Session 6: Social Networks

Jennifer Lin

2022-05-03

Overview

You've likely experienced times in your life where you meet someone who is ostensibly a stranger but after a brief conversation, you realize you have a mutual friend. You might have remarked "What a small world!" Indeed, as network scientists will say, this is the Small World effect. Our daily interactions is full of things that can be represented by networks, which can help us better understand how the world works. As social psychologists might claim, humans are social creatures. We make connections and these links dictate many of the decisions we make in our daily lives. Our personalities are inherently linked to those who we call close friends. Sure, we make friends based on our common interests, but our friends are also a great source of influence. If we are runners, our friends might already be runners or we might influence them to take up running. Running, or any sport for that matter, is more fun with friends or a group of people. Another example is politics. Research shows that people who have someone to go to the polls with tend to vote more often. As cognitive psychologies might also state, we connect things in our world through semantics and shared meanings. Things that share the same meaning are often frequently associated with one another. Why do you think places like Ikea have sections for furniture that go in the same section of your house? When we cook, we know what foods go together and what spices work with certain foods but not others. But we also know that some ingredients, like peanut butter, is rather versatile. It can go with jam in a sandwich, with flour and sugar for a cookie or with carrots, noodles, and scallions (among others) for some good Thai Peanut Butter Noodles. In many ways, networks are everywhere and we have been using them all our lives. Today, we learn some R code that can help us display these connections in a graph.

In this section, our goals are as follows:

1. Understand basic network structure
2. Learn how to generate network data
3. Become familiar with graphing networks in R using base R and ggplot2

This process involves and requires a lot of packages. Here is the list of all of the packages that I use to generate the code that is discussed in this handout.

```
library(dplyr)      # For %>%, filter(), select()
library(igraph)     # For graph_from_data_frame()
library(ggplot2)    # For ggplot()
library(ggraph)     # For ggraph()
library(tidygraph) # For as_tbl_graph()
library(purrr)      # For map()
library(tibble)     # For tibble features
library(tidyr)      # For expand, separate
library(dundermifflin) # For Data
library(stringr)    # For title tools in functions
```

The Running Example

I am a big fan of the *The Office*, the US version of the mockumentary that documents life at the Dunder Mifflin paper company. As such, for this session, I thought it would be fun to take a step away from the social science examples and use The Office to explore social networks[1].

To this presentation, I have included several data files for us to work with this week. However, before we look at those, here is some background on the episodes I selected[2].

[1] After all, there is precedence for this kind of exercise in teaching examples, including things like the Game of Thrones.

[2] These summaries are my own, made with an eye towards highlighting the connections between key members of the episode. For more information, consult Google by searching the episode name. Of course, I would recommend just watching the episode but it is not my place here to promote subscription packages.

1. ***Halloween*** *(Season 2, Episode 5)*: It is Halloween at Dunder Mifflin and the party planning committee has big plans for an office celebration. However, Michael gets a phone call from Jan reminding him that he needs to let somebody go before the end of the day as a response to corporate plans for company downsizing. Angela, Pam, and Meredith put finishing touches on the party. Dwight asks Michael who he is going to fire, while Michael consults Pam, Jim, and Hank over his firing plans. In the meantime, Pam and Jim put Dwight's resume on Monster.com, among other job searching sites to get Dwight a job out of state.

2. ***Valentine's Day*** *(Season 2, Episode 16)*: For this year's Valentine's Day, Michael is off to New York to meet the new CFO, David Wallace. While he is there, he will also be conducting a presentation on the Scranton branch in front of David, Jan and other branch managers: Craig from Albany, Josh from Stanford and Dan from Buffalo. Meanwhile, back at the office, the Valentine's Day festivities is in full swing. Phyllis gets many bouquets of flowers delivered to her from her boyfriend, Bob Vance. As Pam signs each of these off, she becomes increasingly irritated that she did not get anything from Roy. In the meantime, Dwight gets a package from a special someone (Angela) and he intends to return the favor. He consults Pam for advice on gifts and Jim for spelling tips. As this happens, Kelly is wanting to spend Valentine's Day with Ryan, but

Ryan did not seem to be too excited. Kelly consults Jim on advice and Jim invites her to play cards with him and his friends rather than trying to make it work with Ryan.

3. ***Dwight's Speech*** *(Season 2, Episode 17)*: Dwight is the top paper salesman of the year for Dunder Mifflin. It is literally the highest honor that a Northeastern Pennsylvania-based mid-sized paper company regional salesman can attain. For his efforts, he gets a little prize money and is honored at some convention[3]. To prepare for the big speech, Michael spends the morning coaching Dwight on how to give a good speech so that he does not embarrass Michael or the company. When Dwight continuously fails, Michael seems to be increasingly irritated and gives up. Jim offers Dwight some advice, imploring him to speak from his heart. Since Dwight is all about authority, Jim provides him talking points from some of the world's famous dictators such as Benito Mussolini. While Dwight prepares his talk, Pam plans for her wedding and Jim, viably uncomfortable at watching those plans unfold, decides to plan a trip somewhere in the world. In the meantime, everyone else in the office has a fight over the thermostat. Some people, like Oscar and Kevin, like it cold, while others, like Creed, prefer it warm. People pretend that they are getting water from the water cooler, but they are really there to adjust the temperature on the thermostat that is located above it. Meredith keeps thinking that she is getting hot and cold flashes. At the end of the day, Dwight gives his speech to a roaring crowd of salespeople when he urges them to unite and to never acquiesce.

[3] Descriptions come courtesy of Jim's first talking head in the episode.

My working examples in this presentation will be based on the *Halloween* episode, but you are free to choose between *Valentine's Day* and *Dwight's Speech* for your exercise.

Network Structures

In this section, I discuss some core components of social network analyses[4]. First, we look at nodes, or the actors involved in a network. Then, we turn to edges, or the links by which these actors are connected.

[4] For a great paper on Social Network Analyses in political science, see *Network Analysis and Political Science* by Michael Ward, Katherine Stovel, and Audrey Sacks (2011).

Nodes

- **Clustering**: The probability that two associates of a node are linked.
- **Centrality**: How concentrated the links are around a small number of nodes.

Edges

- **Path Length**: Number of steps it takes to connect a pair of nodes. Directly connected nodes have a path length of 1.
- **Degree**: Number of connections. In-Degree is the number of connections sent to a node and Out-Degree is the number of connections sent from a node.
- **Betweeness**: The extent to which a node lies between other nodes. Higher betweeness signals that nodes are more dependent on others for access to information or valued goods.
- **Directed** and **Undirected**: Directed networks occur when out-degrees are not reciprocated while undirected networks are reciprocated.
- **Density**: Ratio of ties in a network to the total number of possible ties.

Adjacency Matrix

An **adjacency matrix** is a square matrix with the names of the nodes as row names and as column names. The content in the body shows whether the two notes are connected (coded as 1) or not (coded as 0). To represent weights, or multiple connections, these codes can increase. Below is an example of an adjacency matrix made using R's `igraph`, a topic we will turn to next.

```
## 5 x 5 sparse Matrix of class "dgCMatrix"
##         Ann Bob Charlie David Emilia
## Ann       .   1       .     1      1
## Bob       1   .       1     .      .
## Charlie   .   1       .     .      1
## David     1   .       .     .      .
## Emilia    1   .       1     .      .
```

Data Structures for Plotting Networks

To plot any network, there are two different options that you can use.

1. Node and Edge List
2. Adjacency Matrix

For this simple example, I will make a friendship network of some students from Northwestern.

Node List

A **node list** gives us a data frame with information about each person or unit in the network. For our made up friendship network, here is the sample node list.

```
person <- c(
  "Ann", "Bob", "Charlie", "David", "Emilia",
  "Fiona", "Gabby", "Henry", "Ian", "Jake", "Kyle")
major <- c(
  "Econ", "Politial Science", "Econ", "Psychology", "Political Science",
  "Psychology + Political Science", "Political Science", "History + Econ",
  "History", "History", "Psychology + History")
gender <- c(
  "Female", "Male", "Male", "Male", "Female", "Female",
  "Female", "Male", "Male", "Male", "Male")
nodelist <- data.frame(person, major, gender)
nodelist
```

```
##     person                          major gender
## 1      Ann                           Econ Female
## 2      Bob               Politial Science   Male
## 3  Charlie                           Econ   Male
## 4    David                     Psychology   Male
## 5   Emilia              Political Science Female
## 6    Fiona Psychology + Political Science Female
## 7    Gabby              Political Science Female
## 8    Henry                 History + Econ   Male
## 9      Ian                        History   Male
## 10    Jake                        History   Male
## 11    Kyle           Psychology + History   Male
```

Edge List

An **edge list** is one that shows us who is connected to whom in the network.

```
person <- c(
  "Ann", "Bob", "Charlie", "David", "Emilia", "Fiona",
  "Gabby", "Henry", "Ian", "Jake", "Kyle", "Kyle")
target <- c(
  "Charlie", "Emilia", "Henry", "Fiona", "Fiona",
  "Gabby", "Bob", "Ian", "Jake", "Henry", "David", "Jake")
edgelist <- data.frame(person, target)
edgelist
```

```
##     person  target
## 1      Ann Charlie
## 2      Bob  Emilia
## 3  Charlie   Henry
## 4    David   Fiona
## 5   Emilia   Fiona
## 6    Fiona   Gabby
## 7    Gabby     Bob
## 8    Henry     Ian
## 9      Ian    Jake
## 10    Jake   Henry
## 11    Kyle   David
## 12    Kyle    Jake
```

Plotting Networks in R

We can easily visualize networks in R. There are two different ways to do so, and they both can achieve a similar goal. First, there is `igraph`, which is based on base R and `tidygraph/ggrpah` which is based on the `tidyverse` environment. I will introduce both of them today but you are generally free to choose the path that works for you.

