Graphs and Networks in Data Science
Lecture Notes

Hanbaek Lyu

DEPARTMENT OF MATHEMATICS, UNIVERSITY OF WISCONSIN - MADISON, WI 53706

Email address: hlyu@math.wisc.edu

WWW.HANBAEKLYU.COM

Contents

Chapter 1. Introduction to Networks and Data Science 4
1.1. Network Science 4
1.1.1. Networks are everywhere 4
1.1.2. Examples of real-world networks 4
1.1.3. How does a network science work? 7
1.2. Networks in Machine Learning and Data Science 8
1.2.1. A quick bite on Machine Learning 8
1.2.2. Graph Machine Learning 11

Chapter 2. Essentials in Graph theory and Applications 13
2.1. Graphs and digraphs 13
2.2. Common families of graphs 18
2.3. Connectedness, trees, and forests 21
2.4. Bipartite graphs 22
2.5. Euler's theorem and Hamiltonian cycles 25
2.6. Matchings in bipartite graphs 29
2.7. Menger's theorem 33
2.8. Digraphs and network flows 36
2.9. Applications of graphs to image segmentation 40
2.9.1. Background and introduction 40
2.9.2. Example 42

Chapter 3. Essentials in Network models 46
3.1. Properties of real-world networks 46
3.1.1. Six degrees of separation: Short average path length 46
3.1.2. Existence of hub nodes: Power-law degree distribution 48
3.1.3. Communities 48
3.2. Erdős-Renyí random graphs 49
3.3. Small worlds 55
3.3.1. Clustering coefficient 55
3.3.2. Watts-Strogatz model 56
3.4. Preferential attachment 59
3.5. Random walk attachment model 64
3.6. Configuration model 67
3.6.1. Definition of the model and applications 68
3.6.2. Theoretical analysis on the configuration model 71
3.7. Stochastic block model 74
3.7.1. Communities 74
3.7.2. Basics of SBM 74
3.7.3. Spectral clustering 76

Chapter 4. Essential Algorithms on Networks 82

4.1. Random walks on graphs 82
4.1.1. Motivation: Sampling subgraphs from sparse networks 82
4.1.2. Elementary results on RWs on graphs 84
4.1.3. Basics of Markov chains 86
4.1.4. PageRank and Centrality 91
4.2. Classical algorithms on networks 96
4.2.1. Breath-first-search and Depth-first-search 96
4.2.2. Min-cost trees 99

Chapter 5. Graph representation learning 104
5.1. Euclidean embedding of graphs 104
5.1.1. Basics of node embedding 104
5.1.2. Adjacency/Laplacian spectral embedding 105
5.1.3. DeepWalk 106

Appendix A. Background in probability theory 110
A.1. Discrete random variables 110
A.2. Binomial, geometric, and Poisson RVs 111
A.3. Continuous Random Variables 113
A.4. Uniform, exponential, and normal RVs 114
A.5. Expectation and variance of sums of RVs 116
A.6. Conditional expectation 117
A.7. Elementary limit theorems 119
A.8. Bounding tail probabilities 121
A.9. Definition and Examples of MLE 124

Appendix. Bibliography 127

CHAPTER 1

Introduction to Networks and Data Science

1.1. Network Science

1.1.1. Networks are everywhere. Many important natural and social phenomena – the formation of public opinion, trending topics on social networks, growth of the stock market, development of cancer cells, the outbreak of epidemics, and anomaly detection in power grids – are closely related to understanding the large-scale collective behavior of complex dynamical systems on *networks*. In such a system, entities (e.g., individuals, molecules, accounts, authors) interact with 'nearby' entities connected through edges representing various relationships (e.g., friendship, physical contact, retweets, collaboration). Examples are pervasive everywhere from nature to the social networks as well as our everyday life (e.g., telecommunications, computer systems, biological structures [CW03, RMLF10, SKF04, RPD14, Alo07], economics [OTT10, TKWH18], cognitive and semantic frameworks, and social interactions [HlHbCfP14, JKG09]).

Due to the ubiquity of networks in diverse fields, a huge amount of data is being produced and encoded in the form of networks. Network data refers to data that represents the relationships and interactions between entities, where individual entities are represented as nodes, and their connections are expressed as edges. This data type is used to model various systems and phenomena, including social networks, transportation networks, biological interactions, internet connectivity, and more. Network science involves studying the structure, patterns, and dynamics of these interconnected systems, using techniques such as graph theory and network analysis to extract meaningful insights based on the relationships within the network. A cornerstone of modern network science is that many seemingly unrelated networks in nature turned out to share similar structures (e.g., sparsity, power-law degree distribution, and small-world property).

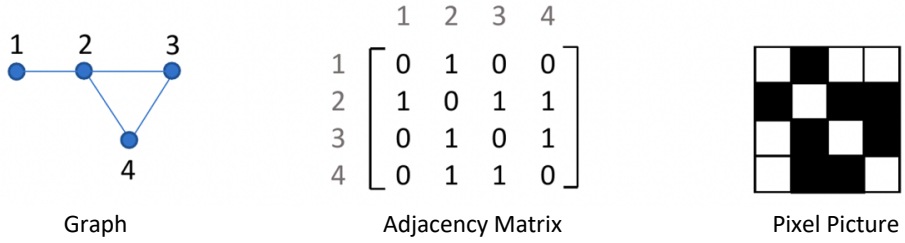


FIGURE 1.1.1. Various representations of a network

1.1.2. Examples of real-world networks. Here we show some examples of real-world networks. Most real-world networks are *sparse*, meaning that if one picks two nodes uniformly at random, then it is highly unlikely that there is an edge between the two nodes. Yet they have rich 'structures' in various 'scales'. We will learn what we mean by structures in networks at different scales later on.

(1) `CORONAVIRUS PPI` (with the shorthand `CORONAVIRUS`): This connected network is curated by `theBiogrid.org` [OSB+19, the20, GJB+20] from 142 publications and preprints.

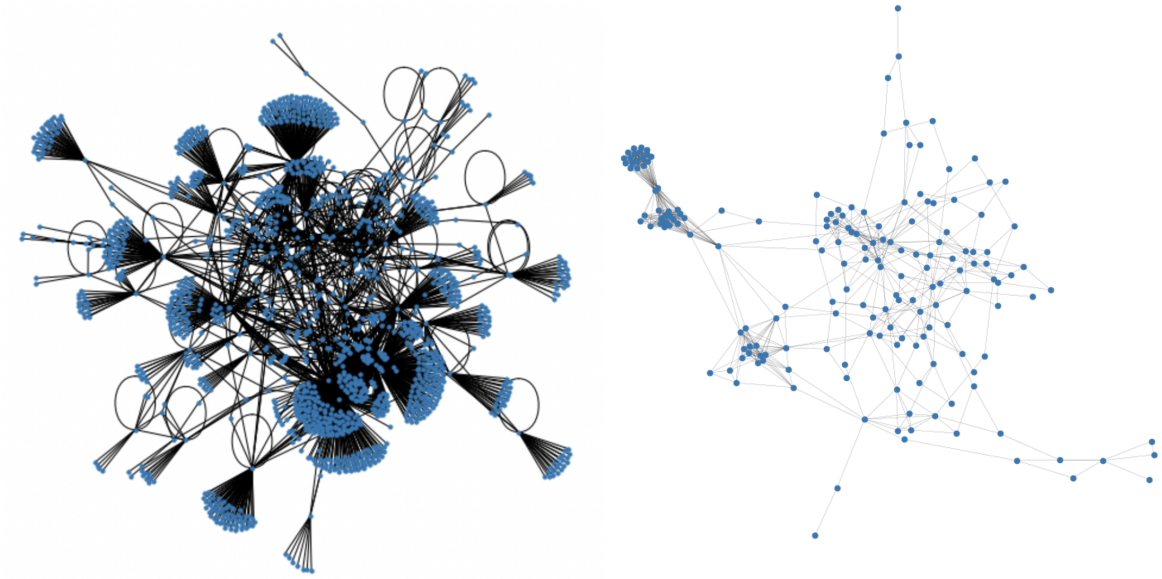


FIGURE 1.1.2. (Left) Coronavirus protein-protein network. (Right) A 200-node subgraph of the astrophysics arXiv collaboration network



FIGURE 1.1.3. (Left) US power grid network. (Right) CS Ph.D. collaboration network.

It has 1,536 proteins that are related to coronaviruses and 2,463 protein-protein interactions (in the form of physical contacts) between them. This network is the largest connected component of the Coronavirus PPI network that we downloaded on 24 July 2020; in total, there are 1,555 proteins and 2,481 interactions. Of the 2,481 interactions, 1,536 of them are for SARS-CoV-2 and were reported by 44 publications and preprints; the rest are related to coronaviruses that cause Severe Acute Respiratory Syndrome (SARS) or Middle Eastern Respiratory Syndrome (MERS).

(2) `ARXIV ASTRO-PH` (with the shorthand `ARXIV`) [LK20, GL16]: This network has 18,722 nodes and 198,110 edges. Its largest connected component has 17,903 nodes and 197,031

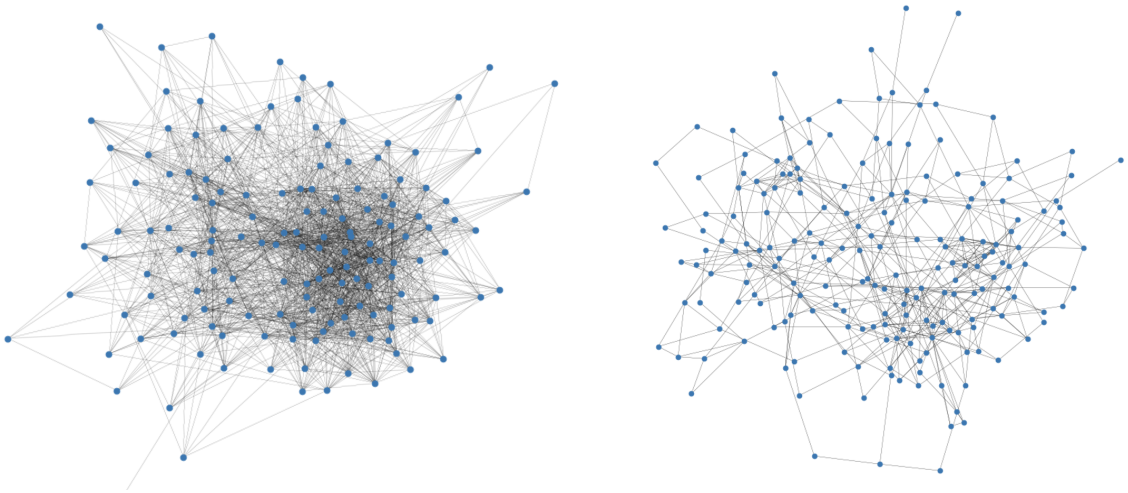


FIGURE 1.1.4. (Left) A 200-node subgraph of CALTECH facebook network. (Left) A 200-node subgraph of UCLA Facebook network.

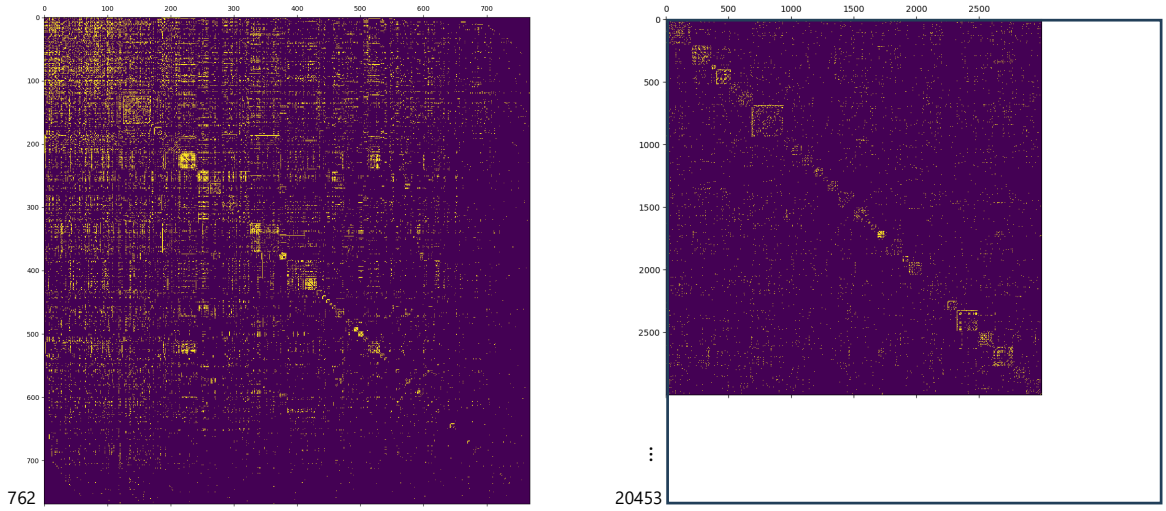


FIGURE 1.1.5. Plot of adjacency matrices of CALTECH and UCLA Facebook networks. Only a portion of the adjacency matrix of UCLA is shown. Yellow = 1 (adjacent) and blue = 0 (non-adjacent).

edges. We use the complete network in our experiments. This network is a collaboration network between authors of astrophysics papers that were posted to the ArXiv preprint server. The nodes represent scientists and the edges indicate coauthorship relationships. This network has 60 self-edges; these edges encode single-author papers.

(3) CALTECH This connected network, which is part of the FACEBOOK100 data set [TMP12] (and which was studied previously as part of the FACEBOOK5 data set [RKMP11]), has 762 nodes and 16,651 edges. The nodes represent user accounts in the Facebook network of Caltech on one day in the fall of 2005, and the edges encode Facebook 'friendships' between these accounts.

(4) UCLA: This connected network, which is part of the FACEBOOK100 data set [TMP12], has 20,453 nodes and 747,604 edges. The nodes represent user accounts in the Facebook network of UCLA on one day in the fall of 2005, and the edges encode Facebook 'friendships' between these accounts.

Exercise 1.1.1 (Why does a real-world social network have to be sparse?)**.** Let G be a social network with n nodes (a node=individual). Two nodes can be connected (friends) or not (not friends), and there is no orientation on the edges. Suppose each person can have at most 500 friends.

(i) What is the number of all possible edges in G? $\binom{n}{2}$

(ii) How many edges can G have at most?

(iii) The *edge density* of G is the total number of edges in G divided by the total number of all possible edges. Show that

$$\text{Edge density of } G \leq \frac{500n}{n(n-1)} = \frac{500}{n-1}.$$

(iv) UW-Madison has admitted $n = 7,550$ freshmen in year 2023. Suppose that each freshman can have at most 100 friends among the freshmen. Deduce that

$$\text{Edge density of the UW Freshmen social network} \leq \frac{100}{7549} \approx 0.0132.$$

This means that if we randomly pick two freshmen, then the chance that they know each other is about 1.3%.

(v) Can you give a reason why the US road network may be sparse?

Network	Type	Nodes (N)	Links (L)	Density (d)	Average degree ($\langle k \rangle$)
Facebook Northwestern Univ.		10,567	488,337	0.009	92.4
IMDB movies and stars		563,443	921,160	0.000006	3.3
IMDB co-stars	W	252,999	1,015,187	0.00003	8.0
Twitter US politics	DW	18,470	48,365	0.0001	2.6
Enron email	DW	87,273	321,918	0.00004	3.7
Wikipedia math	D	15,220	194,103	0.0008	12.8
Internet routers		190,914	607,610	0.00003	6.4
US air transportation		546	2,781	0.02	10.2
World air transportation		3,179	18,617	0.004	11.7
Yeast protein interactions		1,870	2,277	0.001	2.4
C. elegans brain	DW	297	2,345	0.03	7.9
Everglades ecological food web	DW	69	916	0.2	13.3

FIGURE 1.1.6. Summary statistics of some real-world networks. Table excerpted from [MFD20].

1.1.3. How does a network science work? Ideally, one would have a very good understanding of the network if one knows how to 'generate it'[1]. That is, if there is a 'random network model' that generates a network and if one can fit that model to the observed network so that there is a high chance that a randomly generated network from that model is similar to the observed one, then it is reasonable to say that the random network model is a good explanation of the observed network.

However, there are millions of interesting structures inside a network and a single network can exhibit diverse characteristics[2]. Hence in most cases, it is not ideal to expect that there is some magical random network model that can almost perfectly explain every single feature of

[1] Richard Feynman, the theoretical physicist who received the Nobel prize in 1965 for his work developing quantum electrodynamics, once famously said "What I cannot create, I do not understand"

[2] Note that there are more than one billion non-isomorphic connected simple graphs with 11 nodes. See McKay, B. Graphs

the observed network. Instead, we first choose some important structure or statistic of a network that we would like to explain and seek random network models that can explain those chosen characteristics. Finding and computing some network characteristic (or observables) can give meaningful information on the network (just like the simple edge density computation in Exc. 1.1.1)[3]. This pipeline is summarized below.

(1) Detect some network structure of interest
- e.g., communities, hubs, cliques, cycles, spanning trees, Hamiltonian paths, network motifs, chromatic number, Hadwiger number, max-flow, min-cut, treewidth

(2) Compute various statistics
- e.g., degree distribution, edge density, diameter, clustering coefficients

(3) Fit a random network model
- e.g., stochastic block model, preferential attachment, configuration model, Watt-Strogatz model

Figures 1.1.7 and 1.1.8 illustrate the above pipeline for the network structures of community structure' (i.e., social circles) and 'hubs' (i.e., popular students).

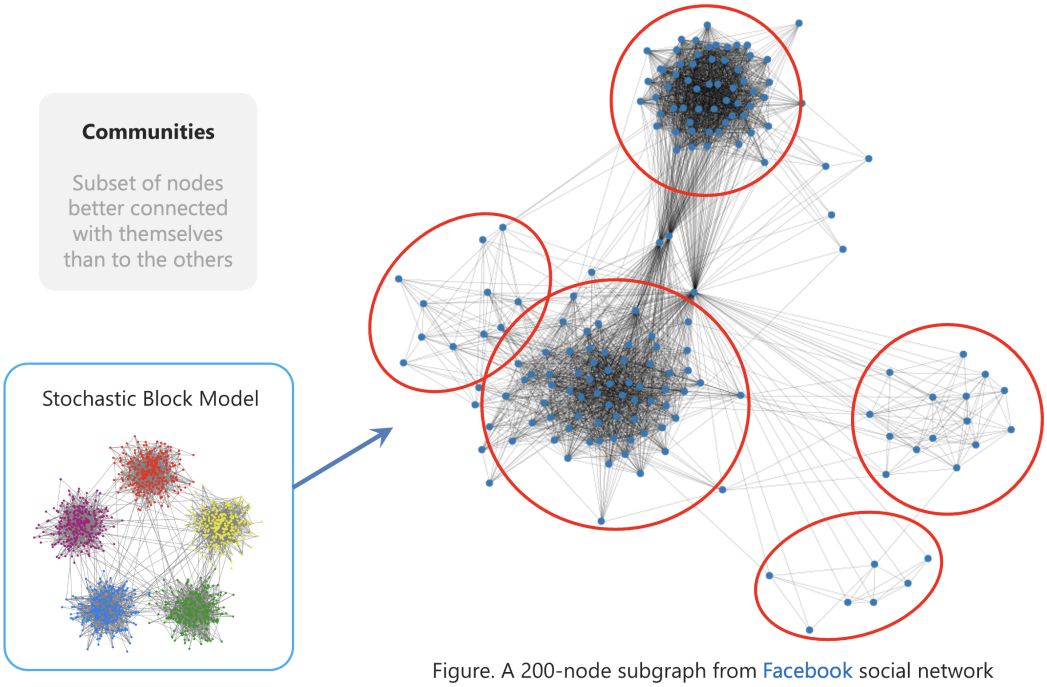


FIGURE 1.1.7. Schematic of community detection

1.2. Networks in Machine Learning and Data Science

1.2.1. A quick bite on Machine Learning. When do we say we learned something, say, how to ride bikes or how to multiply two fractions? Roughly speaking, this is to *recognizing patterns* between input and output so that we can use the recognized patterns to *predict* the output of fresh new input. *Machine Learning* (ML) consists of techniques that enable computers to this learning process. Below are some key components of ML.

[3]Sometimes computing a desired observable could be very difficult to compute (e.g., chromatic number).

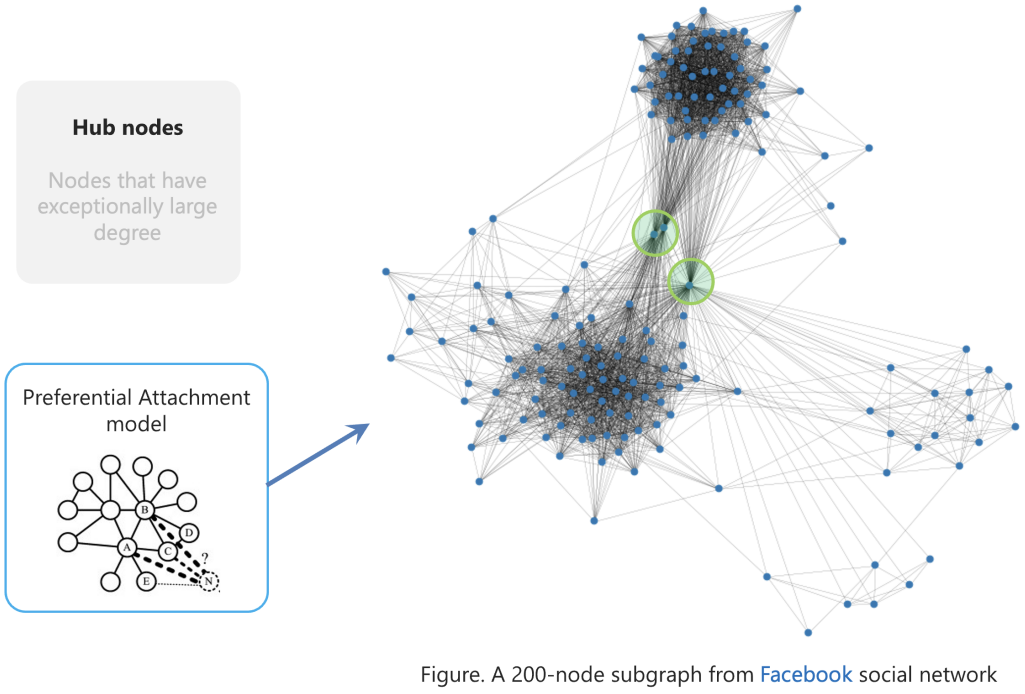


FIGURE 1.1.8. Schematic of hub detection

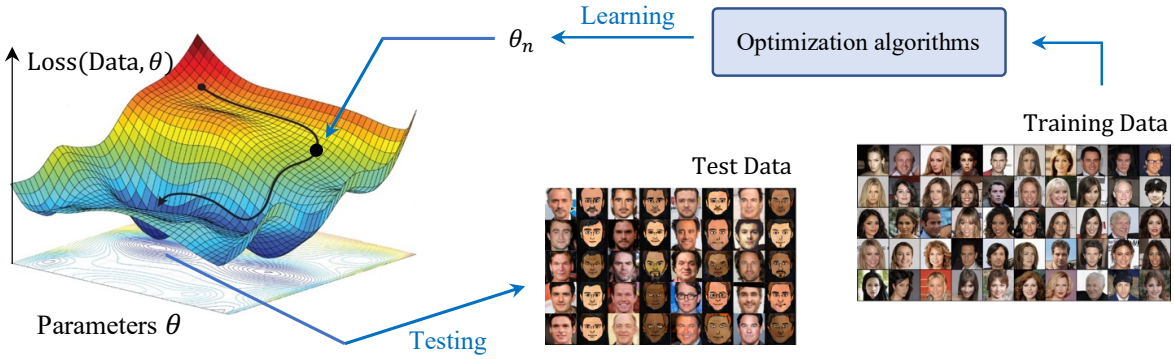


FIGURE 1.2.1. Basic scheme of Machine Learning: (1) Select a model to explain the dataset; (2) Fit the parameters to the training dataset using an optimization algorithm; (3) Test the performance of the fitted model to the test dataset.

1. **Model Selection:** The goal is to predict unknown output $\mathbf{y}$ from a new input $\mathbf{x}$. For this purpose, choose a model $\mathbf{y} \approx \hat{\mathbf{y}}(\mathbf{x}, \boldsymbol{\theta})$ with parameters $\boldsymbol{\theta}$[4]. Given input $\mathbf{x}$ and parameters $\boldsymbol{\theta}$, the model will output the value $\hat{\mathbf{y}}$ that should be close to the true and unknown $\mathbf{y}$.
   e.g., Linear regression, Logistic regression, PCA, NMF, feedforward neural networks, CNN, RNN, etc.
2. **Training Data:** We need a large number of known input and output pairs $(\mathbf{x}_1, \mathbf{y}_1), (\mathbf{x}_2, \mathbf{y}_2), \ldots, (\mathbf{x}_n, \mathbf{y}_N)$, from which we want to recognize patterns between the variables $\mathbf{x}$ and $\mathbf{y}$.
   e.g., vectors, matrices, tensors, images, videos, texts, networks, etc.

[4]Model parameters are usually denoted by θ in statistics and by w (for 'weights') in machine learning. We will use both interchangeably.

3. Training the model: We need to find the optimal parameter $\hat{\boldsymbol{\theta}}$ for which the model $\mathbf{x} \mapsto \hat{\mathbf{y}}(\mathbf{x}, \hat{\boldsymbol{\theta}})$ gives a good fit for the training data. More precisely, this is done by solving a following optimization problem

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta} \in \boldsymbol{\Theta}}{\arg\min}\, \ell(\text{training data}, \boldsymbol{\theta}), \tag{1}$$

where ℓ is a *loss function* that measures the error between the true output y and the predicted output $\hat{\mathbf{y}}$, and $\boldsymbol{\Theta}$ is a set of admissible parameter values. A typical choice of such loss function is the *mean squared error* (MSE)

$$\ell(\text{training data}, \boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^{N} \|\mathbf{y}_i - \hat{\mathbf{y}}(\mathbf{x}_i, \boldsymbol{\theta})\|^2,$$

where 'training data' consists of training examples $(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_N, \mathbf{y}_N)$. Then use *optimizaiton algorithms* to find the optimal parameter $\hat{\boldsymbol{\theta}}$ in (1).

e.g., gradient descent, stochastic gradient descent, expectation-maximization, majorization-minimization, etc.

4. Testing: After an optimial parameter $\hat{\boldsymbol{\theta}}$ is learned, test the 'recognized pattern' $\mathbf{x} \mapsto \hat{\mathbf{y}}(\mathbf{x}, \hat{\boldsymbol{\theta}})$ on *test data* $(\mathbf{x}'_1, \mathbf{y}'_1), \ldots, (\mathbf{x}'_m, \mathbf{y}'_m)$, that were kept hidden during the training step. If the loss $\ell(\text{test data}, \hat{\boldsymbol{\theta}})$ is small, then we have learned the correct pattern successfully.

The above pipeline of ML assumes that the input feature vector $\mathbf{x}$ is a vector. When we wish to analyze data that are not in a vector form, we should choose a proper way to extract feature vectors from them. Below we give an example of this for text data.

Example 1.2.1 (Vectorizing text data)**.** The 20 Newsgroups data set is a collection of approximately 18,000 newsgroup documents, partitioned (nearly) evenly across 20 different newsgroups. The 20 newsgroups collection has become a popular data set for experiments in text applications of machine learning techniques, such as text classification and text clustering[5][6]. Here is an example of raw text in the 'macs.hardware' category:

```
»» 4257 th doc:  Anyone know what would cause my IIcx to not turn on when
I hit the keyboard switch?  The one in the back of the machine doesn't
work either...  The only way I can turn it on is to unplug the machine
for a few minutes, then plug it back in and hit the power switch in the
back immediately...  Sometimes this doesn't even work for a long time...
   I remember hearing about this problem a long time ago, and that a logic
board failure was mentioned as the source of the problem...is this true?
```

After we load the raw documents, we need to extract feature vectors from them. There are two widely used vectorization methods, the *bag-of-words* (BOW) and the *tf-idf* (term-frequency-inverse-document-frequency). We use a standard vocabulary $\mathcal{V}$ of 45534 words provided in sklearn. For a given document $\mathbf{x}$, its BOW and tf-idf feature vectors are defined by

$$\boldsymbol{\phi}_{\text{BOW}}(\mathbf{x}) = \left[\#\text{ of } j\text{th word in } \mathbf{x};\, 1 \le j \le |\mathcal{V}|\right],$$
$$\boldsymbol{\phi}_{\text{tf-idf}}(\mathbf{x}) = \left[\frac{\#\text{ of } j\text{th word in } \mathbf{x}}{\#\text{ of documents that contains the } j\text{th word}};\, 1 \le j \le |\mathcal{V}|\right].$$

Below are the BOW and tf-idf feature vectors of the sample document (4257th) above.

One can think of the tf-idf representation as an inverse-document-frequency (idf) correction to the BOW representation. Namely, even though some words appear many times in a document (e.g., 'The'), if it appears in many other documents, we discount its frequency by the idf. On the

[5] See http://qwone.com/~jason/20Newsgroups/

[6] See also the convenient scikit-learn implementation of this data: 20 Newgroups loader by sklearn

	Coordinate	Bag-of-words	tf-idf
ago	1286	1	0.115122
back	3503	3	0.288116
board	4850	1	0.138195
cause	6355	1	0.125130
either	12551	1	0.108417
failure	14215	1	0.170555
hearing	17842	1	0.170555
hit	18278	2	0.261832
iicx	19200	1	0.217706
immediately	19354	1	0.155037
keyboard	21813	1	0.160377
know	22116	1	0.075488
logic	23500	1	0.158818
long	23525	2	0.203652
machine	23918	2	0.266555
mentioned	25060	1	0.135408
minutes	25592	1	0.145699
plug	30501	1	0.171330
power	30919	1	0.111667
problem	31391	2	0.203382
remember	33429	1	0.119059
sometimes	37293	1	0.135854
source	37380	1	0.127372
switch	39267	2	0.321834
time	40488	2	0.164800
true	41292	1	0.114800
turn	41456	2	0.269503
unplug	42226	1	0.233554
way	43812	1	0.089654
work	44510	2	0.197874

FIGURE 1.2.2. Bag-of-words and tf-idf feature vectors of a sample document from 20Newsgroups.

other hand, if some words appear only a few times in a document (e.g., 'OECD') if it does not appear in many other documents, we boost their frequency (relative to the more widely used words). ▲

1.2.2. Graph Machine Learning. One of the hottest fields in modern machine learning is called *graph machine learning* (GML), which seeks to develop machine learning methods for analyzing network data. A primary purpose of GML is to compress large sparse graph data structures to enable feasible prediction and inference. There are many ways for doing so, from the classical spectral embedding to the more modern DeepWalk [PARS14] (that combines random walk on networks and Word2Vec [MCCD13]) and to Graph Neural Networks [ZCH+20].

Compressing a large sparse network into a low-dimensional vector space is what 'graph representation learning' aims at. This is one of many instances of unsupervised GML. Other instances include clustering and community detection, similarity analysis, pathfinding, graph mining, and so on (see Fig. 1.2.3). "Community detection" identifies groups of densely interconnected nodes within a graph, with applications in anomaly detection, fraud analysis, social network analysis, and biology. "Similarity measurement" involves finding and quantifying similarities between pairs of nodes in a graph, applicable in recommendation systems, entity resolution, and anomaly and fraud detection.

There are also a number of supervised tasks in GML. The standard ones are node property prediction, link prediction, and graph property prediction (see Fig. 1.2.4). "Node property prediction" involves predicting discrete or continuous attributes of nodes, such as classifying financial account fraud or categorizing online retail products. "Link prediction" anticipates the existence of relationships between nodes and predicts attributes of these relationships, valuable for recommendation systems and bioinformatics. "Graph property prediction" predicts discrete or continuous properties of graphs or subgraphs, especially useful in domains where individual networks represent entities, such as material science and bioinformatics.

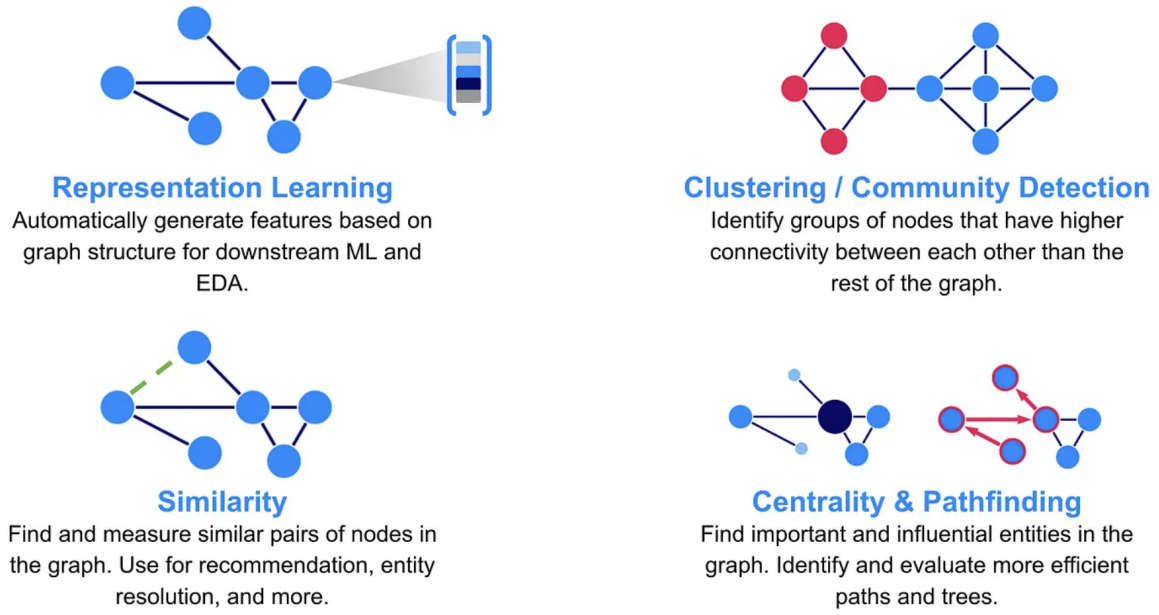


FIGURE 1.2.3. Three instances of unsupervised graph machine learning. Figure credit: Z. Blumenfeld

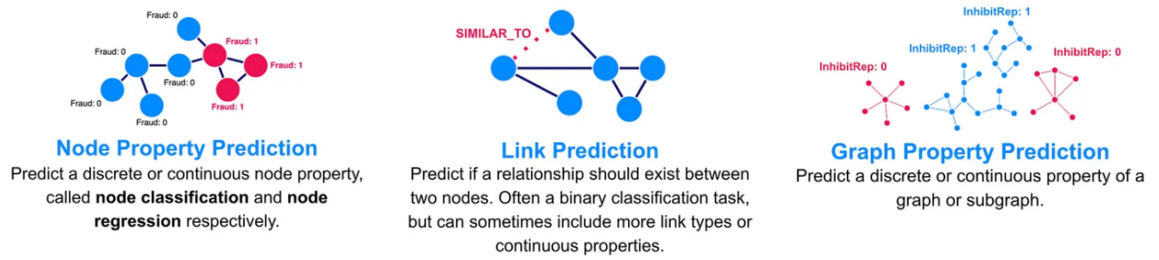


FIGURE 1.2.4. Three instances of supervised graph machine learning. Figure credit: Z. Blumenfeld

Graph machine learning is an exciting new field that combines classical graph theory, spectral graph theory, network science, random networks, and modern machine learning techniques. Throughout this course, we will learn these ingredients at an introductory level and see how they act on actual network data analysis and GML tasks.

CHAPTER 2

Essentials in Graph theory and Applications

2.1. Graphs and digraphs

Graphs are the most basic mathetical objects that represents a network. It consists a set V of nodes (entities in a network) that are connected by edges in the set E (relationships between the entities).

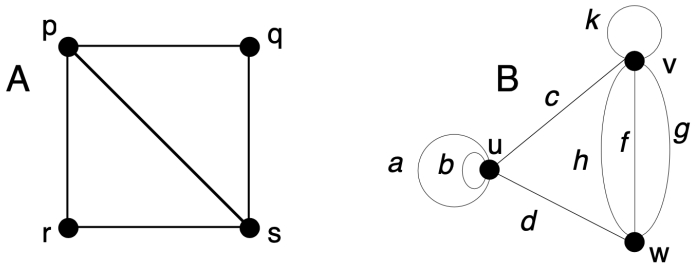


FIGURE 2.1.1. (Left) Simple graph A. (Right) Non-simple graph B with self-loops and multi-edges. Figure excerpted from [GYA18].

Definition 2.1.1 (Graph)**.** A *graph* $G = (V, E)$ is a mathematical structure consisting of two finite sets V and E. The elements of V are called *vertices* (or nodes), and the elements of E are called *edges*. Each edge has a set of one or two vertices associated to it, which are called its *endpoints*. An edge is said to *join* its two endpoints. A vertex joined by an edge to a vertex v is said to be a *neighbor* of v. We say two nodes u, v are *adjacent* if there is an edge e joining them.

Definition 2.1.2. The (open) *neighborhood* of a vertex v in a graph G, denoted $N(v)$, is the set of all the neighbors of v. The *closed neighborhood* of v is given by $N[v] := N(v) \cup \{v\}$.

Example 2.1.3. Consider the graph A in Figure 2.1.1. Its vertex set V_A and edge set E_A are given by

$$V_A = \{p, q, r, s\}, \quad E_A = \{\{p, q\}, \{p, s\}, \{p, r\}, \{q, s\}, \{r, s\}\}.$$

Consider the graph B in Figure 2.1.1. Its vertex set V_B and edge set E_B are given by

$$V_B = \{u, v, w\}, \quad E_B = \{a, b, c, d, k, g, f, h\}.$$

▲

There are different types of undirected edges in graphs. See Definition 2.1.4.

Definition 2.1.4 (Types of undirected edges)**.**

(1) A *proper edge* is an edge that joins two distinct vertices.
(2) A *self-loop* is an edge that joins a single endpoint to itself.
(3) A *multi-edge* is a collection of two or more edges having identical endpoints. The *edge multiplicity* is the number of edges within the multi-edge.
(4) A *simple graph* is a graph with no self-loops and multi-edges.
(5) A *loopless graph* (or multi-graph) may have multi-edges but no self- loops.
(6) A *(general) graph* may have self-loops and/or multi-edges.

Definition 2.1.5 (Degrees of a vertex)**.**

(1) *Adjacent vertices* are two vertices that are joined by an edge.
(2) *Adjacent edges* are two distinct edges that have an endpoint in common.
(3) If vertex v is an endpoint of edge e, then v is said to be *incident* on e, and e is incident on v.
(4) The *degree* (or valence) of a vertex v in a graph G, denoted $\deg(v)$, is the number of proper edges incident on v plus twice the number of self-loops. A vertex of degree d is also called a *d-valent* vertex.
(5) The smallest and largest degrees in a graph G are denoted $\delta_{\min}$ and $\delta_{\max}$ (or $\delta_{\min}(G)$ and $\delta_{\max}(G)$ when there is more than one graph under discussion). Some authors use δ instead of $\delta_{\min}$ and Δ instead of $\delta_{\max}$.
(6) The *degree sequence* of a graph is the sequence formed by arranging the vertex degrees in non-increasing order.

Definition 2.1.6 (Adjacency matrix for undirected graphs)**.** Let $G = (V, E)$ be a simple graph with n nodes. The adjacency matrix A_G is an $n \times n$ binary matrix defined by

$$A_G[i, j] = \mathbf{1}(\text{nodes } i \text{ and } j \text{ are adjacent}),$$

where $\mathbf{1}(\cdot)$ is the indicator function. (So $A_G[i, j] = 1$ if $\{i, j\} \in E$ and 0 otherwise.) For general undirected graphs, we define A_G as

$$A_G[i, j] = (\text{edge multiplicity of } \{i, j\})\mathbf{1}(\text{nodes } i \text{ and } j \text{ are adjacent}),$$

For directed graphs, we define A_G as

$$A_G[i, j] = (\text{edge multiplicity of } (i, j))\mathbf{1}((i, j) \in E),$$

Example 2.1.7 (A Python example of handling a simple graph)**.** In this example, we create a simple graph G with six vertex set $V = \{0,1,2,3,4,5\}$ and edge set $E = \{\{0,1\},\{0,2\},\{0,3\},\{2,5\},\{4,5\}\}$ using NETWORKX in JUPYTER NOTEBOOK.

```
import networkx as nx # for handling graphs/networks
import numpy as np # for basic scientific computing
import matplotlib.pyplot as plt # for plotting

# Create a simple graph object G
G = nx.Graph()

# Start adding edges (cf. Python counts things from 0 not one.)
G.add_edge(0,1)
G.add_edge(0,2)
G.add_edge(0,3)
G.add_edge(2,5)
G.add_edge(4,5)

# plot the graph
fig = plt.figure(figsize=[3,3], constrained_layout=False)
ax = fig.add_subplot()
nx.draw_networkx(G, ax=ax, pos=nx.circular_layout(G), labels={i: i for i in range(6)},
                 node_size=300, node_color="blue", font_size=14, font_color="white")
plt.axis('off');
```

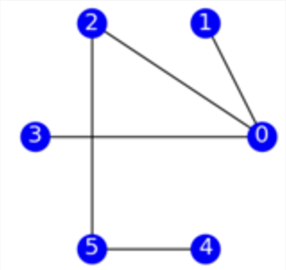


FIGURE 2.1.2. Creating and plotting a simple graph in NETWORKX.

Once a simple graph is created as a NX.GRAPH() object, we can easily compute the degrees of the nodes as well as the degree sequence as in Figure 2.1.3.

We can also easily compute the adjacency matrix and print it out. Also, we can plot it as the pixel picture. See Figure 2.1.4.

▲

```
degree_sequence = []
for v in G.nodes():
    degree_sequence.append(G.degree(v))
    print("node={}, degree={}".format(v, G.degree(v)))
```

```
node=0, degree=3
node=1, degree=1
node=2, degree=2
node=3, degree=1
node=5, degree=2
node=4, degree=1
```

```
#degree_sequence = [d for n, d in G.degree()]
print("degree_sequence=", degree_sequence)
```

```
degree_sequence= [3, 1, 2, 1, 2, 1]
```

```
degree_sequence.sort(reverse=True)
print("degree_sequence_sorted=", degree_sequence)
```

```
degree_sequence_sorted= [3, 2, 2, 1, 1, 1]
```

FIGURE 2.1.3. Computing degrees and degree sequence of the simple graph G.

```
# Get the adjacency matrix of G and print it out

A = nx.adjacency_matrix(G)
print(A.todense())
```

```
[[0 1 1 1 0 0]
 [1 0 0 0 0 0]
 [1 0 0 0 1 0]
 [1 0 0 0 0 0]
 [0 0 1 0 0 1]
 [0 0 0 0 1 0]]
```

```
# plot the adjacency matrix
fig = plt.figure(figsize=[2.5,2.5], constrained_layout=False)
ax = fig.add_subplot()
ax.imshow(A.todense());
```

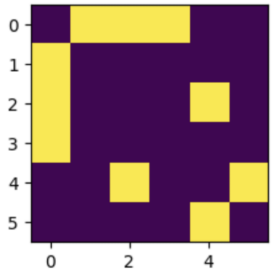


FIGURE 2.1.4. Adjacency matrix of G and its plot.

Definition 2.1.8 (Types of directed edges)**.**

(1) A *directed edge* (or arc) is an edge, one of whose endpoints is designated as the *tail*, and whose other endpoint is designated as the *head*. An arc is said to be *directed from* its tail to its head.
(2) In a general digraph, the head and tail of an arc e may be denoted head(e) and tail(e), respectively.
(3) Two arcs between a pair of vertices are said to be *oppositely directed* if they do not have the same head and tail.
(4) A *multi-arc* is a set of two or more arcs having the same tail and same head. The *arc multiplicity* is the number of arcs within the multi-arc.
(5) A *directed graph* (or digraph) is a graph each of whose edges is directed. A digraph is *simple* if it has neither self-loops nor multi-arcs. In a simple digraph, an arc from vertex u to vertex v may be denoted by uv or by the ordered pair (u,v).

Definition 2.1.9 (Adjacency matrix for undirected graphs)**.** Let $G=(V,E)$ be a directed graph. We define its adjaceny matrix A_G as

$$A_G[i,j] = (\text{edge multiplicity of } (i,j))\mathbf{1}((i,j)\in E),$$

Example 2.1.10 (A Python example of handling a directed graph)**.** In this example, we create a directed graph G with six vertex set $V=\{0,1,2,3,4,5\}$ and edge set $E=\{(0,1),(0,2),(0,3),(2,5),(4,5)\}$ using NETWORKX in JUPYTER NOTEBOOK.

We can also easily compute the adjacency matrix and print it out. Also, we can plot it as the pixel picture. See Figure 2.1.7. Notice that the adjacency matrix of the directed graph G is no longer symmetric as in the undirected graph case.

▲

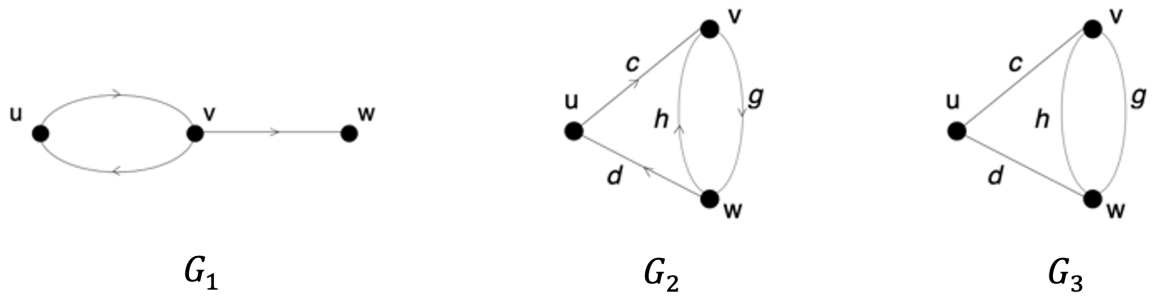


FIGURE 2.1.5. (G_1) Simple digraph with a pair of oppositely directed arcs. (G_2) A digraph. (G_3) The underlying graph of the digraph G_2.

```
# Create a digraph object G
G = nx.DiGraph()

# Start adding edges (cf. Python counts things from 0 not one.)
# The edges are now directed.
G.add_edge(0,1)
G.add_edge(0,2)
G.add_edge(0,3)
G.add_edge(2,5)
G.add_edge(4,5)

# plot the graph
fig = plt.figure(figsize=[3,3], constrained_layout=False)
ax = fig.add_subplot()
nx.draw_networkx(G, ax=ax, pos=nx.circular_layout(G), labels={i: i for i in range(6)},
                 node_size=300, node_color="blue", font_size=14, font_color="white")
plt.axis('off');
```

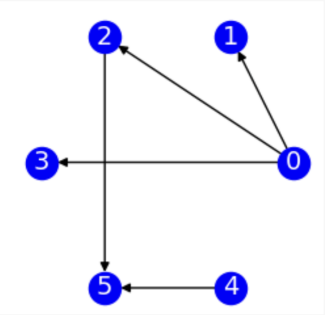


FIGURE 2.1.6. Creating and plotting a simple graph in NETWORKX.

```
# Get the adjacency matrix of G and print it out

A = nx.adjacency_matrix(G)
print(A.todense())
```

```
[[0 1 1 1 0 0]
 [0 0 0 0 0 0]
 [0 0 0 0 1 0]
 [0 0 0 0 0 0]
 [0 0 0 0 0 0]
 [0 0 0 0 1 0]]
```

```
# plot the adjacency matrix
fig = plt.figure(figsize=[2.5,2.5], constrained_layout=False)
ax = fig.add_subplot()
ax.imshow(A.todense());
```

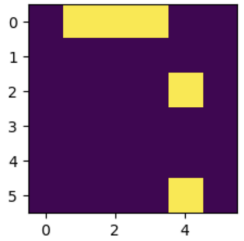


FIGURE 2.1.7. Adjacency matrix of G and its plot.

Graph theory as a mathematical discipline is regarded to begin with the work of Leonhard Euler (1707–1783). The following theorem of Euler establishes a fundamental relationship between the degrees and the number of edges of a graph.

Theorem 2.1.11 (Euler)**.** *Let $G = (V, E)$ be a graph. Then*

$$\sum_{v \in V} \deg_G(v) = 2|E|. \tag{2}$$

PROOF. The LHS of (2) counts each edge in G twice. (Note that by definition, a self-loop is double-counted by the degree function, preciesely to make this double-counting work for general graphs.) □

Exercise 2.1.12. Let $G = (V, E)$ be a graph. Show that there must be an even (including zero) number of nodes of odd degree. (*Hint: Use Theorem 2.1.11.*)

We will start to make some basic observations of graphs.

Proposition 2.1.13. *A non-trivial simple graph G must have at least one pair of vertices whose degrees are equal.*

PROOF. Suppose G has n nodes. Since G is simple, the degrees of the nodes in G takes values in $\{0, 1, 2, \ldots, n-1\}$. However, there cannot be both a node of degree 0 and a node of degree $n-1$ (why?). Thus, there are only $n-1$ distinct values that are possible for the degrees of the n nodes in G. Hence, by the pigeonhole principle, some value of the degree must appear at least twice in the degree sequence of G. □

Proposition 2.1.14. *Suppose $d_1 \geq \cdots \geq d_n$ is a sequence of nonnegative integers whose sum is even. Then there exists a graph $G = (V, E)$ (not necessarily simple) whose degree sequence is exactly $(d_1, d_2, \ldots, d_n)$.*

PROOF. Start with a graph G of n nodes, $v_1, \ldots, v_n$, with no edges. We will suitably add edges to G so that we match the desired degree sequence. The first node, v_1, is supposed to have degree d_1. If d_1 is even, then draw $d_1/2$ self-loops at v_1. Otherwise, draw $(d_1 - 1)/2$ self-loops at v_1. Since the degree function double-counts self-loops, this gives, for node v_1, degree d_1 if d_1 is even and $d_1 - 1$ otherwise. Repeat the same for all other nodes. Since $\sum_{i=1}^{n} d_i$ is even, there are even number of i's such that d_i is odd. Let $i_1, i_2, \ldots, i_{2k}$ denote such indices for some $k \geq 0$. Then add a single edge between the nodes in each of the following k pairs: $(v_{i_1}, v_{i_2}), (v_{i_3}, v_{i_4}) \ldots, (v_{i_{2k-1}}, v_{i_{2k}})$. Then the resulting graph has degree sequence $(d_1, d_2, \ldots, d_n)$. See Fig. 2.1.8 for an illustration. □

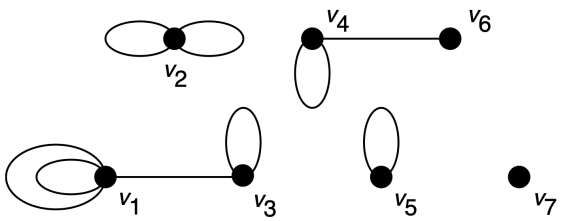


FIGURE 2.1.8. Constructing a graph with degree sequence $(5, 4, 3, 3, 2, 1, 0)$. Figure excerpted from [GYA18].

Note that the construction of a graph that realizes a given nonnegative sequence of integers $d_1 \geq \cdots \geq d_n$ of even sum in the proof of Prop. 2.1.14 as a degree sequence relies heavily on having self-loops. It is possible to realize the same sequence as the degree sequence of a *simple graph*? It turns out that not every sequence can be the degree sequence of a simple graph. We thus introduce the following notion:

Definition 2.1.15 (Graphic sequence)**.** A sequence $d_1 \geq \cdots \geq d_n$ of nonnegative integers is said to be *graphic* if there exists a simple graph G with n nodes $v_1, \ldots, v_n$ s.t. $\deg(v_i) = d_i$ for $i = 1, \ldots, n$.

Exercise 2.1.16. **(i)** Show that $(4, 3, 2, 2, 1)$ is a graphic sequence.
(ii) Show that $(4, 3, 2, 1, 1)$ is not a graphic sequence.
(iii) Show that $(4, 3, 2, 1)$ is not a graphic sequence.

The celebrated Erdős-Gallai condition completely characterizes a sequence of nonnegative integers to be graphic.

Theorem 2.1.17 (Erdős and Gallai [EG60])**.** *For a sequence of nonnegative integers $d_1 \geq \cdots \geq d_n$,*

$$(d_1, d_2, \ldots, d_n) \text{ is graphic} \iff \sum_{i=1}^{k} d_i \leq k(k-1) + \sum_{i=k+1}^{n} \min(d_i, k) \text{ for all } k = 1, \ldots, n.$$

(The condition on the RHS is called the Erdős-Gallai condition.)

PROOF. ($\Longrightarrow$) Suppose $(d_1, \ldots, d_n)$ is graphic. Then there exists a simple graph $G = (V, E)$ with nodes $v_1, \ldots, v_n$ such that $\deg(v_i) = d_i$ for $i = 1, \ldots, n$. Choose arbitrary $k \in \{1, \ldots, n\}$. The case when $k = n$ is trivial (sum of degrees $\leq n(n-1)$) so we may assume $k \leq n-1$. Divide the node set V into $A = \{v_1, \ldots, v_k\}$ and $B = \{v_{k+1}, \ldots, v_n\}$. We will be counting 'half-edges'. There are $d_1 + \cdots + d_k$ half-edges incident to $v_1, \ldots, v_k$. At most $k(k-1)$ of them could end at some other node in A. The rest should end at some node in B. But for each node v_j in B, there can be at most $\max(d_j, k)$ half-edges from v_j that can terminate at some node in A. Thus the inequality in the EG condition follows. This holds for arbitrary $k \in \{1, \ldots, n-1\}$, so this shows the desired implication. (See Fig. 2.1.9 for an illustration.)

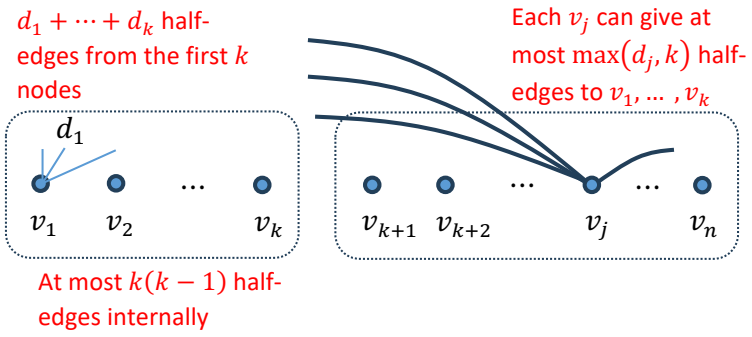


FIGURE 2.1.9. Graphic implies EG condition

($\Longleftarrow$) This direction is more difficult. A recent short proof is given in [TVW10]. The proof proceeds like this. Say a 'subrealization' of the sequence $(d_1, \ldots, d_n)$ is a graph G with n nodes $v_1, \ldots, v_n$ such that $\deg(v_i) \leq d_i$ for all $i = 1, \ldots, k$. The proof proceeds by starting with a graph G with n nodes and no edges, which is clearly a subrealization of the sequence. Define the 'critical index' r to be the largest index such that $d(v_i) = d_i$ for $1 \leq i < r$. Initially $r = 1$ unless $(d_1, \ldots, d_n) = (0, \ldots, 0)$, in which case we are done. While $r \leq n$, we will add an edge incident to v_r with possible rewiring of the existing edges to obtain a new subrealization G' with more edges. After each step, we satisfy:

(a) Maintain the degrees of the first $r-1$ nodes $d_1, \ldots, d_{r-1}$ (they are already saturated);
(b) The 'difficiency' $d_r - \deg(v_r)$ is smaller;
(c) $S = \{v_{r+1}, \ldots, v_n\}$ is an independent set (i.e., no edges in between).

See the reference for more details. □

2.2. Common families of graphs

Definition 2.2.1 (Complete graphs)**.** A *complete graph* is a simple graph such that every pair of vertices is joined by an edge. Any complete graph on n vertices is denoted K_n.

Definition 2.2.2 (Regular graphs)**.** A *regular graph* is a graph whose vertices all have equal degrees. A k-regular graph is a regular graph whose common degree is k.

Example 2.2.3 (Platonic solids)**.** The five regular polyhedra illustrated in Figure 2.2.2 are known as the platonic solids. Their vertex and edge configurations form regular graphs called the *platonic graphs*.

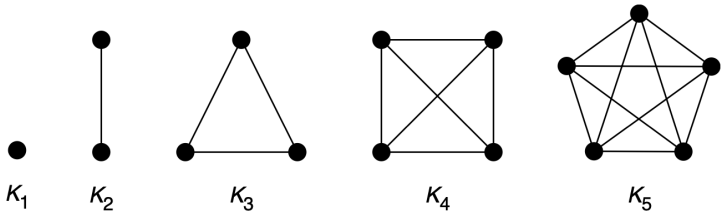


FIGURE 2.2.1. The first five complete graphs. Figure excerpted from [GYA18].

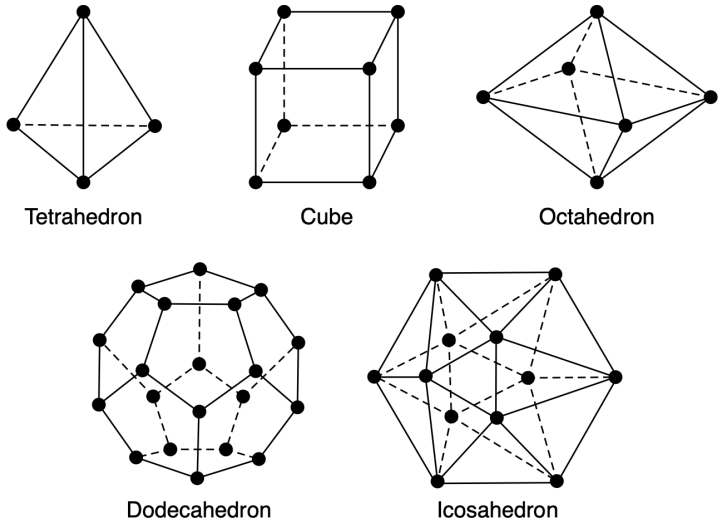


FIGURE 2.2.2. The five platonic graphs [GYA18].

