CHAPTER 6

MATHEMATICS OF NETWORKS

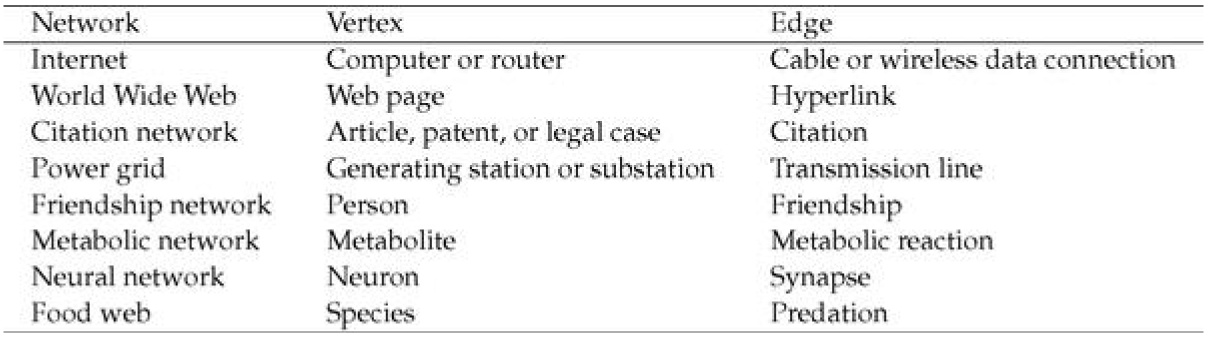
An introduction to the mathematical tools used in the study of networks, tools that will be important to many subsequent developments

IN THE next three chapters we introduce the fundamental quantitative foundations of the study of networks, concepts that are crucial for essentially all later developments in this book. In this chapter we introduce the basic theoretical tools used to describe and analyze networks, most of which come from graph theory, the branch of mathematics that deals with networks. Graph theory is a large field containing many results and we describe only a small fraction of those results here, focusing on the ones most relevant to the study of real-world networks. Readers interested in pursuing the study of graph theory further might like to look at the books by Harary [155] or West [324].

In the two chapters after this one we look first at measures and metrics for quantifying network structure (Chapter 7) and then at some of the remarkable patterns revealed in real-world networks when we apply the mathematics and metrics we have developed to their analysis (Chapter 8).

6.1 NETWORKS AND THEIR REPRESENTATION

To begin at the beginning, a *network*—also called a *graph* in the mathematical literature—is, as we have said, a collection of vertices joined by edges. Vertices and edges are also called *nodes* and *links* in computer science, *sites* and *bonds* in physics, and *actors*[43] and *ties* in sociology. Table 6.1 gives some examples of vertices and edges in particular networks.



Network	Vertex	Edge
Internet	Computer or router	Cable or wireless data connection
World Wide Web	Web page	Hyperlink
Citation network	Article, patent, or legal case	Citation
Power grid	Generating station or substation	Transmission line
Friendship network	Person	Friendship
Metabolic network	Metabolite	Metabolic reaction
Neural network	Neuron	Synapse
Food web	Species	Predation

Table 6.1: Vertices and edges in networks. Some examples of vertices and edges in particular networks.

Throughout this book we will normally denote the number of vertices in a network by *n* and the number of edges by *m*, which is a common notation in the mathematical literature.

Most of the networks we will study in this book have at most a single edge between any pair of vertices. In the rare cases where there can be more than one edge between the same pair of vertices we refer to those edges collectively as a *multiedge*. In most of the networks we will study there are also no edges that connect vertices to themselves, although such edges will occur in a few instances. Such edges are called *self-edges* or *self-loops*.

A network that has neither self-edges nor multiedges is called a *simple network* or *simple graph*. A network with multiedges is called a *multigraph*.[44] Figure 6.1 shows examples of (a) a simple graph and (b) a non-simple graph having both multiedges and self-edges.