For the purposes of this class, we will draw networks of co-appearances of characters in the Office. Here, I define co-appearance as one where two characters share a scene in which they have a spoken line. Therefore, if a character is in a scene but only appears in the background, they will not be included. Here, the nodes are the characters and the edges represent sharing a scene[5]. This network is undirected, since scenes do not reflect an actor intentionally calling attention to another actor. They are just in the frame together, with spoken dialogue. This network is weighted, with weights showing multiple scenes shared in any given episode.

[5] I know writing this out sounds incredibly basic, but for networks, it is crucial that you keep all the components organized.

Data Preparation

For the purposes of the exercises in this class, I have provided the data for you. All you need to do is read the data in your scripts using something like `read.csv()` along with the appropriate file path[6]. I am including the code here to show you what happened in the back end. You need not understand each line of the code in this section. But I am including it here in case you want to apply these methods to other data.

[6] The data are in the folder called `Edges/`. Pick the one that you want and direct your code over.

The data for this exercise come from the `dundermifflin` R package. When you load the package, you can call in the data, which is a transcript of The Office with Season, Episode and Scene labels.

```
# Load data, which is implicitly loaded when the
# dundermifflin package is loaded
data("office_quotes")
```

I need to clean up the data a bit, generating an episode-scene ID. I also rename the `character` variable because R sometimes confuses this with a `character` variable type. To prevent issues that might arise, we call this `Team`.

```
office_quotes_clean <- office_quotes %>%
  mutate(
    # Creates unique scene ID for each scene to use
    #  as a grouping variable
    episodeID = paste0(
      "S", season, "E", episode, "Sc", scene
    )
  ) %>%
  # Rename the "character" variable to "Team" just so R
  #   would not confuse the variable name by the data
  #   type.
  rename(
    Team = character
  )
```

Since our network is based on one episode, there is no need to clean out the entire dataset. Here, I take out the episode of interest and clean out spaces for the edge list generation process later. I also take out the talking heads since they are scenes with only one character.

```
# Find the episode that you want
Season <- office_quotes_clean %>%
  filter(season == 2) %>%
  filter(episode == 5) %>%
  select(episodeID, Team) %>%
  # Get rid of spaces in the name because purrr does
  #   not like this for some reason
  mutate(Team = gsub(" ", "", Team)) %>%
  group_by(episodeID) %>%
  # Only maintain unique names within each scene
  distinct(Team, .keep_all = TRUE) %>%
  # Filters out the talking heads
  filter(n() >= 2) %>%
  filter(!is.na(Team))
```

Now, I generate an edge list. This groups the characters by scene and generates all unique pairs of characters in any given scene.

```
# Generate edge list grouping by the scene ID
Season_edge <- Season %>%
  group_by(episodeID) %>%
  group_split(.keep = FALSE) %>%
  purrr::map_dfr(tidyr::expand,
                 tidyr::crossing(Team, to = Team)) %>%
  purrr::pmap_dfr(~ tibble(!!!sort(c(...)) %>%
                             purrr::set_names(c("from", "to")))) %>%
  filter(from != to)
```

Since this is a weighted network, I need to generate some weights. This also collapses the number of rows in the dataset since we are only interested in the unique pairs.

```
# Generate edge weights based on how many co-occurrences
#  per episode
Season_edge_Weight <- Season_edge %>%
  mutate(
    pair_ID = paste0(from, " " , to)
  ) %>%
  group_by(pair_ID) %>%
  summarise(
    n = n(),
    weight = n/2) %>%
  tidyr::separate(
    pair_ID,
    into = c("from", "to"),
    sep = "\\s",
    remove = TRUE
  ) %>%
  select(from, to, weight)
```

The data files that we are using for this section reflect the `Season_edge_Weight` object for each episode. We will proceed to plot the network in the next two sections, first using `igraph` and then with `ggraph`. For the exercise, simply read it in as you would any other data file.

```
Season_edge_Weight <- read.csv("Edges/Halloween.csv") %>%
  select(-X)
```

Here, I will also generate a relevant nodes list for these data

```
Season_nodes <- Season_edge_Weight %>%
  gather(position, person, from:to) %>%
  distinct(person, .keep_all = FALSE) %>%
  mutate(
    department = case_when(
      person %in% c("Angela", "Oscar", "Kevin") ~ "Accounting",
      person %in% c("Jim", "Dwight", "Stanley", "Phyllis") ~ "Sales",
      person == "Pam" ~ "Reception",
      person %in% c("Michael", "Jan", "Craig", "Dan",
                    "DavidWallace") ~ "Management",
      TRUE ~ "Other Department"
    ),
    color = case_when(
      department == "Accounting" ~ "green4",
      department == "Sales" ~ "blue",
      department == "Reception" ~ "yellow",
      department == "Management" ~ "orange",
      department == "Other Department" ~ "purple"
    )
  )
```

Using `igraph`

The first package I will introduce is `igraph`. This is build more towards a "base R" style. It can also be used in Python, Mathematica, and C.

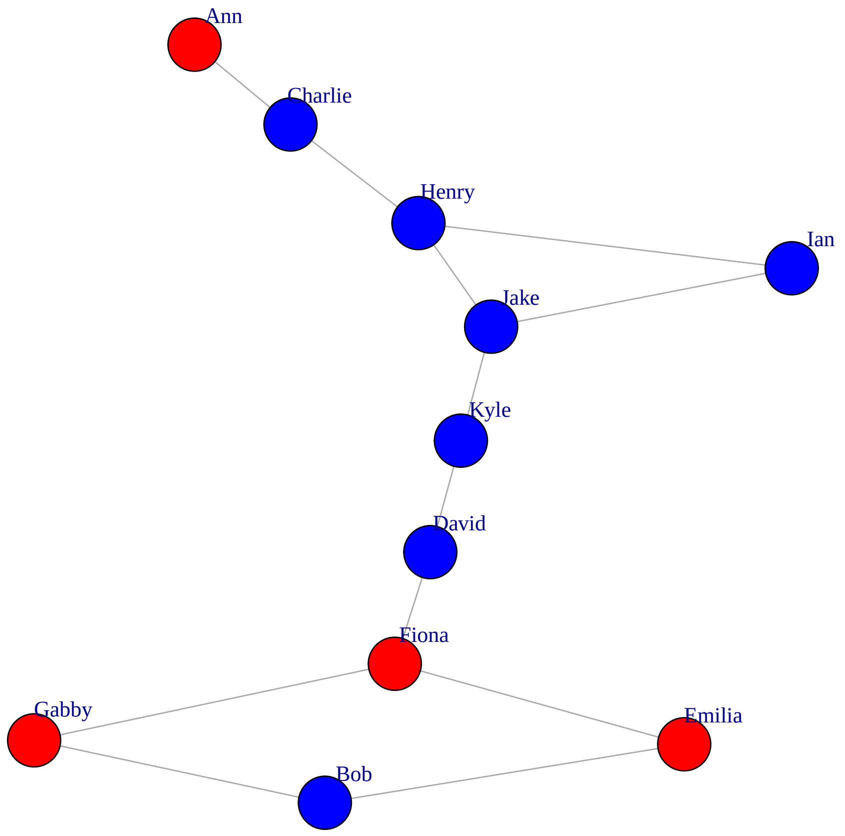
To create any network, you need a specified nodes list, edges list or adjacency matrix. Oftentimes, these come in the form of a data frame, which is the case for our data today. Therefore, we use the `graph_from_data_frame()` function to convert the data frame into something that `igraph` can recognize for networks. By default, it assumes networks are directed.

When we make a network, we need to follow these steps:

1. Make the network object using `graph_from_data_frame()`
2. Plot using `plot()`

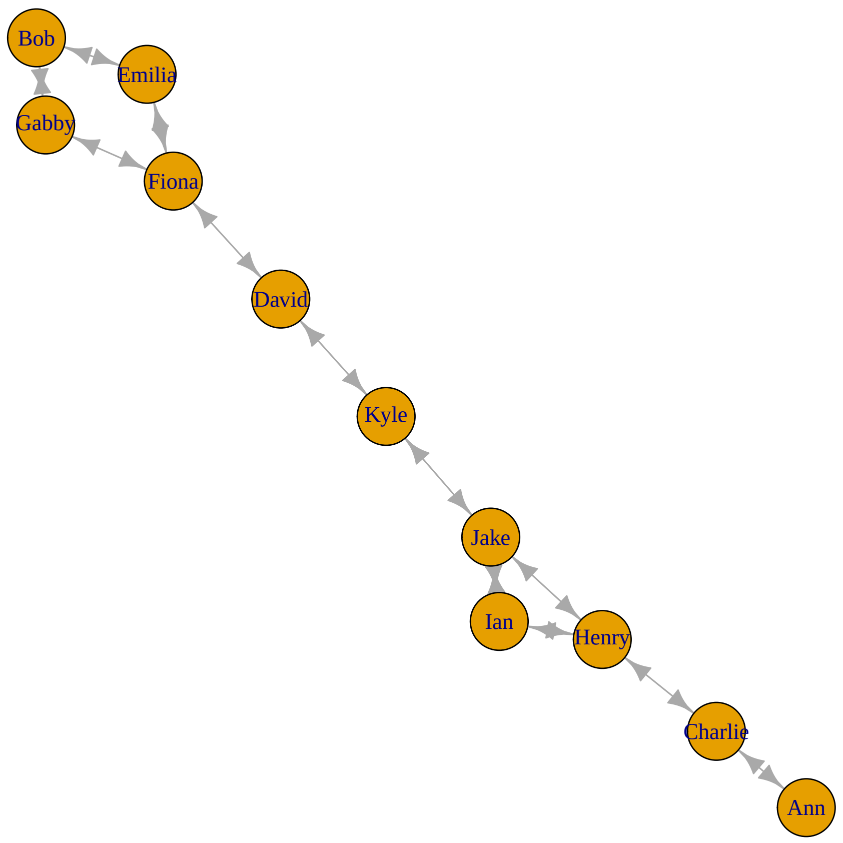
Here is a basic network from the made up friendship network example from before using the nodes and edge list. Before plotting, however, it is worth noting the nodes and edges. The nodes are the students and the edges are whether they share a major. The nodes are colored by gender and the network is unweighted and undirected. Here is the network:

```
# Make the network object
g <- graph_from_data_frame(edgelist, directed=FALSE, vertices=nodelist)
# Determine vertex colors
V(g)$color <- ifelse(nodelist$gender == "Female", "red", "blue")
# Plot the network
plot(
  g,
  # Declare colors to be the ones we preset
  vertex.color=V(g)$color,
  # Change size of label and distance from the node
  vertex.size = 14, vertex.label.dist=1.5,
  # Change label size
  vertex.label.cex = 1)
```