▲

Exercise 2.2.4 (Euler characteristic for platonic graphs)**.** For all five platonic graphs in Example 2.2.3, verify the following *Euler characteristic* formula:

$$|V| - |E| + |F| = 2,$$

where F denotes the set of all faces of the corresponding platonic solid. For example, $|F| = 4$ for tetrahedron and $|F| = 6$ for cube.

Example 2.2.5 (Peterson graph)**.** The *Petersen graph* is the 3-regular graph with 10 nodes and 15 edges represented by the drawings in Figure 2.2.3. Because it possesses a number of interesting graph-theoretic properties, the Petersen graph is frequently used both to illustrate established theorems and to test conjectures. See https://mathworld.wolfram.com/PetersenGraph.html for more information.

▲

Definition 2.2.6 (Different drawings of the same graph)**.** Two graphs $G = (V, E)$ and $G' = (V', E')$ are said to be *isomorphic* if there is a one-to-one mapping $\phi : V \to V'$ between the node sets such that $uv \in E$ if and only if $\phi(u)\phi(v) \in E'$. In other words, there should be a 'renaming function' ϕ that preserves all edges between the two graphs. Such a map ϕ is called the (graph) *isomorphism* from G and G'.

Example 2.2.7. Consider the first drawing of the Peterson graph in Figure 2.2.3**A** with node set $V = \{1, \ldots, 10\}$. This graph is isomorphic to the graph in Figure 2.2.3**C**. With the node labeling shown in Figure 2.2.3**C**, they have exactly the same set of edges $E = \{\{1,6\}, \{1,2\}, \{1,5\}, \ldots\}$.

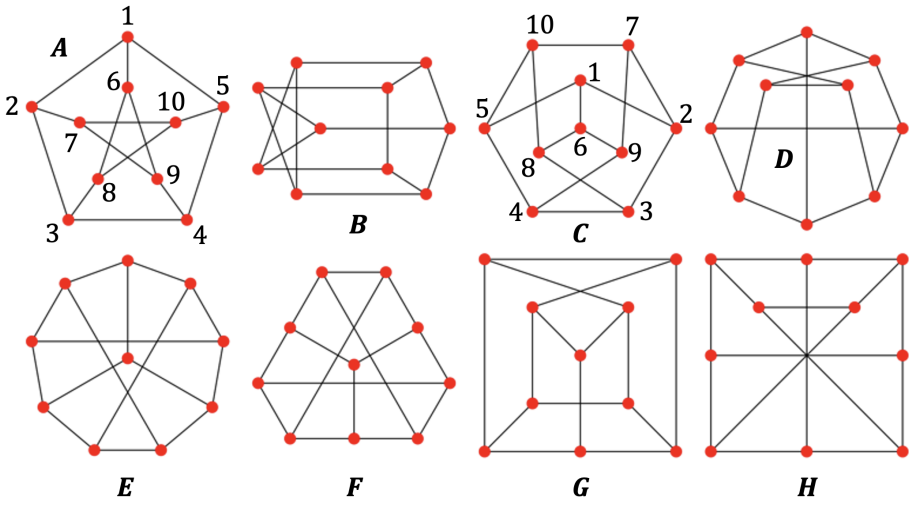


FIGURE 2.2.3. Eight different drawings of the Peterson graph.

Exercise 2.2.8 (Isomorphism is an equivalence relation)**.** Let X be a set. A relation $\sim$ between two elements of X is said to be an *equivalence relation* if it satisfies the following three properties:

(i) (Reflexivity) $x \sim x$ for all $x \in X$;
(ii) (Symmetry) $x \sim y$ implies $y \sim x$ for all $x, y \in X$;
(iii) (Transitivity) $x \sim y$ and $y \sim z$ imply $x \sim z$ for all $x, y, z \in X$.

Show that the (graph) isomorphism defines an equivalence relation on the set of all finite graphs.

Exercise 2.2.9. Show that all eight graphs in Figure (2.2.3) are isomorphic.

Definition 2.2.10 (Paths)**.** A *path graph* P is a simple graph with $|V(P)| = |E(P)| + 1$ that can be drawn so that all of its vertices and edges lie on a single straight line. A path graph with n vertices and $n-1$ edges is denoted P_n.

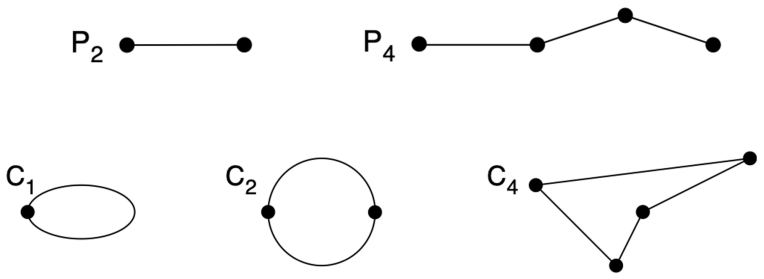


FIGURE 2.2.4. Paths P_2 and P_4 and cycles C_1, C_2, and C_4.

Definition 2.2.11 (Cycles)**.** A *cycle graph* is a single vertex with a self-loop or a simple graph C with $|V(C)| = |E(C)|$ that can be drawn so that all of its vertices and edges lie on a single circle. An n-vertex cycle graph is denoted C_n.

Definition 2.2.12 (Line graph)**.** The line graph $L(G)$ of a graph G has a vertex for each edge of G, and two vertices in $L(G)$ are adjacent if and only if the corresponding edges in G have a vertex in common.

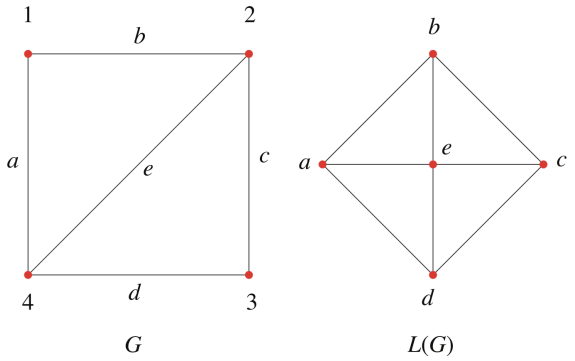


FIGURE 2.2.5. A graph G and its line graph $L(G)$.

Definition 2.2.13 (Subgraphs)**.** Let $G = (V, E)$ and $H = (V', E')$ be two graphs. We say H is a subgraph of G and write $H \subseteq G$ if $V' \subseteq V$ and $E' \subseteq E$. We say H is an *induced subgraph* of G if $H \subseteq G$ and for each pair of nodes (u, v) in H, $\{u, v\} \in E'$ if and only if $\{u, v\} \in G$.

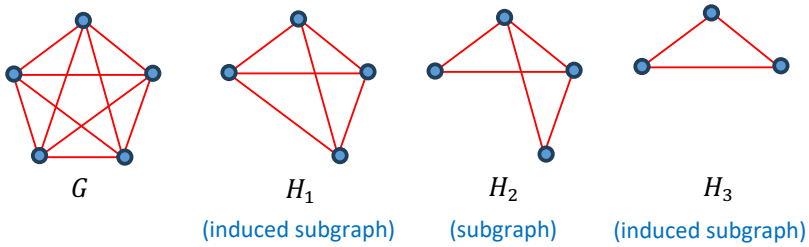


FIGURE 2.2.6. H_1, H_3 are induced subgraphs of G. H_2 is a subgraph of G that is not an induced subgraph.

2.3. Connectedness, trees, and forests

Definition 2.3.1 (Walks and paths)**.** Let $G = (V, E)$ be a directed graph. A (directed) *walk* is a sequence $\mathbf{w} = (v_1, \ldots, v_k)$ of (not necessarily distinct) nodes in G such that $(v_i, v_{i+1}) \in E$ for all $i = 1, \ldots, k$. In this case, k is the length of $\mathbf{w}$ and we say $\mathbf{w}$ is a walk *from* v_1 *to* v_k. The walk is *closed* if $v_1 = v_k$. The walk $\mathbf{w}$ is a (directed) *path* if the nodes $v_1, \ldots, v_k$ are distinct. If G is undirected, walks and paths are defined similarly using undirected edges.

Exercise 2.3.2. Let $G = (V, E)$ be a graph. Suppose there exists a walk from a node u to a node v. Show that there exists a path from u to v.

Definition 2.3.3 (Connectedness for undirected graphs)**.** Let $G = (V, E)$ be an undirected graph. We say G is *connected* if for each pair of nodes (u, v), there exists a walk from u to v in G. A subgraph H in G is a *connected component* of G if H is connected and there is no connected subgraph of G that properly contains H.

Definition 2.3.4 (Connectedness for directed graphs)**.** Let $G = (V, E)$ be a directed graph. We say G is *connected* if for each pair of node (u, v), there exists a (directed) walk from u to v in G. G is *strongly connected* if for each pair of node (u, v), there is a directed walk from u to v and also a directed walk from v to u.

Definition 2.3.5 (Forest and trees)**.** A graph $G = (V, E)$ is a *forest* if it does not contain any cycle as a subgraph. G is a *tree* if it is non-null connected forest.

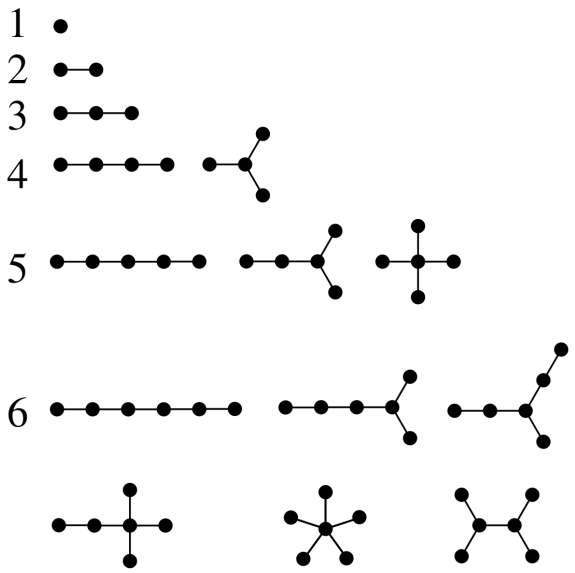


FIGURE 2.3.1. Non-isomorphic trees of different sizes.

Exercise 2.3.6. Let $T = (V, E)$ be a tree. Show that $|V| = |E| + 1$. Is the converse also true? If so, prove it; otherwise, give a counterexample.

Definition 2.3.7 (Spanning trees)**.** Let $G = (V, E)$ be a graph. A subgraph $T \subseteq G$ is called a *spanning tree* of G if T is a tree and $V(T) = V$.

Proposition 2.3.8. *Let $G = (V, E)$ be a graph. Then G is connected if and only if G contains a spanning tree.*

PROOF. If G contains a spanning tree, then G is connected. (Why?)

Conversely, suppose G is connected. Let T be a connected subgraph of G such that $V(T) = V$. Take it to be as small as possible, in the sense that there is no proper subgraph of T that is connected and still contains all nodes of G. Then T must be a tree. (If it contains a cycle, then we can delete an edge without hurting the connectedness.) Then, T must be a spanning tree of G. □

The proof of Proposition 2.3.8 suggests an algorithm to find a spanning tree.

Exercise 2.3.9 (A naive spanning tree algorithm)**.** Consider the following algorithm:

Input: A graph $G = (V, E)$

Initialize: $T \leftarrow G$

While: T contains a cycle C: Delete an edge of C

Return: A spanning tree T of G.

Is this algorithm usable? Efficient? What could be potential issues? (Later we will learn a much more efficient Kruskal's algorithm for finding minimum spanning trees.)

2.4. Bipartite graphs

Definition 2.4.1. A *bipartite graph* G is a graph whose vertex set V can be partitioned into two subsets A and B, such that each edge of G has one endpoint in A and one endpoint in B. The pair (A, B) is called a (vertex) *bipartition* of G, and A and B are called the bipartition subsets.



FIGURE 2.4.1. Two bipartite graphs.

Exercise 2.4.2 (Grid graph)**.** A $m \times n$ *grid graph* is a simple graph G with vertex set $V = \{1,\dots,m\} \times \{1,\dots,n\}$ and edge set E defined as follows. Two nodes (x,y) and (z,w) are adjacent if and only if $|x-z|+|y-w|=1$. (See Figure 2.4.2.) Show that every grid graph is bipartite.

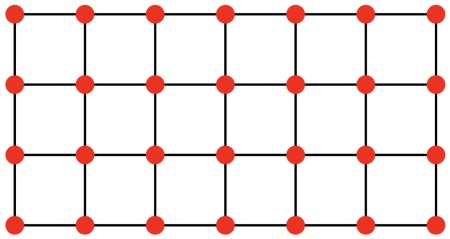


FIGURE 2.4.2. A 4×7 grid graph.

Example 2.4.3 (Complete bipartite graphs)**.** A complete bipartite graph $K_{m,n}$ is a simple bipartite graph with a bipartition (A,B) where $|A| = m$, $|B| = n$, and every pair of nodes $(u,v) \in A \times B$ are adjacent. So there are mn edges $K_{m,n}$. See Figure 2.4.3. ▲

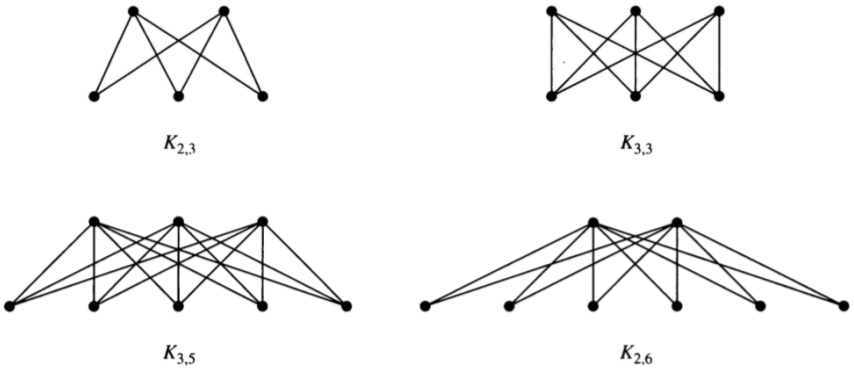


FIGURE 2.4.3. Examples of complete bipartite graphs.

Example 2.4.4 (Factor graphs)**.** A *factor graph* is a bipartite graph representing the factorization of a function. There are two types of nodes — the function nodes and the variable nodes. The edges represent the dependence of functions on variables, so they are between function and variable nodes. This gives a natural bipartition of the graph. In probability theory and its applications, factor graphs are used to represent the factorization of a probability distribution function. They are used for efficient computations of marginal distributions through the sum-product algorithm.

It is also common to see such factor graphs in biological contexts. The 'function nodes' could represent target proteins and the 'variable nodes' could represent drugs. See Figure 2.4.4 for an example. ▲

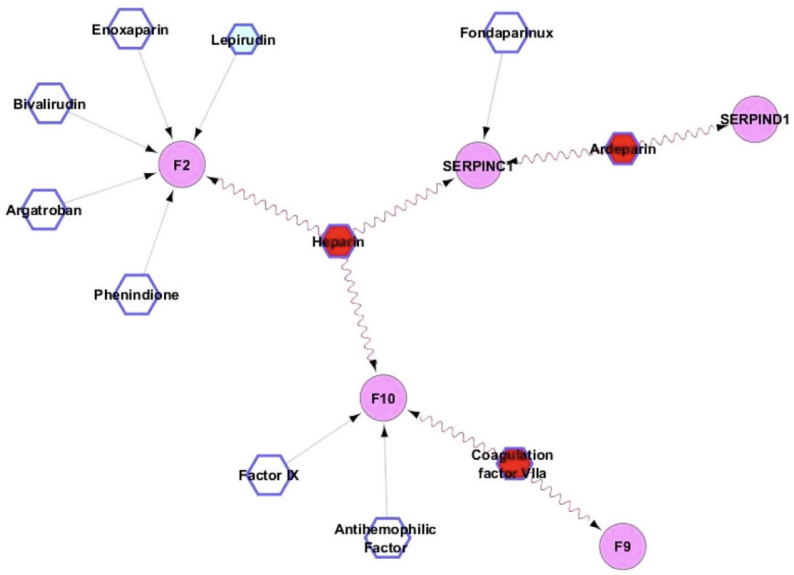


FIGURE 2.4.4. A small component of the drug-target protein network consisting of 11 drugs (hexagons) interacting with five disease-gene products corresponding to the cardivascular disorder class. Figure and caption excerpted from [NA13].

Exercise 2.4.5 (Cycles and bipartition)**.** Let C_n be the cycle graph with n nodes. Show that C_n is bipartite if and only if n is even.

Proposition 2.4.6. *A graph $G = (V, E)$ is bipartite if and only if G does not contain an odd cycle.*

PROOF. "Only if" direction: Easy. Exercise.

"If" direction: Assume G has no odd cycle. Without loss of generality, assume G is connected (why?) Take a spanning tree T of G and view it as a rooted tree. Color the root "red", and its children "blue", and their children "red", and so on. If there is an edge e in G that is not in T, then it must have both endpoints of the same color, since otherwise, it creates an odd cycle in G, contrary to the hypothesis. Now the red/blue partition of the node set gives a bipartition of G, so G is bipartite. See Figure 2.4.5 for an illustration.

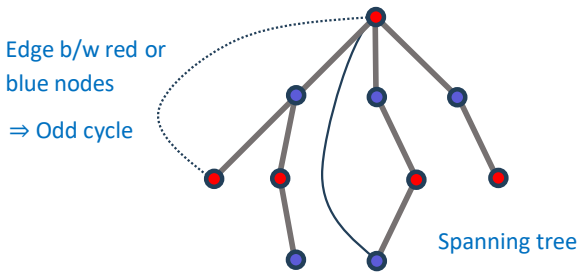


FIGURE 2.4.5. No odd cycle implies bipartition

□

Proposition 2.4.7. *A simple graph $G = (V, E)$ is bipartite if and only if G does not contain an induced odd cycle.*

PROOF. "Only if" direction: Easy. Exercise.

"If" direction: It is enought to show that "G simple non-bipartite ⇒ G has an induced odd cycle". Assume G is simple and is non-bipartite. We wish to show that G contains an induced

subgraph C that is an odd cycle. By Proposition 2.4.6, there exists an odd cycle C in G. Choose C as small as possible. If C is induced, we are done. Otherwise, C has as 'chord' e in G that shortcuts the cycle. (That is, there is an edge $e = uv$ in G, not in C, such that $u, v \in V(C)$.) But then G contains a smaller odd cycle, which contradicts the minimality of C. So C must be induced.

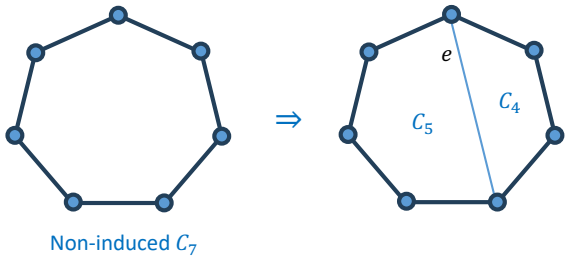


FIGURE 2.4.6. Simple bipartite graph

□

2.5. Euler's theorem and Hamiltonian cycles

The city of Königsberg in Prussia (now Kaliningrad, Russia) was set on both sides of the Pregel River, and included two large islands—Kneiphof and Lomse—which were connected to each other, and to the two mainland portions of the city, by seven bridges. The problem of great historical importance, *seven bridges of Königsberg*, is devise a closed walk through the city that would cross each of those bridges once and only once.

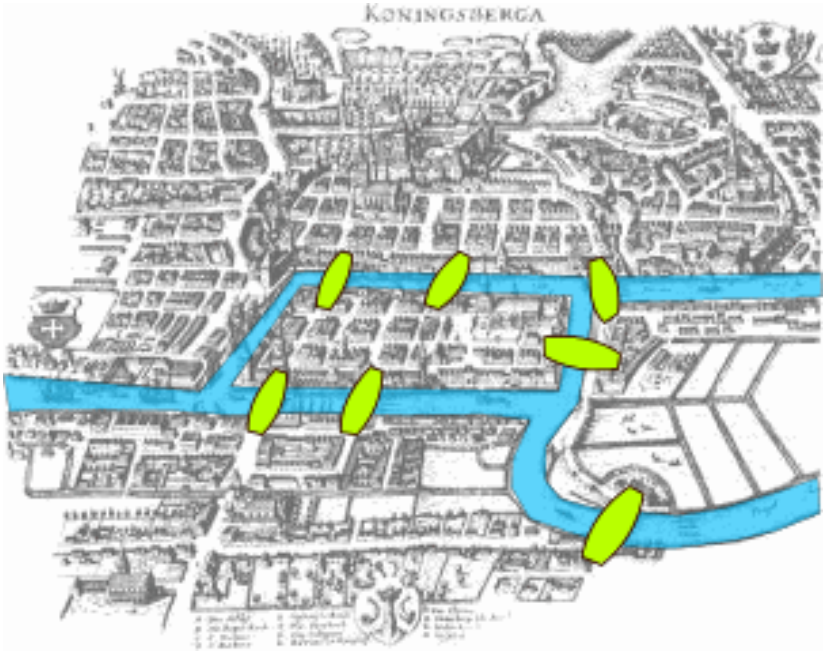


FIGURE 2.5.1. The seven bridges of Königsberg. Figure exerpted from https://en.wikipedia.org/wiki/Seven_Bridges_of_Königsberg.

Leonhard Euler in 1736 proved that it is impossible to walk through the city of Königsberg by crossing each of the seven bridges once and only once and come back to the starting point. The solution and his idea paved the foundations of graph theory and prefigured the idea of topology.

The first step of Euler's approach was to translate this problem into a graph theory problem (so that instead of walking through the city, you can draw paths on a drawing of the representative graph).

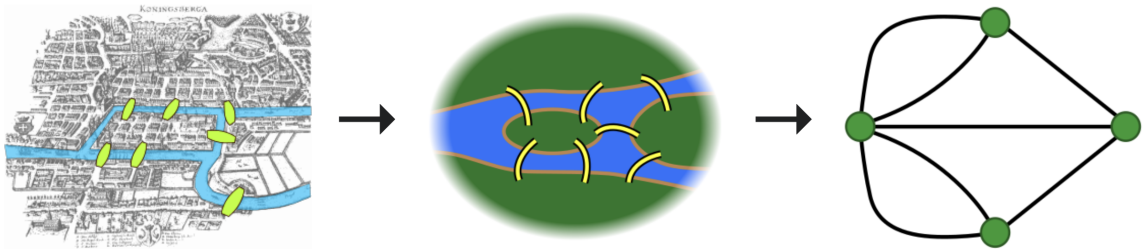


FIGURE 2.5.2. The seven bridges of Königsberg. Figure exerpted from https://en.wikipedia.org/wiki/Seven_Bridges_of_Königsberg.

Now, the problem becomes to find a *closed walk* that uses every edge in the graph in Figure 2.5.2 exactly once. A simple answer to this problem is given by the following observation.

Exercise 2.5.1 (Eulerian circuit implies even degrees)**.** Let $G = (V, E)$ be a graph. A closed walk in G that uses every edge in G exactly once is called an *Eulerian circuit* in G. Show that if G contains an Eulerian circuit, then every node in G must have an even degree. (*Hint:* Traverse the Eulerian circuit. How many times to you go in and out of a node?)

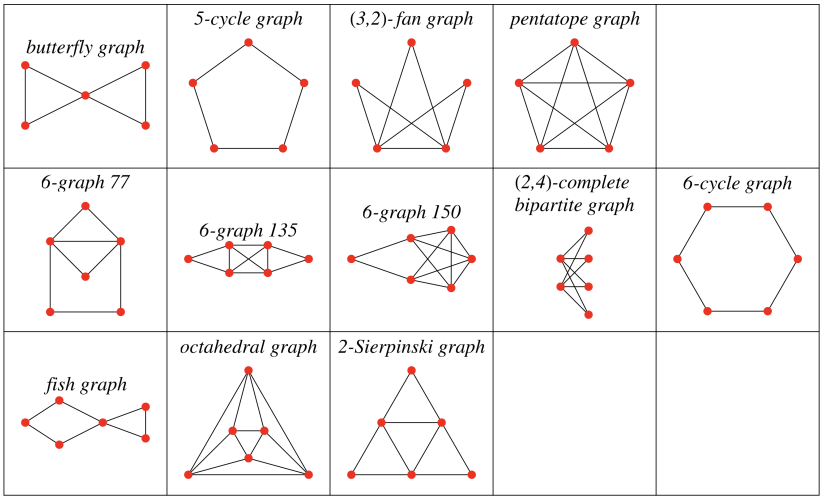


FIGURE 2.5.3. Examples of connected graphs with Eulerian circuit.

So we know a graph with an Eulerian circuit must have all nodes with even degrees. Is this necessary condition for having an Eulerian circuit also sufficient? In other words, If a graph has every node with even degree, then does it always have an Eulerian circuit? This is a bit more difficult to establish. A key ingredient to this direction is the following observation that graphs where all nodes have even degree must have a cycle cover.

Exercise 2.5.2 (Even degree implies cycle cover)**.** Let $G = (V, E)$ be a graph where every node has an even degree. We will show that every edge e is contained in a cycle C in G. (In other words, there exists a set of cycles $\mathscr{C} := \{C_1, \ldots, C_m\}$ in G such that they are edge-disjoint and E is partitioned into the edge sets of C_is. Such a collection $\mathscr{C}$ of cycles is called a *cycle cover* of G.)

(i) Show that G cannot be a forest, so it must contain a cycle C. (*Hint*: Show that every tree must have a node with degree one.)

(ii) By part **(i)**, G contains a cycle C. Delete all edges in G from G. Argue that every node in the resulting graph $G' := G \setminus E(C)$ has an even degree. (*Hint:* If a node v is not contained in C, then the degree of v is unchanged; otherwise, it decreases by an even amount.)

(iii) By parts **(i)**-**(ii)**, we can keep deleting edges in a cycle until we do not have any more edges left. Conclude that every edge e is contained in a cycle in G.

Now we are ready to state and prove Euler's theorem, which completely resolves the Seven Bridges of Königsberg problem and its generalization.

Theorem 2.5.3 (Euler's theorem)**.** *Let G be a connectd graph. Then there is a closed walk using each edge exactly once (i.e., an Eulerian circuit) if and only if every node in G has even degree.*

PROOF. The "only if" direction is easy and is done by Exercise 2.5.1.

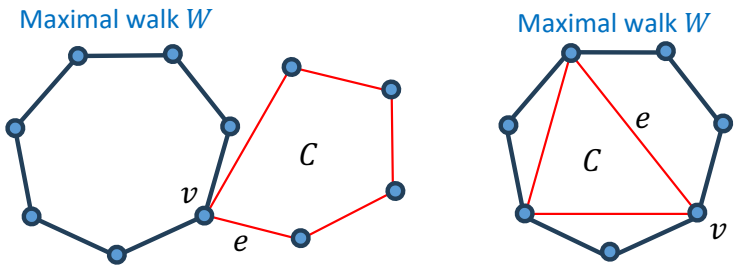


FIGURE 2.5.4. Illustration of the proof of Euler's theorem. If the maximal closed walk W in G does not contain all edges and misses some edge e, then e has an end v in W and e must be contained in a cycle C that is edge-disjoint from W. Then we can further extend the maximal walk W by traversing C at v, a contradiction.

For the "If" direction, suppose all nodes in G have even degree. We first assume that G is at least one edge, since otherwise the problem is vacuous. Then one can choose a closed walk $W = (v_0, e_1, \dots, e_k, v_k)$ so that the edges $e_1, \dots, e_k$ are all distinct. Choose such a closed walk so that k is as large as possible. We claim $\{e_1, \dots, e_k\} = E(G)$, which shows that G is Eulerian. Suppose for a contradiction that $\{e_1, \dots, e_k\} \subsetneq E(G)$. Let $X := |E(G)| \setminus \{e_1, \dots, e_k\}$, the set of edges in G not in this walk W. Since G is connected, there is an edge $e \in X$ that has at least one end, say v, in W. Let H be a subgraph of G with $V(H) = V(G)$ and $E(H) = X$. By construction, every node in H has an even degree, so by Exercise 2.5.2, there is a cycle C in H that contains the edge e. Hence C contains the node v. Now since C is not using any edge in the walk W, we can adjoin C to W to further extend the already maximal walk W, which is a contradiction. See Figure 2.5.4 for an illustration. □

Exercise 2.5.4 (Eulerian trail)**.** Let G be a graph. A walk W in G is an *Eulerian trail* if it is not a closed walk and if it traverses every edge exactly once. Show that G contains an Eulerian trail if and only if G has exactly two nodes with odd degree. (*Hint:* Use Euler's theorem on Eulerian cycle.)

Next, we turn our attention to a similar notion of Hamiltonian cycle.

Definition 2.5.5. A *Hamiltonian cycle* in a graph G is a cycle $C \subseteq G$ with $V(C) = V(G)$ (i.e., a spanning cycle). We say G is *Hamiltionian* if it contains a Hamiltonian cycle.

Exercise 2.5.6. Show that K_n (the complete graph with n nodes) has a Hamitonian cycle.

Exercise 2.5.7. Show that if an Eulerian circuit is a cycle (i.e., it does not visit the same node more than once), then it is a Hamiltonian cycle. Give an example of Eulerian circuit that is *not* a Hamiltonian cycle.

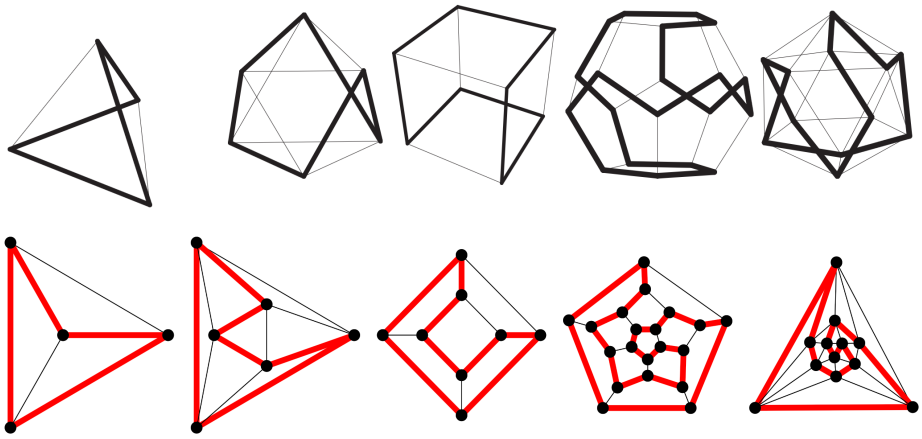


FIGURE 2.5.5. All platonic solids are Hamiltonian.

Exercise 2.5.8 (A necessary condition for having a Hamiltonian cycle)**.** Let $G = (V, E)$ be a graph. Show that if $X \subseteq V$ is such that $G \setminus X$ (the graph obtained by deleting all nodes in X from G along with all incident edges) has $> |X|$ components, then G has no Hamiltonian cycle. (*Hint:* Suppose G is a cycle. Delete k nodes from it. How many connected components can there be at most?)

Exercise 2.5.9. Show that $K_{5,6}$ is not Hamiltonian. (*Hint:* Use Exercise 2.5.8).

Exercise 2.5.10. Show that the Peterson graph (see Figure 2.2.3) is not Hamiltonian.

Unlike Eulerian circuit, there is no known if and only if condition for having a Hamiltonian cycle. The following is a well-known sufficient condition for having a Hamiltonian cycle.

Theorem 2.5.11 (Dirac and Posa)**.** *If G is simple and has n nodes with $n \geq 3$, and if*

$$\deg(u) + \deg(v) \geq n \quad \text{for all non-adjacent nodes } u, v \tag{3}$$

then G contains a Hamiltonian cycle.

PROOF. Fix the number n of nodes. We proceed by an induction on $\binom{n}{2} - |E|$. For the base step, suppose $\binom{n}{2} - |E| = 0$. Then $G = K_n$ so G contains a Hamiltonian cycle. For the induction step, we may assume that $\binom{n}{2} - |E| \geq 1$, and as the induction hypothesis, we assume that any simple graph with n nodes with more than $|E|$ edges with property (3) contains a Hamiltonian cycle. We with to show that G has a Hamiltonian cycle. Since $\binom{n}{2} - |E| \geq 1$, there is a pair of non-adjacent nodes x, y in G. Let $G' = G + xy$, where we add a new edge xy to G. Notice that G' satisfies (3). Hence by the induction hypothesis, G contains a Hamiltonian cycle, say, C. If C does not use the newly added edge xy, then $C \subseteq G$ and $V(C) = V(G') = V(G)$, so C is a Hamiltonian cycle in G, so we are done. Thus we may assume that $xy \in E(G')$. See Figure 2.5.6 for an illustration.

Now order the nodes in C as $v_1, v_2, \ldots, v_n$, where $x = v_1$, $y = v_n$, and $v_i v_{i+1} \in E(G')$ for $i = 1, \ldots, n-1$. Define two sets

$$P := \{2 \leq i \leq n \mid v_1 v_i \in E(G)\},$$
$$Q := \{2 \leq i \leq n \mid v_n v_{i-1} \in E(G)\}.$$

Note that by (3),

$$|P| + |Q| = \deg_G(x) + \deg_G(y) \geq n.$$

But since $P \cup Q \subseteq \{2, 3, \ldots, n\}$, it follows that there is some $i \in P \cap Q$. For such i, we have $v_1 v_i$, $v_n v_{n-1} \in E(G)$. Now traversing the nodes in G in the following order forms a Hamiltonian cycle in G:

$$v_1, v_i, v_{i+1}, \ldots, v_{n-1}, v_n, v_{i-1}, v_{i-2}, \ldots, v_2, v_1.$$

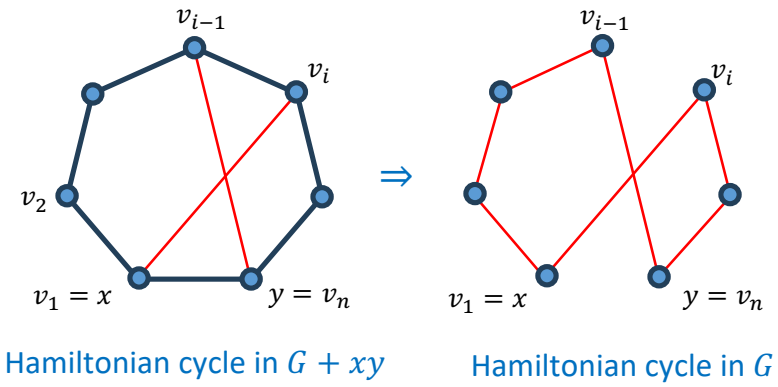


FIGURE 2.5.6. Construction of a Hamiltonian cycle in G by using a Hamiltonian cycle in $G + xy$.

This completes the induction. □

Exercise 2.5.12 (Dirac's theorem)**.** Show that if a n-node simple graph G has minimum degree $\geq n/2$, then G contains a Hamiltonian cycle. (*Hint: Use Theorem 2.5.11*)

Exercise 2.5.13 (Counterexample of Theorem 2.5.11)**.** Find a graph G with a Hamiltonian cycle that does not satisfy the condition in Theorem 2.5.11. (*Hint: Consider cycles.*)

2.6. Matchings in bipartite graphs

Definition 2.6.1 (Matching)**.** A *matching* in $G = (V, E)$ is a subset $M \subseteq E(G)$ so that no edge in M is a loop, and no two edges in M are incident with the same vertex. A matching M is *perfect* if $2|M| = V(G)$ (i.e., M spans G). A matching M is *maximal* if there is no other matching with strictly more edges.

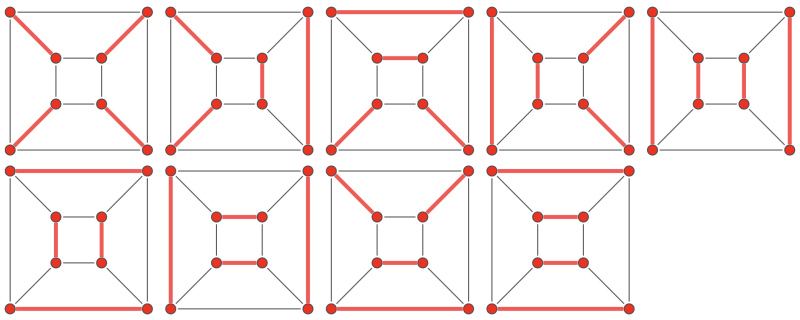


FIGURE 2.6.1. Examples of perfect matching in the cube.

Definition 2.6.2 (Alternating and augmenting paths)**.** If M is a matching in G, a path P of G is *M-alternating* if the edges of P alternately belong to M and to $E(G) \setminus M$. (In other words, for every $v \in V(P)$ with degree 2 in P, some edge of P incident with v belongs to M). It is *M-augmenting* if it is M-alternating, it has distinct ends u, v (say), and no edge of M is incident (in G, not just in P) with both u and v.

Exercise 2.6.3 (Union of two matchings)**.** Let $G = (V, E)$ be a graph. Let M and M' be matchings in G. Define a subgraph $S := (V, M \cup M') \subseteq \mathscr{G}$. Let H be a connected component of S. Show the following:

(i) H is either a cycle or a path. (*Hint:* Show that every node in H must have degree 1 or 2.)

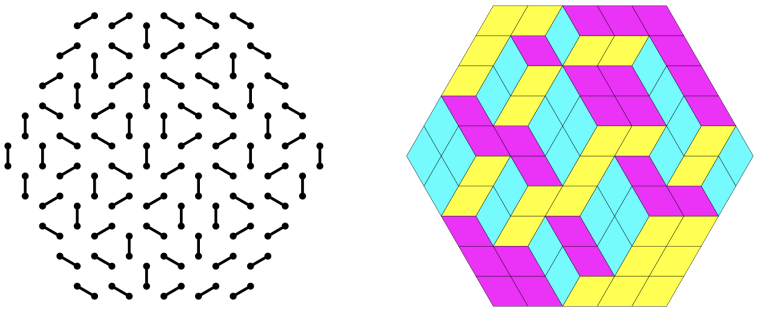


FIGURE 2.6.2. (Left) Matching (dimers) in the honeycomb lattice. (Right) Lozenge tiling. Image credit: Richard Kenyon.

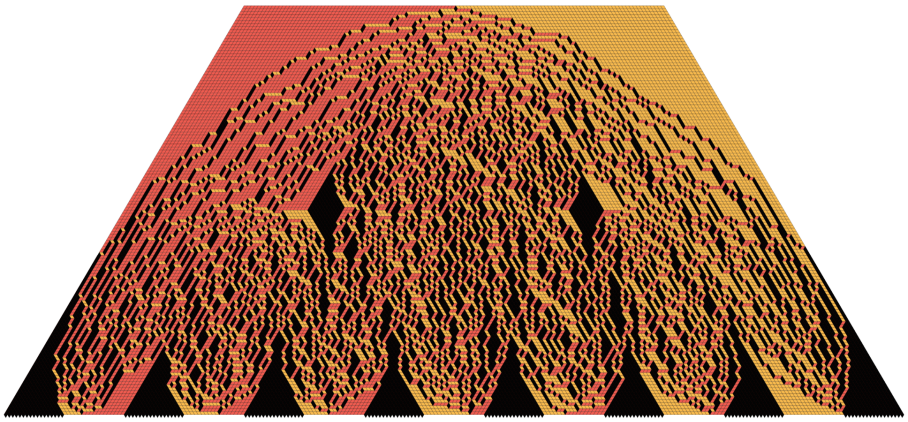


FIGURE 2.6.3. A random lozenge tiling. Image credit: Leonid Petrov.

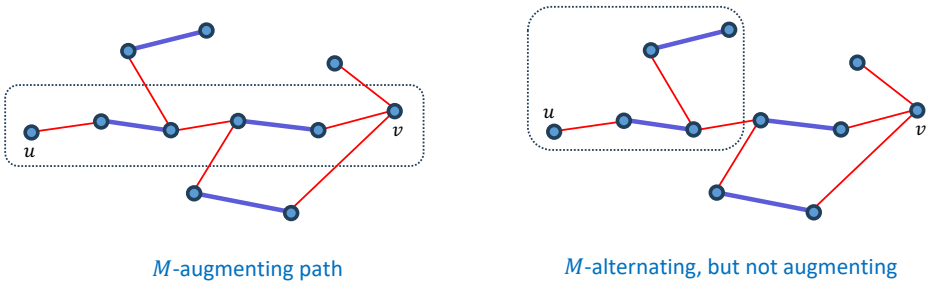


FIGURE 2.6.4. Matching M (thick blue edges in both left and right graphs), M-augmenting path (left in the dotted box), M-alternating but not M-augmenting path (right).

(ii) If H is a cycle, then it contains equal number of edges from M and M'. If H is a path, say from u to v, then traversing it from u to v alternates edges in M and M'. (*Hint:* Every node in H cannot be incident to two distinct edges from M' or M.)

(iii) Suppose H is a path and contains more edges in M' than in M. Then H is a M-augmenting path. (*Hint:* By **(ii)**, the path starts and ends with an M'-edge. Since H is a connected component of S, there is no M-edge outside of H incident to any node H.)

Proposition 2.6.4 (Berge '57)**.** *Let M be a matching in $G = (V, E)$. There is a matching M' with $|M'| > |M|$ if and only if there is an M-augmenting path.*

PROOF. The "if" direction is easy. If we have an M-augmenting path P, then define a new matching M' by taking the symmetric difference $M \Delta E(P)$ between M and $E(P)$. That is, an edge e in G is in M' if and only if e is in eigher M or $E(P)$ but not in both. Then the definitinon of M-augmenting paths ensures that M' is a valid matching in G. (See Figure 2.6.5.)

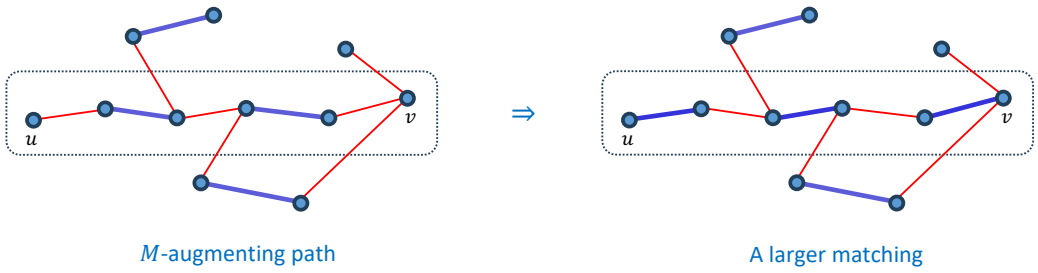


FIGURE 2.6.5. Obtaining a larger matching by using an augmenting path.

For the "only if direction", let M' be a matching in G with $|M|' > |M|$. Let $H = (V, M \cup M')$. Consider the connected components of H. Since $|M'| > |M|$, some component C of H must have more edges in M' than in M. Then C must be an M-augmenting path by Exercise 2.6.3. □

Definition 2.6.5 (Vertex cover)**.** Let $G = (V, E)$ be a graph. A subset $X \subseteq V$ of the vertex set is called a *vertex cover* of G if every edge in G has at least one end in X (i.e., X meets every edge in G).

Exercise 2.6.6. Let G be a graph. Show that

$$(\text{size of a maximal matching in } G) \leq (\text{size of a minimal vertex cover in } G)$$

(*Hint: Every edge in the matching must meet with a node in the vertex cover. No two edges in a matching share a node.*)

Exercise 2.6.7. Find a graph G for which

$$(\text{size of a maximal matching in } G) < (\text{size of a minimal vertex cover in } G).$$

(*Hint: Consider* C_3.)

Exercise 2.6.6 tells us that the size of a maximal matching in a graph G must be at most the size of a minimal vertex cover. Exercise 2.6.7 tells us that one can have a strictly smaller maximal matching than a minimal vertex cover. The following theorem due to Dénes Kőnig (1931) states that, for bipartite graphs, these two quantities always equal.

Theorem 2.6.8 (Kőnig '31)**.** *Let G be bipartite, and $k \geq 0$ an integer. Exactly one of the following holds:*

(i) *G has a matching M with $|M| \geq k$;*

(ii) *There exists $X \subseteq V(G)$ with $|X| < k$ such that X meets every edge of G.*

In partiuclar, it holds that

$$(\text{size of a maximal matching in } G) = (\text{size of a minimal vertex cover in } G) \tag{4}$$

PROOF. (Optional*) The 'strong duality' statement in (4) follows from the first part of the statement. Indeed, recall the 'weak duality' stated in Exercise 2.6.6. If strict inequality holds, then denoting the size of maximal matching as k, there exists a minimal vertex cover of size $> k$. But then **(i)** and **(ii)** in the statement both hold, which is impossible by the first part of the statement.

Next, we argue that **(i)** and **(ii)** cannot hold at the same time. Indeed, suppose **(i)** and **(ii)** both hold. Since X meets every edge of G, it meets every edge in M. Since $|M| > |X|$, at least two edges in M must be incident to the same node in X. But this contradicts that M is a matching.

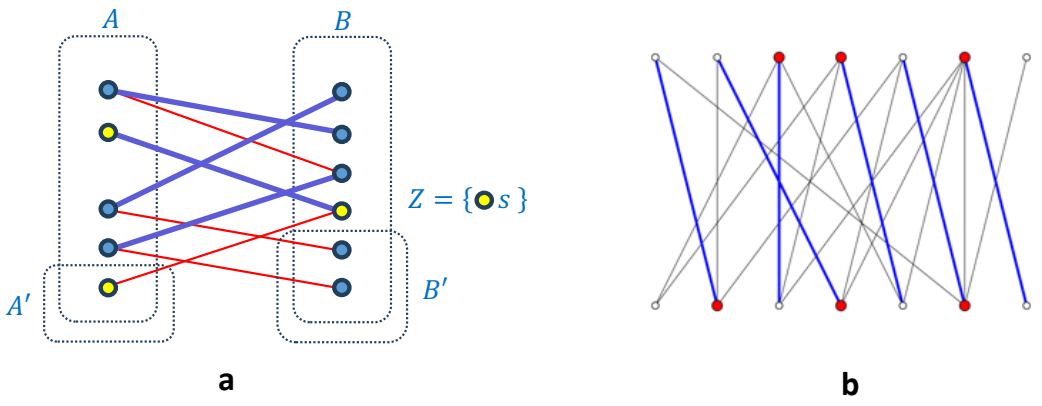


FIGURE 2.6.6. (**a**) Illustration of the proof of Kőnig's theorem. (**b**) An example of a bipartite graph with a maximum matching (blue) and minimum vertex cover (red), both of size six.

Lastly, we show that at least one of **(i)** and **(ii)** must hold. Suppose **(i)** is false. We will show that **(ii)** holds. Choose a matching M in G as large as possible; then $|M| < k$ by **(i)**. Let (A, B) be a bipartition of G (recall that G is bipartite). Let $A' \subseteq A$ be the set of vertices of A not incident with any edge of M, and define $B' \subseteq B$ similarly. Let Z be the set of all vertices v such that there is an M-alternating path between v and some node in A'. Define $X = (A \setminus Z) \cup (B \cap Z)$.

We will make six claims:

(a) $A' \subseteq Z$;
(b) $B' \cap Z = \emptyset$;
(c) Every edge of M with an end in Z has both ends in Z;
(d) Every edge of G with an end in $A \cap Z$ has its other end in $B \cap Z$;
(e) X is a vertex cover;
(f) $|X| = |M| (< k)$.

Then claims (e)-(f) finishes the proof.

Now we justify the above claims. For (a), If there is some node $u \in A' \setminus Z$, then for any neighbor v ($\in B$) of u, $uv \notin M$ since $u \in A'$, and v is not incident to any edge in M since $u \notin Z$. So we can add uv to M and it is still a matching, which contradicts the maximality of M.

For (b), if there is some node $u \in B' \cap Z$, then there is an M-alternating path from u to some node v in A'. This path is in fact M-augmenting since $u \in B'$ and $v \in A'$. So then by Proposition 2.6.4, there is a larger matching than the maximal one M, a contradiction.

For (c), if $e = xy \in M$ and $x \in Z$, then there is an M-alternating path P from x to some node, say, w, in A'. If $e \in E(P)$, then $P \setminus y$ is an M-alternating path from y to w, so $y \in Z$ by definition; Otherwise, $P + y$ is an alternating path from y to w so $y \in Z$.

For (d), let $e = xy \in E(G)$ and suppose $x \in A \cap Z$. That $y \in B$ follows since (A, B) is a bipartition. To see $y \in Z$, let P be an M-alternating path from x to some node $w \in A'$ (recall $x \in Z$). Since $w \in A'$, the last edge in P incident to w is not in M. Since P is M-alternating it contains an even number of edges, it follows that the first edge in P incident to x is in M. Then depending on whether $y \in V(P)$ or not, $P \setminus x$ or $P + y$ is an M-alternating path ending at $w \in A'$. Hence $y \in Z$.

For (e), note that (A, B) is a bipartition of G, so every edge in G must have one end in A and another end in B. If an edge e in G has an end in $A \setminus Z$, then it is covered by X; otherwise, it has an end in $A \cap Z$, and by (c), it must have the other end in $B \cap Z$, so it is again covered by X.

Lastly for (e), write

$$\begin{aligned} |M| &\overset{(c)}{=} \#(\text{edges in } M \text{ having no end in } Z) + \#(\text{edges in } M \text{ having both ends in } Z) \\ &= |A \setminus Z| + |B \cap Z| = |X|, \end{aligned}$$

where the second equality follows since (A,B) is a bipartition of G so every edge in M must have one end in A and another end in B. □

Remark 2.6.9. Theorem 2.6.8 is the first occurance of a 'duality' phenomonon, which in a very rough sense, states that

$$\max(\text{a quantity } F) = \min(\text{a dual quantity } F^*).$$

We will see a similar duality phenominon in two other places (Menger's theorem and max-flow-min-cut theorem).

Exercise 2.6.6 shows that the inequality "$\leq$" in (4) is always true. One can view this as a weak duality. Strong duality means the equality in (4). In optimizaiton literature, strong duality is attained if the objective function is convex. The role of convexity in the setting of Kőnig's theorem is played by the graph being bipartite. If you are familiar with this topic, see also this article.

Corollary 2.6.10 (Perfect matching bipartite regular graphs)**.** *Let $G = (V,E)$ be a bipartite d-regular graph for some $d \geq 1$. Then G has a perfect matching.*

PROOF. Let M and X denote a maximal matching and a minimial vertex cover in G. Suppose for a contradiction that G does not have a perfect matching. Then $|M| < |V|/2$, and by Theorem 2.6.8, $|X| = |M| < |V|/2$. Since each node in X has degree d, it can cover at most d edges in G. Thus

$$|E| \leq d|X| < d|V|/2.$$

However, we have

$$d|V| = \sum_{u \in V} \deg(u) = 2|E|,$$

which is a contradiction. Thus G must have a perfect matching. □

Exercise 2.6.11. Let G be a bipartite graph, with bipartition (A,B). Show that the following are equivalent:

(i) There is a matching M in G so that every vertex in A is an end of some member of M
(ii) For every $X \subseteq A$, there are at least $|X|$ vertices in B with a neighbor in X.

Exercise 2.6.12. Let G be a loopless graph in which every vertex has degree ≥ 1. Let X be the largest matching in G, and let Y be the smallest set of edges of G so that every vertex of G is incident with ≥ 1 edge in Y. Show that $|X| + |Y| = |V(G)|$.

2.7. Menger's theorem

Menger's theorem is one of the cornerstones of structure graph theory. It is another example of a strong duality statement between the maximal number of vertex-disjoint paths between source to target and a minimal order 'separation' of them.

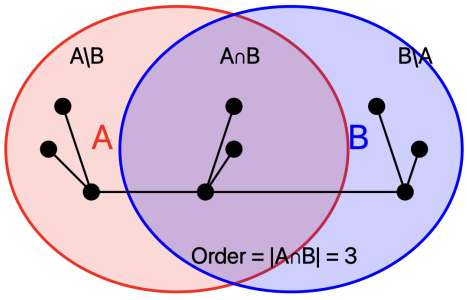


FIGURE 2.7.1. Example of a separation (A,B) of order 3 of a graph G

Definition 2.7.1. A *separation* of G is a pair (A,B) of subsets of $V(G)$ with $A\cup B = V(G)$ such that neither A nor B contains the other and there is no edge between $A\setminus B$ and $B\setminus A$. Its *order* is $|A\cap B|$. For given sets $Q,R\subseteq V$, a separation (A,B) is a (Q,R)*-separation* if $Q\subseteq A$ and $R\subseteq B$.

Example 2.7.2. If G is disconnected and has two connected components, say G_1 and G_2, then $(V(G_1),V(G_2))$ is a separation of G of order 0.

Exercise 2.7.3. Show that a cycle C_n with $n\geq 3$ has a separation of order 2. Show that there is no separation of order less than 2.

The following lemma contains most of Menger's theorem.

Lemma 2.7.4 (Separation lemma)**.** *Let $G=(V,E)$ be a graph, let $Q,R\subseteq V(G)$ with $|Q|,|R|\geq k$ for $k\geq 0$ be an integer. Then exactly one of the following holds:*

(i) *There are k paths $P_1,\dots,P_k$ of G from Q to R, pairwise vertex-disjoint;*
(ii) *There is a separation (Q,R)-separation of order $<k$.*

PROOF. That not both hold is easy. To prove that at least one holds, we use induction on $|V(G)|$. We assume **(ii)** is false and it is enough to prove that **(i)** holds. We proceed by an induction on $|V(G)|$.

Case 1: There is some $v\in Q\cap R$.

If there exists $v\in Q\cap R$, the result follows by deleting v and applying the induction hypothesis to $G\setminus v$. Namely, if there is a separation (A,B) of $G\setminus v$ of order $<k-1$ with $Q\setminus v\subseteq A$ and $R\setminus v\subseteq B$, then $(A\cup\{v\},B\cup\{v\})$ is a (Q,R)-separation of G of order $<k$, so we are done. By the induction hypothesis, we may assume that there exists $k-1$ vertex-disjoint paths $P_1,\dots,P_k$ in $G\setminus v$. Letting $P_k=(\{v\},\emptyset)$, $P_1,\dots,P_k$ are k vertex-disjoint paths in G, so we are done in this case as well.

Case 2: $Q\cap R=\emptyset$ and $Q\cup R=V$.

Suppose $Q\cap R=\emptyset$ and $Q\cup R=V$. Then the result follows from König's theorem (Thm. 2.6.8). Indeed, delete all edges within Q and within R from G and let G' denote the resulting bipartite graph with bipartition (Q,R). Suppose there exists a minimum vertex cover X of size $<k$. Then $(Q\cup X,R\cup X)$ is a (Q,R)-separation of G' of order $|X|<k$. In fact, this is also a (Q,R)-separation of G, since the edges in $G\setminus G'$ are within Q or R. Hence this contradicts **(ii)**. It follows that G' cannot have a minimum vertex cover of size $<k$. Hence by Theorem 2.6.8, there exists a matching M of size $\geq k$ in G'. This gives $|M|\geq k$ vertex-disjoint paths in G'. They are also vertex-disjoint in G, verifying **(i)**.

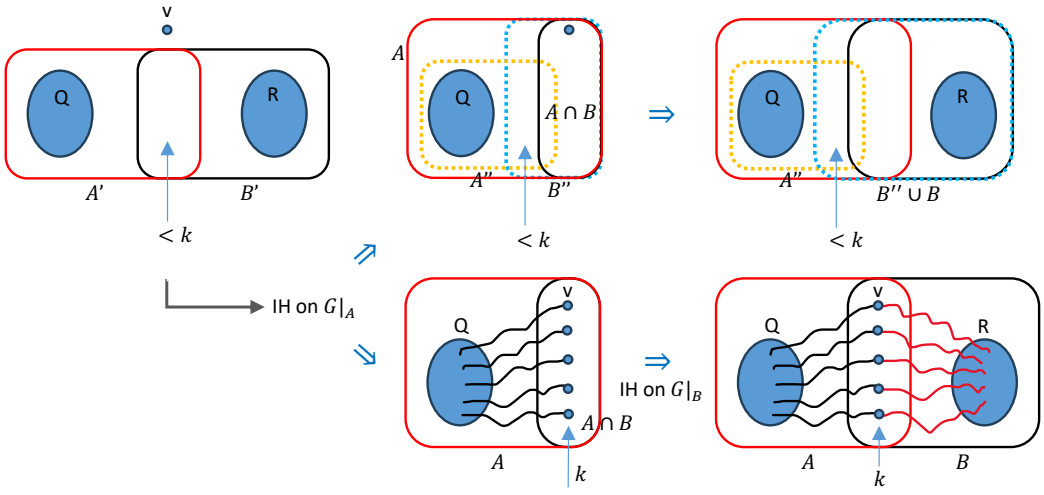


FIGURE 2.7.2. Illustration of the proof of the separation lemma. 'IH' stands for 'induction hypothesis'.