6.2 THE ADJACENCY MATRIX

There are a number of different ways to represent a network mathematically. Consider an undirected network with n vertices and let us label the vertices with integer labels $1 \ldots n$, as we have, for instance, for the network in Fig. 6.1a. It does not matter which vertex gets which label, only that each label is unique, so that we can use the labels to refer to any vertex unambiguously.

If we denote an edge between vertices i and j by (i,j) then the complete network can be specified by giving the value of n and a list of all the edges. For example, the network in Fig. 6.1a has $n = 6$ vertices and edges (1,2), (1,5), (2,3), (2,4), (3,4), (3,5), and (3,6). Such a specification is called an *edge list*. Edge lists are sometimes used to store the structure of networks on computers, but for mathematical developments like those in this chapter they are rather cumbersome.

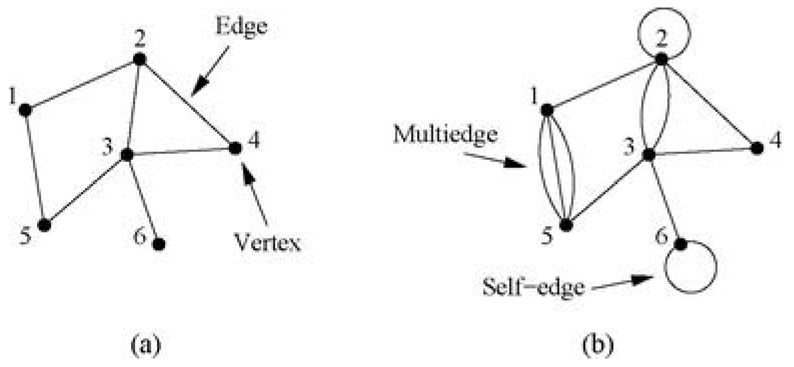


Figure 6.1: Two small networks. (a) A simple graph, i.e., one having no multiedges or self-edges. (b) A network with both multiedges and self-edges.

A better representation of a network for present purposes is the *adjacency matrix*. The adjacency matrix $\mathbf{A}$ of a simple graph is the matrix with elements A_{ij} such that

$$A_{ij} = \begin{cases} 1 & \text{if there is an edge between vertices } i \text{ and } j, \\ 0 & \text{otherwise.} \end{cases}$$

(6.1)

For example, the adjacency matrix of the network in Fig. 6.1a is

$$\mathbf{A} = \begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}. \tag{6.2}$$

Two points to notice about the adjacency matrix are that, first, for a network with no self-edges such as this one the diagonal matrix elements are all zero, and second that it is symmetric, since if there is an edge between *i* and *j* then there is an edge between *j* and *i*.

It is also possible to represent multiedges and self-edges using an adjacency matrix. A multiedge is represented by setting the corresponding matrix element A_{ij} equal to the multiplicity of the edge. For example, a double edge between vertices *i* and *j* is represented by $A_{ij} = A_{ji} = 2$.

Self-edges are a little more complicated. A single self-edge from vertex *i* to itself is represented by setting the corresponding diagonal element A_{ii} of the matrix equal to 2. Why 2 and not 1? Essentially it is because every self-edge from *i* to *i* has two ends, both of which are connected to vertex *i*. We will find that many of our mathematical results concerning the adjacency matrix work equally well for networks with and without self-edges, but only if we are careful to count both ends of every edge, including the self-edges, by making the diagonal matrix elements equal to 2 rather than 1.[45]

Another way to look at this is that non-self-edges appear twice in the adjacency matrix—an edge from *i* to *j* means that both A_{ij} and A_{ji} are 1. To count edges equally, self-edges should also appear twice, and since there is only one diagonal matrix element A_{ii} , we need to record both appearances there.

To give an example, the adjacency matrix for the multigraph in Fig. 6.1b is

$$\mathbf{A} = \begin{pmatrix} 0 & 1 & 0 & 0 & 3 & 0 \\ 1 & 2 & 2 & 1 & 0 & 0 \\ 0 & 2 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 3 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 2 \end{pmatrix}. \tag{6.3}$$

One can also have multiple self-edges (or “multi-self-edges” perhaps). Such edges are represented by setting the corresponding diagonal element of the adjacency matrix equal to twice the multiplicity of the edge.