Using the network object from above that we creatively named `g`, we can get an adjacency matrix and make the same network from the adjacency matrix:

```
# Generate Network object
sample_edges <- get.adjacency(g) %>%
  graph_from_adjacency_matrix()
# Plot
plot(sample_edges)
```



Below, we use the data from *The Office* and indicate that this is not a directed network.

```
# Get data from edge list
Halloween <- graph_from_data_frame(
  Season_edge_Weight, directed = FALSE, vertices=Season_nodes)
# Get weights
E(Halloween)$weight <- Season_edge_Weight$weight
# Get Colors
V(Halloween)$color <- Season_nodes$color
# Get network statistics
# Degrees
degree <- degree(Halloween, mode="all")
# Clustering coefficient
cluster <- transitivity(Halloween)
# Path Length
path_length <- distances(Halloween)
# Average Path Length
mean_path = mean_distance(Halloween)
# Betweeness
betweeness <- betweenness(Halloween)
# Density
```

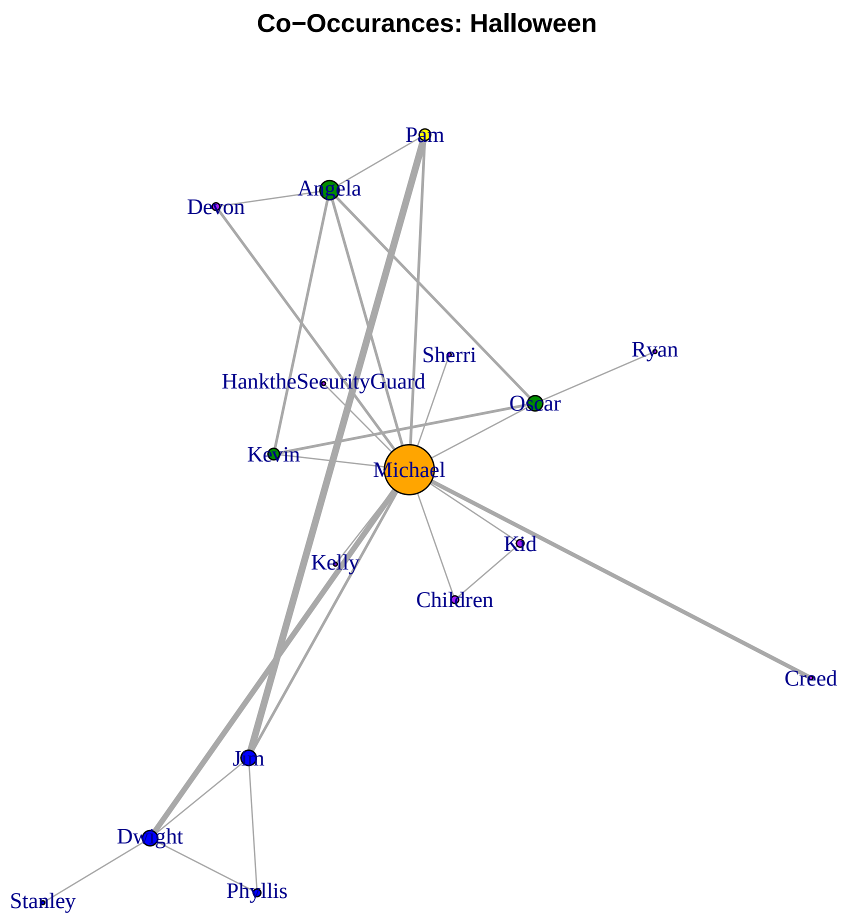
```
density <- edge_density(Halloween, loops=TRUE)
```

To set edge and vertex attributes, `igraph` uses `E(dataframe)` and `V(dataframe)` respectively. Therefore, we can designate the weight column to be the edge weight using `E(Halloween)$weight <- Season_edge_Weight$weight`. On top of this, `igraph` has nifty built in functions to calculate statistics on networks.

Degrees `degree(Network)`
Cluster Coefficient `transitivity(Network)`
Path Length `distances(Network)`
Betweeness `betweenness(Network)`
Density `edge_density(Network)`

To plot a network using `igraph`, we use the base R graphics mechanism, which departs from the ggplot2 framework that we have discussed this quarter.

```
# Generate plot
plot(
  Halloween,
  # Call the edge width the weight
  edge.width = E(Halloween)$weight,
  # Set layout -- computer picks by default
  layout=layout_with_kk,
  # Change vertex size to reflect node degree
  vertex.size=degree,
  # Set uniform color for nodes
  vertex.color=V(Halloween)$color)
# Give your plot a title
title("Co-Occurances: Halloween")
```



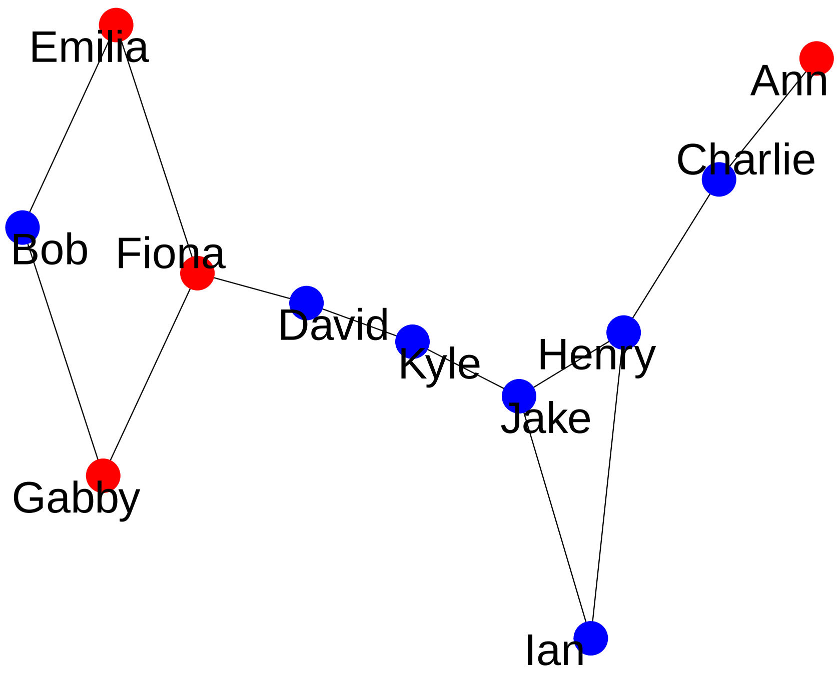
Using `tidygraph` and `ggraph`

If you are, like me, more familiar and comfortable in the `tidyverse` environment, `tidygraph` and `ggraph` provide the same `igraph` features that you can adopt using `tidyverse` syntax[7].

[7] More info on `tidygraph` here: `https://tidygraph.data-imaginist.com/`

The way to convert data frame to network objects is with `as_tbl_graph()` or `tbl_graph()`. Use the former if you are coming from an `igraph` object and the latter to declare your own nodes and edge list. Here is a basic network using the friendship data with the same network data from before.

```
# Declare the network object
g <- tbl_graph(
  edges = edgelist,
  directed = FALSE,
  nodes = nodelist)
# Start the graph
ggraph(g) +
  # Connect the Edges
  geom_edge_link()+
  # Get the nodes and color by the preset colors from before
  geom_node_point(aes(color = gender), size = 14)+
  # Label the nodes with the person's name
  geom_node_text(aes(label = person), size = 14, repel = TRUE) +
  # Add some color
  scale_color_manual(
    breaks = c("Male", "Female"),
    values = c(
      "Male" = "blue",
      "Female" = "red"
    )
  )+
  # Get rid of the grey background
  theme_void()+
  # Take out the legend
  theme(legend.position = 'none')
```

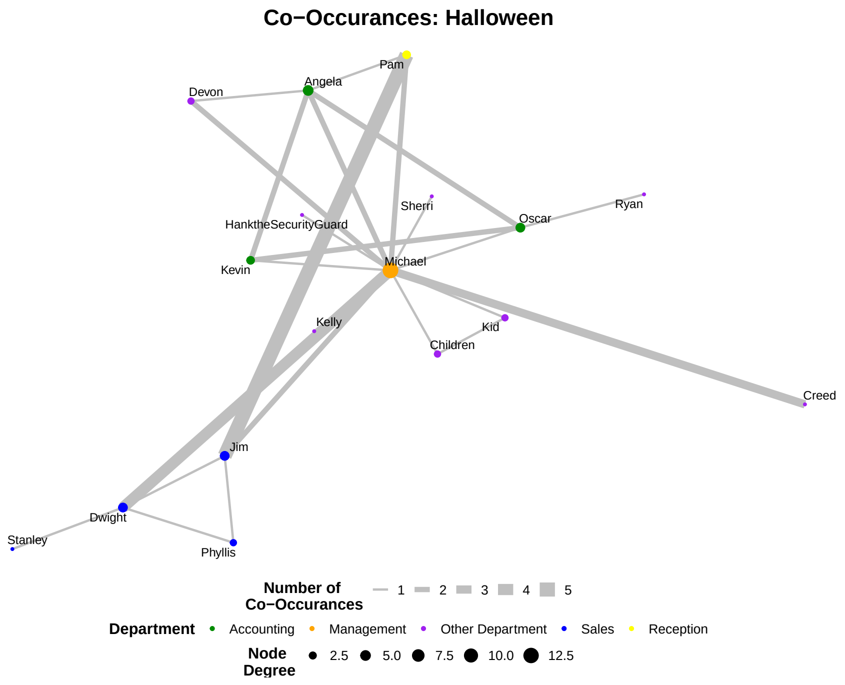


With pipes, we can calculate network statistics like we did with `igraph`. While the functions are a bit different, they communicate the same idea.The difference is the clustering coefficient, which requires some math as a workaround. First, you need to calculate the triangles and degree statistics and then apply `2*triangles/(degree*(degree-1))` to get clusters.

```
# Generate network object from prepped data
Halloween_network <- tbl_graph(edges = Season_edge_Weight, nodes = Season_nodes, directed = FALSE) %>%
  # Calculate network statistics
  mutate(
    degree = centrality_degree(),
    betweeness = centrality_betweenness(),
    path = node_distance_from(),
    mean_path = graph_mean_dist(),
    triangles = local_triangles(),
    cluster = 2*triangles/(degree*(degree-1))
  )
```

The positive aspect of this is that you can use `ggplot2` syntax to plot networks.

