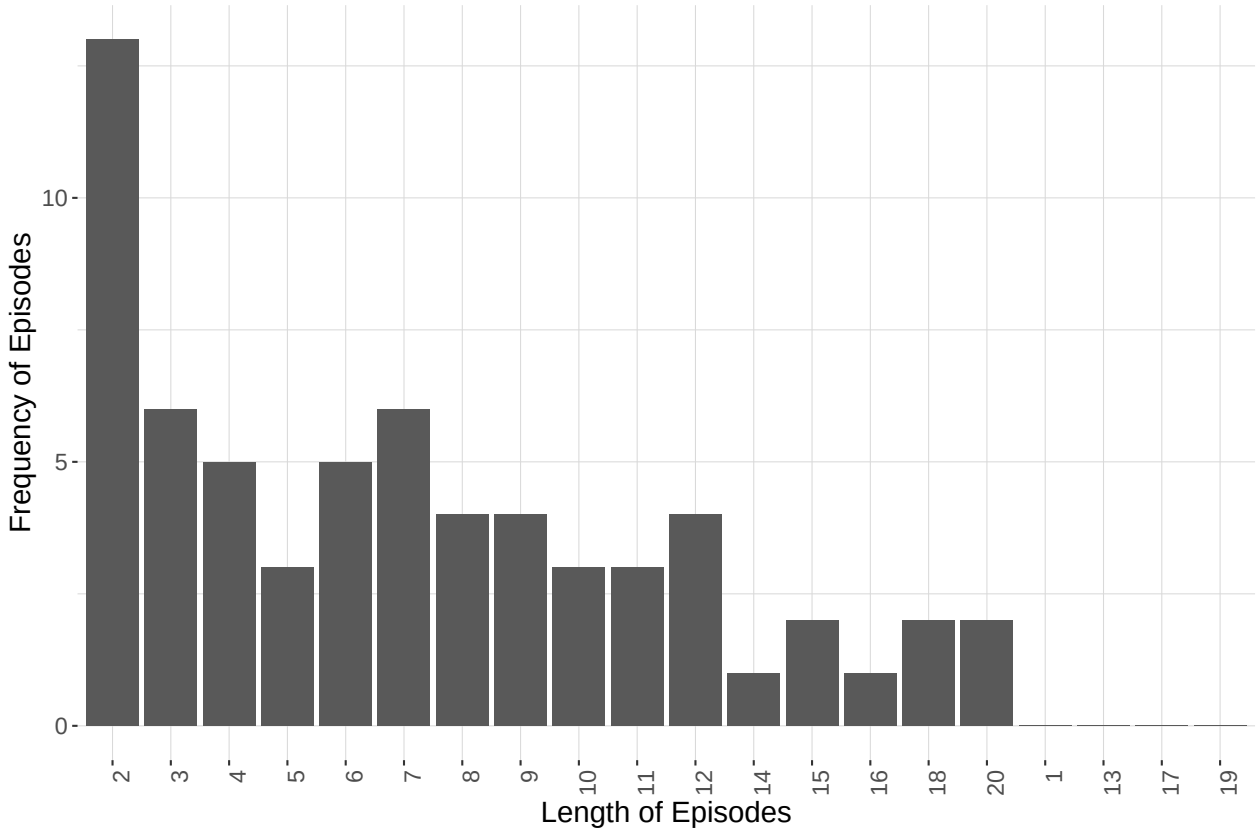
Recall that **discourseGT** does not require an **igraph** object to produce NGT analysis. Rather, `plotNGTData()` utilizes the raw, two column data to generate its output. Additionally,

it requires the duration of the conversation (in minutes) and the number of silent nodes (i.e. participants who did not speak at all) in the discourse network. The function outputs the previously-discussed NGT parameters and three individual graphs. The raw data are also exported alongside the graphs, giving the user greater flexibility in creating their own NGT visualizations.