6.3 WEIGHTED NETWORKS

Many of the networks we will study have edges that form simple on/off connections between vertices. Either they are there or they are not. In some situations, however, it is useful to represent edges as having a strength, weight, or value to them, usually a real number. Thus in the Internet edges might have weights representing the amount of data flowing along them or their bandwidth. In a food web predator-prey interactions might have weights measuring total energy flow between prey and predator. In a social network connections might have weights representing frequency of contact between actors. Such *weighted* or *valued networks* can be represented by giving the elements of the adjacency matrix values equal to the weights of the corresponding connections. Thus the adjacency matrix

$$\mathbf{A} = \begin{pmatrix} 0 & 2 & 1 \\ 2 & 0 & 0.5 \\ 1 & 0.5 & 0 \end{pmatrix}$$

(6.4)

represents a weighted network in which the connection between vertices 1 and 2 is twice as strong as that between 1 and 3, which in turn is twice as strong as that between 2 and 3.[46]

We have now seen two different types of network where the adjacency matrix can have off-diagonal elements with values other than 0 and 1, networks with weighted edges and networks with multiedges.[47] Indeed, if the weights in a weighted network are all integers it is possible to create a network with multiedges that has the exact same adjacency matrix, by simply choosing the multiplicities of the multiedges equal to the corresponding weights. This connection comes in handy sometimes. In some circumstances it is easier to reason about the behavior of a multigraph than a weighted network, or vice versa, and switching between the two can be a useful aid to analysis [242].

The weights in a weighted network are usually positive numbers, but there is no reason in theory why they should not be negative. For example, it is common in social network theory to construct networks of social relations between people in which positive edge weights denote friendship or other cordial relationships and negative ones represent animosity. We discuss such networks further in Section 7.11 when we consider the concept of structural balance.

Given that edges can have weights on them, it is not a huge leap to consider weights on vertices too, or to consider more exotic variables on either edges or vertices, such as vectors or discrete enumerative variables like colors. Many such variations have been considered in the networks literature and we will discuss some of them later in the book. There is one case of variables on edges, however, that is so central to the study of networks that we discuss it straight away.

6.4 DIRECTED NETWORKS

A *directed network* or *directed graph*, also called a *digraph* for short, is a network in which each edge has a direction, pointing *from* one vertex *to* another. Such edges are themselves called *directed edges*, and can be represented by lines with arrows on them—see Fig. 6.2.

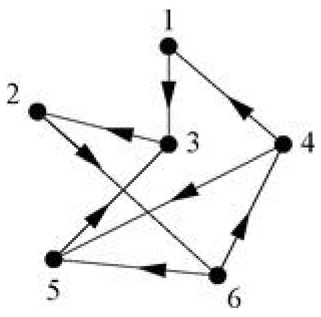


Figure 6.2: A directed network. A small directed network with arrows indicating the directions of the edges.

We encountered a number of examples of directed networks in previous chapters, including the World Wide Web, in which hyperlinks run in one direction from one web page to another, food webs, in which energy flows from prey to predators, and citation networks, in which citations point from one paper to another.

The adjacency matrix of a directed network has matrix elements

$$A_{ij} = \begin{cases} 1 & \text{if there is an edge } \textit{from } j \textit{ to } i, \\ 0 & \text{otherwise.} \end{cases}$$

(6.5)

Notice the direction of the edge here—it runs *from* the second index *to* the first. This is slightly counter-intuitive, but it turns out to be convenient mathematically and it is the convention we adopt in this book.

As an example, the adjacency matrix of the small network in Fig. 6.2 is

$$\mathbf{A} = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix} .$$

(6.6)

Note that this matrix is not symmetric. In general the adjacency matrix of a directed network is asymmetric.