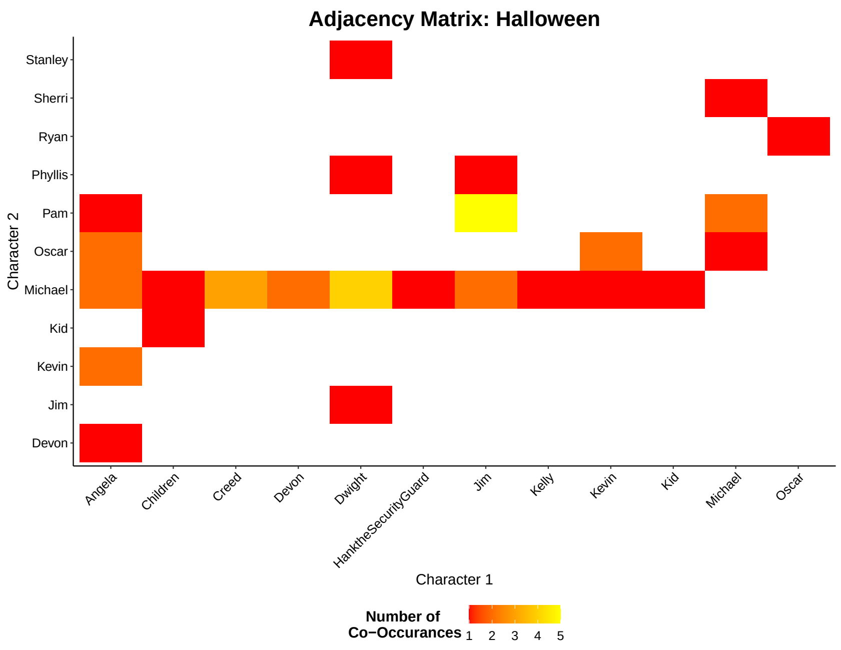
```
# Set up data and layout parameters
ggraph(Halloween_network, layout = 'kk') +
  # Set up edges, weighting the links and color all grey
  geom_edge_link(aes(width = weight), color = "gray75")+
  # Get nodes, and call the size by node degree and color all black
  geom_node_point(aes(size = degree, color = department))+
  # Label the nodes with the name of the characters
  geom_node_text(aes(label = person), repel = TRUE) +
  # Give the edge legend a title
  scale_edge_width(name = "Number of \nCo-Occurances")+
  # Give the size legend a title
  scale_size_continuous(name = "Node \nDegree")+
  # Get the colors
  scale_color_manual(
    name = "Department",
    breaks = c("Accounting", "Management", "Other Department", "Sales", "Reception" ),
    values = c(
      "Accounting" = "green4",
      "Management" = "orange",
      "Other Department" = "purple",
      "Sales" = "blue",
      "Reception" = "yellow" )
  )+
  # Give the plot a title
  labs(
    title = "Co-Occurances: Halloween"
  )+
  # Change the theme
  theme_void()+
  # Use other theme options as you would from ggplot
  theme(
    plot.title         = element_text(
      hjust = 0.5, size = 20, colour="black", face = "bold"),
    plot.subtitle      = element_text(
      hjust = 0.5, size = 16, colour="black", face = "bold"),
    legend.title       = element_text(
      hjust = 0.5, size = 14, colour="black", face = "bold"),
    plot.caption       = element_text(size = 10, colour="black"),
    legend.position    = 'bottom',
    legend.direction   = "horizontal",
    legend.box         = "vertical",
    legend.text        = element_text(size = 12, colour="black")
  )
```


Co−Occurances: Halloween
Pam
Angela
Devon
Sherri
Ryan
Oscar
HanktheSecurityGuard
Kevin
Michael
Kelly
Kid
Children
Creed
Jim
Dwight
Stanley
Phyllis
Number of Co−Occurances 1 2 3 4 5
Department Accounting Management Other Department Sales Reception
Node Degree 2.5 5.0 7.5 10.0 12.5

Plotting Adjacency Matrices

Adjacency matrices can be plotted as networks or as something that looks like a correlation matrix. Here, the same values are applied to the x and y axis, in this case, the nodes of the network. If a pair of actors on the nodes have a connection, you will shade the box where they meet and you can shade it in varying colors to reflect weight. This plot can be made simply using a standard `ggplot` and apply `geom_raster()`.

```
# Data Layer
ggplot(Season_edge_Weight, aes(x = from, y = to, fill = weight)) +
  # Use raster plot to get us the matrix
  geom_raster()+
  # Add titles
  labs(
    title = "Adjacency Matrix: Halloween",
    fill = "Number of \nCo-Occurances"
  )+
  # Change axis labels
  xlab("Character 1")+
  ylab("Character 2")+
  # Change the colors of the fill
  scale_fill_gradient(low = "red", high = "yellow", na.value = NA)+
  # Set your themes
  theme_classic()+
  theme(
    plot.title         = element_text(
      hjust = 0.5, size = 20, colour="black", face = "bold"),
    plot.subtitle      = element_text(
      hjust = 0.5, size = 16, colour="black", face = "bold"),
    legend.title       = element_text(
      hjust = 0.5, size = 14, colour="black", face = "bold"),
    plot.caption       = element_text(size = 10, colour="black"),
    axis.title         = element_text(size = 14, colour="black"),
    axis.text.x        = element_text(
      size = 12, colour="black", angle = 45, hjust = 1),
    axis.text.y        = element_text(size = 12, colour="black"),
    legend.position    = 'bottom',
    legend.direction   = "horizontal",
    legend.text        = element_text(size = 12, colour="black")
  )
```



A Note on Aesthetics

For this session, I added a prereading of Pfeffer (2017), which is a chapter in the Oxford Handbook of Political Networks. In this chapter, Pfeffer emphasizes the importance of good network visualizations and discusses the ways that you can make networks look better. Focusing on Figures 11.5 and 11.7, the author demonstrates the world of a difference that careful layout selection can make on the final product. Here, I will hone in on this issue and make a similar demonstration using data on flights to and from major US airports.

I got data from OpenFlights, which is a website that tracks almost everything related to aviation. Of particular interest is a dataset on routes that airlines commonly take between many airports around the world. We can read in those data here:

```r
# Flights Data
url <- "https://raw.githubusercontent.com/jpatokal/openflights/master/"
data_file <- "data/routes.dat"
flights <- read.csv(paste0(url, data_file), header = FALSE)

names <- c("Airline", "AirlineID", "From", "FromID",
           "To", "ToID", "Codeshare", "stops", "Equipment")
colnames(flights) <- names
```

Here, I am rather agnostic to which airlines to plot. Rather, since there are many airports in the world, I am primarily interested in some of the major airports in the US. Here are just a few that I will consider:

```
Airports <- c(
  "EWR", "IAD", "ORD", "IAH", "DEN", "LAX", "SFO",
  "FLL", "MIA", "CLT", "MSP", "CMH", "ATL", "DFW",
  "JFK", "LGA", "BOS", "DCA", "IAD", "DTW", "SLC",
  "MCO", "LAS", "SEA", "PHL", "BWI", "TPA", "BNA"
  )
```

To start, let's generate a data frame that allows us to get an edge list with origin airport, destination airport and the number of flights between these nodes.

```
short_list <- flights %>%
  filter(From %in% Airports) %>%
  filter(To %in% Airports) %>%
  mutate(
    pair_ID = paste0(From, " " , To)
  ) %>%
  group_by(pair_ID) %>%
  summarise(
    n = n(),
    .groups = 'keep'
    ) %>%
  tidyr::separate(
    pair_ID,
    into = c("from", "to"),
    sep = "\\s",
    remove = TRUE
  )
```

Like before, we can make it a network object and calculate basic network statistics as follows:

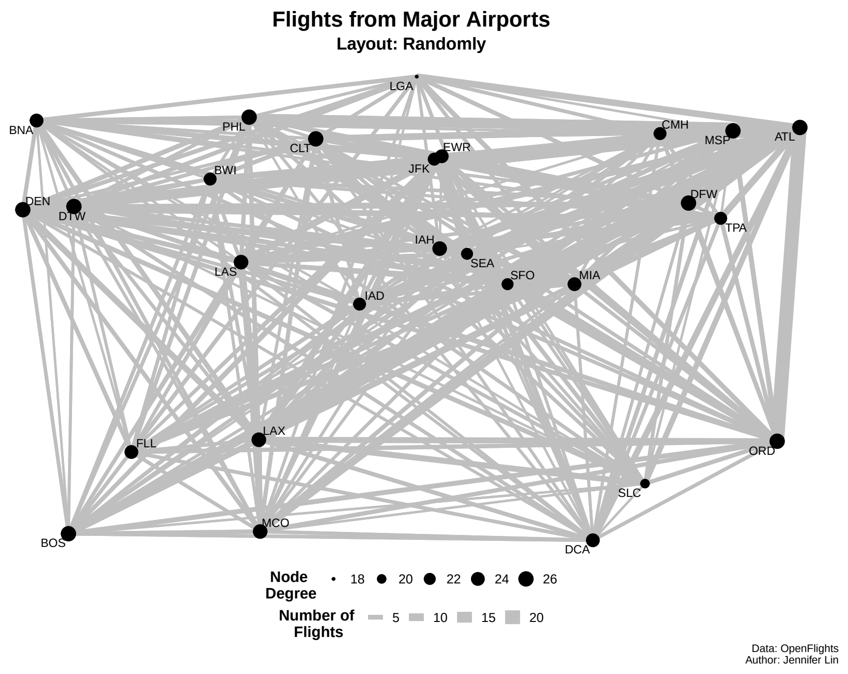
```
network <- as_tbl_graph(short_list) %>%
  mutate(
    degree = centrality_degree(),
    betweeness = centrality_betweenness(),
    path = node_distance_from(),
    mean_path = graph_mean_dist(),
    triangles = local_triangles(),
    cluster = 2*triangles/(degree*(degree-1))
  )
```