Case 3: $Q \cap R = \emptyset$ **and** $Q \cup R \subsetneq V$**.**

We may assume that there exists $v \in V \setminus (Q \cup R)$. From the induction hypothesis applied to $G \setminus v$, we may assume that there is a separation (A', B') of $G \setminus v$ with order at most $k-1$, and with $Q \subseteq A'$ and $R \subseteq B'$ (otherwise there are k vertex-disjoint paths from Q to R in $G \setminus v$, as desired). Let $A = A' \cup \{v\}$ and $B = B' \cup \{v\}$; then (A, B) is a separation of G of order $\leq k$, and $v \in A \cup B$. Since $R \not\subseteq A \cap B$ (because $|R| \geq k$ and $v \in A \cap B \setminus R$), it follows that $|A| < |V|$ and we can apply the induction hypothesis to $G|_A$ (the induced subgraph of G with node set A) with sets Q and $A \cap B$. If there is a $(Q, A \cap B)$-separation (A'', B'') of $G|_A$ of order $< k$, then $(A'', B'' \cup B)$ is a (Q, R)-separation of G of order $|A'' \cap B''| < k$, contrary to **(ii)**. Hence by the induction hypothesis, we must have k disjoint paths from Q to $A \cap B$. Since $|A \cap B| \leq k$, every vertex in $A \cap B$ is an end of one of such k paths. By a symmetric argument, there are k disjoint paths in $G|_B$ between $A \cap B$ and R; and piecing these paths together, we obtain k disjoint paths in G between A and B, and so (i) holds. □

Exercise 2.7.5. From Lemma 2.7.4, deduce the following strong duality:

$$\max\{k \mid (\text{there are } k \text{ vertex-disjoint paths from } Q \text{ to } R)\}$$
$$= \min\{\ell \mid (\text{there is a } (Q,R)\text{-separation of order } \ell)\}.$$

(*Hint:* The inequality $\leq$ is easy. For the other direction, use Lemma 2.7.4.)

Exercise 2.7.6. Deduce[1] Kőnig's theorem from Lemma 2.7.4. (*Hint*: Take (Q, R) in the lemma to be a bipartition).

Theorem 2.7.7 (Menger '27)**.** *Let s, t be distinct non-adjacent vertices of G, and let $k \geq 0$ be an integer. Then exactly one of the following holds:*

(i) *There are k paths $P_1, \ldots, P_k$ of G from s to t, pairwise disjoint except for their ends s, t;*

(ii) *There is a separation (A, B) of G of order $< k$ with $s \in A \setminus B$ and $t \in B \setminus A$.*

PROOF. Let Q be the set of neighbors of s, and R the neighbors of t. Use Lemma 2.7.4 for the graph $G \setminus \{s, t\}$ and sets Q and R. See Figure 2.7.3 for illustration.

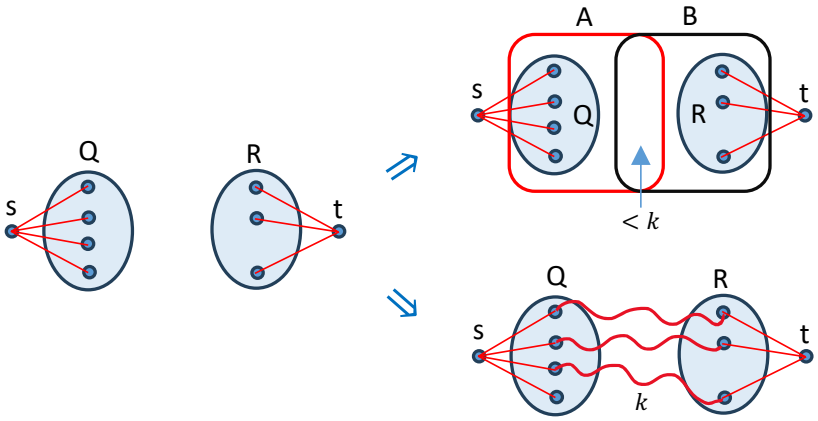


FIGURE 2.7.3. Illustration of the proof of Menger' theorem using Lemma 2.7.4.

□

Exercise 2.7.8. Let s, t be distinct non-adjacent vertices of G, and let $k \geq 0$ be an integer. Suppose that $\max\{\deg(s), \deg(t)\} < k$.

(i) Apply Menger's theorem to show that there exists a separation (A, B) of G of order $< k$ where $s \in A \setminus B$ and $t \in B \setminus A$.

[1] In fact, we have used Kőnig's theorem in the proof of Lemma 2.7.4, so one cannot use Lemma 2.7.4 to prove Kőnig's theorem.

(ii) Construct a separation (A,B) of G of order $< k$ where $s \in A \setminus B$ and $t \in B \setminus A$. (*Hint*: Suppose $\deg(s) < k$. Take $Q = N[s]$ and $R = G \setminus s$.)

Definition 2.7.9. A graph $G = (V,E)$ is *k-connected* if $|V| \geq k+1$ and for every subset $X \subseteq V$ with $|X| < k$, $G \setminus X$ is connected.

Exercise 2.7.10. Show that a graph is k-connected if and only if there is no a separation of order $< k$.

Exercise 2.7.11. Show that if G is k-connected then for any two non-adjacent vertices s,t, there are k paths of G from s to t that are pairwise vertex-disjoint except for their ends. (*Hint:* Apply Menger's theorem.)

Exercise 2.7.12 (An edge version of Menger's theorem)**.** Let $G = (V,E)$ be a graph. Let s,t be distinct vertices of G, and let $k \geq 0$ be an integer. Then exactly one of the following holds:

(i) There are k paths $P_1,\ldots,P_k$ of G from s to t, pairwise edge-disjoint (i.e. $E(P_i) \cap E(P_j) = \emptyset$ for $1 \leq i < j \leq k$).

(ii) There exists $X \subseteq V$ with $s \in X$ and $t \notin X$ such that there are $< k$ edges between X and $V \setminus X$.

(*Hint:* Apply menger's theorem to the line graph of G (see Def. 2.2.12), with Q =the set of all edges incident to s and R = the set of all edges incident to t).

2.8. Digraphs and network flows

Recall that a *directed graph* (or digraph) is a graph such that each edge e is given a "direction," i.e., one end of e is chosen as the head of e and the other as its tail. (A loop has the head and tail the same.) (See Def. 2.1.8.)

Definition 2.8.1. If G is a digraph and $X \subseteq V(G)$, $\delta^+(X)$ denotes the set of all edges with tail in X and head in $V(G) \setminus X$, and $\delta^-(X) = \delta^+(V(G) \setminus X)$. A *directed path* from s to t (denoted as st-path) means a path P in G such that the tail of every edge of P precedes its head as P is traversed from s to t.

Lemma 2.8.2 (Directed path and flow)**.** *Let s,t be distinct vertices of a digraph G. Then exactly one of the following holds:*

(i) *There is a directed path of G from s to t*

(ii) *There exists $X \subseteq V(G)$ with $s \in X$ and $t \notin X$ such that $\delta^+(X) = \emptyset$.*

PROOF. That not both hold is easy. (Try writing down the details.) It then suffices to show that when **(i)** is false, **(ii)** is true. Suppose **(i)** is false. Let X denote the set of all nodes v in G such that there is a directed path from s to v. Then $t \notin X$ since otherwise **(i)** holds. Furthermore, there is no edge xy in G with tail x in X and head y outside X, since then there is a directed path from s to y, so $y \in X$, a contradiction. Thus $\delta^+(X) = \emptyset$. □

Definition 2.8.3 ((s,t)-flow)**.** Let s,t be distinct vertices of a digraph G. An *(s,t)-flow* is a map $\phi : E(G) \to [0,\infty)$ such that for every node $v \notin \{s,t\}$,

$$(\text{"flow into } v\text{"}) = \sum_{e \in \delta^-(v)} \phi(e) = \sum_{e \in \delta^+(v)} \phi(e) = (\text{"flow sent from } v\text{"}) \tag{5}$$

(Think of $\phi(xy)$ the amount of flow from x to y.) We call (5) as the *local balance equation.* If ϕ is an (s,t)-flow, the *total value* (or flux) is defined by

$$\|\phi\| := \underbrace{\sum_{e \in \delta^+(s)} \phi(e)}_{\text{flow sent from } s} - \underbrace{\sum_{e \in \delta^-(s)} \phi(e)}_{\text{flow into } s}.$$

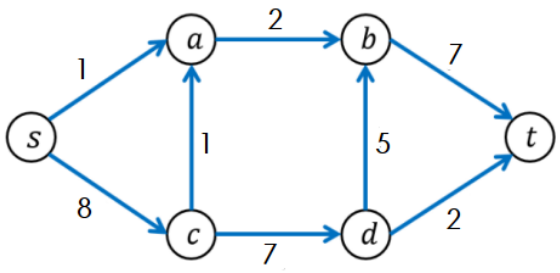


FIGURE 2.8.1. An example of (s,t)-flow. The total value of this flow is $(1+8)-0=9$.

Lemma 2.8.4 (Flux lemma)**.** *Let G be a digraph and let ϕ be an (s,t)-flow of total value k. Then for every $X \subseteq V(G)$ with $s \in X$ and $t \notin X$,*

$$\|\phi\| = \underbrace{\sum_{e\in\delta^+(X)} \phi(e)}_{\text{flow sent from } X} - \underbrace{\sum_{e\in\delta^-(X)} \phi(e)}_{\text{flow into } X}.$$

PROOF. One the one hand, using the local balance equation (5) and the fact that $s \in X$ and $t \notin X$,

$$\begin{aligned}
\|\phi\| &= \sum_{e\in\delta^+(s)} \phi(e) - \sum_{e\in\delta^-(s)} \phi(e) \\
&= \left(\sum_{e\in\delta^+(s)} \phi(e) - \sum_{e\in\delta^-(s)} \phi(e)\right) + \sum_{v\in X\setminus\{s\}} \underbrace{\left(\sum_{e\in\delta^+(v)} \phi(e) - \sum_{e\in\delta^-(s)} \phi(e)\right)}_{\equiv 0} \\
&= \sum_{v\in X} \left(\sum_{e\in\delta^+(v)} \phi(e) - \sum_{e\in\delta^-(s)} \phi(e)\right) \\
&= \sum_{e\in E(G)} \phi(e)\big(\mathbf{1}(e \text{ has tail in } X) - \mathbf{1}(e \text{ has head in } X)\big) \\
&\overset{(a)}{=} \sum_{e\in E(G)} \phi(e)\big(\mathbf{1}(e\in\delta^+(X)) - \mathbf{1}(e\in\delta^-(X))\big) \\
&= \sum_{e\in E(G)} \phi(e)\mathbf{1}(e\in\delta^+(X)) - \sum_{e\in E(G)} \phi(e)\mathbf{1}(e\in\delta^-(X)) \\
&= \sum_{e\in\delta^+(X)} \phi(e) - \sum_{e\in\delta^-(X)} \phi(e),
\end{aligned}$$

where (a) uses the following observation: For all edge e,

$$\mathbf{1}(e \text{ has tail in } X) - \mathbf{1}(e \text{ has head in } X) = \mathbf{1}(e\in\delta^+(X)) - \mathbf{1}(e\in\delta^-(X)).$$

This shows the assertion. □

Lemma 2.8.5. *Let ϕ be an integer-valued (s,t)-flow of total value k in a digraph G. Then there are k directed paths $P_1,\dots,P_k$ of G from s to t so that every edge e is in at most $\phi(e)$ of them.*

PROOF. Since ϕ is integer-valued, so is the total value k. We will proceed by an induction on k. We may assume $k>0$ since there is nothing to show if $k=0$.

We first claim that

$$\exists \text{ directed path } P_1 \text{ from } s \text{ to } t \text{ s.t. } \phi(e)>0 \text{ for all } e\in E(P_1). \tag{6}$$

Indeed, take X to be the set of nodes v such that there is a directed path P from s to v such that $\phi(e)>0$ for all $e\in E(P)$. If $t\in X$, then (6) holds. Suppose for contradiction that $t\notin X$. Then by

Lemma 2.8.4 (and recall that ϕ only takes nonnegative values),

$$\sum_{e\in\delta^+(X)} \phi(e) = \|\phi\| + \sum_{e\in\delta^-(X)} \phi(e) \geq \|\phi\| = k > 0.$$

It follows that there must be an edge $e = xy \in \delta^+$ such that $\phi(e) > 0$. But then $y \in X$ since there is a directed path P from s to x with $\phi(e) > 0$ for all $e \in E(P)$, and appending e to P extends such path to y. By definition of $\delta^+(X)$, the head y must not be in X, a contradiction. This shows (6).

Now let P_1 be as in (6). Define $\phi'(e) = \phi(e) - 1$ if $e \in E(P_1)$, and $\phi'(e) = \phi(e)$ if $e \in E(G) \setminus E(P_1)$. Then ϕ' is an integral (s,t)-flow of total value $k-1$. By the induction hypothesis, directed paths $P_2, \ldots, P_k$ exist so that every edge e is in at most $\phi'(e)$ of them; and then every edge e is in at most $\phi(e)$ of $P_1, \ldots, P_k$, as required. □

Definition 2.8.6 (Max-flow problem)**.** Let s, t be distinct vertices of a digraph G, and let each edge e have a "capacity" $c(e) \in \mathbb{Z}^+$. An (s,t)-flow ϕ is called c-admissible if $\phi(e) \leq c(e)$ for every edge e of G. We wish to find a c-admissible flow of maximum total value:

$$\underset{\substack{\phi:(s,t)\text{-flow on } G\\ \phi = c\text{-admissible}}}{\arg\max} \|\phi\|. \tag{7}$$

Definition 2.8.7 (Augmenting path for a flow)**.** Let ϕ be an integer-valued c-admissible (s,t)-flow in a digraph G. A path P of G from s to some vertex v (not necessarily a directed path) is called an *augmenting path for* ϕ if for every edge e of P:

(i) if P uses e in the forward direction then $\phi(e) + 1 \leq c(e)$ (i.e., ∃ room to add 1)
(ii) if P uses e in the backward direction then $\phi(e) \geq 1$ (i.e., ∃ room to subtract 1).

Proposition 2.8.8. *Let G, s, t, c be as above, and let ϕ be an integer-valued c-admissible flow. If there is an augmenting path P for ϕ from s to t, then there is an integer-valued c-admissible flow of larger total value.*

PROOF. Let ψ be an (s,t)-flow of total value 1, where $\psi(e) = \pm 1$ for every edge e of P (depending on its direction) and $\psi(e) = 0$ for all other edges of G. Then $\phi + \psi$ is a c-admissible flow of total value larger than the total value of ϕ. □

Proposition 2.8.8 suggests that one can improve the current flow by adding a unit amount along an augmenting path, if there is one. This suggests the following classical greedy algorithm for solving the max-flow problem (7), due to Ford and Fulkerson. See Fig. 2.8.2. Also see Fig. 2.8.3 for an example of using this algorithm to find a max-flow.

```
function: FordFulkerson(Graph G,Node S,Node T):
    Initialise flow in all edges to 0
    while (there exists an augmenting path(P) between S and T in residual network graph):
        Augment flow between S to T along the path P
        Update residual network graph
    return
```

FIGURE 2.8.2. Pseudocode for Ford-Fulkerson max-flow algorithm.

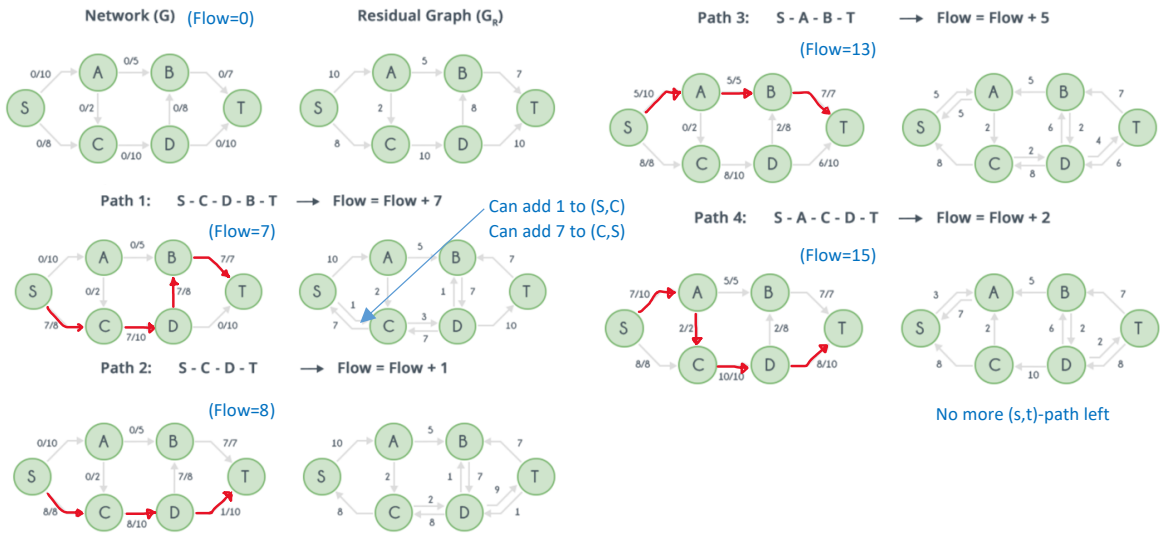


FIGURE 2.8.3. An example of finding max flow by Ford-Fulkerson algorithm.

Definition 2.8.9 (Min-cut)**.** Let G, s, t, c be as before. For each subset of nodes $X \subseteq V$ such that $s \in X$ and $t \notin X$, the set of edges $\delta^+(X)$ is called a (s,t)-*cut*. An (s,t)-cut is called a *min-cut* if it minimizes the quantity $\sum_{e\in\delta^+(e)} c(e)$ (i.e., a smallest bottleneck).

Theorem 2.8.10 (The max flow – min cut theorem, Ford and Fulkerson)**.** *Let G, s, t, c be as before, and let $k \geq 0$ be an integer. Then the following strong duality between max-flow and min-cut holds:*

$$\max_{\substack{\phi:(s,t)\text{-flow on } G \\ \phi = c\text{-admissible}}} \|\phi\| = \min_{\substack{X\subseteq V(G) \\ s\in X,\ t\notin X}} \sum_{e\in\delta^+(X)} c(e). \tag{8}$$

In particular, for each integer $k \geq 0$, exactly one of the following holds:

(i) *There is a c-admissible flow in G from s to t of total value $\geq k$*

(ii) *There exists $X \subseteq V(G)$ with $s \in X$ and $t \notin X$ so that $\sum_{e\in\delta^+(X)} c(e) < k$.*

PROOF. Choose an integer-valued c-admissible flow ϕ from s to t of maximum total value. By Proposition 2.8.8, there is no augmenting path for ϕ from s to t. Let X be the set of all vertices v so that there is an augmenting path from s to v. Then $s \in X$ and $t \notin X$. Also, $\phi(e) = 0$ for all $e \in \delta^-(X)$, and $\phi(e) = c(e)$ for every $e \in \delta^+(X)$, so ϕ has total value $\sum_{e\in\delta^+(X)} c(e)$ by Lemma 2.8.4. Hence (8) holds. That exactly one of **(i)** and **(ii)** hold follows from (8) easily. □

Exercise 2.8.11 (Deriving edge version of Menger's theorem max-flow-min-cut)**.** Derive the edge version of Menger's theroem in Exercise 2.7.12 from the max-flow-min-cut theorem (Thm. 2.8.10).

Hint: We simply make G into a flow network by defining $c(u,v) = 1$ for all directed edges $(u,v) \in E(G)$; if G is undirected, then we simply set $c(u,v) = c(v,u) = 1$ for all $(u,v) \in E(G)$. The result now follows from two claims:

(A) An integer (s,t)-flow of value k implies the existence of k edge-disjoint (s,t)-paths, and vice versa.

(B) A cut of capacity k implies the existence of an (s,t)-edge cut of cardinality k, and vice versa.

Exercise 2.8.12 (Deriving Menger's theorem max-flow-min-cut)**.** Derive Menger's theroem in Theorem 2.7.7 from the max-flow-min-cut theorem (Thm. 2.8.10).

Hint: The conversion uses some small "gadgets". Every vertex v in G is transformed into a pair of vertices $v_{\text{in}}, v_{\text{out}}$, with $c(v_{\text{in}}, v_{\text{out}}) = 1$ and $c(v_{\text{out}}, v_{\text{in}}) = 0$. Every edge (u,v) in G is transformed into an edge from u_{out} to v_{in} with infinite capacity. In the undirected case, we also

create an edge of infinite capacity from v_{out} to u_{in}. Now we solve the max-flow problem with source s_{out} and sink t_{in}. The result now follows from two claims:

(A) An integer s_{out}-t_{in} flow of value k implies the existence of k vertex-disjoint s-t paths, and vice versa.

(B) A cut of capacity k implies the existence of an s_{out}-t_{in} vertex cut of cardinality k, and vice versa.

Exercise 2.8.13 (Deriving Kőnig's theorem from max-flow-min-cut)**.** Derive Kőnig's theorem (Thm. 2.6.8) from the max-flow-min-cut theorem (Thm. 2.8.10).

Hint: We make a bipartite graph into a flow network by attaching a "super-source" to one side and a "super-sink" to the other side. Specifically, if G is our bipartite graph, with two vertex sets X, Y, and edge set E, then we define a flow network $G^\wedge = (X \cup Y \cup \{s,t\}, c, s, t)$ with the following edge capacities:

$$\begin{aligned} c(s,x) &= 1 \quad \text{for all } x \in X \\ c(y,t) &= 1 \quad \text{for all } y \in Y \\ c(x,y) &= \infty \quad \text{for all } (x,y) \in E \\ c(x,y) &\equiv 0 \quad \text{otherwise.} \end{aligned}$$

The result now follows from the claims below:

(A) For any integer flow in this network, the amount of flow on any edge is either 0 or 1. (*Hint:* local balance equation.)

(B) The set of edges (x,y) such that $x \in X$, $y \in Y$, and $f(x,y) = 1$ constitutes a matching in G whose cardinality is equal to $|f|$.

(C) The maximum flow value equals the size of the maximal matching.

(D) If (S,T) is any finite-capacity (s,t)-cut in this network, let $A = (X \cap T) \cup (Y \cap S)$. The set A is a vertex cover in G.

(E) A vertex cover A gives rise to an (s,t)-cut with capacity $|A|$.

Exercise 2.8.14. Let G be a digraph and for each edge e let $\phi(e) \geq 0$ be an integer, so that for every node v, the local balance condition holds:

$$\sum_{e \in \delta^-(v)} \phi(e) = \sum_{e \in \delta^+(v)} \phi(e).$$

Show there is a list $C_1, \ldots, C_n$ of directed cycles (possibly with repetition) so that

$$|\{i : 1 \leq i \leq n,\, e \in E(C_i)\}| = \phi(e) \qquad \text{for all } e \in E(G).$$

(*Hint:* Use induction on $R := \sum_{e \in E(G)} \phi(e)$. For $R \geq 1$, take a directed edge $f = (t,s)$ with $\phi(f) \geq 1$. Then ϕ restricted on $E(G) \setminus \{f\}$, say ϕ', defines an (s,t)-flow on $G' := G \setminus f$. Show the total value of ϕ' is positive. Show that there is a directed (s,t)-path P. Then combining P with the directed edge f gives a directed cycle. Subtract 1 from the flow values along this cycle and complete the induction.)

Exercise 2.8.15 (Approximating by integer-value flow)**.** Let s, t be vertices of a digraph G, and let $\phi : E(G) \to [0,\infty)$ be an (s,t) flow. Show that there is an (s,t) flow $\psi : E(G) \to \mathbb{Z}_{\geq 0}$ so that

(1) its total value is at least that of ϕ; and

(2) $|\psi(e) - \phi(e)| < 1$ for every edge e of G.

2.9. Applications of graphs to image segmentation

2.9.1. Background and introduction. Humans have an impressive skill for recognizing objects in images, a capability that currently surpasses image processing algorithms. *Segmentation*, which

involves dividing an image into meaningful regions, aims to replicate this ability. In an ideal segmentation, all pixels within a region should be assigned a distinct label. A commonly used pipeline is as follows:

1. Get an “over-segmentation" by constructing “superpixels" These superpixels then serve as a basis for more sophisticated algorithms such as conditional random fields (CRF). Four commonly used superpixel algorithms are: (See Figure 2.9.1)
   (1) Felzenszwalb’s efficient graph based segmentation
   (2) Quickshift image segmentation
   (3) Simple Linear Iterative Clustering (SLIC) : K-Means based image segmentation
   (4) Compact watershed segmentation of gradient images
2. Represent the over-segmentation by Region Adjacency Graphs (RAGs).
3. Apply various methods to RAGs to obtain segmentation.

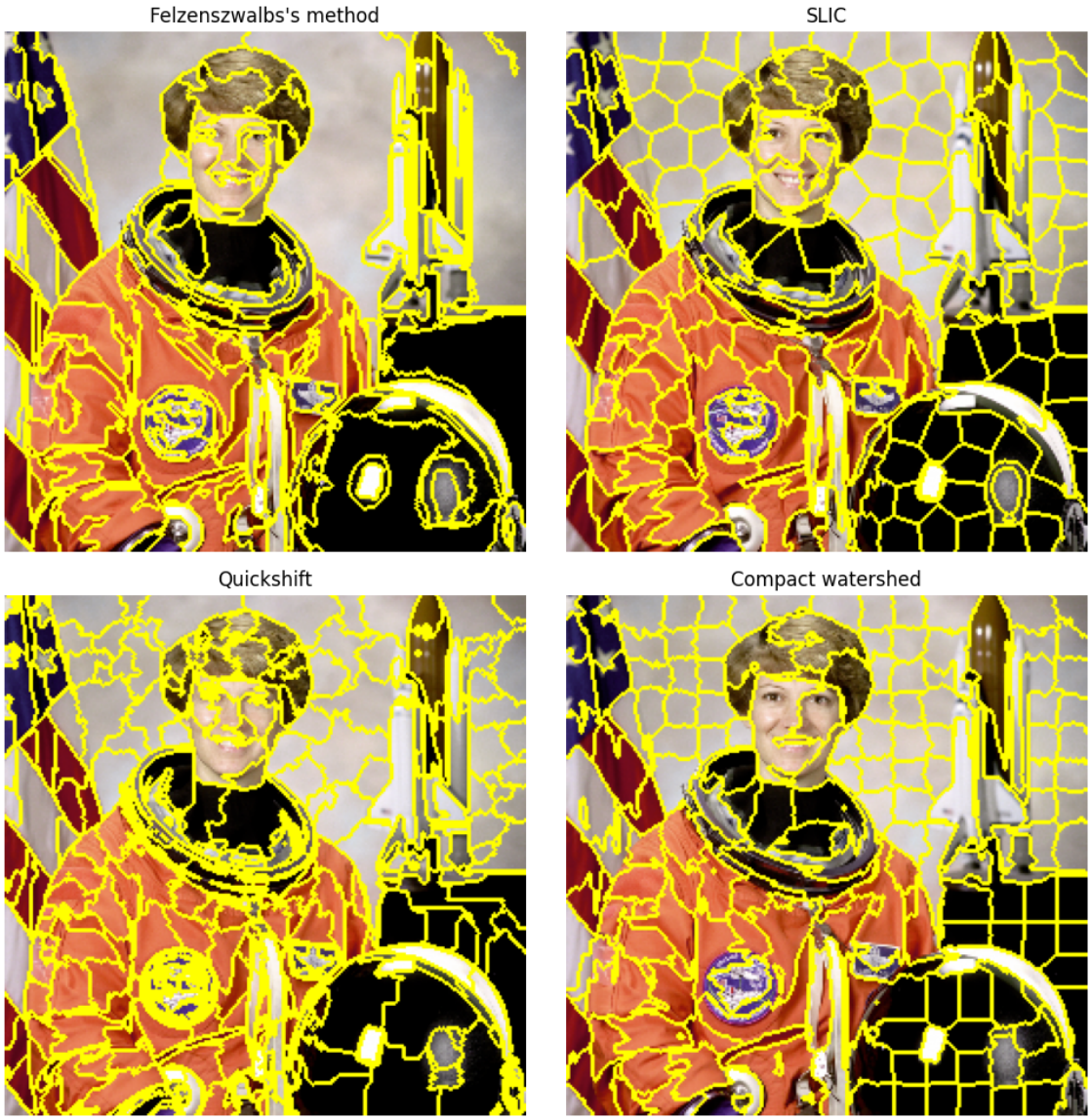


FIGURE 2.9.1. Example of four popular superpixel algorithms. Source: Link

In this section, we use the Python package `scikit-image` to demonstrate how graph theory is used in image segmentation. The existing segmentation functions in `scikit-image` are quite detailed and align with 'superpixel methods'. offering a foundational approach to segmentation. Region Adjacency Graphs (RAGs) are widely used data structure in various segmentation algorithms.

2.9.2. Example. The example in this section is based on Link.

Consider the image in Figure 2.9.2 left. We segment this image using SLIC algorithm. The SLIC algorithm is simply the K-means clustering algorithm done in 5-dimensional space (R, G, B, x, y) using `skimage.segmentation.slic()`. It will compute a desired number of superpixels and obtain an over-segmentation as shown in Figure 2.9.2 right. Each superpixel is a localized cluster of pixels sharing some similar property, in this case their color. The label of each pixel is stored in the labels array.

Next, we compute the RAG from the over-segmentation. Region Adjacency Graphs, as the name suggests, represent adjacency of regions with a graph. Each superpixel becomes a node in RAG. We put an edge between every pair of adjacent regions (regions whose pixels are adjacent). The weight between every two nodes can be defined in a several ways. For this example, we will use the difference of average colors between two regions as their edge weight. The more similar the regions, the less the weight between them. Because we are using difference in mean color to compute the edge weight, the method has been named `rag_mean_color` (default `mod="distance"`). A minimal example of code that produces the RAG in Figure 2.9.2 is shown in Figure 2.9.5. `scikit-image` implements RAG as the familiar `networkx.Graph` object, which we can handle in various ways. For instance, we can draw it as a standalone weighted graph as in Figure 2.9.4, although without the background image, it is hard to interpret what it represents.

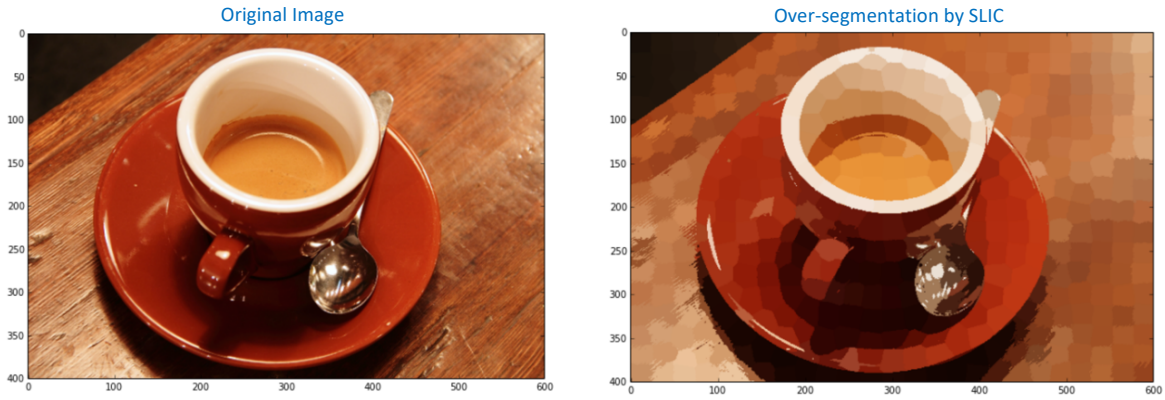


FIGURE 2.9.2. Example of over-segmentation by SLIC.

Now that we have an RAG representing the over-segmentation in Figure 2.9.2, we can obtain further segmentation via various methods applied to the RAG. The simplest option is by edge thresholding. For instance, if we threshold the edges at level 29, we obtain the RAG shown in Figure 2.9.6 left.

Another interseting way for image segmentation using RAG is by using *normalized cut* [SM00]. Namely, gven an image's labels and its similarity RAG, one recursively performs a 2-way normalized cut on it. All nodes belonging to a subgraph that cannot be cut further are assigned a unique label in the output. Usual min-cut tends to prefer cutting isolated nodes, which is not desired for the purpose of image segmentation. On the contrary, normalized cut normalizes the usual cut cost by the sum of all outgoing edge weights to avoid this. Given a graph $G = (V, E)$ with (weighted)

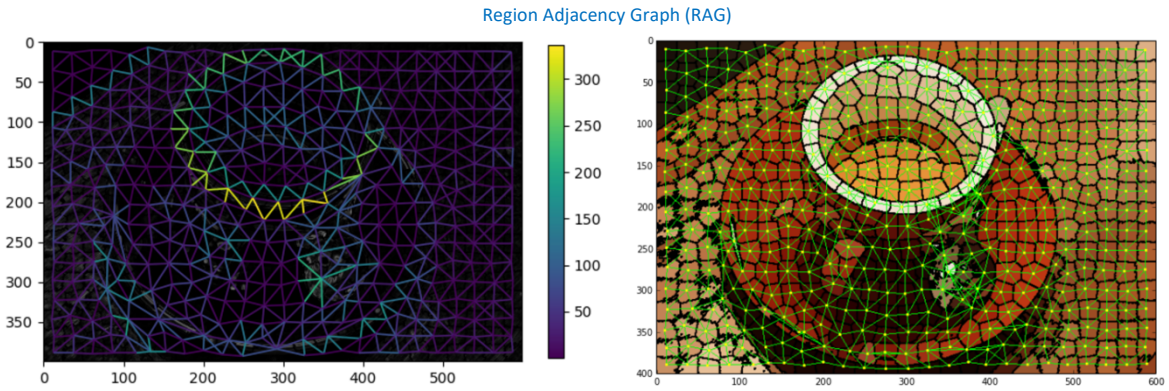


FIGURE 2.9.3. Example of over-segmentation and the corresponding RAG.

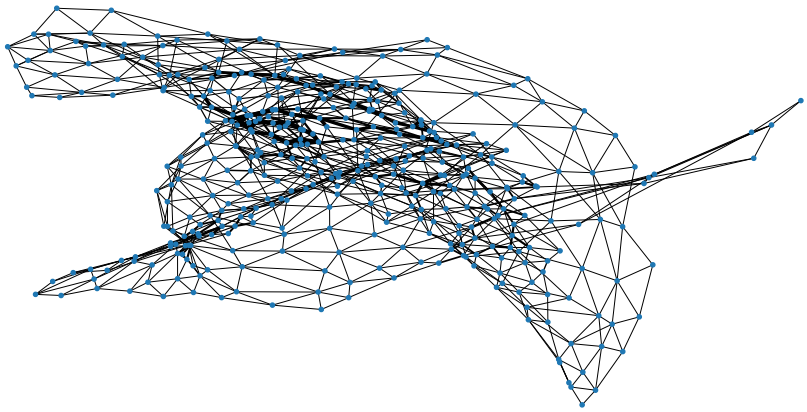


FIGURE 2.9.4. The RAG in Figure 2.9.3 drawn as a weighted graph.

```
from skimage import graph
from skimage import data, segmentation, color, filters, io
from matplotlib import pyplot as plt


img = data.coffee()
gimg = color.rgb2gray(img)

labels = segmentation.slic(img, compactness=30, n_segments=400, start_label=1)
edges = filters.sobel(gimg)
edges_rgb = color.gray2rgb(edges)

g = graph.rag_mean_color(img, labels)
lc = graph.show_rag(labels, g, edges_rgb, img_cmap=None, edge_cmap='viridis',
                    edge_width=1.2)

plt.colorbar(lc, fraction=0.03)
io.show()
```

FIGURE 2.9.5. Example of over-segmentation by SLIC.

adjacency matrix A, the normalized cut problem is the following:

$$\min_{A,B\subseteq V}\left(\mathrm{NCut}(A,B) := \frac{\mathrm{Cut}(A,B)}{\mathrm{Cut}(A,V)} + \frac{\mathrm{Cut}(A,B)}{\mathrm{Cut}(B,V)}\right),$$

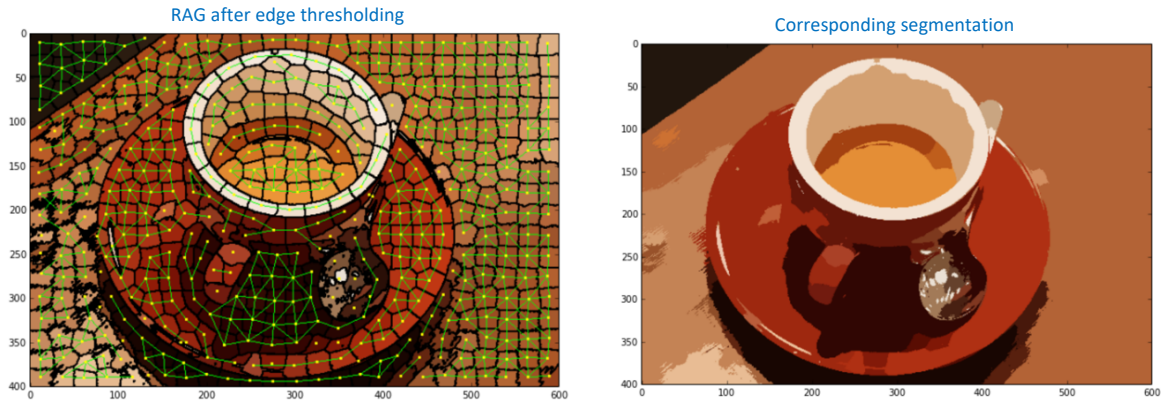


FIGURE 2.9.6. Image segmentation by thresholding RAG.

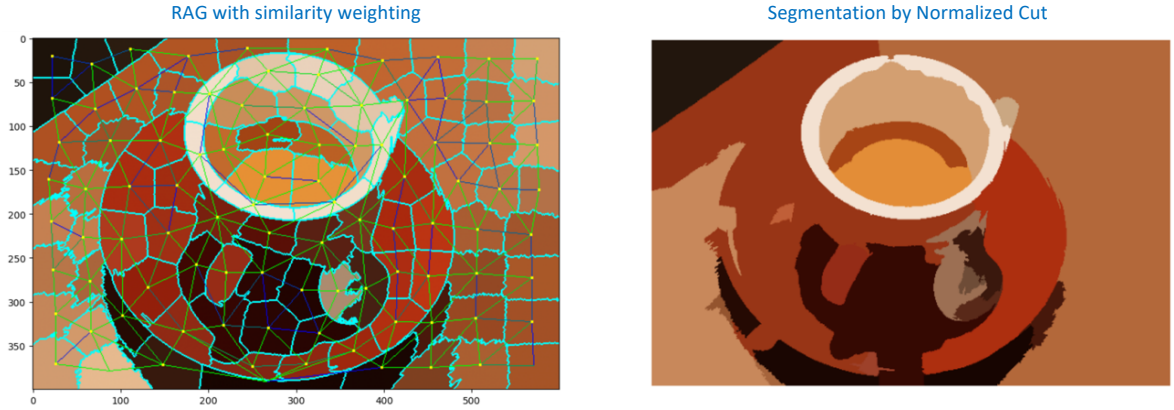


FIGURE 2.9.7. Image segmentation by similarity-based RAG (left) with blue=similar and green=not similar and segmentation based on normalized cut (right).

where for any two subsets $C, D \subseteq V$,

$$\mathrm{Cut}(C,D) := \sum_{(u,v)\in C\times D} A(u,v).$$

The normalization by $\mathrm{Cut}(A,V)$ and $\mathrm{Cut}(B,V)$ helps to avoid extremely skewed cut (e.g., isolating a single node), which is not preferred for proper image segmentation. See Figure 2.9.9.

The basis of this algorithm is the *Minimum Cut Algorithm*, which divides a graph into two parts, A and B such that the weight of the edges going from nodes in Set A to the nodes in Set B is minimized. For the min-cut to work properly, we need to define the weights of our RAG in such a way that *similar regions have larger weight* so that we avoid cutting edges connecting similar regions. This way, removing lesser edges would leave us with edges connecting the similar regions. This can be achieved by the same function `rag_mean_color` with option `mode="similarity"`. See Figure 2.9.7 left for RAG obtained in this way. (There, blue=similar and green=not similar).

There also max-flow-based image segmentation methods [IB20, YBT10].



FIGURE 2.9.8. Image segmentation by thresholding or normalized cut on RAG.

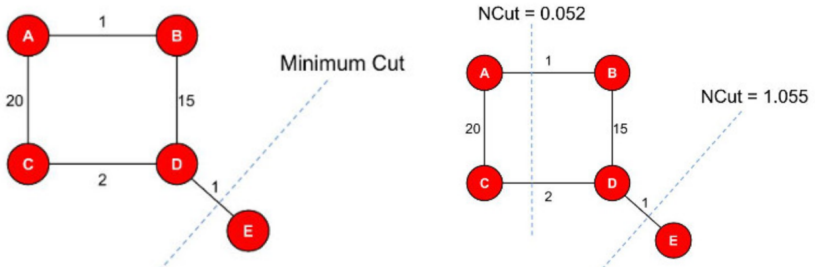


FIGURE 2.9.9. Illustration of normalized cut.

CHAPTER 3

Essentials in Network models

3.1. Properties of real-world networks

3.1.1. Six degrees of separation: Short average path length. The concept of *six degrees of separation* suggests that any person can be linked to any other individual through a chain of social connections that are six steps or fewer. In other words, you can connect any two people using a sequence of "friend of a friend" relationships within a maximum of six steps. This concept is commonly referred to as the "six handshakes rule".

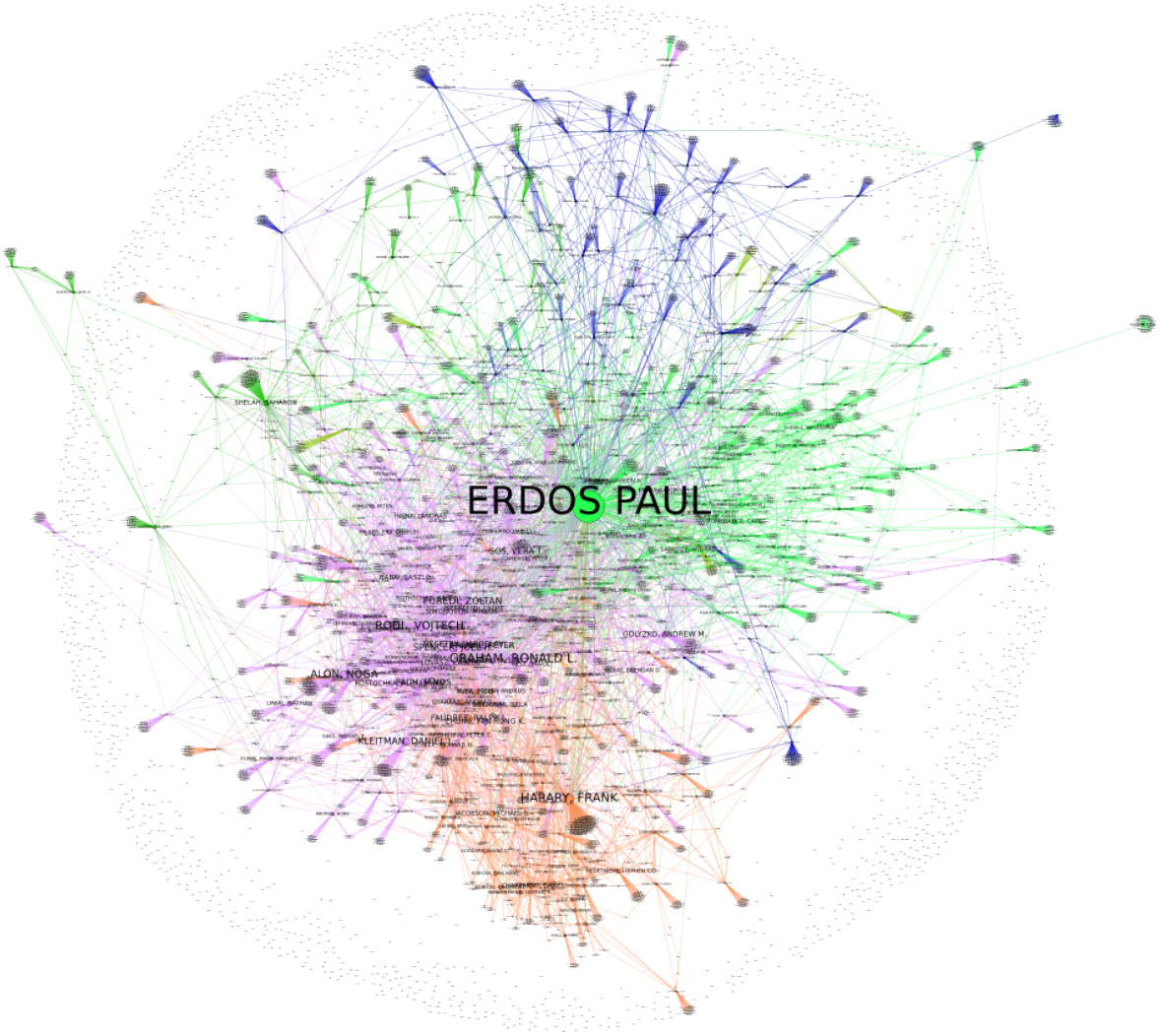


FIGURE 3.1.1. Collaboration network in mathematics. Shortest-path distance to Paul Erdős (1913-1996) is known as the "Erdős number".

In mathematics community, this notion is perhaps best captured by Erdős number. Think of the collaboration network of all mathematicians, where edges between two mathematicians represent having a joint published paper. See Fig. 3.1.1. In this network, there is a dominant hub node for the prominent and extremely collaborative Hungarian mathematician Paul Erdős (1913-1996). Erdős published over 1,500 paper during his lifefime in various fields including combinatorics, graph theory, number theory, analysis, approximation theory, set theory, and probability theory. Because he collaborated with so many mathematicians, it does not take too many edges to cross to reach him from anywhere in the network. The shortest-path distance (see Def. 3.1.1) to Erdős from a mathematician is called his/her Erdős number. For instance, mathematician H. Lyu's Erdős number is only 3! (see Fig. 3.1.2)

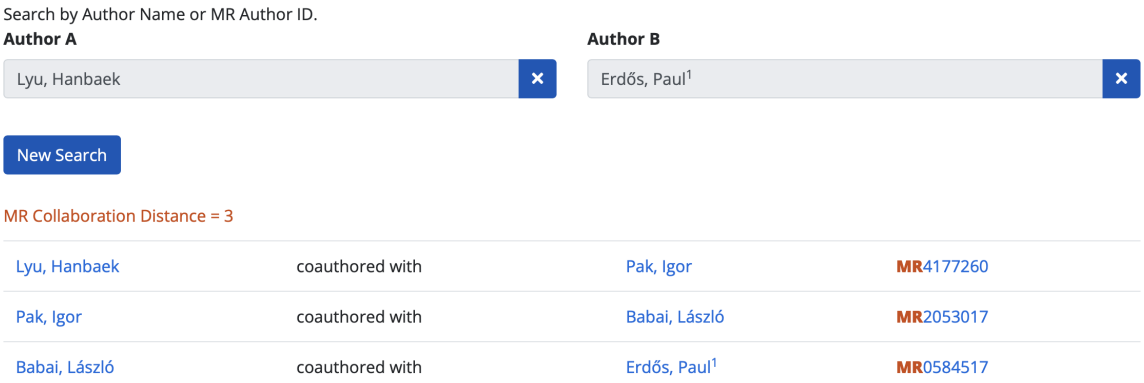


FIGURE 3.1.2. Mathematition H. Lyu's Erdös number is 3.

Definition 3.1.1 (Shortest-path distance)**.** Let $G = (V, E)$ be a graph. The *shortest-path* distance between two nodes u, v, denoted as $d(u, v)$, is the number of edges in the shortest (u, v)-path in G:

$$d(u, v) := \min_{P:(u,v)\text{-path in } G} |E(P)|.$$

Exercise 3.1.2 (Graph as a finite metric space)**.** Let $G = (V, E)$ be a graph and let d denote the shortest-path distance on G. Show that the pair (V, d) is a finite metric space in the following sense:

(i) (identifiability) $d(v, v) = 0$ for all $v \in V$;
(ii) (positivity) $d(u, v) > 0$ for all distinct nodes u, v in G;
(iii) (symmetry) $d(u, v) = d(v, u)$ for all nodes $u, v \in V$;
(iv) (triangle inequality) $d(u, w) \leq d(u, v) + d(v, w)$ for all nodes $u, v, w \in V$.

Exercise 3.1.3 (Average path length and small-worlds)**.** Let $G = (V, E)$ be a graph. Its *average path length* is defined by

$$d_{avg}(G) := \mathbb{E}[d(u, v)]$$

where u, v are random nodes in G chosen independently and uniformly at random. In a rouph sense, we say G has a "small-world" property if

$$d_{avg}(G) = O(\log |V|).$$

(i) Let P_n, the path of n nodes. Compute $d_{avg}(P_n)$. Is this a small-world? (*Hint:* When (X,Y) is uniformly distributed over $\{1,\dots,n\}^2$, compute

$$\begin{aligned}\mathbb{E}[|X-Y|] &= n^{-2}\left(0\cdot n + 1\cdot(n-1)+\cdots+(n-1)\cdot 1 + 1\cdot(n-1)+\cdots+(n-1)\cdot 1\right)\\ &= 2n^{-2}\sum_{k=1}^{n} k(n-k)\\ &= 2n^{-2}\left(\frac{n^2(n+1)}{2}-\frac{n(n+1)(2n+1)}{6}\right)=\frac{n}{3}+O(1).\end{aligned}$$

Then show that $d_{avg}(P_n)=\mathbb{E}[|X-Y|]$.)

(ii) Let G_n be the $n\times n$ grid graph. Compute $d_{avg}(P_n)$. Is this a small-world? (*Hint*: Represent each node by a pair (x,y), where $x,y\in\{1,2,\dots,n\}$. In order to choose a node uniformly at random, choose X,Y independently and uniformly at random from $\{1,\dots,n\}$ and take the resulting random node (X,Y). The shorest-path distance between two nodes (x,y) and (z,w) is their L_1-distance: $|x-z|+|y-w|$. Then use linearity of expectation with the computation in part **(i)**.)

(iii) Let T_n be the binary tree with n nodes. Show that T_n has a small-world property. (*Hint*: Show that there are total $\log_2 n$ levels (up to rounding error). Show that the diameter of T_n (i.e., the largest shortest-path distance between two nodes) is at most $2\log_2 n$. Deduce that $d_{avg}(T_n)=O(\log n)$.)

3.1.2. Existence of hub nodes: Power-law degree distribution. A *scale-free* network is characterized by a degree distribution that adheres to a power law, especially in the asymptotic sense. This means that, as the number of connections (degree) of nodes in the network increases significantly (for large values of k), the fraction $p(k)$ of nodes with k connections follows a power-law relationship expressed as $p(k)\sim k^{-\nu}$, where ν is a parameter typically falling within the range $2<\nu<3$. This range signifies that the second moment (scale parameter) of $k^{-\nu}$ is infinite, while the first moment is finite[1]. In other words, let G be a large scale-free network with n nodes, and choose one node v uniformly at random. Its degree $\deg(v)$ is a random variable. Then the scale-freeness of G means that the variance of the degree $\deg(v)$ of a random node v is extremely large:

$$\mathrm{Var}(\deg(v))\approx\infty.$$

This means that there exists nodes with extremely large degree, which are the hub nodes.

The term "scale-free" implies that certain moments of the degree distribution are undefined, indicating that the network lacks a characteristic scale or "size."

3.1.3. Communities. In the study of complex network, a network is considered to possess *community structure* if its nodes can be categorized into sets (which may potentially overlap) in a way that each set exhibits internal connections stronger than external, inter-community connections. In the case of non-overlapping community identification, this suggests that the network naturally segregates into clusters of nodes with tightly-knit internal connections and looser connections between these clusters. It is typical to observe such community structure in real-world networks. However, overlapping communities are also permissible.

The broader definition of community structure is rooted in the concept that pairs of nodes are more likely to be linked if they both belong to the same community or communities, and less likely to be connected if they do not share communities. Another related yet distinct challenge is community detection, where the objective is to determine the community to which a specific vertex belongs.

[1] However, ν may occasionally fall outside these boundaries.

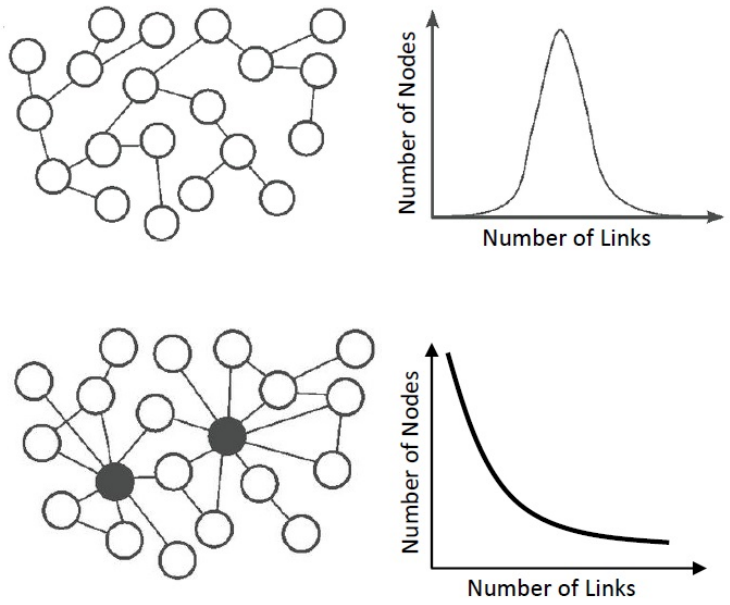


FIGURE 3.1.3. (Top) A sample of Erdős-Renyí random graph. It has binomial degree distribution, which is asymptotically normal. (Bottom) A sample of scale-free network, which has a power-law degree distribution.

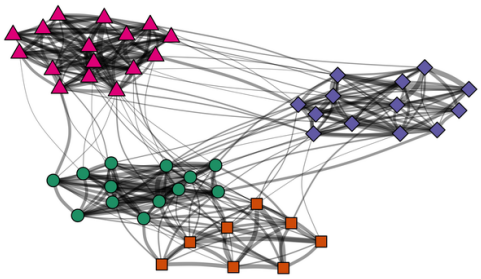


FIGURE 3.1.4. A network with three communities.

3.2. Erdős-Renyí random graphs

In this section, we start with the prototypical model of all random networks, the Erdős-Renyí random graphs. Its importance in combinatoris, graph theory, and probability theory cannot be overstated.

Definition 3.2.1 (Erdős-Renyí random graphs)**.** Construct a random graph with n nodes in the following manner. For each pair of distinct nodes (i, j), include an edge ij independently with probability p and leave it as a non-adjacent pair with probability $1-p$. The resulting random graph is denoted as $G(n, p)$ and is called a *Erdős-Renyí random graph.*

Every edge in $G(n, p)$ is generated radomly. In particular, the number $|E|$ of edges in $G(n, p)$ is a random variable, so its edge density is also random. What kind of random variables are they?

Proposition 3.2.2 (Edge density in ER graph)**.** *Let $G = ([n], E)$ be an ER-random graph with distribution $G(n, p)$. Let $N := \binom{n}{2}$, the maximum number of edges in G.*

(i) *The number $|E|$ of edges in G is a random variable with* Binomial(N, p) *distribution.*
(ii) *The expected number of edges in G is Np. Moreover, the expected edge density is p.*

PROOF. Note that

$$|E| = \sum_{1 \le i < j \le n} \mathbf{1}(ij \in E) \tag{9}$$

Each indicator variable in the above summation is an independent Bernoulli(p) variable according to the definition of $G(n,p)$. There are $\binom{n}{2}$ summands above. Using the fact that the sum of m independent Bernoulli(p) variables (see A.1.5) is a Binomial(m,p) variable (see A.2.1), **(i)** follows.

For **(ii)**, we can take the expectation in (9) and use linearity of expectation to get

$$\begin{aligned}\mathbb{E}[|E|] &= \sum_{1 \le i < j \le n} \mathbb{E}[\mathbf{1}(ij \in E)] \\ &= \sum_{1 \le i < j \le n} p \\ &= Np,\end{aligned}$$

where we have used the fact that the expectation of a Bernoulli(p) variable is $1 \cdot p + 0 \cdot (1-p) = p$. Lastly, the edge density in G is (note that $G(n,p)$ may have self-loops)

$$\frac{|E|}{N}.$$

Taking expectation, we get that the expected edge density of G is p. □

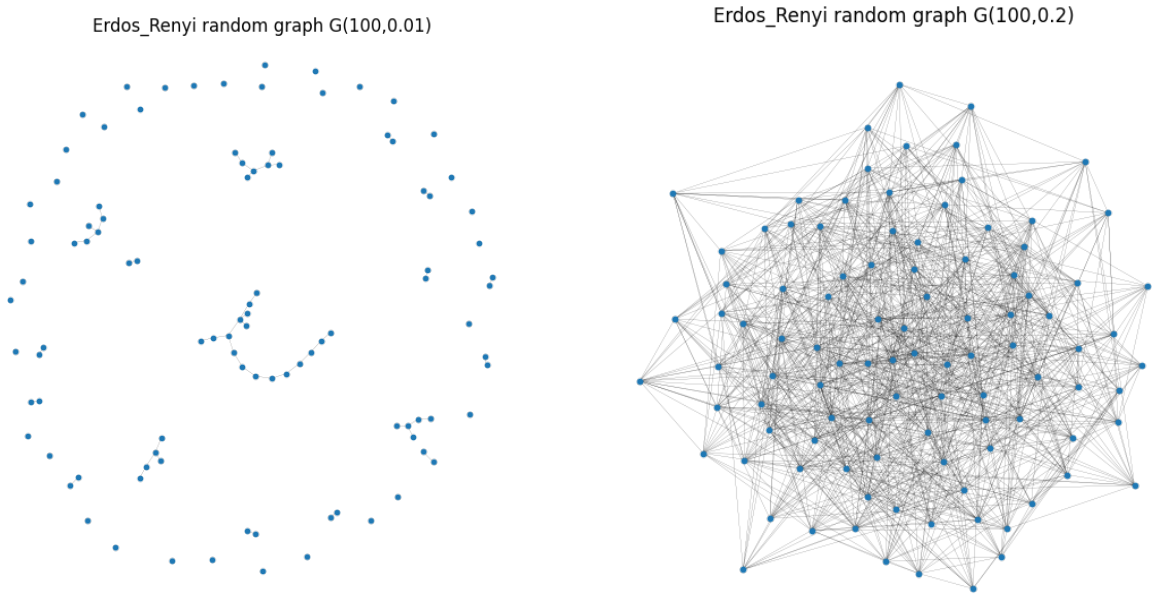


FIGURE 3.2.1. Samples of Erdős-Renyí random graphs.

