Hongtao Hao

Notes on Network Science

Contents

List of Tables

List of Figures

Welcome

This online book is largely inspired by Professor Yong-Yeol ("YY") Ahn[1]'s online class of INFO I606 *Network Science*[2] at Indiana University Bloomington, and the *Network Science*[3] book by Albert-László Barabási[4]. I am deeply grateful to Professor YY Ahn for allowing me to attend the online course and to Professor Barabási for generously making his book available to everyone online. Without their help, this project is impossible.

This book has several custom blocks:

This is a note, a general notice.

This is a reminder, usually just to remind myself of something I do not understand yet.

This is a tip, which helps you to understand a concept or memorize a formula.

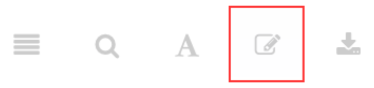
This is a caution, to inform you, for example, an error in publications.

This is an importance note. It tells you something you should consider when understanding a concept, for example, when scholars disagree.

[1] https://yongyeol.com/
[2] http://yongyeol.com/teaching/2020SP_netsci_syllabus.pdf
[3] http://networksciencebook.com/
[4] https://barabasi.com/

This website is built with Bookdown[5] and deployed by Netlify[6]. You can find the full source of this project at https://github.com/hongtaoh/netsci-notes.

If you notice any errors, including typos and mistakes in math formulas, you are welcome to contribute to the project by clicking the "Editing" button on the upper-left of this online book (shown below). You will be directed to fork this project on GitHub before you and suggest changes.



[5]https://bookdown.org/
[6]https://www.netlify.com/
[7]http://creativecommons.org/licenses/by-nc-sa/4.0/

1

Introduction to Network Science

We are going to dive into the field of network science. To learn a new subject, it's always beneficial to learn a little bit about it's history, definition, and significance. These will be the focus our this Chapter.

1.1 Intro to networks

The following is not originally from me, but my notes from reading *Network Science*[1] (Ch. 1.1-1.4) by Albert-László Barabási[2]. Notes are accompanied by my explanations.

1.1.1 Networks and complex systems

A system is a **complex system** when we cannot know its collective behavior by simply understanding each individual component within it. To really understand complex systems, we need to gain a deeper understanding of **networks** behind these syetems.

Despite that networks are different in size, nature, function, etc., they are orgnized by a common set of rules. That's why we are able to study them.

1.1.2 Driving forces behind network science's emerging as a discipline

Networks are not new. They have been studied by scientists in other disciplines for centuries. But why did network science become a discipline in the 21st centuary? There are two driving forces behind it:

1. **The emergence of network maps**

Mapping interactions in networks requires we keep track of huge amounts of data. This was not readily available until the start of the 21st century.

2. **The universality of network characteristics**

In the past, people knew different networks, but they didn't realize that as different as these networks are, they are indeed similar and share the same principles.

1.1.3 Key characteristics of network science

Network science has the following key characteristics:

- **Interdisciplinary**

Scientists in different disciplines working with networks have their own problems to solve and tasks to perform. However, some of their tasks share similarities, and a solution found in discipline A might turn out to be useful and important for discipline B as well.

- **Empirical, & data driven**

Network science is empirical in nature. Every tool developed by network scientists should pass the test of data in real life and should offer people deeper insights about the network.

- **Quantitative and mathematical**

Network science has been borrowing the fromalism from graph theory[3], the conceptual framework from statistical physics, and concepts from engineering.

- **Computational**

Scientists of network science need to deal with a vast amount of data. They'll need to apply computational methods.

[3] https://en.wikipedia.org/wiki/Graph_theory

1.2 Definition and significance of networks

The following came from the online course of INFO 606[4] at Indiana University taught by Professor Yong-Yoel ("YY") Ahn[5].

1.2.1 What is a network

A network is **a system of elements that are connected**. Those elements are called **nodes** or **vertices**, and the connections are called **links** or **edges**. Network science is all about **connections**.

There are two things that we care about in connections in a network: **structure**, and **dynamics**. Structure and dynamics might affect each other. Professor YY took Facebook activity as an example. Interpersonal interactions on and through facebook are governed by the structure of your friends you have on FB; On the other hand, if someone keeps posting things that iritate you, you might unfriend him or her, which will alter the structure of your online friends network.

I thought about international relationships. Relationships between countries vary: some are strong while others are weak. These structures definitely have an effect on the things going on between countries. Then, if a country does something that make another country, or other countries angry, for example, initiating a war, the original structure might witness drastic changes.

1.2.2 Why should we care about networks?

1.2.2.1 Understanding individual elements is far from enough.

Knowing about each neuron in a person's brain cannot gaurantee you can understand how a brain functions, or even rebuild a brain. To do these, we need to how elements interact.

1.2.2.2 Networks are everywhere.

Human body, Internet, electricity, food, forest, the universe, etc., all have networks in them.

Networks are omnipresent. Some networks are waiting for discovery and finding a new network itself can be a contribution. For example, professor YY and his colleagues found this flavor network[6].

Also, the fact that networks are everything mean they have huge influence on our lives. A prime example would be pandemics. A contagious disease won't stay locally anymore, thanks to the advancing global air transportation. The Coronavirus Disease 2019 (COVID-19) was first found in China and in only several months, it became a nightmare for every country in the world.

1.2.2.3 Networks let us make fun analogies.

Network thinking: everything is a system consisting of many individual elements that are connected and interact.

This thought help scientists make analogies and new discoveries. For example, in the 19th centuary, it led to probabilistic and statistical thinking.

Professor YY gave an example of this own. He was studying how hastags spread on Twitter. The goal was to predic viral hastags. Later, some other scientists applied this finding to human brain and published this paper[7].

1.2.3 Think for yourself

Can you give an example of a network in your life?

1.3 Python setup

The following is based on Professor YY's tutorial on Python setup[8].

Throughout this tutorial, we'll use Python[9]. Specifically, we'll use the package called Networkx[10]. Networkx is useful and handy in terms of calculating properties of networks but isn't always helpful for creating visualizations. To visualize networks, we'll use Gephi[11].

[6]https://www.nature.com/articles/srep00196
[7]https://www.sciencedirect.com/science/article/pii/S0896627315004742
[9]https://www.python.org/
[10]https://networkx.github.io/documentation/stable/index.html
[11]https://gephi.org/

We'll be using Anaconda[12] to manage python packages. Jupyter Notebook[13] will be where we write and run our python codes.

Don't worry if you don't know anything about any of them. I'll walk you through them now.

1.3.0.1 Anaconda

You can refer to this video[14] to get a better understanding of why and how to use Anaconda.

To install Anaconda, go here[15] and choose the version for you. For MacOs, choose the "Command Line Installer" **only if you want to feel like a hacker**.

1.3.0.1.1 If you don't have enough storage

The full Anaconda takes more than 3G of your storage. If storage is a problem for you, you can install the lightweight version of it: Miniconda[16]. You can look here[17] to know more about the differences between the two. To put it simply, Anaconda comes with a long list of packages[18] pre-installed, so you don't have to install each of them separately by yourself. However, you'll have to do so for Miniconda.

If you decide to use Miniconda for whatever reason, after installing it, open your `Terminal` and do the following (If you are using Anaconda, you don't need to). I am assuming that you are using a Mac:

```
conda --version # To check whether the installation was successful
```

If the installation was successful, then do the following:

```
# To download jupyter notebook, and jupyterlab:
conda install -c conda-forge vega notebook jupyterlab
# To download packages we'll need to use later:
conda install numpy scipy networkx jupyter
conda install pandas matplotlib seaborn bokeh scikit-learn
```

[12] https://www.anaconda.com/products/individual
[13] https://jupyter.org/install.html
[14] https://www.youtube.com/watch?v=YJC6ldI3hWk
[15] https://www.anaconda.com/products/individual#Downloads
[16] https://docs.conda.io/en/latest/miniconda.html
[17] https://stackoverflow.com/questions/45421163/anaconda-vs-miniconda
[18] https://docs.anaconda.com/anaconda/packages/py3.8_osx-64/

Even if you are using Anaconda, sometimes you might need to use some packages that are not already installed and you'll need to install them by yourself. Simply run `conda install your-package-name`. Most of the packages could be installed that way. Of course, you can still use `pip` to download packages even if you are using Anaconda (or Miniconda). Run `pip your-package-name`.

1.3.0.1.2 If you don't want to use either Anaconda or Miniconda

If your computer is running on a MacOS or a Linux system, then you can also use `pip` to install python packages. First, you need to download Python here[19]. Then, go to `Terminal` and run the following code, which came from Professor YY's tutorial[20].

```
pip3 install numpy scipy networkx jupyter jupyterlab ipython
pip3 install pandas matplotlib seaborn bokeh scikit-learn
```

1.3.0.2 Jupyter Notebook

As Professor YY suggests, to gain a deeper understanding of Jupyter Notebook, you can watch this tutorial[21] and this demonstration[22].

As the above two videos showed, to open jupyter notebook, you simply need to type `jupyter notebook` in your `Terminal`. It's quite straightforward. One trick you might need to know is that you can first go to the directory of your file, for example, `cd Desktop/netsci` and then run `jupyter notebook`. This way, you can use jupyter notebook right in the folder you want. This might save some of your time.

1.3.0.2.1 Jupyter Notebook On the cloud

If you don't want to use jupyter notebook on your computer, you can try Google's Colab[23] which allows you to run python codes on the cloud.

[19] https://www.python.org/downloads/
[20] https://github.com/yy/netsci-course/blob/master/m01-getready/python-setup.md
[21] https://www.youtube.com/watch?v=HW29067qVWk
[22] https://www.youtube.com/watch?v=2eCHD6f_phE
[23] https://colab.research.google.com/notebooks/intro.ipynb

1.3.0.3 Most importantly, Python

I haven't talked about Python, the most important tool we are going to use in this tutorial yet. I don't recommend you to attend a detailed python course now, since it's not necessary for this tutorial.

To get started with Python, read these two instructions:

1. An Informal Introduction to Python[24]
2. More Control Flow Tools[25]

Do expect one, two, or even more days on these if you don't have any experience using Python before. Make sure that you understand most of the codes in the above two instructions before you begin. Open your `jupyter notebook` and try to implement the codes in the above two tutorials now.

1.4 Homework

The following is based on Professor YY's Python Introduction Assignment[26].

Create a notebook, name it as `python-setup`, and run the following codes:

```
import numpy as np
import networks as nx
import matplotlib as plt
% matplotlib inline
```

If you have successfully downloaded Anaconda, you won't see any errors. If you do, go back and read the above instructions more carefully. Chances are that you might have missed some steps. Use Google or other search engines to look for answers to questions not convered here.

Thanks.

[24] https://docs.python.org/3.7/tutorial/introduction.html
[25] https://docs.python.org/3.7/tutorial/controlflow.html

2 Graph Theory and Some Basic Concepts

The following is not originally from me, but my notes from reading *Network Science*[1] (Ch. 2.1-2.3) by Albert-László Barabási[2]. Notes are accompanied by my explanations.

What is a network composed of? How is a network different from another one? In this chapter, we are going to study some very basic concepts of network science: size of a network, number of links, degree, average degree, degree distribution, and adjacency matrix.

