Supplementary Material of "Visualizing Uncertainty in Probabilistic Graphs with Network Hypothetical Outcome Plots (NetHOPs)"

NetHOPs Study Interface, Prototypes, and Analysis Demonstration

1. Study Interface

Our study interface is built in HTML, CSS, and JavaScript. The code is available at https://github.com/dpzhang/NetHOPs

1.1 Participants' Qualification Test

Participants' qualification test is based on a simple static network shown in Figure 1 below.

```r
# Construct network
set.seed(5)
n = 8
p = 1/4
g = sample_gnp(n = n, p = p, directed = T) %>%
  add_vertices(2)
g = g - edge("3|2")

# Community detection
set.seed(1)
plot(cluster_optimal(g), g, edge.color = "black")
```

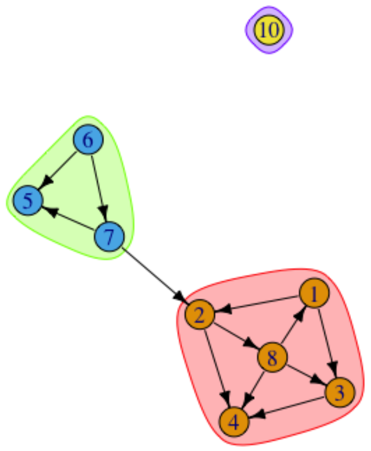


Figure 1: The network used for the participant qualification test.

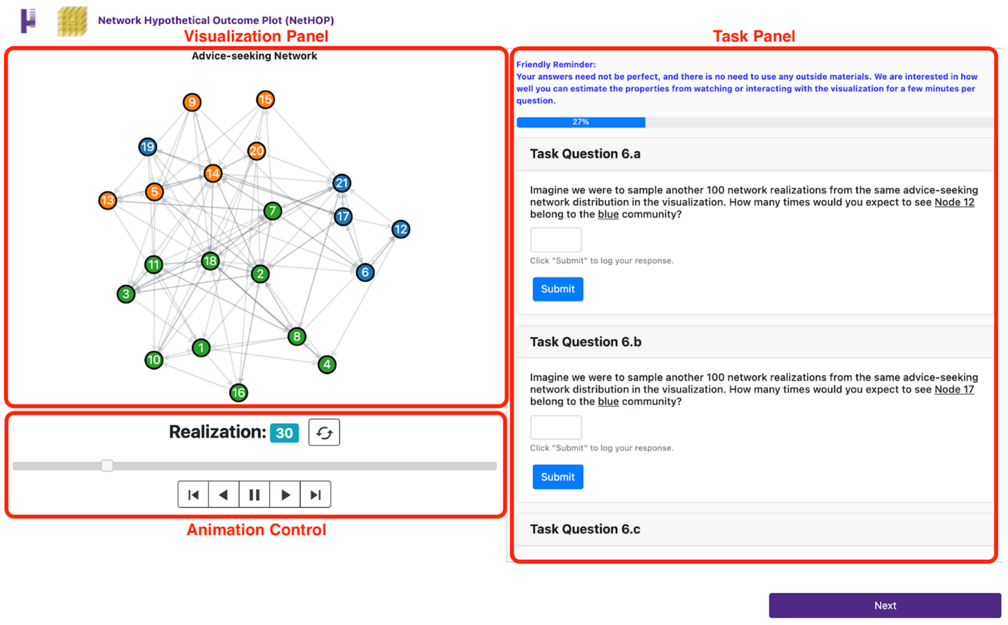
We asked a total of 7 questions:

- Question 1: Nodes How many nodes are there in the network?

- Question 2: Links How many links are there in the network?
- Question 3: Directionality Is it a directed or undirected network?
- Question 4: Isolates How many isolates are there in the network? By isolate, we mean a node that is not connected with any other node.
- Question 5: Communities How many communities are there in the network? By community, we mean a collection or cluster of, at least, 2 nodes.
- Question 6: Shortest Path How many links are there on the shortest path from node 6 to node 1?
- Question 7: On a scale from 0 to 1, what is the density of this network? Please round off your responses to 2 decimal places. (Tip: Please use your responses from Question 1 and Question 2 to compute density)
  - 0 means an empty graph with no link.
  - 1 means a fully connected graph.

1.2 Interface without Tuning

Figure 2 provides an example screen for our interface for the 11 task pages **without tuning**. Participants see a Visualization Panel on the left and a Task Panel on the right. The visualization consists of realizations sampled using our network model.



Friendly Reminder:
Your answers need not be perfect, and there is no need to use any outside materials. We are interested in how well you can estimate the properties from watching or interacting with the visualization for a few minutes per question.

Task Question 6.a

Imagine we were to sample another 100 network realizations from the same advice-seeking network distribution in the visualization. How many times would you expect to see Node 12 belong to the blue community?

Click "Submit" to log your response.

Task Question 6.b

Imagine we were to sample another 100 network realizations from the same advice-seeking network distribution in the visualization. How many times would you expect to see Node 17 belong to the blue community?

Click "Submit" to log your response.

Task Question 6.c

Figure 2: Study interface with default NetHOPs parameters.

Below the animated visualization is the Animation Control, shown in Figure 3. There are seven buttons, and participants were instructed to use the animation control as much as they want when answering the task questions.

Here is a brief description of the possible controls:

- Reset Button: Click to reset the animation back to simulation 1.
- Draggable Slider: drag the marker to specific realization.
- Previous Frame: When paused, click to go to the previous realization.
- Backward Play: Click to play the animation backward.
- Pause: Click to pause the animation.

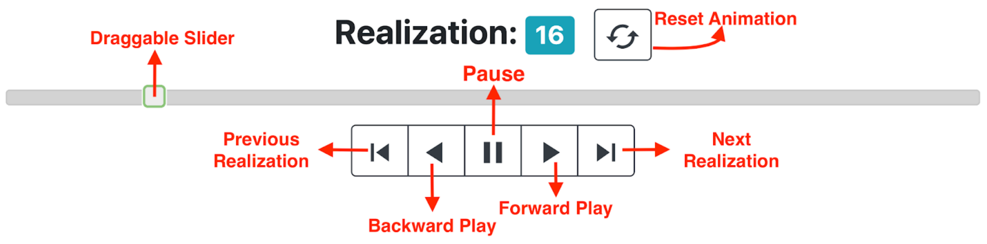
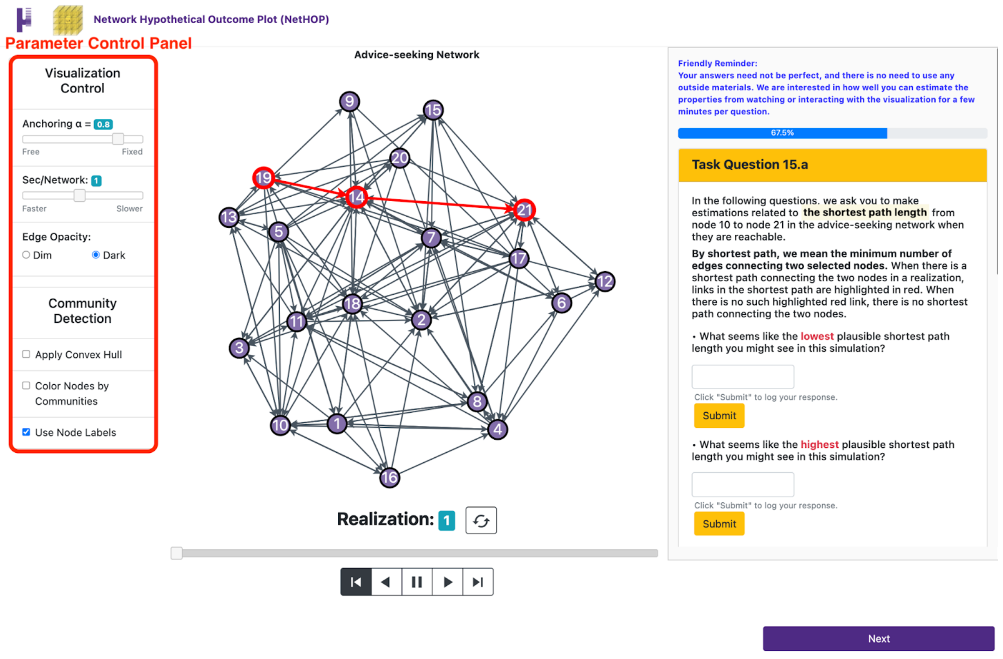


Figure 3: Animation controls used to manipulate NetHOPs rendering.

- Forward Play: Click to play the animation forward.
- Next Frame: When paused, click to go to the subsequent realization.

1.3 Interface with Tuning

We show an example of our interface of the 11 task pages **with tuning** in Figure 4. Participants were instructed to change their responses if they felt confident that tuning the visualization parameters could improve their previous responses.



Edge Opacity: ○ Dim [● Dark]

☐ Apply Convex Hull

☐ Color Nodes by Communities

☑ Use Node Labels

Friendly Reminder:
Your answers need not be perfect, and there is no need to use any outside materials. We are interested in how well you can estimate the properties from watching or interacting with the visualization for a few minutes per question.

Task Question 15.a

In the following questions. we ask vou to make estimations related to **the shortest path length** from node 10 to node 21 in the advice-seeking network when they are reachable.

By shortest path, we mean the minimum number of edges connecting two selected nodes. When there is a shortest path connecting the two nodes in a realization, links in the shortest path are highlighted in red. When there is no such highlighted red link, there is no shortest path connecting the two nodes.

- What seems like the lowest plausible shortest path length you might see in this simulation?

Click "Submit" to log your response.

- What seems like the highest plausible shortest path length you might see in this simulation?

Click "Submit" to log your response.

Figure 4: Study interface that allows participants to customize NetHOPs rendering.

2. Prototype Construction

This section of the supplementary material demonstrates how we constructed our NetHOPs prototype using Krackhardt's CSS dataset.

We load Krackhardt's CSS datasets and import the necessary helper functions required to construct the prototypes.

```r
load('./data/krackhardtCSS/krackhardt_css_data.RData')
source('./scripts/helper-nodeColoring.R')
source('./scripts/helper-dynamicLayout.R')
source('./scripts/helper-miscProto.R')
```

NetHOPs construction pipeline is demonstrated below.

```r
# For the advice-seeking CSS
adviceNets150 = draw_nNets(advice_nets, 150, seed = 1) %>% # draw networks
  # Cluster detection on each sampled network realizations.
  detect_communities(igraph::cluster_optimal, seed = 1) %>%
  # Send the sampled network realizations to our instant-optimal
    # community matching and coloring algorithm
  create_nodeColorAttribute %>%
  # Apply the layout aggregation and anchoring algorithm
  add_anchoringCoordinate(weights = NA,
                          alpha = 0, iter = 500, tol = 1e-04, seed = 1) %>%
  # Normalize the layout coordinate
  normalize_layoutCoordinate

# The same process implemented on the friendship CSS
friendNets150 = draw_nNets(friendship_nets, 150, seed = 1) %>%
  detect_communities(igraph::cluster_optimal, seed = 1) %>%
  create_nodeColorAttribute %>%
  add_anchoringCoordinate(weights = NA,
                          alpha = 0, iter = 500, tol = 1e-04, seed = 1) %>%
  normalize_layoutCoordinate
```

The processed data used to construct the NetHOPs prototypes can be found in the `data` directory.

