1 Network Elements

node: (*n.*) a point in a network or diagram at which lines or pathways intersect or branch.

Having seen several examples of real networks in Chapter 0, let us now learn about the basic definitions and quantities that allow us to describe a network.

1.1 Basic Definitions

In very general terms a network, or graph, is a set of elements, which we call *nodes*, along with a set of connections between pairs of nodes, which we call *links*. The links represent the presence of a relation among the elements represented by the nodes. As we have seen earlier, links can corresponds to social, physical, communication, geographic, conceptual, chemical, biological, or other interactions. We say that two nodes are *adjacent* or *connected* if there is a link between them. It is also common to call connected nodes *neighbors*.

Box 1.1 Definition of a network

A network G has two parts, a set of N elements, called *nodes* or *vertices*, and a set of L pairs of nodes, called *links* or *edges*. The link (i, j) joins the nodes i and j. A network can be *undirected* or *directed*. A directed network is also called a *digraph*. In directed networks, links are called *directed links* and the order of the nodes in a link reflects the direction: the link (i, j) goes from the source node i to the target node j. In undirected networks, all links are bi-directional and the order of the two nodes in a link does not matter. A network can be *unweighted* or *weighted*. In a weighted network, links have associated *weights*: the *weighted link* (i, j, w) between nodes i and j has weight w. A network can be both directed and weighted, in which case it has directed weighted links.

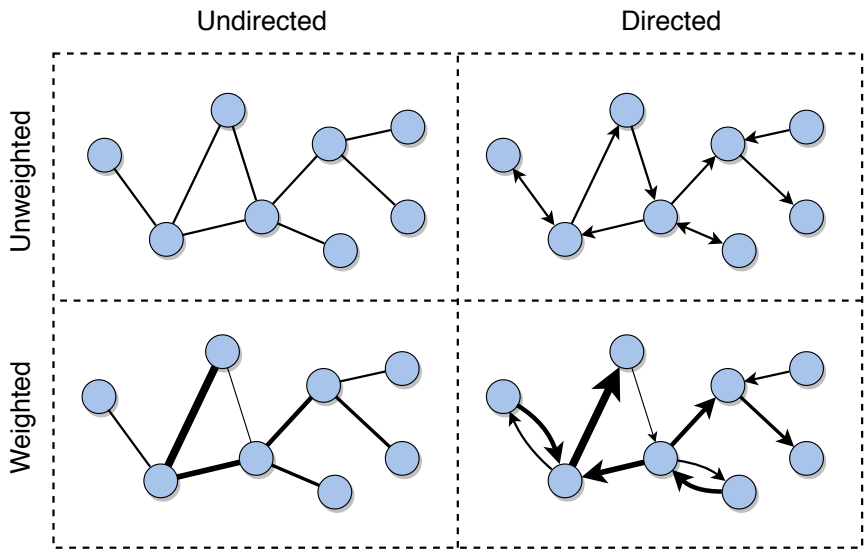

Fig. 1.1 Graphical representations of undirected, directed, and weighted networks. The circles represent the nodes. Pairs of adjacent nodes are connected by a line (link) or arrow (directed link). Arrows indicate the direction of the links. The thickness of a link represents its weight in weighted networks.

Networks provide a general theoretical framework allowing for a convenient conceptual representation of interrelations in a wide array of systems; we have seen several examples in Chapter 0. The study of networks has a long tradition in mathematics, computer science, sociology, and communication research. Recently, networks have also been studied intensely in physics and biology. Different fields concerned with networks often introduced their own nomenclature. For example, in some fields a network is called a *graph*, a node is referred to as a *vertex* and a link is an *edge*. (We will occasionally use these terms.) The rigorous language for the description of networks is found in graph theory, a field of mathematics that can be traced back to the pioneering work of Leonhard Euler in the 18th century. Here we do not want to provide a rigorous introduction to graph theory. We are mostly interested in building a vocabulary and introducing a set of basic notions that will allow us to take our first steps into the world of networks. However, sometimes a formal notation is helpful. In these cases we will include the formal notation in a shaded area or in a box. For example, a more rigorous definition of a network is provided in Box 1.1. In the following chapters we will introduce additional concepts and definitions as needed to analyze real-world systems.

Each network is characterized by the total number of nodes N and the total number of links L. We call N the *size* of the network because it identifies the number of distinct elements composing the system. The numbers of nodes and

links do not suffice in defining a network; we have to specify the way in which the nodes are connected by the links.

There are different types of links, which define different classes of networks. In some networks, such as Facebook (Figure 0.1), the links do not have a direction and we represent them as line segments. We call these networks *undirected*. In other cases, such as Wikipedia (Figure 0.5), links are directed and we represent them as arrows. Networks with directed links are called *directed networks*. We say more about directed networks in Section 1.6 and Chapter 4.

In some cases, such as air transportation networks (Figure 0.7), links have associated weights. These are called *weighted networks*. A network can be both directed and weighted. The email network is an example of weighted directed network, in which link weights and directions represent communication traffic (number of messages) between nodes. We will return to weighted networks in Section 1.7 and Chapter 4. Figure 1.1 provides illustrations of undirected, directed, and weighted networks.