2.1 Königsberg bridge problem (Ch. 2.1)

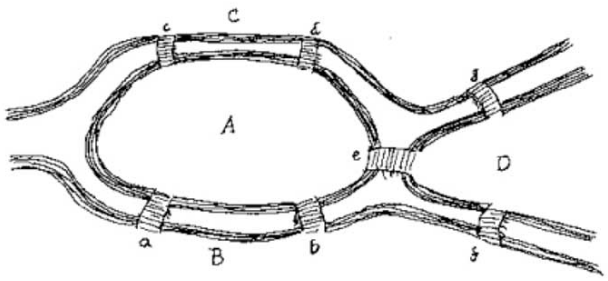


FIGURE 2.1: Leonhard Euler's drawing of the seven bridges in Königsberg, Image credit: beanz Magazine

The above image is from *beanz Magazine*[3].

The challenge: Are you able to wall across all the seven bridges but cross each bridge only once? The starting and ending point is not specified.

[3]https://www.kidscodecs.com/7-bridges-konigsberg/

Before reading on, I highly recommend you to try this exercise[4] on mathsisfun.com[5] by yourself. Getting the answer from others won't help you learn it at the deepest level.

Solution to this challenge: if a path exists which allows a person to cross all the bridges but walk across each bridge only once, which is called a "**Euler Path**", the number of nodes having an odd number of bridges should be either zero or two. When there are two nodes having an odd number of bridges, i.e., having an odd degree (a point we'll learn later), one of the two nodes should be the starting point and the other the end point.

This remarkable discovery by Euler led to the development of a new branch in mathematics: graph theory[6].

2.2 Networks and graphs (Ch. 2.2)

The components in a network are called **nodes** or **vertices** and the direct connections between them are called **links** or **edges**. When we are describing a network in real life, it's better to use **networks**, **nodes**, and **links**, whereas **graphs**, **vertices**, and **edges** are mostly used for an abstract representation of networks. Some people do not care about these distinctions, though.

$\boldsymbol{N}$, meaning the **number of nodes**, is also called the **size of the network**. Nodes are labeled as $\boldsymbol{i}$ = 1,2,3,..., N

$\boldsymbol{L}$, representing the **total number of links** in a network.

For an individual link, we may represent it as (i, j). For example, (2,4) denotes the link connecting nodes 2 and 4.

Links can either be **directed** or **undirected**. If I make a phone call to you, then the link between us is from me to you. It is **directed**. If we are friends on social media, then this link is **undirected**.

A network is **directed** if all of its links are **directed**. If all of its links are **undirected**, then this network is an **undirected** network.

[4] https://www.mathsisfun.com/activity/seven-bridges-konigsberg.html
[5] https://www.mathsisfun.com/
[6] https://en.wikipedia.org/wiki/Graph_theory

2.3 Degree, average degree, and degree distribution (Ch. 2.3)

2.3.1 Degree

The **degree** of a node is simply the number of links connecting it to other nodes.

```
library(igraph)
par(mar = c(1, 1, 1, 1))
g1 <- graph( edges=c(1,2, 1,3, 2,3, 3,4), n=4, directed=FALSE)
plot(g1)
```

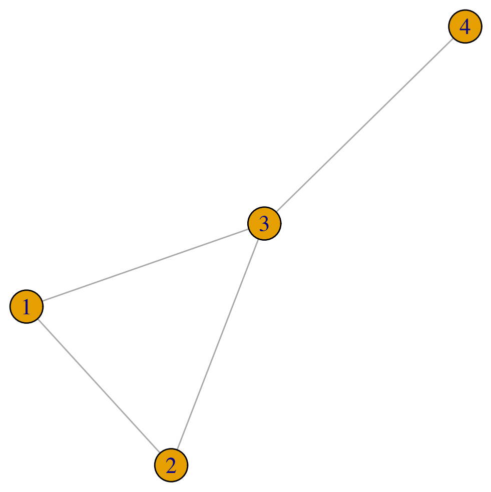


FIGURE 2.2: An undirected network for illustration.

For example, in the above network, nodes 1, 2, and 4 have a degree of two. Node 3 has a degree of three.

For the i^{th} node in a network, we'll denote its degree as k_i. Therefore, for the above network, k_1=k_2=k_4=2, k_3=3.

As mentioned above, the **total number of links** is denoted as $\boldsymbol{L}$. In an undirected network, it is easy to understand that $\boldsymbol{L}$ should be half of the sum of all the node degrees. It should be halved because each link belongs to two nodes and therefore each is counted twice. We have:

$$L = \frac{1}{2}\sum_{i=1}^{N} k_i \tag{2.1}$$

In the above network, $L = 4$.

2.3.2 Average degree

Average degree, denoted as $\langle k \rangle$ is simply the mean of all the node degrees in a network. For the network above (Figure 2.2), $\langle k \rangle = \frac{1}{4} \cdot (k_1 + k_2 + k_3 + k_4) = \frac{1}{4} \cdot (2+2+3+2) = 2.25$. This means that on average, each node in the network has 2.25 links.

According to its definition, we know that $\langle k \rangle = \frac{1}{N}\sum_{i=1}^{N} k_i$. Combined with Eq. (2.1), we have:

$$\langle k \rangle = \frac{2L}{N} \tag{2.2}$$

How can we understand this equation?

I would understand it this way: we are calculating the **average** degree, so the denominator will be $\boldsymbol{N}$, i.e., number of nodes, and the nominator will be the sum of all the nodes' number of links, which should be $2L$ because each link is shared by two nodes.

The above equation Eq. (2.2) is for undirected networks. What about directed networks? Should we calculate their average degree the same way?

```
g2 <- graph( edges=c(1,2, 1,3, 2,3, 3,4), n=4, directed=TRUE)
par(mar = c(1, 1, 1, 1))
plot(g2)
```

Each link in the above network (2.3) is directed. If we calculate the network's average degree according to Eq. (2.2), we will be losing information.

Therefore, we will distinguish between **incoming degree**, denoted as k_i^{in} and **outgoing degree**, denoted as k_i^{out}. k_i^{in} means the number of links from other nodes pointing to node i, and k_i^{out} means the number of links starting from node i and pointing to other nodes.

For a given node i in a directed network, its degree is the sum of incoming degree and outgoing degree. Therefore,

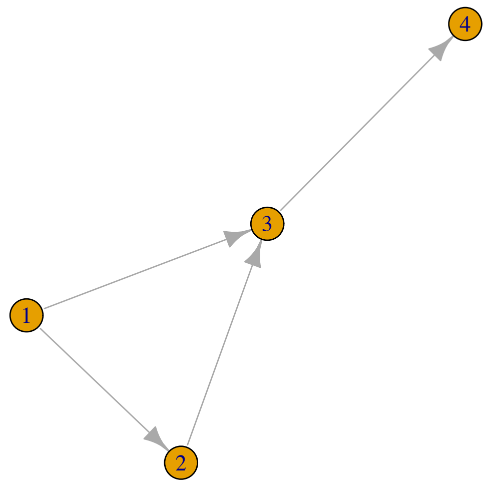


FIGURE 2.3: A directed network for illustration.

$$k_i = k_i^{in} + k_i^{out} \tag{2.3}$$

And $\boldsymbol{L}$, the total number of links in a directed network, is:

$$L = \sum_{i=1}^{N} k_i^{in} = \sum_{i=1}^{N} k_i^{out} \tag{2.4}$$

For a directed link between node i and node j, i.e., $(i,\ j)$, it constitutes an **incoming degree** for one node, but an **outgoing degree** for the other. For example, in the network (2.3), (1,2) counts as an **incoming degree** for node 2, but an **outgoing degree** for node 1.

In fact, if we combine Eq. (2.3) with Eq. (2.4), we will know that Eq. (2.4) is equal to Eq. (2.1). Then why can't we stick to Eq. (2.1) even for a directed network? This is because, I guess, we can have more information, i.e., incoming or outgoing degree, by using Eq. (2.4).

Last, what's the average degree for a directed network?

We can definitely use Eq. (2.2), but, again, we will be missing valuable information. Therefore, we will distinguish between $\langle k_{in} \rangle$ and $\langle k_{out} \rangle$, the two of which, in fact, are equal:

$$\begin{aligned}\langle k_{in} \rangle &= \frac{1}{N}\sum_{i=1}^{N} k_i^{in} \\ &= \langle k_{out} \rangle \\ &= \frac{1}{N}\sum_{i=1}^{N} k_i^{out} \\ &= \frac{L}{N}\end{aligned} \tag{2.5}$$

2.3.3 Degree distribution

In a large network, nodes' degrees vary. For example, there might be 100 nodes with a degree of 10, 50 nodes with a degree of 9, 30 nodes with a degree of 7, etc.. Then what is the probability that a randomly picked node will have a degree of k?

Let's denote this probability as p_k, and call it as **degree distribution**, which, as is described above, is defined as "the probability that a randomly picked node in a network has a degree of k". According to this definition, we know that:

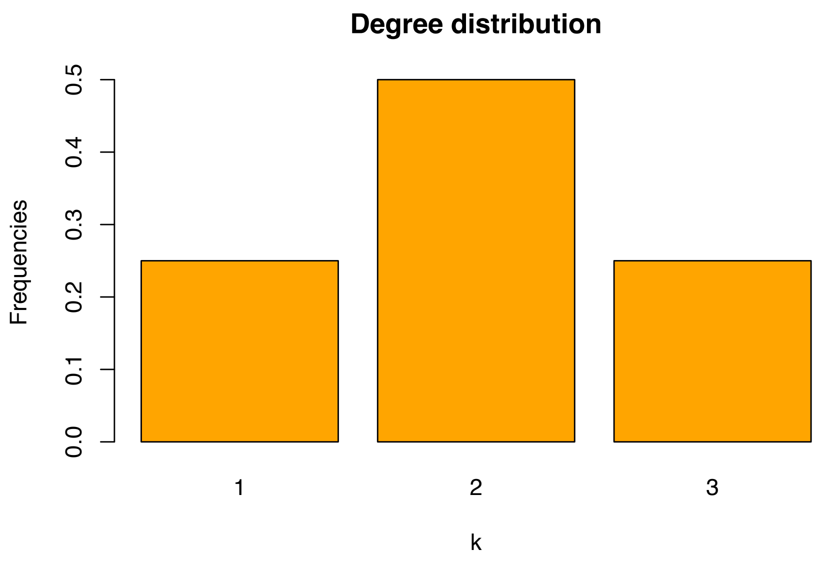
$$p_k = \frac{N_k}{N} \tag{2.6}$$

where N_k represents the number nodes that have a degree of k. From the equation above, we can infer that $N_k = Np_k$

We also know that since p_k is a probability, it should add up to 1:

$$\sum_{i=1}^{\infty} p_k = 1 \tag{2.7}$$

Take the network in Figure 2.2 as an example, its degree distribution is as follows:



2.4 Adjacency matrix

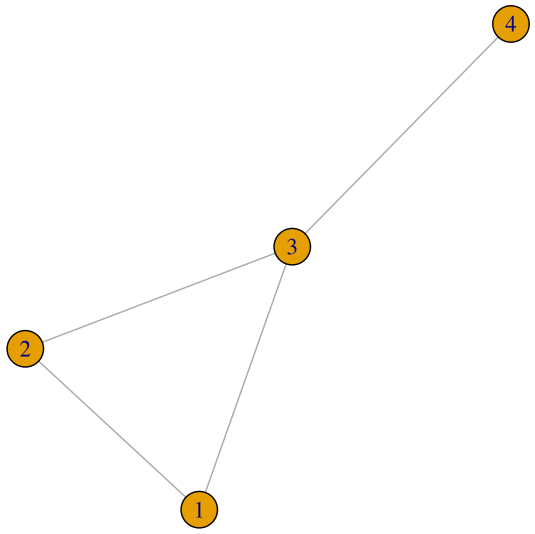
The adjacency matrix of a network that has N nodes has N rows and N columns. If there is a link from node i to node j, then $A_ij = 1$. If node i and node j are not connected, then $A_ij = 0$.