3. Analysis Demonstration

In this section of the supplementary material, we reproduce our analysis by first importing processed user response datasets as shown in the code block below. Due to IRB, we cannot release our full dataset. Among the data frames imported:

3.1 Data Preparation

- `userResponseDF`: is the data frame that contains participants' responses submitted (1) using the default visualization parameters and graphical elements (Table 1 in the paper). (2) given the freedom to tune the visualization parameters and graphical elements.
- `visParamDF`: is the data frame that contains the visualization parameters and graphical elements used in the second iteration tuned by participants.
- `scoreDF`: is the data frame that contains the performance evaluation metrics, that are (1) EMD scores and (2) probability estimation.
- `truthDF`: is the data frame that contains the ground truth network property or statistics for each network realization sampled from the advice-seeking and friendship CSS data.

```r
userResponseDF = read.csv("./data/userResponse.csv")
visParamDF = read.csv("./data/visParam.csv")
scoreDF = read.csv("./data/score.csv")
truthDF = read.csv("./data/groundTruth/truth150.csv") %>%
  mutate(AD = AD * 100, FD = FD * 100) # convert densities to percentage

# Total number of participants
numParticipants = nrow(userResponseDF)
```

```r
# Subset scoreDF by tuning and noTuning
part1DF = scoreDF %>% select(contains("_noTune"))
part2DF = scoreDF %>% select(contains("_tune"))
```

We first present participants' demographics, which was described in Section 4.5 and 5.1 of the paper.

```r
# study the demographics of users
numUsers = userResponseDF %>% nrow
userInfoDF = userResponseDF %>%
  select(starts_with("demographics"))

# Summary statistics of user demographics
userInfoDF %>%
  select(demographics_education,
         demographics_class,
         demographics_networkUse) %>%
  lapply(table) %>%
  lapply(function(x) x / numUsers)
```

```
## $demographics_education
##
##      Graduate Undergraduate
##    0.98039216    0.01960784
##
## $demographics_class
##
##          0          1          2         3+
## 0.09803922 0.17647059 0.37254902 0.35294118
##
## $demographics_networkUse
##
##     always      never      often     rarely  sometimes
## 0.09803922 0.13725490 0.21568627 0.23529412 0.31372549
```

```r
# Calculate user study time completion without removing the outlier
studyTimeVec = userResponseDF %>%
  select(ends_with("timeSpent")) %>%
  apply(1, sum) %>%
  `/`(60) %>%
  round(0)
summary(studyTimeVec)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   14.00   42.50   60.00   73.94   74.00  560.00
```

```r
sd(studyTimeVec)
```

```
## [1] 76.98478
```

3.2 Performance Overview

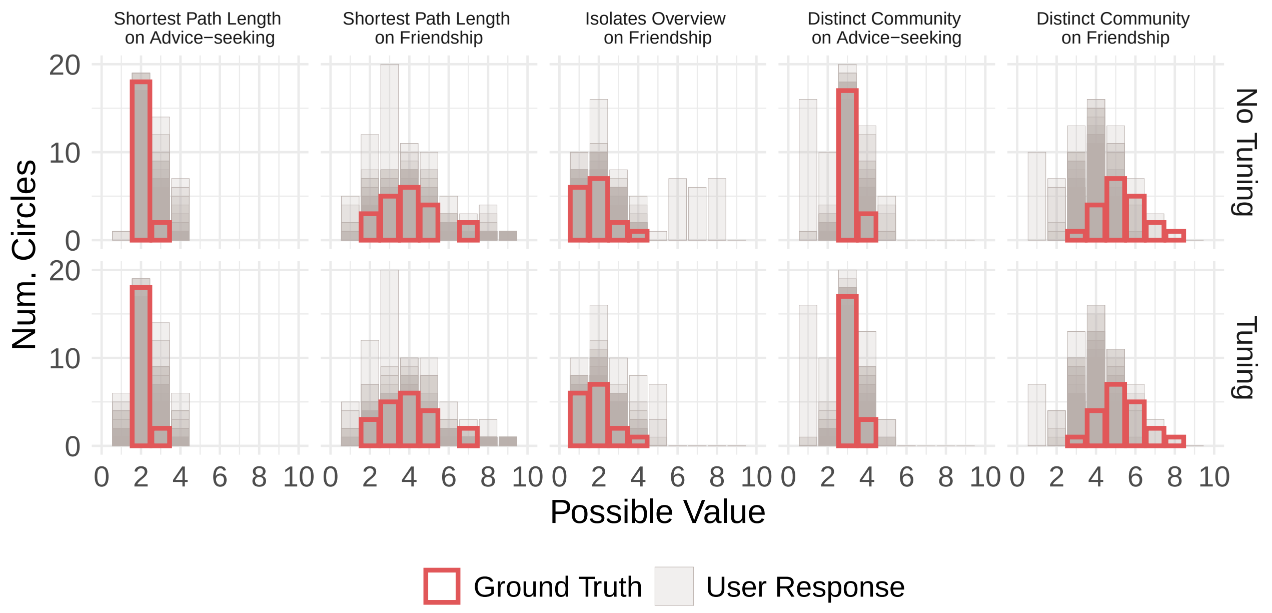
We reproduce Figure 5 of the paper, in which we provide an overview of participants' task performance. We first load the helper functions from the `scripts` directory.

```r
source("./scripts/helper-miscAnalysis.R")
source("./scripts/helper-plot-allUserResponses.R")
```

Participants' responses are superimposed against the ground truth distribution or probability below.

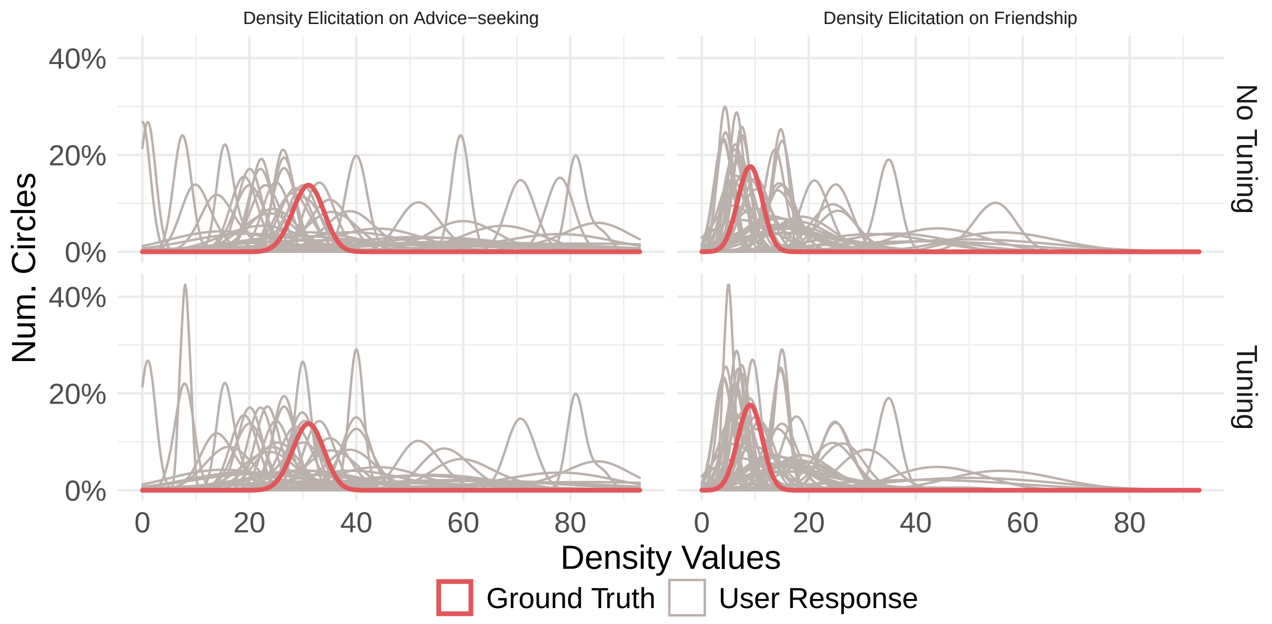
- Task with discrete responses.

```
discretePlot_bar +
  theme(legend.position = "bottom",
        strip.text.x = element_text(size = 8))
```



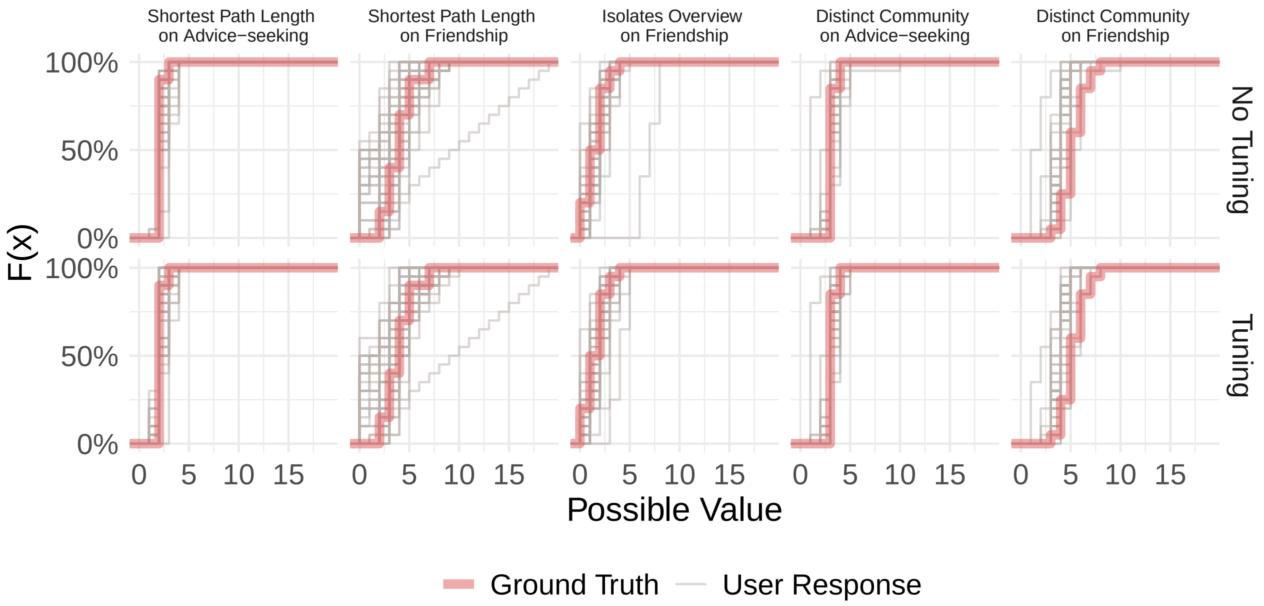
- Task with continuous responses.

```
continuousPlot_density +
  theme(legend.position = "bottom",
        strip.text.x = element_text(size = 8))
```