We can, if we wish, think of undirected networks as directed networks in which each undirected edge has been replaced with two directed ones running in opposite directions between the same pair of vertices. The adjacency matrix for such a network is then symmetric and exactly the same as for the original undirected network.

Like their undirected counterparts, directed networks can have multiedges and self-edges, which are represented in the adjacency matrix by elements with values greater than 1 and by non-zero diagonal elements, respectively. An important point however is that self-edges in a directed network are represented by setting the corresponding diagonal element of the adjacency matrix to 1, not 2 as in the undirected case.[48] With this choice the same formulas and results, in terms of the adjacency matrix, apply for networks with and without self-edges.

6.4.1 COCITATION AND BIBLIOGRAPHIC COUPLING

It is sometimes convenient to turn a directed network into an undirected one for the purposes of analysis—there are many useful analytic techniques for undirected networks that do not have directed counterparts (or at least not yet).

One simple way to make a directed network undirected is just to ignore the edge directions entirely, an approach that can work in some cases, but inevitably throws out a lot of potentially useful information about the network's structure. A more sophisticated approach is to use "cocitation" or "bibliographic coupling," two different but related ideas that derive their names from their widespread use in the analysis of citation networks.

We briefly discussed cocitation in the context of citation networks in Section 4.2.

The *cocitation* of two vertices *i* and *j* in a directed network is the number of vertices that have outgoing edges pointing to both *i* and *j*. In the language of citation networks, for instance, the cocitation of two papers is the number of other papers that cite both. Given the definition above of the adjacency matrix of a directed network (Eq. (6.5)), we can see that $A_{ik}A_{jk} = 1$ if *i* and *j* are both cited by *k* and zero otherwise. Summing over all *k*, the cocitation C_{ij} of *i* and *j* is

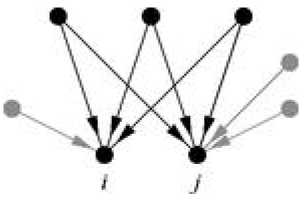
$$C_{ij} = \sum_{k=1}^{n} A_{ik}A_{jk} = \sum_{k=1}^{n} A_{ik}A^{T}_{kj},$$

(6.7)

where A^{T}_{kj} is an element of the transpose of **A**. We can define the *cocitation matrix* **C** to be the $n \times n$ matrix with elements C_{ij}, which is thus given by

$$\mathbf{C} = \mathbf{A}\mathbf{A}^{T}.$$

(6.8)



Vertices *i* and *j* are cited by three common papers, so their cocitation is 3.

Note that **C** is a symmetric matrix, since $\mathbf{C}^T = (\mathbf{A}\mathbf{A}^T)^T = \mathbf{A}\mathbf{A}^T = \mathbf{C}$.

Now we can define a *cocitation network* in which there is an edge between *i* and *j* if $C_{ij} > 0$, for $i \neq j$, i.e., an edge between any two vertices that are cocited in the original directed network. (We enforce the constraint that $i \neq j$ because the cocitation network is conventionally defined to have no self-edges, even though the diagonal elements of the cocitation matrix are in general nonzero—see below.) Better still, we can make the cocitation network a weighted network with positive integer weights on the edges equal to the corresponding elements C_{ij}. Then vertex pairs cited by more common neighbors have a stronger connection than those cited by fewer. Since the cocitation matrix is symmetric, the cocitation network is undirected, making it easier to deal with in many respects than the original directed network from which it was constructed.

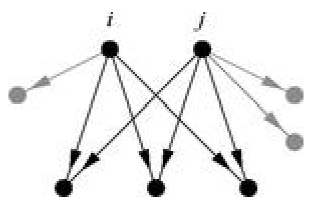
The cocitation network turns out to make a lot of sense in many cases. In citation networks of academic papers, for instance, strong cocitation between papers is often a good indicator of papers that deal with related topics—if two papers are often cited together in the same bibliography they probably have something in common. And the more often they are cited together, the more likely it is that they are related.