Definition 3.2.3 (Degree distribution)**.** Let $G = (V,E)$ be a graph. The *degree distribution* of G is a probability mass function $\mathrm{Deg}_G = (p_0, p_1, \ldots)$ on the integers $\{0,1,\ldots\}$ such that

$$p_k(G) = \text{fraction of nodes in } G \text{ with degree } k.$$

Exercise 3.2.4 (Probabilistic definition of degree distribution)**.** Let $G = (V,E)$ be a graph and let $\mathrm{Deg}_G = (p_0, p_1, \ldots)$ be its degree distribution. Show that for each $k \ge 0$,

$$p_k(G) = \mathbb{P}\left(\text{A uniformly chosen node in } G \text{ has degree } k\right).$$

Exercise 3.2.5 (Normal approximation of the expected degree distribution of $G(n,p)$)**.** Consider the Erdős-Renyí random graph $G = G(n,p)$.

(i) For each node v and an integer $0 \le k < n$, show that $\deg_G(v) \sim \text{Binom}(n-1, p)$:

$$\mathbb{P}\left(\deg_{G(n,p)}(v) = k\right) = \binom{n-1}{k} p^k (1-p)^{n-1-k} \quad \text{for } k = 0, 1, \ldots, n.$$

Use Exercise 3.2.4 to deduce that the expected degree distribution of $G(n,p)$ is $\text{Binom}(n-1, p)$:

$$\mathbb{E}_G[p_k(G)] = \binom{n}{k} p^k (1-p)^{n-1-k} \quad \text{for } k = 0, 1, \ldots, n.$$

(*Hint*: Let U be a uniformly chosen random node in G. For any fixed node u, show that the conditional probability $\mathbb{P}(\deg_G(U) \mid U = u) = \mathbb{P}(\text{Binom}(n,p) = k)$. Then use iterated expectation: (see Exc. A.6.4)

$$\mathbb{E}_G[p_{\le k}(G)] = \mathbb{E}_G\left[\mathbb{P}(\deg_G(U) \le k \mid G)\right] = \mathbb{P}(\deg_G(U) \le k) = \mathbb{E}_U\left[\mathbb{P}\left(\deg_G(U) = k \,\middle|\, U\right)\right].$$

The conditional probability inside the expectation in U equals $\mathbb{P}(\text{Binom}(n,p) = k)$, which does not depend on U.)

(ii) Let $U \sim \text{Uniform}(\{1, \ldots, n\})$. Use the central limit theorem to deduce

$$\lim_{n\to\infty} \mathbb{P}\left(\frac{\deg_G(U) - np}{\sqrt{np(1-p)}} \le x\right) = \mathbb{P}(N(0,1) \le x),$$

where $N(0,1)$ denote the standard normal random variable. Informally, the expected degree distribution of $G(n,p)$ is asymptotically $N(np, np(1-p))$. (*Hint:* Here we are using the central limit theorem (Thm. A.7.3) to approximate a binomial distribution by a normal distribution (see Exc. A.7.4).) See Figure 3.2.2.

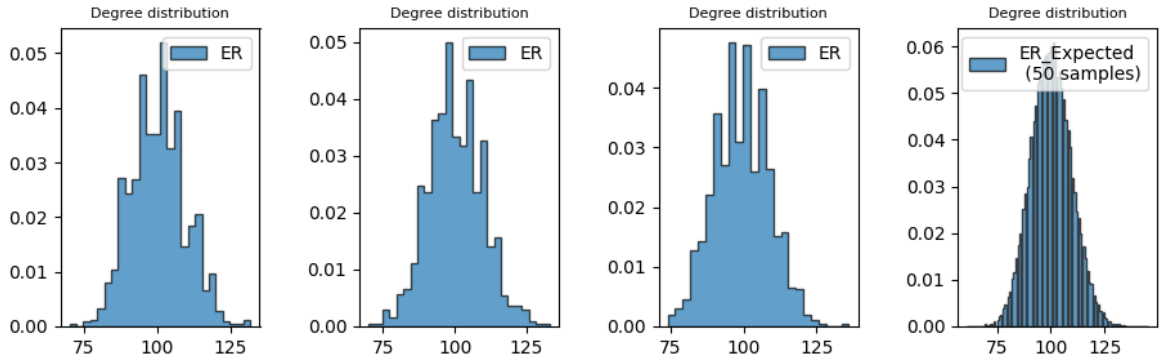


FIGURE 3.2.2. Degree distribution of three $G(n,p)$ graphs with $n = 1000$ and $p = 0.1$. At far right, we show the average degree distribution of 50 samples of $G(n,p)$ graphs, which is close to Binomial(100, 99).

Exercise 3.2.6 (Number of triangles in $G(n,p)$)**.** Let $T = T(n,p)$ denote the total number of triangles in $G(n,p)$.

(i) For each three distinct nodes i, j, k in G, let $Y_{ijk} := \mathbf{1}(ij, jk, ki \in E)$, which is the indivator variable for the even that there is a triangle with node set $\{i, j, k\}$. Show that

$$Y_{ijk} \sim \text{Bernoulli}(p^3).$$

(ii) Show that we can write

$$T = \sum_{1 \le i < j < k \le n} \mathbf{1}(ij, jk, ki \in E). \tag{10}$$

Deduce that the expected number of triangles is

$$\mathbb{E}[T] = \binom{n}{3} p^3.$$

(iii)* Show that

$$\mathrm{Var}(T) = \binom{n}{3} p^3 + 12\binom{n}{4} p^5 + 30\binom{n}{5} p^6 + 20\binom{n}{6} p^6 - \binom{n}{3}^2 p^6.$$

(*Hint:* First compute $\mathbb{E}[T^2]$ and use the fact that $\mathrm{Var}(T) = \mathbb{E}[T^2] - \mathbb{E}[T]^2$. For computing $\mathbb{E}[T^2]$, use (10) and consider possible cases according to the number of overlapping edges.)

Erdős-Renyí random graph is known to exihibit many interesting *phase transition* behavior, as one properly scales the edge probability p according to the size n and take $n \to \infty$. Here, a 'phase transition' refers to the phenominon that certain property of interest changes drastically as one varies a model parameter. The following well-known result states the sharp phase transition behavior of the connectivity of ER random graphs.

Theorem 3.2.7 (Sharp phase transition in the connectivity of ER graph, Erdős-Renyí '61)**.** *Consider $G(n,p)$ with $p = p_n = \lambda \log n/n$, where $\lambda > 0$ is a fixed constant. Then*

$$\lim_{n\to\infty} \mathbb{P}(G(n,p_n) \text{ is connected}) = \begin{cases} 0 & \text{if } \lambda < 1 \\ 1 & \text{if } \lambda > 1. \end{cases}$$

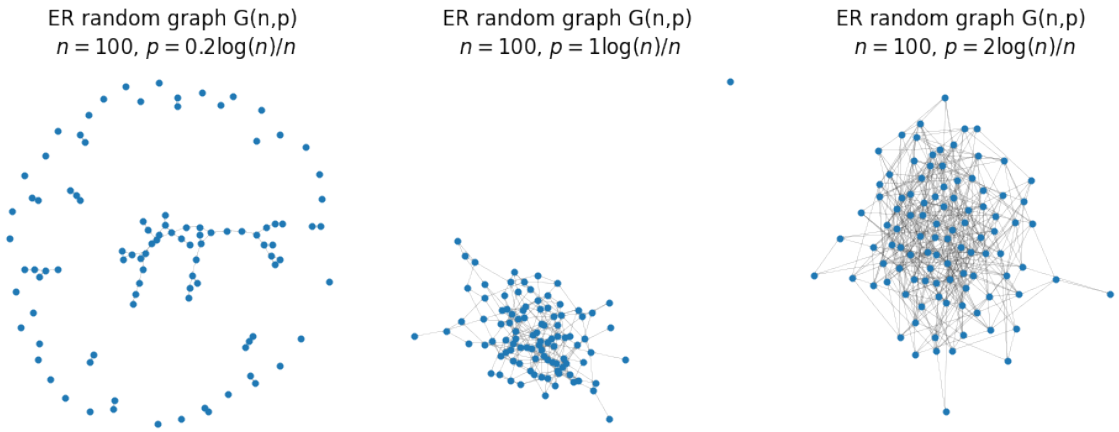


FIGURE 3.2.3. Sharp phase transition of Erdős-Renyí random graphs $G(n,p)$ with $p = \lambda \log n/n$ in λ. The critical value is at $\lambda = 1$.

PROOF OF THEOREM 3.2.7. (Optional*) Suppose $p_n = \frac{\log n + c_n}{n}$, where c_n is a sequence to be specified. Let X_k denote the number of connected components of size k in $G(n,p_n)$. The key is to show

$$\mathbb{P}(G(n,p_n) \text{ is connected}) = \mathbb{P}(X_1 = 0) + o(1). \tag{11}$$

Another important fact that will be used in this proof is that the number of isolated nodes follow a Poisson distribution with mean e^{-c} when $c_n = np - \log n \to c$ (see [FK16, Thm 3.1]). Thus

$$\mathbb{P}(X_1 = 0) = \mathbb{P}\left(\mathrm{Poisson}(e^{-c}) = 0\right) = e^{-e^{-c}}.$$

Combining with (11), we deduce that, when $c_n \to c$,

$$\lim_{n\to\infty} \mathbb{P}(G(n,p_n) \text{ is connected}) = e^{-e^{-c}}.$$

In case $c_n \to \pm\infty$ we use monotonicity of connectivity of ER random graphs in the edge density and the above result to deduce

$$\lim_{n\to\infty} \mathbb{P}(G(n,p_n) \text{ is connected}) = \begin{cases} 0 & \text{if } c_n \to -\infty \\ 1 & \text{if } c_n \to \infty. \end{cases}$$

In particular, if $p_n = \lambda \log n/n = \frac{\log n + c_n}{n}$ with $c_n = (\lambda - 1)\log n$, then $c_n \to \infty$ if $\lambda > 1$ and $c_n \to -\infty$ if $\lambda < 1$, so the result follows.

We now give a bit more detail on how to establish (11). First note that

$$\{G(n,p_n) \text{ is not connected}\} = \bigcup_{k=1}^{n/2} \{X_k \geq 1\}.$$

Since X_1 is the number of isolated nodes,

$$\mathbb{P}(X_1 \geq 1) \leq \mathbb{P}(G(n,p_n) \text{ is not connected}) \leq \mathbb{P}(X_1 \geq 1) + \mathbb{P}\left(\bigcup_{k=2}^{n/2} \{X_k \geq 1\}\right).$$

Note that

$$\mathbb{P}\left(\{1,\dots,k\} \text{ is separated from the rest}\right) = (1-p)^{k(n-k)},$$

since for every pair of nodes between $\{1,\dots,k\}$ and the rest, there should be no edge, which occurs independently with probability $1-p$. Hence by Markov's inequality (see Prop A.8.1),

$$\begin{aligned} \mathbb{P}(X_k \geq 1) &\leq \mathbb{E}[X_k] \\ &\leq \binom{n}{k} \mathbb{P}(\text{A chosen } k\text{-nodes form a connected subgraph})(1-p)^{k(n-k)} \\ &\leq \binom{n}{k} k^{k-2} p^{k-1} (1-p)^{k(n-k)} =: u_k, \end{aligned}$$

where the last inequality uses the fact that there are k^{k-2} spanning trees in K_k and each spanning tree has $k-1$ edges, so

$$\begin{aligned} &\mathbb{P}(\text{A chosen } k\text{-nodes form a connected subgraph}) \\ &\quad \leq \mathbb{P}(\text{At least one spanning tree on } k \text{ nodes have all edges in } G(n,p)) \leq k^{k-2} p^{k-1}. \end{aligned}$$

For $2 \leq k \leq 10$, note that

$$\begin{aligned} u_k &\leq e^k n^k \left(\frac{\log n + c}{n}\right)^{k-1} e^{-k(n-10)\frac{\log n + c}{n}} \\ &\leq (1+o(1)) e^{k(1-c)} \left(\frac{\log n}{n}\right)^{k-1} \\ &\leq (1+o(1)) e^{10(1-c)} \frac{\log n}{n}, \end{aligned}$$

and for $10 < k \leq n/2$,

$$\begin{aligned} u_k &\leq \left(\frac{ne}{k}\right)^k \left(\frac{\log n + c}{n}\right)^{k-1} e^{-k\frac{\log n + c}{2}} \\ &\leq n \left(\frac{e^{1-c/2+o(1)} \log n}{n^{1/2}}\right)^k \end{aligned}$$

Therefore, by a union bound,

$$\begin{aligned}\mathbb{P}\left(\bigcup_{k=2}^{n/2}\{X_k \geq 1\}\right) &\leq \sum_{k=1}^{n/2}\mathbb{P}(X_k \geq 1)\\ &\leq \sum_{k=1}^{n/2} u_k\\ &\leq (1+o(1))e^{10(1-c)}\frac{\log n}{n} + \sum_{10<k\leq n/2} n^{1+o(1)-k/2} = O(n^{o(1)-1}) = o(1).\end{aligned}$$

This shows (11). □

Next, we turn our attention to modeling real-world networks using Erdős-Renyí random graphs. Given an observed network $G = (V, E)$, how should we choose the parameters n and p in $G(n, p)$ so that G is best modeled? Clearly we should set $n = |V|$, otherwise there is no chance that we G is generated from $G(n, p)$. But how about p? A reasonable guess is that we should set

$$\hat{p} = \text{edge density in } G,$$

since p is also the expected edge density in $G(n, p)$ (see Prop. 3.2.2). We will justify this using the framework of maximum lilkelihood. (See Sec. A.9.)

Proposition 3.2.8 (MLE for $G(n, p)$)**.** *The Maximum Likelihood Estimate* $(\hat{n}, \hat{p})$ *for* $G(n, p)$ *with observed graph* $G = (V, E)$ *is* $\hat{n} = |V|$ *and* $\hat{p} =$ edge density in G.

PROOF. We first write the likelihood function for observing G from $G(n, p)$. Without loss of generality, we assume the nodes in G are labeled as $1, 2, \ldots, |V|$. Denote $N = n + \binom{n}{2}$, the total number of all possible edges. Then

$$L(G; n, p) = \mathbf{1}(n = |V|)\, p^{|E|}(1-p)^{N-|E|}. \tag{12}$$

Namely, the likelihood is zero if $n \neq |V|$. Also, there are $|E|$ edges to be observed (each with prob. p) and $N - |E|$ edges not to be observed (each with prob. $1 - p$). All of these events are independent. This justifies the likelihood in (12). Then, by the definition of MLE,

$$(\hat{n}, \hat{p}) = \operatorname*{arg\,max}_{n,p} \mathbf{1}(n = |V|)\, p^{|E|}(1-p)^{N-|E|}.$$

From this we get $n = |V|$. Suppose $|V| = n$. For $\hat{p}$, taking the log likelihood,

$$\hat{p} = \operatorname*{arg\,max}_{p\in[0,1]} \big(|E| \log p + (N - |E|)\log(1-p)\big). \tag{13}$$

Taking derivative of the log likelihood function in p and setting it equal to zero,

$$\frac{|E|}{p} - \frac{N-|E|}{1-p} = 0.$$

Solving the above equation in p, we see that there is a unique critical point for (13) at $p = |E|/N$. The only other extreme points are $p = 0, 1$, at which clearly (13) is not maximized. This shows $\hat{p} = |E|/|N|$, the edge density in G. (Rmk: The MLE problem for fitting $G(n, p)$ to the observed graph G is reduced to the MLE problem for fitting Binomial(N, p) to a sample with empirical frequency $|E|/N$. See Exc. A.9.2.) □

Example 3.2.9 (Fitting ER to CALTECH)**.** CALTECH is a connected network, which is part of the FACEBOOK100 data set [TMP12] (and which was studied previously as part of the FACEBOOK5 data set [RKMP11]), has 762 nodes and 16,651 edges. The nodes represent user accounts in the Facebook network of Caltech on one day in the fall of 2005, and the edges encode Facebook 'friendships' between these accounts.

In order to explain CALETCH by the Erdős-Renyí random graph model $G(n, p)$, we choose the MLEs $\hat{n} = 762$ and $\hat{p} =$the edge density in CALETCH$= 0.057$. Now, if we generate a graph from

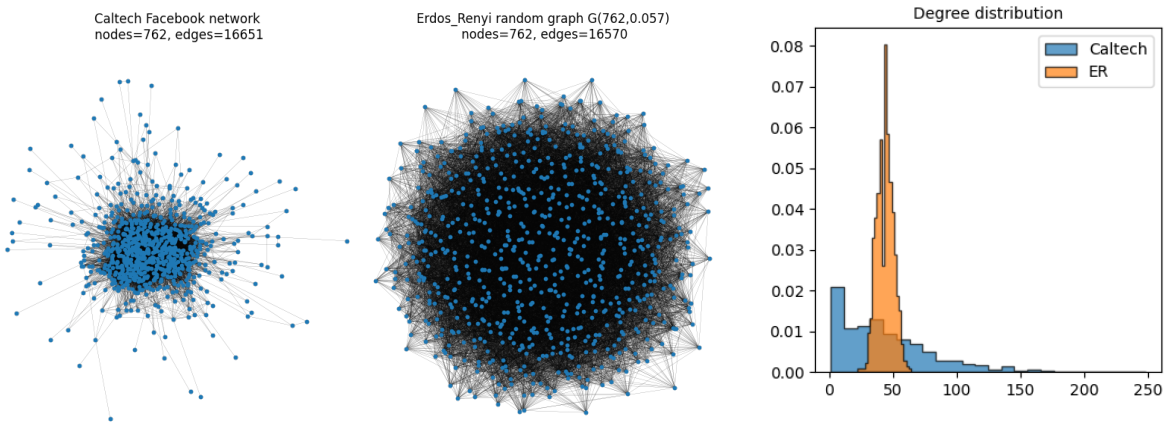


FIGURE 3.2.4. Fitting $G(n,p)$ to CALTECH Facebook networks.

$G(\hat{n},\hat{p})$, would we get something similar to CALTECH? Not quite. According to Exc. 3.2.5, we know that the degree distribution of $G(\hat{n},\hat{p})$ is a binomial distribution, which can be closely approximated by the normal distribution. However, as can be seen in Figure 3.2.4, the degree distribution of CALTECH is far from being binomial or normal. So in this case the Erdős-Renyí random graph is a not a good model to explain CALTECH. ▲

3.3. Small worlds

3.3.1. Clustering coefficient. In graph theory, a clustering coefficient gauges the extent to which nodes within a graph display a tendency to form clusters. Research indicates that within many real-world networks, especially in social networks, nodes tend to form closely-knit groups characterized by a relatively dense interconnection of relationships. This tendency is generally higher than the typical likelihood of a random connection between two nodes, as noted by Holland and Leinhardt in 1971 and further explored by Watts and Strogatz in 1998.

The 'local clustering coefficient' of a vertex in a graph measures the proximity of its neighbors to forming a clique, as introduced by Duncan J. Watts and Steven Strogatz in 1998. This metric was devised to assess if a graph exhibits characteristics of a small-world network.

Definition 3.3.1 (Clustering coefficients)**.** Let $G=(V,E)$ be a graph. For each node $v\in V$, the *local clustering coefficient* of v is defined by

$$C(v) := \frac{\#\text{triangles in } G \text{ containing } v}{\binom{\deg(v)}{2}}.$$

The *(average) clustering coefficient* of G is the mean of the local clustering coefficients:

$$C(G) := \frac{1}{|V|}\sum_{v\in V} C(v) = \mathbb{E}\left[C(U)\right],$$

where U is a uniformly chosen random node in G.

Exercise 3.3.2 (Clustering coefficient of Erdős-Renyí random graph)**.** Let $G=G(n,p)$ be the Erdős-Renyí random graph on n nodes with edge probability p.

(i) Fix a node i. Let N_i denote the number of neighbors of i in G excluding i itself. Show that $N_i \sim \text{Binomial}(n-1,p)$.

(ii) Let $T(i)$ denote the number of triangles in G that contains i. Show that $T(i)$ given $N(i)=d$ has distribution Binomial$(\binom{d}{2},p)$.

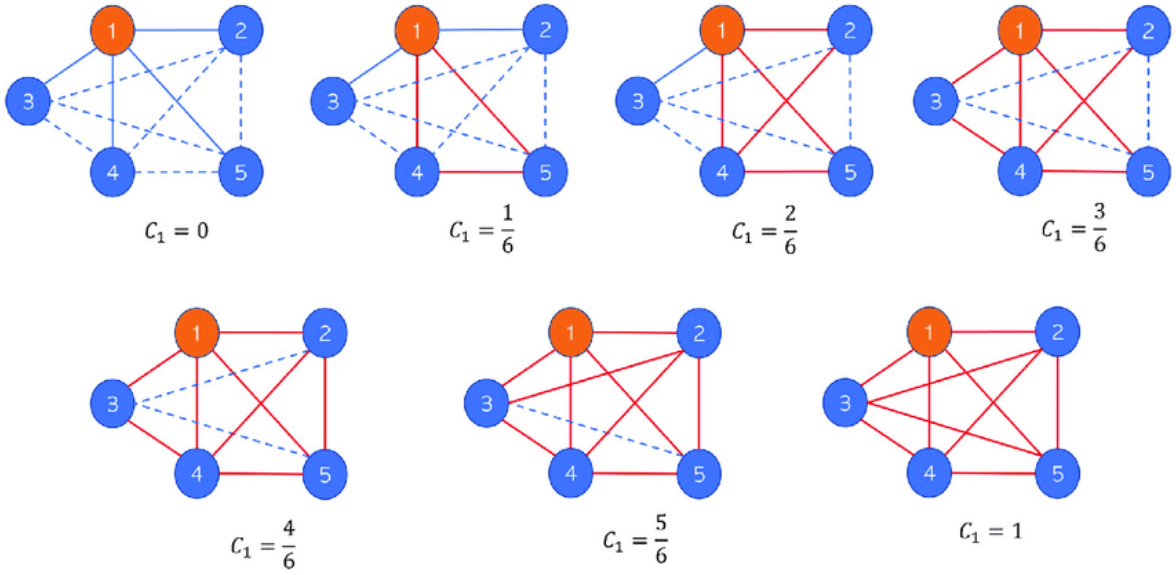


FIGURE 3.3.1. Examples of clustering coefficietns of nodes.

(iii) Show that, conditional on node i having $d \geq 2$ neighbors, the clustering coefficient $C(i)$ of i is distributed as

$$\binom{d}{2}^{-1} \text{Binomial}\left(\binom{d}{2}, p\right)$$

If $d \leq 1$, then we will define $C(i) = 0$.

(iv) Using iterated expectatation, justify the following:

$$\mathbb{E}[C(i) \,|\, N(i) \geq 2] = \mathbb{E}\left[\mathbb{E}\left[C(i)\,\middle|\, N(i)\right]\,\middle|\, N(i) \geq 2\right] = \mathbb{E}\left[p\right] = p.$$

Then deduce

$$\begin{aligned}\mathbb{E}[C(i)] &= \mathbb{E}[C(i) \,|\, N(i) \geq 2]\,\mathbb{P}(N(i) \geq 2) + \mathbb{E}[C(i) \,|\, N(i) \leq 1]\,P(N(i) \leq 1) \\ &= \mathbb{E}[C(i) \,|\, N(i) \geq 2]\,\mathbb{P}(N(i) \geq 2) \\ &= p\left(1 - (1-p)^{n-1} - (n-1)p(1-p)^{n-2}\right) \leq p.\end{aligned}$$

Hence the expected clustering coefficient of a given node i is approximately p given that $(1-p)^n = o(1)$ (This happens if $p \gg 1/n$.) If $(1-p)^n \geq c$ for some constant c (This happens if $p = O(1/n)$), then the last expression is smaller than p.

(v) Using iterated expectation, justify the following:

$$\mathbb{E}[C(G)] = \mathbb{E}_G\left[\mathbb{E}_U[C(U) \,|\, G]\right] = \mathbb{E}_U\left[\mathbb{E}_G[C(U) \,|\, U]\right] = \mathbb{E}_U[C(1)] = C(1).$$

Therefore, by **(iv)**, the expected average clustering coefficient of $G(n,p)$ is approximately p. In particular, if one tries to model sparse networks with ER $G(n,p)$, p is very small and consequently the graphs generated from such ER model will have very small average clustering coefficient.

3.3.2. Watts-Strogatz model. While ER graphs provide a simple and powerful tools for analyzing many properties of networks, they do not exhibit two important properties observed in many real-world networks:

(i) (*Large clustering coefficient*) They do not generate local clustering and triadic closures. Instead, because they have a constant, random, and independent probability of two nodes being connected, ER graphs have a low clustering coefficient.

(ii) (*Hubs*) They do not account for the formation of hubs. Formally, the degree distribution of sparse ER graphs $G(n, \lambda \log n/n)$ converges to a Poisson distribution, rather than a power law observed in many real-world, scale-free networks. See Figure 3.2.4.

The Watts and Strogatz model was designed as the simplest possible model that addresses the first of the two limitations. It accounts for clustering while retaining the short average path lengths of the ER model. It does so by interpolating between a randomized structure close to ER graphs and a regular ring lattice. Consequently, the model is able to at least partially explain the "small-world" phenomena in a variety of networks, such as the power grid, neural network of C. elegans, networks of movie actors, or fat-metabolism communication in budding yeast.[4]

The Watts–Strogatz model is a random graph model that yields graphs displaying small-world characteristics, such as short average path distances and large average clustering coefficients. Duncan J. Watts and Steven Strogatz introduced this model in a 1998 article published in the scientific journal Nature. Furthermore, the name "(Watts) beta model" arose when Watts employed the symbol (beta) to define the model in his widely read science book, Six Degrees.

Definition 3.3.3 (Watts-Strogatz model)**.** The *Watts-Strogatz model* is a random network model $\mathrm{WS}(G_0, p)$ that takes in two parameters: $G_0 = (V, E)$ the baseline graph and p the edge rewiring probability. In order to generate a random graph G from $\mathrm{WS}(G_0, p)$, we do the following 'edge rewiring' (see Fig. 3.3.2):

1. Give an arbitrary orientation on the edges of G_0 (e.g., by tossing a fair coin) and make a directed baseline graph $\vec{G}_0$.

2 For every original directed edge (u, v) in $\vec{G}_0$, with probability p, sample a node v' uniformly at random and replace (u, v) with (u, v'); do nothing with probability $1 - p$.

Common choices of the baseline graph G_0 are lattices or ring graphs. See Figures 3.3.3 and 3.3.4.

```python
def WS(G, p=0.1, random_orientation=False):
    # Watts-Strogatz model with baseline graph G and edge rewiring probability p
    # G is undirected. Flip fair coins for each edge of G to get initial orientation.
    # For each oriented edge, resample the head node uniformly at random with probability p, independently.
    # Do nothing for that edge with probability 1-p.

    # Give random orientation by crea
    if random_orientation:
        G1 = random_orientation(G)
    else: #G is already a digraph
        G1 = G

    nodes = list(G1.nodes())
    G2 = nx.Graph()

    for e in G1.edges():
        U = np.random.rand()
        if U < p:
            i = np.random.choice(np.arange(len(nodes)))
            v = nodes[i]
            G2.add_edge(e[0],v)
        else:
            G2.add_edge(e[0],e[1])
    return G2
```

FIGURE 3.3.2. A Python implementation of Watts-Strogatz model.

Exercise 3.3.4 (Watts-Strogatz model with rewiring probability 1)**.** Consider the Watts-Strogatz model $\mathrm{WS}(G_0, p)$ with a baseline graph $G_0 = (V, E_0)$ with n nodes and m edges. Each edge gets randomly oriented and rewired with probability p. If $p = 1$, the WS model is considered to be 'completely random'. One way to make sense of this statement is that every graph H with n nodes and m edges has a positive chance to be realized under $\mathrm{WS}(G_0, 1)$. In this exerise, we will see that this is not necessarily true.

(i) Fix a graph H with the same node set V as the baselind graph G_0 and edge set E with $|E| = m$. Suppose that there exists an edge $e = \{u, v\} \in E$ and every edge in G_0 is not incident with both u and v. Show that

$$\mathbb{P}\big(\text{Some edge in } G_0 \text{ is rewired and becomes } e\big) = 0.$$

Consequently, conclude that

$$\mathbb{P}\left(\mathrm{WS}(G_0, 1) = H\right) = 0.$$

(ii) (Optional*) The counterexample in part **(i)** necessarily assume G_0 to be disconnected. Why is this so? Now suppose G_0 and H are both connected. Then do we always have that

$$\mathbb{P}\left(\mathrm{WS}(G_0, 1) = H\right) > 0?$$

Watts-Strogatz model typically starts with a baseline graph that has large average clustering coefficient and average shortest-path length. Standard choices are the circulant graph (see the leftmost graph in Fig. 3.3.4) and triangular lattice graph (see Fig. 3.3.3 left). We then start rewiring the original edges with a rewiring probability p (see Fig. 3.3.3 right). Each rewired edge is likely to connect two nodes that were far away from each other in the baseline graph, so they act as "shortcuts", decreasing average shortest path distance drastically. Since many triangles that were in the baseline graph are still there after rewiring a small portion of the edges, we observe still large average clustering coefficient but small average shorted-path distance (see Fig. 3.3.4 right and Fig. 3.3.7). However, if we rewire almost all edges (p closed to one), then almost all edges are connecting randomly chosen pair of nodes so the resulting graph is like a sparse Erdős-Renyí random graph, with edge probability equals to the edge density in the baseline graph (see Fig. 3.3.5 and 3.3.6).

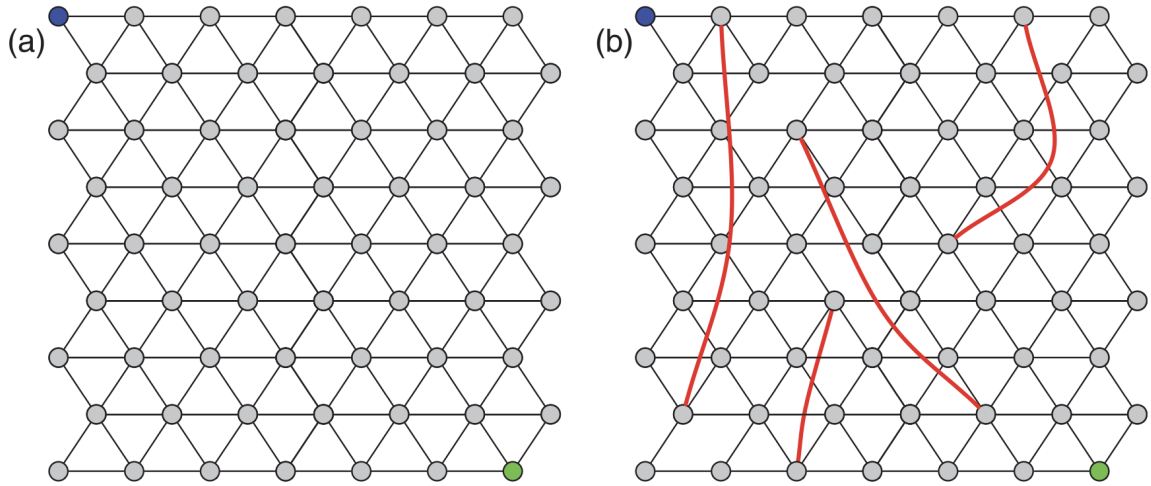


FIGURE 3.3.3. Construction of Watts-Strogatz model starting from triangular lattice. Figure exerpted from [MFD20].

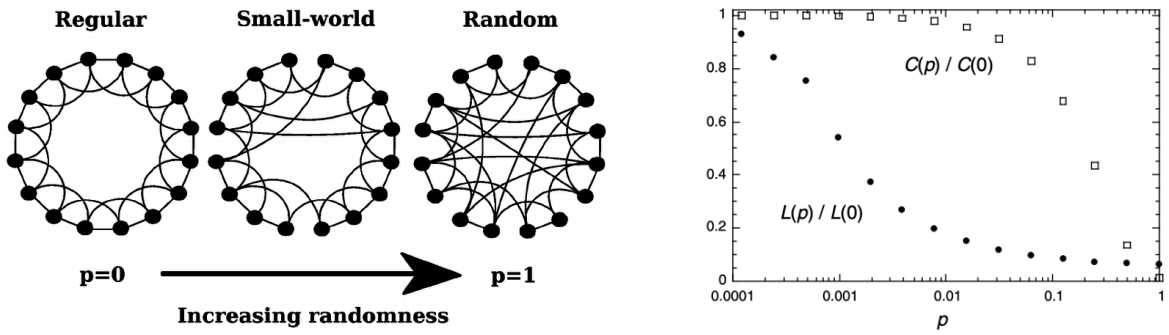


FIGURE 3.3.4. (left) Construction of Watts-Strogatz model starting from a ring with nearest two neighbors. (right) When p is somewhere between 0 an 1, there the random graph exibhits large clustering coefficient $C(p)$ and small average path length $L(p)$. Figure excerpted from [WS98].

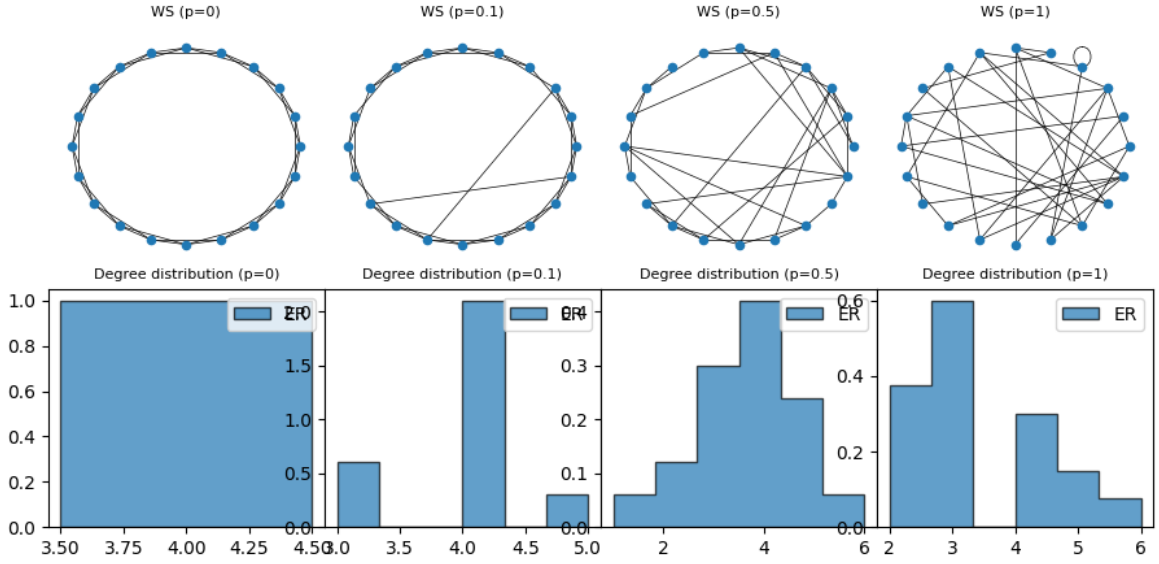


FIGURE 3.3.5. Watts-Strogatz networks with a circulant baseline graph and their degree distribution.

3.4. Preferential attachment

A *preferential attachment process* belongs to a category of processes where a certain resource, often wealth or credit, is allocated among a group of individuals or entities based on what they already possess. Consequently, those who are already affluent receive a larger share compared to those with fewer resources. "Preferential attachment" is just one of the various terms used to describe such processes. They are also known as the Yule process, cumulative advantage, the rich get richer, and the Matthew effect. They are closely connected to Gibrat's law. The primary reason for the scientific interest in preferential attachment is its ability, in appropriate conditions, to create power law distributions. However, if preferential attachment exhibits non-linear characteristics, the observed distributions may deviate from a power law. These mechanisms have the potential to produce distributions that closely resemble power laws for limited time intervals.

The *Barabási-Albert* (BA) model is a computational method designed to create random scale-free networks by employing a preferential attachment process. Many systems found in nature and created by humans, such as the Internet, the World Wide Web, citation networks, and certain social networks, are believed to exhibit a scale-free structure. In these systems, a few nodes, known as hubs, have significantly higher degrees than the majority of other nodes within the network. The

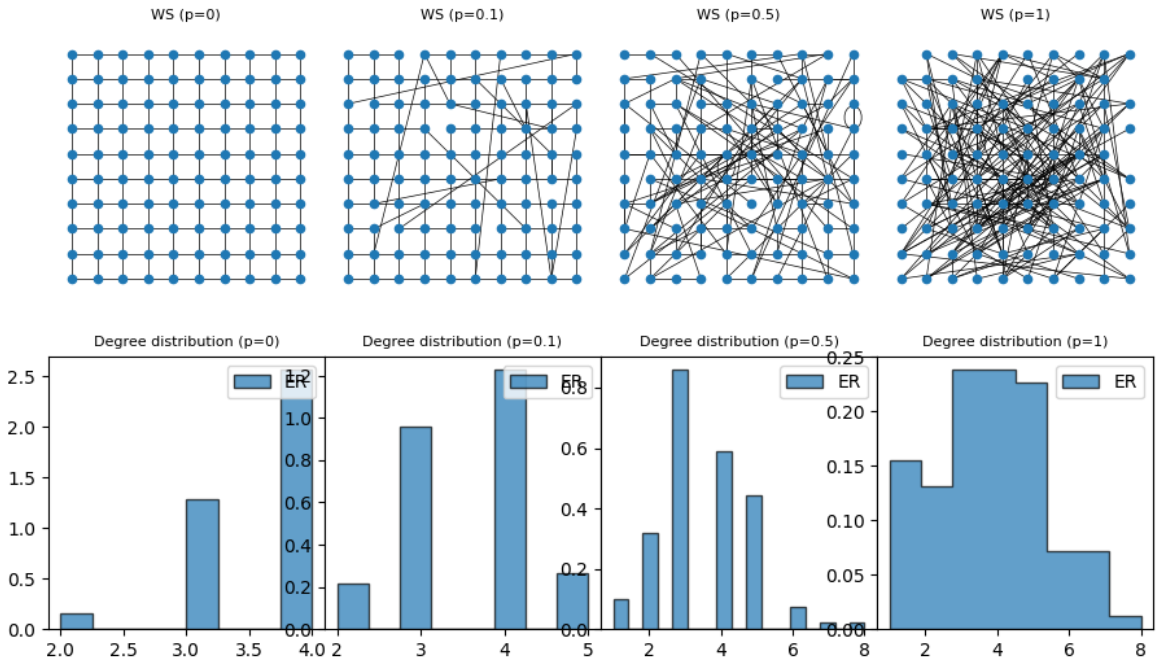


FIGURE 3.3.6. Watts-Strogatz networks with a lattice baseline graph and their degree distribution.

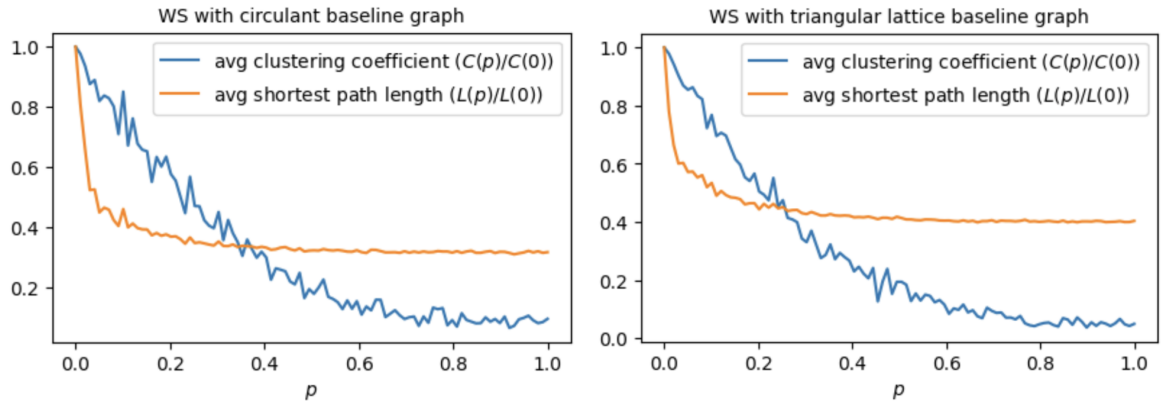


FIGURE 3.3.7. Watts-Strogatz networks with a circulant and a lattice baseline graph and their normalized average clustering coefficient and average shortest path length with respect to the rewiring probability p.

BA model seeks to provide an explanation for the presence of such highly connected nodes in real-world networks. This algorithm is named after its creators, Albert-László Barabási and Réka Albert.

Definition 3.4.1 (Barabási-Albert model)**.** The *Barabási-Albert* (BA) model is a dynamically evolving network model denoted as $\mathrm{BA}(G_0, m, n)$ that has three parameters: G_0 the baseline graph with $m_0 \leq m$ nodes, m the number of links for each new node, and n the total nodes to be achieved. $\mathrm{BA}(G_0, m, n)$ outputs a random network with n nodes by the following preferential attachement growth process:

$G \leftarrow G_0$
While $V(G) < n$:
  Add a new node $u = |V(G)| + 1$ to G
  For $i = 1, \ldots, m$:

1. Choose a node v in $G \setminus u$ at random, where the probability $p(v)$ that node v is chosen is proportional to its degree:

$$p(v) \propto \deg_G(v).$$

2. Add an edge $e = \{u, v\}$ to the graph and let $G \leftarrow G + e$.

End For

Typical choises of the baseline graph G_0 in BA model are (1) the graph with m_0 nodes and no edges and (2) the graph with a single node with m self-loops.

```python
def BA(G0=None, m0=1, m=1, n=100, alpha=1):
    # Barabasi-Albert model with baseline graph G = single node with m0 self-loops
    # Each new node has m edges pointing to some nodes in the existing graph
    # alpha=1 -> preferential attachment: The head of each new directed edge is chosen randomly with probability
    # proportional to the degree
    # alpha=0 ->: Uniform attachment: The head of each new directed edge is chosen uniformly at random
    # alpha \notin\{0,1} -> nonlinear preferential attachment: The head of each new directed edge is chosen
    # randomly with probability proportional to the degree^alpha

    if G0 is not None:
        G = G0
    else:
        G = nx.MultiGraph() # baseline graph with a single node and m0 self-loops
        for i in np.arange(m0):
            G.add_edge(1,1)

    for s in np.arange(1,n):
        for j in np.arange(m):
            # form a degree distribution
            degrees = np.asarray([G.degree(n)**(alpha) for n in G.nodes()])
            deg_dist = degrees*(1/np.sum(degrees))
            v = np.random.choice(G.nodes(), p=deg_dist)
            G.add_edge(s,v)

    return G
```

FIGURE 3.4.1. A Python implementation of Barabási-Albert model.

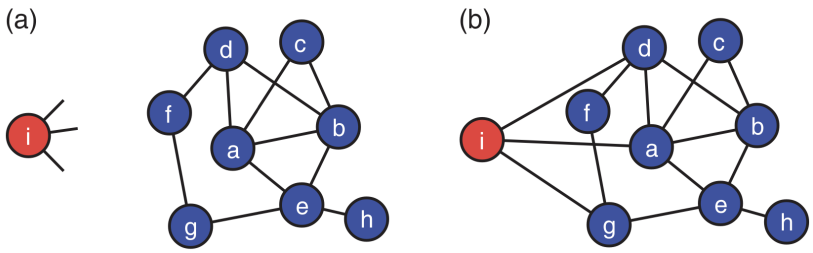


FIGURE 3.4.2. Illustration of a network growh. One can visualize the new node with m new edges as a 'stub' with m arms. Each arm is connected to one of the existing nodes according to some rule, e.g., preferential attachment. Figure exerpted from [MFD20].

The most important feature of the BA model is that it can explain how a power-law degree distribution can emerge in real-world networks (e.g., the Web) by the simple mechanism of preferential attachement and growth. This claim was originally made by

Theorem 3.4.2 (Thm. 8.3 in [VDH09])**.** *The exected degree distribution of the BA model* $\mathrm{BA}(G_0, m, n)$ *has a power-law distribution: For some constant* $\gamma > 0$, p_k *= fraction of nodes with degree* k*,*

$$\lim_{n\to\infty} \mathbb{E}[p_k] \sim k^{-\gamma}$$

for all k *less than any fixed constant.*

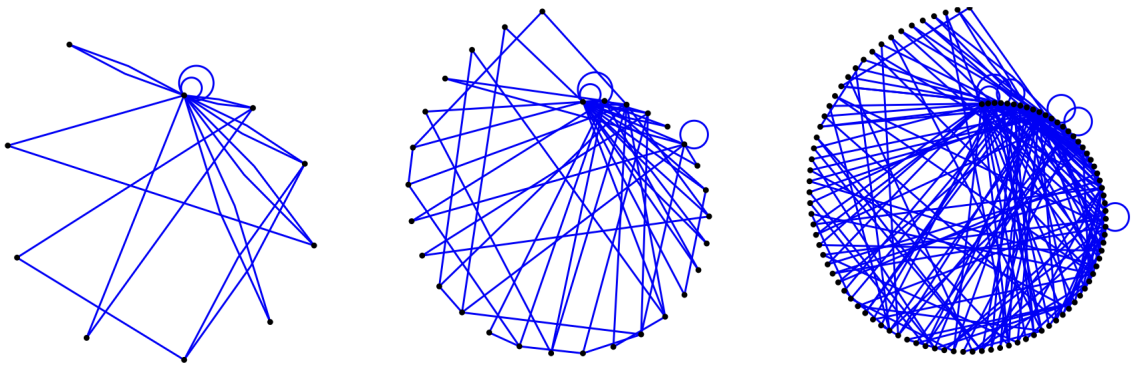


FIGURE 3.4.3. BA random graph with $m = 2$ of sizes 10, 30 and 100. Figure exerpted from [VDH09].

PROOF. The rigorous proof of this statement is out of the scope for this course. See [VDH09, Ch. 8] for more details. Below, we will provide a heuristic argument that "justifies" the power-law degree distribution with exponent $\gamma = 3$, for the case when $m = 1$. This argument was given originally by Barabási-Albert [BA99].

We will think of the BA model as a continuous-time process of nodes $i = 1, 2, \ldots, n$ arriving and being connected to some chosen nodes. Let $d_i(t)$ denote the degree of node i at time t, and let t_i denote the time that node i arrives. Even though $d_i(t)$ is not differentiable in t, we will differentiate anyways and for all $t > t_i$, write

$$\frac{d}{dt} d_i(t) = \frac{d_i(t)}{\sum_{k \le i} d_k(t)} = \frac{d_i(t)}{2t}.$$

The reasoning is as follows. When a new node is added at time $t > t_i$, then the probability that node i will be chosen to be connected to this new node will be proportional to $d_i(t)$, and the normalization constant is the sum of all degrees of existing nodes at time t. By time t, there should be t edges added so the total sum of degrees is $2t$. Integrating the above differential equation in t, we get

$$\log d_i(t) = \frac{1}{2}(\log t) + C.$$

Plugging in $t = t_i$ and using $d_i(t_i) = 1$, we find $C = -\frac{1}{2} \log t_i$, so

$$\log d_i(t) = \log \sqrt{\frac{t}{t_i}} \quad \text{or} \quad d_i(t) = \sqrt{\frac{t}{t_i}}.$$

Now we consider the expected degree distribution. Among all realization, the time that a particular node from 1 up to t will arive during $[0, t]$ will be uniformly distributed. Hence

$$\mathbb{P}(d_i(t) > k) = \mathbb{P}\left(\sqrt{\frac{t}{t_i}} > k\right) = \mathbb{P}\left(t_i < \frac{t}{k^2}\right) = \frac{1}{k^2}.$$

It follows that

$$\mathbb{E}[p_k] = \mathbb{P}(d_i(t) = k) = \mathbb{P}(d_i(t) > k) - \mathbb{P}(d_i(t) > k - 1) \approx k^{-3}.$$

(There are so many holes in this heuristic but it is still worth following the logic.) □

In order to see why preferential attachment is important, we consider the following generalization of Barabási-Albert model, where now the probability that an existing node is chosen to be connected with a new node will be proprtional to its degree to the power α, for somce constant $\alpha \ge 0$. When $\alpha = 1$, it becomes the usual Barabási-Albert model. When $\alpha = 0$, then the attachement is uniform (in this cases there is no power-law distribution emerging). We can also choose α

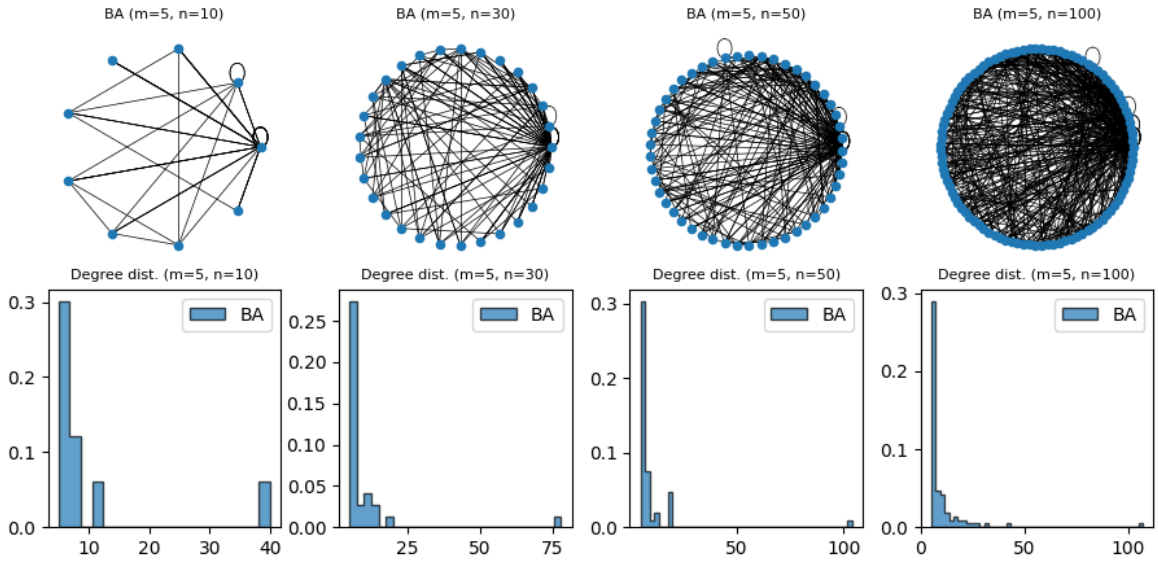


FIGURE 3.4.4. BA random graph with $m = 5$ of sizes 10, 30, 50, 100.

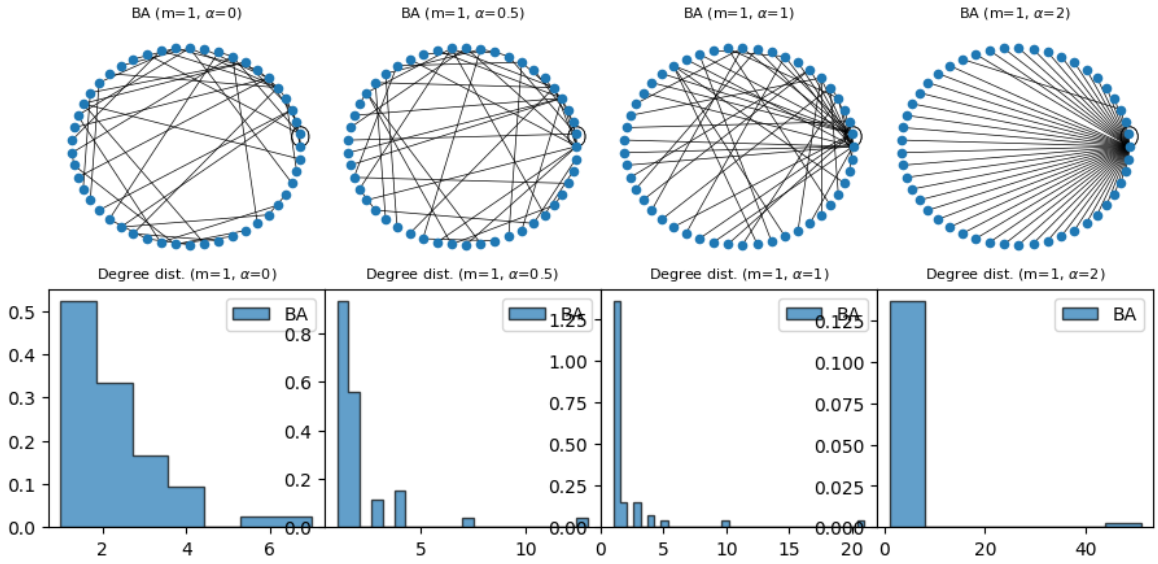


FIGURE 3.4.5. BA random graph with $m = 1$ of size 50 and attachement exponent $\alpha = 0, 0.5, 1, 2$.

not equal to 0 or 1. For instance, we can choose $\alpha > 1$ to strengthen preferential attachement. See the experiments in Figures 3.4.5 and 3.4.6. Note that a power-law appears as a linear function in the log-log plot:

$$y = x^{-\gamma} \quad \Longleftrightarrow \quad \log y = -\gamma \log x.$$

See how the degree distribution for $\alpha = 0$ in Figure 3.4.6 is a concave function, indicating that the degree distribution has an exponential decay. It appears as a linear function for $\alpha = 1$, indicating a power-law distirbution. For $\alpha = 1.2$, basically there is one node that takes all degree.

Definition 3.4.3 (Generalized Barabási-Albert model)**.** The *generalized Barabási-Albert* (GBA) model is a dynamically evolving network model denoted as BA(G_0, m, n, α) that has four parameters: The

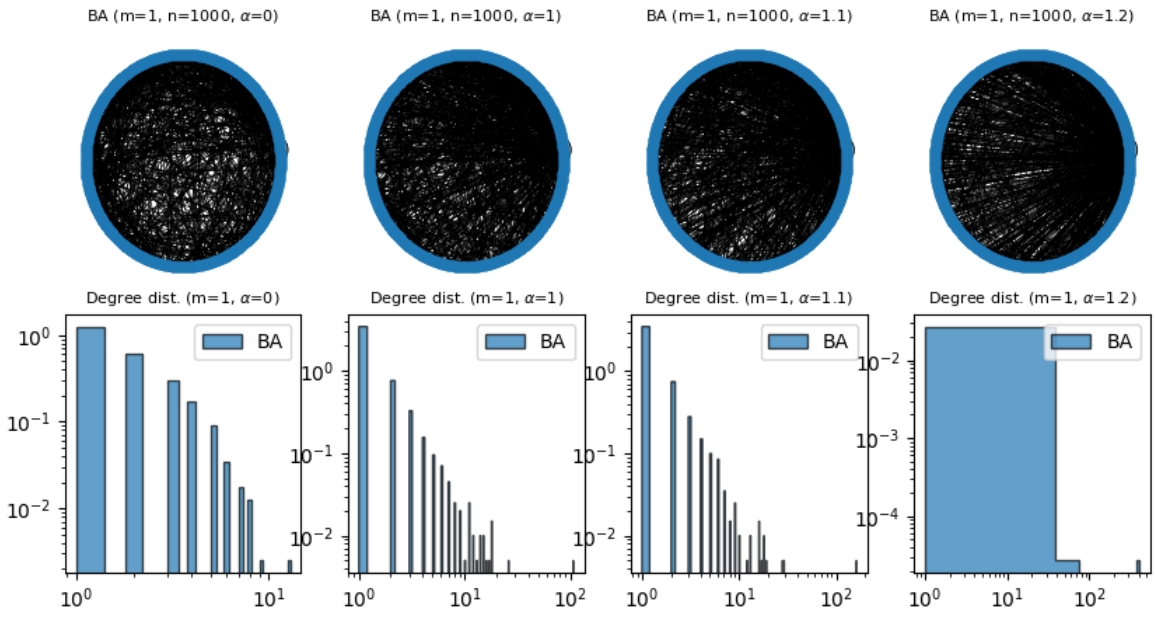


FIGURE 3.4.6. BA random graph with $m = 1$ of sizes 10, 30, 50, 100.

first three parameters are the same as in the BA model and α is the exponent in the degree distribution used to sample nodes in the existing network during attachement steps. BA(G_0, m, n, α) outputs a random network with n nodes by the following preferential attachement growth process:

$G \leftarrow G_0$
While $V(G) < n$:
  Add a new node $u = |V(G)| + 1$ to G
  For $i = 1, \ldots, m$:
    1. Choose a node v in $G \setminus u$ at random, where the probability $p(v)$ that node v is chosen is proportional to its degree:

$$p(v) \propto \deg_G(v)^\alpha.$$

    2. Add an edge $e = \{u, v\}$ to the graph and let $G \leftarrow G + e$.
  End For

Exercise 3.4.4. Generate a BA network with $n = 500$ nodes and $m = 3$. Using the python, carry out the following tasks:

(i) Plot the graph.
(i) Plot the degree distribution in log-log scale. Perform linear regression and estimate the exponent in the power-law decay in the degree distribution.
(iii) Compute the average degree and compare it with m. Interpret the result.
(iv) Calculate the average clustering coefficient.
(v) Verify that the generate graph is connected. Then compute the average path length.