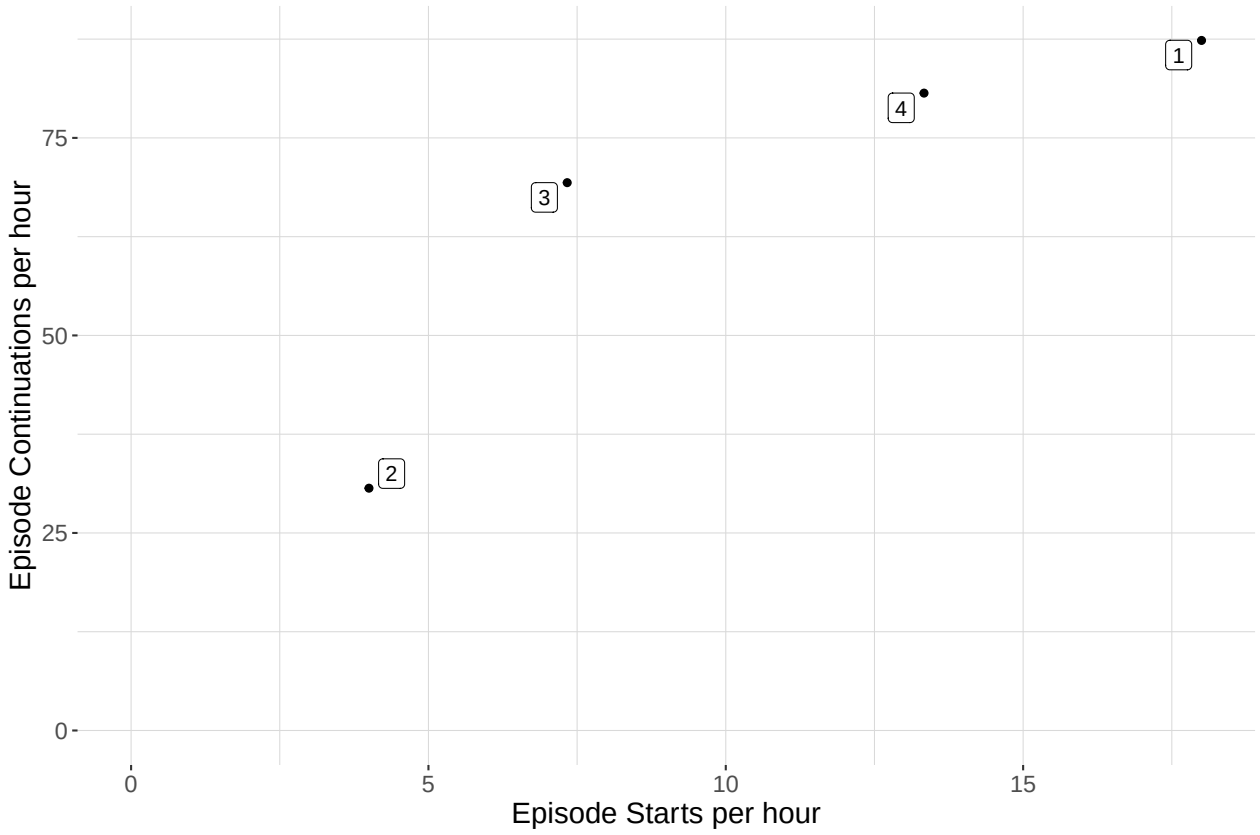
```
options(width = 60)
plotNGTData(data = data, convoMinutes = 90, silentNodes = 0)
```

```
## $ngt_std_stats1
##   participant ep_start ep_cont total_count
## 1           1       27     131         158
## 2           2        6      46          52
## 3           3       11     104         115
## 4           4       20     121         141
##   total_edges_in_out edge_by_part ep_starts_hour
## 1                314          157      18.000000
## 2                104           52       4.000000
## 3                230          115       7.333333
## 4                282          141      13.333333
##   ep_conts_hour
## 1      87.33333
## 2      30.66667
## 3      69.33333
## 4      80.66667
##
## $ngt_std_stats2
##    length_of_ep freq_of_ep
## 1             2         13
## 2             3          6
## 3             4          5
## 4             5          3
## 5             6          5
## 6             7          6
## 7             8          4
## 8             9          4
## 9            10          3
## 10           11          3
## 11           12          4
## 12           14          1
## 13           15          2
## 14           16          1
## 15           18          2
## 16           20          2
## 17            1          0
```