For an undirected network, the link (i, j) has two representations: $A_ij = A_ji$. That's why the adjacency matrix of an undirected network is always symmetric.

Lets have a look at the adjacency matrix of `g1` and `g2`.

For `g1`:

```
par(mar = c(1, 1, 1, 1))
plot(g1)
```

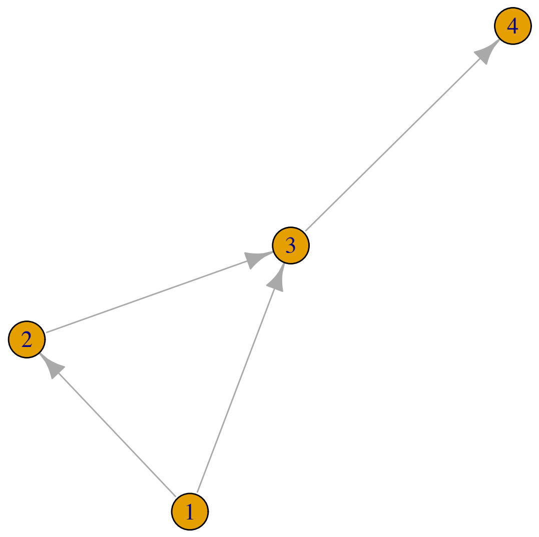


```
as_adjacency_matrix(g1)
```

```
## 4 x 4 sparse Matrix of class "dgCMatrix"
##
## [1,] . 1 1 .
## [2,] 1 . 1 .
## [3,] 1 1 . 1
## [4,] . . 1 .
```

For g2:

```
par(mar = c(1, 1, 1, 1))
plot(g2)
```



```
as_adjacency_matrix(g2)
```

```
## 4 x 4 sparse Matrix of class "dgCMatrix"
##
## [1,] . 1 1 .
## [2,] . . 1 .
## [3,] . . . 1
## [4,] . . . .
```

Each dot in the matrix means 0. The adjacency matrices here did not label the column name. To better understand it, you can imagine that the columns are also labeled as [1,], [2,], [3,] and [4,] as the rows are.

2.5 Chapter 2 Homework

This part is based on Professor YY's module 2 assignment of measuring the friendship paradox[7].

You can find the homework here[8].

The following task prompts are based on the above homework. You can use them to keep practicing coding in Python.

2.5.1 Task 2-1: `Networkx` Basics

- 2-1-1. Import `networkx` as `nx`.
- 2-1-2. Set both random seed and numpy random seed to be `42`.
- 2-1-3. Create a networkx graph named `my_first_graph` which contains three nodes and 2 edges, i.e., (1,2) and (2,3).
- 2-1-4. In the codes you used for Task `2-3`: Which element is a `class`? Which is an `object`? Which is a `method`?
- 2-1-5. Print the number of nodes in `my_first_graph`. *Hint* : `len()`
- 2-1-6. Print the degree of Node 2, in the format of "Node 2's degree: m".
- 2-1-7. Print Node 2's neighbors, in the format of "Node 2's neighbors: [m, n, p, q]". *Hint* : `neighbor()` or `neighbors()`?
  - Note: When you directly return Node 2's neighbors, the result will be `dict_keyiterator at 0x10ec84860`, which is an iterator object[9]. Try to solve this problem either by a for loop or through a list comprehension[10]
- 2-1-8. Print all the node's neighbors in `my_first_graph` in the form of "Node 1's neighbors: [m, n, p, q]" using a python for loop.
  - *Hint* : The for loop will iterate over the list returned by the `nodes()` function.
- 2-1-9. Print all the edges in `my_first_graph`.
- 2-1-10. Plot `my_first_graph`. *Hint* : `.draw()`

[8] https://github.com/yy/netsci-course/blob/master/m02-whynetworks/friendship_paradox.ipynb
[9] https://www.w3schools.com/python/python_iterators.asp
[10] https://python-3-patterns-idioms-test.readthedocs.io/en/latest/Comprehensions.html

2.5.2 Task 2-2: Friendship paradox example

- 2-2-1. Create a star graph with 20 nodes, and plot it.
- 2-2-2. Calculate the average degree of the star graph you just created. Use a for loop to calculate the total degrees.
  - *Hint* : You might find the operator of `+=` useful for `2-2-2`. `a += b` is short for `a = a + b`.
- 2-2-3. Do the same thing using numpy's mean function[11] which takes in a list and returns the mean of this list.

I do not understand Professor YY's note here: "so if you print a numpy float it will display differently than if you printed a python float, which is why we get the intentional truncation of the value when displayed".

- 2-2-4. Get `star_graph`'s average degree using networkx's `info()` function.

2.5.3 Task 2-3: Generating random and scale-free networks

- 2-3-1. Generate a random network with 20 nodes and a connection probability of 0.3. *Hint* : `erdos_renyi_graph`[12].
- 2-3-2. Generate a scale-free network with 20 nodes with each new node having three edges to attach to existing nodes. *Hint* : `barabasi_albert_graph()`[13]

2.5.4 Task 2-4: Homework by Professor YY

Answer Q1 - Q4[14] as explained by Professor YY.

[11] https://numpy.org/doc/stable/reference/generated/numpy.mean.html

[12] https://networkx.github.io/documentation/stable/reference/generated/networkx.generators.random_graphs.erdos_renyi_graph.html

[13] https://networkx.github.io/documentation/stable/reference/generated/networkx.generators.random_graphs.barabasi_albert_graph.html

[14] https://github.com/yy/netsci-course/blob/master/m02-whynetworks/friendship_paradox.ipynb

3

Small World Phenomenon

The following is not originally from me, but my notes from reading *Network Science*[1] (Ch. 2.6-2.10) by Albert-László Barabási[2]. Notes are accompanied by my explanations.

Before we can understand the small world phenomenon, we need to have more basic knowledge about a network: shortest path, average path length, Breath-first search algorithm, components, and local and average clustering coefficient.

3.1 Weight, paths, connectedness, and clustering

3.1.1 Weighted network

So far, we assumed that the weight of all links in a network is the same. However, networks can be weighted. Take social network as an example. Although you are connected to many people, the weight of these connections is unequal. You probably talk to your family and your close friends more often than to your acquaintances. In an adjacency matrix, suppose you are represented by node i, your mom node j and one of your acquaintances node m. Also suppose the weight of the link between you and your mom is 10, which is ten times the weight of the link between you and the acquaintance. Then we have $A_{ij} = 10$ and $A_{im} = 1$.

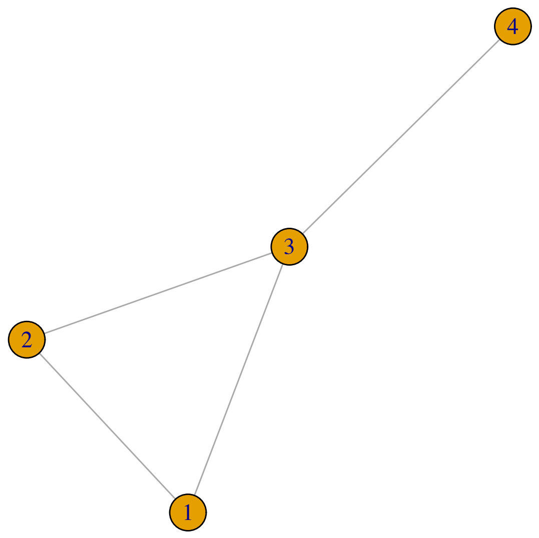
3.1.2 Shortest path, network diameter, and average path length

3.1.2.1 Shortest path

In real world, objects have physical distances between each other, and that's why you need to fly for three hours from New York to Florida. In networks, physical distances are replaced by **path lengths**: how many links does the path contain?

For example, in `g1`, the path length between node 1 and node 4 can either be 2 ($1 \rightarrow 3 \rightarrow 4$) or 3 ($1 \rightarrow 2 \rightarrow 3 \rightarrow 4$):

```r
par(mar = c(1, 1, 1, 1))
plot(g1)
```



Obviously, the path length of 2 is smaller than 3. In this case, $1 \rightarrow 3 \rightarrow 4$ is the **shortest path** (denoted as $d_{i,j}$), which is defined as the path with the fewest number of links. In the above example, $d_{1,4} = 2$.

In an undirected network, $d_{i,j} = d_{j,k}$. This often does not hold true in a directed network because:

1. The existence of (i, j) does not guarantee the existence of (j, i);

2. Even if both (i,j) and (j,i), the two often are unequal to each other.

For example,

```
g3 <- graph( edges=c(1,2, 1,3, 2,3, 3,4, 4,2), n=4, directed=TRUE)
par(mar = c(1, 1, 1, 1))
plot(g3)
```

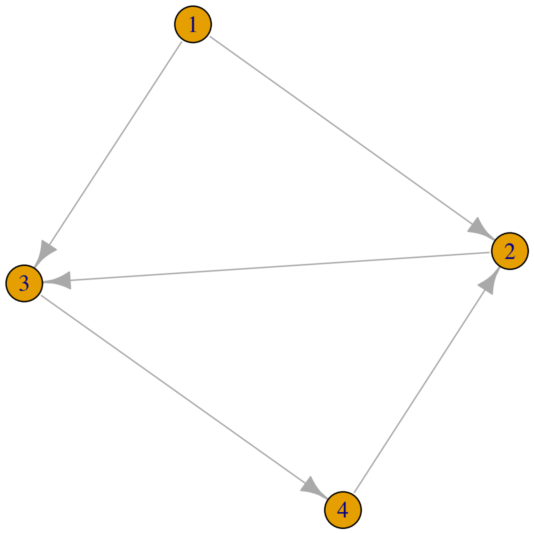


FIGURE 3.1: shortest paths in a directed network

There are paths ($1 \to 2$, and $1 \to 3 \to 4 \to 2$) from node 1 to node 2 but we don't have on any from node 2 to node 1. Furthermore, $d_{2,4} = 2$ whereas $d_{4,2} = 1$.

Please note that all the above discussions on shortest path lenghts are based on the assumption that it is a simple contagion. In a complex contagion, things are very different because multiple reinforcements are necessary. Read Centola and Macy (2007) for more detailed explanations.

3.1.2.2 Network diameter

Among all the shortest paths in a network, the longest one is called the **diameter** of this network, denoted as d_{max}. In the above Network 3.1, $d_{max} = 2$: $d_{1,4} = d_{2,4} = 2$.

