11. Social Networks

Self-Paced Workshop #11

Prof. Jack Reilly

Fall 2026

Drawing Relationships

Most data this semester has been one row per *thing*: a respondent, a country, a state. Network data, however, is inherently *relational*: it's about the relationships *between* things. This means we need more data: not just information about the things themselves, but also information about how they are related to each other.

This workshop builds the lecture's example step by step: the network of agreement among the nine justices of the Supreme Court in the 2020-2021 term, using the Harvard Law Review's statistics on how often each pair voted together. The tool is the `igraph` package.

The lecture's vocabulary, in brief:

- A network has **nodes** (the justices) joined by **edges** (agreement between two of them).
- Edges can be **directed** (I named you as a friend; did you name me?) or **undirected**, and **weighted** (some ties are stronger than others) or **unweighted** (all ties are the same). In our case, agreement is undirected — if Roberts agrees with Thomas 46% of the time, Thomas agrees with Roberts 46% of the time — and weighted, by that percentage.
- **Centrality** measures how important a node is to the network. *Degree* counts a node's links; *betweenness* counts how many shortest paths run through it. We will use a third, *eigenvector* centrality, which suits a weighted network like this one, and captures (conceptually) something like degree centrality, but with an extra emphasis on the centrality of not just you, but of your closest connections, as well.

Before you start

The boxes share one R session; run them top to bottom. `igraph` draws a network in a slightly different arrangement every time, so your graphs will not match the lecture's exactly, and will change if you run a box twice. That is expected, and Step 11 is how to stop it.

Part 1. Network Data

Step 1. Load the package and the data

Every network needs two datasets: one describing the **nodes**, and one describing the **edges**. Both are posted online, so we can read them straight from their web addresses:

```
#Load library
library("igraph")

#Load nodelist and edgelist as objects
nodes <- read.csv("https://jacklreilly.github.io/networkscc/supreme_nodes.csv", header=T, as.
links <- read.csv("https://jacklreilly.github.io/networkscc/supreme_edgelist.csv", header=T,
```

(In your own R, download the two files into your project's data folder and read them as `read.csv("supreme_nodes.csv", ...)`, as in the lecture.) Look at each one:

```
nodes
```

The **node list** is an ordinary-looking dataset, like anything else we have used: one row per justice, one column per attribute — name, abbreviation, chief or associate, the president who appointed them and that president's party, the year, and years of seniority. The `id` column is the important one.

```
links
```

The **edge list** has one row per edge. `from` and `to` are node `ids` — that is how the two datasets link up — and the other columns are attributes of the edge: here a `type` (agreement) and a `weight`, the share of cases the two justices agreed on. Because the network is undirected, each pair appears only once: there is a `j01`–`j02` row but no `j02`–`j01` row. A directed network would need both.[1]

Step 2. Make it a network

`graph_from_data_frame()` combines the two into one network object. `d` is the edge list, `vertices` the node list (*vertex* is another word for node), and `directed=F` says the ties run both ways:

```
#Convert to an igraph network. "d" identifies the edgelist; "vertices" identifies the nodelis
net <- graph_from_data_frame(d=links, vertices=nodes, directed=F)
net
```

The first line of the printout summarizes it: `UNWB` means **U**ndirected, **N**amed nodes, **W**eighted edges, followed by 9 nodes and 36 edges. (The **B**, for *bipartite* — a network with two kinds of node — is a false alarm: `igraph` reads any node attribute called `type` as marking one, and our node list happens to use `type` for Chief or Associate.) Then come the attributes, each marked `(v/...)` for a vertex attribute or `(e/...)` for an edge attribute.

[1] Nine justices make 9 × 8 / 2 = 36 pairs, and there are 36 rows. Every justice is linked to every other, because every pair agreed on at least some cases. That is why the *weights* carry all the information in this network.

The same network can also be written as an **adjacency matrix**: a row and a column for every node, with each cell holding the weight of the edge between them. `igraph` will produce one if you tell it which attribute to put in the cells:

```
#If we want to output an adjacency matrix, we can
#(we just need to identify the attribute to encode)
adjacency<-as_adjacency_matrix(net, attr="weight")
adjacency
```

Read across a row to see one justice's agreement with all the others; the dots are the diagonal, a justice with themselves. The matrix is symmetric because the network is undirected. Unlike an edge list, an adjacency matrix can hold only one attribute per edge.