The cocitation matrix thus plays a role similar to an adjacency matrix for the cocitation network. There is however one aspect in which the cocitation matrix differs from an adjacency matrix: its diagonal elements. The diagonal elements of the cocitation matrix are given by

$$C_{ii} = \sum_{k=1}^{n} A_{ik}^2 = \sum_{k=1}^{n} A_{ik},$$

(6.9)

where we have assumed that the directed network is a simple graph, with no multiedges, so that all elements A_{ik} of the adjacency matrix are zero or one. Thus C_{ii} is equal to the total number of edges pointing to *i*—the total number of papers citing *i* in the citation network language. In constructing the cocitation network we ignore these diagonal elements, meaning that the network's adjacency matrix is equal to the cocitation matrix but with all the diagonal elements set to zero.



Vertices *i* and *j* cite three of the same papers and so have a bibliographic coupling of 3.

Bibliographic coupling is similar to cocitation. The *bibliographic coupling* of two vertices in a directed network is the number of other vertices to which both point. In a citation network, for instance, the bibliographic coupling of two papers *i* and *j* is the number of other papers that are cited by both *i* and *j*. Noting that $A_{ki}A_{kj} = 1$ if *i* and *j* both cite *k* and zero otherwise, the bibliographic coupling of *i* and *j* is

$$B_{ij} = \sum_{k=1}^{n} A_{ki} A_{kj} = \sum_{k=1}^{n} A_{ik}^{T} A_{kj},$$

(6.10)

and we define the *bibliographic coupling matrix* **B** to be the $n \times n$ matrix with elements B_{ij} so that

$$\mathbf{B} = \mathbf{A}^{T}\mathbf{A}.$$

(6.11)

The bibliographic coupling matrix is again a symmetric matrix and the off-diagonal elements can be used to define a weighted undirected network, the *bibliographic coupling network*, in which there is an edge with weight B_{ij} between any vertex pair i, j for which $B_{ij} > 0$. The diagonal elements of **B** are

$$B_{ii} = \sum_{k=1}^{n} A_{ki}^{2} = \sum_{k=1}^{n} A_{ki}.$$

(6.12)

Thus B_{ii} is equal to the number of other vertices that vertex i points to—the number of papers i cites, in the citation language.

Bibliographic coupling, like cocitation, can be a useful measure of connection between vertices. In a citation network, for example, if two papers cite many of the same other papers it is often a good indication that they deal with similar subject matter, and the number of common papers cited can be an indicator of how strongly they overlap.

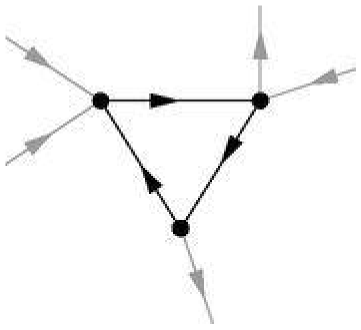
Although cocitation and bibliographic coupling are mathematically similar measures they can in practice give noticeably different results. In particular, they are affected strongly by the number of ingoing and outgoing edges that vertices have. For two vertices to have strong cocitation—to be pointed to by many of the same other vertices—they must both have a lot of incoming edges in the first place. In a citation network, for instance, two papers can only have strong cocitation if they are both well cited and hence strong cocitation is limited to influential papers, review articles, books, and similar highly cited items. Conversely, two papers can only have strong bibliographic coupling if they both cite many others, i.e., if they have large bibliographies. In practice, the sizes of bibliographies vary less than the number of citations papers receive, and hence bibliographic coupling is a more uniform indicator of similarity between papers than cocitation. The Science Citation Index, for example, makes use of bibliographic coupling in its "Related Records" feature, which allows users to find papers similar to a given paper. Cocitation would be less appropriate in this situation, since it tends not to work well for papers with few citations.