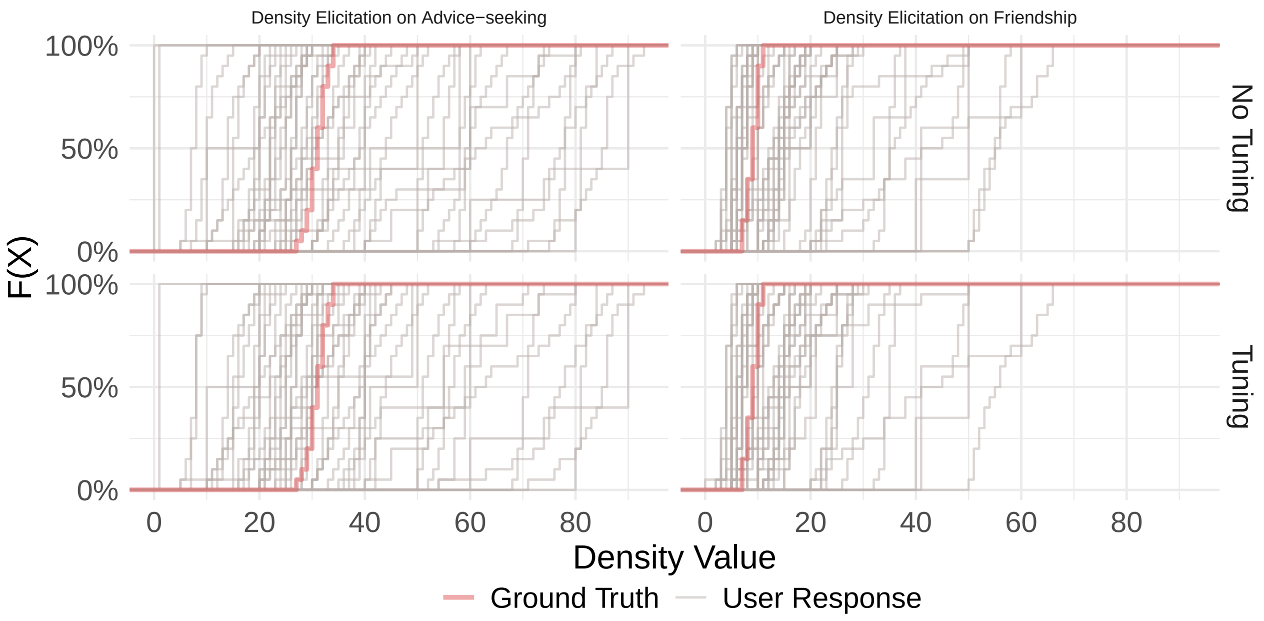


- Empirical cumulative distributions functions for all distribution elicitation tasks. Barplots and density curves differentiate the discrete and continuous responses.

```
discretePlot_cdf +
  theme(legend.position = "bottom",
        strip.text.x = element_text(size = 8))
```

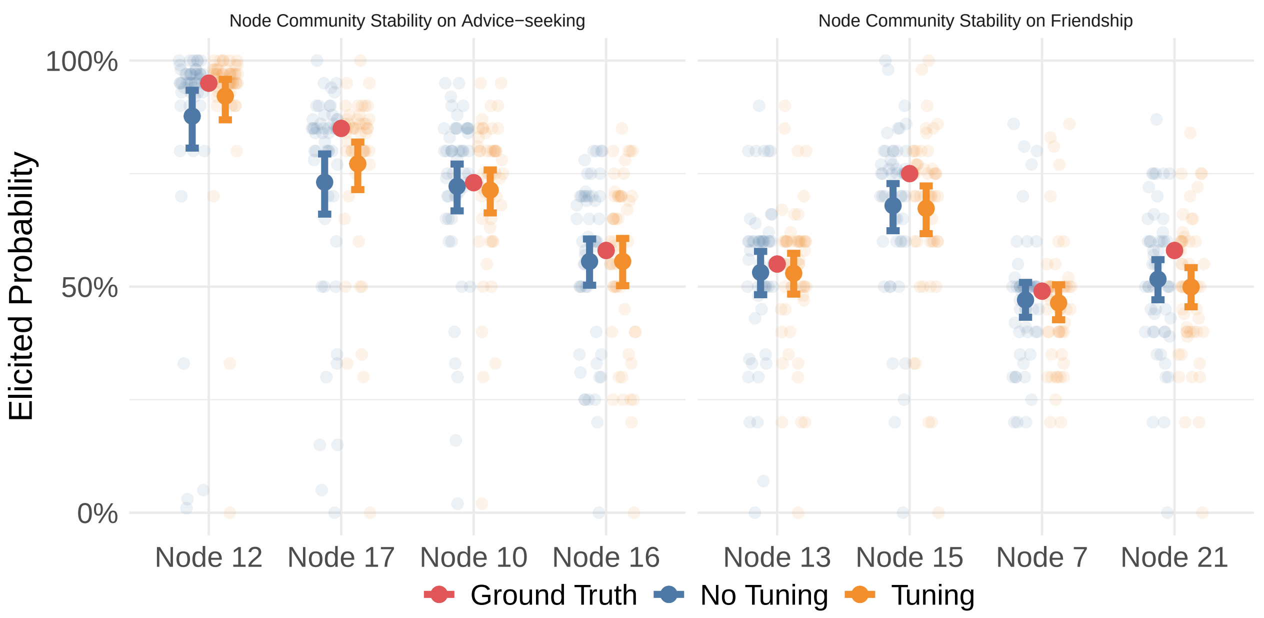


```
continuousPlot_cdf +
  theme(legend.position = "bottom",
        strip.text.x = element_text(size = 8))
```

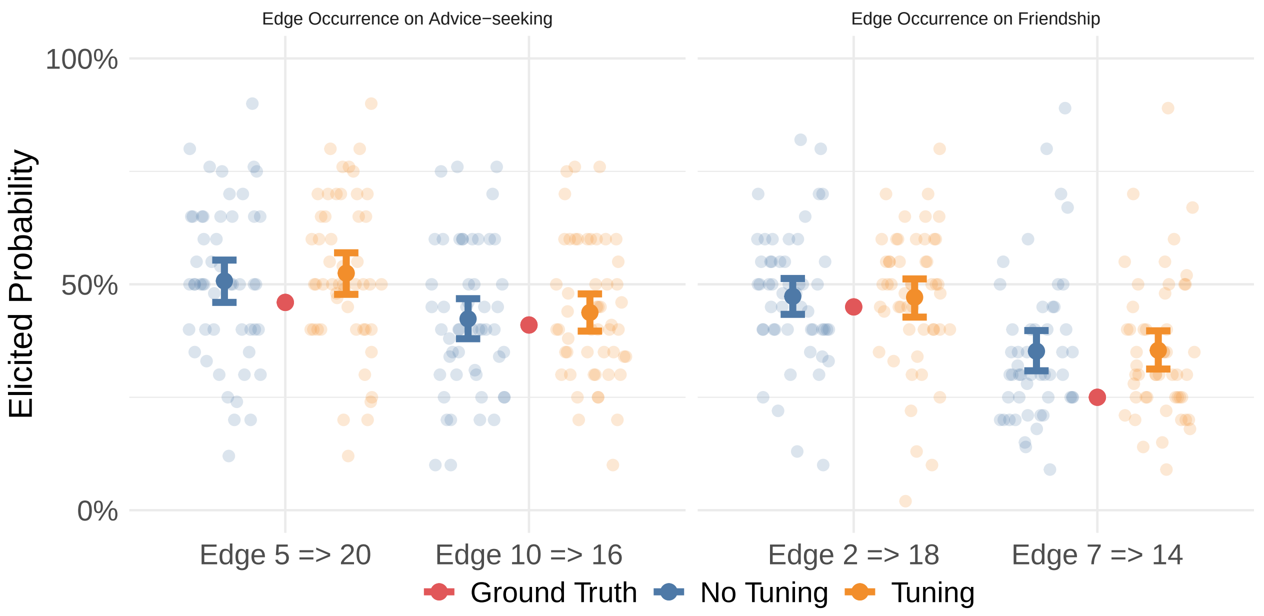


- Strip plots present participants' deterministic probability estimates with 95% bootstrapped CIs for each attribute-based task from the study. The ground truth probabilities for each task are shown as red points between the CIs. We elicited four probability estimates for node-attribute tasks and two for edge-attribute tasks from advice-seeking and friendship networks.

```
nodeStabilityPlot +
  theme(legend.position = "bottom",
        strip.text.x = element_text(size = 8))
```



```r
edgeOccurrencePlot +
  theme(legend.position = "bottom",
        strip.text.x = element_text(size = 8))
```



As mentioned in Section 5.1, we discovered a data anomaly for the browsing task of the friendship network. About half of participants (22) incorrectly assumed some realizations have the shortest path length of zero (Figure 6 A, left, the second column), when theoretically it should be infinity and therefore not counted when sketching the distributions. This conceptual misunderstanding led to more divergence (i.e., higher EMD scores) in response quality for this task.

```r
# Subset the browsing task of the friendship network
taskDF = userResponseDF %>%
  select(contains(paste0("FSP", "_")) &
           contains(paste0("_", "tune")) &
           matches("distLabels|distResponse"))
head(taskDF, 5)
```

```
##   FSP_tune_distLabels FSP_tune_distResponse
## 1   0,1,2,3,4,5,6,7,8     6,0,1,4,4,3,1,1,0
## 2 0,1,2,3,4,5,6,7,8,9   8,0,1,1,4,2,1,1,1,1
## 3   0,1,2,3,4,5,6,7,8     9,0,2,1,4,1,1,1,1
## 4 0,1,2,3,4,5,6,7,8,9   6,0,1,3,3,2,2,1,1,1
## 5             2,3,4,5               2,8,8,2
```

```
# Loop through all 51 user responses to check number of bals in the "0" bucket
obs_lst = list()
for(i in 1:nrow(taskDF)){
  labels = taskDF %>% select(contains("distLabels")) %>%
    slice(i) %>% pull %>%
    as.character %>% strsplit(",") %>% unlist %>% as.numeric
  responses = taskDF %>% select(contains("distResponse")) %>%
    slice(i) %>% pull %>%
    as.character %>% strsplit(",") %>% unlist %>% as.numeric
  obs = rep(labels, times = responses)
  obs_lst %<>% list.append(obs)
}
# Check how many participants committed the error
obs_lst %>%
  lapply(function(x) 0 %in% x) %>%
  unlist %>% mean %>% `*`(numParticipants)
```

```
## [1] 22
```