Step 3. Working with the network

`igraph` has its own way of reaching inside a network. The lecture's table of the main commands:

Command	Usage
`E(net)`	Access the edges of the network
`V(net)`	Access the nodes of the network
`E(net)$weight`	Access a particular attribute (weight) of the edges of the network
`V(net)$seniority`	Access a particular attribute (seniority) of the nodes of the network
`V(net)[president=="Obama"]`	Returns all vertices with appointing president "Obama"
`E(net)[weight>.9]`	Returns all edges with weight greater than .9
`net[1,]`	Returns first row of network matrix
`net[5,7]`	Returns cell in the fifth row, seventh column
`plot.igraph(net)` or `plot(net)`	Plots the network (!)

Run them, one per line:

```
E(net)
V(net)
E(net)$weight
V(net)$seniority
V(net)[president=="Obama"]
E(net)[weight>.9]
net[1,]
net[5,7]
```

`E()` and `V()` work like `$` does for a data frame: they get you to the edges or the nodes, and `$` then picks an attribute. The square brackets *filter*, like `filter()` does: the only two Obama

appointees are `j05` and `j06` (Sotomayor and Kagan), and the only pair agreeing more than 90% of the time is `j01`–`j08`, Roberts and Kavanaugh. `net[5,7]` reads the adjacency matrix: Sotomayor and Gorsuch agreed 24.4% of the time.

Try it yourself: which edges have a weight *below* 0.2? Which justices were appointed before 2000?

```
E(net)[weight>.9]
```

Part 2. Drawing the Network

Step 4. The first plot

```
plot.igraph(net)
```

And . . . OK. It's there, but it's kind of uninspired. We might even call it ugly. What is wrong with it?

- The links are overplotted: 36 lines crossing in a small space, some so close they're hard to distinguish.
- The links are undifferentiated: a 93% agreement looks the same as a 13% one.
- The node labels are uninformative: `j01` tells a reader nothing.
- The colors carry no meaning.

In lecture, we set four goals for this graph — identify who the justices are, show the political factors outside the Court, show who is most central (and so likely decisive), and show agreement and disagreement — and a fix for each: **label** the nodes, **color** them by the party of the appointing president, **size** them by centrality, and do something with the **edges**, their thickness or color.

There are two ways to control how `igraph` draws a network. You can pass *parameters* to `plot.igraph()`, or you can write graphical attributes *into the network object*, where `plot.igraph()` will find them. If you do both, the parameters in the plot command win. (We will use both.)

Step 5. Label the nodes

The `vertex.label` parameter sets the labels, and `V(net)$justice` is the column of names:

```
plot.igraph(net,
  vertex.label=V(net)$justice,
  vertex.label.family="Helvetica")
```

The names are long for the circles. Abbreviations fit better, with the labels bold (`vertex.label.font=2`), black, and the circles larger:

```
plot.igraph(net,
   vertex.label=V(net)$abbrev,
   vertex.label.family="Helvetica",
   vertex.label.font=2,
   vertex.label.color="black",
   vertex.size=20)
```

Or keep the full names and make the circles big enough to hold them:

```
plot.igraph(net,
   vertex.label=V(net)$justice,
   vertex.label.family="Helvetica",
   vertex.label.font=2,
   vertex.label.color="black",
   vertex.size=48)
```

Every option for nodes starts `vertex.`, and every option for edges starts `edge.`; `?igraph.plotting` lists them all.

Step 6. Color by appointing party

Colors this time are written *into the network*, as a node attribute called `color`, which `plot.igraph()` looks for automatically:

```
#Generate colors based on party and write into network:
V(net)$color <- ifelse(V(net)$party=="Republican","coral1","cadetblue3")

plot.igraph(net,
            vertex.label=V(net)$justice,
            vertex.label.family="Helvetica",
            vertex.label.font=2,
            vertex.label.color="black",
            vertex.size=48)
```

`ifelse()` is base R's `if_else()`: coral for Republican appointees, blue for Democratic ones. Notice the plot command is the same as before: we no longer have to keep color in the graph code, because it is written into the network object itself.