Bibliographic coupling also has the advantage that it can be computed as soon as a paper is published and the contents of the paper's bibliography are known. Cocitation, on the other hand, cannot be computed until a paper has been cited by other papers, which usually doesn't happen until at least a few months after publication, and sometimes years. Furthermore, the cocitation of two papers can change over time as the papers receive new citations, whereas bibliographic coupling is fixed from the moment the papers are published. (This could be an advantage or a disadvantage—there are situations in which changes in cocitation could reveal interesting information about the papers that cannot be gleaned from an unchanging measure like bibliographic coupling.)

In addition to their use as measures of vertex similarity, the cocitation and bibliographic coupling matrices are also used in search algorithms for directed networks, and in particular in the so-called HITS algorithm, which we describe in Section 7.5.

6.4.2 ACYCLIC DIRECTED NETWORKS

A *cycle* in a directed network is a closed loop of edges with the arrows on each of the edges pointing the same way around the loop. Networks like the World Wide Web have many such cycles in them. Some directed networks however have no cycles and these are called *acyclic* networks.[49] Ones with cycles are called *cyclic*. A self-edge—an edge connecting a vertex to itself—counts as a cycle, and so an acyclic network also has no self-edges.



A cycle in a directed network.

The classic example of an acyclic directed network is a citation network of papers, as discussed in Section 4.2. When writing a paper you can only cite another paper if it has already been written, which means that all the directed edges in a citation network point backward in time. Graphically we can depict such a network as in Fig. 6.3, with the vertices time-ordered—running from bottom to top of the picture in this case—so that all the edges representing the citations point downward in the picture.[50] There can be no closed cycles in such a network because any cycle would have to go down the picture and then come back up again to get back to where it started and there are no upward edges with which to achieve this.

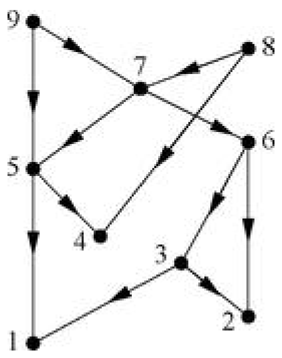


Figure 6.3: An acyclic directed network. In this network the vertices are laid out in such a way that all edges point downward. Networks that can be laid out in this way are called acyclic, since they possess no closed cycles of edges. An example of an acyclic network is a citation network of citations between papers, in which the vertical axis would represent date of publication, running up

the figure, and all citations would necessarily point from later papers to earlier ones.

It is less obvious but still true that if a network is acyclic it can be drawn in the manner of Fig. 6.3 with all edges pointing downward. The proof that this can be done turns out to be useful, because it also provides us with a method for determining whether a given network is acyclic.

Suppose we have an acyclic directed network of *n* vertices. There must be at least one vertex somewhere on the network that has ingoing edges only and no outgoing ones. To see this consider starting from any vertex in the network and making a path across the network by following edges, each in the correct direction denoted by its arrow. Either such a path will eventually encounter a vertex with no outgoing edges, in which case we are done, or each vertex it encounters has one or more outgoing edges, in which case we choose one such edge and continue our path. If the path never reaches a vertex with no outgoing edges, then it must eventually arrive back at a vertex that has been visited previously—at most we can visit all *n* vertices in the network once before the path either terminates or we are forced to revisit a vertex. However if we revisit a vertex then we have gone around a cycle in the network, which cannot be since the network is acyclic. Thus we must always in the end find a vertex with no outgoing edges and hence at least one such vertex always exists.

In practice, it is not necessary to actually construct the paths through the network to find a vertex with no outgoing edges—since we know that such a vertex exists, we can simply look through each vertex in turn until we find one.

We now take this vertex with no outgoing edges and draw it at the bottom of our picture. We remove this vertex from the network, along with any edges attached to it, and repeat the process, finding another vertex with no outgoing edges in the remaining network. We draw this second vertex above the first one in the figure, remove it from the network and repeat again. And so forth.

When we have drawn all vertices, we then add the directed edges between them to the picture. Since each edge, by definition, has incoming edges only from vertices drawn after it—and therefore above it—all edges in the final picture must be pointing downward. Note that the particular order in which we draw the vertices, and hence the picture we produce, is not necessarily unique. If at any stage in the process of drawing the vertices there is more than one vertex with no outgoing edges then we have a choice about which one we pick and hence a choice between overall vertex orders.