- We present the bootstrapped performance metrics (EMD and probability estimation error) for each task below. To do so, we first load the helper functions.

```
source("./scripts/helper-performanceAnalysis.R")
```

3.3 Performance Evaluation

Below are the 95% CIs for tasks evaluated by EMD. These tasks include (1) Topology (the number of distinct communities), (2) Overview (isolates and density), (3) Browsing (shortest path length).

```
CIDF %>%
  mutate(taxonomy = case_when(str_detect(task, "Community") ~ "Topology",
                              str_detect(task, "Density") ~ "Overview-Density",
                              str_detect(task, "Isolate") ~ "Overview-Isolate",
                              str_detect(task, "Path") ~ "Browsing")) %>%
  filter(question == "EMD") %>%
  select(taxonomy, data, tuneFlag, lower_ci, mu_hat, upper_ci) %>%
  knitr::kable(.)
```

taxonomy	data	tuneFlag	lower_ci	mu_hat	upper_ci
Topology	Advice-seeking	No Tuning	1.4747264	1.809804	2.144881
Topology	Advice-seeking	Tuning	1.5067171	1.850000	2.193283
Topology	Friendship	No Tuning	2.7452983	3.050980	3.356663
Topology	Friendship	Tuning	2.5455102	2.850000	3.154490
Overview-Isolate	Friendship	No Tuning	0.7592818	1.050000	1.340718

taxonomy	data	tuneFlag	lower_ci	mu_hat	upper_ci
Overview-Isolate	Friendship	Tuning	0.7828337	1.021569	1.260304
Browsing	Advice-seeking	No Tuning	1.6832313	2.087255	2.491278
Browsing	Advice-seeking	Tuning	1.7035661	2.109804	2.516042
Browsing	Friendship	No Tuning	1.6638456	2.088235	2.512625
Browsing	Friendship	Tuning	1.5368061	1.965686	2.394566
Overview-Density	Advice-seeking	No Tuning	13.9810680	18.060784	22.140501
Overview-Density	Advice-seeking	Tuning	12.2857893	16.080392	19.874995
Overview-Density	Friendship	No Tuning	7.3216690	10.604902	13.888135
Overview-Density	Friendship	Tuning	6.6775630	9.583333	12.489104

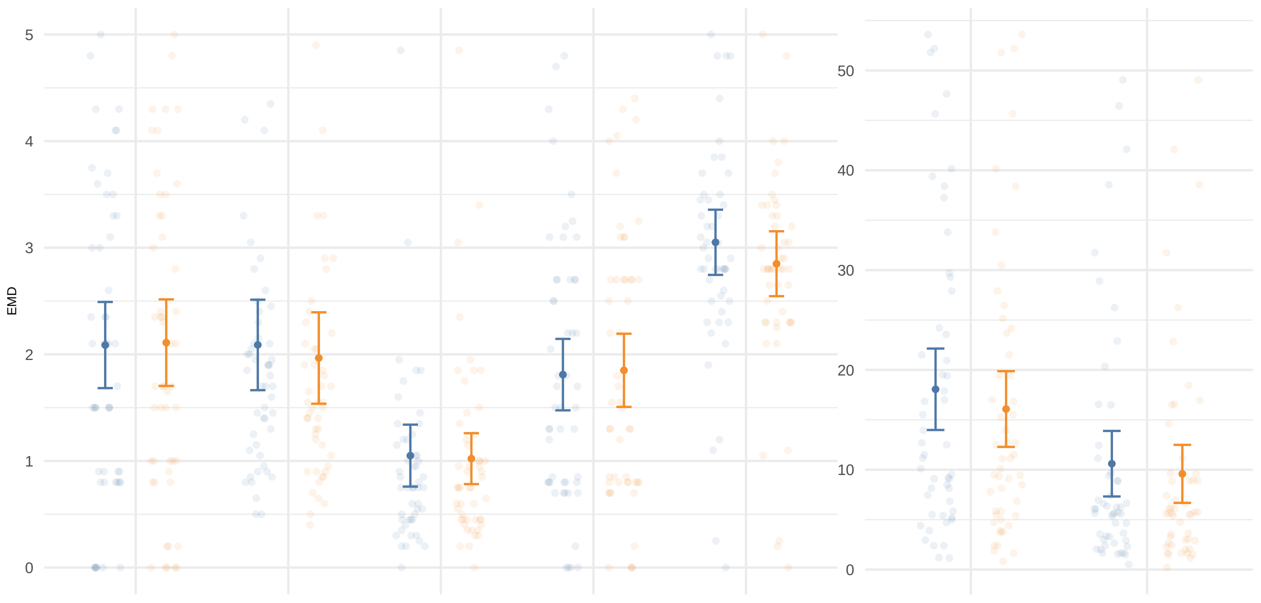
Below are the 95% CIs for tasks evaluated by probability estimation error, which include node-attribute (distinct community) and edge attribute (edge occurrence) tasks.

```
CIDF %>%
  mutate(taxonomy = case_when(str_detect(task, "Stability") ~ "Node",
                              str_detect(task, "Occurrence") ~ "Edge")) %>%
  filter(question != "EMD") %>%
  select(taxonomy, data, question, tuneFlag, lower_ci, mu_hat, upper_ci) %>%
  knitr::kable(.)
```

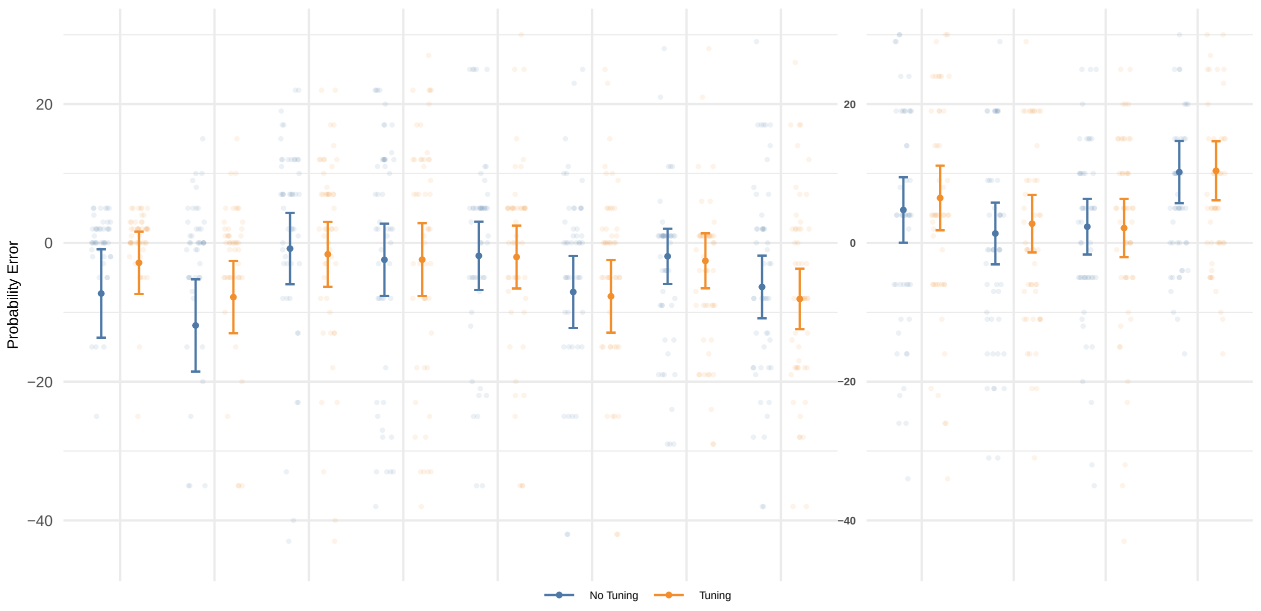
taxonomy	data	question	tuneFlag	lower_ci	mu_hat	upper_ci
Node	Advice-seeking	Node 12	No Tuning	-13.661799	-7.2941176	-0.9264361
Node	Advice-seeking	Node 12	Tuning	-7.351102	-2.8627451	1.6256114
Node	Advice-seeking	Node 17	No Tuning	-18.547639	-11.9019608	-5.2562825
Node	Advice-seeking	Node 17	Tuning	-13.031496	-7.8235294	-2.6155625
Node	Advice-seeking	Node 10	No Tuning	-5.966028	-0.8235294	4.3189695
Node	Advice-seeking	Node 10	Tuning	-6.324605	-1.6470588	3.0304876
Node	Advice-seeking	Node 16	No Tuning	-7.637672	-2.4313725	2.7749267
Node	Advice-seeking	Node 16	Tuning	-7.658373	-2.4117647	2.8348436
Node	Friendship	Node 13	No Tuning	-6.782300	-1.8627451	3.0568095
Node	Friendship	Node 13	Tuning	-6.571456	-2.0392157	2.4930249
Node	Friendship	Node 15	No Tuning	-12.268571	-7.0784314	-1.8882914
Node	Friendship	Node 15	Tuning	-12.932058	-7.7058824	-2.4797072
Node	Friendship	Node 7	No Tuning	-5.923509	-1.9411765	2.0411565
Node	Friendship	Node 7	Tuning	-6.552626	-2.5882353	1.3761552
Node	Friendship	Node 21	No Tuning	-10.873455	-6.3529412	-1.8324277
Node	Friendship	Node 21	Tuning	-12.442700	-8.0784314	-3.7141624
Edge	Advice-seeking	Edge 5 => 20	No Tuning	0.029712	4.7450980	9.4604841
Edge	Advice-seeking	Edge 5 => 20	Tuning	1.810551	6.4705882	11.1306254
Edge	Advice-seeking	Edge 10 => 16	No Tuning	-3.100137	1.3529412	5.8060192
Edge	Advice-seeking	Edge 10 => 16	Tuning	-1.378358	2.7647059	6.9077701
Edge	Friendship	Edge 2 => 18	No Tuning	-1.674296	2.3333333	6.3409626
Edge	Friendship	Edge 2 => 18	Tuning	-2.064228	2.1372549	6.3387375
Edge	Friendship	Edge 7 => 14	No Tuning	5.715437	10.1960784	14.6767201
Edge	Friendship	Edge 7 => 14	Tuning	6.136300	10.3921569	14.6480134

To visualize all CIs, we reproduce Figure 6 of the paper below.

```
allEMD
```



```
allDeterministic
```