There are several other classes of networks. A network might have more than one type of nodes. For example, the movie-star network (Figure 0.2(a)) has two types of nodes representing movies and people. In this network, a link connects an actor or actress to a movie, but there are no links among people or among movies. This is an instance of a so-called *bipartite network*. In a bipartite network, there are two groups of nodes such that links only connect nodes from different groups and not nodes from the same group. Other examples of bipartite networks include those that capture the relationships between songs and artists, between classes and students, and between products and customers. More on bipartite networks in Chapter 4.

A network might have multiple types of links, in which case it is called a *multiplex* network. To use the movie-star example again, we could imagine adding links between actors and/or actresses who are married to each other. In the example of the Wikipedia (Figure 0.5), in addition to the hyperlinks, we might have weighted links representing clicks from Wikipedia users, and/or undirected links between articles that share editors. These and other more complex types of networks are further discussed in Section 1.8.

1.2 Handling Networks in Code

To manage, analyze, and visualize networks with more than a handful of nodes and links, we need to use software tools or write our own code. There are many network analysis and visualization tools, as well as libraries to handle networks in many programming languages. Throughout the book we will occasionally mention a couple of these tools. For instance, the visualizations in Chapter 0 are generated with an application called *Gephi*. However, we believe that to get a hands-on understanding of networks it is necessary to "get our hands dirty" and write some code. We assume that students using this book have some familiarity with Python,

a popular programming language among both novice and expert coders.[1] To make our life easier, we will use *NetworkX* (`networkx.github.io`), a Python package for the creation, manipulation, and study of the structure, dynamics, and function of networks. NetworkX provides data structures, algorithms, measures, and generators for networks, as well as rudimentary visualization facilities.[2]

Once we import NetworkX, we can easily create an undirected network (a "Graph") and add a few nodes and links. Nodes are referred by integer IDs and links are called edges:

```
import networkx as nx # always import NetworkX first!
G = nx.Graph()
G.add_node(1)
G.add_node(2)
G.add_edge(1,2)
```

We can add several nodes or links at once:

```
G.add_nodes_from([3,4,5,...])
G.add_edges_from([(3,4),(3,5),...])
```

Here is how we get lists of nodes, links, and neighbors of a given node:

```
G.nodes()
G.edges()
G.neighbors(3)
```

And here is how you loop over nodes or links:

```
for n in G.nodes:
    print(n, G.neighbors(n))
for u,v in G.edges:
    print(u, v)
```

Similarly, we can create a directed network ("DiGraph"):

```
D = nx.DiGraph()
D.add_edge(1,2)
D.add_edge(2,1)
D.add_edges_from([(2,3),(3,4),...])
```

Note that the link from node **1** to node **2** is distinct from the link from **2** to **1** because this network is directed. Also note that when we add a link, the nodes are added automatically if they don't already exist. This is convenient. There are functions for getting the size and number of links:

```
D.number_of_nodes()
D.number_of_edges()
```

In a directed network, when we ask for the neighbors of a node, we get both the

[1] We offer an introductory tutorial on Python in Appendix A; it can also be downloaded from the book's Github repository at `github.com/CambridgeUniversityPress/FirstCourseNetworkScience`.

[2] We offer an introductory tutorial on NetworkX on the book's Github repository.

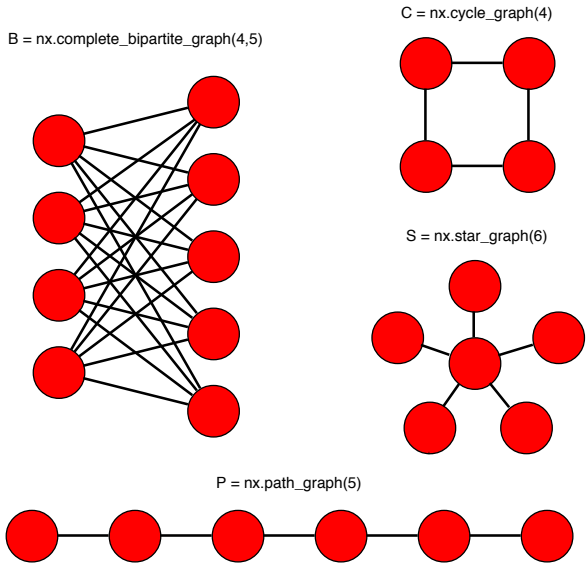

Fig. 1.2 A few simple networks generated by NetworkX functions: complete bipartite (B), cycle (C), star (S) and path (P). The concept of a *complete* network is introduced in the next section.

nodes linking to and from it. But there are also functions to only get the edges linking to and from, respectively called predecessors and successors:

```
D.neighbors(2)
D.predecessors(2)
D.successors(2)
```

Finally, there are functions to generate networks of many types. Typically these functions need arguments that specify the number of nodes or links. Here is code to generate a few networks, shown in Figure 1.2:

```
B = nx.complete_bipartite_graph(4,5)
C = nx.cycle_graph(4)
P = nx.path_graph(5)
S = nx.star_graph(6)
```

We strongly recommend that you read the NetworkX tutorial[3] and bookmark its documentation.[4] And remember, Google and Stack Overflow are your friends when you are stuck!

[3] networkx.github.io/documentation/stable/tutorial.html
[4] networkx.github.io/documentation/stable/

1.3 Density and Sparsity

The maximum number of links in a network is bounded by the possible number of distinct connections among the nodes of the system. The maximum number of links is therefore given by the number of pairs of nodes. A network with the maximum number of links, in which all possible pairs of nodes are connected by links, is called a *complete network*.