Before we make the plot, let's square away the theme function:

```
theme_gnetwork <- function(){
  theme_void()+
    theme(
      plot.title        = element_text(
        hjust = 0.5, size = 20, colour="black", face = "bold"),
      plot.subtitle     = element_text(
        hjust = 0.5, size = 16, colour="black", face = "bold"),
      legend.title      = element_text(
        hjust = 0.5, size = 14, colour="black", face = "bold"),
      plot.caption      = element_text(size = 10, colour="black"),
      legend.position   = 'bottom',
      legend.direction  = "horizontal",
      legend.box        = "vertical",
      legend.text       = element_text(size = 12, colour="black")
    )
}
```

Now, let's plot

```
ggraph(network, layout = 'igraph', algorithm = "randomly")+
  geom_edge_link(aes(width = n), color = "gray75")+
  geom_node_point(aes(size = degree), colour = "black")+
  geom_node_text(aes(label = name), repel = TRUE) +
  scale_edge_width(name = "Number of \nFlights")+
  scale_size_continuous(name = "Node \nDegree")+
  labs(
    title = "Flights from Major Airports",
    subtitle = "Layout: Randomly",
    caption = "Data: OpenFlights
    Author: Jennifer Lin"
  )+
  theme_gnetwork()
```



The results of this graph is what is commonly known as, in very unscientific terms, a mess. Sure, you can clearly see the nodes (airports) and maybe see the connections between them. But, by looking at this network, it is hard to tell if there is one airport that might be a central hub that all other airports have a direct flight to. It might also be harder to see which airports fly to which other ones more than others.

As such, let's take a page from Pfeffer (2017) – quite literally, page 293 – and analyze this network using global thresholds. As Pfeffer writes:

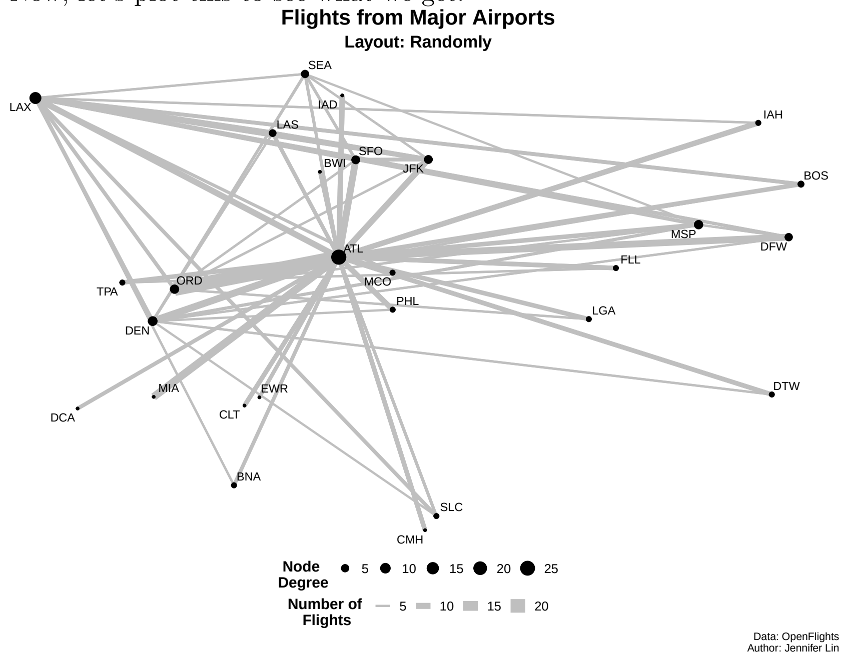
> When the network is weighted (each line is described by a value), we can remove all lines lower than a defined value (threshold). This threshold can be derived from the data (e.g., more than $1 billion trade volume) or from the readability of the figure (increase threshold until a clear structure is visible).

In short, we are setting a bar to which a weight must cross before being included in the network. This bar might be based on data, readability, or most ideally, theory. For this network, I am going to set the lower bound at 5. This is largely based on data. There are enough observations above this threshold to make an interesting graph and anything lower would render something that is still too busy.

To filter routes with more than 5 airlines servicing, we can use the following code, and convert to network object and calculate statistics while we are at it.

```
network <- short_list %>%
  filter(n >= 5) %>%
  as_tbl_graph() %>%
  mutate(
    degree = centrality_degree(),
    betweeness = centrality_betweenness(),
    path = node_distance_from(),
    mean_path = graph_mean_dist(),
    triangles = local_triangles(),
    cluster = 2*triangles/(degree*(degree-1))
  )
```

Now, let's plot this to see what we get.



While this looks better, there are many different network layouts that `igraph` and, subsequently, `ggraph` will allow us to use. Across the next few pages, I plot each of these layouts so you can see how the layouts work[8]. But, here is also a table to give you a glimpse of what each layout is, to aid you in your decision making process[9].

[8] Note that there are more layouts here than there are from the slides so please cross reference this document when you do your homework.

[9] Most of the descriptions here are taken from the ggraph documentation

randomly Places nodes uniformly random

circle Place nodes in a circle in the order of their index.

fr Places nodes according to the force-directed algorithm of Fruchterman and Reingold.

sphere Place nodes uniformly on a sphere - less relevant for 2D visualizations of networks.

drl Uses the force directed algorithm from the DrL toolbox to place nodes.

lgl Uses the algorithm from Large Graph Layout to place nodes.

kk Uses the spring-based algorithm by Kamada and Kawai to place nodes.

mds Perform a multidimensional scaling of nodes using either the shortest path or a user supplied distance.

graphopt Uses the Graphopt algorithm based on alternating attraction and repulsion to place nodes.

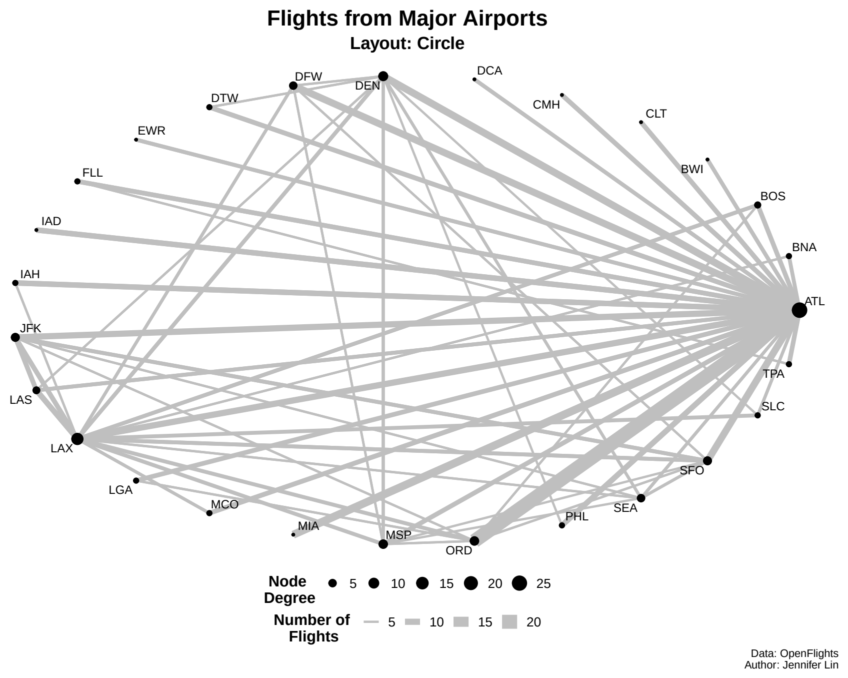
grid Place nodes on a rectangular grid.

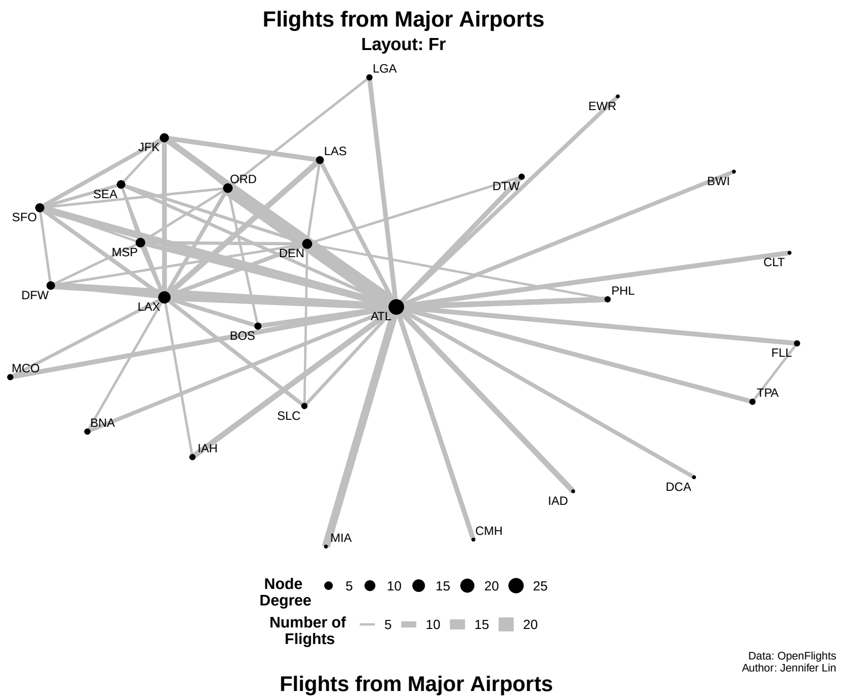
gem Place nodes on the plane using the GEM force-directed layout algorithm.

