Lecture 11: Networks II (Counting on Graphs), Causality and Experiments
Modeling Social Data, Spring 2019
Columbia University

Deniz Ulcay

April 12, 2019

1 Introduction

In this lecture, we covered the following key topics:

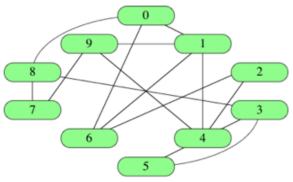
- Network Representations (Data Structures)
- Network Topologies and Watts-Strogatz Networks
- Describing Networks
- Causality and (Natural) Experiments

2 Network Representations (Data Structures)

We discussed three different data structures that can be used to represent networks: Edge list, adjacency matrix and adjacency list.

2.1 Edge List

One way to represent a network is a list of $|E|$ edges, called an edge list. Each edge is represented as a tuple of the two vertices the edge is incident on. Even though edge lists are simple to store, performing computations on them can be difficult. The space complexity of an edge list is $\Theta(E)$. If we wanted to check whether an edge exists between two vertices, we would need to search the entire list, which would take $O(E)$ time.



[[0,1], [0,6], [0,8], [1,4], [1,6],
[1,9], [2,4], [2,6], [3,4], [3,5],
[3,8], [4,5], [4,9], [7,8], [7,9]]

Simple for storage, but difficult
to compute with

Figure 1: A network with 10 vertices (left) and the edge list representation of its edges (right).

2.2 Adjacency Matrix

The second approach involves building a $|V|x|V|$ matrix of 1s and 0s for a network with $|V|$ vertices where row i, column j is 1 if there exists an edge between vertices i and j. The space complexity of an adjacency matrix is $\Theta(V^2)$. Thus, if our network was relatively sparse, the matrix would mostly consist of 0s and take up as much space as a dense network to represent. Yet, checking whether an edge exists between vertices i and j can be performed in constant time $O(1)$.

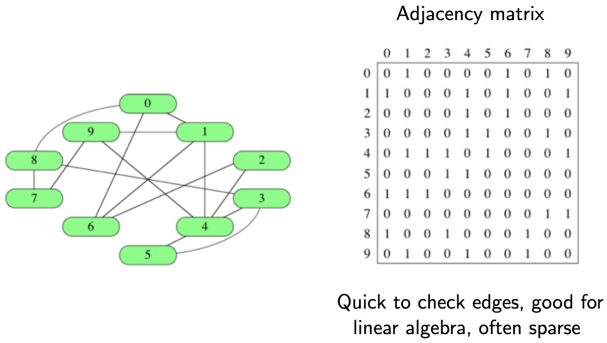


Figure 2: A network with 10 vertices (left) and the adjacency matrix representation of its edges (right).

2.3 Adjacency List

The last data structure we covered, adjacency list is a combination of edge list and adjacency matrix. An adjacency list of a network with $|V|$ vertices has length $|V|$ where each element i is a list of vertices adjacent to vertex i. The space complexity of an adjacency matrix is $\Theta(E)$ where $|E|$ is the number of edges in the network. Checking whether a vertex i has an edge to vertex j takes $O(d)$ time where d is the degree of the vertex i.

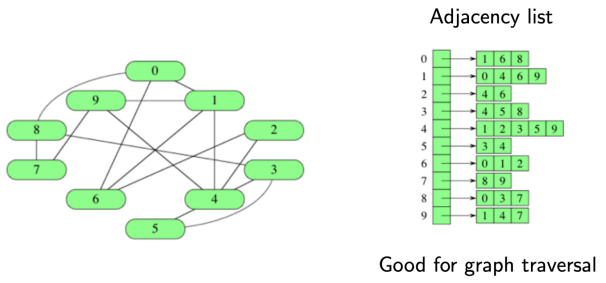


Figure 3: A network with 10 vertices (left) and the adjacency list representation of its edges (right).

Note: See Networks.Rmd for the representation of some toy and real networks as various data structures using igraph.

3 Network Topologies and Watts-Strogatz Networks

While going over the capabilities of igraph, an R library (see Networks.Rmd), we learned about several network topologies (Figures 4, 5 and 6) and generated some small world networks using a random graph generation model called the Watts-Strogatz model (Figures 7, 8, and 9).