The maximum number of links in an undirected network with N nodes is the number of distinct pairs of nodes:

$$L_{max} = \binom{N}{2} = N(N-1)/2. \tag{1.1}$$

Intuitively each node can connect to $N-1$ other nodes, and there are N of them. However, that would count each pair twice, so we divide by two. In a directed network, each pair of nodes should be counted twice, one for each direction, so $L_{max} = N(N-1)$. Counting the possible pairs of objects among a set of N objects is something that we will encounter again later in the book. Mathematicians have a name for the formula $\binom{N}{2}$: "N choose two."
A bipartite network is *complete* if each node in one group is connected to all nodes in the other group (see example B in Figure 1.2). In this case $L_{max} = N_1 \times N_2$, where N_1 and N_2 are the sizes of the two groups.

The fraction of possible links that actually exist, which is the same as the fraction of pairs of nodes that are actually connected, is called the *density* of the network. A complete network has maximal density: one. However, the actual number of links is typically much smaller than the maximum, as most pairs of nodes are not directly connected to each other. Therefore the density is often much, much smaller than one — by orders of magnitude in most real-world, large networks. This is an important feature that helps in dealing with network structure. We call it *sparsity*. Intuitively the less edges are in a network, the sparser it is.

The density of a network with N nodes and L links is

$$d = L/L_{max}. \tag{1.2}$$

In an undirected network this is given by

$$d = L/L_{max} = \frac{2L}{N(N-1)} \tag{1.3}$$

and in a directed network the density is

$$d = L/L_{max} = \frac{L}{N(N-1)}. \tag{1.4}$$

In a complete network, $d = 1$ by definition, since $L = L_{max}$. In a sparse network,

Table 1.1 Basic statistics of network examples. Network types can be (D)irected and/or (W)eighted. When there is no label the network is undirected and unweighted. For directed networks, we provide the average in-degree (which coincides with the average out-degree).

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

$L \ll L_{max}$ and therefore $d \ll 1$. When a network grows very large, we can observe how the number of links increases as a function of the number of nodes. We say that the network is sparse if the number of links grows proportionally to the number of nodes ($L \sim N$), or even slower. If instead the number of links grows faster, *e.g.,* quadratically with network size ($L \sim N^2$), then we say that the network is dense.

To illustrate the importance of network sparsity, let us consider the example of Facebook. At the time when this book is being written, Facebook has around 2 billion users ($N \approx 2 \times 10^9$). If this was a complete network, there would be $L \approx 10^{18}$ links — that is a number with 18 zeroes, and there is no way to store so much data! But fortunately social networks are very sparse and Facebook is no exception. Each user has on average a thousand friends or less, so that the density is approximately $d \approx 10^{-6}$. That is still a lot of data, but Facebook can manage it.

Table 1.1 presents basic statistics about size and density of the network examples illustrated in Chapter 0.[5] Although these networks are very different from each other, they are all sparse.

NetworkX makes it easy to measure the density of directed and undirected networks:

```
nx.density(G)
nx.density(D)
CG = nx.complete_graph(8471) # a large complete network
```

[5] Datasets for these networks are available in the book's Github repository: `github.com/CambridgeUniversityPress/FirstCourseNetworkScience`

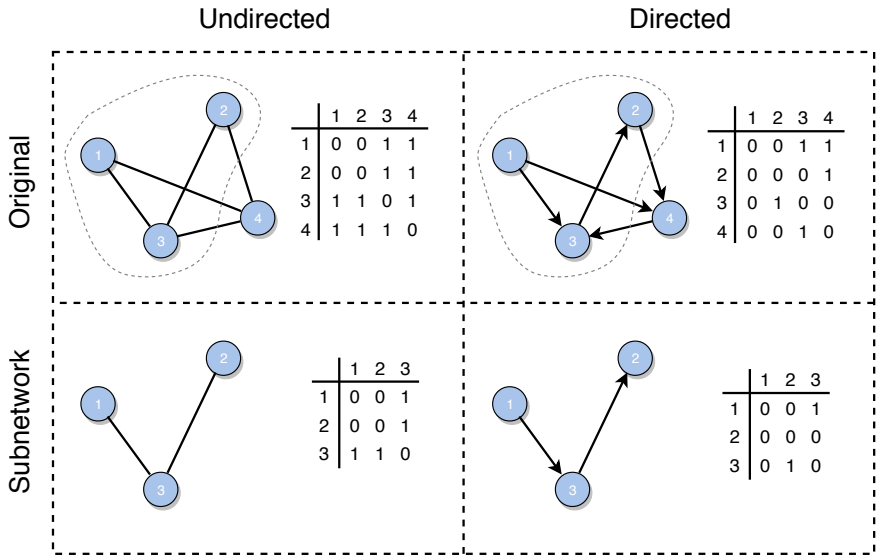

Fig. 1.3 Network and subnetwork examples. We also show the adjacency matrix representation of each network (see Section 1.9).

```
print(nx.density(CG))          # no need for a calculator!
```

1.4 Subnetworks

In many cases, we are interested in a subset of a network, which is itself a network and is called a *subnetwork* (or *subgraph*). A subnetwork is obtained by selecting a subset of the nodes and *all* of the links among these nodes.