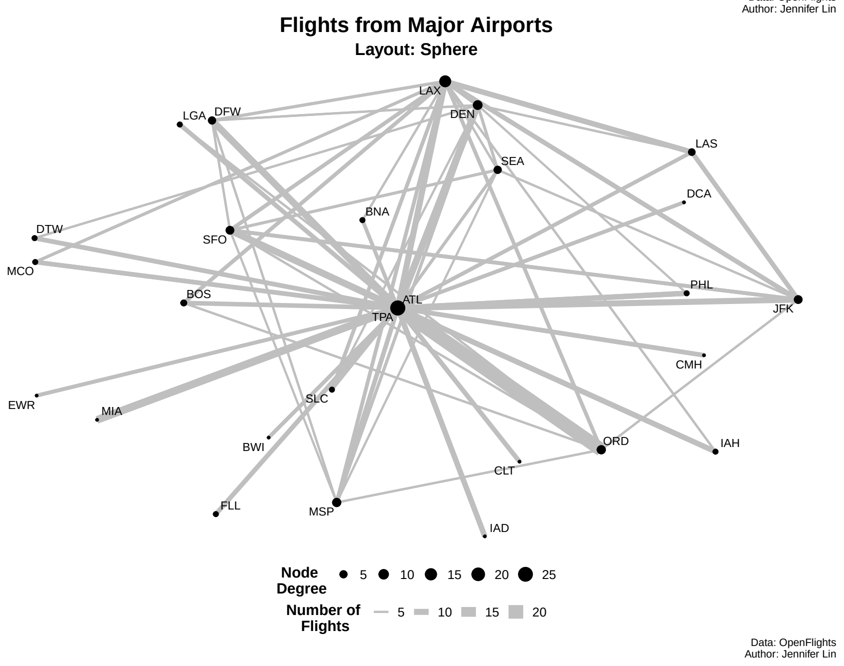
dh Uses Davidson and Harels simulated annealing algorithm to place nodes.

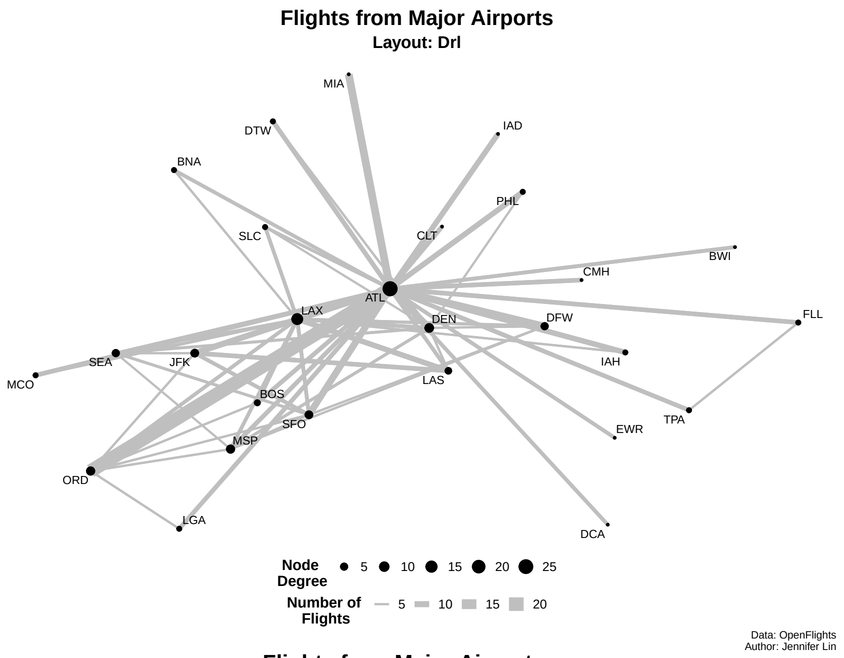
star Place one node in the center and the rest equidistantly around it.

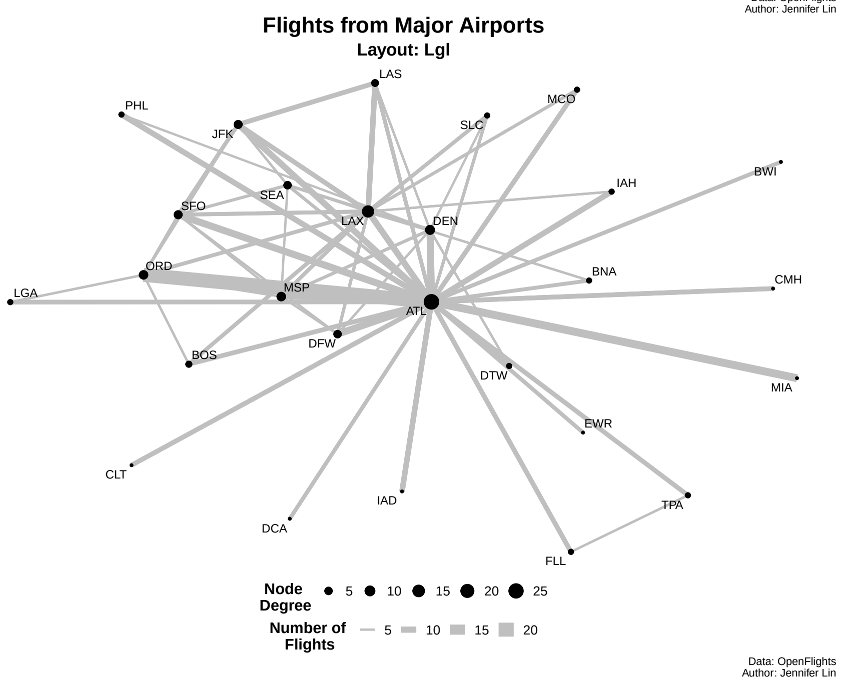
nicely Tries to pick an appropriate layout.

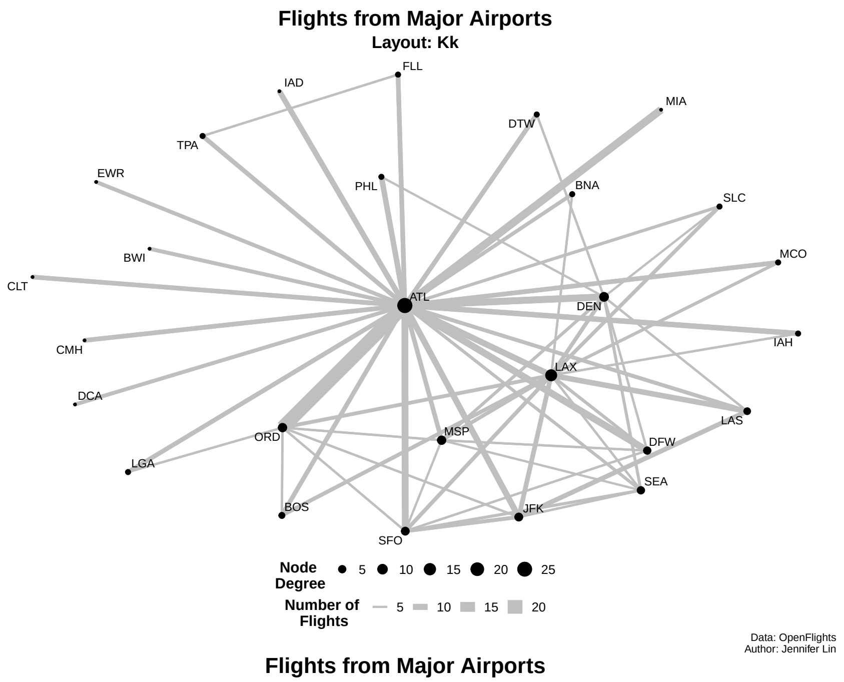


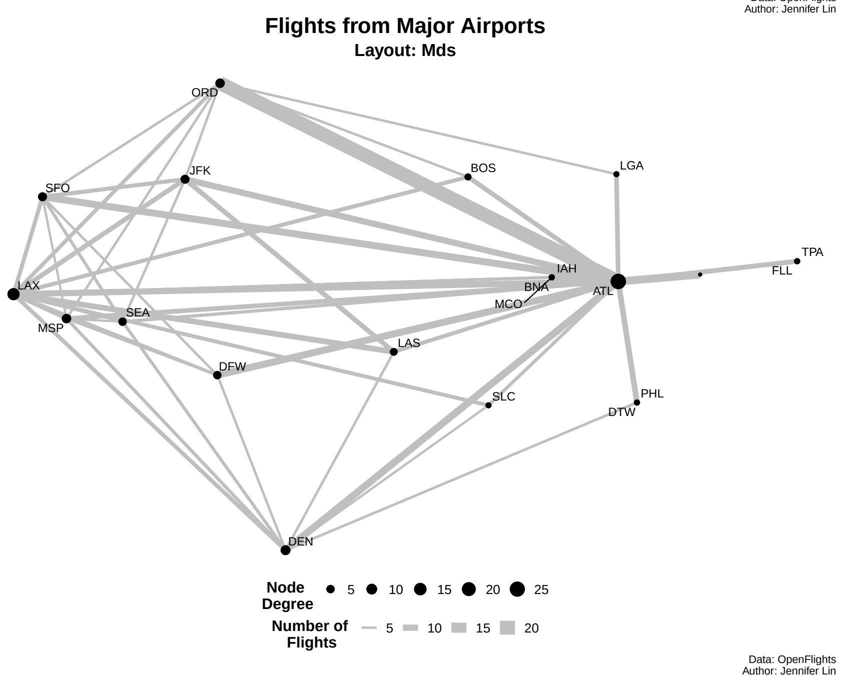

Flights from Major Airports
Layout: Fr
LGA
EWR
JFK
LAS
SEA
ORD
DTW
BWI
SFO
MSP
DEN
CLT
DFW
LAX
ATL
PHL
BOS
FLL
MCO
TPA
SLC
BNA
IAH
IAD
DCA
MIA
CMH
Node Degree
5
10
15
20
25
Number of Flights
5
10
15
20
Data: OpenFlights
Author: Jennifer Lin