This process is a useful one for visualizing acyclic networks. Most computer algorithms for drawing such networks work by arranging the vertices in order along one axis in just this way, and then moving them around along the other axis to make the network structure as clear and visually pleasing as possible (which usually means minimizing the number of times that edges cross).

The process is useful for another reason too: it will break down if the network is cyclic, and therefore it gives us a way to test whether a given network is acyclic. If a network contains a cycle, then none of the vertices in that cycle will ever be removed during our process: none of them will be without outgoing edges until one of the others in the cycle is removed, and hence none of them can ever be removed. Thus, if the network contains a cycle there must come a point in our process where there are still vertices left in the network but all of them have outgoing edges. So a simple algorithm for determining whether a network is acyclic is:

1. Find a vertex with no outgoing edges.
2. If no such vertex exists, the network is *cyclic*. Otherwise, if such a vertex does exist, remove it and all its ingoing edges from the network.
3. If all vertices have been removed, the network is *acyclic*. Otherwise go back to step 1.

The adjacency matrix of an acyclic directed network has interesting properties. Suppose we construct an ordering of the vertices of an acyclic network as described above, so that all edges point in one direction, and suppose we then label the vertices in that order. Then there can be an edge from vertex j to vertex i only if $j > i$. Put another way, the adjacency matrix $\mathbf{A}$ (whose element A_{ij} records the presence of an edge *from j to i*) has all its non-zero elements above the diagonal—it is upper triangular. For instance, the adjacency matrix of the network shown in Fig.

6.3 is

$$\mathbf{A} = \begin{pmatrix} 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

(6.13)

Note also that the diagonal elements of the adjacency matrix are necessarily zero, since an acyclic network has no self-edges. Triangular matrices with zeros on the diagonal are called *strictly triangular*.

If the vertices of an acyclic network are not numbered in order as described above, then the adjacency matrix will not be triangular. (Imagine swapping rows and columns of the matrix above, for instance.) However, we can say that for every acyclic directed network there exists at least one labeling of the vertices such that the adjacency matrix will be strictly upper triangular.

The adjacency matrix also has the property that all of its eigenvalues are zero if and only if the network is acyclic. To demonstrate this, we must demonstrate the correspondence in both directions, i.e., that the adjacency matrix of an acyclic network has all eigenvalues zero and also that a network is acyclic if its adjacency matrix has all eigenvalues zero.

The former is the easier to prove. If a network is acyclic then we can order and label the vertices as described above and hence write the adjacency matrix in strictly upper triangular form. The diagonal elements of a triangular matrix, however, are its eigenvalues, and since these are all zero it follows immediately that all eigenvalues are zero for an acyclic network.

To show the converse, that the network is acyclic if the eigenvalues are all zero, it suffices to demonstrate that any cyclic network must have at least one non-zero eigenvalue. To demonstrate this we make use of a result derived in Section 6.10. There we show that the total number L_r of cycles of length r in a network is

$$L_r = \sum_{i=1}^{n} \kappa_i^r,$$

(6.14)

where κ_i is the ith eigenvalue of the adjacency matrix. Suppose a network is cyclic. Let r be the length of one of the cycles it contains. Then by definition $L_r > 0$ for this network. However, this can only be the case if at least one of the terms in the sum on the right-hand side of Eq. (6.14) is greater than zero, and hence the adjacency matrix has at least one non-zero eigenvalue. If all eigenvalues are zero, therefore, the network cannot be cyclic.