Figure 1.3 provides some illustrations of subnetworks of undirected and directed networks. The abundance of certain types of subnetworks and their properties are important in the characterization of real networks. As an example, a *clique* is a complete subnetwork: a subset of nodes all linked to each other. Any subnetwork of a complete network is a clique because all pairs of nodes in the network are connected and therefore all pairs of nodes in any subnetwork are also connected.

A special type of subnetwork is the *ego network* of a node, which is the subnetwork consisting of the chosen node — called the *ego* — and its neighbors. Ego networks are often studied in social network analysis.

Using NetworkX we can generate a subnetwork of a given network by specifying a subset of nodes:

```
K5 = nx.complete_graph(5)
```

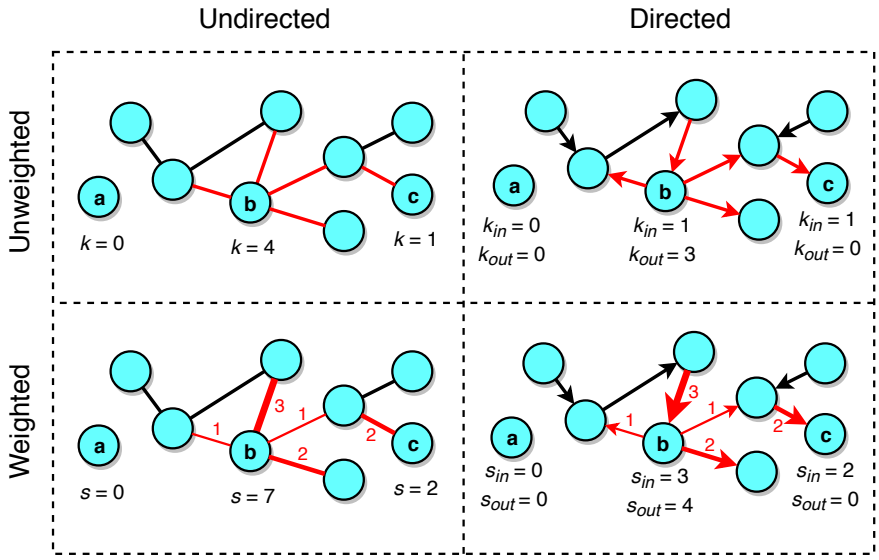

Fig. 1.4 Illustrations of degree and strength in directed, undirected, weighted, and unweighted networks. The links of nodes **a**, **b**, and **c** along with their weights are highlighted in red, and their degrees or strengths are shown.

```
clique = nx.subgraph(K5, (0,1,2))
```

1.5 Degree

The *degree* of a node is its number of links, or of neighbors. We use k_i to denote the degree of node i. Figure 1.4 illustrates the degree of a few nodes in an undirected network. A node with no neighbors, such as node **a** in the figure, has degree zero ($k = 0$) and is called a *singleton*.

The average degree of a network is denoted by $\langle k \rangle$. It is an important property and is related (directly proportional) to its density.

The average degree of a network is defined as

$$\langle k \rangle = \frac{\sum_i k_i}{N}. \tag{1.5}$$

Since each link contributes to the degree of two nodes in an undirected network, the numerator of Eq. 1.5 can be written as $2L$. From the definition of density for

an undirected network (Eq. 1.3), $2L = dN(N-1)$. Therefore

$$\langle k \rangle = \frac{2L}{N} = \frac{dN(N-1)}{N} = d(N-1) \tag{1.6}$$

and conversely

$$d = \frac{\langle k \rangle}{N-1}. \tag{1.7}$$

This makes sense: the maximum possible degree of a node is $k_{max} = N-1$, obtained when the node is connected to every other node. Intuitively, the density is the ratio between the average and maximum degree.

Table 1.1 shows the average degree of the network examples illustrated in Chapter 0. NetworkX has a function that returns the degree of a given node. Without arguments, it returns a dictionary with the degree of each node:

```
G.degree(2) # returns the degree of node 2
G.degree()  # returns the degree of all nodes of G
```

In Chapter 3 we will see that the degrees of a network's individual nodes are very important properties to characterize the structure of the network. So far we have defined the degree in undirected networks. Next we extend the definition to directed and weighted networks.

1.6 Directed Networks

In the graphical representation of a network, the directed nature of the links is depicted by means of an arrow, indicating the direction of each link. The main difference between directed and undirected networks is represented in Figure 1.1. In an undirected network, the presence of a link between two nodes connects the adjacent nodes in both directions. On the other hand, the presence of a link in a directed network does not necessarily imply the presence of a link in the opposite direction. This fact has important consequences for the connectedness of a directed network, as will be discussed in more detail in Chapter 2.

When we consider the degree of a node in a directed network, we have to think of incoming and outgoing links separately. The number of incoming links, or predecessors, of node i is called *in-degree* and denoted by k_i^{in}. And the number of outgoing links, or successors, is called *out-degree* and denoted by k_i^{out}. Figure 1.4 illustrates the in- and out-degree of a few nodes in a directed network.

We already defined the density for a directed network (Eq. 1.4). We can define average in-degree and average out-degree similarly to Eq. 1.5.

NetworkX has functions that return the in-degree and out-degree of a given node.

If the network is directed, the `degree` function returns the total degree, which is the sum of in-degree and out-degree:

```
D.in_degree(4)
D.out_degree(4)
D.degree(4)
```

1.7 Weighted Networks

In the graphical representation of a network, the weighted nature of the links is depicted by means of lines of different width, indicating the weight of each link. A weight of zero is equivalent to the absence of a link. The main difference between weighted and unweighted networks is represented in Figure 1.1.