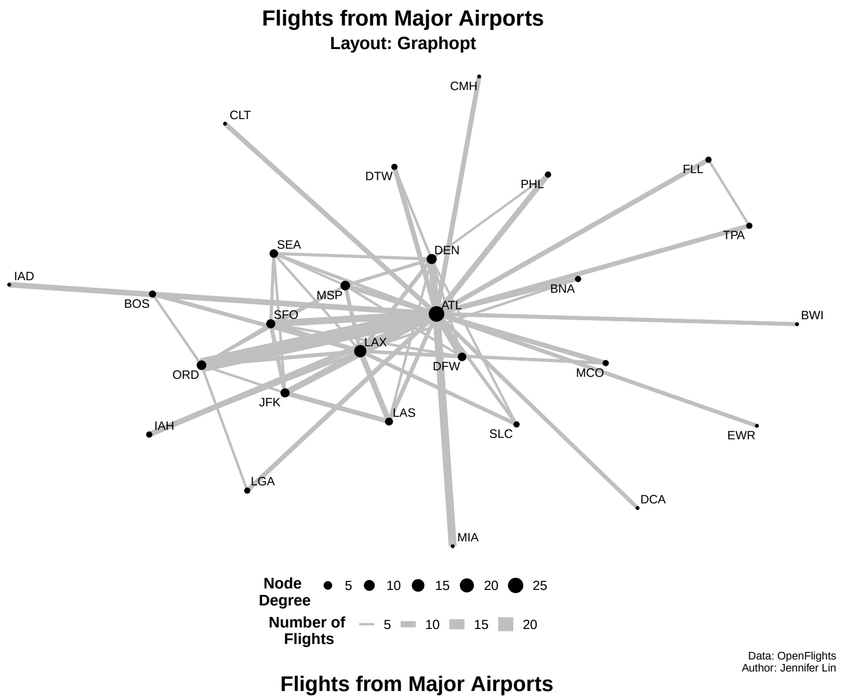

Flights from Major Airports
Layout: Sphere
LAX
DEN
LGA
DFW
LAS
SEA
DCA
BNA
DTW
SFO
MCO
PHL
BOS
ATL
TPA
JFK
CMH
EWR
SLC
MIA
BWI
ORD
IAH
CLT
FLL
MSP
IAD
Node Degree
5
10
15
20
25
Number of Flights
5
10
15
20
Data: OpenFlights
Author: Jennifer Lin

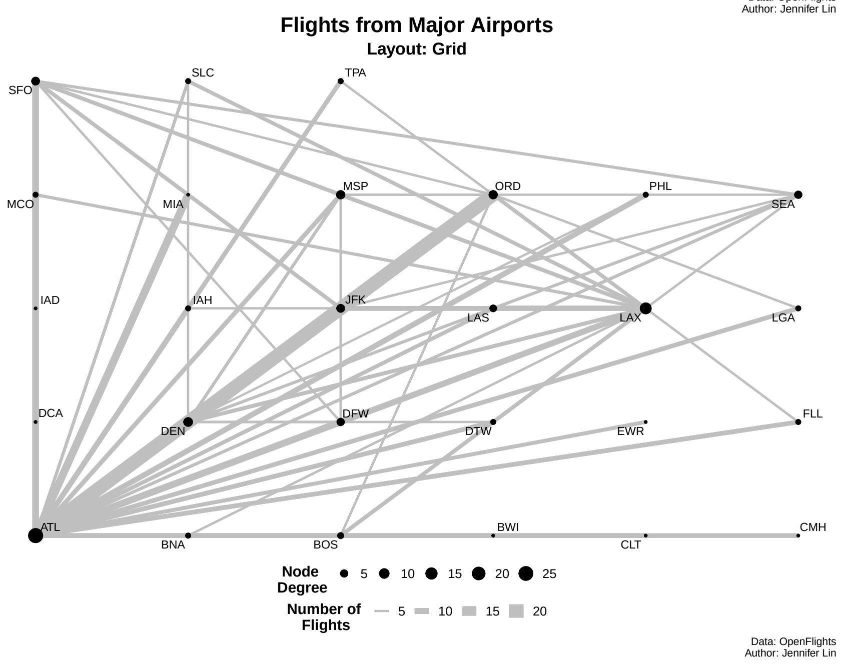

Flights from Major Airports
Layout: Drl
MIA
DTW
IAD
BNA
PHL
SLC
CLT
BWI
CMH
ATL
LAX
DEN
DFW
FLL
SEA
JFK
IAH
MCO
LAS
BOS
SFO
TPA
EWR
MSP
ORD
LGA
DCA
Node Degree 5 10 15 20 25
Number of Flights 5 10 15 20
Data: OpenFlights
Author: Jennifer Lin

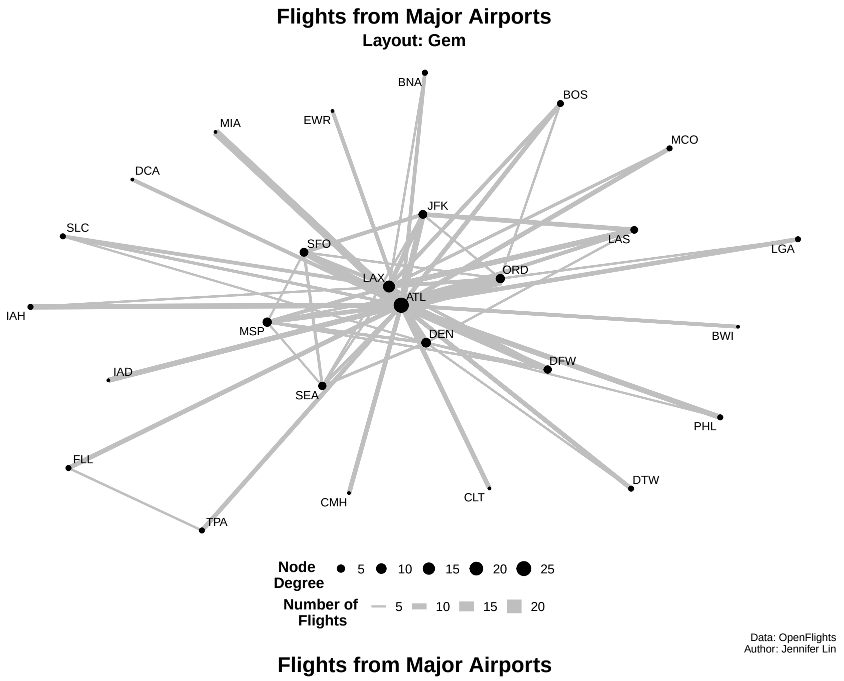

Flights from Major Airports
Layout: Lgl
LAS
PHL
MCO
SLC
JFK
BWI
IAH
SEA
SFO
LAX
DEN
BNA
ORD
CMH
LGA
MSP
ATL
DFW
BOS
DTW
MIA
EWR
CLT
IAD
TPA
DCA
FLL
Node Degree 5 10 15 20 25
Number of Flights 5 10 15 20
Data: OpenFlights
Author: Jennifer Lin

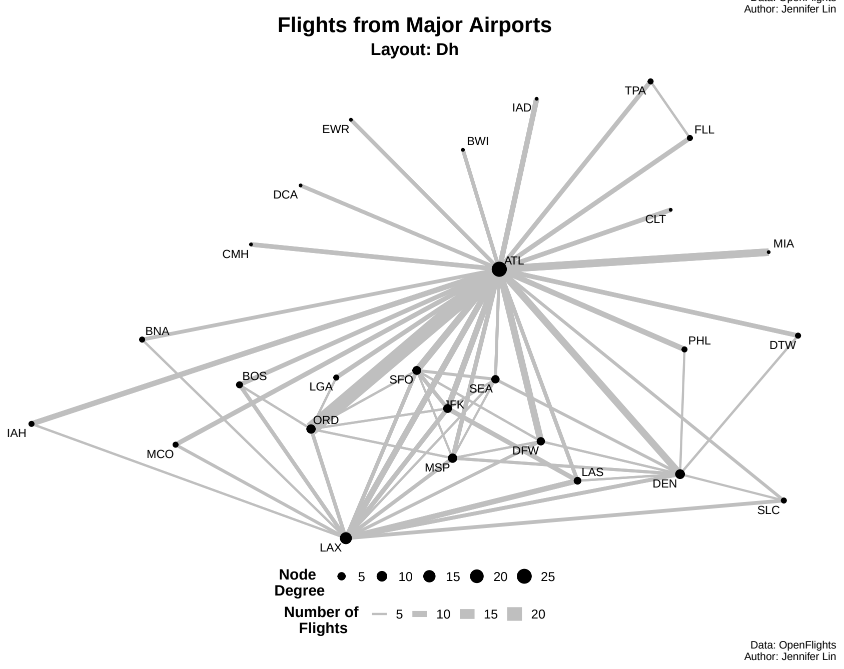

Flights from Major Airports
Layout: Kk
FLL
IAD
MIA
DTW
TPA
EWR
PHL
BNA
SLC
MCO
BWI
CLT
ATL
DEN
IAH
CMH
LAX
DCA
LAS
ORD
MSP
DFW
LGA
SEA
BOS
JFK
SFO
Node Degree
5
10
15
20
25
Number of Flights
5
10
15
20
Data: OpenFlights
Author: Jennifer Lin