Step 7. Size the nodes by centrality

Eigenvector centrality scores each node by how strongly it is tied to *other* well-connected nodes, so a justice who agrees a lot with justices who themselves agree a lot with everyone scores highest. Calculate it, write it into the network, and use it for the size:

```
V(net)$eigen<-eigen_centrality(net)$vector

plot.igraph(net,
   vertex.label=V(net)$justice,
```

```
   vertex.label.family="Helvetica",
   vertex.label.font=2,
   vertex.label.color="black",
   vertex.size=V(net)$eigen*50)
```

Older code, including some tutorials you will find online, calls this function `evcent()`. That name was deprecated in igraph 2.0.0, to make `igraph`'s function names consistent; it still runs, with a warning. Use `eigen_centrality()` in your own code.

The scores run from 0 to 1, with the most central justice at exactly 1, so `*50` turns them into circle sizes. Which justice is it?

```
round(V(net)$eigen, 3)
```

Kavanaugh, at 1.0, with Barrett and Roberts close behind — the Court's center in that term — and Sotomayor the least central.

Try it yourself: `degree(net)` gives each justice's degree centrality. Every justice has degree 8. Why, and why is degree therefore no help in this network?

```
degree(net)
```

Step 8. Mark the Chief Justice

While we are at it: give the Chief Justice a thicker outline, with another attribute written into the network:

```
V(net)$width<- ifelse(V(net)$type=="Chief",2,1)

plot.igraph(net,
   vertex.label=V(net)$justice,
   vertex.label.family="Helvetica",
   vertex.label.font=2,
   vertex.label.color="black",
   vertex.size=V(net)$eigen*50,
   vertex.frame.width=V(net)$width)
```

Step 9. Tie strength

Finally, the edges. Make each line's width proportional to its weight, so strong agreement draws a thick line and weak agreement a thin one:

```
plot.igraph(net,
   edge.width=E(net)$weight*5,
   vertex.label=V(net)$justice,
   vertex.label.family="Helvetica",
   vertex.label.font=2,
   vertex.label.color="black",
```

```
   vertex.size=V(net)$eigen*50,
   vertex.frame.width=V(net)$width)
```

All four goals are met: names, party, centrality, and agreement, in one graph.

Try it yourself: the weights only run from 0.125 to 0.927, so the thinnest lines are still fairly thick. Try `edge.width=E(net)$weight^2*8`, which exaggerates the difference.

```
plot.igraph(net,
   edge.width=E(net)$weight*5,
   vertex.label=V(net)$justice,
   vertex.label.family="Helvetica",
   vertex.label.font=2,
   vertex.label.color="black",
   vertex.size=V(net)$eigen*50,
   vertex.frame.width=V(net)$width)
```

Step 10. Layouts

Run any box above twice and the network rearranges itself. `igraph` generates layouts quasi-randomly within similar constraints, so each drawing is a little different. The `layout` option picks the method; the circle layout puts every node on a circle:

```
plot.igraph(net,
   edge.width=E(net)$weight*5,
   vertex.label=V(net)$justice,
   vertex.label.family="Helvetica",
   vertex.label.font=2,
   vertex.label.color="black",
   vertex.size=V(net)$eigen*50,
   vertex.frame.width=V(net)$width,
   layout=layout_in_circle)
```

The circle layout is the same every time, but it throws away the information the other layouts carry in where the nodes sit.

Try it yourself: try `layout=layout_as_star` and `layout=layout_on_grid`. Which tells you least about the Court?

Step 11. Locking in a layout

Better: compute a layout once, store it, and reuse it, so the graph looks the same every time. `layout_with_fr()` is the Fruchterman-Reingold method, which pulls strongly tied nodes together; giving it the weights makes high-agreement justices sit close to each other:

```
nice_layout <- layout_with_fr(net,
  weights=E(net)$weight)

plot.igraph(net,
```

```
   edge.width=E(net)$weight*5,
   vertex.label=V(net)$abbrev,
   vertex.label.family="Helvetica",
   vertex.label.font=2,
   vertex.label.color="black",
   vertex.size=V(net)$eigen*50,
   vertex.frame.width=V(net)$width,
   layout=nice_layout)
```

The first line computes the layout and stores it as `nice_layout`; the plot then uses it. Every later plot that says `layout=nice_layout` will put the justices in exactly these places. (Re-running *this* box gives a new arrangement, because its first line computes a fresh one.)