A weighted network can be directed or undirected; let us first assume the simpler case of an undirected weighted network. We can measure the degree of a node in a weighted network by disregarding the weights. However, it may be important to consider the weights. We can therefore define the *weighted degree*, or *strength* of a node, as the sum of the weights of its links. Similarly, we can define *in-strength* and *out-strength* in the case of a directed weighted network. Both cases are illustrated in Figure 1.4.

The weighted degree, or *strength*, of a node i in an undirected weighted network is denoted by:

$$s_i = \sum_j w_{ij} \tag{1.8}$$

where w_{ij} is the weight of the link between nodes i and j. We assume $w_{ij} = 0$ if there is no link between i and j. We can analogously generalize in-degree and out-degree to in-strength and out-strength in a directed weighted network:

$$s_i^{in} = \sum_j w_{ji} \tag{1.9}$$

$$s_i^{out} = \sum_j w_{ij} \tag{1.10}$$

where w_{ij} is the weight of the directed link from i to j.

In NetworkX, both graphs and digraphs can have "weight" attributes attached to links. When adding multiple weighted links, each is specified as a triple where the third element is the weight:

```
W = nx.Graph()
W.add_edge(1,2,weight=6)
W.add_weighted_edges_from([(2,3,3),(2,4,5)])
```

We can get a list of links with their associated weight data, for example if we need to print the links with large weights:

```
for (i,j,w) in W.edges(data='weight'):
    if w > 3:
        print('(%d, %d, %d)' % (i,j,w)) # skip link (2,3)
```

Finally, we can get the strength of a given node using the `degree` function and specifying the weight attribute:

```
W.degree(2, weight='weight') # strength of node 2
                             # is 6 + 3 + 5 = 14
```

1.8 Multilayer and Temporal Networks

In the US air transportation network of Figure 0.7, the links represent direct flights between airports, regardless of which particular airlines operates those flights. But classifying the flights according to their respective airlines is valuable in a number of situations. We may wish to predict the propagation of scheduling delays through an airline's network, or investigate the consequences of such delays on the movement of passengers. In fact, each commercial airline tries to reschedule passengers on its own flights first because it is expensive to rebook them on another company's flights. Therefore the air transportation network of a specific airline has its own identity, even though it is intertwined with the networks of other airlines. In these cases it is beneficial to represent the system as a *multilayer network*, *i.e.*, a combination of layers, where each layer is the air transportation network of a specific airline: the nodes are the airports, the links flights operated by the same company.

If each layer in a multilayer network is built upon the same set of nodes, the network is called a *multiplex*. The air transportation network is an example of multiplex. Another example is a social network in which the different layers represent different types of social relationships. For example, one layer could represent friendship ties, another layer family ties, another coworker ties, and so on. The nodes in each layer represent the same individuals.

A *temporal network* is a special case of multiplex. Links are dynamic, in that the respective node-node interactions occur at different times. Nodes may also have a dynamic character, in that they may appear and disappear at different stages of the network evolution. For instance, networks of user activity on Twitter are temporal because posts, retweets, and mentions occur at different times, which can be identified by their timestamps. We can divide the time span of a temporal network into consecutive intervals: all nodes and links existing during each interval constitute a *snapshot* of the system. Each snapshot can be interpreted as one layer of a multiplex, as illustrated in Figure 1.5.

In a multilayer network there are *intralayer links*, connecting pairs of nodes in the same layer, and *interlayer links*, connecting pairs of nodes in different layers. In

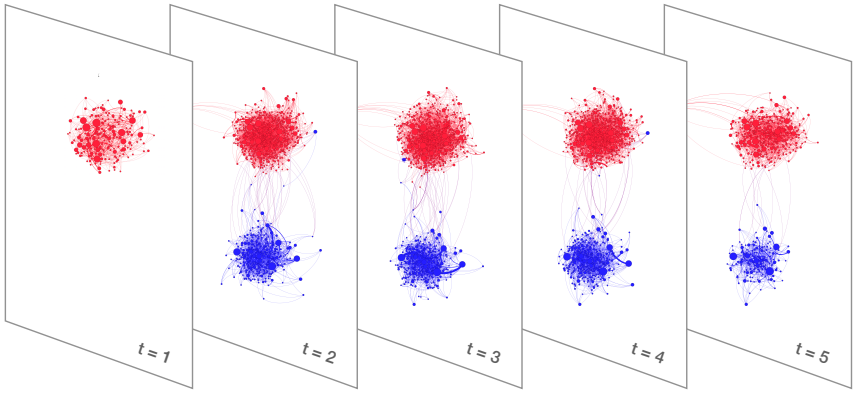

Fig. 1.5 Temporal network of political retweets. Each snapshot includes retweet links with timestamps in a particular time interval. By aggregating these snapshots over time we obtain the static network shown in Figure 0.3.

the special case of multiplex networks, interlayer links connect each node of a layer with its counterpart in the other layers. Such links are called *couplings*, because they couple copies of the same node in different layers.