3.5. Random walk attachment model

In Section 3.4, we have introduced the Barabási-Albert preferential attachment model that was based on two mechnisms: (1) Growth and (2) Preferential attachement. This model were able to explain the spontaneous emergence of power-law degree distribution and hub nodes often observed in many real-world networks that grows incrementally (e.g., the internet, the Web, social networks, protein-protein interaction networks). However, there are two drawbacks of this model:

Weakness 1. Preferential attachment mechanism is unrealistic in that each new node must have the knowledge of the degree distribution of the existing network, while in many real world networks each node only has a local view of the network.

Weakness 2. Barabási-Albert model tends to have a small clustering coefficient due to its inherent structure and growth mechanism.

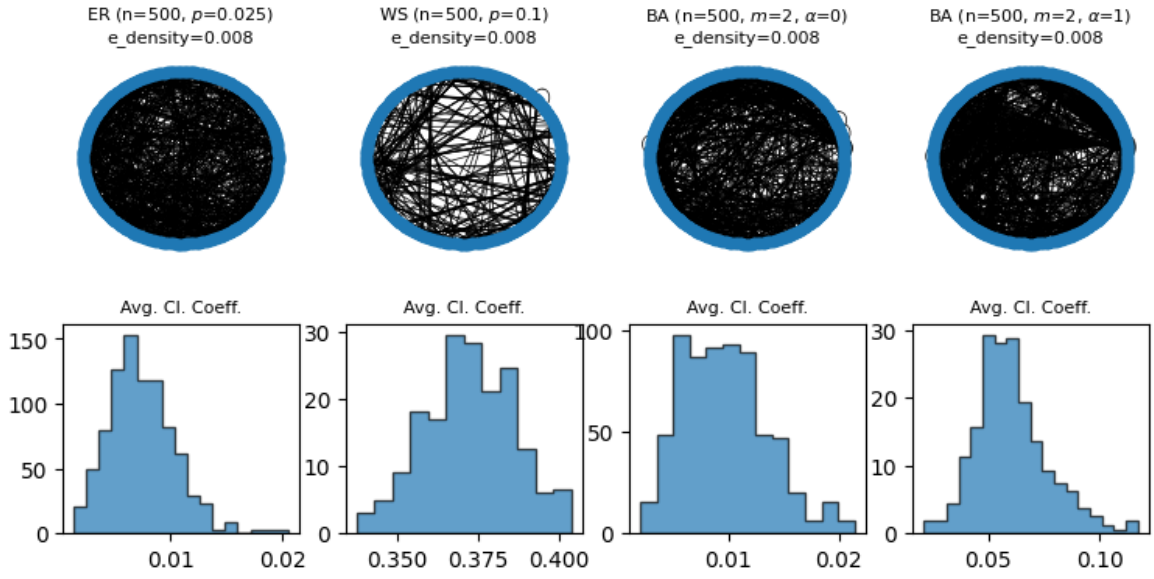


FIGURE 3.5.1. Comparison of the histogram of average clustering coefficients of 100 realizations of 500-node random graphs from ER, WS, and BA networks of the same edge density. Except the WS network, the clustering coefficients of other networks are vanishingly small.

Let's discuss a bit more on the second weakenss of the BA model above. Recall that the clustering coefficient measures the degree to which nodes in a network tend to cluster together. In a network with preferential attachment, the connections are more likely to be focused on a few highly connected hubs, which reduces the tendency for nodes to form tightly connected local clusters. Instead, most nodes in the network have a relatively small number of connections, and the network exhibits a "hub-and-spoke" structure. This is in contrast to other network models, like the small-world model or random networks, where the clustering coefficient tends to be higher because nodes are more likely to form local connections with their neighbors. The preferential attachment model's emphasis on connecting to highly connected nodes at the expense of local connections results in a smaller clustering coefficient.

Bianconi et al. [BDIF14] proposed a variant of BA network that address both issues mentioned above. The key idea is the following. After we add the first edge that connects the new node u with a uniformly chosen node v_1, we choose the next neighbor v_2 of the new node to be added by jumping into another neighbor of the previous node v_1 with probability p (in this case we close the triangle $\{u, v_1, v_2\}$), and v_2 is chosen yet again uniformly at random from all nodes with probability $1-p$. When $p = 1$, then v_i is always a uniformly chosen neighbor of v_{i-1}. Such sequence $(v_1, v_2, \ldots)$ is called a simple symmetric random walk on G, hence the name 'random walk attachement'. See Figure 3.5.2 for illustration. For large p, we have high chance of p of closing a triangle, so we can expect the graph generated from this model will have high average clustering coefficient. Moreover, we do not use th global degree distribution and only use local information[2].

[2]Choosing a node uniformly at random can be done by multiple steps of random walks, which is again requires only local moves. We will talk about such a Markov Chain Monte Carlo sampling later.

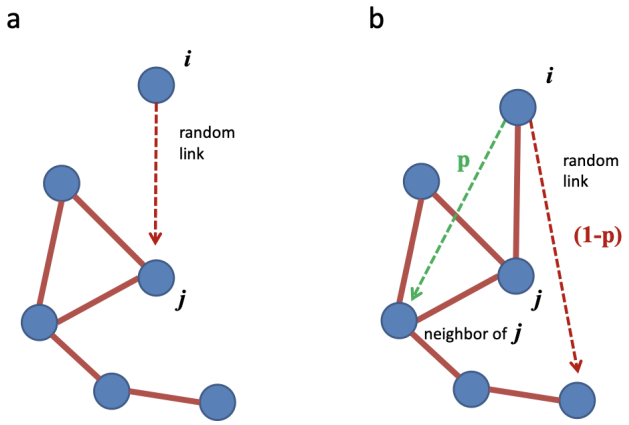


FIGURE 3.5.2. One link associated to a new node i is attached to a randomly chosen node j, the other links are attached to neighbors of j with probability p, closing triangles, or to other randomly chosen nodes with probability $1p$. Figure exerpted from [BDIF14].

Definition 3.5.1 (Random walk attachment model)**.** The *random walk attachment* (RWA) model is a dynamically evolving network model denoted as RWA(G_0, m, n, p) that has four parameters: G_0 the baseline graph with $m_0 \leq m$ nodes, m the number of links for each new node, and n the total nodes to be achieved, and p is the probability of 'triangle closure'. RWA(G_0, m, n, p) outputs a random network with n nodes by the following random walk attachement growth process:

$G \leftarrow G_0$
While $V(G) < n$:
  Add a new node $u = |V(G)| + 1$ to G
  For $i = 1, \ldots, m$:
    1. If $i = 1$, then choose v_1 uniformly at random from the existing graph and let $G \leftarrow G + \{u, v_1\}$.
    2. If $i \geq 2$, then using an independent coin flip, choose

$$v_i \sim \begin{cases} \text{Uniform}(N(v_{i-1})) & \text{with prob. } p \\ \text{Uniform}(V(G)) & \text{with prob. } 1-p \end{cases}.$$

    Then let $G \leftarrow G + \{u, v_i\}$.
  End For

See Figure 3.5.4 for examples. Note that RWA networks (with $p = 1$) has more triangles and also has higher average clustering coefficients than BA networks (with $\alpha = 1$) of the same parameters n and m.

Next, we will experimentally verify that RWA networks with large triangle closure probability p has higher clustering coefficients than BA networks and also maintains a power-law degree distribution. See Figure 3.5.5.

Exercise 3.5.2. Generate a RWA network with $n = 500$ nodes, $m = 3$, and $p = 0.9$. Using the python, carry out the following tasks:

(i) Plot the graph.
(ii) Plot the degree distribution in log-log scale. Does it have a power-law degree distribution? Verify this by performing linear regression and reporting the R^2 value.
(iii) Compute the average degree and compare it with m. Interpret the result.
(iv) Calculate the average clustering coefficient.
(v) Verify that the generate graph is connected. Then compute the average path length.

```python
def RWA(G0=None, m0=1, m=1, n=100, p=0.9):
    # Random Walk Attachement model with baseline graph G = single node with m0 self-loops
    # Each new node has m edges pointing to some nodes in the existing graph
    # When adding a new node $u$, we add edges {u,v_1}, \dots, {u,v_m}
    # v_1 is uniformly chosen among all nodes in G
    # v_2 is uniformly chosen among all neighbors of v_1 with probability p;
    # (p is the probability of closing a triangle)
    # with the rest of probability, v_2 is chosen the same way as v_1
    # Do the same for the rest of v_3,..,v_m.

    if G0 is not None:
        G = G0
    else:
        G = nx.MultiGraph() # baseline graph with a single node and m0 self-loops
        for i in np.arange(m0):
            G.add_edge(1,1)

    for s in np.arange(1,n-m0):
        v = np.random.choice(G.nodes())
        for j in np.arange(m):
            U = np.random.rand()
            if (j == 1) or (U>p):
                candidates = list(G.nodes())
            else:
                candidates = list(G.neighbors(v))
            if (len(candidates)>1) and (s in candidates):
                candidates.remove(s)
            v = np.random.choice(candidates)
            G.add_edge(s,v)

    return G
```

FIGURE 3.5.3. A Python implementation of random walk attachment model.

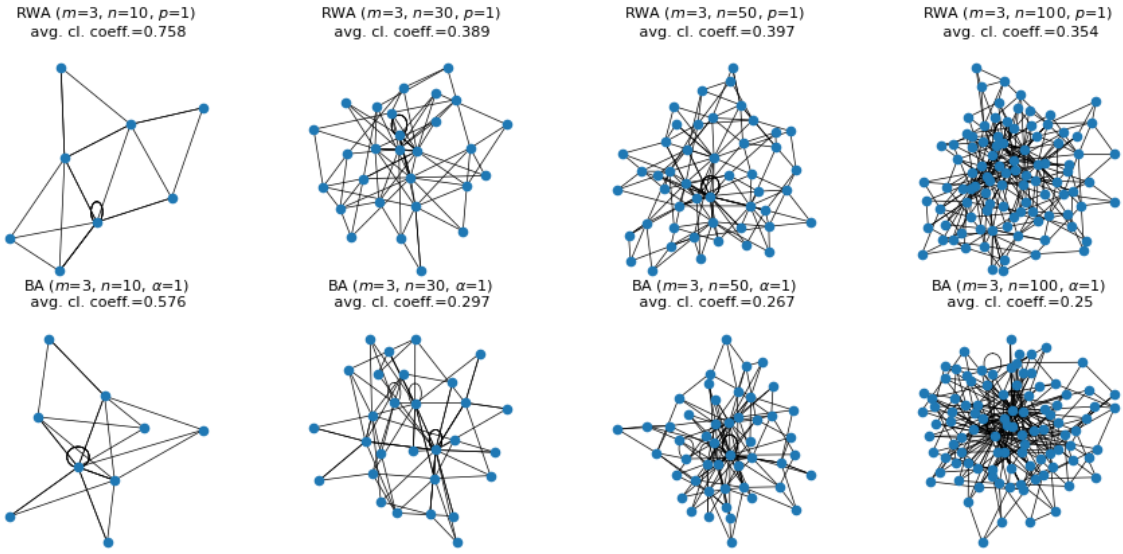


FIGURE 3.5.4. Comparison of the BA and RWA networks for various parameter choices. Note that RWA has more triangles and also has higher average clustering coefficients than BA networks.

3.6. Configuration model

Configuration model is a mathematical model used in network theory and graph theory to create random graphs that have a specified degree sequence. In other words, it allows one to generate random networks where each node has a predetermined number of edges. Unlike in all of the previous random network models (ER, WS, BA, and RWA), the configuration model provides the flexibility for the user to specify any desired degree distribution for the network. It helps explore the structure of networks with specific degree sequences, which can be crucial for understanding the

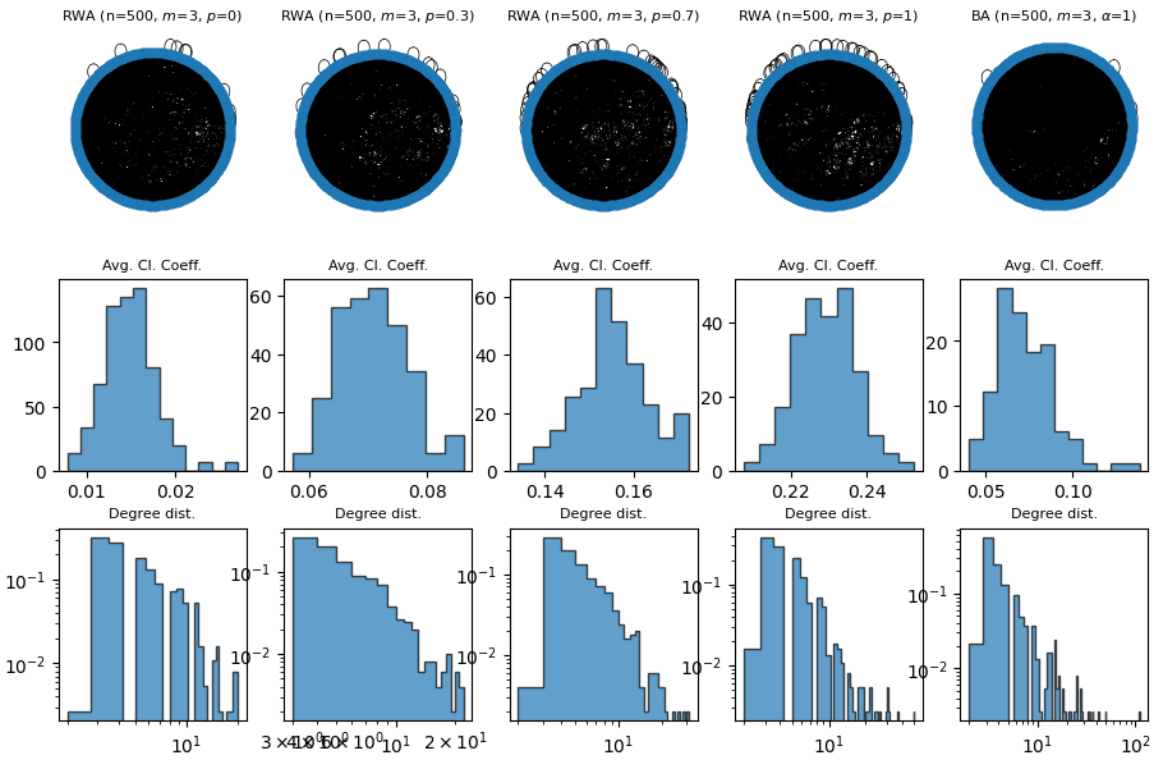


FIGURE 3.5.5. Comparison of the histogram of average clustering coefficients of 100 realizations of 500-node random graphs from ER, WS, and BA networks of the same edge density. Except the WS network, the clustering coefficients of other networks are vanishingly small.

evolution and behavior of various complex systems, such as social networks, biological networks, and the World Wide Web.

3.6.1. Definition of the model and applications. The problem of sampling a *simple* graph with a given degree sequence uniformly at random is not quite straightforward. For one reason, there may not exist any simple graph with the given degree sequence. In Theorem 2.1.17, we studied a necessary and sufficient condition for a sequence of nonnegative integers to be 'graphic', meaning that it is the degree sequence of some simple graph. Even if we have a graphic sequence, it is till not quite easy to sample one simple graph that satisfies the degree sequence uniformly at random. Therefore, we relax the requirement and allow multigraphs (e.g., with self-loops and multi-edges) given the prescribed degree sequence. The following random multgraph model, called the configuration model, is able to generate a random multigraph with given degree sequence.

Definition 3.6.1 (Configuration model)**.** Let $\mathbf{d} = (d_1, \dots, d_n)$ denote a degree sequence of an n-node graph possibly with self-loops and multi-edges. (Necessarily $d_i \in \mathbb{N}$ and $\sum_i d_i$ =even.) The *configuration model* CM($\mathbf{d}$) is a random multigraph model with n nodes and degree sequence $\mathbf{d}$, which is generated as follows (see Fig. 3.6.3 for a python implementation):

(i) Suppose we have a degree sequence $\mathbf{d} = (d_1, \dots, d_n)$. For each node, create "stubs" (nodes + half-edges) at each node. Stubs are essentially placeholders for edges.

(ii) Choose two remaining half-edges uniformly at random. Pair them into an edge. Repeat this process until all half-edges are exhausted.

Example 3.6.2 (Hypothesis testing using configuration model)**.** Configuration model also serves as the null-model for testing whether certain property of a network is purely determined by its

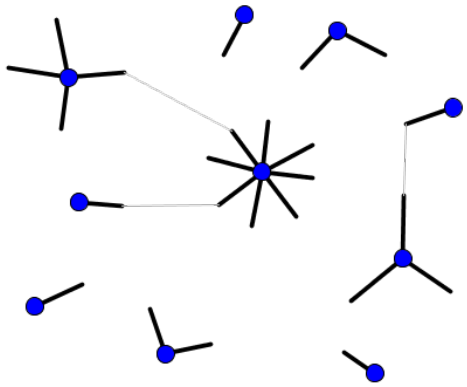


FIGURE 3.6.1. Illustration of the algorithm for sampling from configuration model. Nodes (blue) and half-edges (black) form stubs. Two half-edges, chosen uniformly at random, are connected and become an edge.

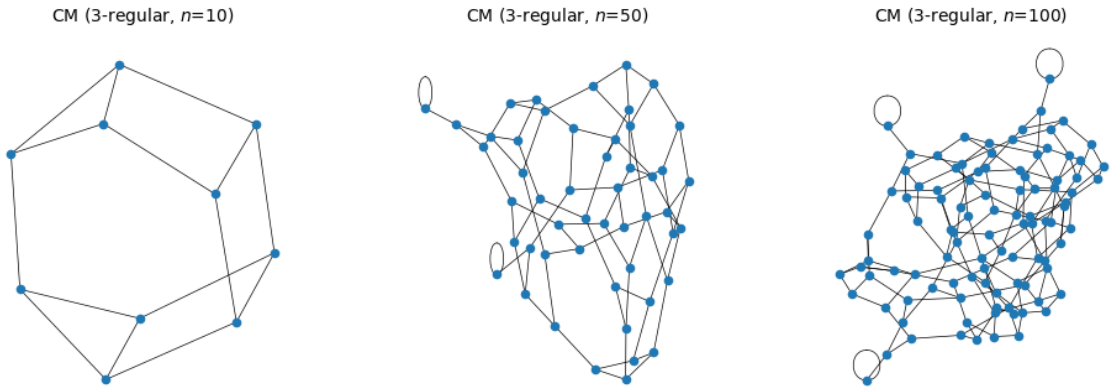


FIGURE 3.6.2. Examples of graphs generated from the configuration model with 3-regular graphs with $n \in \{10, 50, 100\}$ nodes.

degree sequence or not. That is, by comparing a real network to a set of configuration model networks with the same degree sequence, one can determine whether observed network properties are statistically significant or merely a consequence of the degree distribution. For this purpose, we may want to define the configuration model in a way that it is 'as random as possible' while satisfing the prescribed degree sequence. We will discuss this aspect in the following subsection. [3]

To illustrate how hypothesis testing is done, first take a subgraph H (87 nodes, 653 edges) of CALTECH, which was sampled by taking 100 steps of random walk on CALTECH and taking the induced subgraph on the sampled nodes. The null hypothesis is that this *subgraph H is generated from the configuration model* CM($\mathbf{d}$)*, where* $\mathbf{d}$ *is the degree sequence of* H. (See Fig. 3.6.4.) We test this hypothesis using four statistics: average degree, average clustering coefficient, average shortest path length, and the size of maximum matching. We generate 1000 graphs from the corresponding configuration model and compute the histogram of the corresponding statistics. The p-value of a test statistic with value x_0 is defined as

$$\mathbb{P}\Big([\text{Value of the statistic for a randomly generated graph } G \sim \text{CM}(\mathbf{d})] \geq x_0\Big).$$

[3] The configuration model is in fact not uniformly random among all multigraphs with given degree sequence, but it is uniformly random if it is conditioned to be simple.

```python
def CM(d):
    # Configuration model with degree sequence d = [d1, ... , dn] (a list or array)
    # di \ge 0 and sum to even
    d = list(d)
    stubs_list = []
    for i in np.arange(len(d)):
        for j in np.arange(d[i]):
            stubs_list.append([i,j])

    G = nx.MultiGraph()
    while len(stubs_list)>0:
        ss = np.random.choice(np.asarray(len(stubs_list)), 2, replace=False)
        s1, s2 = ss
        half_edge1 = stubs_list[s1]
        half_edge2 = stubs_list[s2]
        G.add_edge(half_edge1[0], half_edge2[0])
        stubs_list.remove(half_edge1)
        stubs_list.remove(half_edge2)

    return G
```

FIGURE 3.6.3. A Python implementation of the configuration model.

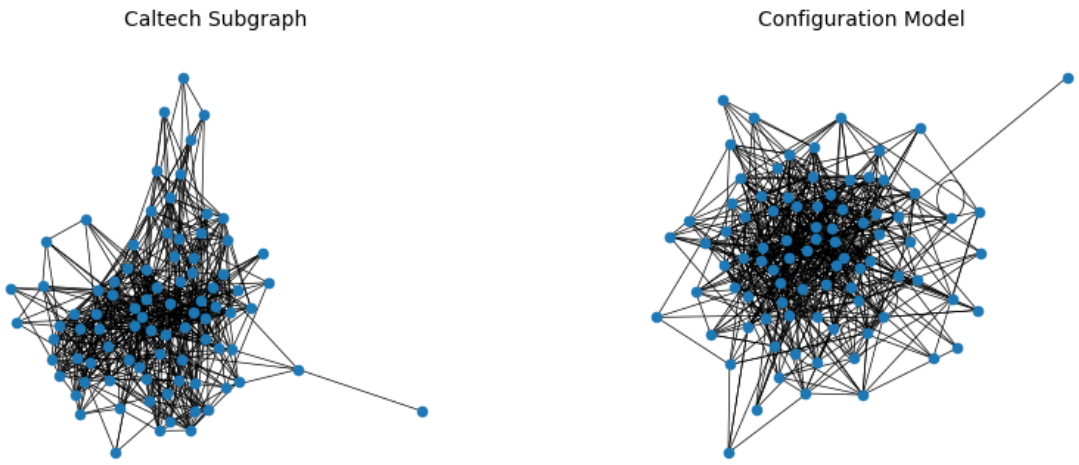


FIGURE 3.6.4. A subgraph H sampled from CALTECH and a graph generated from $CM(\mathbf{d})$, where $\mathbf{d}$ is the degree sequence of H.

This can be easily computed by the proportion of sampled graphs under the null model that has larger values of the statistic of interest than x_0. The small the p-value is, the less likely that the test graph H is generated from the null model.

The results are shown in the first row of Figure 3.6.5. Clearly, the average degree is completely determined by the degree sequence. So the corresponding p-value will always be 1, so this statistic is inconclusive. Note that the average clustering coefficient and average shortest path lengths have very small p-values against the null model. Therefore, it is unlikely that H is generated from the null model (which is the CM model here). Interestingly, the size of maximum matching shows large p-value. However, this does not imply that H is generated by the CM model, since the previous two statistics had very small p-values.

Lastly, we do the similar experiments for a graph generated from the configuration model. In this case, one can expect that whatever test statistic one uses, it will have a large p-value. This is indeed the case. See the second row of Figure 3.6.5. ▲

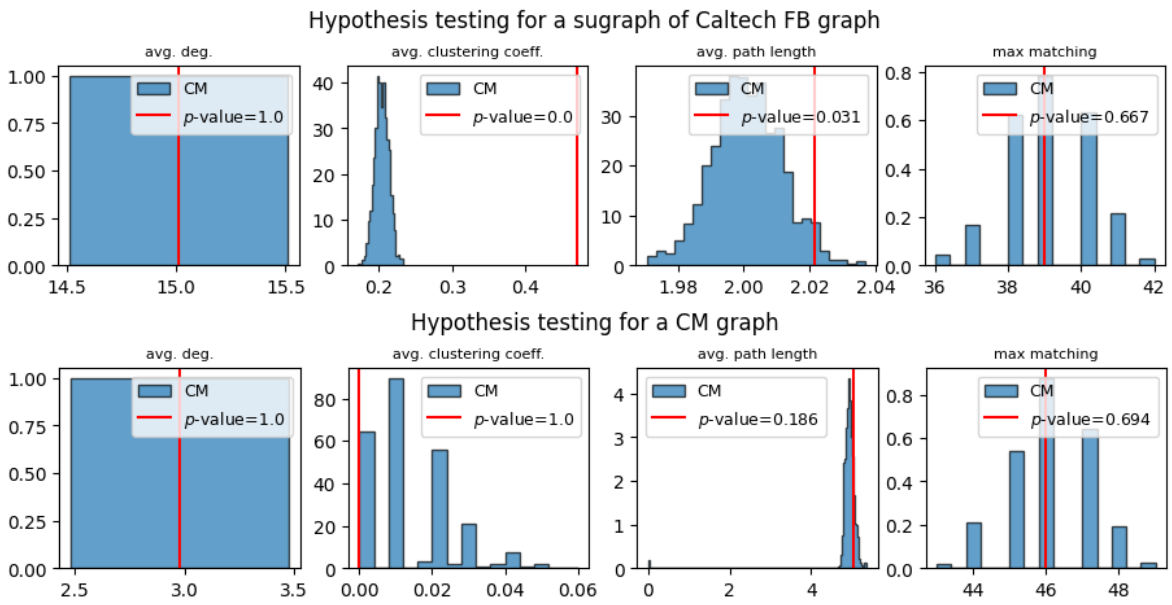


FIGURE 3.6.5. (Top) Hypothesis testing for subgraph H (87 nodes, 653 edges) of CALTECH. The null hypothesis is that this subgraph H is generated from the configuration model $CM(\mathbf{d})$, where $\mathbf{d}$ is the degree sequence of H. We test this hypothesis using four statistics. (Bottom) Hypothesis testing for a 100-node 3-regular graph generated from the configuration model.

Exercise 3.6.3. Perform a similar hypothesis testing as in Example 3.6.2 under the following null hypotheses:

(i) The subgraph H of CALTECH is generated from the Erdős-Renyí model. (*Hint:* First find the MLEs for the parameters of the model, and then sample 1000 graphs from that model and produce similar plots in Figure 3.6.5.)

(ii) The subgraph H of CALTECH is generated from Barabási-Albert model. (*Hint:* We have not discussed MLE for the BA model. Use the default baseline graph G_0. Choose the parameter m (new edges per each new node) so that $m(n-1)$ approximately equals the number of edges in H, where $n = |V(H)|$. Then then sample 1000 graphs from that model and produce similar plots in Figure 3.6.5.)

For both null hypotheses, state your conclusion and explain your reasoning. (You may use the code in the Jupyter notebook.)

3.6.2. Theoretical analysis on the configuration model. In this subsection, we provide some theoretical analysis on the configuration model. The main result in this section is Theorem 3.6.9.

To explain the term "configuration model," we now present an equivalent way of defining the configuration model in terms of *uniform matchings*.

Definition 3.6.4 (Uniform pairing graph)**.** Suppose we have a degree sequence $\mathbf{d} = (d_1, \dots, d_n)$. Let $\ell_n := \sum_{i=1}^n d_i$. Construct a graph $G = (V, E)$ where the nodes in $V = \{1, \dots, \ell_n\}$ correspond to the half-edges of the random multigraph in the configuration model. For the edges in G, first pair two uniformly selected nodes and add an edge there; then among the unpaired nodes, pair two uniformly selected nodes and give an edge, an so on. This process continues until all vertices are paired to another unique vertex. For each $i \in V$, let $\sigma(i)$ denote the node that i is paired with. Such a map σ is called a 'configuration' (pairing), which we may identify with the graph G. This generates a simple graph on ℓ_n nodes and $\ell_n/2$ edges. We will denote the corresponding random graph model as Conf($\mathbf{d}$).

Exercise 3.6.5. Show that the number of pairings of $2m$ nodes are

$$(2m-1)(2m-3)\cdots 3\cdot 1 =: (2m-1)!!.$$

Exercise 3.6.6 (Uniform pairing)**.** Define a random pairing σ of $2m$ nodes sequentially as follows. First choose a uniformly random pair of nodes from $\{1,\ldots.2m\}$ and pair them. Then among the remaining $2m-2$ nodes, choose a uniformly random pair of nodes pair them. Repeat this process until all nodes are paired. In this exercise, we will show that σ is uniformly distributed among all pairings of $2m$ nodes.

(i) Let $U_1,\ldots,U_{2m}$ be i.i.d. Uniform$(0,1)$ random variables. Let $U^{(1)} < U^{(2)} < \cdots < U^{(2m)}$ denote their order statistics, that is, $U^{(j)}$ is the jth largest value among $U_1,\ldots,U_{2m}$. Let π be the unique (random) permutation on $\{1,\ldots,2m\}$ such that $U^{(i)} = U_{\pi(i)}$ for all $i = 1,\ldots,2m$.

Define a paring σ' on $2m$ nodes by $\{\pi(1),\pi(2)\},\{\pi(3),\pi(4)\},\ldots,\{\pi(2m-1),\pi(2m)\}$. Show that σ' is uniformly distributed over all parings of $2m$ nodes.

(*Hint*: Since $U_1,\ldots,U_{2m}$ are i.i.d., the random paring σ'' obtained by first permuting the values of these $2m$ RVs and then following the same construction will have the same distribution as σ'. Hence σ' must be uniformly distributed.)

(ii) Show that the 'first pair' $\{\pi(1),\pi(2)\}$ in σ' is uniformly distributed among all pairs of $2m$ nodes. Show that for each pair $\{i,j\}$ disjoint from $\{\pi(1),\pi(2)\}$,

$$\mathbb{P}\Big(\{\pi(3),\pi(4)\} = \{i,j\} \,\Big|\, \{\pi(1),\pi(2)\}\Big) = \text{Const.}$$

Similarly, for each $k \le m$ and for each $\{i,j\}$ disjoint from $\{\pi(1),\ldots,\pi(2k-2)\}$,

$$\mathbb{P}\Big(\{\pi(2k-1),\pi(2k)\} = \{i,j\} \,\Big|\, \{\pi(1),\pi(2)\},\ldots,\{\pi(2k-3),\pi(2k)\}\Big) = \text{Const.}$$

(*Hint*: Conditional on the first two order statistics $U_{\pi(1)}, U_{\pi(2)}$, the remaining RVs U_i for $i \notin \{\pi(1),\pi(2)\}$ are i.i.d., so the next two order statistics $U_{\pi(3)}, U_{\pi(4)}$ form any pair between the remaining nodes uniformly at random. See Exercise A.6.7.) Conclude that σ' and σ have the same distribution.

(iii) Conclude that σ is uniformly distributed among all pairings of $2m$ nodes. That is, for each pairing τ on $2m$ nodes,

$$\mathbb{P}(\sigma = \tau) = \frac{1}{(2m-1)!!}.$$

(Hint: Use Exercise 3.6.5.)

Proposition 3.6.7 (Distribution of uniform paring graph)**.** *Suppose we have a degree sequence* $\mathbf{d} = (d_1,\ldots,d_n)$*. Let* $\ell_n := \sum_{i=1}^n d_i$*. Then* Conf$(\mathbf{d})$ *is uniformly distributed over all pairings of nodes* $1,\ldots,\ell_n$*. That is, for each pairing* τ *on* $2m$ *nodes,*

$$\mathbb{P}(\text{Conf}(\mathbf{d}) = \tau) = \frac{1}{(\ell_n - 1)!!}.$$

PROOF. Recall that a graph $G \sim$ Conf$(\mathbf{d})$ is identified with a paring σ on the nodes $1,\ldots,\ell_n$ that gives the edge set: $\{i,j\} \in E(G)$ if and only if $j = \sigma(i)$. By Exercise 3.6.6, we know that the random paring σ that describes the edge set of the random graph Conf$(\mathbf{d})$ is uniformly distributed among all pairings of ℓ_n nodes. By Exercise 3.6.5, we know that there are $(\ell_n - 1)!!$ such pairings. Hence the assertion follows. □

Proposition 3.6.8 (Uniform matching graph to configuration model)**.** *Suppose we have a degree sequence* $\mathbf{d} = (d_1,\ldots,d_n)$*. Let* $\ell_n := \sum_{i=1}^n d_i$*. Let* $G \sim$ Conf$(\mathbf{d})$*. Identify vertices* $1,\ldots,d_1$ *in* G *to form vertex 1, vertices* $d_1+1,\ldots,d_1+d_2$ *in* G *to form vertex 2, and so on. Let* G' *denote the resulting multigraph on* n *nodes. Then* G' *has degree sequence* $\mathbf{d}$ *and is distributed according to* CM$(\mathbf{d})$.

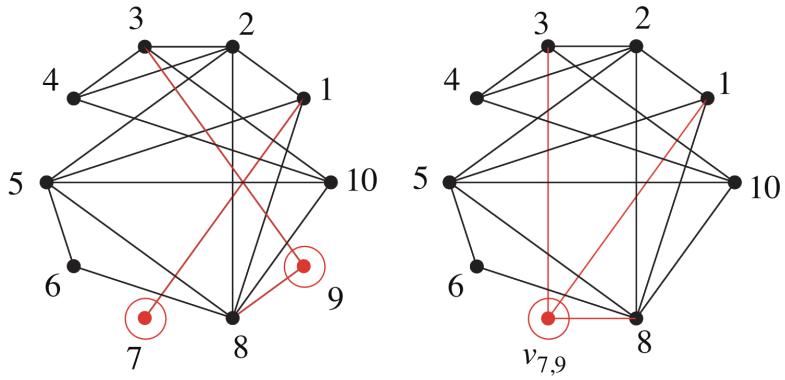


FIGURE 3.6.6. Example of vertex identification. Identifying nodes 7 and 9 gives a new node $v_{7,9}$, which is adjacent to all neighbors of nodes 7 and 9.

PROOF. Let $1,\dots,\ell_n$ denote the nodes in Conf($\mathbf{d}$). Rename the nodes as

$$[1,1],\dots,[1,d_1],[2,1],\dots,[2,d_2],\dots,[n,1],\dots,[n,d_n].$$

That is, $[i,j]$ can be thought of as the jth half-edge at node i in the configuration model. When generating CM($\mathbf{d}$), these half-edges were paired sequentially by choosing two unpaired half-edges uniformly at random. Let σ denote the resulting random pairing of the half-edges. According to the correspondence, σ can also be regarded as the pairing of ℓ_n nodes used to generate Conf($\mathbf{d}$). Then the node identification (e.g., nodes $1,\dots,d_1$ with 1, nodes $d_1+1,\dots,d_2$ with 2, etc.) is exactly identifying the union of the half-edges $[1,1],\dots,[1,d_1]$ with the stud at node 1 with d_1 half-edges, and so on. Hence generating a uniform matching graph from Conf($\mathbf{d}$) and identifying the nodes as described gives a random multigraph that is distributed according to CM($\mathbf{d}$). □

We note that not all multigraphs with a given degree sequence have the same probability under the configuration model. That is, not every multigraph is equally likely and the measure obtained is not the uniform measure on all multigraphs with the prescribed degree sequence (see, e.g., [JKLP93], Sec. 1). Indeed, there is a weight $1/j!$ for every multi-edge of multiplicity j, and a factor $1/2$ for every self-loop:

Theorem 3.6.9 (The law of CM($\mathbf{d}$))**.** *Let $\mathbf{d}=(d_1,\dots,d_n)$ be a degree sequence. Let G be a multigraph with n nodes and weighted adjacency matrix $A=(a_{ij})_{1\le i,j\le n}$ with degree sequence $\mathbf{d}$ such that*

$$d_i = 2a_{ii} + \sum_{j\in[n]} a_{ij} \quad \text{for } i=1,\dots,n.$$

(There are a_{ij} multi-edges between i,j distinct and a_{ii} self-loops at i.) Then we have

$$\begin{aligned}\mathbb{P}(\mathrm{CM}(\mathbf{d})=G) &= \frac{\prod_{i=1}^n d_i!}{((\sum_i d_i)-1)!!}\,\frac{1}{2^{a_{ii}}\prod_{1\le i\le j\le n} a_{ij}!}\\ &\propto \left(\frac{1}{2}\right)^{a_{11}+\cdots+a_{nn}} \prod_{1\le i\le j\le n}\frac{1}{a_{ij}!}.\end{aligned}$$

PROOF. Let $S(G)$ denote the set of all pairings of ℓ_n nodes that give rise to the multigraph G after vertex identificaiton. Then by Propositions 3.6.7 and 3.6.8,

$$\mathbb{P}(\mathrm{CM}(\mathbf{d})=G) = \mathbb{P}(\mathrm{Conf}(\mathbf{d})\in S(G)) = \frac{|S(G)|}{(\ell_n-1)!!}.$$

Hence it remains to show

$$|S(G)| = \frac{\prod_{i=1}^n d_i!}{\prod_{i=1}^n 2^{a_{ii}}\prod_{1\le i\le j\le n} a_{ij}!}.$$

The numerator accounts for the fact that one can permute all d_i half-edges incident to node i for all $i = 1,\dots,n$, which would generate different pairings but still the same multi-graph after node identification. However, when the half-edge-permutation preserves a_{ij} multi-edges between nodes i and j, then it creates the same pairings. The factor $\prod_{1\le i\le j\le n} a_{ij}!$ in the denominator compensates for this fact. Lastly, when two nodes i, j in Conf($\mathbf{d}$) form a self-loop and they are permuted to j, i, then it creates the same pairing and the multi-graph. Hence the factor $\prod_{i=1}^{n} 2^{a_{ii}}$ compensates for this fact. □

Exercise 3.6.10 (Configuration model condition to be simple)**.** Let $\mathbf{d} = (d_1,\dots,d_n)$ be a degree sequence. Let G be a simple graph with degree sequence $\mathbf{d}$. Show that

$$\mathbb{P}\Big(\mathrm{CM}(\mathbf{d}) = G \,\Big|\, \mathrm{CM}(\mathbf{d}) \text{ is simple}\Big) = \frac{\prod_{i=1}^{n} d_i!}{(\ell_n - 1)!!} \frac{1}{\mathbb{P}(\mathrm{CM}(\mathbf{d}) \text{ is simple})}.$$

In particular, conditional on CM($\mathbf{d}$) being simple, CM($\mathbf{d}$) is distributed uniformly among all simple graphs with degree sequence $\mathbf{d}$. (*Hint*: Use Theorem 3.6.9.)

3.7. Stochastic block model

The stochastic block model (SBM) is a generative random graph model. It is designed to exhibit a tendency to generate graphs with 'communities', which are groups of nodes that are linked to each other with specific edge densities. In other words, connections are more frequent within communities than between them. This mathematical concept was initially introduced in 1983 in the social network context by Paul W. Holland and colleagues. The stochastic block model plays a significant role in the fields of statistics, machine learning, and network science, where it serves as a valuable reference for the challenge of identifying community structures within graph data.

3.7.1. Communities. The fundamental objective in community detection or graph clustering is to divide the nodes in a graph into clusters where the connections within each cluster are stronger. Community detection and clustering are fundamental challenges in the fields of machine learning and data mining. Many datasets can be represented as networks of interacting entities, and one of the primary concerns in such networks is to identify similarities between items, either as a final goal or as an initial step toward other learning tasks.

Communities in networks and the problem of detecting them seem to make sense. But what do we mean by "communities", really? Is there any precise definition for them? Not really. So there are several basic questions:

(1) Are there really communities? Various "community detection" algorithms may output "community structures", but are these meaningful or artefacts?
(2) If there are ground-truth communities, can we always extract them; fully, or partially?
(3) If there are no ground-truth communities, what is a good benchmark to measure the performance of "community detection" algorithms?

Stochastic block models provide a solid foundation to answer the above questions on community structures in networks.

3.7.2. Basics of SBM.

Definition 3.7.1 (Stochastic Block Model)**.** *Stochastic block model* (SBM) is a random network model denoted as SBM(W, c) with two parameters: W is the $(k \times k)$ community weight matrix and c is the $(n \times 1)$ community assigment vector with entries from $[k] = \{1, 2, \dots, k\}$. Each node $i \in [n]$ belongs to the community $c(i)$. Two distinct nodes $i, j \in [n]$ are connected by an edge independently with probability $W_{c(i),c(j)}$. There are no self-loops and multi-edges.

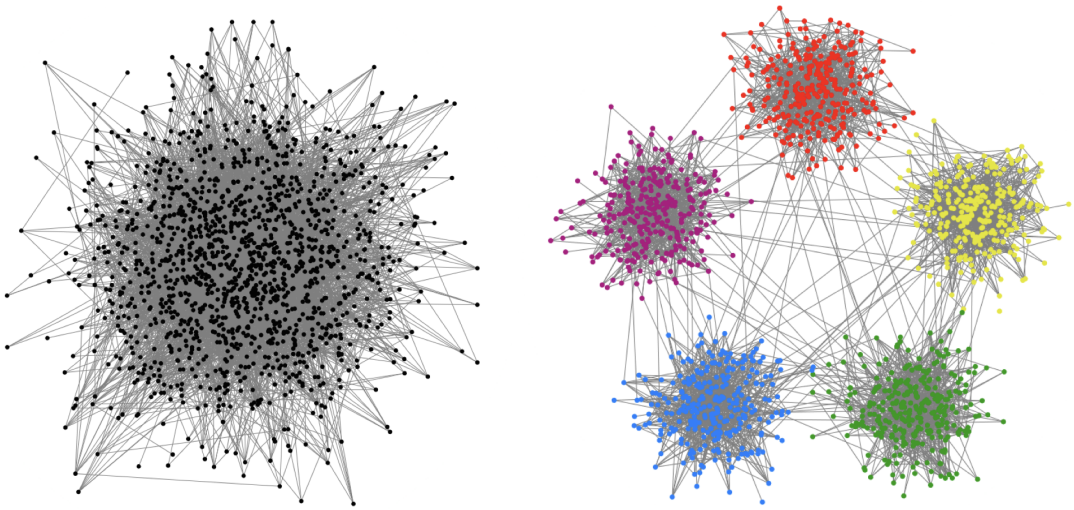


FIGURE 3.7.1. Two isomorphic graphs sampled from SBM with five communities. There are 1000 vertices, 5 balanced communities, within-cluster probability of 1/50 and across-cluster probability of 1/1000. The goal of community detection in this case is to obtain the right graph (with the true communities) from the left graph (scrambled) up to some level of accuracy. In such a context, community detection may be called graph clustering. In general, communities may not only refer to denser clusters but more generally to groups of vertices that behave similarly. Figure excerpted from [Abb17].

Example 3.7.2 (Erdős-Renyí graphs)**.** Take $W = [p]$, the (1×1) matrix of single entry $p \in [0,1]$. Let $c = [1,1,\ldots,1] \in \mathbb{R}^n$. That is, all n nodes belong to the same community, and within each community, two nodes are connected independently with probability p. Thus, SBM(W,c) is the same as $G(n,p)$. That is, these two random graph models generate simple graphs with the same distribution. ▲

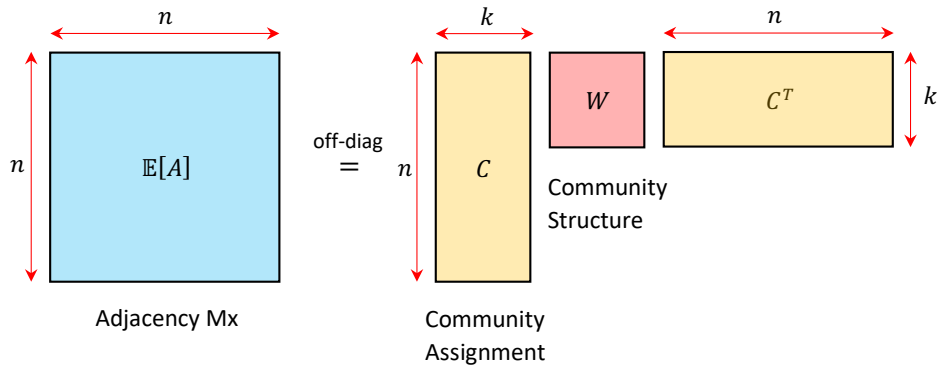


FIGURE 3.7.2. The structure of the expected adjacency matrix of SBM(W,c). $C \in \mathbb{R}^{n\times k}$ is the one-hot encoding matrix of the community assignment vector $c \in \{1,\ldots,k\}^n$. The equality between the two matrices holds only for the off-diagonal entries, as the diagonal entries of A are zero (no self-loops).

Proposition 3.7.3. *Let $G \sim$ SBM(W,c) and let A denote the adjacency matrix of G. (A is a $(n \times n)$ symmetric random binary matrix.) Then we have (see Fig. 3.7.2)*

$$\mathbb{E}[A]_{ij} = [CWC^T]_{ij} \qquad \text{for all } 1 \le i < j \le n,$$

where $C = (n \times k)$ community assignment matrix with entries given by

$$C_{i,j} = \mathbf{1}(\text{node } i \text{ belongs to community } j).$$

PROOF. Let i, j be two distinct nodes in G. One the one hand, recall that $A_{ij} \sim \text{Bernoulli}(p_{ij})$ where $p_{ij} = W_{c(i),c(j)}$. It follows that $\mathbb{E}[A]_{i,j} = p_{ij}$. On the other hand, for each $1 \le \ell \le n$, let C_ℓ denote the ℓth row of the $(n \times k)$ matrix of C. Note that

$$C_\ell = [0, \ldots, 0, 1, 0 \ldots, 0]$$

where the 1 is at the $c(\ell)$th position. Then

$$\left[CWC^T\right]_{i,j} = C_i W C_j^T = W_{c(i),c(j)} = \mathbb{E}[A]_{i,j}.$$

This shows the assertion. □

Based on Proposition 3.7.3, we can implement SBM in Python as in Figure 3.7.3.

```
def SBM(W, c):
    # Stochastic block model;
    # W = (k x k) community weight matrix
    # c = (n x 1), entries from [k]; community assignment vector
    k = W.shape[0]
    n = len(c)

    # C = (n x k) one-hot encoding of community assignment matrix
    C = list2onehot(c, list_classes=[i for i in range(k)])

    # C = (n x n) probability matrix = expected adjacency matrix = C W C.T
    P = C @ W @ C.T

    # Now sample the edges according to P
    G = nx.Graph()
    G.add_nodes_from(range(n))
    nodes = list(G.nodes())

    for i in np.arange(n):
        for j in np.arange(i+1,n):
            U = np.random.rand()
            if U < P[i,j]:
                G.add_edge(nodes[i],nodes[j])
    return G
```

FIGURE 3.7.3. A Python implementation of SBM.

Using the code above, we can sample graphs from SBM models with various choices of W and c. The examples in Figure 3.7.4 are drawn from SBM(W, c) where the community weight matrix W is (3×3) with within-cluster probability of 0.8 and between-cluster probability of 0.1 and c assigns r nodes per each cluster for $r \in \{3, 10, 30, 50\}$. We can observe community structure with three communities, which becomes more evident when each cluster has more nodes. However, as shown in Figure 3.7.5, if we use a different community weight matrix W, now with smaller within-cluster probability of 0.5 and larger between-cluster probability of 0.3, then the community structure is not quite visible even with large number of nodes per cluster.

Even if we see clear community structure in one drawing of a SBM graph, it might be completely invisible if we permute the nodes and re-draw the same graph, see Figure 3.7.6. Hence, if we are given a graph with possibly a community structure there, it is still a non-trivial task to figure out the communities and the node assignments.

3.7.3. Spectral clustering. Let $n = 2m$ and let $c = [1, \ldots, 1, 2, \ldots, 2] \in \mathbb{R}^{2m}$ be the community assignment vector, where 1s and 2s are repeated m times. Let C be the $n \times 2$ community assignment matrix. Take the following 2×2 community weight matrix

$$W = \begin{bmatrix} p & q \\ q & p \end{bmatrix},$$

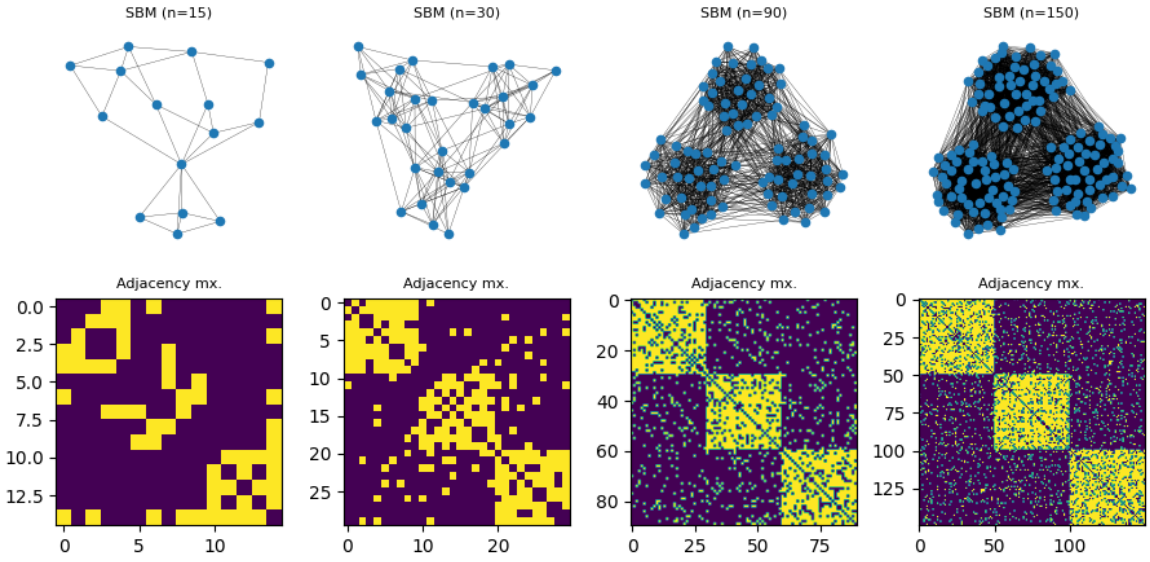


FIGURE 3.7.4. Realizations of SBM(W,c), where the community weight matrix W is (3×3) with within-cluster probability of 0.8 and between-cluster probability of 0.1 and c assigns r nodes per each cluster for $r\in\{3,10,30,50\}$.

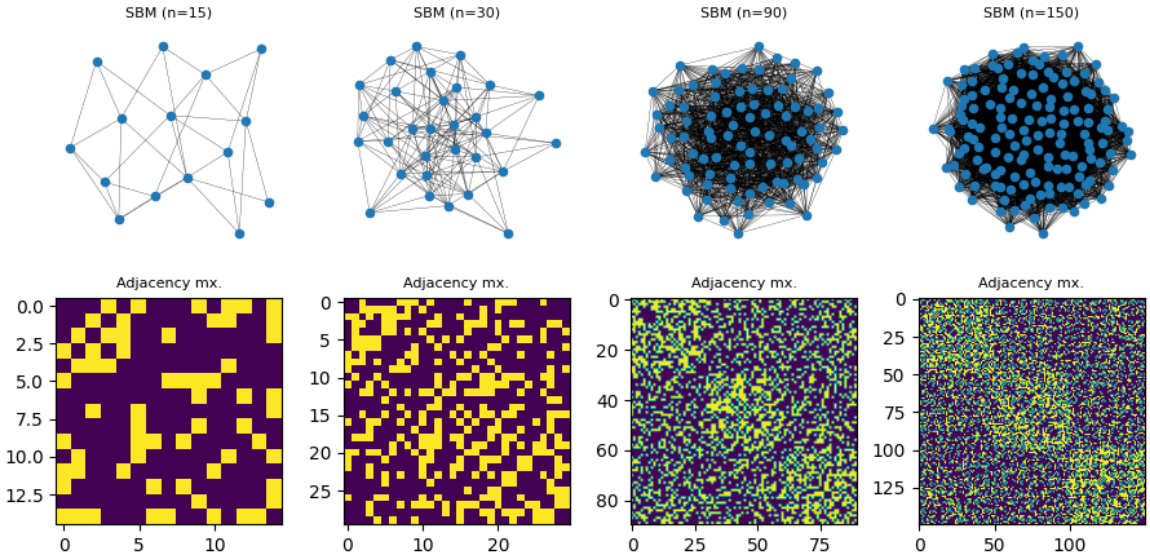


FIGURE 3.7.5. Realizations of SBM(W,c), where the community weight matrix W is (3×3) with within-cluster probability of 0.5 and between-cluster probability of 0.3 and c assigns r nodes per each cluster for $r\in\{3,10,30,50\}$.

where $0<q<p<1$. Then

$$P := CWC^T = \left[\begin{array}{ccc|ccc} p & \cdots & p & q & \cdots & q \\ \vdots & & \vdots & \vdots & & \vdots \\ p & \cdots & p & q & \cdots & q \\ \hline q & \cdots & q & p & \cdots & p \\ \vdots & & \vdots & \vdots & & \vdots \\ q & \cdots & q & p & \cdots & p \end{array}\right].$$

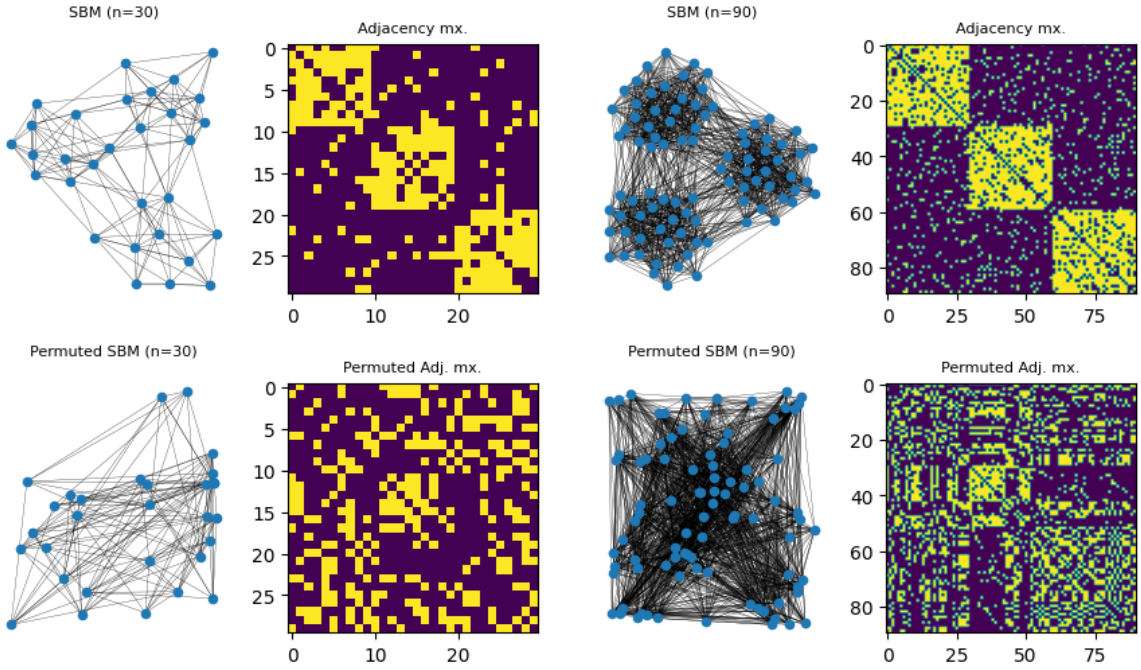


FIGURE 3.7.6. Realizations of SBM(W,c), where the community weight matrix W is (3×3) with within-cluster probability of 0.5 and between-cluster probability of 0.3 and c assigns r nodes per each cluster for $r\in\{3,10,30,50\}$.

The above matrix is symmetric and has rank 2, so there are two distinct real eigenvalues. The eigenvalue-eigenvector pairs are

$$\lambda_1=\frac{(p+q)n}{2}, \qquad v_1=\frac{1}{\sqrt{n}}[1,\ldots,1]\in\mathbb{R}^n,$$
$$\lambda_2=\frac{(p-q)n}{2}, \qquad v_2=\frac{1}{\sqrt{n}}[1,\ldots,1,-1,\ldots,-1]\in\mathbb{R}^n.$$

Notice that the sign of the coordinates of the second eigenvector v_2 coincides exactly with the true community structure. Importantly, this observation is still true if we permute the nodes arbitrarily. Therefore, *the signs of the second eigenvector of P above reveals the community membership of all nodes.*

To see the above claim, suppose we permute the node labels by a permutation σ on $\{1,2,\ldots,n\}$ so that the new community assignment vector c' is

$$c'=[c_{\sigma(1)},\ldots,c_{\sigma(n)}].$$

Let Q denote the $n\times n$ permutation matrix, where its ith row is the indicator vector of the value $\sigma(i)$ (so only a single 1 in each row). Note that Q^T corresponds to the inverse permutation σ^{-1}, so $Q^TQ=I$, that is, Q is an orthonormal matrix (see Exc. 3.7.5). Let C' be the community assignment matrix corresponding to c'. Noting that $C'=QC$ (see Exc. 3.7.5),

$$C'W(C')^T=Q(CWC^T)Q^T.$$

Hence the matrix in the LHS above is a conjugation of CWC^T, so they have the same eivengalues. For the idenvectors, note that $CWC^T v_2 = \lambda_2 v_2$, so denoting $v_2' := Qv_2$,

$$\begin{aligned} C'W(C')^T v_2' &= Q(CWC^T)Q^T Q v_2 \\ &= QCWC^T v_2 \\ &= Q\lambda_2 v_2 \\ &= \lambda_2 Q v_2 \\ &= \lambda_2 v_2'. \end{aligned}$$

Thus v_2' is the eigenvector of $C'W(C')^T$ corresponding to λ_2. Furthermore, multiplying v_2 to Q amounts to permuting the entries of v_2 by the permutation σ (see Exc. 3.7.5), so

$$Qv_2 = \frac{1}{\sqrt{n}}[(v_2)_{\sigma(1)}, \ldots, (v_2)_{\sigma(n)}] \in \mathbb{R}^n.$$

From this, we conclude that the signs of v_2' still give the correct community assignment vector.