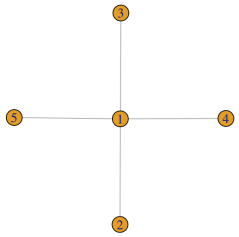


Figure 4: Star Network

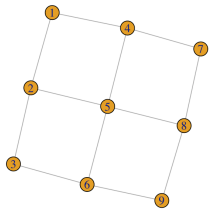


Figure 5: Lattice Network

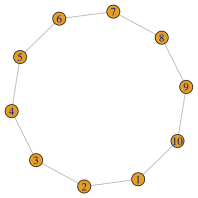


Figure 6: Ring Network

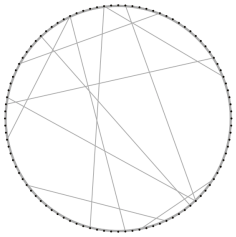


Figure 7: Mostly a ring

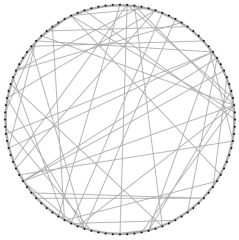


Figure 8: Some rewiring

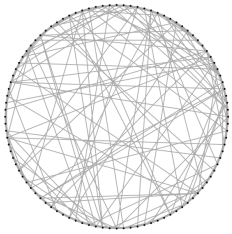


Figure 9: Lots of rewiring

4 Describing Networks

Higher the dimension and complexity of a network, harder it gets to visualize. We discussed four different statistics for describing and analyzing a network, and algorithms one can use to calculate these statistics:

- Degree: How many connections does a node have?
  - Degree distributions
- Path Length: What's the shortest path between two nodes?
  - Breadth first search
- Clustering: How many friends of friends are also friends?
  - Triangle counting
- Components: How many disconnected parts does the graph have?
  - Connected components

4.1 Degree Distributions

The degree of a node in a network is the number of connections it has to other nodes. The degree distribution of a network is the probability distribution of these degrees across the entire network. Working with some real life networks, we plotted some degree distributions and learned ways to make these plots more readable (See Networks.Rmd). Below are the code for and the visualization of the out-degree distribution (edges coming out of nodes) of the Wikipedia voting network and its log visualization:

```
wiki_degree_dist <- wiki_edges %>%
  group_by(src) %>%
  summarize(degree=n()) %>%
  group_by(degree) %>%
  summarize(num_nodes=n())
```

Figure 10: R code for out-degree distribution

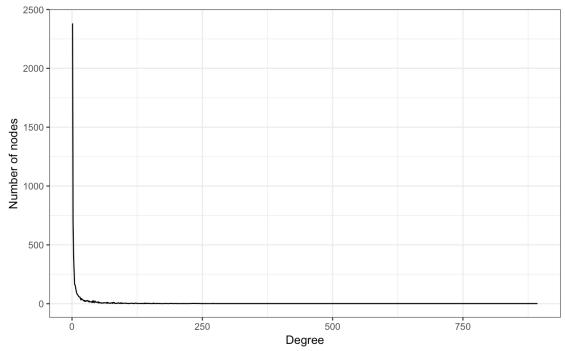


Figure 11: Visualization of Out-Degree Distribution

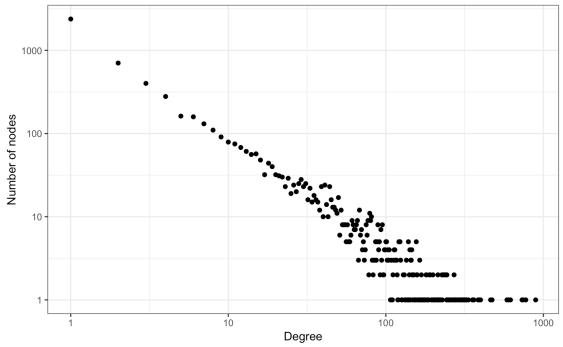


Figure 12: Visualization of Out-Degree Log Distribution

4.2 Path Length

Path length is the shortest distance between two nodes of a network. The path length distribution of a network is the distribution of the shortest paths between all node pairs in the network (See Network.Rmd for igraph way of calculating path lengths of a network and its visualization). The mean and median path lengths of a network are useful ways to analyze and understand a network. To calculate single-source shortest paths to all other nodes of a network, we can use an algorithm called Breadth First Search:

Breadth First Search

Initialize all node distances to infinitely far
Set source node distance to zero
Initialize current boundary as source node

While current boundary is not empty: {
Create empty list for new boundary
For each node in current boundary: {
loop over undiscovered neighbors and set neighbor distance as current distance+1 and add neighbor to the next boundary
}
Set current boundary = next boundary
}

Note: See counting_on_networks.Rmd for BFS R code.