3.1.2.3 Average path length

Average path length is defined as the average length of shortest paths between any two nodes. For example, in Network 3.1, $d_{1,2} = d_{1,3} = d_{2,3} = d_{3,4} = d_{4,2} = 1$, $d_{1,4} = d_{2,4} = d_{3,2} = d_{4,3} = 2$. Therefore, the average is $\frac{(1 \cdot 5 + 2 \cdot 4)}{5+4} = 1.44$. Lets test it:

```r
average.path.length(g3)
```

```
## [1] 1.444444
```

3.1.2.4 Breath-first search (BFS) algorithm

As the size of a network gets larger, it will be more difficult to know what is the shortest path between a pair of nodes. For example, without using computers,

For example, in the network @ref9fig:random3-10 below, what is the shortest path from node 3 to node 90?

```r
set.seed(42)
random3 <- erdos.renyi.game(100, 0.1, type = "gnp")
plot(random3)
```

We are not even sure whether the two nodes are connected or not, let alone knowing the shortest path. Of course, computers can do this very fast:

```r
shortest_paths(random3, from = 3, to = 90)
```

```
## $vpath
## $vpath[[1]]
## + 4/100 vertices, from 15a6d78:
## [1]  3  8 28 90
##
```

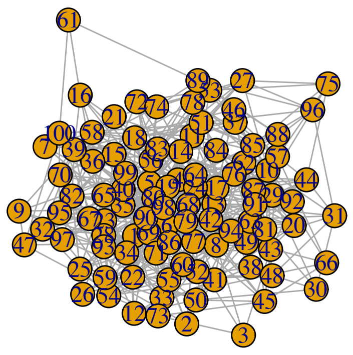


FIGURE 3.2: A random network to explain Breath-first search (BFS) algorithm

```
##
## $epath
## NULL
##
## $predecessors
## NULL
##
## $inbound_edges
## NULL
```

From the result, we know that $d_{3,90} = 3$. The path is: $3 \rightarrow 8 \rightarrow 28 \rightarrow 90$. But how did the computer find this path?

The most frequently used algorithm to find the shortest path between two nodes is called "Breadth-First Search (BFS)". It starts from the "source" node, which is labeled as 0. Neighbors of the "source" node is then labeled as 1, then the neighbors' neighbors, until it reaches the "target" node.

For example, we are interested in knowing the shortest path from the source node ("S") to the target ("T") in the following Network 3.3:

The breadth-first search algorithm is shown in Figure 3.4 below:

We can see that the shortest path from the "source" to the "target" is 3.

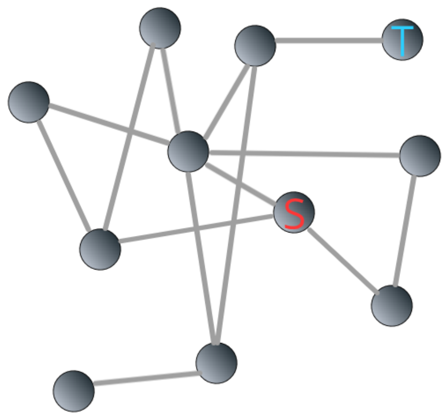


FIGURE 3.3: Illustrating breadth-first search algorithm: source and target

3.1.3 Connectedness

In a disconnected network, subnetworks are called *components* or *clusters*. A *bridge* can connect two components.

For a large network, there are two ways to find out whether it consists of *components*:

1. Using linear algebra to look at the network's adjacency matrix could be rearranged into a block diagonal;
2. Using BSF algorithm.

For example, in Figure 3.5, the network contains two components:

```
disconnectedG <- graph( edges=c(1,2, 1,3, 2,3, 1,4, 5,6),
                        n=6,
                        directed=FALSE)
par(mar = c(1, 1, 1, 1))
plot(disconnectedG)
```

In its adjacency matrix, nonzero elements are contained in square blocks along the diagonal and all the remaining elements are zeros:

If the components are bridged:

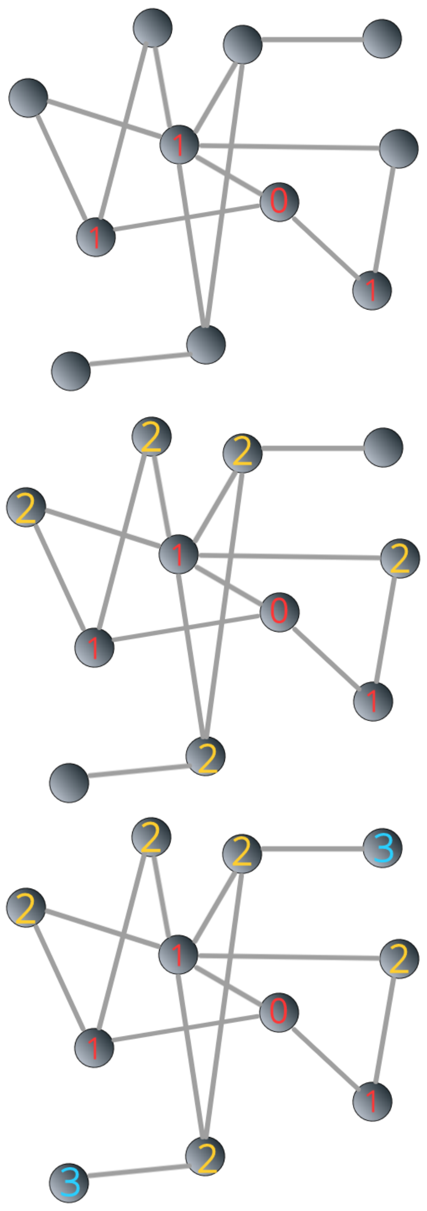


FIGURE 3.4: Illustrating breadth-first search algorithm: steps

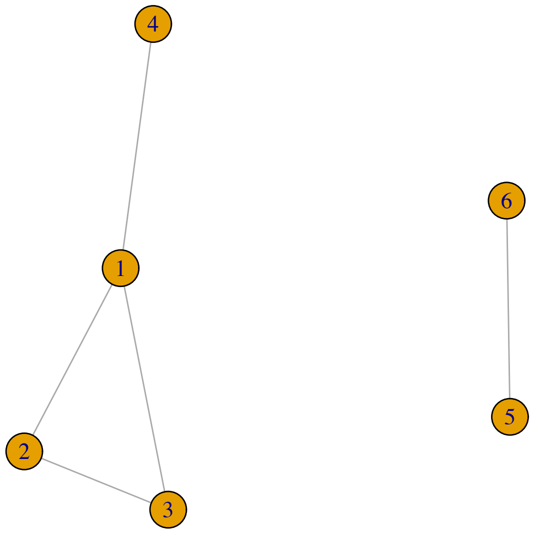
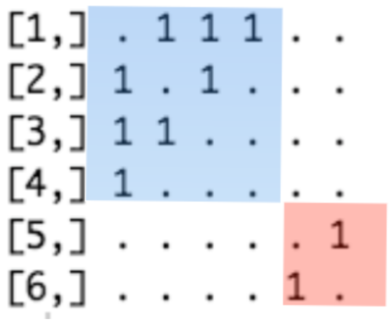


FIGURE 3.5: An example of disconnected network



```
[1,] . 1 1 1 . .
[2,] 1 . 1 . . .
[3,] 1 1 . . . .
[4,] 1 . . . . .
[5,] . . . . . 1
[6,] . . . . 1 .
```

FIGURE 3.6: Adjacency matrix of the disconnected network

```r
connectedG <- graph( edges=c(1,2, 1,3, 2,3, 1,4, 3,5, 5,6), n=6,
                     directed=FALSE)
par(mar = c(1, 1, 1, 1))
plot(connectedG)
```

Then its adjacency matrix cannot be rearranged in a block diagonal form:

```
## 6 x 6 sparse Matrix of class "dgCMatrix"
```

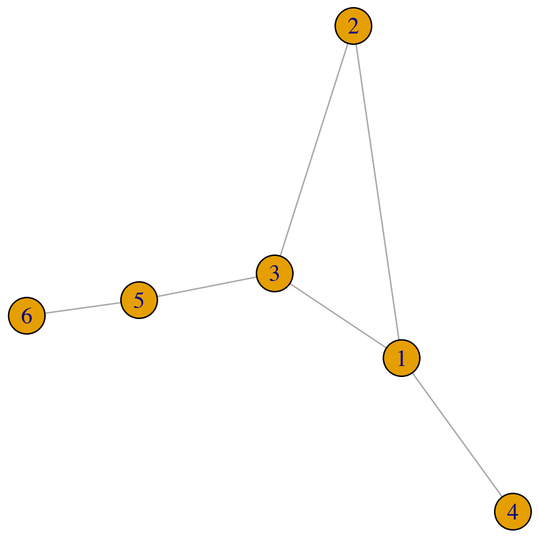


FIGURE 3.7: An example of connected network

```
##
## [1,] . 1 1 1 . .
## [2,] 1 . 1 . . .
## [3,] 1 1 . . 1 .
## [4,] 1 . . . . .
## [5,] . . 1 . . 1
## [6,] . . . . 1 .
```

Then how can we find whether a network is connected or disconnected using BFS algorithm?

It's very easy. In a connected network, we can reach all the nodes from the "source node". Therefore, the number of labeled nodes is equal to the size of the network, N. However, in a disconnected network, nodes in different components cannot be reached, so the number of labeled nodes is smaller than N.

To illustrate, let's go back to the network 3.3. This time, we add a component consisting two nodes to it.

We apply BFS algorithm to the network 3.8. The starting point is the same as above. You can see that the process ceased when the label reaches 3:

How can we identify the remaining component(s)? In the example above, we

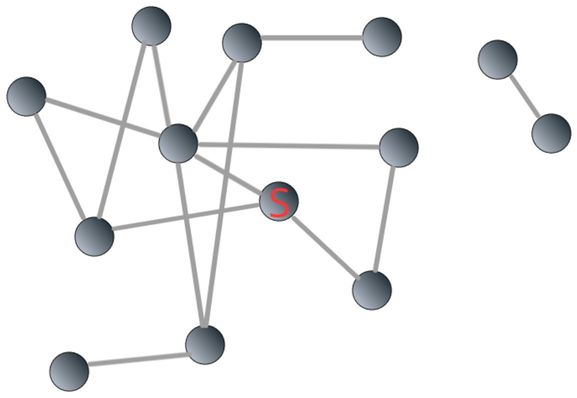


FIGURE 3.8: A disconnected network composed of two components

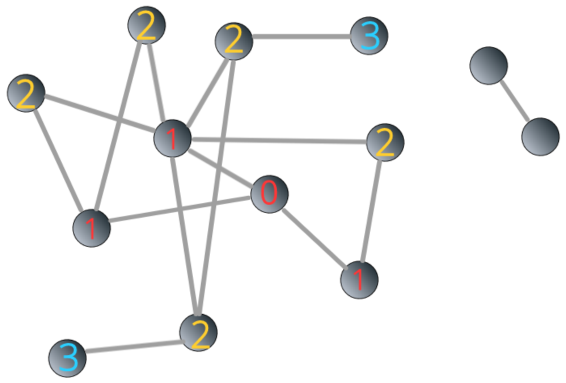


FIGURE 3.9: BFS stops in a disconnected network

assign label 4 to a randomly unlabeled node j and all the other nodes reachable from node j.

What if there are multiple components? It's the same as the above step. After labeling all reachable nodes, if the number of labeled nodes is still smaller than N, then we assign label 5 to a randomly unlabeled node m, and all other nodes reachable from node m. The process goes on until the number of labeled nodes is equal to N.

3.1.4 Clustering coefficient

To what degree are a node's neighbors connected with each other? That's what clustering coefficient measures.