We summed each participants' EMD scores for all tasks completed with and without tuning and computed the average amount of improvement. We found that the ability to control NetHOPs rendering could improve participants' distribution elicitation by 4% on average (Note: negative values indicate decreased EMD score, which means performance improvement).

```
# Does tuning improve participants' distribution elicitation on average?
noTune_EMD = part1DF %>%
  select(!contains(c("AUN", "FUN", "AEO", "FEO"))) %>% abs %>% apply(1, sum)
tune_EMD = part2DF %>%
  select(!contains(c("AUN", "FUN", "AEO", "FEO"))) %>% abs %>% apply(1, sum)
EMD_diff = data.frame(diffPercent = (tune_EMD - noTune_EMD) / noTune_EMD)
mu_hat_EMD_diff = EMD_diff %>% pull(diffPercent) %>% mean
set.seed(2333)
```

```r
EMD_diff %>%
  specify(response = diffPercent) %>%
  generate(reps = 10000, type = "bootstrap") %>%
  calculate(stat = "mean") %>%
  get_confidence_interval(type = "se", point_estimate = mu_hat_EMD_diff) %>%
  mutate(mu_hat = mu_hat_EMD_diff) %>%
  select(lower_ci, mu_hat, upper_ci)
```

```
## # A tibble: 1 x 3
##   lower_ci  mu_hat upper_ci
##      <dbl>   <dbl>    <dbl>
## 1   -0.120 -0.0392   0.0413
```

We computed a similar statistic by aggregating the total absolute error for probability estimation tasks, and found tuning visualization parameters did not improve probability estimation.

```r
# Does tuning improve participants' deterministic guess on average?
noTune_guess = part1DF %>%
  select(contains(c("AUN", "FUN", "AEO", "FEO"))) %>% abs %>% apply(1, sum)
tune_guess = part2DF %>%
  select(contains(c("AUN", "FUN", "AEO", "FEO"))) %>% abs %>% apply(1, sum)
guess_diff = data.frame(
  diffPercent = (tune_guess - noTune_guess) / noTune_guess)
mu_hat_guessDiff = guess_diff %>% pull(diffPercent) %>% mean
set.seed(2333)
guess_diff %>%
  specify(response = diffPercent) %>%
  generate(reps = 10000, type = "bootstrap") %>%
  calculate(stat = "mean") %>%
  get_confidence_interval(type = "se", point_estimate = mu_hat_guessDiff) %>%
  mutate(mu_hat = mu_hat_guessDiff) %>%
  select(lower_ci, mu_hat, upper_ci)
```

```
## # A tibble: 1 x 3
##   lower_ci  mu_hat upper_ci
##      <dbl>   <dbl>    <dbl>
## 1   -0.133 0.00454    0.142
```

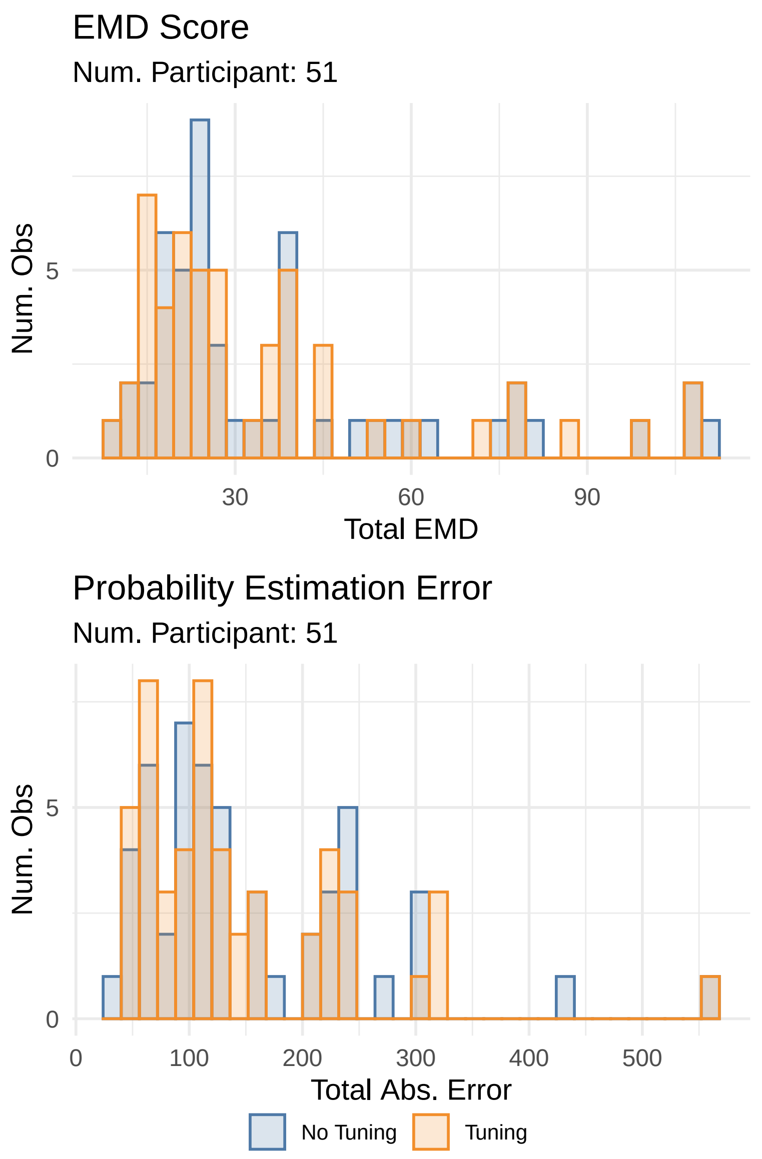
We found 22% (11) of our participants provided the same answers in the second iteration of tasks.

```r
source("./scripts/helper-checkSameResponse.R")
(noTuneResponseDF == tuneResponseDF) %>%
  apply(1, sum) %>%
  `==`(ncol(noTuneResponseDF)) %>% sum
```

```
## [1] 11
```

Since we observe tuning visualization parameters only slightly helped participants' EMD scores, a natural follow-up question is: what are the distributions of EMD score and probability estimation error look like?

```r
source("./scripts/helper-distPerformancePlot.R")
ggarrange(EMDScoreDistPlot, probabilityErrorDistPlot,
          nrow = 2, common.legend = T, legend = "bottom")
```



The distributions of the two performance metrics indicate that tuning did help some participants. Below, we present the summed performance metrics including EMD (top) and probability estimation error (bottom) with and without tuning. To investigate the dynamics between visualization parameters and accuracy, we ranked all participants by their performance with tuning. From the rank, we grouped top-performers from the first quartile and bottom-performers from the fourth quartile, with each group consisting of 13 participants.

```r
# The within variance of vis parameters by subtask is minimal within each task
# To test it, please run the following code
#source("./scripts/helper-paramVariance.R")
#compute_allParam_variance_bySubtasks(visParamDF)

# For distribution sketching tasks, we select the visualization parameters used
  # for the distribution builder
# For probability estimation tasks, we randomly select a set of the
  # visualization parameters for the first task
source("./scripts/helper-reduceVisParamDF.R")
sVisParamDF = reduce_visParamDF(visParamDF)
```

```r
# Rank all participants by score
rankingDF = data.frame(user = userResponseDF$userID,
                       email = userResponseDF$demographics_email,
                       totalScore = scoreDF %>% abs %>% apply(1, sum),
                       part1Score = part1DF %>% abs %>% apply(1, sum),
                       part2Score = part2DF %>% abs %>% apply(1, sum)) %>%
  arrange(part2Score)

# Group the total scores thats in the 1st quartile and 3rd quartile
firstQuartile = quantile(rankingDF$part2Score, 0.25)
thirdQuartile = quantile(rankingDF$part2Score, 0.75)
# Find the participants corresponding tot he score
user1Q = rankingDF %>%
  filter(part2Score <= firstQuartile) %>%
  pull(user) %>%
  as.character
user3Q = rankingDF %>%
  filter(part2Score >= thirdQuartile) %>%
  pull(user) %>%
  as.character

# Create dataframes for each group of participants
  # vis parameters used by top-performers
topVisParamDF = sVisParamDF[userResponseDF$user %in% user1Q,]
  # scores of the top-performers
topScoreDF = scoreDF[userResponseDF$user %in% user1Q,]
  # vis parameters used by bottom-performers
bottomVisParamDF = sVisParamDF[userResponseDF$user %in% user3Q,]
  # scores of the bottom-performers
bottomScoreDF = scoreDF[userResponseDF$user %in% user3Q,]
```

Below are the EMD score summaries of the top-performers and bottom-performers.

```r
# Top-performer EMD score summary
topUser_avgScore = topScoreDF %>%
  select(matches("AC|FC|I|ASP|FSP|AD|FD") & matches("_tune")) %>%
  apply(1, mean)
present_summaryStats(topUser_avgScore) %>%
  knitr::kable(.)
```

Min.	Median	Mean	sd	Max.
1.735714	2.835714	3.091209	1.175747	5.328571

```r
# Bottom-performer EMD score summary
bottomUser_avgScore = bottomScoreDF %>%
  select(matches("AC|FC|I|ASP|FSP|AD|FD") & matches("_tune")) %>%
  apply(1, mean)
present_summaryStats(bottomUser_avgScore) %>%
  knitr::kable(.)
```

Min.	Median	Mean	sd	Max.
3.114286	6.3	7.800549	4.658958	15.42143

Below are the summaries of performance for probability estimation tasks by top-performers and bottom-performers.

```r
# Top-performer probability estimates
topUser_avgGuess = topScoreDF %>%
  select(matches("AUN|FUN|AEO|FEO") & matches("_tune")) %>%
  apply(1, mean) %>% abs
present_summaryStats(topUser_avgGuess) %>%
  knitr::kable(.)
```

Min.	Median	Mean	sd	Max.
0.0833333	1.25	1.923077	1.533372	4.916667

```r
# Bottom-performer probability estimates
bottomUser_avgGuess = bottomScoreDF %>%
  select(matches("AUN|FUN|AEO|FEO") & matches("_tune")) %>%
  apply(1, mean) %>% abs
present_summaryStats(bottomUser_avgGuess) %>%
  knitr::kable(.)
```

Min.	Median	Mean	sd	Max.
1.333333	12.58333	14.32692	12.09819	44.83333

3.4 Visualization Parameters and Graphical Visual Aids

We focused on how anchoring and animation speed were used by each group and present the bootstrapped mean with 95% CIs to assess how the parameters seemed to relate to task performance.

```r
source("./scripts/helper-paramCI.R")
```