4.3 Connected Components

In a network, a connected component is a subgraph in which any two nodes are connected to each other by paths, and which is connected to no additional nodes in the overall graph. The algorithm to determine the connected components in a network uses BFS and is therefore fairly straightforward.

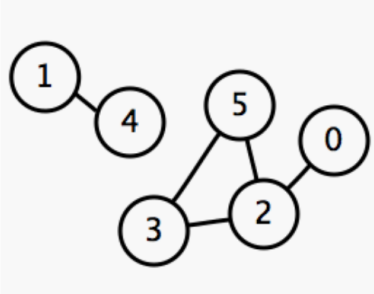


Figure 13: A network with two connected components.

Connected Components Algorithm

(1) Choose any node and perform BFS, labeling all the other nodes accessible from the chosen source node as component x.
(2) Update label, choose any unlabeled node in the network and repeat (1) until all nodes are labeled.

4.4 Clustering

Clustering is the task of grouping a set of objects in such a way that objects in the same group are more similar to each other than to those in other groups. One way to think about it is in terms of mutual friends (connected nodes). How many friends of friends are also friends? For this end, we covered an algorithm referred to as triangle counting (See counting_on_networks.Rmd).

Counting Triangles

Initialize counter for the number of triangles at each node
Loop over each node: {
Get the list of friends for node
Add a count of 1 for each pair of node's friends that are connected
}

5 Causality and (Natural) Experiments

We started the new topic with a discussion on the difference between 'Prediction' and 'Causation'. We concluded that while 'Prediction' focuses on observing an environment and forecasting what is going to happen based on observation, 'Causation' focuses on changing the environment and anticipating the effects.

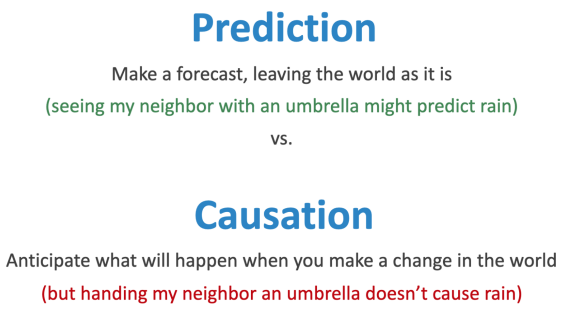


Figure 14: Prediction versus Causation

Then we looked at the difference between 'Reverse Causal Inference' (causes of effects) and 'Forward Causal Inference' (effects of causes). While 'Reverse Causal Inference' is questioning 'What caused Y?', 'Forward Causal Inference' is asking 'What are the effects of X?'. In general, 'Forward Causal Inference' is easier than 'Reverse Causal Inference'.

Examples of 'Reverse Causal Inference':

- What makes an email spam?
- What caused my kid to get sick?
- Why did the stock market drop?

Examples of 'Forward Causal Inference':

- How does education impact future earnings?
- What is the effect of advertising on sales?
- How does hospitalization effect health?

We then took a closer look at the last example, 'How does hospitalization effect health?':

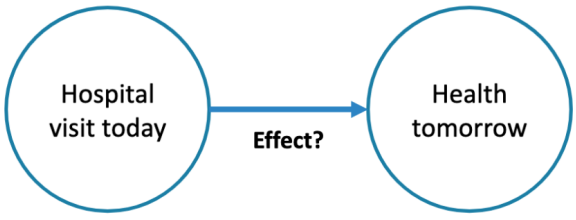


Figure 15: Arrow means 'X causes Y'

As we discussed in class, this representation of the cause-effect relationship between going to the hospital today and one's health tomorrow can lead to a problematic conclusion: Going to the hospital makes you sick. In such a case, regardless of how many observations one might have, the conclusions one draws will be biased. Thus, this relationship cannot be explained independently of some other common factor.