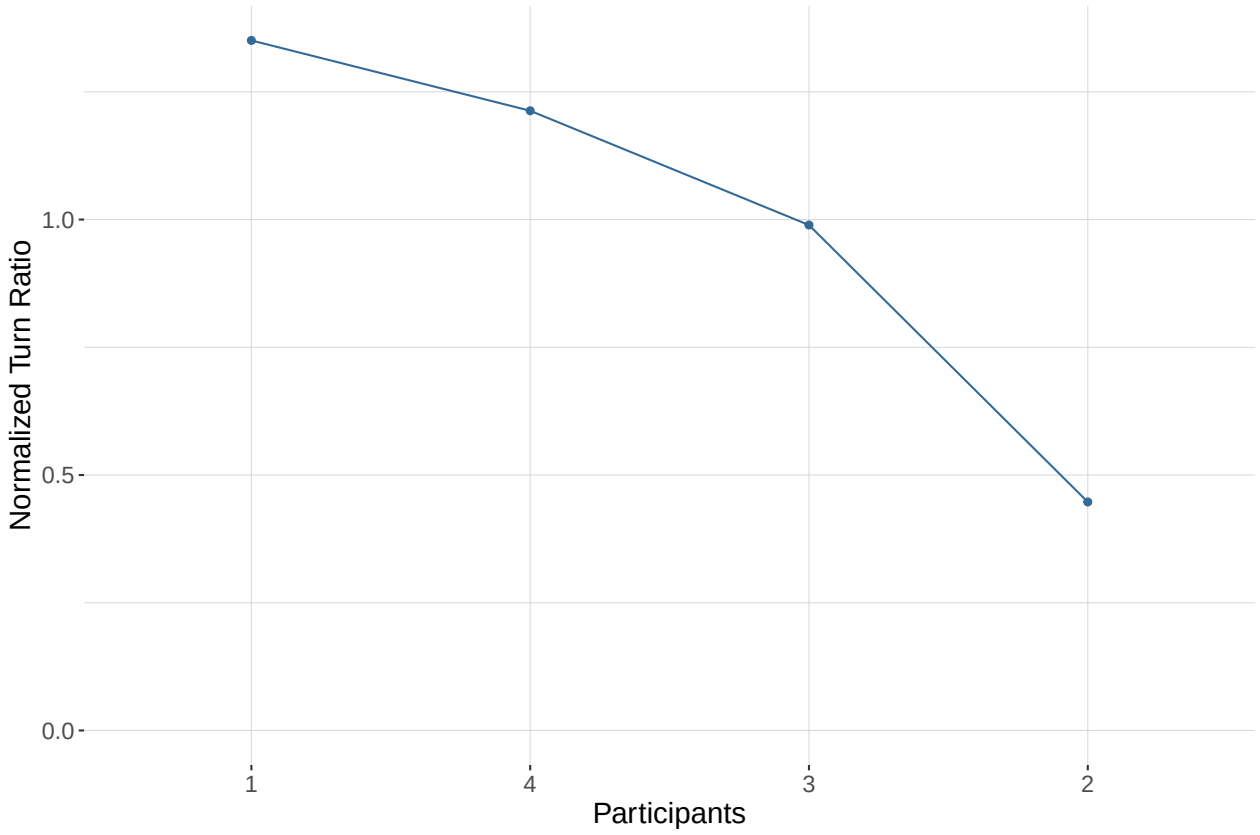
```
## 18           13          0
## 19           17          0
## 20           19          0
##
## $ngt_adv_stats
##   participant normalized_turn_ratio indv_SDI_arg      SDI
## 1           1             1.3505376   -0.3666006 1.318946
## 2           2             0.4473118   -0.2449920 1.318946
## 3           3             0.9892473   -0.3455207 1.318946
## 4           4             1.2129032   -0.3618325 1.318946
##         SEI
## 1 0.9514183
## 2 0.9514183
## 3 0.9514183
## 4 0.9514183
##
## $episodes_plot
```




```
##
## $qvr_plot
```