- 95% CI for the anchoring alpha parameter.

```r
# Alpha: Top vs. Bottom
compute_CI_subset(masterDF, parameter = "alpha", user = "Top")
```

```
## # A tibble: 1 x 3
##   lower_ci mu_hat upper_ci
##      <dbl>  <dbl>    <dbl>
## 1    0.694  0.788    0.881
```

```r
compute_CI_subset(masterDF, parameter = "alpha", user = "Bottom")
```

```
## # A tibble: 1 x 3
##   lower_ci mu_hat upper_ci
##      <dbl>  <dbl>    <dbl>
## 1    0.588  0.684    0.781
```

```r
# Alpha: Advice vs Friendship
compute_CI_subset(masterDF, parameter = "alpha", data = "Advice-seeking")
```

```
## # A tibble: 1 x 3
##   lower_ci mu_hat upper_ci
##      <dbl>  <dbl>    <dbl>
## 1    0.659   0.76     0.86
```

```
compute_CI_subset(masterDF, parameter = "alpha", data = "Friendship")
```

```
## # A tibble: 1 x 3
##   lower_ci mu_hat upper_ci
##      <dbl>  <dbl>    <dbl>
## 1    0.628  0.718    0.809
```

```
# Alpha: Top advice vs. Top friendship
compute_CI_subset(masterDF, parameter = "alpha", user = "Top", data = "Advice-seeking")
```

```
## # A tibble: 1 x 3
##   lower_ci mu_hat upper_ci
##      <dbl>  <dbl>    <dbl>
## 1    0.705    0.8    0.895
```

```
compute_CI_subset(masterDF, parameter = "alpha", user = "Top", data = "Friendship")
```

```
## # A tibble: 1 x 3
##   lower_ci mu_hat upper_ci
##      <dbl>  <dbl>    <dbl>
## 1    0.668  0.758    0.849
```

```
# Alpha: Bottom advice vs. Bottom friendship
compute_CI_subset(masterDF, parameter = "alpha", user = "Bottom", data = "Advice-seeking")
```

```
## # A tibble: 1 x 3
##   lower_ci mu_hat upper_ci
##      <dbl>  <dbl>    <dbl>
## 1    0.587  0.694    0.801
```

```
compute_CI_subset(masterDF, parameter = "alpha", user = "Bottom", data = "Friendship")
```

```
## # A tibble: 1 x 3
##   lower_ci mu_hat upper_ci
##      <dbl>  <dbl>    <dbl>
## 1    0.547  0.642    0.736
```

- 95% CI for the animation speed.

```
# Animation Speed: Top vs Bottom
compute_CI_subset(masterDF, parameter = "frameRate", user = "Top")
```

```
## # A tibble: 1 x 3
##   lower_ci mu_hat upper_ci
##      <dbl>  <dbl>    <dbl>
## 1    0.895   1.02     1.16
```

```
compute_CI_subset(masterDF, parameter = "frameRate", user = "Bottom")
```

```
## # A tibble: 1 x 3
##   lower_ci mu_hat upper_ci
##      <dbl>  <dbl>    <dbl>
## 1    0.726  0.815    0.905
```

```r
# Alpha: Advice vs Friendship
compute_CI_subset(masterDF, parameter = "frameRate", data = "Advice-seeking")
```

```
## # A tibble: 1 x 3
##   lower_ci mu_hat upper_ci
##      <dbl>  <dbl>    <dbl>
## 1    0.665  0.788    0.912
```

```r
compute_CI_subset(masterDF, parameter = "frameRate", data = "Friendship")
```

```
## # A tibble: 1 x 3
##   lower_ci mu_hat upper_ci
##      <dbl>  <dbl>    <dbl>
## 1    0.725  0.822    0.918
```

```r
# Alpha: Top advice vs. Top friendship
compute_CI_subset(masterDF, parameter = "frameRate", user = "Top", data = "Advice-seeking")
```

```
## # A tibble: 1 x 3
##   lower_ci mu_hat upper_ci
##      <dbl>  <dbl>    <dbl>
## 1    0.834  0.975     1.12
```

```r
compute_CI_subset(masterDF, parameter = "frameRate", user = "Top", data = "Friendship")
```

```
## # A tibble: 1 x 3
##   lower_ci mu_hat upper_ci
##      <dbl>  <dbl>    <dbl>
## 1    0.899   1.01     1.13
```

```r
# Alpha: Bottom advice vs. Bottom friendship
compute_CI_subset(masterDF, parameter = "frameRate", user = "Bottom", data = "Advice-seeking")
```

```
## # A tibble: 1 x 3
##   lower_ci mu_hat upper_ci
##      <dbl>  <dbl>    <dbl>
## 1    0.691  0.792    0.893
```

```r
compute_CI_subset(masterDF, parameter = "frameRate", user = "Bottom", data = "Friendship")
```

```
## # A tibble: 1 x 3
##   lower_ci mu_hat upper_ci
##      <dbl>  <dbl>    <dbl>
## 1     0.74  0.822    0.903
```

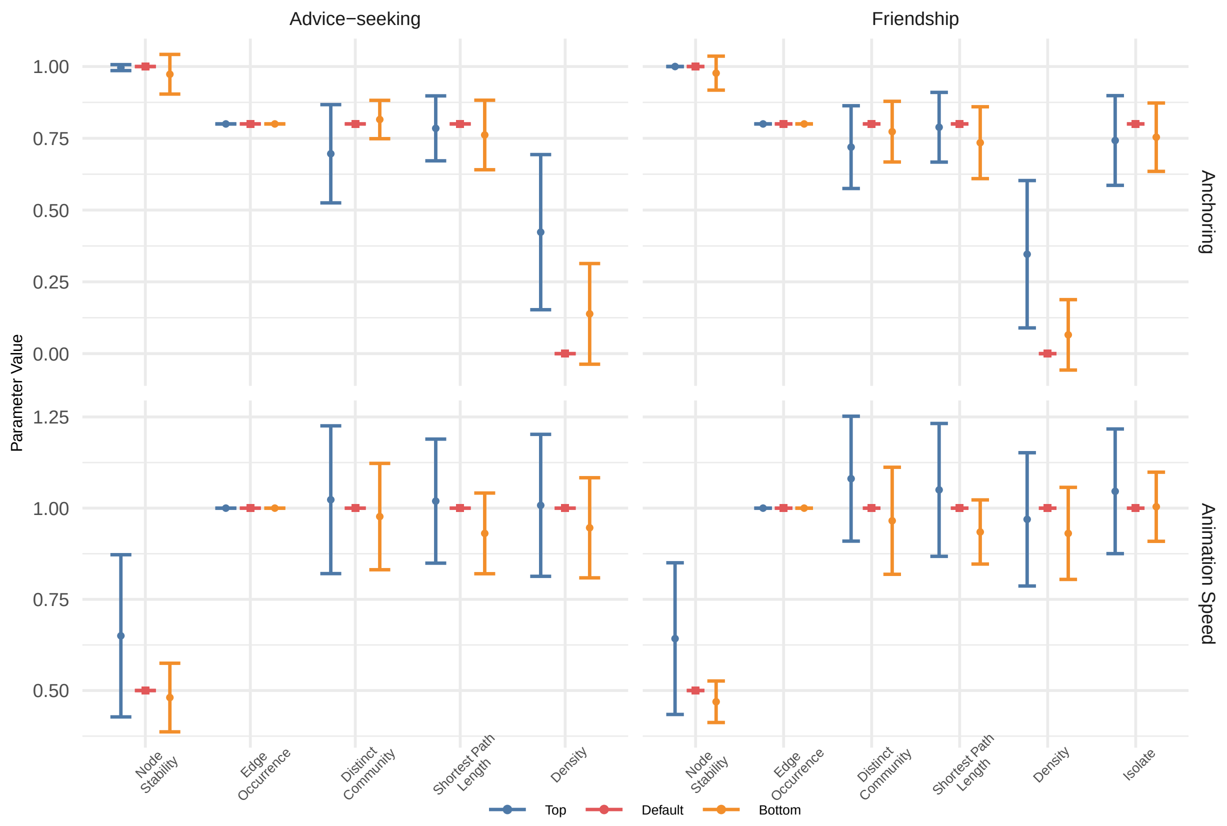
We compute the 95% CI for each of the visualization parameters (anchoring alpha and animation speed) by task and by group for each CSS dataset. We show a glimpse of the CI data below.

```r
source("./scripts/helper-sliderSwitcherParamCI.R")
head(ciDF, 10)
```

```
## # A tibble: 10 x 7
##    lower_ci mu_hat upper_ci task                   param    userType data
##       <dbl>  <dbl>    <dbl> <fct>                  <chr>    <fct>    <chr>
##  1    0.525  0.696    0.867 "Distinct\nCommunity" Anchori~ Top      Advice-seek~
##  2    0.749  0.815    0.882 "Distinct\nCommunity" Anchori~ Bottom   Advice-seek~
##  3    0.575  0.719    0.863 "Distinct\nCommunity" Anchori~ Top      Friendship
##  4    0.667  0.773    0.879 "Distinct\nCommunity" Anchori~ Bottom   Friendship
##  5    0.586  0.742    0.899 "Isolate"              Anchori~ Top      Friendship
##  6    0.635  0.754    0.873 "Isolate"             Anchori~ Bottom   Friendship
##  7    0.672  0.785    0.898 "Shortest Path\nLeng~ Anchori~ Top      Advice-seek~
##  8    0.640  0.762    0.883 "Shortest Path\nLeng~ Anchori~ Bottom   Advice-seek~
##  9    0.667  0.788    0.910 "Shortest Path\nLeng~ Anchori~ Top      Friendship
## 10    0.609  0.735    0.860 "Shortest Path\nLeng~ Anchori~ Bottom   Friendship
```

To visualize all parameter CIs, we recreate Figure 7 of the paper below.

```
sliderParamPlot
```