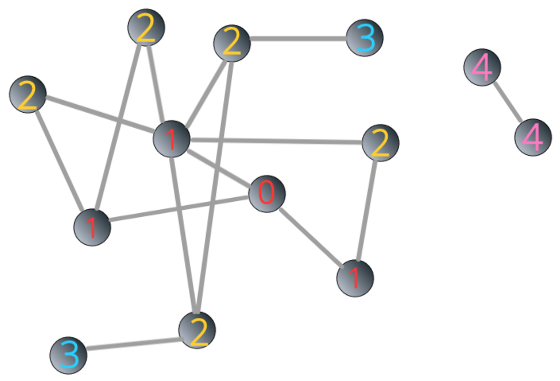


FIGURE 3.10: Identifying components in a disconnected network through BFS algorithm

If the degree of node i is k_i, then node i's **local clustering coefficient** is defined as

$$\begin{aligned} C_i &= L_i \div \frac{k_i \cdot (k_i - 1)}{2} \\ &= \frac{2 \cdot L_i}{k_i \cdot (k_i - 1)} \end{aligned} \tag{3.1}$$

L_i represents the actual number of links between node i's k_i neighbors, and $\frac{k_i \cdot (k_i - 1)}{2}$ means the highest number of links possible between those neighbors.

Local clustering coefficient of a node is the probability that two neighbors of a given node has a link to each other. It measures the local density of links in a network.

Local clustering coefficient is for nodes. If we are interested in the degree of clustering for a network, we need to calculate **average clustering coefficient**, denoted as $\langle C \rangle$ which is simply the average of local clustering coefficient of all the nodes in a given network:

$$\langle C \rangle = \frac{1}{N} \cdot \sum_{i=1}^{N} C_i \tag{3.2}$$

3.2 Chapter 3 Homework

This part is based on Professor YY's module 3 assignment of shortest path length distribution[3].

You can find the homework here[4].

[4] https://github.com/yy/netsci-course/blob/master/m03-smallworld/shortest_path_length_distribution.ipynb

4

Weak Ties and Watts-Strogatz model

In this chapter, we are going to learn how to generate random networks. Then, we will study some important characteristics of random networks, such as average degree, and degree distribution. We are also going to read two of the most influential papers and get an idea of weak ties and Watts-Strogatz model.

4.1 Random networks (Barabási Ch.3)

The following is not originally from me, but my notes from reading *Network Science*[1] (Ch. 3) by Albert-László Barabási[2]. Notes are accompanied by my explanations.

4.1.1 Generating a random network

Most of networks in real life look like randomly constructed. Then, how can we, as humans, produce random networks that we see in nature?

Simply by putting links randomly between nodes.

But how can we make sure that we are putting links **randomly**?

Think about it for a minute before reading on.

If you have studied statistics, you would know that **randomness** is all about **probability**. If we say that we are going to **randomly** pick a person from a group, then we'll need to make sure that **each** person in the group has an **equal** chance to be picked. Otherwise, the choice is not **random**.

Now, let's go back to our original problem: how can we **randomly** put links between nodes?

For example, we now have 100 isolated nodes, which are called **singletons**, as shown below. What to do next?

```
g4_1 <- graph( edges=NULL, n=100)
plot(g4_1)
```

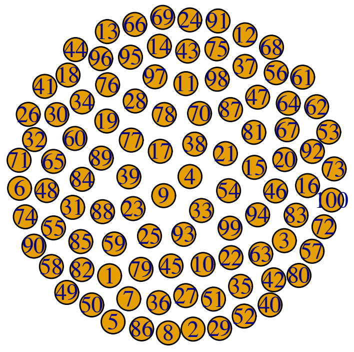


FIGURE 4.1: 100 isolated nodes

Frist, we'll set up a value p ($0 \leq p \leq 1$), called **link probability**. Then, we'll pick a pair of nodes (STEP 1), for example, i and j, and generate a ramdom number r for this pair (STEP 2). If $r < p$, then we connect i and j. Otherwise, we keep them disconnected. We'll repeat step 1 and step 2 for all node pairs, the total number of which should be $\frac{N(N-1)}{2}$. This way, we will make sure that the probability of each pair being connected is exactly p. Why so?

```
plot(c(0, 1), c(0, 0.2),
     type= "n",
     xlab = "",
     ylab = "",
     yaxt="none")
# Solution: https://rstudio-pubs-static.s3.amazonaws.com/
# 297778_5fce298898d64c81a4127cf811a9d486.html
abline(v=0.1, col="red")
rect(0,0,0.1,0.2,col="lightgreen")
rect(0.1,0,1,0.2,col="red")
```

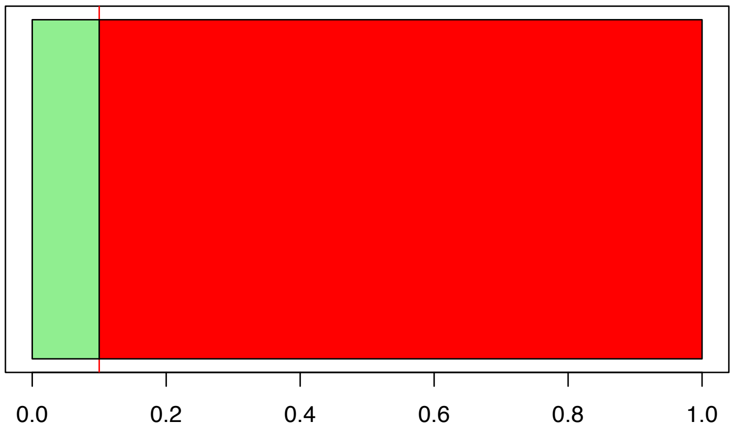


FIGURE 4.2: Illustrating why the probability of each pair being connected is equal to the link probability

The reason can be shown in Figure 4.2. If we set $p = 0.1$, then only 10% of random numbers will fall in the range of $[0, 0.1)$ (the area shown in light green), the remaining 90% will fall in $[0.1, 1]$[3].

Okay, let's set $p = 0.1$ and see what will happen:

```r
# Solution from: https://rpubs.com/lgadar/generate-graphs
set.seed(42)
erdosRenyi <- erdos.renyi.game(100, 0.1, type = "gnp")
plot(erdosRenyi)
```

The method described above is used in a model called $G(N, p)$ **model**, which was introduced independently by Edgar Nelson Gilbert. In this model, the number of nodes (N), and link probability (p) are fixed.

There is a similar model called $G(N, L)$ **model** where the number of nodes and the number of links are fixed. This model was introduced by Pál Erdős and Alfréd Rényi.

In practice, we use $G(N, p)$ **model** more. But since Pál Erdős and Alfréd Rényi

[3]I am a little bit uncertain here because I don't know where to put 0.1.

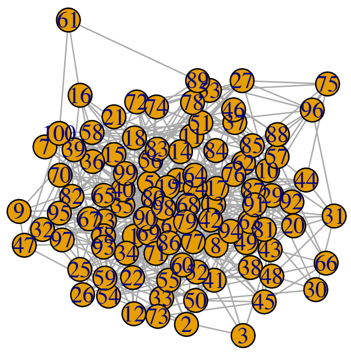


FIGURE 4.3: An Erdős-Rényi network

played such an important rold in advancing our understanding of random networks, we still name networks generated by the $G(N, p)$ **model** as an **Erdős-Rényi network**.

In Chapter 3.2[4] of *Network Science*[5] , there is a mistake: the second step of generating a random network said that "if the number exceeds p, connect the selected node pair with a link". However, it should be "if the number is smaller than p, connect the selected node pair with a link."

4.1.2 Average degree, and the expected number of links in a random network

We can think of the process of first generating a random number for a pair of nodes, then comparing it with p, and finally deciding whether to link the pair or not as tossing a coin (Menczer et al., 2020).

Imagine we have a biased coin which gives us heads with probability p, which is equal to the **link probability** we talked about before. Take $p = 0.1$ as an example. If we toss the coin for ten times, then we are expecting $10 * 0.1 = 1$

head, right? For the same token, when we have 100 tosses, we will be expecting 10 heads.

In the procudure of random network generation discussed above, we concluded that a pair of nodes being connected has a probability of p, which is the same as the probability of us having a head when we toss a coin. Since we are expecting 10 heads out of 100 tosses, then how many connected pairs of nodes are expected, or, how many links are expected, if we examine 100 pairs? 10, right?

How do we get this number? We simply multiply the total number of tosses in the case of flipping coins (or the total number of pairs of nodes we have in the case of random network generation) by p. Now, in the $G(N, p)$ model, we have N nodes. It's easy to understand that we will have $\binom{N}{2} = \frac{N(N-1)}{2}$ pairs of nodes to examine. So, **the number of links we are expecting in this random network** is:

$$\langle L \rangle = \frac{N(N-1)}{2} p \tag{4.1}$$

$\langle L \rangle$ here stands for the **expected value**. A given random network generated by $G(N, p)$ does not necessarily have exactly $\frac{N(N-1)}{2}p$ links. But as we generate more and more random networks using the $G(N, p)$ model, the average number of links will be $\frac{N(N-1)}{2}p$. That's what we mean by **expected value**. You can look at **Image 3.3** in *Network Science*[6] for examples and illustrations.

Then, in this random network, what is the **average degree**?

Recall Eq. (2.2). Replacing L with $\langle L \rangle$ from Eq. (4.1), we have:

$$\langle k \rangle = \frac{2\langle L \rangle}{N} = p(N-1) \tag{4.2}$$

How to remember Eq. (4.2):

The average degree of a random network is the product of p, the **link probability**, and $(N-1)$, the maximum number of links (or neighbors) a node can have.

You can read Ch. 3.3[7] of *Network Science*[8] for more detailed mathematical reasoning.

[6]http://networksciencebook.com/
[7]http://networksciencebook.com/chapter/3#number-of-links
[8]http://networksciencebook.com/

4.1.3 Degree distribution

4.1.3.1 Binomial distribution

First of all, if you are not familiar with combinations and permutations, or that you have forgotten what you learnt in your high school, you are encouraged to go through this amazing tutorial[9] on mathsisfun.com[10].

Then, carefully read throught this tutorial on binomial distribution[11].

If you are able to understand the tutorials above, then you should know that if we have a biased coin which produces heads with probability p, the probability of having k heads out of n tosses is:

$$\binom{n}{k}p^k(1-p)^{n-k} \tag{4.3}$$

How to understand it?

We can look at it this way: tossing a coin n times, we have 2^n different outcomes (i.e., combinations of heads and tails), and the number of outcomes (or combinations, if you want) that have k heads is $\binom{n}{k}$ outcomes. However, we cannot simply use $\binom{n}{k} \div 2^n$ to calculate the probability of having k heads. Why? Because this is a **biased** coin, so each outcome (or combination) has different probabilities.

What should we do then?

Now we know the number of outcomes that will produce k heads out of n tosses. It will be great if we know the probability of each of these outcomes and sum them up. Bingo!

When we think more deeply, we will know that each of these $\binom{n}{k}$ outcomes has exactly the same probability: $p^k(1-p)^{n-k}$. But why? Read this tutorial on binomial distribution[12] again, especially the tree diagram. Also, you'll find this tutorial[13] on the probability of independent events helpful.

Now, let's go back to random networks.

[9] https://www.mathsisfun.com/combinatorics/combinations-permutations.html
[10] https://www.mathsisfun.com/
[11] https://www.mathsisfun.com/data/binomial-distribution.html
[12] https://www.mathsisfun.com/data/binomial-distribution.html
[13] https://www.mathsisfun.com/data/probability-events-independent.html

TABLE 4.1: Comparing random network generation and flipping coins

	p	N	k
random network generation	link probability	number of nodes	number of node pairs successfully connected
tossing coins	probability of having a head in one toss	number of tosses	number of heads

For a given node i, the maximum number of links it can have is $N-1$. Let's denote the probability of node i having k links as p^k. Eq. (4.3), we know that:

$$p^k = \binom{N-1}{k} p^k (1-p)^{N-1-k} \tag{4.4}$$

In this binomial distribution, the mean is:

$$E(x) = Np \tag{4.5}$$

Its standard deviation is:

$$\sigma_i = [p(1-p)N]^{\frac{1}{2}} \tag{4.6}$$

And its second moment is:

$$E(x^2) = p(1-p)N + p^2N^2 \tag{4.7}$$

Sorry that I am currently not capable of proving Eq. (4.5) to Eq. (4.7). For now, just memorize them.