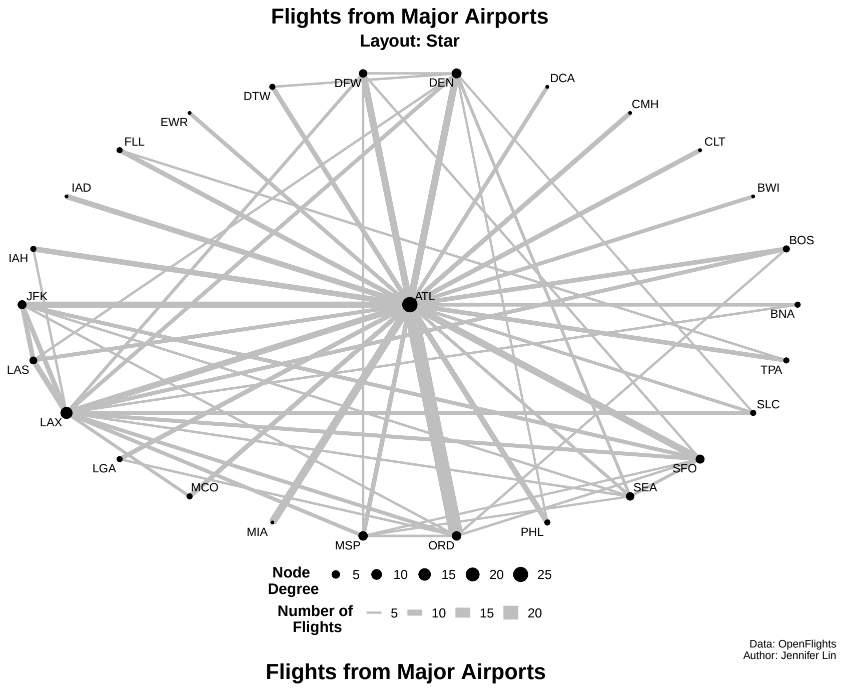

Flights from Major Airports
Layout: Mds
ORD
JFK
BOS
LGA
SFO
TPA
IAH
FLL
LAX
BNA
MCO
ATL
SEA
MSP
LAS
DFW
SLC
PHL
DTW
DEN
Node Degree
5
10
15
20
25
Number of Flights
5
10
15
20
Data: OpenFlights
Author: Jennifer Lin

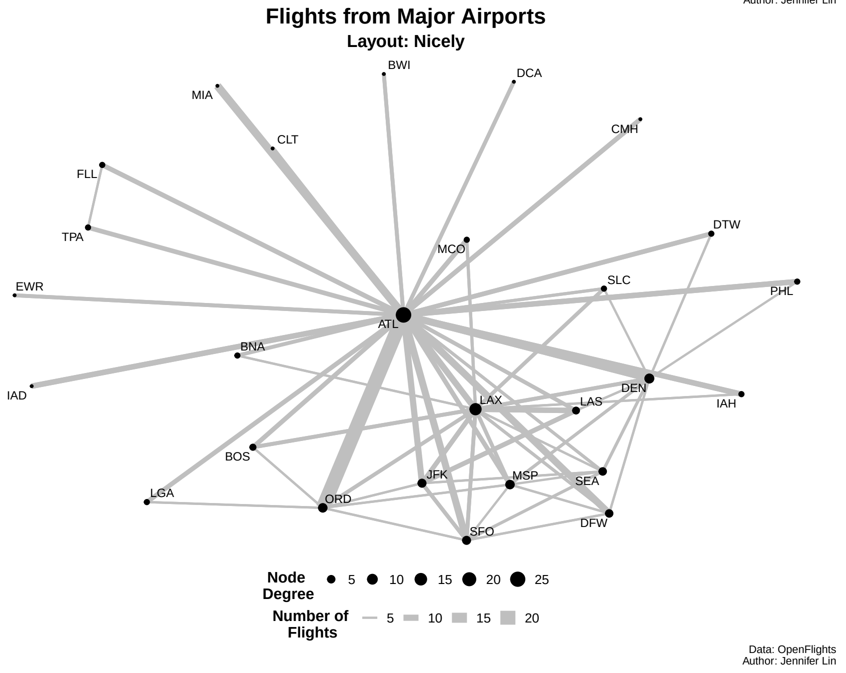

Flights from Major Airports
Layout: Graphopt
CMH
CLT
DTW
PHL
FLL
TPA
SEA
DEN
IAD
MSP
BNA
BOS
ATL
SFO
BWI
LAX
ORD
DFW
MCO
JFK
LAS
IAH
SLC
EWR
LGA
DCA
MIA
Node Degree
5
10
15
20
25
Number of Flights
5
10
15
20
Data: OpenFlights
Author: Jennifer Lin


Flights from Major Airports
Layout: Grid
SFO
SLC
TPA
MSP
ORD
PHL
MCO
MIA
SEA
IAD
IAH
JFK
LAS
LAX
LGA
DCA
DEN
DFW
DTW
EWR
FLL
ATL
BNA
BOS
BWI
CLT
CMH
Node Degree
5
10
15
20
25
Number of Flights
5
10
15
20
Data: OpenFlights
Author: Jennifer Lin


Flights from Major Airports
Layout: Gem
BNA
BOS
MIA
EWR
MCO
DCA
JFK
SLC
LAS
LGA
SFO
ORD
LAX
ATL
IAH
MSP
DEN
BWI
DFW
IAD
SEA
PHL
FLL
DTW
CMH
CLT
TPA
Node Degree
5
10
15
20
25
Number of Flights
5
10
15
20
Data: OpenFlights
Author: Jennifer Lin


Flights from Major Airports
Layout: Dh
TPA
IAD
FLL
EWR
BWI
DCA
CLT
MIA
CMH
ATL
BNA
PHL
DTW
BOS
LGA
SFO
SEA
JFK
IAH
ORD
MCO
DFW
MSP
LAS
DEN
SLC
LAX
Node Degree
5
10
15
20
25
Number of Flights
5
10
15
20
Data: OpenFlights
Author: Jennifer Lin



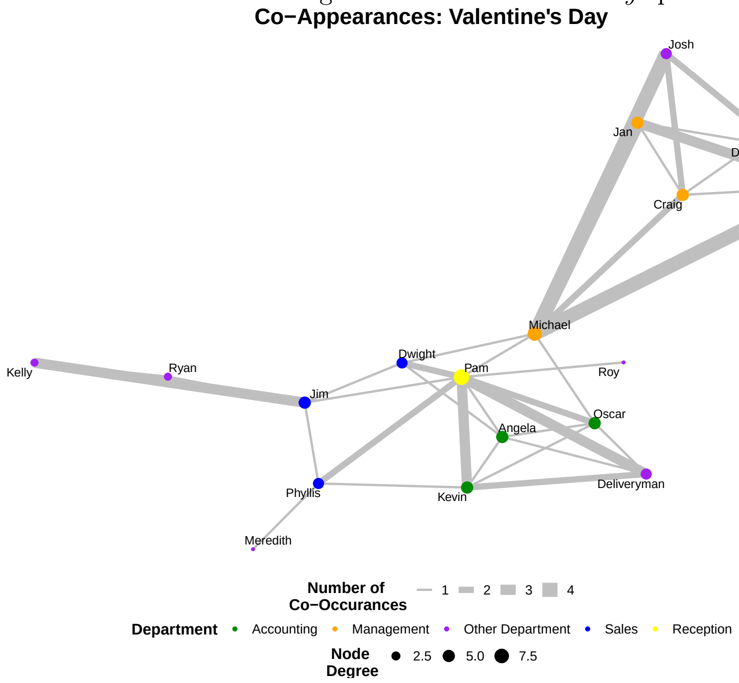


Exercises

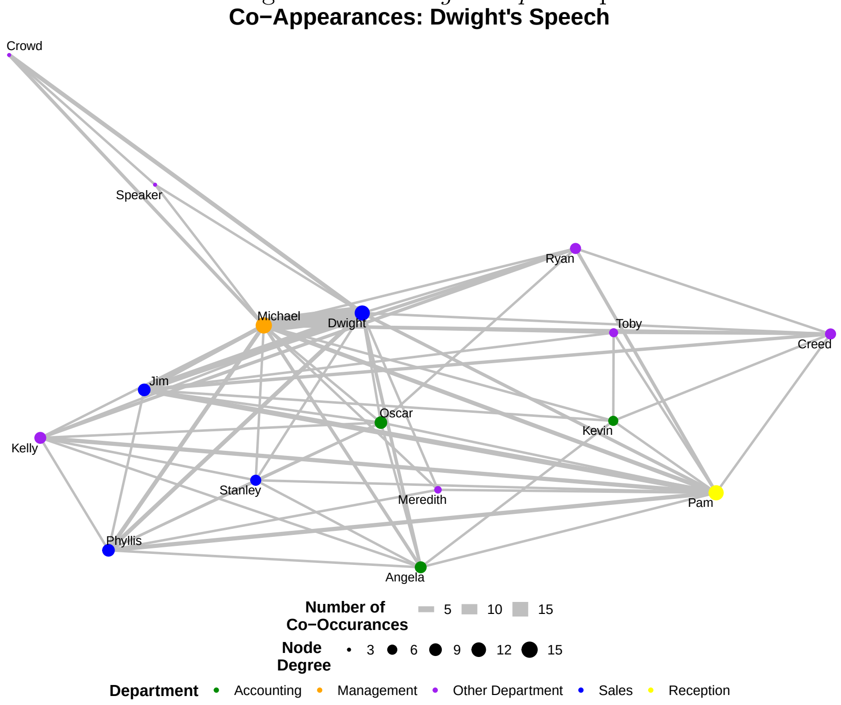
1. Read in the Edge Weight files for either *Valentine's Day* or *Dwight's Speech*
2. Reproduce the following networks (or generate something to a similar effect) for either episode
3. Revisit the Episode Descriptions that I include in the handout. Do

your results make sense given the context of the episode? Discuss. Why or Why not?

Here is the network that I got for the *Valentine's Day* episode



Here is the network I got for the *Dwight's Speech* episode.


Co−Appearances: Dwight's Speech
Crowd
Speaker
Ryan
Michael
Dwight
Toby
Creed
Jim
Oscar
Kevin
Kelly
Stanley
Meredith
Pam
Phyllis
Angela
Number of Co−Occurances
5
10
15
Node Degree
3
6
9
12
15
Department
Accounting
Management
Other Department
Sales
Reception