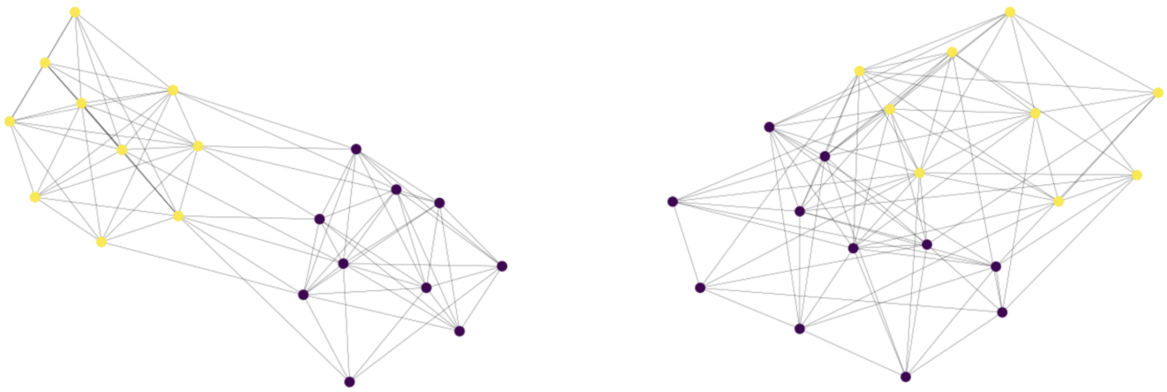


FIGURE 3.7.7. Example of spectral clustering on two-community SBM graphs.

By Proposition 3.7.3, we know that P coincides with $\mathbb{E}[A]$ on the off-diagonal entries, where A is the adjacency matrix of the random graph $G \sim \text{SBM}(W, c)$. If we write

$$A = P + E,$$

where E is the random 'error matrix', then the contribution of E to the eigenvalues/eigenvectors of A is small[4]. Therefore, the second eigenvector of A (the observed adjacency matrix) should be very close to the second eigenvector of P (the expected adjacency matrix). This suggests the following "spectral clustering algorithm":

Spectral clustering

1. Compute the second eigenvector v_2 of the observed adjacency matrix A.

2. Cluster nodes based on the sign of the entries of v_2.

Note that the spectral clustering algorithm in Figure 3.7.8 only works for detecting $k = 2$ (non-overlapping) clusters. Is there a variant of spectral clustering algorithms for $k \geq 2$ clusters? The answer is yes, and it takes the form of spectral embedding (representing each node as a d dimensional vectors using top d eigenvectors of the Laplacian matrix, and then applying k-means clustering on them). We will get back to this algorithm in next section.

[4]Use Wyle's theorem and the fact that $\|E\|_{op} = O(\sqrt{n})$ with high probability. The entries of A are random, but its spectral properties are more or less the same.

```python
def spectral_clustering(G, k=2):
    ### Spectral clustering of a graph G into k disjoint clusters
    ### Currently it only runs with k=2.
    ### Larger k needs spectral embedding + K-means clustering. (Will cover it later)
    A1 = nx.adjacency_matrix(G).toarray()
    w, v = np.linalg.eigh(A1)
    v2 = v[:,-2]
    c_hat = np.sign(v2)
    nx.draw(G, node_color=c_hat, with_labels=False, width=0.2, node_size=50)
    return c_hat
```

FIGURE 3.7.8. A Python implementation of spectral clustering with $k = 2$.

See Figure 3.7.7 for examples of spectral clustering applied to graphs generated from SBM with two communities. In the example below, we show examples of applying spectral clustering for graphs without ground-truth communities.

Example 3.7.4 ($k = 2$ spectral clustering on a CALTECH subgraph)**.** Take a subgraph H (86 nodes, 734 edges) of CALTECH, which was sampled by taking 100 steps of random walk on CALTECH and taking the induced subgraph on the sampled nodes. The sampled subgraph a priori does not have true communities to be discoverd. Nonetheless, we can apply various community detection algorithms and hope that we can identify some groups of closely connected nodes. In Figure 3.7.9, we show two communities captured by the spectral clustering algoirthm that "detects" two communities. There some metrics that evaluates how good a given community assignment is (e.g., modularity). We will get back to this topic later.

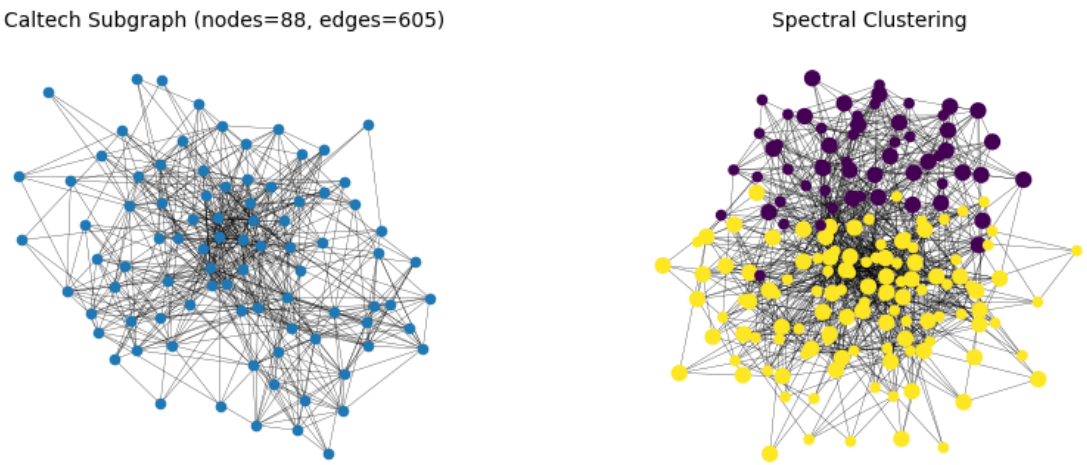


FIGURE 3.7.9. Spectral clustering with $k = 2$ on a subgraph of CALTECH.

▲

Exercise 3.7.5 (Permutation matrices)**.** Let σ be a permutation on the coordinates $\{1,\dots,n\}$. The permutation matrix Q^σ is an $n \times n$ matrix defined by

$$Q_{ij} = \mathbf{1}(\sigma(i) = j) \quad \text{for all } 1 \le i, j \le n.$$

(i) Show that $Q^{\sigma^{-1}} = (Q^\sigma)^T$ and $(Q^\sigma)^{-1} = (Q^\sigma)^T = Q^{\sigma^{-1}}$.
(ii) Let $v = [v_1,\dots,v_n] \in \mathbb{R}^n$ be a vector. Show that

$$Q^\sigma v = [v_{\sigma(1)},\dots,v_{\sigma(n)}].$$

(iii) Let $c = [c_1, \ldots, c_n] \in \{1, \ldots, k\}^n$ be a community assignment vector and let $C \in \mathbb{R}^{n \times k}$ be the corresponding community assignment matrix. Let $c' := [c_{\sigma(1)}, \cdots, c_{\sigma(n)}]$ be the community assignement vector after permuting the nodes by σ. Let C' be the community assigment matrix corresponding to c'. Show that

$$C' = Q^{\sigma} C.$$

CHAPTER 4

Essential Algorithms on Networks

4.1. Random walks on graphs

In this section, we will study one of the most fundamental randomized algorithms on networks: *Random Walks* (RW). Starting from a given node, one selects a random neighbor (typically uniformly at random) and jumps there, and repeat this process. The result is randomly walking on the graph, always along the edges. This seemingly simple-minded procedure on graphs is surprisingly powerful and reveals a lot of information about the structure of the graph. Also, RW serves as an important tool to sample subgraphs from networks by only using local information. Mathematically, RW can be analyzed within the framework of Markov chains.

Definition 4.1.1 (Random walk on graphs)**.** Let $G = (V, E)$ be a graph. The (simple symmetric)[1] *Random walk* on G is a sequence of random nodes $(X_t)_{t\geq 0}$ such that

$$X_{t+1} \,|\, X_t \quad \sim \quad \text{Uniform}(N(X_t)).$$

That is, X_{t+1} given X_t is chosen uniformly at random among the neighbors of X_t.

```python
def RW(G, x0=None, steps=1, return_history=False):
    # simple symmetric random walk on graph G
    # initialization at x0
    if x0 is None:
        x = np.random.choice(G.nodes())
    else:
        x = x0

    history = []
    for i in np.arange(steps):
        if len(list(G.neighbors(x))) == 1:
            print("RW is stuck at isolated node")
            x = np.random.choice(G.nodes()) # re-initialize uniformly at random
        else:
            x = np.random.choice(list(G.neighbors(x)))

        if return_history:
            history.append(x)

    if not return_history:
        return x
    else:
        return history
```

FIGURE 4.1.1. A Python implementation of random walks on graphs.

4.1.1. Motivation: Sampling subgraphs from sparse networks. Perhaps the simplest way of sampling k-node subgraph from a graph G is by choosing the node set uniformly at random, and then taking the induced subgraph on that random node set. This way of sampling a k-node subgraph is called the *independent sampling* (see Fig. 4.1.2).

[1] A random walk is 'simple' if it always jumps to the nearest neighbors. On lattice graphs, one can think of random walks that can jump to nodes that are not immediate neighbors. A random walk is 'symmetric' if each move is unbiased. One can give bias to a random walk, e.g., so that it prefers to jump to a node with large degree.

```python
def IID_sampling(G, steps=1):
    # sample uniformly random nodes independently 'steps' times
    history = []
    for i in np.arange(steps):
        x = np.random.choice(G.nodes()) # re-initialize uniformly at random
        history.append(x)
    return history
```

FIGURE 4.1.2. A Python implementation of random walks on graphs.

While independent sampling this is extremely simple to implement and is a basis of other more sophisticated sampling algorithms (e.g., importance sampling), it is not a good subgraph sampling algorithm for large sparse graphs, since the expected edge density of the random subgraph is going to be the same. For instance, CALTECH has edge density 0.056. Sampling a $k = 100$ node subgraph by independent sampling will give a random subgraph H with expected edge density 0.056. Hence the expected number of edges in H is $\binom{100}{2}0.056 = 277.2$. A random subgraph with a much smaller number of nodes with the same small edge density tend to be disconnected (see Fig. 4.1.3). More details on this is given in Exercise 4.1.2.

On the contrary, subgraph sampling by RW will always give a connected subgraph. That is, we perform random walk on a graph k steps (Fig. 4.1.9) or until we collect k distinct nodes, and then take the induced subgraph on the set of sampled nodes. Subgraphs sampled by RWs are always connected and tend to contain significantly more edges, better capturing the local geometry in the network (see Fig. 4.1.3 left).

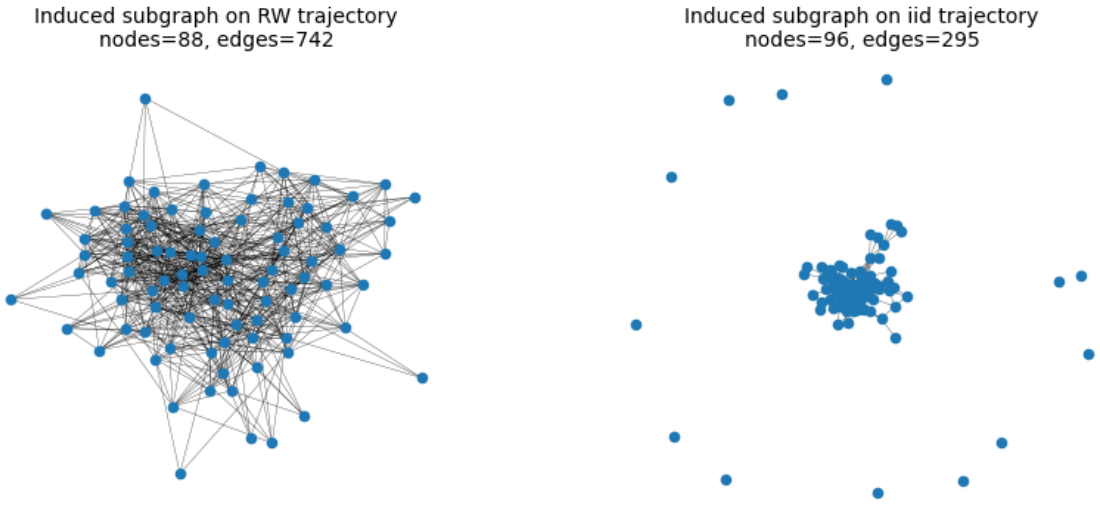


FIGURE 4.1.3. Two induced subgraphs from CALTECH with node sets sampled by (left) a 100-step RW trajectory and (right) 100 times of i.i.d. uniform sampling.

Exercise 4.1.2 (Independent sampling and expected edge density)**.** Let $G = (V, E)$ be a simple graph. Let A, B be two independent, distinct, and uniformly chosen nodes in G.

(i) Show that

$$\mathbb{P}(A \text{ and } B \text{ are adjacent}) = \frac{|E|}{\binom{|V|}{2}} = \text{edge density of } G.$$

(*Hint:* Use iterated expectation and the fact that conditional on A, B is uniformly distributed over $V \setminus \{A\}$.)

(ii) Let $A_1, \dots, A_k$ be k independent, distinct, and uniformly chosen nodes in G. Let H denote the induced subgraph of G on the random node set $\{A_1, \dots, A_k\}$. (This way of sampling a k-node subgraph is called the *independent sampling*.) Show that

$$\mathbb{E}\left[\frac{|E(H)|}{\binom{k}{2}}\right] = (\text{edge density of } G).$$

(*Hint:* Note that you can write $|E(H)| = \sum_{1\le i<j\le k} \mathbf{1}(i \sim j)$. Use linearity of expectation and **(i)**.)

(iii) Explain why independent sampling may not be a good way to sample subgraphs from a large and sparse graphs.

4.1.2. Elementary results on RWs on graphs. In this section, we summerize some elementary results on random walks on connected graphs. First, if we run a random walk for many steps, how often will it visit a particular node v in the graph? Would it visit v quite frequently if v has large degree? Theorem 4.1.3 shows that the proportion of times is proportional to the degree. Theorem 4.1.3 is empirically verified in the experiments shown in Figure 4.1.4.

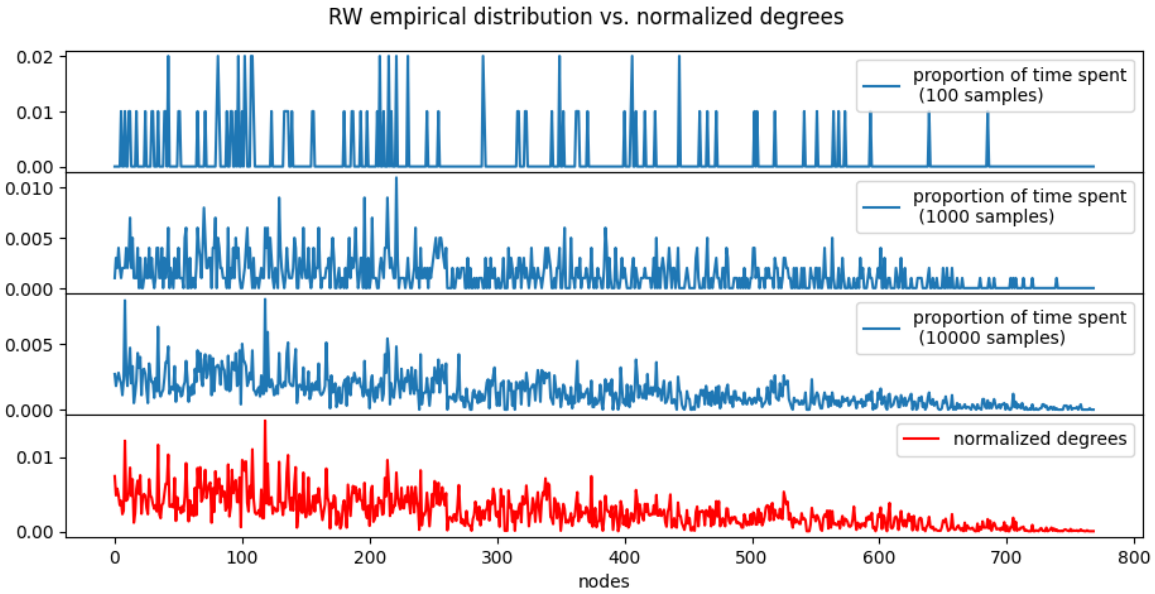


FIGURE 4.1.4. Normalized proportion of times that RW on CALTECH spends at each node for $N = 100, 1000, 10000$ steps (top three rows) and the normalized degrees $\deg(v)/2|E|$ (bottom).

Theorem 4.1.3 (Empirical frequency of RWs on G)**.** *Let $(X_t)_{t\ge 0}$ denote (simple symmetric) random walk on a connected graph $G = (V, E)$. Then*

$$\lim_{N\to\infty} \frac{1}{N} \sum_{t=1}^{N} \mathbf{1}(X_t = v) = \frac{\deg(v)}{2|E|}.$$

That is, the asmpyotitc proportion of times that the RW spends at a node is proportional to its degree.

PROOF. Omitted. See [SV, Thm 2.94]. □

Theorem 4.1.3 says that the probability distribution on the nodes of G given by normalizing the degrees is a special distribution that describes the long-term behavior of the RW on G in terms of the proportion of times spent at each node. In fact, this distribution is 'stationary' with respect to RW on G, in the sense of Theorem 4.1.4.

Theorem 4.1.4 (Stationary distribution of RWs on G)**.** *Let $(X_t)_{t\geq 0}$ denote (simple symmetric) random walk on a connected graph $G = (V, E)$, $V = [n]$. Suppose the initial location X_0 is distributed according to the following probability distribution*

$$\pi := \frac{1}{2|E|}[\deg(1), \deg(2), \ldots, \deg(n)]. \tag{14}$$

(That is, $\mathbb{P}(X_0 = v) = \pi(v)$ for each $v \in V$.) Then for each $t \geq 0$, X_t is distributed according to π. Furthermore, π above is the unique such probability distribution.

PROOF. See Section 4.1.3. □

Next, what happens to the distribution of X_t when t is large? Note that this the distribution of the random walk at time t, which is NOT the same as the empirical proportion of times up to t as in Theorem 4.1.3. However, remarkably, it turns out that the random walk after many steps is distributed accoriding to the same limiting distribution π in (14), provided that the underlying graph is *non-bipartite*. We have seen that a graph is non-bipartite if and only if it contains at least one odd cycle (Prop. 2.4.6). This convergence of multi-step distribution of RWs on non-bipartite graphs is stated in Theorem 4.1.5 below.

Theorem 4.1.5 (Convergence of multi-step distribution of RW on graphs)**.** *Let $(X_t)_{t\geq 0}$ be a random walk on a connected graph $G = (V, E)$ with an odd cycle. Let π denote the unique stationary distribution π. Then for each $x, y \in V$,*

$$\lim_{t\to\infty} \mathbb{P}(X_t = y \,|\, X_0 = x) = \pi(y).$$

PROOF. Omitted. □

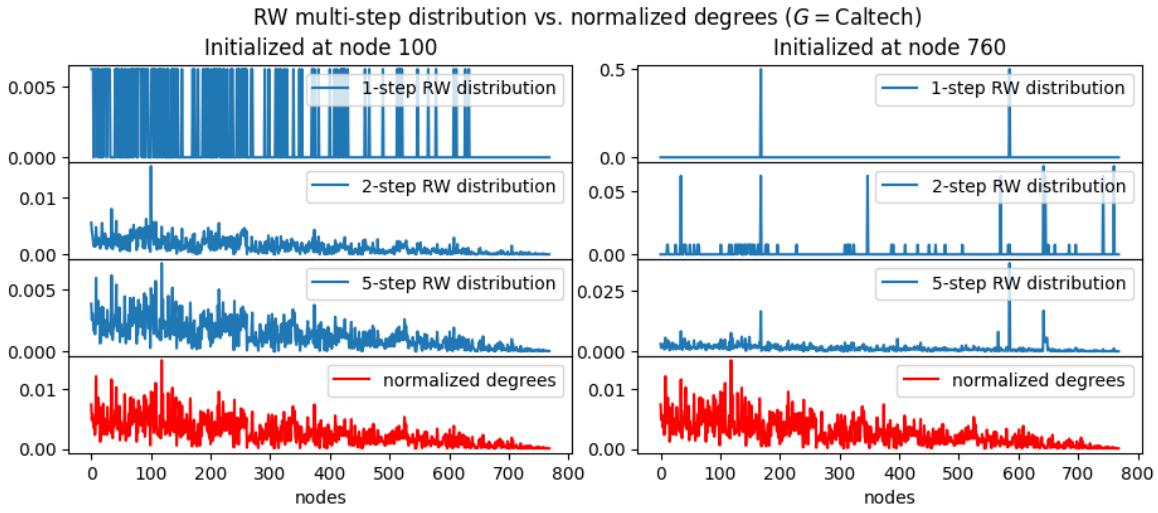


FIGURE 4.1.5. Comparison of multi-step distribution of RW on CALTECH with two initializations

What goes wrong with RWs on bipartite graphs?

Example 4.1.6 (RW on torus)**.** Let $\mathbb{Z}_n$ be the set of integers modulo n. Let $G = (V, E)$ be a graph where $V = \mathbb{Z}_n \times \mathbb{Z}_n$ and two nodes $u = (u_1, u_2)$ and $v = (v_1, v_2)$ are adjacent if and only if

$$|u_1 - u_2| + |v_1 - v_2| = 1.$$

Such a graph G is called the $n \times n$ *torus* and we write $G = \mathbb{Z}_n \times \mathbb{Z}_n$. Intuitively, it is obtained from the $n \times n$ square grid by adding boundary edges to wrap around (see Figure 4.1.6 left).

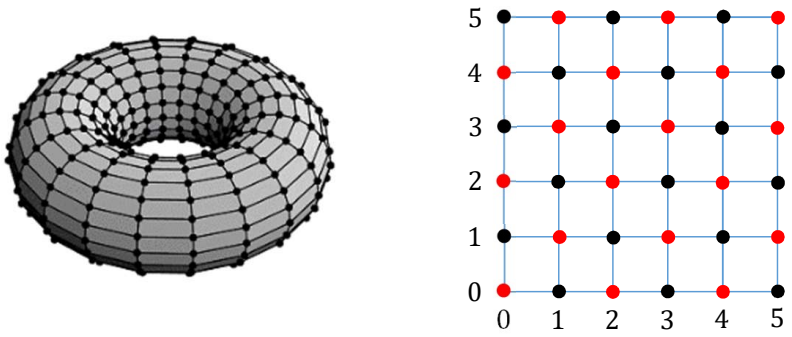


FIGURE 4.1.6. (Left) Torus graph (Figure excerpted from [LP17]). (Right) RW on torus $G = \mathbb{Z}_6 \times \mathbb{Z}_6$ has period 2.

Now let $(X_t)_{t\geq 0}$ be a random walk on G. Since G is connected, X_t is irreducible. Since all nodes in G have degree 4, the uniform distribution on $\mathbb{Z}_n \times \mathbb{Z}_n$, which we denote by π, is the unique stationary distribution of X_t. Let π_t denote the distribution of X_t.

For instance, consider $G = \mathbb{Z}_6 \times \mathbb{Z}_6$. As illustrated in Figure 4.1.6 below, observe that if X_0 is one of the red nodes (where sum of coordinates is even), then X_t is at a red node for any t = even and at a black node (where sum of coordinates is odd) at t = odd. Hence, π_t is supported only on the 'even' nodes for even times and on the 'odd' nodes for the odd times. Hence π_t does not converge in any sense to the uniform distribution π.

The following example of random walks on cycles in Figure 4.1.7 illustrates the same point on the 'one-dimensional torus'. Note that even after 200 steps, RW on C_{14} has zero probability of being at every other nodes, while for RW on C_{15}, the probabilities evens out.

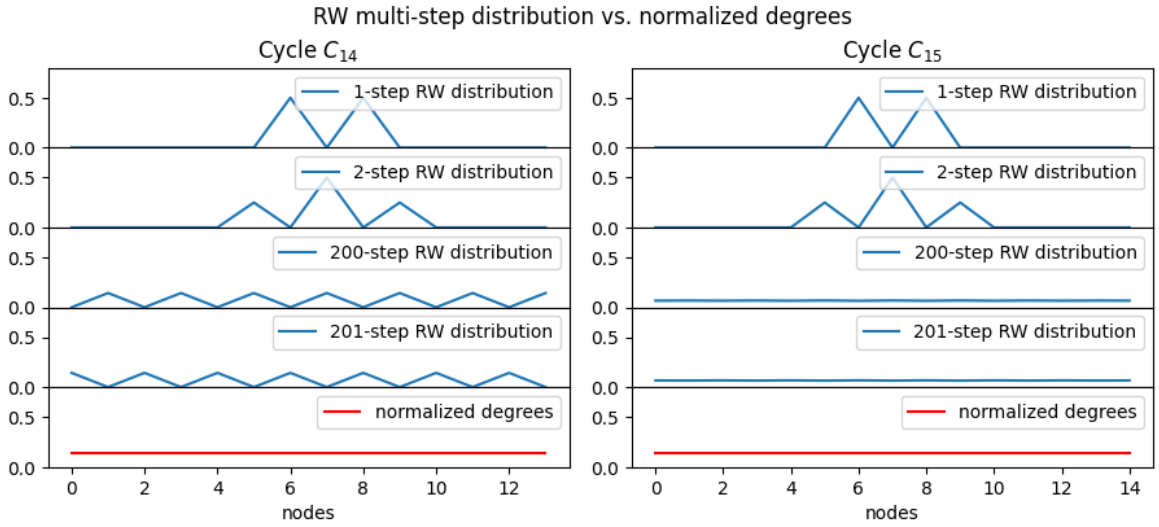


FIGURE 4.1.7. Comparison of multi-step distribution of RW on C_n

▲

4.1.3. Basics of Markov chains. We introduce a brief Markov chain theory in order to analyze random walks on graphs and networks in the following section.

Roughly speaking, *Markov processes* are used to model temporally changing systems where future state only depends on the current state. For instance, if the price of bitcoin tomorrow depends only on its price today, then bitcoin price can be modeled as a Markov process. (Of course, the entire history of price often affects decisions of buyers/sellers so it may not be a realistic assumption.)

Even through Markov processes can be defined in vast generality, we concentrate on the simplest setting where the state and time are both discrete.

Definition 4.1.7 (Markov chains)**.** Let $\Omega = \{1, 2, \cdots, m\}$ be a finite set, which we call the *state space*. Consider a sequence $(X_t)_{t\geq 0}$ of Ω-valued RVs, which we call a *chain*. We call the value of X_t the *state* of the chain at time t. In order to narrow down the way the chain $(X_t)_{t\geq 0}$ behaves, we introduce the following properties:

(i) (Markov property) The distribution of X_{t+1} given the history $X_0, X_1, \cdots, X_t$ depends only on X_t. That is, for any values of $j_0, j_1, \cdots, j_t, k \in \Omega$,

$$\mathbb{P}(X_{t+1} = k \,|\, X_t = j_t, X_{t-1} = j_{t-1}, \cdots, X_1 = j_1, X_0 = j_0) = \mathbb{P}(X_{t+1} = k \,|\, X_t = j_t).$$

(ii) (Time-homogeneity) The transition probabilities

$$p_{ij} = \mathbb{P}(X_{t+1} = j \,|\, X_t = i) \qquad i, j \in \Omega$$

do not depend on t.

When the chain $(X_t)_{t\geq 0}$ satisfies the above two properties, we say it is a (discrete-time and time-homogeneous) *Markov chain*. We define the *transition matrix* P to be the $m \times m$ matrix of transition probabilities:

$$P = (p_{ij})_{1\leq i,j\leq m} = \begin{bmatrix} p_{11} & p_{12} & \cdots & p_{1m} \\ p_{21} & p_{22} & \cdots & p_{2m} \\ \vdots & \vdots & & \vdots \\ p_{m1} & p_{m2} & \cdots & p_{mm} \end{bmatrix}.$$

Finally, since the state X_t of the chain is a RV, we represent its probability mass function (PMF) via a row vector

$$\mathbf{r}_t = [\mathbb{P}(X_t = 1), \mathbb{P}(X_t = 2), \cdots, \mathbb{P}(X_t = m)].$$

The most important example for Markov chains for us is the random walk on graphs.

Example 4.1.8 (Random walk on graphs as a Markov chain)**.** For a simple graph $G = (V, E)$, let A_G denote its adjacency matrix. Consider we 'walk around' the nodes of a given simple graph $G = (V, E)$ following edges: at each time, we jump from one node to one of the neighbors with equal probability. For instance, if we are currently at node 2 and if 2 is adjacent to 3, 5, and 6, then we jump to one of the three neighbors with probability 1/3. The location of this jump process at time t can be described as a Markov chain. Namely, a Markov chain $(X_t)_{t\geq 0}$ on the node set V is called a *random walk on* G if

$$\mathbb{P}(X_{t+1} = j \,|\, X_t = i) = \frac{A_G(i, j)}{\deg_G(i)}. \tag{15}$$

Note that its transition matrix P is obtained by normalizing each row of the adjacency matrix A_G by the corresponding degree (see Fig. 4.1.8). In a matrix form, the transition matrix P of RW on G is given by

$$P = D^{-1} A_G,$$

where D is the $|V| \times |V|$ diagonal matrix of degrees: $D = \mathrm{Diag}(\deg(1), \ldots, \deg(|V|))$.

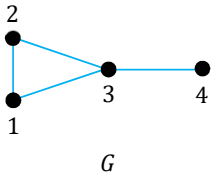
▲

Example 4.1.9. Let $\Omega = \{1, 2\}$ and let $(X_t)_{t\geq 0}$ be a Markov chain on Ω with the following transition matrix

$$P = \begin{bmatrix} p_{11} & p_{12} \\ p_{21} & p_{22} \end{bmatrix}.$$

```python
def RW_transition_mx(G):
    ### Compute random walk transition matrix of a graph G
    A = nx.adjacency_matrix(G).todense()
    P = np.zeros(shape=A.shape)
    nodes = list(G.nodes())
    for i in np.arange(A.shape[0]): ## normalize rows of A by degree
        if G.degree(nodes[i])>0:
            P[i,:] = A[i,:]/G.degree(nodes[i])
    return P
```

FIGURE 4.1.8. A Python implementation of computing the transition matrix of RW on G.



G

$$A_G = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \qquad P = \begin{bmatrix} 0 & 1/2 & 1/2 & 0 \\ 1/2 & 0 & 1/2 & 0 \\ 1/3 & 1/3 & 0 & 1/3 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

FIGURE 4.1.9. A 4-node simple graph G, its adjacency matrix A_G, and associated random walk transition matrix P

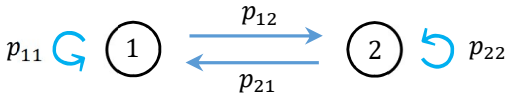


FIGURE 4.1.10. State space diagram of a 2-state Markov chain

We can also represent this Markov chain pictorially as in Figure 4.1.10, which is called the 'state space diagram' of the chain $(X_t)_{t\geq 0}$.

For some concrete example, suppose

$$p_{11} = 0.2, \quad p_{12} = 0.8, \quad p_{21} = 0.6, \quad p_{22} = 0.4.$$

If the initial state of the chain X_0 is 1, then

$$\begin{aligned} \mathbb{P}(X_1 = 1) &= \mathbb{P}(X_1 = 1 \,|\, X_0 = 1)\mathbb{P}(X_0 = 1) + \mathbb{P}(X_1 = 1 \,|\, X_0 = 2)\mathbb{P}(X_0 = 2) \\ &= \mathbb{P}(X_1 = 1 \,|\, X_0 = 1) = p_{11} = 0.2 \end{aligned}$$

and similarly,

$$\begin{aligned} \mathbb{P}(X_1 = 2) &= \mathbb{P}(X_1 = 2 \,|\, X_0 = 1)\mathbb{P}(X_0 = 1) + \mathbb{P}(X_1 = 2 \,|\, X_0 = 2)\mathbb{P}(X_0 = 2) \\ &= \mathbb{P}(X_1 = 2 \,|\, X_0 = 1) = p_{12} = 0.8. \end{aligned}$$

Also we can compute the distribution of X_2. For example,

$$\begin{aligned} \mathbb{P}(X_2 = 1) &= \mathbb{P}(X_2 = 1 \,|\, X_1 = 1)\mathbb{P}(X_1 = 1) + \mathbb{P}(X_2 = 1 \,|\, X_1 = 2)\mathbb{P}(X_1 = 2) \\ &= p_{11}\mathbb{P}(X_1 = 1) + p_{21}\mathbb{P}(X_1 = 2) \\ &= 0.2 \cdot 0.2 + 0.6 \cdot 0.8 = 0.04 + 0.48 = 0.52. \end{aligned}$$

In general, the distribution of X_{t+1} can be computed from that of X_t via a simple linear algebra. Note that for $i = 1, 2$,

$$\begin{aligned} \mathbb{P}(X_{t+1} = i) &= \mathbb{P}(X_{t+1} = i \,|\, X_t = 1)\mathbb{P}(X_t = 1) + \mathbb{P}(X_{t+1} = i \,|\, X_t = 2)\mathbb{P}(X_t = 2) \\ &= p_{1i}\mathbb{P}(X_t = 1) + p_{2i}\mathbb{P}(X_t = 2). \end{aligned}$$

This can be written as

$$[\mathbb{P}(X_{t+1}=2),\mathbb{P}(X_{t+1}=2)] = [\mathbb{P}(X_{t+1}=2),\mathbb{P}(X_{t+1}=2)]\begin{bmatrix} p_{11} & p_{12} \\ p_{21} & p_{22}\end{bmatrix}.$$

That is, if we represent the distribution of X_t as a row vector, then the distribution of X_{t+1} is given by multiplying the transition matrix P to the left. ▲

We generalize our observation in Example 4.1.9 in the following exercise.

Exercise 4.1.10. Let $(X_t)_{t\geq 0}$ be a Markov chain on state space $\Omega = \{1,2,\cdots,m\}$ with transition matrix $P = (p_{ij})_{1\leq i,j\leq m}$. Let $\mathbf{r}_t = [\mathbb{P}(X_t=1),\cdots,\mathbb{P}(X_t=m)]$ denote the row vector of the distribution of X_t.

(i) Show that for each $i\in\Omega$,

$$\mathbb{P}(X_{t+1}=i) = \sum_{j=1}^{m} p_{ji}\mathbb{P}(X_t=j).$$

(ii) Show that for each $t\geq 0$,

$$\mathbf{r}_{t+1} = \mathbf{r}_t P.$$

(iii) Show by induction that for each $t\geq 0$,

$$\mathbf{r}_t = \mathbf{r}_0 P^t.$$

Exercise 4.1.11. Let $(X_t)_{t\geq 0}$ be a Markov chain on state space $\Omega = \{1,2,\cdots,m\}$ with transition matrix $P = (p_{ij})_{1\leq i,j\leq m}$.

(i) (Multi-step transition prob.) Show that for each $x,y\in\Omega$ and $a,b\geq 1$,

$$\mathbb{P}\left(X_{a+b}=y \,|\, X_a=x\right) = P^b(x,y).$$

Hint: Use Exercise 4.1.10.

(ii) (The Chapman-Kolmogorov eq.) Show that for each $x,y\in\Omega$ and $n,m\geq 1$,

$$P^{n+m}(x,y) = \sum_{z\in\Omega} P^n(x,z)P^m(z,y).$$

While right-multiplication of P advances a given row vector of distribution one step forward in time, left-multiplication of P on a column vector computes the expectation of a given function with respect to the future distribution. This point is clarified in the following exercise.

Exercise 4.1.12. Let $(X_t)_{t\geq 0}$ be a Markov chain on a state space $\Omega = \{1,2,\cdots,m\}$ with transition matrix P. Let $f:\Omega\to\mathbb{R}$ be a function. Suppose that if the chain X_t has state x at time t, then we get a 'reward' of $f(x)$. Let $\mathbf{r}_t = [\mathbb{P}(X_t=1),\cdots,\mathbb{P}(X_t=m)]$ be the distribution of X_t. Let $\mathbf{v} = [f(1),f(2),\cdots,f(m)]^T$ be the column vector representing the reward function f.

(i) Show that the expected reward at time t is given by

$$\mathbb{E}[f(X_t)] = \sum_{i=1}^{m} f(i)\mathbb{P}(X_t=i) = \mathbf{r}_t\mathbf{v}.$$

(ii) Use part (i) and Exercise 4.1.10 to show that

$$\mathbb{E}[f(X_t)] = \mathbf{r}_0 P^t \mathbf{v}.$$

(iii) The total reward up to time t is a RV given by $R_t = \sum_{k=0}^{t} f(X_t)$. Show that

$$\mathbb{E}[R_t] = \mathbf{r}_0(I+P+P^2+\cdots+P^t)\mathbf{v}.$$

At each time, the state of the Markov chain X_t is a random variable, so it is described by its probability distribution over the state space Ω. A special probability distirbution π on Ω is a 'stationary distribution' of the Markov chain if X_t follows π, then X_{t+1} also follows π for all t. A stationary distribution is useful in analyzing the long-term behavior of the Markov chain.

Definition 4.1.13 (Stationary distribution of a MC)**.** Let $(X_t)_{t\geq 0}$ be a Markov chain on state space $\Omega = \{1, 2, \cdots, m\}$ with transition matrix P. If π is a distribution on Ω such that

$$\pi = \pi P,$$

then we say π is a *stationary distribution* of the Markov chain $(X_t)_{t\geq 0}$.

In Exercise 4.1.10, we observed that we can simply multiply the transition matrix P to a given row vector $\mathbf{r}_t$ of distribution on the state space Ω in order to get the next distribution $\mathbf{r}_{t+1}$. Hence if the initial distribution of the chain is π, then its distribution is invariant in time.

Example 4.1.14. Consider the 2-state Markov chain $(X_t)_{t\geq 0}$ with transition matrix (as in Exercise 4.1.9)

$$P = \begin{bmatrix} 0.2 & 0.8 \\ 0.6 & 0.4 \end{bmatrix}.$$

Then $\pi = [3/7, 4/7]$ is a stationary distribution of X_t. Indeed,

$$[3/7, 4/7] = [3/7, 4/7] \begin{bmatrix} 0.2 & 0.8 \\ 0.6 & 0.4 \end{bmatrix}.$$

Furthermore, this is the unique stationary distribution. To see this, let $\pi = [\pi_1, \pi_2]$ be a stationary distribution of X_t. Then $\pi = \pi P$ gives

$$\begin{aligned} 0.2\pi_1 + 0.6\pi_2 &= \pi_1 \\ 0.8\pi_1 + 0.4\pi_2 &= \pi_2. \end{aligned}$$

These equations lead to

$$4\pi_1 = 3\pi_2.$$

Since π is a probability distribution, $\pi_1 + \pi_2 = 1$ so $\pi = [3/7, 4/7]$ is the only solution. This shows the uniqueness of the stationary distribution for X_t. ▲

We now show that the probability distribution π of normalized degrees in (14).

Proof of Theorem 4.1.4. What is the stationary distribution of random walk on G? There could be many, but here is a typical one that always works. Let π be the probability distribution on the nodes in (14) given by normalized degrees. We wish to show that $\pi = \pi P$, where P is the transition matrix for the random walk on G as in (15). Indeed, this is a simple calculation:

$$\begin{aligned} \sum_{i\in V} \pi(j) P(j,i) &= \sum_{i\in V} \frac{\deg_G(i)}{2|E|} \frac{A_G(i,j)}{\deg_G(i)} \\ &= \frac{1}{2|E|} \sum_{i\in V} A_G(i,j) = \frac{\deg_G(j)}{2|E|} = \pi(j). \end{aligned}$$

This shows that π is a stationary distribution for the RW on G. The uniqueness of stationary distribution can be shown in multiple ways (e.g., coupling, maximal principle for harmonic functions), but we omit the details here. □

Exercise 4.1.15 (RW on directed graphs)**.** Let $G = (V, E)$, $V = [n]$ be a directed graph such that every node has positive out-degree. (In general, we may a self-loop at each node with no out-neighbors.) RW on G is a sequence of random nodes $(X_t)_{t\geq 0}$ generated as follows:

$$X_{t+1} \,|\, X_t \sim \text{Uniform}(N^+(X_t)),$$

where $N^+(v)$ is the set of all out-neighbors of node v.

(i) Show that RW on G is a Markov chain. Show that its transition matrix P is given by

$$P_{ij} = \frac{\mathbf{1}(i \to j)}{\deg^+(i)} \quad \text{for all } 1 \le i, j \le n,$$

where $\{i \to j\}$ denotes the event that there is a directed edge from i to j and $\deg^+(j)$ denotes the out-degree of i.

(ii) The existence of a stationary distribution for P can be shown by linear algebraic arguments (e.g., Perron-Frobenious theorem for nonnegative matrices, maximum left eigenvalue of a stochastic matrix). Is it the case that such stationary distribution assigns weights proportional to the in- or out-degrees? Construct some examples and test the equations $\pi^{\pm} = \pi^{\pm} P$, where $\pi^{\pm}$ is the normalized out/in degree sequence.

Exercise 4.1.16. Implement RW on a digraph G in Python and produce plots similar to Figures 4.1.4 and 4.1.5. As the underlying directed graph, you may use a randomly oriented version of CALTECH or use any other real-world directed network you can find. You may use the codes in notebook6.

Random walk on disconnected graphs must have multiple (in fact, infinitely many) stationary distribution.

Exercise 4.1.17 (Random walk on disconnected graphs)**.** Let $G = (V, E)$ be a graph with two disjoint components G_1 and G_2. Let $(X_n)_{n\ge 0}$ be a random walk on G and denote by P its transition matrix.

(i) Let P_1 and P_2 be the transition matrices for random walks on G_1 and G_2, respectively. Let $V_1 = \{1, 2, \cdots, n\}$ and $V_2 = \{n+1, n+2, \cdots, n+m\}$ be the set of nodes in G_1 and G_2. Then show that P is of the following block diagonal form

$$P = \begin{bmatrix} P_1 & O \\ O & P_2 \end{bmatrix}.$$

(ii) Let $\pi_1 = [\pi_1^{(1)}, \cdots, \pi_1^{(n)}]$ and $\pi_2 = [\pi_2^{(1)}, \cdots, \pi_2^{m}]$ be any stationary distributions for P_1 and P_2. Let

$$\bar{\pi}_1 = [\pi_1^{(1)}, \cdots \pi_1^{(n)}, 0 \cdots, 0]$$
$$\bar{\pi}_2 = [0, \cdots, 0, \pi_2^{(1)}, \cdots, \pi_2^{m}]$$

be distributions on $V = V_1 \cup V_2 = \{1, 2, \cdots n, n+1, \cdots, n+m\}$. Show that for any $\lambda \in [0, 1]$, $\pi := \lambda\bar{\pi}_1 + (1-\lambda)\bar{\pi}_2$ is a stationary distribution for P.

(iii) Recall that random walk on arbitrary graph has a stationary distribution, where probability of each node is proportional to its degree. Conclude that the random walk $(X_n)_{n\ge 0}$ on G has infinitely many stationary distribution.

4.1.4. PageRank and Centrality. The *PageRank* algorithm, developed by Larry Page and Sergey Brin at Stanford University, is the foundation of Google's search engine ranking system. The primary motivation behind PageRank is to assess the importance of webpages based on their link structure. The basic idea is that *a webpage is more important if it is linked to by other important webpages.*

Here are the key motivations behind the PageRank algorithm:

1. (*Link Structure as a Vote of Confidence*) PageRank views a link from one webpage to another as a vote of confidence or recommendation. If a webpage is linked to by many other pages, it is seen as more important or valuable.
2. (*Webpage Importance*) The algorithm aims to measure the importance of a webpage within the entire web ecosystem. It assumes that more important pages are likely to receive more links from other important pages.

3. (*Avoiding Manipulation*) PageRank was designed to counteract simple manipulations and spam tactics. It is not just about the quantity of links but also considers the quality of the linking pages. A link from a highly ranked page carries more weight than a link from a low-ranked page.
4. (*Recursive Evaluation*) The algorithm evaluates pages recursively. If a page is linked to by other important pages, the importance of that page increases. This recursive nature allows the algorithm to capture the interconnectedness of webpages.
5. (*Ranking Search Results*) PageRank is a critical component of Google's search algorithm. By assigning a numerical weight to each element of a hyperlinked set of documents (webpages), it helps Google determine the relevance and importance of pages when delivering search results.

At the core, PR is a random walk with occational 'teleportation'.

Definition 4.1.18 (PageRank)**.** Let $G = (V, E)$ be a digraph with $V = [n]$. Add self-loops at nodes with out-degree zero. *PageRank* (PR) on G is the sequence of random nodes $(X_t)_{t\geq 0}$ with parameter (damping factor) $\alpha \in [0,1]$ generated by the following rule: For each $t \geq 0$,

$$\begin{cases} U \sim \text{Uniform}([0,1]) & \text{(Independently)} \\ X_{t+1} \,|\, X_t \sim \text{Uniform}(N^+(X_t)) & \text{if } U < \alpha \\ X_{t+1} \,|\, X_t \sim \text{Uniform}(V) & \text{if } U \geq \alpha, \end{cases}$$

where $N^+(v)$ denotes the set of all out-neighbors of node v. In words, at each step, with probability α, X_{t+1} is sampled uniformly at random among the neighbors of X_t (as in RW on G), and with probability $1-\alpha$, X_{t+1} is teleported to a uniformly chosen node in G. The *m-step PR vector* is the m-step empirical frequency vector:

$$\mathbf{PR}^{(m)} := (\text{PR}^{(m)}(1), \ldots, \text{PR}^{(m)}(n)),$$

where $\text{PR}^{(m)}(i) := \frac{1}{m}\sum_{t=1}^{m} \mathbf{1}(X_t = i)$. The entries of the vector $\mathbf{PR}^{(m)}$ is also called the *PageRank centrality*. (See Figure 4.1.11 for python implementation.)

PR gives a model of a random surfer who reaches their target site after several clicks, then switches to a random page. One can run PR on a temporaly evolving directed graph (like the Web graph) and continuously and recursively maintain the most up-to-date PR vector. It gives the empirical frequency of visiting a node till now, which can be used to rank the nodes. This gives the notion of *PageRank centrality*, which a measure of the importance or centrality of a node within a network, based on the PageRank algorithm (Def. 4.1.18). Originally developed by Larry Page and Sergey Brin as part of the Google search engine algorithm, PageRank centrality has found applications in various network analysis contexts.

RW on a directed graph can get stuck unless the graph is strongly connected (see Def. 2.3.4). There is no guarantee that the Web graph is strongly connected at any given time. One of the advantages of PR over RW is that it always 'converges' regardless of the underlying graph being strongly connected. This is due to the random teleportation in PR, which is abstent in RW.

Theorem 4.1.19 (Convergence of empirical PR vector)**.** *Let $G = (V, E)$, $V = [n]$ be any (fixed) directed graph and let $(X_t)_{t\geq 0}$ be PR on G with parameter $\alpha \in (0,1)$. Then there exists a unique probability distribution $\mathbf{PR} = (\text{PR}(1), \ldots, \text{PR}(n))$ on V such that*

$$\lim_{m\to\infty} \mathbf{PR}^{(m)} \to \mathbf{PR}. \tag{16}$$

Furthermore, $\mathbf{PR}$ is the unique stationary distribution for PR.

PROOF. Omitted. □

```python
def PR(G, x0=None, steps=1, return_history=False, alpha=0):
    from tqdm import trange
    # PageRank on digraph G
    # initialization at x0
    # alpha = PageRank parameter; probability of making RW move; 1-alpha is the probability of teleportation.
    if x0 is None:
        x = np.random.choice(G.nodes())
    else:
        x = x0

    history = []
    for i in trange(steps):
        U = np.random.rand()
        if U>alpha:
            x = np.random.choice(G.nodes()) # re-initialize uniformly at random
        elif (len(list(G.neighbors(x))) == 0):
            #print("RW is stuck at isolated")
            x = x
        else:
            x = np.random.choice(list(G.neighbors(x))) # DiGraph.neighbors gives out-neighbors

        if return_history:
            history.append(x)

    if not return_history:
        return x
    else:
        return history
```

FIGURE 4.1.11. A Python implementation of PageRank on G.

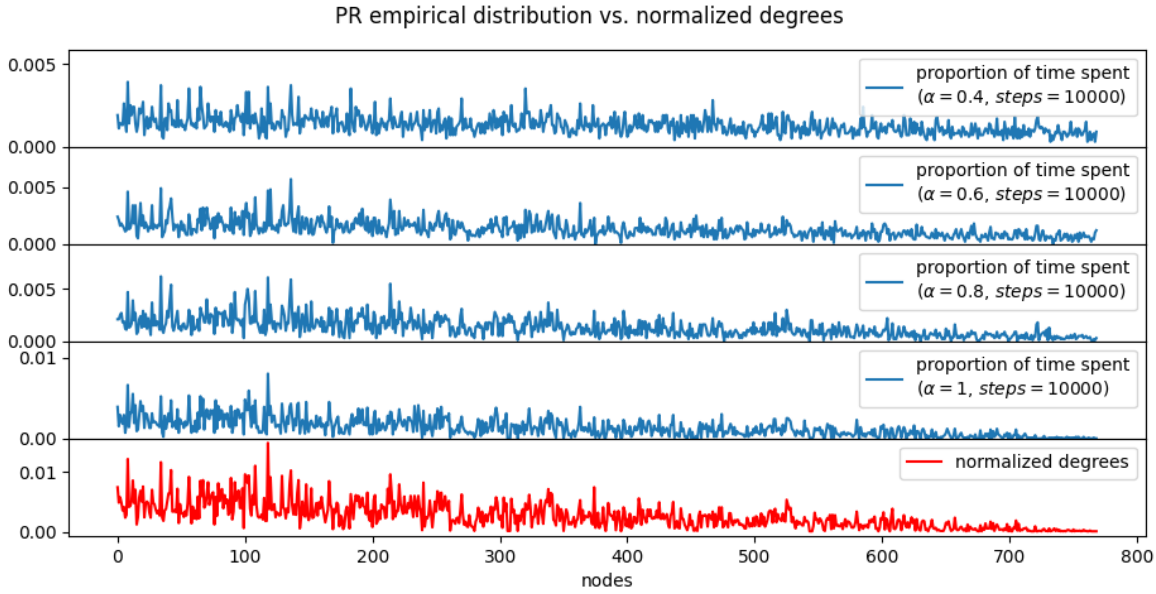


FIGURE 4.1.12. Normalized proportion of times that PR on CALTECH spends at each node for $N = 10000$ steps with $\alpha \in \{0.4, 0.6, 0.8, 1\}$ (top rows) and the normalized degrees $\deg(v)/2|E|$ (bottom).

In Exercise 4.1.20, we show that PR is indeed a Markov chain and its transition matrix is a convex combination between the RW transition matrix and the 'constant transition matrix' for sampling uniformly random nodes.

Exercise 4.1.20 (PageRank as a Markov chain)**.** Let $G = (V, E)$, $V = [n]$ be any (fixed) directed graph where every node has positive out-degree. Let $(X_t)_{t\geq 0}$ be PR on G with parameter $\alpha \in (0, 1)$.

(i) Show that PR is a Markov chain.

(ii) Let $(Y_t)_{t\geq 0}$ be an i.i.d. sequence of uniformly random nodes in G. Show that it is a Markov chain with transition matrix $n^{-1}\mathbf{1}\mathbf{1}^T$, where $\mathbf{1} = (1, \ldots, 1)^T \in \mathbb{R}^n$.

(iii) Show that the transition matrix Q of PR is given by

$$Q = \alpha P + (1-\alpha)n^{-1}\mathbf{1}\mathbf{1}^T,$$

where P denotes the transition matrix of RW on G. (Hence PR is a convex combination of RW and uniform sampling.)

(iv) Show that PR is an 'irreducible' Markov chain, by showing that

$$\mathbb{P}(X_{t+1} = v \mid X_t = u) \geq (1-\alpha)n^{-1} \quad \text{for all } u, v \in V.$$

That is, with probability at least α/n, PR can jump from any node to any other node.

As with the empirical distribution, the multi-step distributions also converge to the same limiting PageRank distribution **PR**. This result is stated in Theorem 4.1.21, which is empirically validated in Figure 4.1.13.

Theorem 4.1.21 (Convergence of multi-step distribution of PR on graphs)**.** *Let $(X_t)_{t\geq 0}$ be a PR on a graph $G = (V, E)$ (not necessarily non-bipartite and connected) with damping factor $\alpha \in (0,1)$. Then for each $x, y \in V$,*

$$\lim_{t\to\infty} \mathbb{P}(X_t = y \mid X_0 = x) = \mathbf{PR}(v),$$

where **PR** *is the limiting PageRank distribution in* (16)*.*

PROOF. Omitted. □

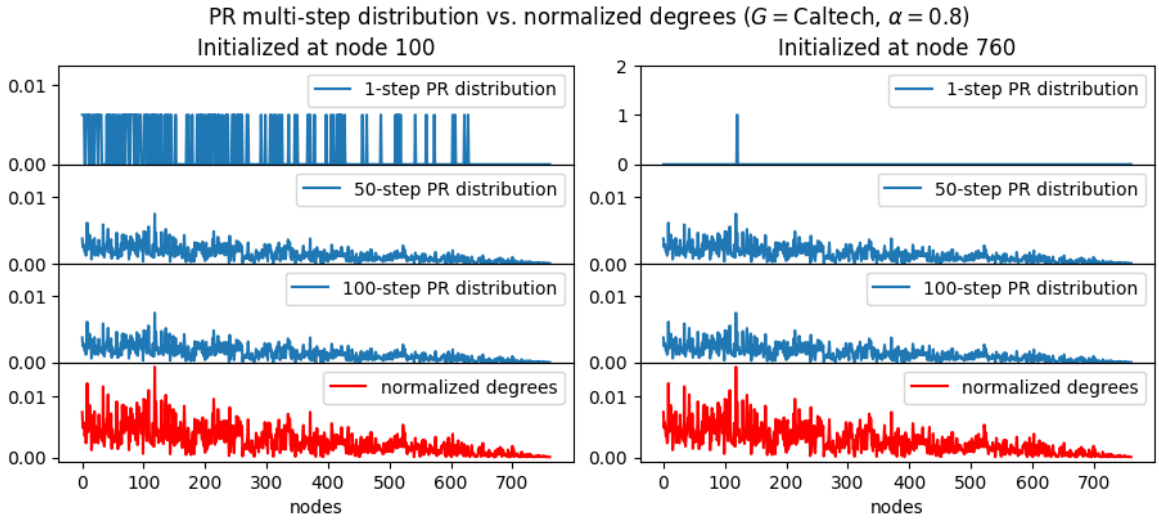


FIGURE 4.1.13. Comparison of multi-step distribution of PR with $\alpha = 0.8$ on CALTECH with two initializations. Compare with Figure 4.1.5.

Example 4.1.22 (Justin Bieber Retweet network)**.** Here wse use PageRank to rank the users in a retweet network `RETWEET` containing the keyword "Justin Bieber". This is a directed temporal network with 10,000 nodes (users) and 90,000 directed temporal edges (retweets) acquired from Network Data Repository[2]. Each temporal edge is a triple $(\texttt{tail}, \texttt{head}, \texttt{timestamp}) = (u, v, t)$, indicating that user u is retweeted by user v at time t. For this experiment, we disgard the timestamps so that the existence of directed edge (u, v) indicates that user u is ever retweeted by user v. The resulting digraph is shown in Figure 4.1.14, which contains a large number of small connected components.

[2]This is NOT the original source of the data. It is just a repository of network data that curates from various first sources.

Figure 4.1.15 shows 10,000-step PR vectors (empirical frequencies) for various damping parameters α. Running a RW (PR with $\alpha = 1$) on the digraph RETWEET will get stuck at some node with no out-edges. Consequently, the 10,000-step empirical frequency vector is concentrated at one node (see Fig. 4.1.15 $\alpha = 1$.) For $\alpha \in \{0.4, 0.6, 0.8\}$, the PR vector is spread out over all nodes, but there are a few nodes that gets noticibly large PR centrality. For instance, the first node has extremely large PR centrality. From the normalized degree distribution in Fig, 4.1.15 we see that the first node has large out-degree, meaning that there are many users that retweet the first user. However, PR centrality is not directly related to the degrees, as can be seen from other large-degree nodes (index around 3500 and 6700) getting small PR centrality.