4.1.3.2 Poisson distribution

Most real networks are sparce, so its average degree $\langle k\rangle$ is much smaller than the size of the network, N. Considering this limit, we usually use Poisson distribution to describe a random network's degree distribution because of simplicity:

$$p_k = e^{-\langle k\rangle} \frac{\langle k\rangle^k}{k!} \tag{4.8}$$

Eq. (4.4) and Eq. (4.8) are collectively called **degree dostribution of a random network**.

Something needs remembering:

1. Binomial form is the exact version; Poisson distribution is only an approximation;
2. We'd better use Binomial distribution to describe a small network, for example, $N = 10^2$, but use Poisson distribution for large networks, for example, $N = 10^3$ or 10^4;
3. In Poisson distribution, the standard deviation is $\sqrt{\langle k \rangle}$;
4. The Poisson distribution tells us that for two networks, as long as they have the same $\langle k \rangle$, their degree distribution is almost exactly the same despite different sizes, i.e., N.

I did not quite understand the lecture on the Poisson distribution by Professor YY.

4.1.4 Poisson distribution does not capture reality

Poisson distribution undoutedly accurately describes the degree distribution of random networks, but we need to ask, do random netwoks reflect reality?

Reading Ch. 3.5[14], will let us know that if random networks can describe social networks in our daily lives, we would expect that:

1. Most people will have $\langle k \rangle \pm \sqrt{\langle k \rangle}$ friends;
2. The highest number of friends a person can have is not that different than the smallest possible number.

However, we know that this is not the case in real life. Many people have over 5,000 contacts on Facebook and WeChat.

From the figure shown in Ch. 3.5[15], we will know that in real networks, both **the number of high degree nodes**, and **the standard deviation of the distribution**, are much larger than what is expected from random networks.

[14] http://networksciencebook.com/chapter/3#not-poisson
[15] http://networksciencebook.com/chapter/3#not-poisson

4.2 Reading two seminal papers

4.2.1 *The Strength of Weak Ties* by Granovetter (1973)

Please read the following articles carefully by yourself before reading my notes.

Granovetter, M. S. (1973). The strength of weak ties[16]. *American journal of sociology, 78*(6), 1360-1380.

This is a highly influential paper: more than 58K as of Nov. 2020.

- Overlap in friendship circles: Suppose there are two individuals, A & B, and a group of people, $S = C, D, E$. The stronger the tie between A and B, the more people in Group S will be tied to **both** A and B.
- If there is a strong tie between A and B, and between A and C. It's very likely that B and C also are friends, because

  1. A common contact will bring B and C into interaction;
  2. The stronger the tie, the more similar are individuals to each other. A is similar to B, and A is similar to C, and thus B is similar to C, making it more likely for them to be friends; and
  3. If B and C are aware of each other but are not friends, they might have a "psychological strain" since their feelings are not consistent with A.

Therefore, this tried in Figure 4.4 is very unlikely to occur:

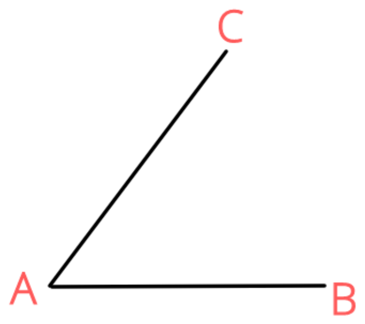


FIGURE 4.4: An impossible tried, adapted from Fig.1 of Granovetter (1973)

[16] https://www.cs.cmu.edu/~jure/pub/papers/granovetter73ties.pdf

- A bridge is a link connecting two otherwise disconnected groups. “No strong tie is a bridge”. All bridges are weak ties.
- A local bridge of n: if this bridge does not exist, what is the shortest path between these two points?
- No strong tie is a local bridge.
- Whatever is is to be diffused can reach a larger number of people if passed through weak rather than strong ties[17].
- People are more likely to find jobs through weak ties rather than strong.

Something I do not understand yet in this paper: “Thus, network fragmentation, by reducing drastically the number of paths from any leader to his potential followers, would inhibit trust in such leaders.”

4.2.2 Watts-Strogatz model of small world

Please read the following paper:

Watts, D. J., & Strogatz, S. H. (1998). Collective dynamics of ‘small-world’networks[18]. *Nature, 393*(6684), 440-442.

It’s another canonical study in the field of network science, also highly influential.

Here[19] is an amazing tutorial & practice to understand WS model. And here[20] is another one I found.

[17]Centola and Macy (2007) advanced this idea by distinguishing between complex contagions and simple contagions like an epidemic. They argued that the strength of weak ties should not be simply generalized to complex contagions, which require affirmation from multiple sources. Therefore, not only the length, but also, and maybe more importantly, the width of the ties influences complex contagions.

[18]`http://materias.df.uba.ar/dnla2019c1/files/2019/03/watts-collective_dynamics-nature_1998.pdf`

[19]`https://dwulff.github.io/_Networks/Downloads/watts_strogatz.html`

[20]`http://worrydream.com/ScientificCommunicationAsSequentialArt/`

5 Scale-Free Networks

Networks in real life are rarely random. Most of them are "scale-free". What does "scale-free" mean? What are the defining features of scale-free networks? And how do we generate these networks? These are the topics we are going to cover in this Chapter 5.

5.1 Power-Law Distributions

The following came from the online course of INFO I606[1] at Indiana University by Professor Yong-Yoel ("YY") Ahn[2].

5.1.1 Differences between FB study and the Milgram study

The differences between the Facebook study[3] and the Milgram study:

1. In FB study, researchers knew the whole graph. They would have this in mind when calculating the shortest paths. However, participants in the Milgram study did not know the whole graph;
2. There wasn't any lost "packages" in FB study.

One thing we need to be cautious with the FB study is that there were quite a number of people having 500-5,000friends, as can be seen in the study PDF[4]. But is it possible in real life? Dunbar's number[5] suggests that we as humans can maintain at most 150 friends. Therefore, the FB study might have overestimated the distance between people.

[3] https://research.fb.com/blog/2016/02/three-and-a-half-degrees-of-separation/
[4] https://arxiv.org/pdf/1111.4503.pdf
[5] https://www.newyorker.com/science/maria-konnikova/social-media-affect-math-dunbar-number-friendships

5.1.2 Power law

Before discussing power-law distributions, Professor YY talked about the connection between binomial, Poisson, and normal distributions. It is beyond my scope of knowledge. However, you are encouraged to read this[6] article on this topic.

The above picture of the bird came from here[7].

Check out this video[8] to understand **Zipf's law**. Also, here[9] is an amazing tutorial[10] on the connection between **Zipf's law** and **power laws**.

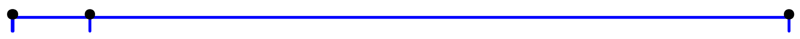
The most frequently used word in English is **the**. Let's denote its frequency as 1. Then the frequency of the second most frequently used word is $\frac{1}{2}$. The number is $\frac{1}{3}$ for the third most frequently used word, and will be $\frac{1}{4}$ for the fourth most frequently used word. That's a **power law distribution**.

5.1.3 Logarithmic scale

To really understand **scale free networks** and **power law distribution**, we need to get an idea of **logarithmic scale** first. Professor YY provided several very helpful Youtube videos that illustrate log scale.

Inspired by the conversation between Vi and Sal[11], I am asking you this question. In the following number line, from 1 to 1 million, where is $1,000$?

Okay, let's label both $1,000$ and $100,000$ in the above number line:



I plotted four points: 0, 1000, 100000, and 1000000, but can only see three.

[7] http://gdut_yy.gitee.io/doc-csstdg4/ch8.html#_8-2-padding
[8] https://www.youtube.com/watch?v=fCn8zs912OE
[9] https://www.hpl.hp.com/research/idl/papers/ranking/ranking.html
[10] I haven't throughly read and understood it yet...
[11] https://www.youtube.com/watch?v=4xfOq00BzJA

Why? It's because 0 and 1000 are almost at the same place. But I guess you've probably thought $1,000$ was in the place of $100,000$, right?

You can learn some basics of **log scale** by watching this video[12] by Khan Academy.

One of the reasons why people need log scale is that it allows us to include a much wider range of numbers in limited space. For example, it is useful when we measure earthquakes[13].

5.1.4 Properties of power-law distributions

5.1.4.1 Porperty 1: A fat tail allowing for many outliers

The following came from the online course of INFO 606[14] at Indiana University taught by Professor Yong-Yoel ("YY") Ahn[15].

A power-law distribution can be expressed as

$$p(x) = Cx^{-\lambda} \tag{5.1}$$

where $\lambda > 1$ [16].

If we take a logarithm of Eq. (5.1), we'll have

$$\begin{aligned} \ln p(x) &= \ln(Cx^{-\lambda}) && (5.2)\\ &= \ln C - \lambda \ln x && (5.3) \end{aligned}$$

If you are not familiar with ln or how the above calculations are done, do read this tutorial[17] on logarithm.

If $a^b = c$, then, $\log_a c = b$, where $c > 0$, $a > 0$, and $a \neq 1$. In this example, a is called **base** of the logarithm.

[12] https://www.youtube.com/watch?v=sBhEi4L91Sg
[13] https://www.youtube.com/watch?v=RFn-IGlayAg
[16] Read here for why: http://tuvalu.santafe.edu/~aaronc/courses/7000/csci7000-001_2011_L2.pdf.
[17] https://www.math10.com/en/algebra/logarithm-log-ln-lg.html

- If the **base** is 10: $\log_{10} a$ will be denoted as $\lg a$
- If the **base** is e: $\log_e a$ will be denoted as $\ln a$

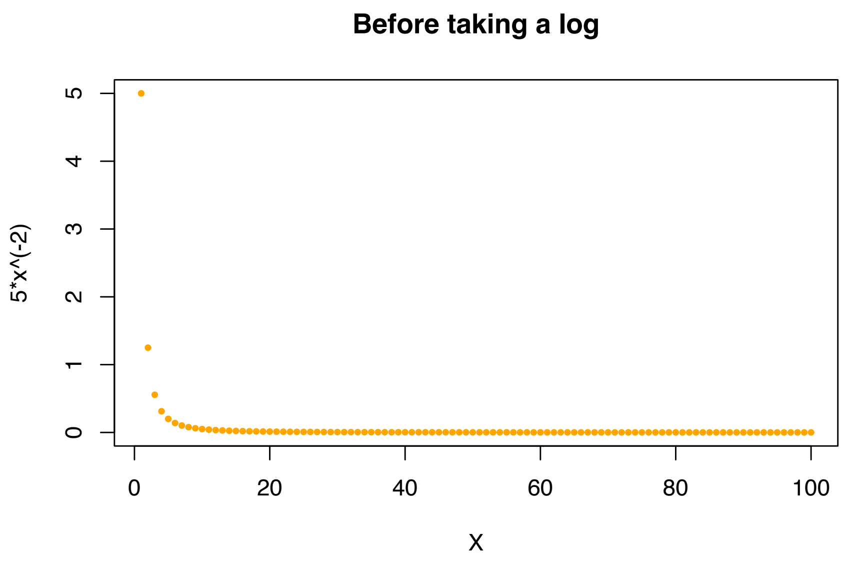
Some logarithmic identities:

1. $log_a b^c = c \cdot log_a b$;
2. $log_a b \cdot c = log_a b + log_a c$;
3. $log_a \frac{b}{c} = log_a b - log_a c$;
4. $log_a^c b = \frac{1}{n} \cdot log_a b$

Since $\ln C$ is a constant, what we get is basically **a linear function**. To better understand it, let's plot it.