`nice_layout` is just a matrix — an x and a y coordinate for each justice — so you can also save it to a file, and read it back next time instead of computing a new one:

```
write.csv(nice_layout, "layout.csv", row.names = FALSE)
nice_layout <- as.matrix(read.csv("layout.csv"))
nice_layout
```

Where does `layout.csv` go? In RStudio or Positron, into your project folder, where it stays: next week's script can read it and draw the same picture. Here in the browser, R has its own temporary storage, which lasts only until you reload the page, so the file works for the rest of this session and then is gone. (In other words: nothing is written to your computer from the web browser.)

Draw the network with the layout read back from the file:

```
plot.igraph(net,
   edge.width=E(net)$weight*5,
   vertex.label=V(net)$abbrev,
   vertex.label.family="Helvetica",
   vertex.label.font=2,
   vertex.label.color="black",
   vertex.size=V(net)$eigen*50,
   vertex.frame.width=V(net)$width,
   layout=nice_layout)
```

The same picture as before. `row.names = FALSE` matters here.[2] Without it, `write.csv()` adds a column of row numbers, the matrix comes back with *three* columns, and `igraph` quietly uses the row numbers as the horizontal positions. There is no error, and the result still looks like a network, which is the danger: the justices are simply spaced evenly from left to right in the order of the node list, and the picture no longer matches the one you saved.

[2]The lecture slide leaves `row.names = FALSE` out. To see what happens, delete it from the `write.csv()` box, run that box, then run the plot box after it. Put it back and run both again to recover.

Try it yourself: re-run the `layout_with_fr()` box until you get an arrangement you like better, then run the `write.csv()` box and the plot box to save it and check it.

Step 12. Highlighting groups

`mark.groups` draws a shaded shape around sets of nodes. Here, the Trump, Obama, and Bush Jr. appointees, by their position in the node list (justices 7, 8 and 9; 5 and 6; 1 and 4):

```
plot.igraph(net,
   mark.groups=list(c(7,8,9), c(5,6), c(1,4)),
   mark.col=c("#FFE4E1","#C5E5E7", "#ECD89A"),
   mark.border=NA,
   edge.width=E(net)$weight*5,
   vertex.label=V(net)$justice,
   vertex.label.family="Helvetica",
   vertex.label.font=2,
   vertex.label.color="black",
   vertex.size=V(net)$eigen*50,
   vertex.frame.width=V(net)$width)
```

The colors are given as *hex codes*: `#` and then two digits each for red, green, and blue. That is the most precise way to name a color in R, and every color picker will give you one.

Try it yourself: add `layout=nice_layout` so the highlighted graph uses your locked-in layout.

Step 13. More than one way: a heatmap

Don't let yourself be caught thinking you must draw a network with a network graph. The adjacency matrix from Step 2 is a table of numbers, and a heatmap colors each cell by its value:

```
palf <-
colorRampPalette(
c("white", "darkgreen"))
netm <- as_adjacency_matrix(net,
attr="weight", sparse=F)
colnames(netm) <- V(net)$justice
rownames(netm) <- V(net)$justice
heatmap(netm, Rowv = NA,
Colv = NA, col = palf(100),
 scale="none", margins=c(10,10) )
```

`colorRampPalette()` makes a function that blends from white to dark green; `palf(100)` asks it for 100 shades. `sparse=F` asks for an ordinary matrix rather than the compact one Step 2 printed. As with `evcent()`, older code may use `get.adjacency()`, the deprecated name of `as_adjacency_matrix()`. The heatmap is symmetric, because the network is undirected; dark cells are pairs that agree often, and the diagonal is white because a justice is not linked to themselves.

So What Can We See?

A strong graphic presents information quickly, at a glance, and rewards a closer look with more detail. The finished network from Steps 9 and 11 shows:

- **Centrality**, twice: by node size, and by position, since the layout pulls together justices who agree.
- **Party**, by color.
- **Tie strength**, by line width.
- **Identity**, by the labels.

The Supreme Court is a nice network to begin with: small and simple. The same techniques work for larger networks, which bring their own problems — nodes get smaller or disappear, the focus moves to overall structure, and it gets easy to draw a "hairball" of lines that shows nothing.

Work on Your Own

Problem Set 9 asks you to draw a network of your own: a node list, an edge list, and the steps on this page. These are the basics.

For more, Ognyanova's network visualization tutorial, the core reading for this week and a source for the lecture, goes much further with the same `igraph` commands. If you prefer the `ggplot2` way of building a graph, the `ggraph` package draws `igraph` networks with layers like Workshop 7's. `igraph` predates `ggraph` substantially, though, so many people who work with networks still haven't quite moved over.