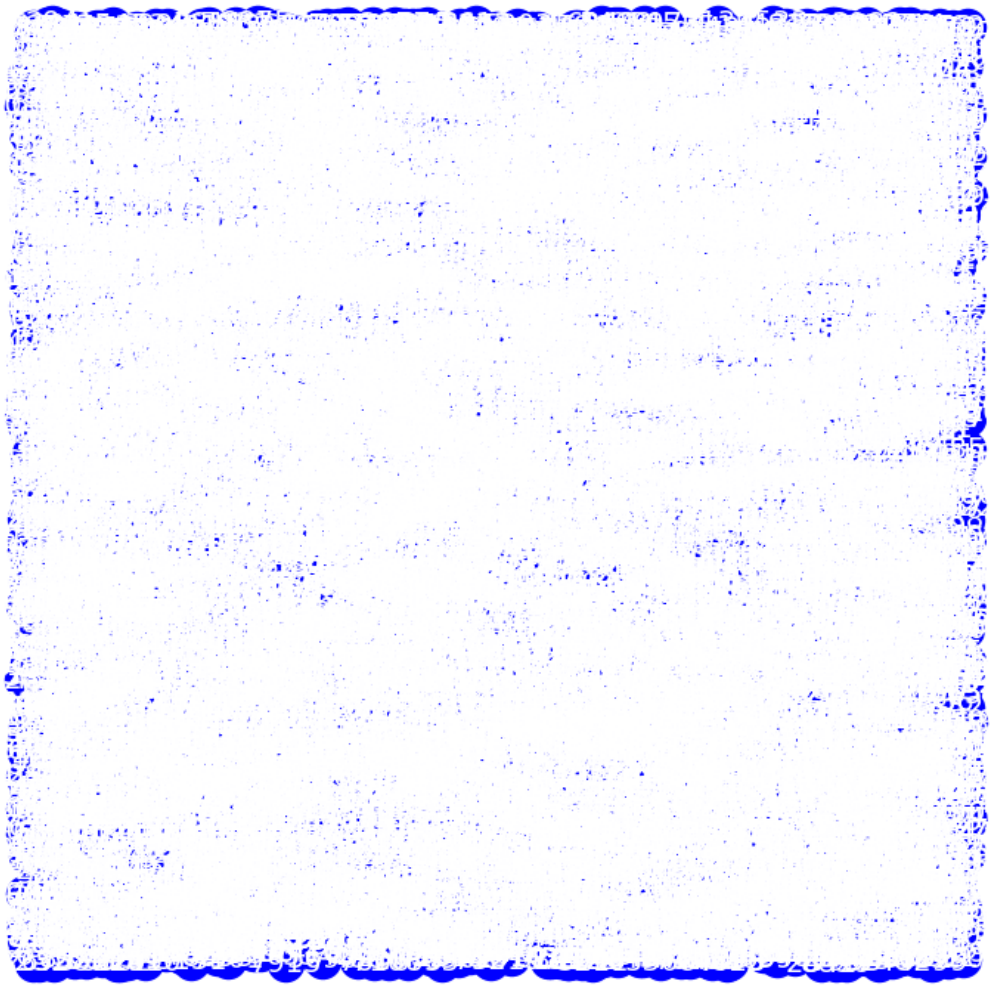


FIGURE 4.1.14. Plot of Justin Bieber retweet network. It has 9,000 nodes and 10,000 directed (and temporal) edges. A directed edge (i, j) represent user i getting retweeted by user j some time.

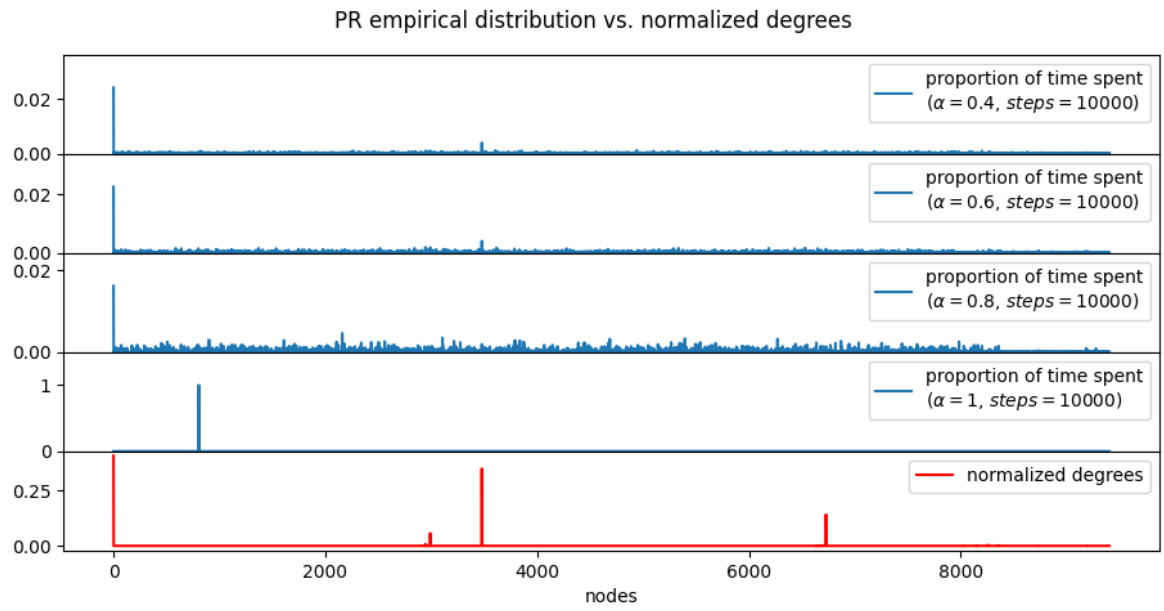


FIGURE 4.1.15. Normalized proportion of times that PR on `RETWEET` spends at each node for $N = 10000$ steps with $\alpha \in \{0.4, 0.6, 0.8, 1\}$ (top rows) and the normalized degrees $\deg(v)/2|E|$ (bottom).

4.2. Classical algorithms on networks

In this section, we study some fundamental algorithms on networks for network search, building spanning trees, and computing shortest path distances, etc. A useful source of popular graph algorithms is here.

4.2.1. Breath-first-search and Depth-first-search. Breadth-First Search (BFS) is a fundamental algorithm used in graph theory to explore and traverse graph structures. It systematically visits all the vertices of a graph in breadthward motion, starting from a designated source vertex and moving outward through the graph level by level. The key idea is to explore all the neighbors of a vertex before moving on to the next level.

Here's a step-by-step explanation of the Breadth-First Search (BFS) algorithm:

1. (*Initialization*)
- **(i)** Mark all vertices as `unvisited`.
- **(ii)** Create a queue data structure and enqueue the source vertex.
- **(iii)** Mark the source vertex as `visited`.

2. (*BFS iteration*) While the queue is not empty:
- **(i)** Dequeue a vertex from the front of the queue. This vertex is the current vertex.
- **(ii)** Process the current vertex (print, store, or perform any desired operation).
- **(iii)** Enqueue all `unvisited` neighbors of the current vertex.
- **(iv)** Mark each visited neighbor as `visited` and enqueue it.

3. (*Termination*) The algorithm continues until the queue becomes empty, meaning all reachable vertices have been visited.

The BFS algorithm ensures that vertices closer to the source vertex are explored before moving on to vertices that are farther away. This property makes BFS useful for various applications, including finding the shortest path between two vertices in an unweighted graph, checking for connectivity, and discovering the structure of a graph.

The time complexity of BFS is generally $O(|V| + |E|)$. The space complexity is also $O(|V|)$ due to the queue data structure.

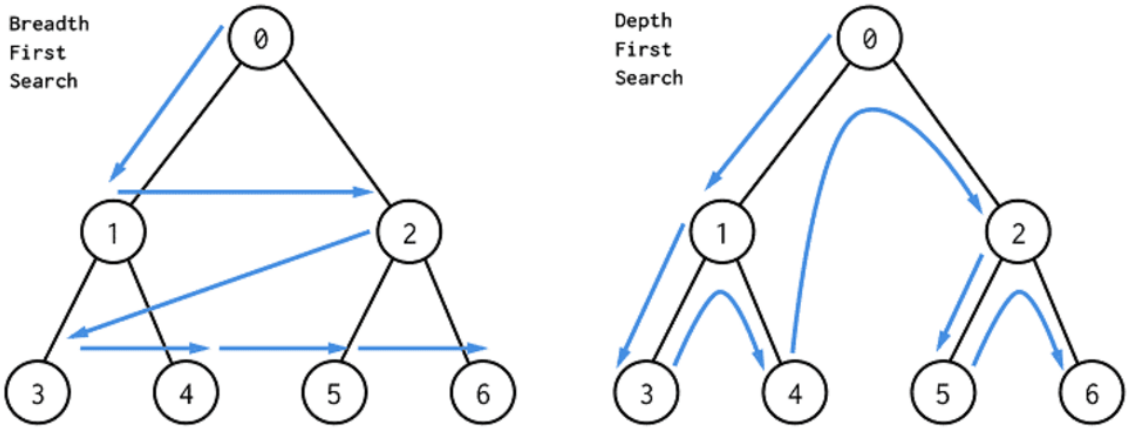


FIGURE 4.2.1. Illustration of breath-first-search (BFS) and depth-first-search (DFS) algorithms. Image credit: source

```python
def DFS(G, s, visited=None):  #function for DFS
    if visited is None:
        visited = []
    ### Depth-first-search
    if s not in visited:
        visited.append(s)
        for u in list(G.neighbors(s)):
            visited = DFS(G, u, visited=visited) # recursive call of DFS
    return visited
```

```python
def BFS(G, s): #function for BFS
    queue = []
    visited = []
    visited.append(s)
    queue.append(s)
    while len(queue)>0:   # Creating loop to visit each node
        v = queue.pop(0)  # first remaining entry in queue
        for u in G.neighbors(v):
            if u not in visited:
                visited.append(u)
                queue.append(u)
    return visited
```

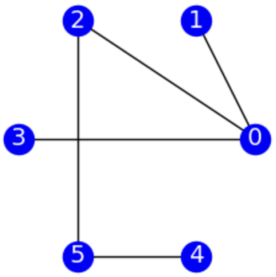
FIGURE 4.2.2. Python implementation of BFS and DFS.

Depth-First Search (DFS) is another fundamental algorithm used in graph theory to explore and traverse graph structures. Unlike Breadth-First Search (BFS), DFS explores as far as possible along each branch before backtracking. It plunges as deeply as possible along each branch before backing up, which is why it's called "depth-first."

Here's a step-by-step explanation of the Depth-First Search (DFS) algorithm:

1. (*Initialization*)
- **(i)** Mark all vertices as unvisited.
- **(ii)** Choose a starting vertex (source) and mark it as visited.
- **(iii)** Mark the source vertex as visited.

2. (*DFS iteration*) While the queue is not empty:
- **(i)** Explore a vertex.




```
DFS(G, 0)
```
```
[0, 1, 2, 5, 4, 3]
```
```
BFS(G, 0)
```
```
[0, 1, 2, 3, 5, 4]
```

FIGURE 4.2.3. An example of BFS and DFS on a graph.

(ii) For each unvisited neighbor of the current vertex, perform DFS on the neighbor (recursively). This ensures that the algorithm goes as deep as possible along each branch before backtracking.

3. (*Termination*) The algorithm continues until all reachable vertices are visited.

DFS can also be implemented using a stack to manage the backtracking explicitly. The recursive approach is often more concise and easier to understand. DFS is used for various applications, including topological sorting, finding connected components in a graph, and solving problems related to cycles in a graph.

The time complexity of DFS is $O(|V|+|E|)$. The space complexity depends on the implementation, with the recursive version typically having a space complexity of $O(h)$, where h is the maximum depth of the recursion. The stack-based version has a space complexity of $O(V)$ when using an explicit stack.

Example 4.2.1 (Connected components of a graph)**.** We can use BFS and DFS algorithms to compute all connected components of a graph (and also the connected component containing a given node). Simply observe that when we call BFS(G,v) or DFS(G,v) in Figure 4.2.2, we get all nodes in the connected component of G containing v. Therefore, we simply need to call these functions recursively until we exhaust the node set of G. See Figure 4.2.4 for a Python implementation of such function using DFS.

```python
def compute_connected_components(G):
    ### compute all connected components of G
    ### output = dictionary of {component index : nodes in the component}
    nodes = list(G.nodes())
    connected_components = {}
    remaining_nodes = list(G.nodes())
    i = 0
    while len(remaining_nodes) > 0 :
        comp = DFS(G,remaining_nodes[0])
        connected_components.update({i: comp.copy()})
        remaining_nodes = [v for v in remaining_nodes if v not in comp]
        i += 1
    for i in connected_components.keys():
        print("Size of the {}th component = {}".format(i, len(connected_components.get(i))))
    return connected_components
```

FIGURE 4.2.4. Python implementation of a function computing all connected components of G using DFS.

To illustrate its use, recall that CALTECH contains four components of sizes 762, 3, 2, and 2. The original graph of CALTCH and its connected components computed by DFS as in Figure 4.2.4 is shown in Figure 4.2.5. ■

Exercise 4.2.2. Implement a function that computes connected components of a graph using BFS. Plot and compare the connected components computed by BFS and DFS. Is there any difference?

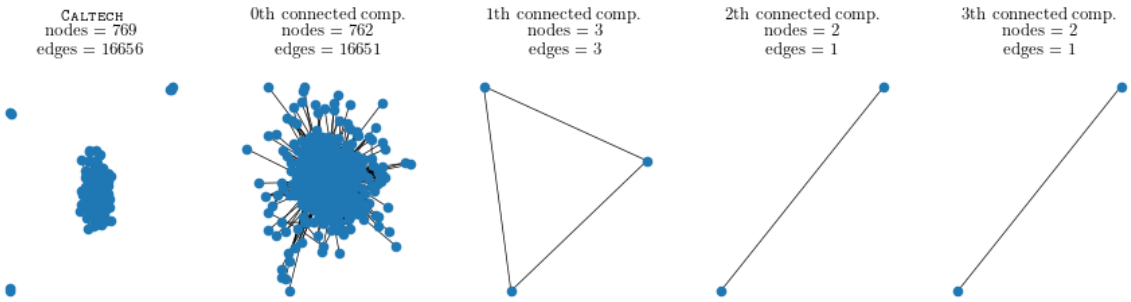


FIGURE 4.2.5. An example of using DFS to compute all connected components in CALTECH.

Example 4.2.3 (Computing spanning trees)**.** Recall that a subgraph T of a graph G is a 'spanning tree' of G if it is a tree and contains all nodes of G. BFS and DFS algorithms can naturally be used to compute spanning trees, although the spanning trees computed by these two algorithms may look very different, due to the nature of these search algorithms. Roughly speaking, BFS-spanning trees are rounded and DFS-spanning trees are elongated. See Figure 4.2.6 for G = the largest connected component of CALTECH. See Figure 4.2.7 for python implementations.

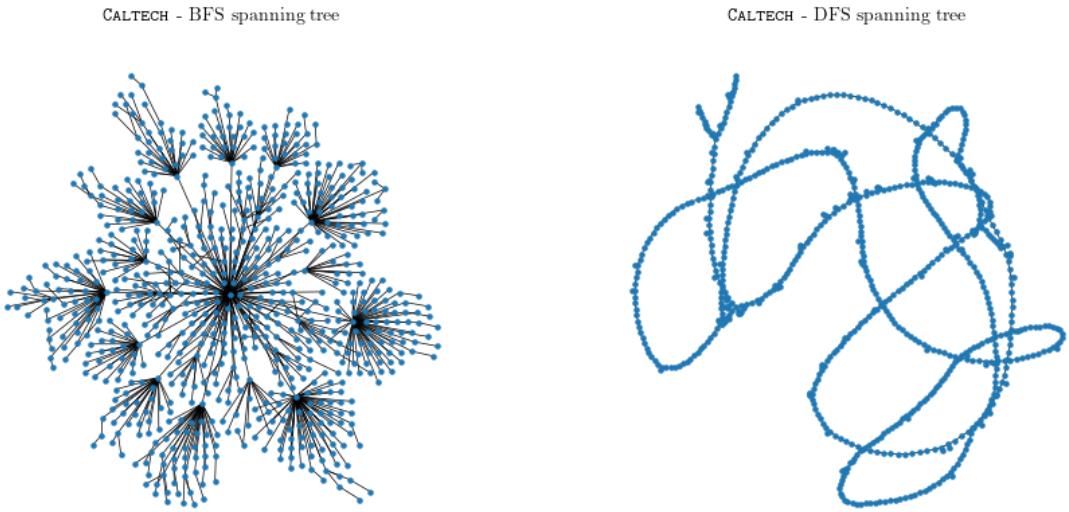


FIGURE 4.2.6. Spanning trees of CALTECH computed by BFS (left) and DFS (right).

Exercise 4.2.4. Implement a function that computes all triangles in a graph using BFS and DFS. Compare your code with `nx.triangles`.

4.2.2. Min-cost trees. In Section 4.2.1, we have constructed algorithms that compute spanning trees of a given connected graph, by using BFS and DFS. In many applications, there are costs associated to the edges when selecting to from spanning trees. In this case, we would like to find a spanning tree that consists of the smallest possible total weights. Such a spanning tree is called the 'min-cost tree' or 'minimum spanning tree' (MST).

Definition 4.2.5 (Minimum spanning tree)**.** Let G be a finite connected graph and let $\phi : E(G) \to [0,\infty)$ be edge weights. A spanning tree T of G is called a (ϕ-)*minimum spanning tree* (MST) of G if it minimizes the total cost $\sum_{e\in E(T)} \phi(e)$:

$$T \in \operatorname*{arg\,min}_{T\subseteq G;\text{spanning tree}} \sum_{e\in E(T)} \phi(e)$$

```python
def Spanning_tree_BFS(G, s): #function for BFS
    queue = []
    visited = []
    edges_T = []
    visited.append(s)
    queue.append(s)
    while len(queue)>0:   # Creating loop to visit each node
        v = queue.pop(0)  # first remaining entry in queue
        for u in G.neighbors(v):
            if u not in visited:
                visited.append(u)
                queue.append(u)
                edges_T.append([u,v])
    return visited, edges_T
```

```python
def Spanning_tree_DFS(G, s, visited=None, edges_T=None):  #function for DFS
    if visited is None:
        visited = []
        edges_T = []
    if s not in visited:
        visited.append(s)
        for u in list(G.neighbors(s)):
            if u not in visited:
                edges_T.append([s,u])
            visited, edges_T = Spanning_tree_DFS(G, u, visited=visited, edges_T=edges_T) # recursive call of DFS
    return visited, edges_T
```

FIGURE 4.2.7. Python implementation of computing spanning trees using BFS and DFS.

Note that MST always exists since there are at least one and only finitely many spanning trees of a finite connected graph G.

Exercise 4.2.6. Let T be a tree and let u, v be two distinct nodes of T. Show that there exists a unique path between u and v.

Definition 4.2.7 (Fundamental cycle)**.** Let G be a graph and let T be a spanning tree of G. For each edge $f = uv \in E(G \setminus T)$, the *fundamental cycle* C of f with respect to T is the unique cycle in G that consists of f and the unique (u, v)-path in T.

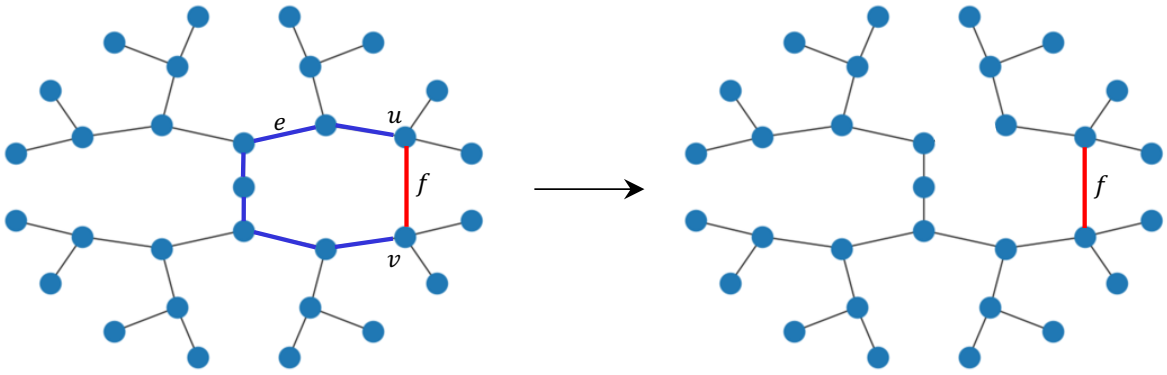


FIGURE 4.2.8. Switching edges e and f to obtain an alternative spanning tree.

The following observation is crucial in verifying correctness Kruskal's MST algorithm (Def. 4.2.10).

Proposition 4.2.8 (Edge switching and spanning tree)**.** *Let G be a graph and let T be a spanning tree of G. Suppose f is an edge in $G \setminus T$ and let C be the fundamental cycle of f. For any edge e in C, $T - e + f$ is a spanning tree of G.*

PROOF. Let $f = uv$. If $f = e$, there is nothing to show. Suppose $f \neq e$. Suppose for a contradiction that $T' := T - e + f$ contains a cycle, say, C'. Then C' must contain f since otherwise $C' \subseteq T$, which contradicts to the fact that T is a tree. By traversing C', we find a (u, v)-path P' in T not

containing f. Then $P' \subseteq T - e$, so P' is a (u,v)-path not containing e. But this is a contradiction since there is a unique (u,v)-path in T and it contains e. (See Figure 4.2.8.) □

Proposition 4.2.9. *Let G, ϕ be as above, and let T be a MST. Let $f \in E(G \setminus T)$, and let e be an edge of the fundamental cicle of f with respect to T. Then $\phi(e) \leq \phi(f)$.*

PROOF. If $\phi(e) > \phi(f)$, then $T' := T - e + f$ is a spanning tree (by Prop. 4.2.8) and has strictlly smaller total cost than T does, which is a contradiction. Hence it must be that $\phi(e) \leq \phi(f)$. □

Kruskal's algorithm is a greedy algorithm that builds the edge set of a MST recursively. At each step, one needs to add in an edge that does not introduce cycles and has the smallest possible cost. The magic is that this greedy algorithm produceds globally optimal spanning tree, i.e., MST.

Definition 4.2.10 (Kruskal's algorithm for MST)**.** Let $G = (V,E)$ be a connected graph with $|V| = n$ and let $\phi : E \to [0,\infty)$ be a cost function. Recursively build edge set E' as follows:

$$f \leftarrow f \in E \setminus E' \text{ s.t. } E' \cup \{f\} \text{ has no cycle in } G \text{ and } \phi(f) \text{ is as small as possible}; \tag{17}$$
$$E' \leftarrow E' \cup \{f\}.$$

Terminate the algorithm once $|E'| = n-1$. Then the output $E' = \{e_1, \ldots, e_{n-1}\}$ is the edge set of a MST.

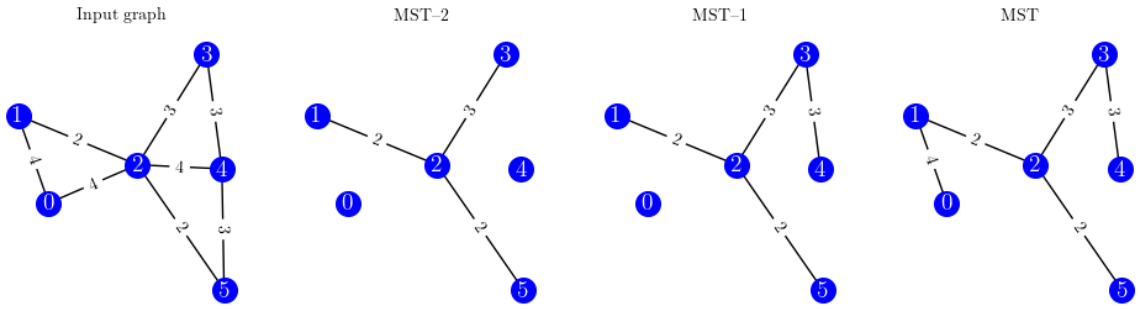


FIGURE 4.2.9. An example of computing MST using Kruskal's algorithm.

There are two points we need to verify for the above algorithm.

1. (*Well-defined?*) Can we execute each step (17) of Kruskal's algorithm until $|E'| = n-1$?

2. (*Correct?*) Is the output $E' = \{e_1, \ldots, e_n\}$ really the edge set of a MST?

Point 1 is easy to see. Suppose $|E'| < n-1$. Then by construction the induced subgraph T' of G with edge set E' is a tree that does not contain all nodes in G. Since G is connected, then there is an edge $f = uv$ such that u is in T' but v is not. Then $T' + f$ is still a tree. Therefore, Kruskal's algorithm is well-defined and it outputs the edge set of a spanning tree of G. Then we only need to check if this spanning tree is a MST.

Theorem 4.2.11 (Correctness of Kruskal's algorithm)**.** *Kruskal's algorihm produces a MST for any n-node connected graph G with cost function ϕ.*

PROOF. See Exc. 4.2.12. □

Exercise 4.2.12 (Correctness of Kruskal's algorithm)**.** In this exercise, we will show that Kruskal's algorihm produces a MST for any n-node connected graph G with cost function ϕ.

(i) Let $E' = \{e_1, \ldots, e_{n-1}\}$ be the edge set generated by Kruskal's algorithm (Def. 4.2.10). Show that output of Kruskal's algorithm is an MST if the following hold:

there exists an MST T with $E(T) = E$.

(*Hint*: You can use the fact that the output of Kruskal's algorithm is a spanning tree. So only needs to show its total cost is the smallest possible.)

(ii) For the first step, argue that that there exists an MST T such that $e_1 \in E(T)$. (*Hint*: Fix an MST T. Suppose e_1 is not in T. Let C be the fundamental cycle of e_1 w.r.t. T, choose an edge f in $C - e_1$, and form a spanning tree $T' := T - f + e_1$. Using the fact that e_1 is the first edge chosen by Kruskal, argue that this is an MST containing e_1.)

(iii) By an induction, suppose there exists an MST T such that $e_1, \dots, e_{i-1} \in E(T)$. Show that there exists another MST T' such that $e_1, \dots, e_i \in E(T')$. (*Hint:* If $e_i \in E(T)$, then we are done with $T' = T$. If $e_i \notin E(T)$, follow the same procedure in the previous part. To do so, let C be the fundamental cycle of e_i w.r.t. T and take and edge f in $C - e_i$. Let $T' = T - f + e_i$. Use Prop. 4.2.8 to deduce $\phi(f) \le \phi(e_i)$. Use the fact that Kruskal chose e_i rather than f to deduce $\phi(f) \ge \phi(e_i)$. Comebine this to deduce that T' is an MST.)

(iv) Complete the induction argument and show the claim in part **(i)** holds.

```python
def Kruskal_MST(wtd_edgelist):
    ### Kruskal's MST algorithm
    ### input = list of weighted edges; nodes need to be indexed by integers
    ## "find_root" and "forest_add_edge" maintain rooted forest structure to efficiently check
    ## whether adding a new edge  introduces a cycle

    def find_root(parent, i):
        if parent[i] == i:
            return i
        return find(parent, parent[i])

    def forest_add_edge(parent, rank, x, y):
        xroot = find_root(parent, x)
        yroot = find_root(parent, y)
        if rank[xroot] < rank[yroot]:
            parent[xroot] = yroot
        elif rank[xroot] > rank[yroot]:
            parent[yroot] = xroot
        else:
            parent[yroot] = xroot
            rank[xroot] += 1
        return parent, rank

    result = []
    i, e = 0, 0
    # Sort weighted edges
    wtd_edgelist = sorted(wtd_edgelist, key=lambda item: item[2])
    parent = []
    rank = []
    nodes = list(set([v[0] for v in wtd_edgelist] + [v[1] for v in wtd_edgelist]))
    # Important for find ft. that nodes in this list is in increasing order
    for node in nodes:
        parent.append(node)
        rank.append(0)
    while e < len(parent) - 1:
        u, v, w = wtd_edgelist[i]
        i = i + 1
        x = find_root(parent, u)
        y = find_root(parent, v)
        if x != y:
            e = e + 1
            result.append([u, v, w])
            parent, rank = forest_add_edge(parent, rank, x, y)

    return result
```

FIGURE 4.2.10. A Python implementation of Kruskal's MST algorithm.

Exercise 4.2.13. Let G be the largest connected component of CALTECH, which is a simple connected graph with 762 nodes and 16651 edges.

(i) Give edge weights independently uniformly at random (i.e., each edge e gets weight $U_e \sim \text{Uniform}([0,1])$ and U_e's are all independent). Compute the MST T_1 of the resulting weighted graph using Kruskal's algorithm.

(ii) For each edge uv in G, give edge weight $\deg(u)\deg(v)$. Compute the MST T_2 of the resulting weighted graph using Kruskal's algorithm.

(iii) Plot the spanning trees T_1 and T_2. (You may use the code in notebook7). Is there any difference? If so, can you explain? How do these spanning trees compare to the BFS and DFS spanning trees in Figure 4.2.6?

CHAPTER 5

Graph representation learning

5.1. Euclidean embedding of graphs

5.1.1. Basics of node embedding. *Node embedding* is a technique that embeds a given graph $G = (V, E)$ into a Euclidean space $\mathbb{R}^d$ so that each node v in G is mapped to a d-dimensional vector $\phi(v) \in \mathbb{R}^d$ and the 'overall structure' of G is maintained in the 'point cloud' $\{\phi(v) : v \in V\} \subseteq \mathbb{R}^d$. In application, it is desired to choose the embedding dimension d to be much less than the number of nodes $|V|$, so that we get a low-dimensional Euclidean representation of G. A graph by itself is a combinatorial object, not Euclidean. So when trying to embed a graph into a low-dimensional Euclidean space, there could be distortion and loss of information. The key is to find the best way to minimize such distortion. The obtained spectral embedding can be used for various tasks, such as clustering or visualization. - In the lower-dimensional space, nodes that are close to each other are likely to be structurally similar in the original graph.

The basic mathematical framework of node embedding is *matrix factorization*. If G is an n-node graph, it can be expressed as certain $n \times n$ matrix X_G without any loss of information (e.g., the adjacency matrix). Then we seek to factorize this $n \times n$ matrix X_G into the product of smaller matrices W and H with $d \leq n$ columns and rows, respectively:

$$\min_{W \in \mathbb{R}^{n \times d}, H \in \mathbb{R}^{d \times n}} \|X_G - WH\|_F^2. \tag{18}$$

While the jth column of X_G is the full n-dimensional representation of the jth node in G, the jth column of H provides a compressed, d-dimensional representation $\phi(j)$ of the same node. Therefore, reading off the columns of H will give us the d-dimensional node embedding. See Figure 5.1.1 for illustration.

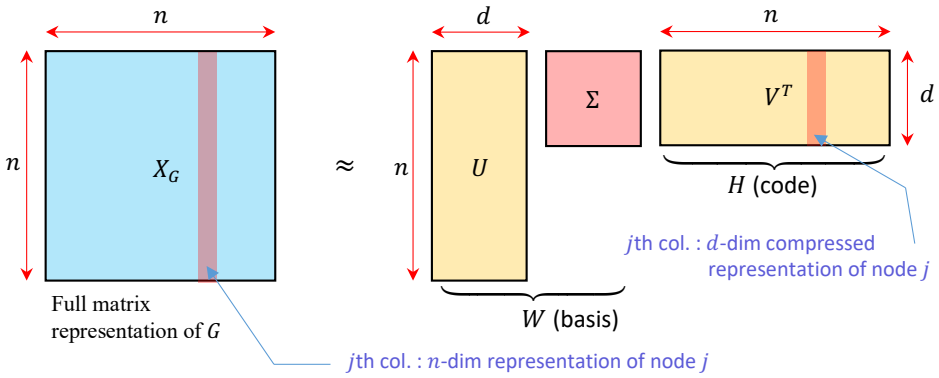


FIGURE 5.1.1. Illustration of node embedding by matrix factorization. Each column of the full matrix representation X_G of G is approximated by a linear combination of the columns in the basis matrix W with coefficients given by the code matrix H. The columns of H give compressed, d-dimensional representation of the ndoes.

The above (18) is a non-convex unconstrained minimization problem, and fortunately there is a good way to write down a globally optimal solution. Namely, use the *singular value decomposition* to write

$$X_G = U\Sigma V^T,$$

where

$$U, V = (n \times n) \text{ orthonormal matrices (i.e., } U^T U = V^T V = I)$$
$$\Sigma = (n \times n) \text{ diagonal matrix of singular values } \sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n \text{ of } X_G.$$

Then by the Eckart-Young-Mirsky theorem, a global optimum of (18) is given whenever

$$WH = U[:, :d]\,\Sigma[:d, :d]\,V[:, :d]^T,$$

where $U[:, :d]$ is the $n \times d$ matrix consisting of the first d columns of U (using Python notation) and $\Sigma[:d, :d]$ is the $d \times d$ diagonal matrix of diagonal entries $\sigma_1 \geq \cdots \geq \sigma_d$. For instance, we can choose

$$W = U[:, :d]\,\Sigma[:d, :d] \in \mathbb{R}^{n \times d}, \qquad H = V[:, :d]^T \in \mathbb{R}^{d \times n}.$$

Then, each column of the full matrix representation X_G of G is approximated by a linear combination of the columns in the basis matrix W with coefficients given by the code matrix H. The columns of H give compressed, d-dimensional representation of the nodes.

In the folloing sections, we will take X_G to be the adjacency matrix or the normalized Laplacian matrix of G and perform node embedding.

5.1.2. Adjacency/Laplacian spectral embedding. Spectral embedding of graphs is a technique used to represent graph data in a lower-dimensional space based on the eigenvalues and eigenvectors of certain matrices associated with the graph. Spectral embedding is particularly useful when dealing with high-dimensional or complex graph-structured data. It has applications in various fields, including machine learning, network analysis, and data visualization. Notably, spectral clustering is a technique that often employs spectral embedding for clustering analysis. The spectral embedding approach provides a way to capture important structural information about the graph in a more compact form, facilitating downstream analysis and interpretation.

The primary matrix X_G used in spectral embedding is the (normalized) *Laplacian matrix*, and the resulting representation is often referred to as a spectral embedding. Given a graph G, its *Laplician matrix* is defined as

$$L_G := D - A,$$

where D is the diagonal matrix of the degrees of the nodes in G and A is the adjacency matrix of G. Note that

$$L_G = D^{1/2}(I - D^{-1/2}AD^{-1/2})D^{1/2}.$$

Accordingly, the *normalized Laplacian matrix* is defined by

$$L'_G := D^{-1/2}L_G D^{-1/2} = I - D^{-1/2}AD^{-1/2}.$$

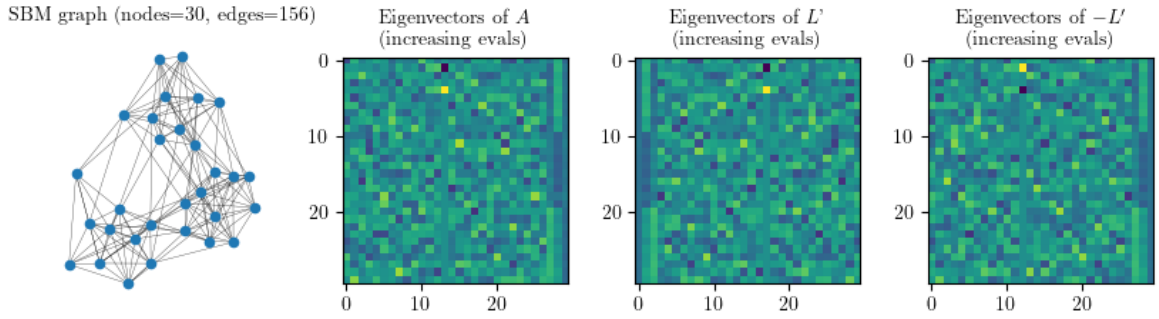


FIGURE 5.1.2. (Left) An SBM graph with 30 nodes, (middle) Eigenvectors (columns) of the adjacency matrix, (right) Eigenvectors of the normalized laplacian matrix (in increasing eigenvalue order).

In many instances of node embedding through the formulation in Figure 5.1.1, the $(n \times n)$ matrix X_G is real and symmetric. Two most frequently used examples are

1. (*Adjacency spectral embedding*): $X_G = A$;

2. (*Laplacian spectral embedding*): $X_G = -L'_G$ (or equivalentely, $X_G = D^{-1/2}AD^{-1/2}$)

In this case, SVD of X_G equals its spectral decomposition, so from the discussion in the previous section ($V = U$ in Figure 5.1.1), we have the following simple spectral embedding pipeline:

(i) Compute the first d eivengectors of X_G corresponding to the top d largest eigenvalues. Call this $n \times d$ matrix U.

(ii) Embed each node i to the d-dimensional vector $U[i,:]$.

In standard Laplacian spectral embedding, one uses *bottom* d eigenvectors of the normalized Laplacian matrix L'_G. This is in fact consistent of our use of top d eigenvectors of $-L'_G$ or $D^{-1/2}AD^{-1/2}$ since

$$\textit{Bottom } d \textit{ eigenvectors of } L'_G \;=\; \textit{Top } d \textit{ eigenvectors of } -L_G \textit{ (or } D^{-1/2}AD^{-1/2}\textit{)}.$$

See Figures 5.1.2 and 5.1.3.

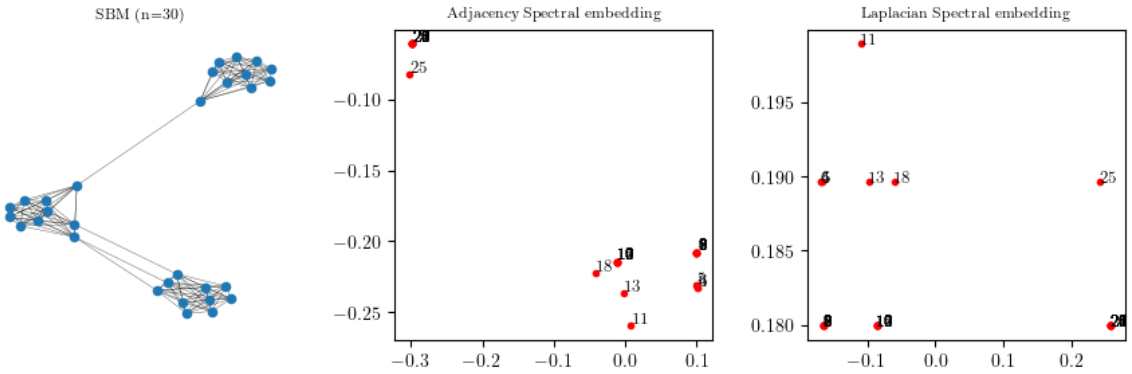


FIGURE 5.1.3. (Left) An SBM graph with 30 nodes, (middle) Adjacency spectral embedding with $d = 2$, (right) Laplacian spectral embedding with $d = 2$.

5.1.3. DeepWalk. DeepWalk is a method for learning continuous representations of nodes in a network, also known as node embeddings, using techniques inspired by natural language processing and word embeddings. It was introduced by Bryan Perozzi, Rami Al-Rfou, and Steven Skiena in their paper "DeepWalk: Online Learning of Social Representations" in 2014 [PARS14].

Here's an overview of the key concepts and steps involved in DeepWalk:

1. (Random walk generation) DeepWalk starts by generating random walks on the graph. Generate n_{RW} many random walk trajectories of length k started at every node v in G. Collect the sampled random walk trajectories (See Fig. 5.1.4 left). This is the "corpus" of "graphical sentences", where nodes are regarded as words and the grammer is that only adjacenct nodes can appear next to each other in each sentence.

2. (Skip-Gram model) The generated random walks are treated as sentences, and the Skip-Gram model (a.k.a. word2vec) from natural language processing [MSC+13] is applied. The Skip-Gram model is a type of neural network that learns to predict the context (neighbor) nodes given a target node (center node) within a sequence of nodes.

3. (Embedding Learning) The Skip-Gram model is trained to maximize the likelihood of observing the context nodes given the target nodes in the random walks. This process results in learning distributed representations (embeddings) for each node in the graph.

RW sequences

```
[['436', '1', '545', '1', '9'],
 ['259', '1112', '1102', '1112', '1340'],
 ['259', '752', '15', '9', '259'],
 ['752', '259', '752', '15', '401'],
 ['15', '737', '1117', '612', '1756'],
 ['9', '1', '545', '1', '436'],
 ['259', '15', '436', '259', '269'],
 ['259', '15', '243', '1007', '1026'],
 ['464', '4', '602', '566', '467'],
 ['464', '484', '368', '553', '602'],
 ['634', '1460', '2405', '1460', '1138'],
 ['634', '198', '634', '198', '296'],
 ['334', '566', '1958', '566', '622'],
 ['4', '602', '4', '464', '565'],
 ['566', '1214', '566', '2630', '2537'],
 ['612', '1756', '612', '180', '332'],
 ['171', '5', '171', '5', '171'],
                  ⋮
```

DeepWalk node embedding

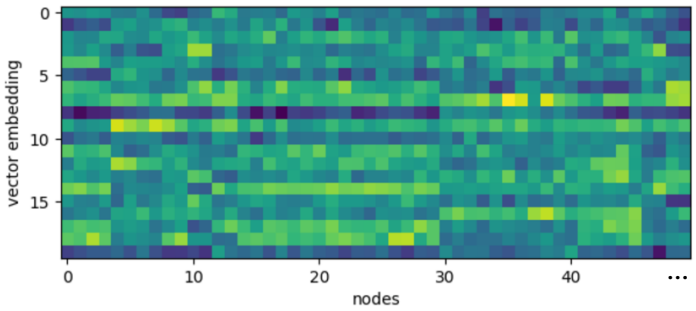


FIGURE 5.1.4. Example of RW paths of length 5 on CORA and 20-dimensional DeepWalk embedding.

4. (Mapping Nodes to Vectors) After training, the weights of the neural network's hidden layer are used as the embedding vectors for the nodes. These vectors capture the structural information and relationships between nodes in the graph. (See Fig. 5.1.4 right.)
5. (Applications) The learned node embeddings can be used for various downstream tasks such as node classification, link prediction, and community detection. The idea is that nodes with similar roles or structural positions in the graph will have similar embeddings.

DeepWalk is particularly effective in capturing the structural properties of graphs, especially in the context of social networks, citation networks, and other relational data. It leverages the idea that nodes with similar network neighborhoods should have similar representations in the embedding space. DeepWalk and similar methods, such as node2vec, have been influential in the development of graph representation learning techniques, contributing to the broader field of network embedding methods.

Example 5.1.1. Below, we will demonstrate the use of DeepWalk node embeding for a particular node classification problem. The dataset we are going to be using is called CORA [MNRS00]. It consists of 2708 scientific publications classified into one of seven classes. The citation network consists of 5429 links. Each publication in the dataset is described by a 0/1-valued word vector indicating the absence/presence of the corresponding word from the dictionary. The dictionary consists of 1433 unique words.

We compute 20-dimensional node embedding in three methods – Adjacency/Laplacian spectral embedding and DeepWalk. Once we assign feature vectors to each node, we then perform multi-class classification on a 80/20 train/test split dataset using multinomial logistic classifier. (See notebook8) for implementation details). Figure 5.1.6 shows the result of multi-class classification. The results are shown as the 7×7 confusion matrix. Each (i, j) entry of this matrix represent the number of instances of true label i gets classified into class j by the algorithm. Hence, a perfect algorithm will result in a diagonal confusion matrix. The accuracy is computed by the total number of correctly classified instances divided by the total number of instances[1].

In Figure 5.1.6, we see the DeepWalk encoding shows the highest classificaiton accuracy of around 69% when we use RW walk length 20. The confusion matrix for that instance is quite close to a diagonal matrix. Spectral embedding apparently collapses the important information in that the logistic classifier classifies every test example into a single class. This indirectly shows that DeepWalk node embeding better preserves structural information for the nodes that are critical for node classification tasks. ▲

[1]This is also the trace of the confusion matrix normalized by the total sum of its entries.

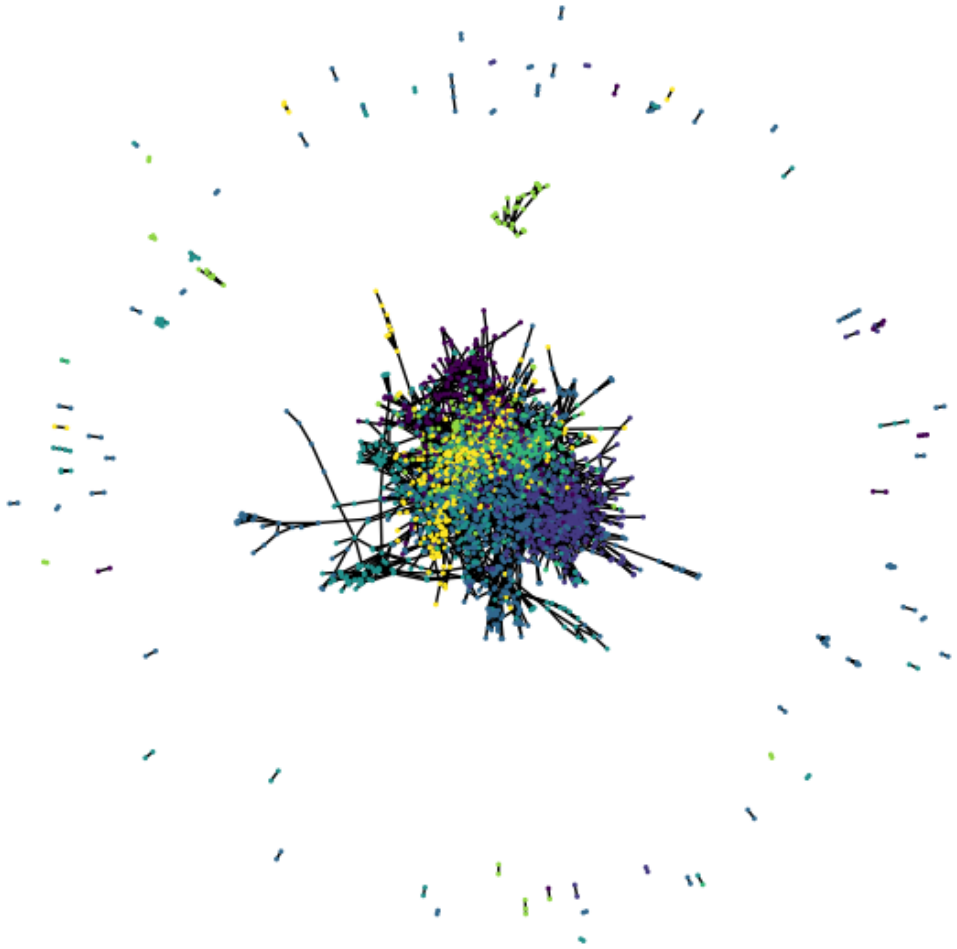


FIGURE 5.1.5. Plot of `CORA` graph. There are 2708 nodes 5429 edges. Each node belongs to one of the seven classes, which are shown by node colors.

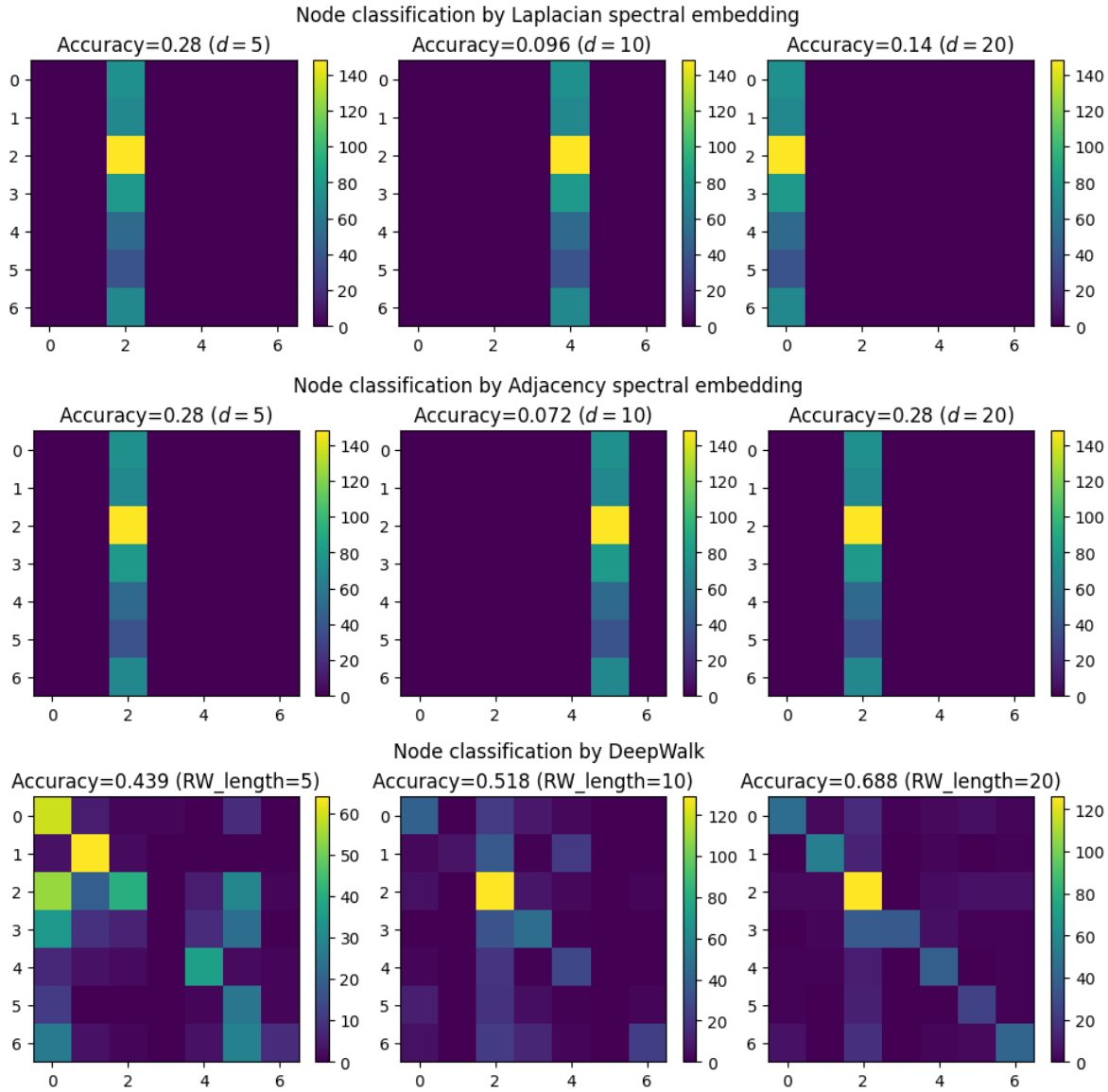


FIGURE 5.1.6. Configuration matrices and accuracies of node classification on `CORA` dataset using DeepWalk and Multinomial Logistic Classifier.

APPENDIX A

Background in probability theory

A.1. Discrete random variables

Given a finite probability space $(\Omega, \mathbb{P})$, a (discrete) *random variable* (RV) is any real-valued function $X : \Omega \to \mathbb{R}$. We can think of it as the outcome of some experiment on Ω (e.g., height of a randomly selected friend). We often forget the original probability space and specify a RV by *probability mass function* (PMF) $f_X : \mathbb{R} \to [0,1]$,

$$f_X(x) = \mathbb{P}(X = x) = \mathbb{P}(\{\omega \in \Omega \mid X(\omega) = x\}).$$

Namely, $\mathbb{P}(X = x)$ is the likelihood that the RV X takes value x.

Example A.1.1. Say you win \$1 if a fair coin lands heads and lose \$1 if lands tails. We can set up our probability space $(\Omega, \mathbb{P})$ by $\Omega = \{H, T\}$ and $\mathbb{P}$ = uniform probability measure on Ω. The RV $X : \Omega \to \mathbb{R}$ for this game is $X(H) = 1$ and $X(T) = -1$. The PMF of X is given by $f_X(1) = \mathbb{P}(X = 1) = \mathbb{P}(\{H\}) = 1/2$ and likewise $f_X(-1) = 1/2$.

Exercise A.1.2. Let $(\Omega, \mathbb{P})$ be a probability space and $X : \Omega \to \mathbb{R}$ be a RV. Let f_X be the PMF of X, that is, $f_X(x) = \mathbb{P}(X = x)$ for all x. Show that f_X adds up to 1, that is,

$$\sum_x f_X(x) = 1,$$

where the summation runs over all numerical values x that X can take.

There are two useful statistics of a RV to summarize its two most important properties: Its average and uncertainty. First, if one has to guess the value of a RV X, what would be the best choice? It is the *expectation* (or mean) of X, defined as below:

$$\mathbb{E}(X) = \sum_x x\,\mathbb{P}(X = x).$$

Exercise A.1.3. For any RV X and real numbers $a, b \in \mathbb{R}$, show that

$$\mathbb{E}[aX + b] = a\mathbb{E}[X] + b.$$

Exercise A.1.4 (Tail sum formula for expectation)**.** Let X be a RV taking values on positive integers.

(i) For any x, show that

$$\mathbb{P}(X \geq x) = \sum_{y=x}^{\infty} \mathbb{P}(X = y).$$

(ii) Use (i) and Fubini's theorem to show

$$\sum_{x=1}^{\infty} \mathbb{P}(X \geq x) = \sum_{y=1}^{\infty} \sum_{x=1}^{y} \mathbb{P}(X = y)$$

(iii) From (ii), conclude that

$$\mathbb{E}(X) = \sum_{x=1}^{\infty} \mathbb{P}(X \geq x).$$

On the other hand, say you play two different games where in the first game, you win or lose \$1 depending on a fair coin flip, and in the second game, you win or lose \$10. In both games, your expected winning is 0. But the two games are different in how much the outcome fluctuates around the mean. This notion if fluctuation is captured by the following quantity called *variance*:

$$\mathrm{Var}(X) = \mathbb{E}[(X - \mathbb{E}(X))^2].$$

Namely, it is the expected squared difference between X and its expectation $\mathbb{E}(X)$.

Here are some of the simplest and yet most important RVs.

Exercise A.1.5 (Bernoulli RV)**.** A RV X is a *Bernoulli* variable with (success) probability $p \in [0,1]$ if it takes value 1 with probability p and 0 with probability $1-p$. In this case we write $X \sim \mathrm{Bernoulli}(p)$. Show that $\mathbb{E}(X) = p$ and $\mathrm{Var}(X) = p(1-p)$.

Exercise A.1.6 (Indicator variables)**.** Let $(\Omega, \mathbb{P})$ be a probability space and let $E \subseteq \Omega$ be an event. The *indicator variable* of the event E, which is denoted by $\mathbf{1}_E$, is the RV such that $\mathbf{1}_E(\omega) = 1$ if $\omega \in E$ and $\mathbf{1}_E(\omega) = 0$ if $\omega \in E^c$. Show that $\mathbf{1}_E$ is a Bernoulli variable with success probability $p = \mathbb{P}(E)$.

The following exercise ties the expectation and the variance of a RV into a problem of finding a point estimator that minimizes the mean squared error.

Exercise A.1.7 (Variance as minimum MSE)**.** Let X be a RV. Let $\hat{x} \in \mathbb{R}$ be a number, which we consider as a 'guess' (or 'estimator' in Statistics) of X. Let $\mathbb{E}[(X-\hat{x})^2]$ be the *mean squared error* (MSE) of this estimation.

(i) Show that

$$\begin{aligned}\mathbb{E}[(X-\hat{x})^2] &= \mathbb{E}[X^2] - 2\hat{x}\mathbb{E}[X] + \hat{x}^2 \\ &= (\hat{x} - \mathbb{E}[X])^2 + \mathbb{E}[X^2] - \mathbb{E}[X]^2 \\ &= (\hat{x} - \mathbb{E}[X])^2 + \mathrm{Var}(X).\end{aligned}$$

(ii) Conclude that the MSE is minimized when $\hat{x} = \mathbb{E}[X]$ and the global minimum is $\mathrm{Var}(X)$. In this sense, $\mathbb{E}[X]$ is the 'best guess' for X and $\mathrm{Var}(X)$ is the corresponding MSE.

A.2. Binomial, geometric, and Poisson RVs

Example A.2.1 (Binomial RV)**.** Let $X_1, X_2, \cdots, X_n$ be independent and identically distributed Bernoulli p variables. Let $X = X_1 + \cdots + X_n$. One can think of flipping the same probability p coin n times. Then X is the total number of heads. Note that X has the following PMF

$$\mathbb{P}(X = k) = \binom{n}{k} p^k (1-p)^{n-k}$$

for k nonnegative integer, and $\mathbb{P}(X = k) = 0$ otherwise. We say X follows the Binomial distribution with parameters n and p, and write $X \sim \mathrm{Binomial}(n, p)$.

We can compute the mean and variance of X using the above PMF directly, but it is much easier to break it up into Bernoulli variables and use linearity. Recall that $X_i \sim \mathrm{Bernoulli}(p)$ and we have $\mathbb{E}(X_i) = p$ and $\mathrm{Var}(X_i) = p(1-p)$ for each $1 \le i \le n$ (from Exercise A.1.5). So by linearity of expectation (Exercise A.5.1),

$$\mathbb{E}(X) = \mathbb{E}(X_1 + \cdots + X_n) = \mathbb{E}(X_1) + \cdots + \mathbb{E}(X_n) = np.$$

On the other hand, since X_i's are independent, variance of X is the sum of variance of X_i's (Exercise A.5.6) so

$$\mathrm{Var}(X) = \mathrm{Var}(X_1 + \cdots + X_n) = \mathrm{Var}(X_1) + \cdots + \mathrm{Var}(X_n) = np(1-p).$$

Example A.2.2 (Geometric RV)**.** Suppose we flip a probability p coin until it lands heads. Let X be the total number of trials until the first time we see heads. Then in order for $X = k$, the first $k-1$ flips must land on tails and the kth flip should land on heads. Since the flips are independent with each other,

$$\mathbb{P}(X = k) = \mathbb{P}(\{T, T, \cdots, T, H\}) = (1-p)^{k-1} p.$$

This is valid for k positive integer, and $\mathbb{P}(X = k) = 0$ otherwise. Such a RV is called a *Geometric RV* with (success) parameter p, and we write $X \sim \text{Geom}(p)$.