Episode Continuations per hour
75
50
25
0
0
5
10
15
Episode Starts per hour
1
2
3
4

```
##
## $ntr_plot
```




```
##
## $saveDataVar
## [1] 3
```

2.9 Exporting to Disk

The `writeData()` function accepts specific **discourseGT** function output and exports it as a permanent file to a specified directory on the user's disk. It can save the generated summary object, any plots, and weighted edge lists. Images will automatically export as a `.tiff` at 300 DPI, and console output will be exported as a `.txt` file.

The following example exports the generated summary to disk:

```
writeData("Sample Data 1", summaryData, dirpath = tempdir())
```

References

Albert Chai, Joshua P. Le, Andrew S. Lee, and Stanley M. Lo. Applying graph theory to examine the dynamics of student discussions in small-group learning. *CBE - Life Sciences Education*, 18, 2019. doi: 10.1187/cbe.18-11-0222.

T. M. J. Fruchterman and E. M. Reingold. Graph drawing by force-directed placement. *Software - Practice and Experience*, 21:1129–1164, 1991. doi: 10.1002/spe.4380211102.

M. Girvan and M. E. J. Newman. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences*, 99:7821–7826, 2002. doi: 10.1073/pnas.122653799.

Tomihisa Kamada and Satoru Kawai. An algorithm for drawing general undirected graphs. *Information Processing Letters*, 31(1):7–15, 1989. ISSN 0020-0190. doi: https://doi.org/10.1016/0020-0190(89)90102-6. URL http://www.sciencedirect.com/science/article/pii/0020019089901026.

Dilhara Liyanage, Stanley M. Lo, and Sally S. Hunnicutt. Student discourse networks and instructor facilitation in process oriented guided inquiry physical chemistry classes. *Chem. Educ. Res. Pract.*, 22:93–104, 2021. doi: 10.1039/D0RP00031K. URL http://dx.doi.org/10.1039/D0RP00031K.

Martin Morgan and Marcel Ramos. *BiocManager: Access the Bioconductor Project Package Repository*. Bioconductor, 2021. URL https://cran.r-project.org/web/packages/BiocManager/index.html. R package version 1.30.16.

Alex Pico, Tanja Muetze, Paul Shannon, Ruth Isserlin, Shraddha Pai, Julia Gustavsen, and Georgi Kolishovski. *Functions to Access and Control Cytoscape*. Bioconductor, 2021. URL https://bioconductor.org/packages/release/bioc/html/RCy3.html. R package version 2.12.4.

Edward M. Reingold and John S. Tilford. Tidier drawings of trees. *IEEE Transactions on Software Engineering*, 7(2), 1981.

Barret Schloerke, Jason Crowley, Di Cook, Francois Briatte, Moritz Marbach, Edwin Thoen, Amos Elberg, and Joseph Larmarange. *GGally: Extension to 'ggplot2'*, 2021. URL https://CRAN.R-project.org/package=GGally. R package version 2.1.2.

Paul Shannon, Andrew Markiel, Owen Ozier, Nitin S. Baliga, Jonathan T. Wang, Daniel Ramage, Nada Amin, Benno Schwikowski, and Trey Ideker. Cytoscape: a software environment for integrated models of biomolecular interaction networks. *Genome Research*, 13:2498–2504, 2003. URL https://www.cytoscape.org/.

Hadley Wickham, Winston Chang, Lionel Henry, Thomas Lin Pedersen, Kohske Takahashi, Claus Wilke, and Kara Woo. *ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics*, 2021. URL https://CRAN.R-project.org/package=ggplot2. R package version 3.3.5.