Before we take the log:

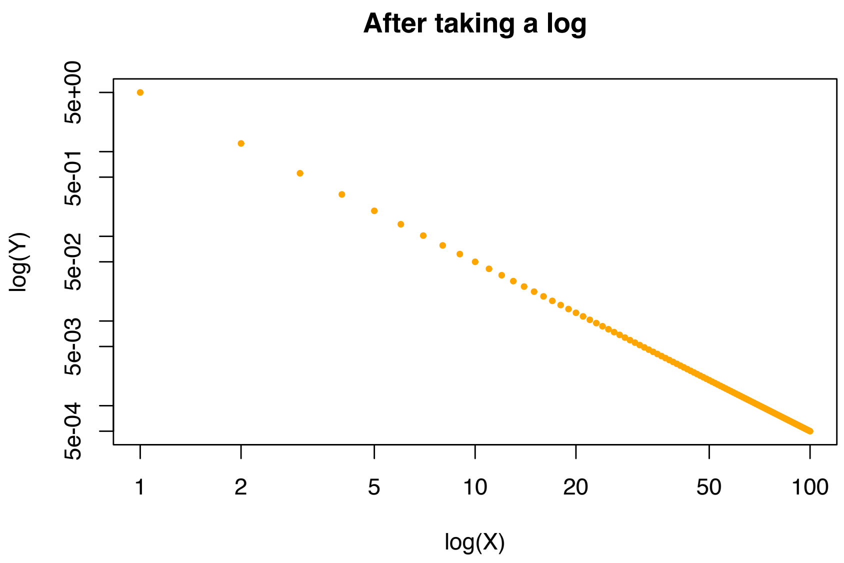
```
x <- 1:100
y <- 5*x^(-2)
plot (x, y, pch=19, cex=0.5, col="orange",
      xlab="X", ylab="5*x^(-2)",
      main="Before taking a log")
```



I learned a lot about how to style the points from reading Dr.Katya Ognyanova's blog post[18].

After we take a log:

```
x <- 1:100
y <- 5*x^(-2)
plot (x, y, pch=19, cex=0.5, col="orange",
      xlab="log(X)", ylab="log(Y)",
      main="After taking a log",
      log="xy"
      )
```



A very important property of power-law distribution on a log-log scale is it has a **heavy tail** or a so-called **fat tail**. Lots of other distributions, like binomial, Poisson, or normal distributions, simply do not have this fat tail, which is super useful when we have lots of outliers in the tail of the distribution.

As we can clearly see below, except for the power-law distribution on a log-log scale, in all the three other distributions, y quickly reaches zero:

[18] https://kateto.net/networks-r-igraph

```
par(mfrow=c(2,2))
# The idea of the following codes comes from
# https://www.statology.org/plot-normal-distribution-r/
x_dnorm <- seq(-6, 6, length=100)
y_dnorm <- dnorm(x_dnorm)
plot(x_dnorm, y_dnorm, pch=19, cex=0.5, col="skyblue",
     main="Normal Distribution")
# The following codes are from
# https://www.tutorialspoint.com/r/r_binomial_distribution.htm
x_dbinom <- seq(0,50,by = 1)
y_dbinom <- dbinom(x_dbinom,50,0.5)
plot(x_dbinom, y_dbinom, pch=19, cex=0.5, col="red",
     main = "Binomail Distribution")
# The following codes are from https://statisticsglobe.com/poisson-
# distribution-in-r-dpois-ppois-qpois-rpois
x_dpois <- seq(-5, 50, by = 1)
y_dpois <- dpois(x_dpois, lambda = 10)
plot(x_dpois, y_dpois, pch=19, cex=0.5, col="purple",
     main = "Poisson Distribution")
x <- 1:100
y <- 5*x^(-2)
plot (x, y, pch=19, cex=0.5, col="orange",
      xlab="log(X)", ylab="log(Y)",
      main="Power-law Distribution (Log-Log Scale)",
      log="xy")
```

If we have many outliers in the tail, for example, in the household income distribution shown below by Donovan et al. (2016), which of the four distributions shown above will you choose to fit the income distribution? Definitely we'll use the power-law distribution which has a fat tail.

We can also compare a normal distribution and a power-law distribution side by side:

```
par(mfrow=c(2,1))
x_dnorm_2 <- seq (0, 10000, by = 1)
y_dnorm_2 <-dnorm(x_dnorm_2)
plot (x_dnorm_2, y_dnorm_2, pch=19, cex=0.1, col="blue",
      xlab="x_dnorm", ylab="y_dnorm",
      main="Normal Distribution")
x <- 0:10000
y <- 5*x^(-2)
plot (x, y, pch=19, cex=0.1, col="orange",
```

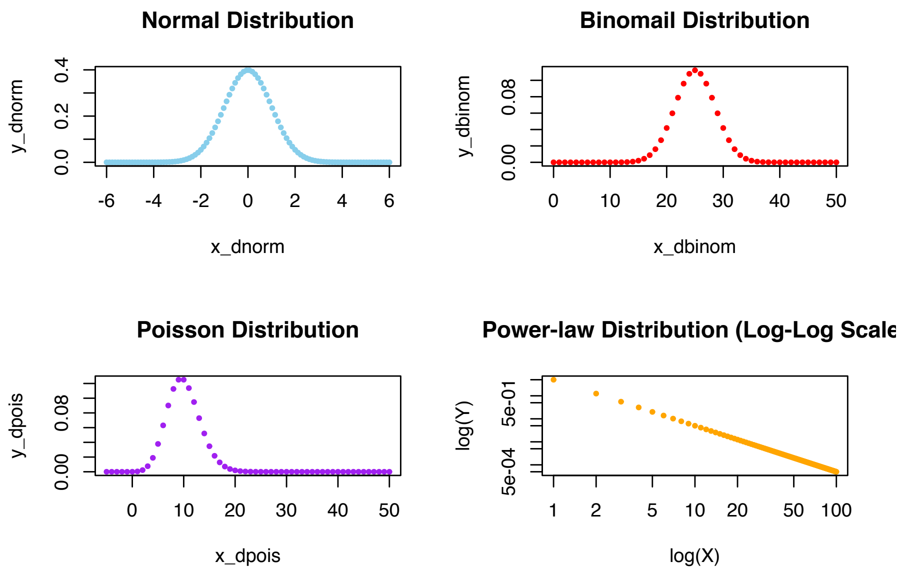
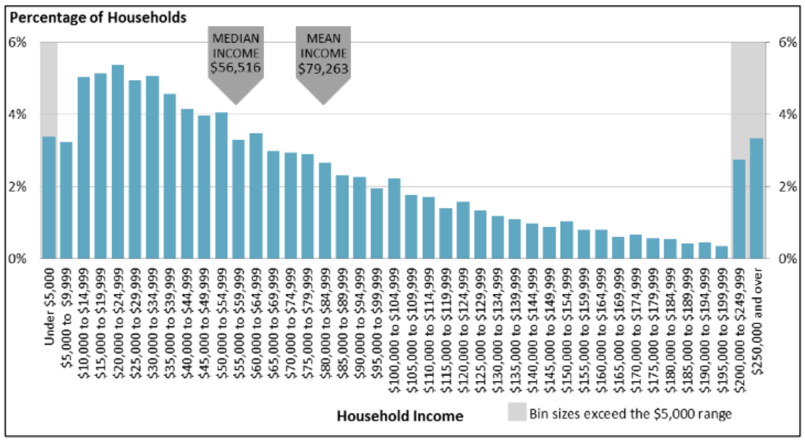


FIGURE 5.1: Comparing Four Different Distributions



Source: U.S. Census Bureau, *Annual Social and Economic Supplement*, available at http://www.census.gov/hhes/www/cpstables/032015/hhinc/toc.htm.

FIGURE 5.2: US Household Income Distribution in 2015, Visualization by @donovan2016us

```
     xlab="log(X)", ylab="log(Y)",
     main="Power-law Distribution (Log-Log Scale)",
     log="xy")
```

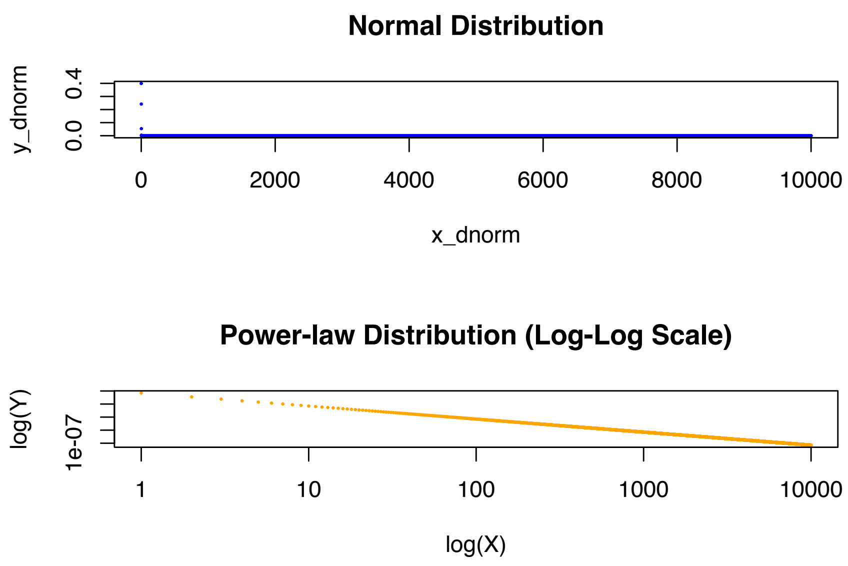


FIGURE 5.3: Comparing Normal Distribution and Power-law Distribution Side by Side

In power-law distribution plotted on a log-log scale, we do expect a lot of outliers.

One thing I still did not understand is the range of x. Should it always be ≥ 1, or ≥ 0? Why or why not?

5.1.4.1.1 Property 2: scale-freeness

To deeply understand what the term "scale-free" means, you can read Chapter 4.4[19] in Network Science[20].

I'll explain in my own words here. To understand "scale-free", we must first

[19] http://networksciencebook.com/chapter/4#scale-free
[20] http://networksciencebook.com/

understand “scale”. Mark Newman (2005) defines it this way: A scale is ”a typical value around which individual elements are centered (p.1). Therefore, **scale-free** means that we don’t know around **which** value individual elements in the distribution are centered. A normal distribution is **not** scale-free because we expect that more than 99% of the data falls within three standard deviations from the mean.

You can read*Power laws, Pareto distributions and Zipf’s law*[21] by Mark Newman (2005) for details. This paper is also a must read if you want to gain a deeper understanding about power law.)

In a power-law distribution, it’s totally different. For example, if we have $y = 5{*}x^{-2}$. Let’s imagine that human height follows this distribution. Suppose your height is x, and n people have the same height as you. Then you’ll see $\frac{1}{4} \cdot n$ people twice as tall as you[22], and $4 \cdot n$ people half of your size. The thing is, **you don’t know where you are located in the distribution!** You can be as tiny as 10 inches or as tall as 100 feet. Wherever you stand, you will always find $\frac{1}{4} \cdot n$ people twice as tall as you, and $4 \cdot n$ people half of your size. **Think about it!** What a crazy world! That’s why we call it **scale-free**: no inherent scale in the distribution.