Confounds

The effect and cause might be confounded by a common cause and be changing together as a result. In fact, there could be many confounds. Going back to our hospital/health example, it is easy to recognize that visiting the hospital today and one's health tomorrow both depend on another cause, one's health today. If one only observes the cause and effect changing together, one cannot estimate the effect of hospitalization alone on health tomorrow.

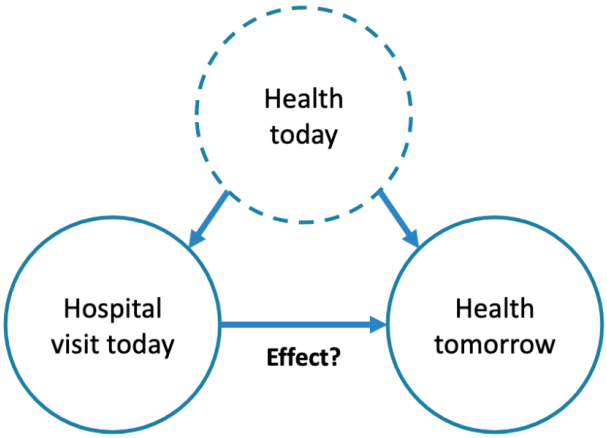


Figure 16: Dashed circle means 'unobserved'.

Observational Estimates

Let us say all sick people in our dataset went to the hospital today and healthy people stayed at home. The observed difference in health tomorrow is:

$$\Delta_{obs} = [(\text{Sick and went to hospital}) - (\text{Sick if stayed home})] + [(\text{Sick if stayed home}) - (\text{Healthy and stayed home})]$$

Figure 17: Here we are trying to compare something that did happen with something that could have happened. Unfortunately, we cannot clone people and make their clones stay at home.

Selection Bias

Selection bias is the baseline difference between those who opted in to the treatment and those who did not.

$$\Delta_{obs} = \overbrace{[(\text{Sick and went to hospital}) - (\text{Sick if stayed home})]}^{\text{Causal effect}} + \underbrace{[(\text{Sick if stayed home}) - (\text{Healthy and stayed home})]}_{\text{Selection bias}}$$

Figure 18: Selection bias is likely negative here, making the observed difference an underestimate of the causal effect.

Simpson's Paradox

Selection bias can be so large that observational and causal estimates give opposite effects (e.g. going to the hospital makes you sick). This is called the Simpson's Paradox.

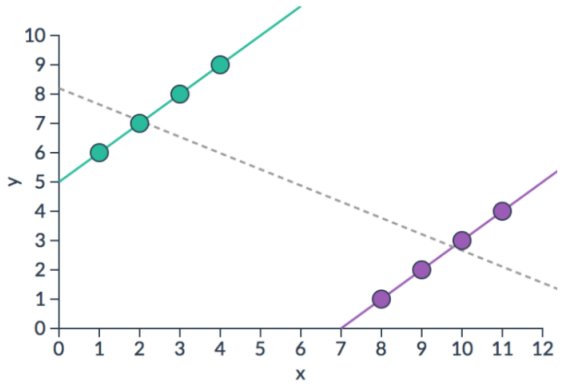


Figure 19: Simpson's Paradox

Controlled Experiments

Counterfactuals

To isolate the causal effect, we have to change one and only one thing (hospital visits) and compare the outcomes. Unfortunately, we never get to observe what would have happened if we did something else so we have to estimate it.

Figure 20: Reality versus Counterfactual

Random Assignment

We cannot clone people but by randomly assigning their actions with a coin flip, we can get results almost as good as cloning. In the hospital example, we can use randomization to create two groups that differ only in which treatment they receive. This allows us to restore symmetry.

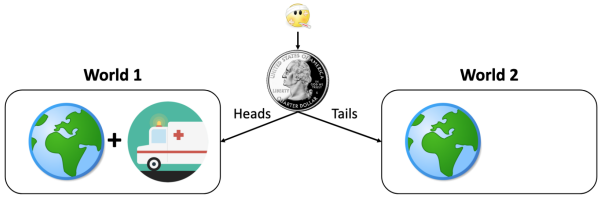


Figure 21: Randomly assign both sick and healthy people to which treatment they will receive.

Figure 22: The resulting worlds closely resemble the reality versus counterfactual separation we desire.