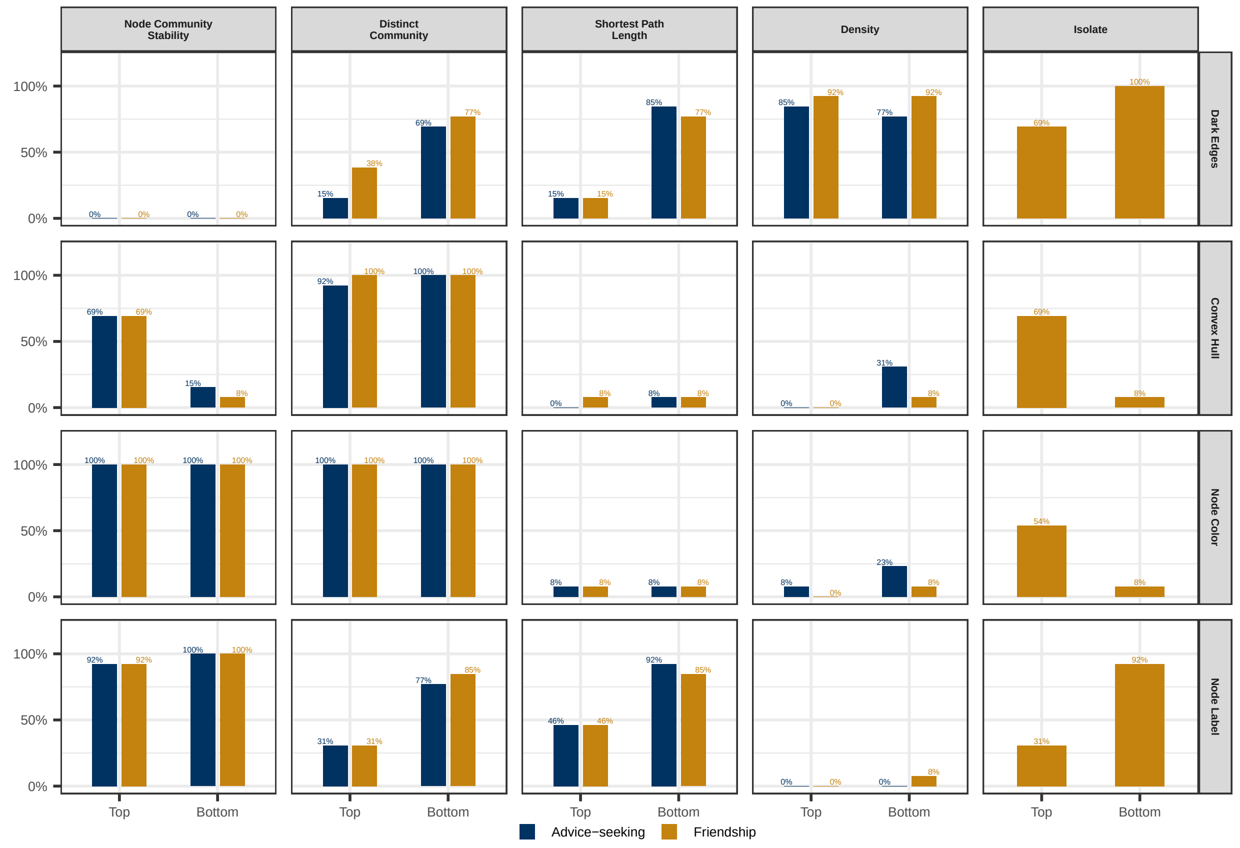
We compute the percentage of participants who used each of the graphical elements (edge opacity, node color, convex hulls, and node labels) by task and by group for each CSS dataset. Again, we show a glimpse of the percentage data below.

```
head(switcherPlotDF, 10)
```

```
## # A tibble: 10 x 5
## # Groups:   param, task [6]
##    param       task                         userType percent data
##    <fct>       <fct>                        <fct>      <dbl> <chr>
##  1 Convex Hull "Distinct\nCommunity"        Top       0.923  Advice-seeking
##  2 Convex Hull "Density"                    Top       0      Advice-seeking
##  3 Convex Hull "Shortest Path\nLength"      Top       0      Advice-seeking
##  4 Convex Hull "Node Community\nStability"  Top       0.692  Advice-seeking
##  5 Convex Hull "Distinct\nCommunity"        Top       1      Friendship
##  6 Convex Hull "Density"                    Top       0      Friendship
##  7 Convex Hull "Shortest Path\nLength"      Top       0.0769 Friendship
##  8 Convex Hull "Node Community\nStability"  Top       0.692  Friendship
##  9 Convex Hull "Isolate"                    Top       0.692  Friendship
## 10 Dark Edges  "Distinct\nCommunity"        Top       0.154  Advice-seeking
```

To visualize the use of graphical elements and visual aids, we recreate Figure 8 of the paper below.

```
switcherPlot
```



3.5 Precision of Inference & Time-Accuracy Correlation

As mentioned in Section 5.4 of the paper, two forms of error can impact the precision of inferences about network statistics made with NetHOPs: perceptual and cognitive errors related to how accurately analysts can estimate probabilities from the visualization (which applies to any visualizations of distributions), and approximation error introduced by sampling from the network model.

We load the scripts for all the helper functions needed to quantify the sampling error introduced when taking a set of random draws from the graph model.

```
source("./scripts/helper-EMDBias.R")
```

The script processing time is long. The output EMD bias can be directly loaded from the `data` directory.

```
EMDBiasDF = read.csv("./data/EMDbias.csv")
EMDBiasDF %>% head(10) %>% knitr::kable(.)
```

AC	FC	FI	ASP	FSP	AD	FD
0.00	0.50	0.45	0.8	0.40	0.35	0.35
0.80	0.60	0.45	0.9	0.55	0.40	0.25
0.85	0.25	0.20	0.0	0.50	0.40	0.45
0.85	0.25	0.35	0.9	0.40	0.45	0.35
0.20	1.00	0.55	0.9	0.50	0.50	0.55
0.00	0.50	0.15	0.0	0.60	0.30	0.45
0.00	0.50	0.30	0.9	0.45	0.40	1.00

AC	FC	FI	ASP	FSP	AD	FD
0.00	1.00	0.85	0.0	0.35	0.40	0.60
1.50	0.45	0.15	0.9	0.55	0.35	0.40
1.50	0.15	0.45	0.9	0.80	0.60	0.40

We cannot compute approximation error against the ground truth model, we can infer it by re-sampling N sets of network realizations from the model and using the distributions of network statistics from each re-sampled set to compute EMD scores against those from the set of NetHOPs. We can then quantify the sampling error of the distribution for each network statistic in the unit of EMD by constructing a confidence interval. Therefore, if a participant perfectly perceived and sketched a distribution and received an EMD score of zero using NetHOPs, her perception could be off by an $\mu_{EMD} \pm SE_{EMD}$ amount. This approach is generalizable and can be easily applied to compute the sampling error of any probability estimation tasks.

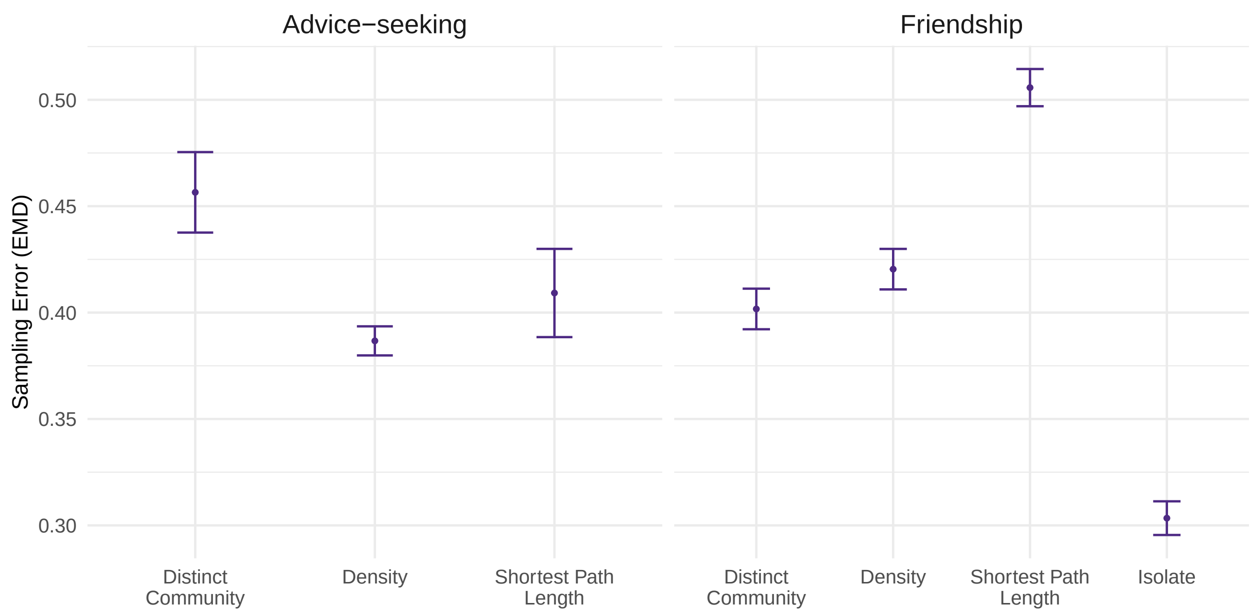
We show a glimpse of the EMD bias CIs computed.

```r
biasCIDF = data.frame(mean = EMDBiasDF %>% apply(2, mean) ,
                      se = EMDBiasDF %>%
                        apply(2, function(x) sd(x) / sqrt(length(x)))) %>%
  mutate(var = rownames(.)) %>%
  mutate(network = ifelse(substring(var, 1, 1) == "A",
                          "Advice-seeking", "Friendship")) %>%
  mutate(task = case_when(var == "AC" ~ "Distinct\nCommunity",
                          var == "FC" ~ "Distinct\nCommunity",
                          var == "FI" ~ "Isolate",
                          var == "AD" ~ "Density",
                          var == "FD" ~ "Density",
                          var == "ASP" ~ "Shortest Path\nLength",
                          var == "FSP" ~ "Shortest Path\nLength")) %>%
  mutate(task = factor(task, levels = c("Distinct\nCommunity", "Density",
                                        "Shortest Path\nLength", "Isolate")))
biasCIDF %>% head(10) %>% knitr::kable(.)
```

	mean	se	var	network	task
AC	0.4565	0.0189093	AC	Advice-seeking	Distinct Community
FC	0.4017	0.0095387	FC	Friendship	Distinct Community
FI	0.3034	0.0079159	FI	Friendship	Isolate
ASP	0.4092	0.0207299	ASP	Advice-seeking	Shortest Path Length
FSP	0.5057	0.0087463	FSP	Friendship	Shortest Path Length
AD	0.3867	0.0068359	AD	Advice-seeking	Density
FD	0.4204	0.0095147	FD	Friendship	Density

We recreate Figure 9 of the paper to visualize the sampling error of the network statistics.

```r
ggplot(biasCIDF, aes(x=task, y=mean, group = task)) +
  geom_point(color = "#4E2A84", size = 0.8)+
  geom_errorbar(aes(ymin=mean-se, ymax=mean+se), width=.2,
                position=position_dodge(0.05), color = "#4E2A84") +
  labs(x = NULL, y = "Sampling Error (EMD)") +
  facet_grid(cols = vars(network), scale = "free", drop = T) +
  theme_minimal() +
  theme(axis.title.y = element_text(size = 10),
        axis.text.x = element_text(size = 9),
        strip.text.x = element_text(size = 12))
```



As also mentioned in Section 5.4, our participants may not view all NetHOPs' realizations. While we did not log exact realizations viewed, we can use time spent on tasks to infer approximate viewing.