Traditionally, multiplex networks have been analyzed by aggregating data from the different layers and then studying the resulting network. For instance, the networks of Figure 0.7 and Figure 0.3 are aggregations of multiplex networks corresponding to different airlines and time intervals, respectively. The aggregated network is typically weighted, even when the links of the multiplex are not, because there are usually multiple links joining the same pair of nodes in different layers, which turn into a single weighted link in the aggregated system. For example, the links in Figure 0.3 are weighted by the number of times a user retweets another. But aggregation discards a lot of valuable information provided by the original multilayer system. In the air transportation case, merging networks corresponding to different airlines prevents us from studying transitions of passengers between such networks, which may become necessary in case of strikes or technical problems affecting a specific airline.

In general, each layer could be characterized by its own set of nodes and links. Therefore layers may represent entirely different graphs and the resulting system is a *network of networks*. Here, interlayer links may represent dependency relationships between the nodes of the networks. Consider the electrical power grid, which connects power generating stations and demand centers through high voltage transmission lines. The stations are controlled by computers that monitor and manage the production and transmission of electricity. These computers are connected through the Internet. In turn, Internet routers depend on power stations for

their electricity supply. Therefore we have a system with two coupled networks: the power grid and the Internet.

In such a coupled system, one network can affect the other to optimize delivery; the grid can be reconfigured to reroute power when needed. However, this kind of network of networks can also introduce unpredictable vulnerabilities. A software problem or attack can take down one or more nodes in the power grid, and without electricity the Internet in an area could also go down, leading to failures of other nodes and, in an extreme case, a catastrophic domino effect called a *cascading failure* affecting a large portion of the grid. For these reasons, networks of networks are the subject of intense study.

To keep things simple, in this book we mainly focus on networks with a single type of node and a single type of link. In an undirected network, we will assume that there can be at most one link connecting a pair of nodes. (If the network is directed, there can be two links, one in each direction, as shown in Figure 1.1.) In addition, we will not consider *self-loops*, or links connecting a node to itself; we will assume that each link connects two distinct nodes.

1.9 Network Representations

To store/retrieve a network in/from a computer file or memory, we need a way to formally represent its nodes and links. There are several possible network representations. The simplest is the *adjacency matrix*, an $N \times N$ matrix in which each element represents the link between the nodes indexed by the corresponding row and column.

Element a_{ij} of the adjacency matrix represents the link between nodes i and j. $a_{ij} = 1$ if i and j are adjacent, $a_{ij} = 0$ otherwise.

In Figure 1.3 we show the graphical illustrations of different undirected and directed networks and their corresponding adjacency matrices.

For undirected networks the adjacency matrix is symmetric: we can swap rows and columns and the matrix does not change. Therefore half of the matrix contains redundant information. For directed networks, the adjacency matrix is not symmetric. For unweighted networks, the elements take only values one or zero to indicate the presence or absence of a link, respectively. For weighted networks, matrix elements can take any values corresponding to the link weights. We have already encountered the adjacency matrix elements for weighted networks (the w_{ij} in Eqs. 1.8, 1.9 and 1.10).

In NetworkX, we can get and print adjacency matrices and use the matrix representation to get and set link attributes:

```
print(nx.adjacency_matrix(G)) # graph
G.edge[3][4]
```

```
G.edge[3][4]['color']='blue'
print(nx.adjacency_matrix(D)) # digraph
D.edge[3][4]
D.edge[4][3] # not the same as the previous one
print(nx.adjacency_matrix(W)) # weighted graph
W.edge[2][3]
W.edge[2][3]['weight'] = 2
```

While the adjacency matrix representation matches the mathematical formalism of networks, it is not efficient for storing real networks, which are typically large and sparse. The required storage space grows like the square of the network size (N^2), but if the network is sparse, most of this space is wasted storing zeros (non-existing links). With large sparse networks, a more compact network representation is the *adjacency list*, a data structures that stores the list of neighbors for each node. Adjacency lists represent sparse networks efficiently because the non-existing links are ignored; only the existing links (non-zero values of the adjacency matrix) are considered.

NetworkX provides facilities to loop over a network's adjacency list and retrieve links and their attributes. For example, here is one way to print the neighbors of each node:

```
for n,neighbors in G.adjacency():
    for number,link_attributes in neighbors.items():
        print('(%d, %d)' % (n,number))
```

A third, equally efficient network representation is the *edge list*, which lists each link as a pair of connected nodes. We may also need to list the nodes separately in case of singletons, which would not appear in any of the pairs. In the case of weighted networks, each link is represented as a triple, where the third element is the weight.

In this book we will use the edge list representation to store networks. NetworkX has functions to write and read network files using this representation. You can view the format of an edge list file for yourself:

```
nx.write_edgelist(G, "file.edges")
G2 = nx.read_edgelist("file.edges")       # G2 same as G
nx.write_weighted_edgelist(W, "wf.edges") # store weights
with open("wf.edges") as f:
    for line in f:
        print(line)
W2 = nx.read_weighted_edgelist("wf.edges") # W2 same as W
```

1.10 Drawing Networks

We can learn a lot about a network by drawing it and inspecting its graphical representation. This requires a *network layout algorithm* to place each node on a

plane. (There are also sophisticated 3-D layouts, but we do not discuss them in this book.) There are many layout algorithms that are appropriate for representing different kinds of networks; for example, we used a *geographic layout* to draw the air transportation network in Figure 0.7. For relatively small networks, layouts that place nodes along concentric circles or layers can reveal important hierarchical structure. The most popular class of network layout algorithms are *force-directed layout algorithms*, which are used to visualize most of the example networks in Chapter 0. The inset of Figure 0.7 uses a force-directed layout as well.