The mean and variance of X can be easily computed using its moment generating function, which we will learn soon in this course. For their direct computation, note that

$$\begin{aligned}
\mathbb{E}(X) - (1-p)\mathbb{E}(X) &= (1-p)^0 p + 2(1-p)^1 p + 3(1-p)^2 p + 4(1-p)^3 p \cdots \\
&\qquad - \left[(1-p)^1 p + 2(1-p)^2 p + 3(1-p)^3 p + \cdots\right] \\
&= (1-p)^0 p + (1-p)^1 p + (1-p)^2 p + (1-p)^3 p \cdots \\
&= \frac{p}{1-(1-p)} = 1,
\end{aligned}$$

where we recognized the series after the second equality as a geometric series. This gives

$$\mathbb{E}(X) = 1/p.$$

(In fact, one can apply Exercise A.1.4 and quickly compute the expection of a Geometric RV.)

Exercise A.2.3. Let $X \sim \text{Geom}(p)$. Use a similar computation as we had in Example A.2.2 to show $\mathbb{E}(X^2) = (2-p)/p^2$. Using the fact that $\mathbb{E}(X) = 1/p$, conclude that $\text{Var}(X) = (1-p)/p^2$.

Example A.2.4 (Poisson RV)**.** A RV X is a *Poisson RV* with rate $\lambda > 0$ if

$$\mathbb{P}(X = k) = \frac{\lambda^k e^{-\lambda}}{k!}$$

for all nonnegative integers $k \geq 0$. We write $X \sim \text{Poisson}(\lambda)$.

Poisson distribution is obtained as a limit of the Binomial distribution as the number n of trials tend to infinity while the mean np is kept at constant λ. Namely, let $Y \sim \text{Binomial}(n, p)$ and suppose $np = \lambda$. This means that we expect to see λ successes out of n trials. Then what is the probability that we see, say, k successes out of n trials, when n is large? Since the mean is λ, this probability should be very small when k is large compared to λ. Indeed, we can rewrite the Binomial PMF as

$$\begin{aligned}
\mathbb{P}(Y = k) &= \frac{n(n-1)(n-2)\cdots(n-k+1)}{k!} p^k (1-p)^{n-k} \\
&= \frac{n}{n}\left(1-\frac{1}{n}\right)\left(1-\frac{2}{n}\right)\cdots\left(1-\frac{k-1}{n}\right)\frac{(np)^k}{k!}(1-p)^{n-k} \\
&= \left(1-\frac{1}{n}\right)\left(1-\frac{2}{n}\right)\cdots\left(1-\frac{k-1}{n}\right)\frac{\lambda^k}{k!}\left(1-\frac{\lambda}{n}\right)^{n-k}.
\end{aligned}$$

As n tends to infinity, the limit of the last expression is precisely the right hand side of (A.2.4). [1]

Exercise A.2.5. Let $X \sim \text{Poisson}(\lambda)$. Show that $\mathbb{E}(X) = \text{Var}(X) = \lambda$.

[1] Later, we will interpret the value of a Poisson variable $X \sim \text{Poisson}(\lambda)$ as the number of customers arriving during a unit time interval, where the waiting time between consecutive customers is distributed as an independent exponential distribution with mean $1/\lambda$. Such an arrival process is called the Poisson process.

A.3. Continuous Random Variables

So far we have only considered discrete RVs, which takes either finitely many or countably many values. While there are many examples of discrete RVs, there are also many instances of RVs which various continuously (e.g., temperature, height, weight, price, etc.). To define a discrete RV, it was enough to specify its PMF. For a continuous RV, *probability distribution function* (PDF) plays an analogous role of PMF. We also need to replace summation $\sum$ with an integral $\int dx$.

Namely, X is a *continuous RV* if there is a function $f_X : \mathbb{R} \to [0,\infty)$ such that for any interval $[a,b]$, the probability that X takes a value from an interval $(a,b]$ is given by integrating f_X over the interval $(a,b]$:

$$\mathbb{P}(X \in (a,b]) = \int_a^b f_X(x)\,dx.$$

The *cumulative distribution function* (CDF) of a RV X (either discrete or continuous), denoted by F_X, is defined by

$$F_X(x) = \mathbb{P}(X \le x).$$

By definition of PDF, we get

$$F_X(x) = \int_{-\infty}^{x} f_X(t)\,dt.$$

Conversely, PDFs can be obtained by differentiating corresponding CDFs.

Exercise A.3.1. Let X be a continuous RV with PDF f_X. Let a be a continuity point of f_X, that is, f_X is continuous at a. Show that $F_X(x)$ is differentiable at $x = a$ and

$$\frac{dF_X}{dx}\Big|_{x=a} = f_X(a).$$

The expectation of a continuous RV X with pdf f_X is defined by

$$\mathbb{E}(X) = \int_{-\infty}^{\infty} x f_X(x)\,dx,$$

and its variance $\mathrm{Var}(X)$ is defined by the same formula (A.1).

Exercise A.3.2 (Tail sum formula for expectation)**.** Let X be a continuous RV with PDF f_X and suppose $f_X(x) = 0$ for all $x < 0$. Use Fubini's theorem to show that

$$\mathbb{E}(X) = \int_0^{\infty} \mathbb{P}(X \ge t)\,dt.$$

Example A.3.3 (Higher moments)**.** Let X be a continuous RV with PDF f_X and suppose $f_X(x) = 0$ for all $x < 0$. We will show that for any real number $\alpha > 0$,

$$\mathbb{E}[X^\alpha] = \int_0^{\infty} x^\alpha f_X(x)\,dx.$$

First, Use Exercise A.3.2 and to write

$$\begin{aligned}\mathbb{E}[X^\alpha] &= \int_0^{\infty} \mathbb{P}(X^\alpha \ge x)\,dx \\ &= \int_0^{\infty} \mathbb{P}(X \ge x^{1/\alpha})\,dx \\ &= \int_0^{\infty}\int_{x^{1/\alpha}}^{\infty} f_X(t)\,dt\,dx.\end{aligned}$$

We then use Fubini's theorem to change the order of integral. This gives

$$\mathbb{E}(X) = \int_0^{\infty}\int_{x^{1/\alpha}}^{\infty} f_X(t)\,dt\,dx = \int_0^{\infty}\int_0^{t^\alpha} f_X(t)\,dx\,dt = \int_0^{\infty} t^\alpha f_X(t)\,dt,$$

as desired. ▲

Exercise A.3.4. Let X be a continuous RV with PDF f_X and suppose $f_X(x) = 0$ for all $x < 0$. Let $g : \mathbb{R} \to \mathbb{R}$ be a strictly increasing function. Use Fubini's theorem and tail sum formula for expectation to show

$$\mathbb{E}[g(X)] = \int_0^\infty g(x) f_X(x)\, dx.$$

A.4. Uniform, exponential, and normal RVs

In this section, we introduce three important continuous RVs.

Example A.4.1 (Uniform RV)**.** X is a *uniform RV* on the interval $[a, b]$ (denoted by $X \sim \text{Uniform}([a, b])$) if it has PDF

$$f_X(x) = \frac{1}{b-a} \mathbf{1}(a \le x \le b).$$

An easy computation gives its CDF:

$$\mathbb{P}(X \le x) = \begin{cases} 0 & x < a \\ (x-a)/(b-a) & a \le x \le b \\ 1 & x > b. \end{cases}$$

▲

Exercise A.4.2. Let $X \sim \text{Uniform}([a, b])$. Show that

$$\mathbb{E}(X) = \frac{a+b}{2}, \qquad \text{Var}(E) = \frac{(b-a)^2}{12}.$$

Example A.4.3 (Exponential RV)**.** X is an *exponential* RV with rate λ (denoted by $X \sim \text{Exp}(\lambda)$) if it has PDF

$$f_X(x) = \lambda e^{-\lambda x} \mathbf{1}(x \ge 0).$$

Integrating the PDF gives its CDF

$$\mathbb{P}(X \le x) = (1 - e^{-\lambda x}) \mathbf{1}(x \ge 0).$$

Using Exercise A.3.2, we can compute

$$\mathbb{E}(X) = \int_0^\infty e^{-\lambda t}\, dt = \left[-\frac{e^{-\lambda t}}{\lambda} \right]_0^\infty = 1/\lambda.$$

▲

Exercise A.4.4. Let $X \sim \text{Exp}(\lambda)$. Show that $\mathbb{E}(X) = 1/\lambda$ directly using definition (A.3). Also show that $\text{Var}(X) = 1/\lambda^2$.

Example A.4.5 (Normal RV)**.** X is a *normal* RV with mean μ and variance σ^2 (denoted by $X \sim N(\mu, \sigma^2)$) if it has PDF

$$f_X(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}.$$

If $\mu = 0$ and $\sigma^2 = 1$, then X is called a standard normal RV. Note that if $X \sim N(\mu, \sigma^2)$, then $Y := X - \mu$ has PDF

$$f_Y(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}}.$$

Since this is an even function, it follows that $\mathbb{E}(Y) = 0$. Hence $\mathbb{E}(X) = \mu$. ▲

Exercise A.4.6 (Gaussian integral)**.** In this exercise, we will show $\int_{-\infty}^\infty e^{-x^2}\, dx = \sqrt{\pi}$.

(i) Show that

$$\int x e^{-x^2}\, dx = -\frac{1}{2} e^{-x^2} + C.$$

(ii) Let $I = \int_{-\infty}^{\infty} e^{-x^2}\,dx$. Show that

$$I^2 = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} e^{-(x^2+y^2)}\,dx\,dy.$$

(iii) Use polar coordinate (r,θ) to rewrite

$$I^2 = \int_0^{2\pi}\int_0^{\infty} e^{-r^2}\,r\,dr\,d\theta = 2\pi\int_0^{\infty} re^{-r^2}\,dr.$$

Then use (i) to deduce $I^2 = \pi$. Conclude $I = \sqrt{\pi}$.

Exercise A.4.7. Let $X \sim N(\mu,\sigma^2)$. In this exercise, we will show $\mathrm{Var}(X) = \sigma^2$.

(i) Show that $\mathrm{Var}(X) = \mathrm{Var}(X-\mu)$.

(ii) Use integration by parts and Exercise A.4.6 to show that

$$\int_0^{\infty} x^2e^{-x^2}\,dx = \left[x\left(-\frac{1}{2}e^{-x^2}\right)\right]_0^{\infty} + \int_0^{\infty}\frac{1}{2}e^{-x^2}\,dx = \frac{\sqrt{\pi}}{4}.$$

(iii) Use change of variable $x = \sqrt{2}\sigma t$ and (ii) to show

$$\int_0^{\infty}\frac{x^2}{\sqrt{2\pi\sigma^2}}e^{-\frac{x^2}{2\sigma^2}}\,dx = \frac{2\sigma^2}{\sqrt{\pi}}\int_0^{\infty} t^2e^{-t^2}\,dt = \frac{\sigma^2}{2}.$$

Use (i) to conclude $\mathrm{Var}(X) = \sigma^2$.

Proposition A.4.8 (Linear transform)**.** *Let X be a RV with PDF f_X. Fix constants $a,b\in\mathbb{R}$ with $a>0$, and define a new RV $Y = aX+b$. Then*

$$f_{aX+b}(y) = \frac{1}{|a|}f_X((y-b)/a).$$

PROOF. First suppose $a>0$. Then

$$F_Y(y) = \mathbb{P}(Y\le y) = \mathbb{P}(aX+b\le y) = \mathbb{P}(X\le (y-b)/a) = \int_{-\infty}^{(y-b)/a} f_X(t)\,dt.$$

By differentiating the last integral by y, we get

$$f_Y(y) = \frac{dF_Y(y)}{dy} = \frac{1}{a}f_X((y-b)/a).$$

For $a<0$, a similar calculation shows

$$F_Y(y) = \mathbb{P}(Y\le y) = \mathbb{P}(aX+b\le y) = \mathbb{P}(X\ge (y-b)/a) = \int_{(y-b)/a}^{\infty} f_X(t)\,dt,$$

so we get

$$f_Y(y) = \frac{dF_Y(y)}{dy} = -\frac{1}{a}f_X((y-b)/a).$$

This shows the assertion. □

Example A.4.9 (Linear transform of normal RV is normal)**.** Let $X\sim N(\mu,\sigma^2)$ and fix constants $a,b\in\mathbb{R}$ with $a\neq 0$. Define a new RV $Y = aX+b$. Then since

$$f_X(x) = \frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right),$$

by Proposition A.4.8, we have

$$f_Y(y) = \frac{1}{\sqrt{2\pi(a\sigma)^2}}\exp\left(-\frac{(y-b-a\mu)^2}{2(a\sigma)^2}\right).$$

Notice that this is the PDF of a normal RV with mean $a\mu + b$ and variance $(a\sigma)^2$. In particular, if we take $a = 1/\sigma$ and $b = \mu/\sigma$, then $Y = (X-\mu)/\sigma \sim N(0,1)$, the standard normal RV. This is called *standardization* of normal RV.

A.5. Expectation and variance of sums of RVs

In this section, we learn how we can compute expectation and variance for sums of RVs. For expectation, we will see that we can swap the order to summation and expectation. This is called the *linearity of expectation*, whose importance cannot be overemphasized.

Exercise A.5.1 (Linearity of expectation)**.** In this exercise, we will show that the expectation of sum of RVs is the sum of expectation of individual RVs.

(i) Let X and Y be RVs. Show that

$$\sum_y \mathbb{P}(X = x, Y = y) = \mathbb{P}(X = x).$$

(ii) Verify the following steps:

$$\begin{aligned}
\mathbb{E}(X+Y) &= \sum_z z\,\mathbb{P}(X+Y=z) \\
&= \sum_z \sum_{\substack{x,y \\ x+y=z}} (x+y)\mathbb{P}(X=x,\, Y=y) \\
&= \sum_{x,y} (x+y)\,\mathbb{P}(X=x,\, Y=y) \\
&= \sum_{x,y} x\,\mathbb{P}(X=x,\, Y=y) + \sum_{x,y} y\,\mathbb{P}(X=x,\, Y=y) \\
&= \sum_x x\left(\sum_y \mathbb{P}(X=x, y=y)\right) + \sum_y y\left(\sum_x \mathbb{P}(X=x,\, Y=y)\right) \\
&= \sum_x x\mathbb{P}(X=x) + \sum_y y\mathbb{P}(Y=y) \\
&= \mathbb{E}(X) + \mathbb{E}(Y).
\end{aligned}$$

(iii) Use induction to show that for any RVs $X_1, X_2, \cdots, X_n$, we have

$$\mathbb{E}(X_1 + X_2 + \cdots + X_n) = \mathbb{E}(X_1) + \mathbb{E}(X_2) + \cdots + \mathbb{E}(X_n).$$

Our first application of linearity of expectation is the following useful formula for variance.

Exercise A.5.2. For any RV X, show that

$$\mathrm{Var}(X) = \mathbb{E}(X^2) - \mathbb{E}(X)^2.$$

When we try to write down the variance of a sum of RVs, in addition to the variance of each RV, we have extra contribution from each pairs of RVs called the covariance.

Exercise A.5.3. In this exercise, we will see how we can express the variance of sums of RVs. For two RVs X and Y, define their *covariance* $\mathrm{Cov}(X,Y)$ by

$$\mathrm{Cov}(X,Y) = \mathbb{E}(XY) - \mathbb{E}(X)\mathbb{E}(Y).$$

(i) Use Exercises A.5.2 and A.5.1 to show that

$$\mathrm{Var}(X+Y) = \mathrm{Var}(X) + \mathrm{Var}(Y) + 2\mathrm{Cov}(X,Y).$$

(ii) Use induction to show that for RVs $X_1, X_2 \cdots, X_n$

$$\mathrm{Var}\left(\sum_{i=1}^n X_i\right) = \sum_{i=1}^n \mathrm{Var}(X_i) + 2\sum_{1\le i,j\le n} \mathrm{Cov}(X_i, X_j)$$

When knowing something about one RV does not yield any information of the other, we say the two RVs are independent. Formally, we say two RVs X and Y are *independent* if for any two subsets $A_1, A_2 \subseteq \mathbb{R}$,

$$\mathbb{P}(X \in A_1 \text{ and } Y \in A_2) = \mathbb{P}(X \in A_1)\mathbb{P}(Y \in A_2).$$

We say two events or RVs are *dependent* if they are not independent.

Example A.5.4. Flip two fair coins at the same time, and let $X = 1$ if the first coin lands heads and $X = -1$ if it lands tails. Let Y be a similar RV for the second coin. Suppose each of the four outcomes in $\Omega = \{H, T\}^2$ are equality likely. Clearly knowing about one coin does not give any information of the other. For instance, the first coin lands on heads with probability $1/2$. Whether the first coin lands on heads or not, the second coin will land on heads with probability $1/2$. So

$$\mathbb{P}(X = 1 \text{ and } Y = 1) = \frac{1}{2} \cdot \frac{1}{2} = \mathbb{P}(X = 1)\mathbb{P}(Y = 1).$$

One can verify the above equation for possible outcomes. Hence X and Y are independent. ▲

Exercise A.5.5. Let X, Y be two indepenent RVs. Show that

$$\mathbb{E}[XY] = \mathbb{E}[X]\mathbb{E}[Y].$$

In the following exercise, we will see that we can swap summation and variance as long as the RVs we are adding up are independent.

Exercise A.5.6. Recall the definition of covariance given in Exercise A.5.3.

(i) Show that if two RVs X and Y are independent, then $\mathrm{Cov}(X, Y) = 0$

(ii) Use Exercise A.5.3 to conclude that if $X_1, \cdots, X_n$ are independent RVs, then

$$\mathrm{Var}(X_1 + \cdots + X_n) = \mathrm{Var}(X_1) + \cdots + \mathrm{Var}(X_n).$$

A.6. Conditional expectation

Let X, Y be discrete RVs. Recall that the expectation $\mathbb{E}(X)$ is the 'best guess' on the value of X when we do not have any prior knowledge on X. But suppose we have observed that some possibly related RV Y takes value y. What should be our best guess on X, leveraging this added information? This is called the *conditional expectation of X given $Y = y$*, which is defined by

$$\mathbb{E}[X|Y = y] = \sum_x x\mathbb{P}(X = x|Y = y).$$

This best guess on X given $Y = y$, of course, depends on y. So it is a function in y. Now if we do not know what value Y might take, then we omit y and $\mathbb{E}[X|Y]$ becomes a RV, which is called the *conditional expectation of X given Y*.

Example A.6.1. Suppose we have a biased coin whose probability of heads is itself random and is distributed as $Y \sim \mathrm{Uniform}([0, 1])$. Let's flip this coin n times and let X be the total number of heads. Given that $Y = y \in [0, 1]$, we know that X follows $\mathrm{Binomial}(n, y)$ (in this case we write $X|U \sim \mathrm{Binomial}(n, Y)$). So $\mathbb{E}[X|Y = y] = ny$. Hence as a random variable, $\mathbb{E}[X|Y] = nY \sim \mathrm{Uniform}([0, n])$. So the expectation of $\mathbb{E}[X|Y]$ is the mean of $\mathrm{Uniform}([0, n])$, which is $n/2$. This value should be the true expectation of X. ▲

The above example suggests that if we first compute the conditional expectation of X given $Y = y$, and then average this value over all choice of y, then we should get the actual expectation of X. Justification of this observation is based on the following fact

$$\mathbb{P}(Y = y\,|\,X = x)\mathbb{P}(X = x) = \mathbb{P}(X = x,\, Y = y) = \mathbb{P}(X = x\,|\,Y = y)\mathbb{P}(Y = y).$$

That is, if we are interested in the event that $(X, Y) = (x, y)$, then we can either first observe the value of X and then Y, or the other way around.

Proposition A.6.2 (Iterated expectation)**.** *Let X, Y be discrete RVs. Then* $\mathbb{E}(X) = \mathbb{E}[\mathbb{E}[X|Y]]$.

PROOF. We are going to write the iterated expectation $\mathbb{E}[\mathbb{E}[X|Y]]$ as a double sum and swap the order of summation (Fubini's theorem, as always).

$$\begin{aligned}
\mathbb{E}[\mathbb{E}[X|Y]] &= \sum_y \mathbb{E}[X|Y=y]\mathbb{P}(Y=y) \\
&= \sum_y \left(\sum_x x\,\mathbb{P}(X=x|Y=y)\right)\mathbb{P}(Y=y) \\
&= \sum_y \sum_x x\,\mathbb{P}(X=x|Y=y)\mathbb{P}(Y=y) \\
&= \sum_y \sum_x x\,\mathbb{P}(X=x,\,Y=y) \\
&= \sum_x \sum_y x\,\mathbb{P}(Y=y|X=x)\mathbb{P}(X=x) \\
&= \sum_x x\left(\sum_y \mathbb{P}(Y=y|X=x)\right)\mathbb{P}(X=x) \\
&= \sum_x x\,\mathbb{P}(X=x) = \mathbb{E}(X).
\end{aligned}$$

□

Remark A.6.3. Here is an intuitive reason why the iterated expectation works. Suppose you want to make the best guess $\mathbb{E}(X)$. Pretending you know Y, you can imporove your guess to be $E(X\,|\,Y)$. Then you admit that you didn't know anything about Y and average over all values of Y. The result is $\mathbb{E}[\mathbb{E}[X\,|\,Y]]$, and this should be the same best guess on X when we don't know anything about Y.

All our discussions above hold for continuous RVs as well: We simply replace the sum by integral and PMF by PDF. To summarize how we compute the iterated expectations when we condition on discrete and continuous RV:

$$\mathbb{E}[\mathbb{E}[X\,|\,Y]] = \begin{cases} \sum_y \mathbb{E}[X\,|\,Y=y]\mathbb{P}(Y=y) & \text{if } Y \text{ is discrete} \\ \int_{-\infty}^{\infty} \mathbb{E}[X\,|\,Y=y] f_Y(y)\,dy & \text{if } Y \text{ is continuous.} \end{cases}$$

Exercise A.6.4 (Iterated expectation for probability)**.** Let X, Y be RVs.

(i) For any $x \in \mathbb{R}$, show that $\mathbb{P}(X \le x) = \mathbb{E}[\mathbf{1}(X \le x)]$.
(ii) By using iterated expectation, show that

$$\mathbb{P}(X \le x) = \mathbb{E}[\mathbb{P}(X \le x\,|\,Y)],$$

where the expectation is taken over for all possible values of Y.

Example A.6.5 (Example A.6.1 revisited)**.** Let $Y \sim \text{Uniform}([0,1])$ and $X \sim \text{Binomial}(n, Y)$. Then $X|Y = y \sim \text{Binomial}(n, y)$ so $\mathbb{E}[X|Y=y] = ny$. Hence

$$\mathbb{E}[X] = \int_0^1 \mathbb{E}[X\,|\,Y=y] f_Y(y)\,dy = \int_0^1 ny\,dy = n/2.$$

▲

Example A.6.6. Let $X_1 \sim \text{Exp}(\lambda_1)$ and $X_2 \sim \text{Exp}(\lambda_2)$ be independent exponential RVs. We will show that

$$\mathbb{P}(X_1 < X_2) = \frac{\lambda_1}{\lambda_1 + \lambda_2}$$

using the iterated expectation. Using iterated expectation for probability,

$$\begin{aligned}\mathbb{P}(X_1 < X_2) &= \int_0^\infty \mathbb{P}(X_1 < X_2 \,|\, X_1 = x_1)\lambda_1 e^{-\lambda_1 x_1}\, dx_1 \\ &= \int_0^\infty \mathbb{P}(X_2 > x_1)\lambda_1 e^{-\lambda_1 x_1}\, dx_1 \\ &= \lambda_1 \int_0^\infty e^{-\lambda_2 x_1} e^{-\lambda_1 x_1}\, dx_1 \\ &= \lambda_1 \int_0^\infty e^{-(\lambda_1+\lambda_2) x_1}\, dx_1 = \frac{\lambda_1}{\lambda_1+\lambda_2}.\end{aligned}$$

▲

Exercise A.6.7 (Order statistics and i.i.d. Uniform RVs)**.** $U_1, \ldots, U_m$ be i.i.d. Uniform$(0,1)$ random variables. Let $U^{(1)} \le U^{(2)} \le \cdots \le U^{(m)}$ denote their order statistics, that is, $U^{(j)}$ is the jth largest value among $U_1, \ldots, U_m$.

(i) Show that, for each i, $\mathbb{P}(U^{(1)} = U_i) = 1/m$. That is, each U_i is equally likely to be the first order statistic, $U^{(1)}$.

(ii) Show that, with probability 1, $U^{(1)} < U^{(2)} < \cdots < U^{(m)}$. (*Hint*: First show that $\mathbb{P}(U_1 = U_2) = 0$. Then by union bound, $\mathbb{P}(U_i$'s are not all distinct$) \le \binom{2m}{2}\mathbb{P}(U_1 = U_2) = 0$.)

Consequently, almost surely, there is a permutation π on $\{1, \ldots, m\}$ such that $U^{(i)} = U_{\pi(i)}$ for all $i = 1, \ldots, m$.

(iii) Show that, conditional on $U^{(1)} = x \in (0,1)$, $U_{\pi(2)}, \ldots, U_{\pi(m)}$ are i.i.d. Uniform$([x,1])$. (*Hint:* First justify the following:

$$\begin{aligned}&\mathbb{P}\left(U_{\pi(2)} \le y_2, \ldots, U_{\pi(m)} \le y_m \,\middle|\, U_{\pi(1)} = x, \pi(1) = 1\right) \\ &\quad= \mathbb{P}\left(U_2 \le y_2, \ldots, U_m \le y_m \,\middle|\, U_1 = x, U_2, \ldots, U_m \ge x\right) \\ &\quad= \mathbb{P}\left(U_2 \le y_2, \ldots, U_m \le y_m \,\middle|\, U_2, \ldots, U_m \ge x\right) \\ &\quad= \prod_{i=2}^m \frac{\mathbb{P}(x \le U_i \le y_i)}{\mathbb{P}(x \le U_i)} \\ &\quad= \prod_{i=2}^m \left(\frac{y_i - x}{1-x}\right).\end{aligned}$$

Then average over the value of $\pi(1)$, which is uniformly distributed over $1, \ldots, m$, and use iterated expectation, to show

$$\mathbb{P}\left(U_{\pi(2)} \le y_2, \ldots, U_{\pi(m)} \le y_m \,\middle|\, U_{\pi(1)} = x\right) = \prod_{i=2}^m \left(\frac{y_i - x}{1-x}\right) = \prod_{i=2}^m \mathbb{P}(U_{\pi(i)} \le y_i \,|\, U_{\pi(1)} = x).$$

The expression in the middle is exactly the CDF of $m-1$ i.i.d. Uniform$(x,1)$ RVs. The second equality shows the conditional independence of $U_{\pi(2)}, \ldots, U_{\pi(m)}$.)

A.7. Elementary limit theorems

The primary subject in this note is the sequence of i.i.d. RVs and their partial sums. Namely, let $X_1, X_2, \cdots$ be an (infinite) sequence of i.i.d. RVs, and define their nth partial sum $S_n = X_1 + X_2 + \cdots + X_n$ for all $n \ge 1$. If we call X_i the ith step size or *increment*, then the sequence of RVs $(S_n)_{n\ge1}$ is called a *random walk*, where we usually set $S_0 = 0$. Think of X_i as the gain or loss after betting once in a casino. Then S_n is the net gain of fortune after betting n times. Of course there are ups and downs in the short term, but what we want to analyze using probability theory is the long-term behavior of the random walk $(S_n)_{n\ge1}$. Results of this type are called limit theorems.

Suppose each increment X_k has a finite mean μ. Then by linearity of expectation and independence of the increments, we have

$$\mathbb{E}\left(\frac{S_n}{n}\right) = \frac{\mathbb{E}[S_n]}{n} = \mu,$$
$$\mathrm{Var}\left(\frac{S_n}{n}\right) = \frac{\mathrm{Var}(S_n)}{n^2} = \frac{n\,\mathrm{Var}(X_1)}{n^2} = \frac{\mathrm{Var}(X_1)}{n}.$$

So the sample mean S_n/n has constant expectation and shrinking variance. Hence it makes sense to guess that it should behave as the constant μ, without taking the expectation. That is,

$$\lim_{n\to\infty} \frac{S_n}{n} = \mu.$$

But this expression is shaky, since the left hand side is a limit of RVs while the right hand side is a constant. In what sense the random sample means converge to μ? This is the content of the *law of large numbers*, for which we will prove a weak and a strong versions.

The first limit theorem we will encounter is called the Weak Law of Large Numbers (WLLN), which is stated below:

Theorem A.7.1 (WLLN)**.** *Let $(X_k)_{k\geq 1}$ be i.i.d. RVs with mean $\mu < \infty$ and let $S_n = \sum_{k=1}^n X_i$, $n \geq 1$ be a random walk. Then for any positive constant $\varepsilon > 0$,*

$$\lim_{n\to\infty} \mathbb{P}\left(\left|\frac{S_n}{n} - \mu\right| > \varepsilon\right) = 0.$$

In words, the probability that the sample mean S_n/n is *not* within ε distance from its expectation μ decays to zero as n tends to infinity. In this case, we say the sequence of RVs $(S_n/n)_{n\geq 1}$ converges to μ *in probability*.

The second version of law of large numbers is call the *strong law of large numbers* (SLLN), which is available if the increments have finite variance.

Theorem A.7.2 (SLLN)**.** *Let $(X_k)_{k\geq 1}$ be i.i.d. RVs and let $S_n = \sum_{k=1}^n X_i$, $n \geq 1$ be a random walk. Suppose $\mathbb{E}[X_1] = \mu < \infty$ and $\mathbb{E}[X_1^2] < \infty$. Then*

$$\mathbb{P}\left(\lim_{n\to\infty} \frac{S_n}{n} = \mu\right) = 1.$$

To make sense out of this, notice that the limit of sample mean $\lim_{n\to} S_n/n$ is itself a RV. Then SLLN says that this RV is well defined and its value is μ with probability 1. In this case, we say the sequence of RVs $(S_n/n)_{n\geq 1}$ converges to μ *with probability 1* or *almost surely*.

Perhaps one of the most celebrated theorems in probability theory is the *central limit theorem* (CLT), which tells about how the sample mean S_n/n "fluctuates" around its mean μ. From A.7, if we denote $\sigma^2 = \mathrm{Var}(X_1) < \infty$, we know that $\mathrm{Var}(S_n/n) = \sigma^2/n \to 0$ as $n \to \infty$. So the fluctuation decays as we add up more increments. To see the effect of fluctuation, we first center the sample mean by subtracting its expectation and "zoom in" by dividing by the standard deviation $\sigma/\sqrt{n}$. This is where the name 'central limit' comes from: it describes the limit of centered random walks.

Theorem A.7.3 (Central Limit Theorem)**.** *Let $(X_k)_{k\geq 1}$ be i.i.d. RVs and let $S_n = \sum_{k=1}^n X_i$, $n \geq 1$. Suppose $\mathbb{E}[X_1] < \infty$ and $\mathbb{E}[X_1^2] = \sigma^2 < \infty$. Let $Z \sim N(0,1)$ be a standard normal RV and define*

$$Z_n = \frac{S_n - \mu n}{\sigma\sqrt{n}} = \frac{S_n/n - \mu}{\sigma/\sqrt{n}}.$$

Then Z_n converges to Z as $n \to \infty$ in distribution, namely,

$$\lim_{n\to} \mathbb{P}(Z_n \leq z) = \mathbb{P}(Z \leq z).$$

In words, the centered and rescaled RV Z_n is asymptotically distributed as a standard normal RV $Z \sim N(0,1)$. In this case, we say Z_n converges to Z as $n \to \infty$ *in distribution*. This is a remarkable result since as long as the increments X_k have finite mean and variance, it does not matter which distribution that they follow: the 'central limit' always looks like a standard normal distribution. Later in this section, we will prove this result by using the MGF of S_n and Taylor-expanding it up to the second order term.

Exercise A.7.4 (Normal approximation of binomial distribution)**.** Let $(X_n)_{n\geq 1}$ be a sequence of i.i.d. Bernoulli(p) RVs. Let $S_n = X_1 + \cdots + X_n$.

(i) Let $Z_n = (S_n - np)/\sqrt{np(1-p)}$. Show that as $n \to \infty$, Z_n converges to the standard normal RV $Z \sim N(0,1)$ in distribution.

(ii) Using Theorem A.7.3, conclude that if $Y_n \sim \text{Binomial}(n,p)$, then

$$\frac{Y_n - np}{\sqrt{np(1-p)}} \Rightarrow Z \sim N(0,1).$$

(iii) From **(ii)**, deduce that have the following approximation

$$\mathbb{P}(Y_n \leq x) \approx \mathbb{P}\left(Z \leq \frac{x - np}{\sqrt{np(1-p)}}\right),$$

which becomes more accurate as $n \to \infty$.

A.8. Bounding tail probabilities

In this subsection, we introduce two general inequalities called the Markov's and Chebyshef's inequalities. They are useful in bounding tail probabilities of the form $\mathbb{P}(X \geq x)$ using the expectation $\mathbb{E}[X]$ and variance $\text{Var}(X)$, respectively. Their proofs are quite simple but they have lots of nice applications and implications.

Proposition A.8.1 (Markov's inequality)**.** *Let $X \geq 0$ be a nonnegative RV with finite expectation. Then for any $a > 0$, we have*

$$\mathbb{P}(X \geq a) \leq \frac{\mathbb{E}[X]}{a}.$$

PROOF. Consider an auxiliary RV Y define as follows:

$$Y = \begin{cases} a & \text{if } X \geq a \\ 0 & \text{if } X < a. \end{cases}$$

Note that we always have $Y \leq X$. Hence we should have $\mathbb{E}[Y] \leq \mathbb{E}[X]$. But since $\mathbb{E}[Y] = a\mathbb{P}(X \geq a)$, we have

$$\lambda\mathbb{P}(X \geq a) \leq \mathbb{E}[X].$$

Dividing both sides by $a > 0$ gives the assertion. □

Example A.8.2. We will show that, for any RV Z, $\mathbb{E}[Z^2] = 0$ implies $\mathbb{P}(Z = 0) = 1$. Indeed, Markov's inequality gives that for any $a > 0$,

$$\mathbb{P}(Z^2 \geq a) \leq \frac{\mathbb{E}[Z^2]}{a} = 0.$$

This means that $\mathbb{P}(Z^2 = 0) = 1$, so $\mathbb{P}(Z = 0) = 1$. ▲

Proposition A.8.3 (Chebyshef's inequality)**.** *Let X be any RV with $\mathbb{E}[X] = \mu < \infty$ and $\text{Var}(X) < \infty$. Then for any $a > 0$, we have*

$$\mathbb{P}(|X - \mu| \geq a) \leq \frac{\text{Var}(X)}{a^2}.$$

PROOF. Applying Markov's inequality for the nonnegative RV $(X-\mu)^2$, we get

$$\mathbb{P}(|X-\mu| \geq a) = \mathbb{P}((X-\mu)^2 \geq a^2) \leq \frac{\mathbb{E}[(X-\mu)^2]}{a^2} = \frac{\mathrm{Var}(X)}{a^2}.$$

□

Example A.8.4. Let $X \sim \mathrm{Exp}(\lambda)$. Since $\mathbb{E}[X] = 1/\lambda$, for any $a > 0$, the Markov's inequality gives

$$\mathbb{P}(X \geq a) \leq \frac{1}{a\lambda},$$

while the true probability is

$$\mathbb{P}(X \geq a) = e^{-\lambda a}.$$

On the other hand, $\mathrm{Var}(X) = 1/\lambda^2$ so Chebyshef's inequality gives

$$\mathbb{P}(|X - 1/\lambda| \geq a) = \frac{1}{a^2\lambda^2}.$$

If $1/\lambda \leq a$, the true probability is

$$\begin{aligned}\mathbb{P}(|X-1/\lambda| \geq a) &= \mathbb{P}(X \geq a + 1/\lambda) + \mathbb{P}(X \leq -a + 1/\lambda) \\ &= \mathbb{P}(X \geq a + 1/\lambda) = e^{-\lambda(a+1/\lambda)} = e^{-1-\lambda a}.\end{aligned}$$

As we can see, both Markov's and Chebyshef's inequalities give loose estimates, but the latter gives a slightly stronger bound. ▲

Example A.8.5 (Chebyshef's inequality for bounded RVs)**.** Let X be a RV taking values from the interval $[a,b]$. Suppose we don't know anything else about X. Can we say anything useful about tail probability $\mathbb{P}(X \geq \lambda)$? If we were to use Markov's inequality, then certainly $a \leq \mathbb{E}[X] \leq b$ and in the worst case $\mathbb{E}[X] = b$. Hence we can at least conclude

$$\mathbb{P}(X \geq \lambda) \leq \frac{b}{\lambda}.$$

On the other hand, let's get a bound on $\mathrm{Var}(X)$ and use Chebyshef's inequality instead. We claim that

$$\mathrm{Var}(X) \leq \frac{(b-a)^4}{4},$$

which would yield by Chebyshef's inequality that

$$\mathbb{P}(|X - \mathbb{E}[X]| \leq \lambda) \leq \frac{(b-a)^2}{4\lambda^2}.$$

Intuitively speaking, $\mathrm{Var}(X)$ is the largest when the value of X is as much spread out as possible at the two extreme values, a and b. Hence the largest variance will be achieved when X takes a and b with equal probabilities. In this case, $\mathbb{E}[X] = (a+b)/2$ so

$$\mathrm{Var}(X) = \mathbb{E}[X^2] - \mathbb{E}[X]^2 = \frac{a^2+b^2}{2} - \frac{(a+b)^2}{4} = \frac{(b-a)^2}{4}.$$

▲

Exercise A.8.6. Let X be a RV taking values from the interval $[a,b]$.

(i) Use the usual 'completing squares' trick for a second moment to show that

$$0 \leq \mathbb{E}[(X-t)^2] = (t - \mathbb{E}[X])^2 + \mathrm{Var}(X) \quad \forall\, t \in \mathbb{R}.$$

(ii) Conclude that $\mathbb{E}[(X-t)^2]$ is minimized when $t = \mathbb{E}[X]$ and the minimum is $\mathrm{Var}(X)$.

(iii) By plugging in $t = (a+b)/2$ in (A.8.6), show that

$$\mathrm{Var}(X) = \mathbb{E}[(X-a)(X-b)] + \frac{(b-a)^2}{4} - \left(\mathbb{E}[X] - \frac{a+b}{2}\right)^2.$$

(iv) Show that $\mathbb{E}[(X-a)(X-b)] \le 0$.

(v) Conclude that $\mathrm{Var}(X) \le (b-a)^2/4$, where the equality holds if and only if X takes the extreme values a and b with equal probabilities.

Exercise A.8.7 (Paley-Zigmund inequality)**.** Let X be a nonnegative RV with $\mathbb{E}[|X|] < \infty$. Fix a constant $\theta \ge 0$. We prove the Paley-Zigmund inequality, which gives a lower bound on the tail probabilities and also implies the so-called 'second moment method'.

(i) Write $X = X\mathbf{1}(X > \theta\mathbb{E}[X]) + X\mathbf{1}(X \le \theta\mathbb{E}[X])$. Show that

$$\begin{aligned}\mathbb{E}[X] &= \mathbb{E}[X\mathbf{1}(X \le \theta\mathbb{E}[X])] + \mathbb{E}[X\mathbf{1}(X > \theta\mathbb{E}[X])] \\ &\le \theta\mathbb{E}[X] + \mathbb{E}[X\mathbf{1}(X > \theta\mathbb{E}[X])].\end{aligned}$$

(ii) Use Cauchy-Schwartz inequality (Exc 2.11 in Lecture note 2) to show

$$\begin{aligned}(\mathbb{E}[X\mathbf{1}(X > \theta\mathbb{E}[X])])^2 &\le \mathbb{E}[X^2]\mathbb{E}[\mathbf{1}(X > \theta\mathbb{E}[X])^2] \\ &= \mathbb{E}[X^2]\mathbb{E}[\mathbf{1}(X > \theta\mathbb{E}[X])] \\ &= \mathbb{E}[X^2]\mathbb{P}(X > \theta\mathbb{E}X).\end{aligned}$$

(iii) From (i) and (ii), derive

$$\mathbb{E}[X] \le \theta\mathbb{E}[X] + \sqrt{\mathbb{E}[X^2]\mathbb{P}(X > \theta\mathbb{E}X)}.$$

Conclude that

$$\mathbb{P}(X > \theta\mathbb{E}X) \ge \frac{(1-\theta)^2\mathbb{E}[X]^2}{\mathbb{E}[X^2]}.$$

(iv) (Second moment method) From (iii), conclude that

$$\mathbb{P}(X > 0) \ge \frac{\mathbb{E}[X]^2}{\mathbb{E}[X^2]}.$$

Exercise A.8.8 (Kolmogorov's maximal inequality)**.** Let $X_1, X_2, \cdots$ be i.i.d. RVs with $\mathbb{E}[X_i] = 0$. Denote $S_n = X_1 + \cdots + X_n$ and $S_0 = 0$. In this exercise, we will show that

$$\mathbb{P}\left(\max_{1\le k\le n} |S_k| \ge t\right) \le t^{-2}\mathrm{Var}(S_n). \tag{19}$$

(i) Let $\tau = \inf\{k \ge 0 : |S_k| \ge t\}$ denote the first time that $|S_k|$ exceeds t. Show that

$$\mathbb{E}[S_n^2] \ge \sum_{k=1}^{n} \mathbb{E}[S_n^2\mathbf{1}(\tau = k)].$$

(ii) For each $1 \le k \le n$, note that $S_k\mathbf{1}(\tau = k)$ and $S_n - S_k = X_{k+1} + \cdots + X_n$ are independent. Deduce that

$$\mathbb{E}[S_k\mathbf{1}(\tau = k)(S_n - S_k)] = 0.$$

(iii) For each $1 \le k \le n$, write $S_n^2 = S_k^2 + 2S_k(S_n - S_k) + (S_n - S_k)^2$. Using (ii), show that

$$\begin{aligned}\mathbb{E}[S_n^2\mathbf{1}(\tau = k)] &\ge \mathbb{E}[(S_k^2 + 2S_k(S_n - S_k))\mathbf{1}(\tau = k)] \\ &= \mathbb{E}[(S_k^2\mathbf{1}(\tau = k)] + 2\mathbb{E}[S_k\mathbf{1}(\tau = k)(S_n - S_k)] \\ &= \mathbb{E}[(S_k^2\mathbf{1}(\tau = k)] \ge t^2\mathbb{E}[\mathbf{1}(\tau = k)].\end{aligned}$$

(iv) From (i)-(iii), deduce that

$$\mathbb{E}[S_n^2] \ge t^2\sum_{k=1}^{n}\mathbb{E}[\mathbf{1}(\tau = k)] = t^2\mathbb{E}\left[\sum_{k=1}^{n}\mathbf{1}(\tau = k)\right] = t^2\mathbb{P}(\tau \le n) = t^2\mathbb{P}\left(\max_{1\le k\le n}|S_k| \ge t\right).$$

Conclude Kolmogorov's maximal inequality (19).

A.9. Definition and Examples of MLE

We have mentioned that the core problem in statistics, is to infer an unknown RV X from its sample values $x_1, \cdots, x_n$. When we had absolutely no knowledge on X, in the previous sections we have seen some methods for EDA on the sample, using the sample mean, variance, and quantiles, etc. However, in many cases, we can narrow down our inference problem by imposing a probabilistic (statistical) model for X. Namely, say we know X is a Bernoulli RV with unknown success probability p. We then only need to estimate the parameter p using our sample, not the entire distribution of X. MLE gives a systematic approach to this parameter estimation problem. Below we give the pipeline of MLE.

Pipeline of MLE.
Input: An unknown RV X with distribution $f_{X;\theta}$, parameterized by θ in a parameter space Ω. Also have sample values $x_1, x_2, \cdots, x_n$.
Objective: Obtain an estimation $\hat{\theta}$ of θ using the sample.
Method: Choose $\hat{\theta} \in \Omega$ so that it maximizes the following *likelihood function*

$$L(x_1, \cdots, x_n; \theta) = \prod_{i=1}^{n} f_{X,\theta}(x_i).$$

Here the RV $\hat{\theta}(X_1, \cdots, X_n)$ is called the *maximum likelihood estimator* of θ, and its observed value $\hat{\theta}(x_1, \cdots, x_n)$ is called the *maximum likelihood estimate* of θ.

What is the reasoning behind maximizing the likelihood function as above? The underlying assumption is that, *you are seeing the sample values $x_1, \cdots, x_n$ because it was the most likely to see them!* Namely, suppose we have designed the sampling procedure well-enough so that we get i.i.d. samples $X_1, \cdots, X_n$ each with distribution $f_{X;\theta}$. Then we have

$$L(x_1, \cdots, x_n; \theta) = \mathbb{P}(X_1 = x_1, \cdots, X_n = x_n; \theta).$$

Namely, the likelihood function on the LHS is the probability of observing a sequence of specific sample values $x_1, \cdots, x_n$ under the i.i.d. assumption and assuming the parameter value θ. Hence, MLE chooses $\hat{\theta}$ to be the value of the parameter in Ω under which the probability of obtaining the current sample is maximized.

Example A.9.1 (MLE for Bernoulli RVs)**.** Suppose $X \sim$ Bernoulli(p) for some unknown $p \in [0,1]$. Suppose we have sample values $x_1, \cdots, x_n$ obtained from an i.i.d. sampling for X. Denote $\bar{x} = n^{-1}\sum_{i=1}^{n} x_i$. We will show that the MLE for p is $\bar{X}$, that is,

$$\hat{p} = \bar{X}. \tag{20}$$

Let $X_1, \cdots, X_n$ be i.i.d. with distribution Bernoulli(p). Note that

$$\begin{aligned}\mathbb{P}(X_i = x_i; p) &= \begin{cases} p & \text{if } x_i = 1 \\ 1-p & \text{if } x_i = 0 \end{cases} \\ &= p^{x_i}(1-p)^{1-x_i}.\end{aligned}$$

Hence the likelihood function is given by

$$\begin{aligned}L(x_1, \cdots, x_n; p) &= \prod_{i=1}^{n} \mathbb{P}(X_i = x_i; p) \\ &= \prod_{i=1}^{n} p^{x_i}(1-p)^{1-x_i} \\ &= p^{\sum_{i=1}^{n} x_i}(1-p)^{n-\sum_{i=1}^{n} x_i}.\end{aligned}$$

In order to maximize the likelihood function, it suffices to maximize its logarithm [2]. The *log likelihood function* is given by

$$l(x_1,\cdots,x_n;p) := \log L(x_1,\cdots,x_n;p) = n\bar{x}\log p + (n-n\bar{x})\log(1-p).$$

To maximize the log likelihood function, we take partial derivative in p:

$$\frac{\partial l(x_1,\cdots,x_n;p)}{\partial p} = \frac{n\bar{x}}{p} - \frac{n-n\bar{x}}{1-p}.$$

Setting this equal to zero, we obtain

$$\frac{\bar{x}}{p} = \frac{1-\bar{x}}{1-p}.$$

Rearranging, we obtain $p = \bar{x}$. Hence we have (20) as desired. ▲

Exercise A.9.2 (MLE for Binomial RV)**.** Let $X \sim \text{Binomial}(n,p)$ where n is known but p is not.

(i) Write $x = x_1 + \cdots + x_m$. Show that the log likelihood function is given by

$$l(x_1,\cdots,x_m;p) = \left(\sum_{i=1}^{m} \log\binom{n}{x_i}\right) + x\log p + (mn-x)\log(1-p).$$

(ii) Show that the MLE for p is $\bar{X}/n$.

Exercise A.9.3 (MLE for Geometric RV)**.** Let $X \sim \text{Geom}(p)$, which is a discrete RV with PMF $\mathbb{P}(X = k) = (1-p)^{k-1}p$, $k = 1,2,3,\cdots$.

(i) Show that the log likelihood function is given by

$$l(x_1,\cdots,x_n;p) = n\log p + \left(\left(\sum_{i=1}^{n} x_i\right) - n\right)\log(1-p).$$

(ii) Show that the MLE for p is $1/\bar{X}$.[3]

Exercise A.9.4 (MLE for Poisson RV)**.** Let $X \sim \text{Poisson}(\lambda)$, which is a discrete RV with PMF $\mathbb{P}(X = k) = \lambda^k e^{-\lambda}/k!$, $k = 0,1,2,\cdots$.

(i) Show that the log likelihood function is given by

$$l(x_1,\cdots,x_n;\lambda) = \log\lambda \sum_{i=1}^{n} x_i - n\lambda - \log(x_1!x_2!\cdots x_n!).$$

(ii) Show that the MLE for λ is $\bar{X}$.[4]

Example A.9.5 (MLE for Exponential RVs)**.** Suppose $X \sim \text{Exp}(\lambda)$ for some unknown $\lambda > 0$. We have sample values $x_1,\cdots,x_n$ obtained from an i.i.d. sampling for X. Denote $\bar{x} = n^{-1}\sum_{i=1}^{n} x_i$. We will show that the MLE for λ is $1/\bar{X}$, that is,

$$1/\hat{\lambda} = \bar{X}.$$

Let $X_1,\cdots,X_n$ be i.i.d. with distribution $\text{Exp}(\lambda)$, which have the following PDF

$$f_X(x;\lambda) = \lambda e^{-\lambda x}\mathbf{1}(x \geq 0).$$

[2] since log is a strictly increasing function.

[3] Recall that $\mathbb{E}[\text{Geom}(p)] = 1/p$, so we are saying the MLE for the population mean is $\bar{X}$.

[4] Recall that $\mathbb{E}[\text{Poisson}(\lambda)] = \lambda$, so we are saying the MLE for the population mean is $\bar{X}$.

Noting that $x_1, \cdots, x_n \geq 0$, it follows that the likelihood function is given by

$$\begin{aligned} L(x_1, \cdots, x_n; \lambda) &= \prod_{i=1}^{n} f_X(x_i; \lambda) \\ &= \lambda^n \prod_{i=1}^{n} e^{-\lambda x_i} = \lambda^n e^{-\lambda \sum_{i=1}^{n} \bar{x}} = \lambda^n e^{-\lambda n \bar{x}}. \end{aligned}$$

So the log likelihood function is given by

$$l(x_1, \cdots, x_n; \lambda) = n \log \lambda - \lambda n \bar{x}.$$

To maximize the log likelihood function, we take partial derivative in λ:

$$\frac{\partial l(x_1, \cdots, x_n; \lambda)}{\partial \lambda} = \frac{n}{\lambda} - n\bar{x}.$$

Setting this equal to zero, we obtain $1/\lambda = \bar{x}$. Hence $1/\hat{\lambda} = \bar{X}$, as desired. ▲

Bibliography

[Abb17] Emmanuel Abbe, *Community detection and stochastic block models: recent developments*, The Journal of Machine Learning Research **18** (2017), no. 1, 6446–6531.

[Alo07] Uri Alon, *Network motifs: Theory and experimental approaches*, Nature Reviews Genetics **8** (2007), no. 6, 450–461.

[BA99] Albert-László Barabási and Réka Albert, *Emergence of scaling in random networks*, science **286** (1999), no. 5439, 509–512.

[BDIF14] Ginestra Bianconi, Richard K Darst, Jacopo Iacovacci, and Santo Fortunato, *Triadic closure as a basic generating mechanism of communities in complex networks*, Physical Review E **90** (2014), no. 4, 042806.

[CW03] Gavin C. Conant and Andreas Wagner, *Convergent evolution of gene circuits*, Nature Genetics **34** (2003), no. 3, 264–266.

[EG60] P Erdos and Tibor Gallai, *Graphen mit punkten vorgeschriebenen grades*, Mat. Lapok **11** (1960), 264–274.

[FK16] Alan Frieze and Michał Karoński, *Introduction to random graphs*, Cambridge University Press, 2016.

[GJB+20] David E. Gordon, Gwendolyn M. Jang, Mehdi Bouhaddou, Jiewei Xu, Kirsten Obernier, Kris M. White, Matthew J. O'Meara, Veronica V. Rezelj, Jeffrey Z. Guo, and Danielle L. Swaney, *A SARS-CoV-2 protein interaction map reveals targets for drug repurposing*, Nature **583** (2020), 1–13.

[GL16] Aditya Grover and Jure Leskovec, NODE2VEC*: Scalable feature learning for networks*, 2016, pp. 855–864.

[GYA18] Jonathan L Gross, Jay Yellen, and Mark Anderson, *Graph theory and its applications*, Chapman and Hall/CRC, 2018.

[HlHbCfP14] Xu Hong-lin, Yan Han-bing, Gao Cui-fang, and Zhu Ping, *Social network analysis based on network motifs*, Journal of Applied Mathematics **2014** (2014), 874708.

[IB20] Ashif Sikandar Iquebal and Satish Bukkapatnam, *Consistent estimation of the max-flow problem: Towards unsupervised image segmentation*, IEEE Transactions on Pattern Analysis and Machine Intelligence **44** (2020), no. 5, 2346–2357.

[JKG09] Krzysztof Juszczyszyn, Przemyslaw Kazienko, and Bogdan Gabrys, *Temporal changes in local topology of an email-based social network*, Computing and Informatics **28** (2009), no. 6, 763–779.

[JKLP93] Svante Janson, Donald E Knuth, Tomasz Luczak, and Boris Pittel, *The birth of the giant component*, Random Structures & Algorithms **4** (1993), no. 3, 233–358.

[LK20] Jure Leskovec and Andrej Krevl, *SNAP Datasets: Stanford Large Network Dataset Collection*, Available at http://snap.stanford.edu/data, 2020.

[LP17] David A. Levin and Yuval Peres, *Markov chains and mixing times*, American Mathematical Society, Providence, RI, USA, 2017.

[MCCD13] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean, *Efficient estimation of word representations in vector space*, arXiv preprint arXiv:1301.3781 (2013).

[MFD20] Filippo Menczer, Santo Fortunato, and Clayton A Davis, *A first course in network science*, Cambridge University Press, 2020.

[MNRS00] Andrew Kachites McCallum, Kamal Nigam, Jason Rennie, and Kristie Seymore, *Automating the construction of internet portals with machine learning*, Information Retrieval **3** (2000), 127–163.

[MSC+13] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean, *Distributed representations of words and phrases and their compositionality*, Advances in neural information processing systems **26** (2013).

[NA13] Jose C Nacher and Tatsuya Akutsu, *Structural controllability of unidirectional bipartite networks*, Scientific reports **3** (2013), no. 1, 1647.

[OSB+19] Rose Oughtred, Chris Stark, Bobby-Joe Breitkreutz, Jennifer Rust, Lorrie Boucher, Christie Chang, Nadine Kolas, Lara O'Donnell, Genie Leung, and Rochelle McAdam, *The BioGRID interaction database: 2019 update*, Nucleic Acids Research **47** (2019), no. D1, D529–D541.

[OTT10] Takaaki Ohnishi, Hideki Takayasu, and Misako Takayasu, *Network motifs in an inter-firm network*, Journal of Economic Interaction and Coordination **5** (2010), no. 2, 171–180.

[PARS14] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena, *Deepwalk: Online learning of social representations*, Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining, 2014, pp. 701–710.

[RKMP11] Veronica Red, Eric D. Kelsic, Peter J. Mucha, and Mason A. Porter, *Comparing community structure to characteristics in online collegiate social networks*, SIAM Review **53** (2011), 526–543.

[RMLF10] Jason M. K. Rip, Kevin S. McCann, Denis H. Lynn, and Sonia Fawcett, *An experimental test of a fundamental food web motif*, Proceedings of the Royal Society B: Biological Sciences **277** (2010), no. 1688, 1743–1749.

[RPD14] Konstantin Ristl, Sebastian J. Plitzko, and Barbara Drossel, *Complex response of a food-web module to symmetric and asymmetric migration between several patches*, Journal of Theoretical Biology **354** (2014), 54–59.

[SKF04] Olaf Sporns, Rolf Kötter, and Karl J. Friston, *Motifs in brain networks*, PLoS Biology **2** (2004), no. 11, e369.

[SM00] Jianbo Shi and Jitendra Malik, *Normalized cuts and image segmentation*, IEEE Transactions on pattern analysis and machine intelligence **22** (2000), no. 8, 888–905.

[SV] Timo Seppalainen and Benedek Valko, *Introduction to stochastic processes*.

[the20] theBiogrid.org, *Coronavirus* PPI *network*, Retrieved from https://wiki.thebiogrid.org/doku.php/covid (downloaded 24 July 2020, Ver. 3.5.187.tab3).

[TKWH18] Frank W. Takes, Walter A. Kosters, Boyd Witte, and Eelke M. Heemskerk, *Multiplex network motifs as building blocks of corporate networks*, Applied Network Science **3** (2018), no. 1, 39.

[TMP12] Amanda L. Traud, Peter J. Mucha, and Mason A. Porter, *Social structure of Facebook networks*, Physica A **391** (2012), no. 16, 4165–4180.

[TVW10] Amitabha Tripathi, Sushmita Venugopalan, and Douglas B West, *A short constructive proof of the erdős–gallai characterization of graphic lists*, Discrete mathematics **310** (2010), no. 4, 843–844.

[VDH09] Remco Van Der Hofstad, *Random graphs and complex networks*, Available on http://www. win. tue. nl/rhofstad/NotesRGCN. pdf **11** (2009), 60.

[WS98] Duncan J Watts and Steven H Strogatz, *Collective dynamics of 'small-world'networks*, nature **393** (1998), no. 6684, 440–442.

[YBT10] Jing Yuan, Egil Bae, and Xue-Cheng Tai, *A study on continuous max-flow and min-cut approaches*, 2010 ieee computer society conference on computer vision and pattern recognition, IEEE, 2010, pp. 2217–2224.

[ZCH+20] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun, *Graph neural networks: A review of methods and applications*, AI open **1** (2020), 57–81.