5.2 Scale-free property

The following is not originally from me, but my notes from reading *Network Science*[23] (Ch. 4.1-4.2) by Albert-László Barabási[24]. Notes are accompanied by my explanations.

- In real networks, there are hubs that are forbidden in random networks.
- No matter what properties of a network we are studying, we need to base our understanding on the network’s **degree distribution**.

The book compares the Poisson distribution and power-law distribution by drawing the degree distribution of the WWW. I wanted to understand these two distributions better so I decided to try drawing them by myself.

The data I used is called “Internet”[25] from http://www-personal.umich.edu/~mejn/netdata/. I’ll use R to plot it. See the result in 5.4

[21] https://arxiv.org/pdf/cond-mat/0412004.pdf
[22] Just input $2x$ in the above formula.
[25] http://www-personal.umich.edu/~mejn/netdata/as-22july06.zip

```
library(here)
internet <- read_graph(here("data","as-22july06.gml"),
                       format = "gml")
# vcount(internet) # Getting an idea of its size // 22963
deg_internet <- degree(internet)
# mean(deg_internet) # Calculating average degree // 4.218613
x <- 0:max(deg_internet) # Setting X-axis
deg.dist_internet <- degree_distribution(internet,
                                         cumulative = TRUE,
                                         mode = "all")
# This will be the degree distribution of our dataset:
y1 <- deg.dist_internet
y2 <- dpois(x, lambda = mean(deg_internet))
# This is to prevent scientific notation on the axes:
options(scipen = 5)
plot( x, y1, pch=19, cex=0.5,
      col="orange",
      xlab="Degree(log)",
      ylab="Probability(log)",
      log = "xy"
      )
# par(new=TRUE)
points(x, y2, pch=19,
     cex=0.5,
     # log = "xy", # We can, in fact, do without this line
     # because the above plot is already on a log-log scale.
     col="blue",
     xlab = "",
     ylab = ""
     )
```

The above code is based on this post[26].

We can see that the degree distribution of the real network is different from Poisson distribution but rather roughly follows a power law. We call networks like this as **scale-free networks**.

A quick note: in the above codes, `y2 <- dpois(x, lambda = mean(deg_internet))` ensures that both networks share the same $\langle k \rangle$. Therefore, the result in the figure tells us that the degree distribution of this real network cannot be approximated by Poisson distribution.

[26] https://stackoverflow.com/questions/6853204/plotting-multiple-curves-same-graph-and-same-scale

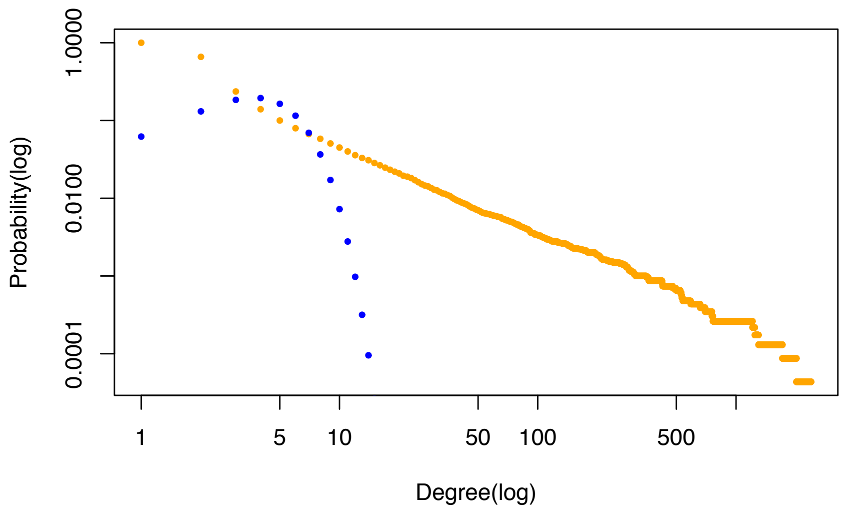


FIGURE 5.4: Poisson distribution does not explain degree distribution in real networks

5.2.1 Hubs

The following is not originally from me, but my notes from reading *Network Science*[27] (Ch. 4.3 - 4.7) by Albert-László Barabási[28]. Notes may be accompanied by my explanation.

Hubs are nodes with high degrees. To understand that there are more hubs in scale-free networks than in random networks, we can refer to Figure 5.4. The two distributions have exactly the same average degree, 4.22.

We will find that compared to random networks, scale-free networks have more low-degree nodes and high-degree nodes, but fewer nodes with degrees near the average degree. This tells us that in random networks, most nodes have comparable degrees (around the average degree of the network) whereas in scale-free networks, there are so many nodes with extremely low and extremely high degrees.

Another difference lies in how the maximum degree (denoted as k_{max}) changes as the size of networks increases. For both types of networks, k_{max} grows as N

becomes larger. The difference is that k_{max} grows faster (polynomially[29]) with N in scale-free networks but it grows slower (logarithmatically[30]) in random networks. Also, if a random network and a scale-free network has the same average degree $\langle k \rangle$, the k_{max} in the scale-free network is almost always larger than that in the random network.

Read the section of *The Largest Hub* in Chapter 4.3[31] for formula derivation. I am currently not capable of doing it.

5.2.2 Universality

Both World Wide Web (WWW) and the Internet are networks. WWW is a network of information. Its nodes are documents and its links are URLs. In contrast, the Internet is a network of infrastructure. Its nodes are computers and its links are cables and wireless connections. Although they are different, their degree distribution can be both approximated with a power law, meaning that there are a large amount of high degree nodes ("hubs") and low degree nodes. In other words, both reveal scale-free properties.

Since scale-free properties are so prevalent, we call scale-free property a universal characteristic for networks.

As ubiquitous as it is, the scale-free property isn't for every networks. Many networks do not share this property, for example, power grid and material networks. Scale-freeness requires that the number of links a node can have is arbitrary, and not systematically constrained. If this requirement is not fulfilled, then hubs are restricted and the network won't show a scale-free property.

Not all scholars agree on the universality of scale-free networks in real life, for example, *Scale-free networks are rare*[32] by Broido and Caluset (2019). And here[33] is the response to this paper from Barabási.

5.2.3 Ultra-small property, and the role of degree exponent

Small-world property is the only characteristic in real networks that is well explained by random network models, more specifically, by the Watts-Strogatz model.

[29]https://en.wikipedia.org/wiki/Polynomial
[30]https://en.wikipedia.org/wiki/Logarithmic_growth
[31]http://networksciencebook.com/chapter/4#hubs

I still need to work on these two sections. For now, I will just skip them.

Barabási Ch: 4.3-4.7 Reading Quiz needs to be done again!

5.3 The Barabási-Albert Model

The Barabási-Albert model, or BA model for short, addresses this question: how does a network's degree distribution form a power law?

Two hidden assumptions of the random network model do not hold water in real networks:

1. The random network model assumes that the number of nodes is fixed. In real networks, however, this number continues to grow as new nodes join the network. That is to say, the size of a real network keeps growing.
2. The random network model assumes that a link is generated randomly. However, in real networks, new nodes prefer to link to high-degree nodes. This phenomenon is called **preferential attachment**.

The BA model recognized the coexistence of **network growth** and **preferential attachment** in real networks and proposed a method of generating networks with scale-free properties, i.e., hubs, and a degree distribution that follows a power law.

I am wondering: does a power law distribution already implies that there are hubs in the network?

We begin with m_0 nodes. Each node's degree is no less than 1. Links between these m_0 nodes are arbitrary.

1. At each timestep, a new node i with a degree of m ($m \leq m_0$) is added to the network.

2. The probability that node i connects to an old node, denoted as node j, is **proportional** to the old node's degree k_j (Menczer et al., 2020):

$$\Pi(i,j) = \frac{k_j}{\sum_l k_l} \tag{5.4}$$

The denominator in Eq. (5.4) is the sum of all degrees in the old network, so the sum of all these probabilities is equal to one (Menczer et al., 2020).

I am not sure whether I can explain why the sum of all these probabilities is equal to one this way: Let us say the probability that node i will connect to node 1 is P_1, to node 2, P_2, to node 3, P_3 ..., if we add these probabilities up, it will (and has to) be equal to one.

In Menczer et al. (2020), in Box 5.4, it says that m here is the average degree of the network. I did not quite understand why.

The image 5.4 made me wonder whether my computation of 5.4 was wrong. I will check it later.

Update: Yeah, i was wrong. Originally, when calculating `deg.dist_internet`, I set `cumulative` to be `FALSE`. This is due to the fact that I did not quite understand power law distribution at that time. I need to set it to be `TRUE`.

5.3.1 Improving the BA model

The following is largely from the online course of INFO 606[34] at Indiana University taught by Professor Yong-Yoel ("YY") Ahn[35]. My notes are accompanied.

Suppose you are an incoming node and are about to attach yourself to other nodes. According to Eq. (5.4), you first should calculate all the degree nodes in the existing network and then calculate the proportion of a specific node's degree against the sum of all the degrees. This proportion will be the probability that you will connect to this node.

There is a problem in the first step. You need to know all the nodes before you can get the sum of all degrees. However, this is very difficult. For instance, when you enter a university, and are about to develop a friendship with other people, is it realistic for you to first know how many friends **EVERY** person at this university has? Probably not.

Is it possible for us to achieve preferential attachment without requiring knowledge about each and every node in the existing network?

Yes. Try this: The incoming node randomly chooses a link from the existing network, then connect to either end of this link. Hubs have more links connecting to it, so it is more likely for this incoming node to connect to a hub. If you have difficulty understanding why so, consider the extreme example of a star graph.

```
starG <- make_star(n = 12, mode = "undirected", center = 1 )
plot (starG)
```

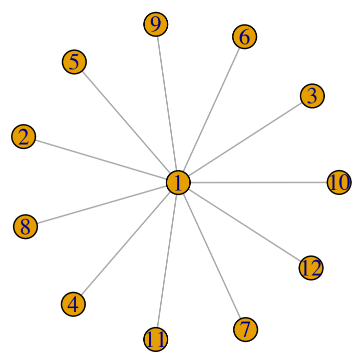


FIGURE 5.5: A star graph

In Fig. 5.5, no matter which link you pick, it is always connected to the hub.

If you have followed the understood the logic so far, you should be able to know that if $k_i = 2 \cdot k_j$ (meaning node i has twice as many links as node j), then the incoming node is twice as likely to connect to node i. This means that the probability of connecting to node i is linearly **proportional** to k_i, which is how the BA model defines preferential attachment.

Professor YY said at the end of the video that this process can also result in a high clustering coeffient of the network. I need to brush up on clustering coeffient a little bit to understand explain why.

Bibliography

Centola, D. and Macy, M. (2007). Complex contagions and the weakness of long ties. *American journal of Sociology*, 113(3):702–734.

Donovan, S. A., Labonte, M., and Dalaker, J. (2016). The us income distribution: Trends and issues. *CRS Report, US Congressional Research Service, Washington, DC*.

Granovetter, M. S. (1973). The strength of weak ties. *American journal of sociology*, 78(6):1360–1380.

Menczer, F., Fortunato, S., and Davis, C. A. (2020). *A First Course in Network Science*. Cambridge University Press.

Newman, M. E. (2005). Power laws, pareto distributions and zipf's law. *Contemporary physics*, 46(5):323–351.