The goals of a force-directed layout algorithm are to place the nodes so that connected nodes are positioned close to each other, all the links are of similar length, and the number of link crossings is minimized. To get an idea of how force-directed layout works, imagine a force that repels any two nodes from each other, like the force between two particles with the same electrical charge. Further imagine a spring connecting any two linked nodes, generating an attractive force when they are too far from each other. Force-directed layout algorithms simulate such a physical system so that nodes move to minimize the energy of the system: connected nodes will move toward each other and away from nodes not connected to them.

The result is not only an aesthetically pleasing drawing, but also, sometimes, a visualization of the most obvious communities in the network, as we have seen in Chapter 0. For example, in Figure 0.3, because people in a community (progressive or conservative) are densely connected to each other, they end up clustered together in the layout.

NetworkX has a function to draw a network, which uses a rudimentary network layout algorithm:

```
import matplotlib.pyplot
nx.draw(G)
```

Note that drawing requires a plot interface, such as Matplotlib's. This works reasonably well for small networks with, say, less than a hundred nodes. For larger networks, there are better visualization tools. The examples in Chapter 0 are visualized with Gephi's *ForceAtlas2* layout algorithm.

1.11 Summary

We have presented some basic definitions and quantities that allow us to describe a network:

1 A network is made of two sets of elements: the nodes and links connecting pairs of nodes.
2 A subnetwork is a subset of the network including some of its nodes and all of the links among them.
3 In directed networks, links have a direction. There may be a link from node **1**

to node **2**, and not necessarily one from **2** to **1**. In undirected networks, links are reciprocal.

4 In weighted networks, links have associated weights that represent connection attributes like importance, similarity, distance, traffic, etc. In unweighted networks, all links are the same.

5 Multilayer networks have different types of nodes and links, divided into interconnected layers. If the nodes are the same in each layer, the multilayer network is called a multiplex.

6 The density of a network is the fraction of node pairs that are connected. A network is complete if all pairs of nodes are connected, so that the density is one. Most real networks are sparse, meaning that they have very small density.

7 The degree of a node is the number of neighbors. In directed networks, nodes have in-degree and out-degree measuring the number of incoming and outgoing links, respectively. If the network is weighted, the strength of a node is the sum of the weights of its links. The nodes of weighted directed networks have in-strength and out-strength.

8 Adjacency lists and edge lists are efficient representations to store sparse networks.

9 NetworkX is a popular and convenient programming library to code networks in the Python language.

The definitions in this chapter form a basic vocabulary for network science. More quantities and properties will be introduced in future chapters so that we can describe, analyze, and model real networks and learn what they tell us about the underlying systems and phenomena.

1.12 Further Readings

There are several other excellent textbooks on network science to go beyond the introductory material in this book. Caldarelli and Chessa (2016) dig a bit deeper into the data science of several case studies. If you are interested in branching into physics, consider the textbook by Barabási (2016); or if you want to explore the connections to economics and sociology, we recommend the textbook by Easley and Kleinberg (2010). For more advanced physics, math, and social science topics, there are many books to choose from (Wasserman and Faust, 1994; Caldarelli, 2007; Barrat et al., 2008; Newman, 2018; Cohen and Havlin, 2010; Bollobás, 2012; Dorogovtsev and Mendes, 2013; Latora et al., 2017).

Kivelä et al. (2014) and Boccaletti et al. (2014) have provided influential reviews on multilayer networks. Temporal networks are reviewed by Holme and Saramäki (2012). Gao et al. (2012) analyze networks of networks. Catastrophic failure in these networks is discussed by Reis et al. (2014); Radicchi (2015).

For background on network drawing, see Di Battista et al. (1998). Force-directed network layout (also known as spring layout) algorithms were introduced by Eades (1984) and improved by Kamada and Kawai (1989); Fruchterman and Reingold (1991). The ForceAtlas2 layout algorithm, used for many visualizations in this book, was developed by Jacomy et al. (2014).

Exercises

1.1 Go through the Chapter 1 Tutorial on the book's Github repository.[6]

1.2 Consider a network with N nodes. Given a single link, what is the maximum number of nodes that link can connect? Given a single node, what is the maximum number of links that can connect to that node?

1.3 Consider the road map in Figure 0.9. The grid-like structure of this network means that most nodes have the same degree. What is the most common degree for nodes in this network?

1.4 Consider the road map in Figure 0.9. Manhattan has a lot of one-way streets. This implies that a good network model of traffic flow would probably have directed links. Consider a subgraph of this network with grid-like connectivity and all one-way streets (*i.e.,* each node is a 4-way intersection of two one-way streets). What is the most common in-degree of nodes in this subgraph? What is the most common out-degree?

1.5 What network quantity can we use to represent the volume of traffic between each pair of adjacent intersections in the Manhattan road map (Figure 0.9)?

1.6 Consider a directed network of N nodes. Now consider the total in-degree, *i.e.,* the sum of the in-degree over all nodes in the network. Compare this to the analogous total out-degree. Which of the following must hold true for any such network?

- **a.** Total in-degree must be less than total out-degree
- **b.** Total in-degree must be greater than total out-degree
- **c.** Total in-degree must be equal to total out-degree
- **d.** None of these hold true in all instances

1.7 Consider a Twitter retweet network, where users are nodes and we want to show how many times a given user has retweeted another user. What link type best captures this relation?