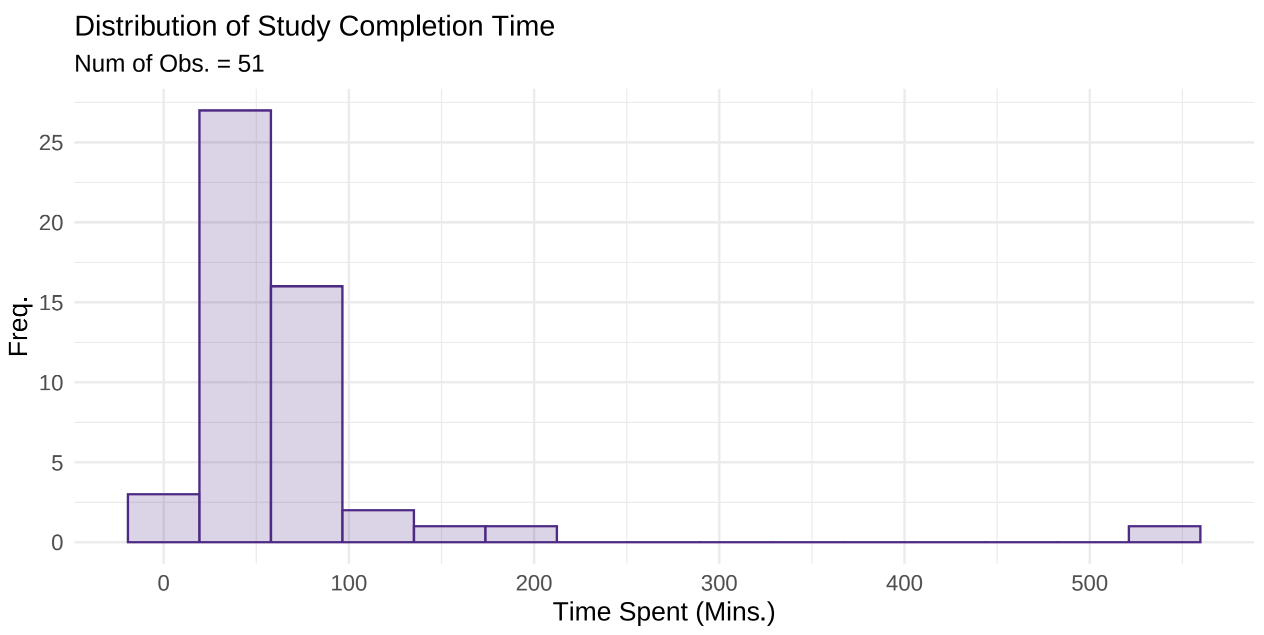
We first present the summary statistics of the task completion time without the outlier.

```
# Identify the outlier
outlier_i = userResponseDF %>%
  select(ends_with("timeSpent")) %>%
  select(-test_timeSpent, -demographics_timeSpent) %>%
  `/`(60) %>%
  apply(1, sum) %>%
  which.max()

# Compute the task completion time by participants
taskCompletionTimeVec = userResponseDF %>%
  select(ends_with("timeSpent")) %>%
  select(-test_timeSpent, -demographics_timeSpent) %>%
  `/`(60) %>%
  apply(1, sum)

# Present the summary statistics
taskCompletionTimeVec_noOutlier = taskCompletionTimeVec[-outlier_i]
c(summary(taskCompletionTimeVec_noOutlier), "sd" = sd(taskCompletionTimeVec_noOutlier))
```

```
##       Min.    1st Qu.     Median       Mean    3rd Qu.       Max.         sd
##   8.816667  35.833333  49.383333  55.040667  64.808333 176.250000  32.447425
```

We show the distribution of study completion time below.

```
source("./scripts/helper-correlationAnalysis.R")
timeDistPlot
```



We show the distributions of completion time by task.

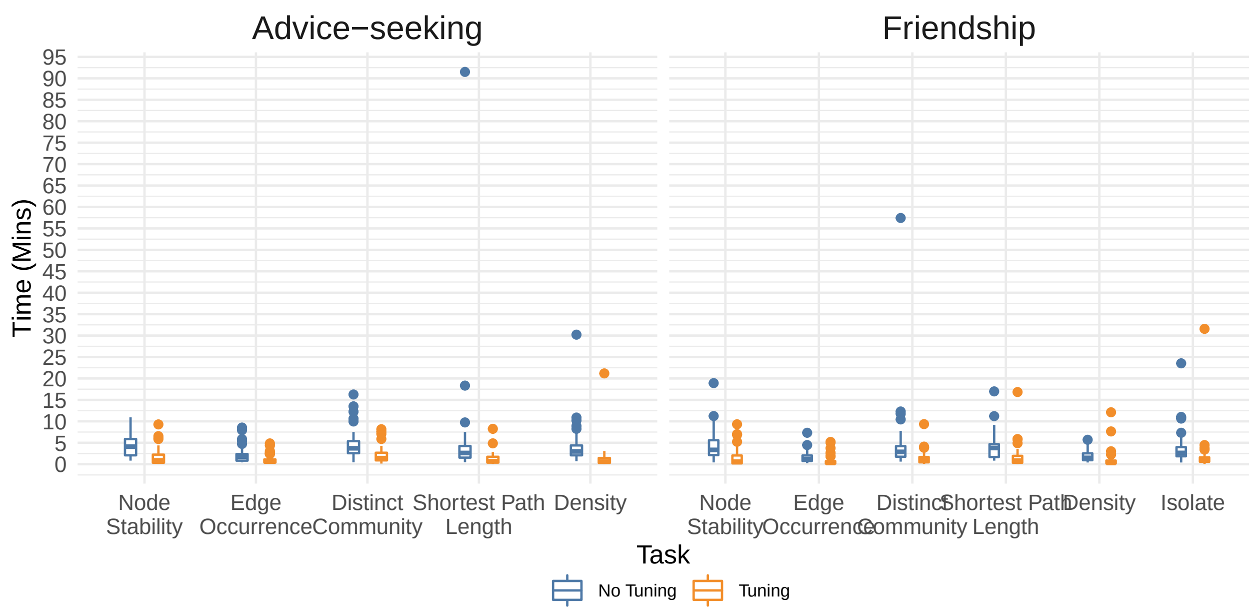
- Construct the dataframe that records the amount of time spent on each task.

```
timeSpentDF %>% select(-taskName) %>% head(10) %>% knitr::kable(.)
```

variable	value	tuneFlag	network	task
AC_noTune_timeSpent Community	2.7333333	No Tuning	Advice-seeking	Distinct
AC_noTune_timeSpent Community	2.4833333	No Tuning	Advice-seeking	Distinct
AC_noTune_timeSpent Community	16.2666667	No Tuning	Advice-seeking	Distinct
AC_noTune_timeSpent Community	4.8500000	No Tuning	Advice-seeking	Distinct
AC_noTune_timeSpent Community	4.2833333	No Tuning	Advice-seeking	Distinct
AC_noTune_timeSpent Community	0.8833333	No Tuning	Advice-seeking	Distinct
AC_noTune_timeSpent Community	6.2500000	No Tuning	Advice-seeking	Distinct
AC_noTune_timeSpent Community	2.7000000	No Tuning	Advice-seeking	Distinct
AC_noTune_timeSpent Community	3.9500000	No Tuning	Advice-seeking	Distinct
AC_noTune_timeSpent Community	3.8166667	No Tuning	Advice-seeking	Distinct

- Visualize the distribution of time spent by task.

```
timeDist_byTask_plot
```



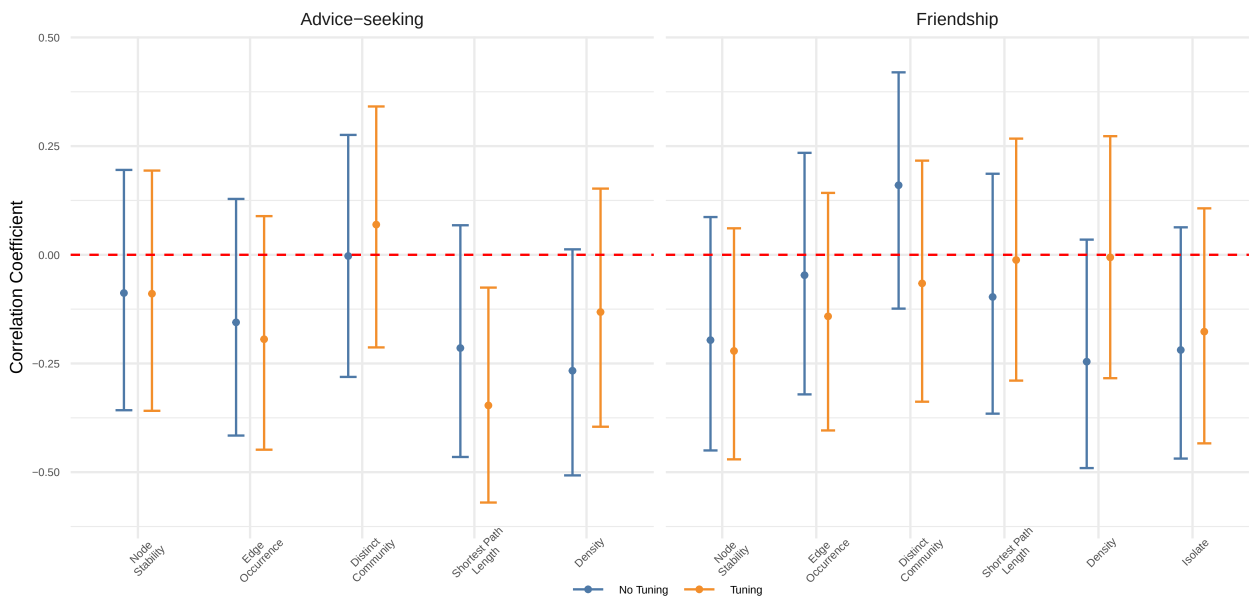
To assess whether spending more time on a task correlated with performance, we conducted a correlation analysis between task completion time and response quality by computing Pearson's correlation coefficient for each task, then using Fisher's Z-Transformation to meet the normality assumption.

```
cor_byTaskDF %>% select(-taskName) %>%
  select(task, network, tuneFlag, lwr.ci, cor, upr.ci) %>%
  head(10) %>% knitr::kable(.)
```

task	network	tuneFlag	lwr.ci	cor	upr.ci
Distinct Community	Advice-seeking	No Tuning	-0.2809246	-0.0027955	0.2757668
Distinct Community	Advice-seeking	Tuning	-0.2130161	0.0694494	0.3412014
Distinct Community	Friendship	No Tuning	-0.1238156	0.1600480	0.4196986
Distinct Community	Friendship	Tuning	-0.3379045	-0.0657400	0.2165707
Isolate	Friendship	No Tuning	-0.4687505	-0.2189734	0.0632281
Isolate	Friendship	Tuning	-0.4337563	-0.1767483	0.1068565
Shortest Path Length	Advice-seeking	No Tuning	-0.4650488	-0.2144622	0.0679413
Shortest Path Length	Advice-seeking	Tuning	-0.5698667	-0.3464777	-0.0754018
Shortest Path Length	Friendship	No Tuning	-0.3654396	-0.0969539	0.1864248
Shortest Path Length	Friendship	Tuning	-0.2893845	-0.0120037	0.2672370

Recreate Figure 10 of the paper and show the CIs of the correlation.

```
corOverViewPlot
```


Advice−seeking
Friendship
Correlation Coefficient
Node Stability
Edge Occurrence
Distinct Community
Shortest Path Length
Density
Isolate
No Tuning
Tuning