- **a.** Undirected, unweighted
- **b.** Undirected, weighted
- **c.** Directed, unweighted
- **d.** Directed, weighted

[6] `github.com/CambridgeUniversityPress/FirstCourseNetworkScience`

1.8 Consider a hashtag co-occurrence graph from Twitter. In this network, hashtags are the nodes, and a link between two hashtags indicates how often those two hashtags appear in tweets together. What link type would best capture this relation?

a. Undirected, unweighted
b. Undirected, weighted
c. Directed, unweighted
d. Directed, weighted

1.9 Consider a network created from characters in a story or play. The nodes are people, and a link exists between two nodes if those characters ever engage in dialogue. Which type of edge could represent this relation? Justify your answer.

a. Undirected, unweighted
b. Undirected, weighted
c. Directed, unweighted
d. Directed, weighted

1.10 Suppose we want to make a more complex version of dialog network that captures how much each character speaks and to whom. What type of link would best represent this relation?

a. Undirected, unweighted
b. Undirected, weighted
c. Directed, unweighted
d. Directed, weighted

1.11 Imagine that your social network has a subnetwork where you and 24 of your friends (25 people total) are all friends with each other. What is such a subnetwork called? And how many links are contained in the subnetwork?

1.12 Consider an undirected network with N nodes. What is the maximum number of links this network can have?

1.13 Consider a bipartite network of N nodes, N_1 nodes of type 1 and N_2 nodes of type 2 (so that $N_1 + N_2 = N$). What is that maximum number of links in this network?

1.14 Given a complete network A with N nodes, and a bipartite network B also with N nodes, which of the following holds true for any $N > 2$:

a. Network A has more links than network B
b. Network A has the same number of links as network B
c. Network A has fewer links than network B
d. None of these hold true for all such $N > 2$

1.15 Recall that in a complete network there exists a link between each pair of nodes. We know a complete undirected network of N nodes has $N(N-1)/2$ edges. Must any undirected network of N nodes and $N(N-1)/2$ links be complete? Explain why or why not.

1.16 Consider this adjacency matrix:

$$\begin{array}{c} \\ A \\ B \\ C \\ D \\ E \\ F \end{array} \begin{array}{c} \begin{array}{cccccc} A & B & C & D & E & F \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 2 & 1 & 3 & 1 & 1 & 0 \end{pmatrix} \end{array} \tag{1.11}$$

An entry in the ith row and jth column indicates the weight of the link from node i to node j. For instance, the entry in the 2nd row and 3rd column is 2, meaning the weight of the link from node **B** to node **C** is 2. What kind of network does this matrix represent?

a. Undirected, unweighted
b. Undirected, weighted
c. Directed, unweighted
d. Directed, weighted

1.17 Consider the network defined by the adjacency matrix in Eq. 1.11. How many nodes are in this network? How many links? Are there any self-loops?

1.18 Consider the network defined by the adjacency matrix in Eq. 1.11. Are there any nodes with outgoing links to every other node? If so, which nodes? Are there any nodes with in-links from every other node? If so, which nodes?

1.19 Consider the network defined by the adjacency matrix in Eq. 1.11. A *sink* is defined as a node with in-links but no out-links. Which nodes in the network, if any, have this property?

1.20 Consider the network defined by the adjacency matrix in Eq. 1.11. What is the in-strength of node **C**? What is its out-strength?

1.21 Convert the network defined by the adjacency matrix in Eq. 1.11 to an undirected, unweighted graph. (When converting a directed graph to an undirected one, nodes i and j are connected in the undirected graph if there is a directed link from i to j, or from j to i, or both.) You may want to print out the resulting matrix and/or draw a network diagram for reference. How many nodes are in this converted network? How many links?

1.22 Consider the unweighted, undirected version of the network defined by the adjacency matrix in Eq. 1.11, constructed as explained in Ex. 1.21. What is the minimum degree in this network? What is the maximum degree? What is the mean degree? What is the density?

1.23 Imagine two different undirected networks, each with the same number of nodes and links. Must both networks have the same maximum and minimum degree? Explain why or why not. Must they have the same mean degree? Explain why or why not.

1.24 We have seen that Facebook's network is incredibly sparse. Assume it has approximately 1 billion users, each with 1000 friends on average.

- Suppose Facebook releases its annual report and it shows that while the number of users in the network has stayed the same, the average number of friends per user has increased. Would this imply that the network density increased, decreased, or stayed the same?
- Suppose instead that both the number of users and the average number of friends per user doubled. Would this imply the network density increased, decreased, or stayed the same?

1.25 Netflix keeps data on customer preferences using a big bipartite network connecting users to titles they have watched and/or rated. Netflix's movie library contains approximately 100k titles if you count streaming and DVD-by-mail. In the fourth quarter of 2013, Netflix reported having about 33 million users. Assume the average user's degree in this network is 1000. Approximately how many links are in this network? Would you consider this network sparse or dense? Explain.

1.26 Netflix keeps data on customer preferences using a big bipartite network connecting users to titles. Suppose that from 2013 to 2014 Netflix's library has remained the same size, while the number of users has increased. Further suppose that the average user's degree in this network has remained constant. Has the density of this network increased, decreased, or stayed the same?