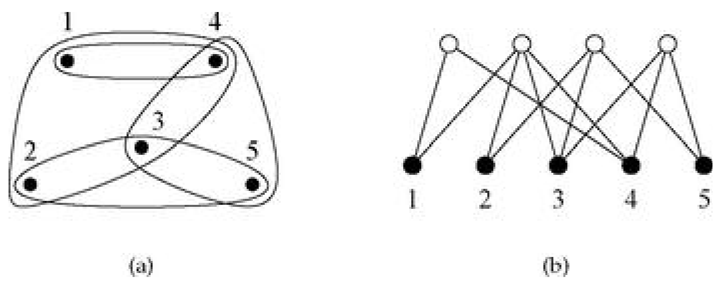
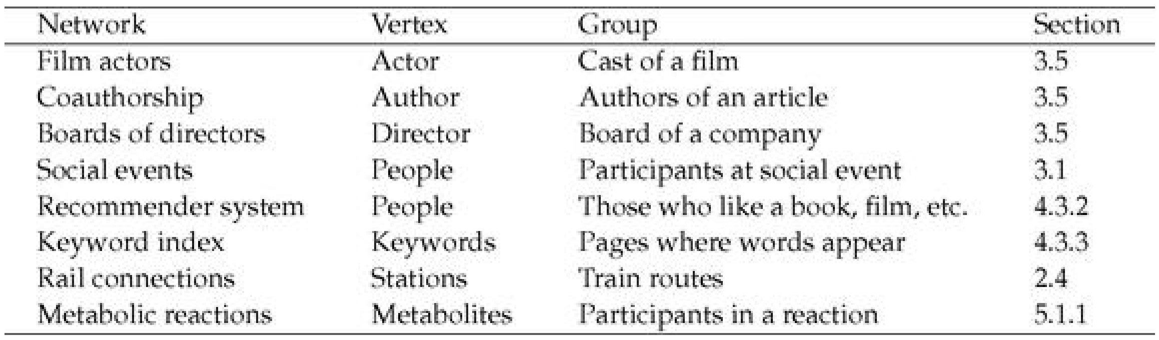


Figure 6.4: A hypergraph and corresponding bipartite graph. These two networks show the same information—the membership of five vertices in four different groups. (a) The hypergraph representation in which the groups are represented as hyperedges, denoted by the loops circling sets of vertices. (b) The bipartite representation in which we introduce four new vertices (open circles) representing the four groups, with edges connecting each vertex to the groups to which it belongs.

Matrices with all eigenvalues zero are called *nilpotent matrices*. Thus one could also say that a network is acyclic if and only if it has a nilpotent adjacency matrix.

6.5 HYPERGRAPHS

In some kinds of network the links join more than two vertices at a time. For example, we might want to create a social network representing families in a larger community of people. Families can have more than two people in them and the best way to represent family ties in such families is to use a generalized kind of edge that joins more than two vertices.[51] Such an edge is called a *hyperedge* and a network with hyperedges is called a *hypergraph*. Figure 6.4a shows a small example of a hypergraph in which the hyperedges are denoted by loops.



Network	Vertex	Group	Section
Film actors	Actor	Cast of a film	3.5
Coauthorship	Author	Authors of an article	3.5
Boards of directors	Director	Board of a company	3.5
Social events	People	Participants at social event	3.1
Recommender system	People	Those who like a book, film, etc.	4.3.2
Keyword index	Keywords	Pages where words appear	4.3.3
Rail connections	Stations	Train routes	2.4
Metabolic reactions	Metabolites	Participants in a reaction	5.1.1

Table 6.2: Hypergraphs and bipartite graphs. Examples of networks that can be represented as hypergraphs or equivalently as bipartite graphs. The last column gives the section of this book in which each network is discussed.

Many of the networks that we will encounter in this book can be presented as hypergraphs. In particular, any network in which the vertices are connected together by common membership of groups of some kind can be represented in this way. In sociology such networks are called "affiliation networks" and we saw several examples of them in Section 3.5. Directors sitting on the boards of companies, scientists coauthoring papers, and film actors appearing together in films are all examples of such networks (see Table 6.2).

We will however talk very little about hypergraphs in this book, because there is another way of representing the same information that is more convenient for our purposes—the bipartite network.

6.6 BIPARTITE NETWORKS

The membership of vertices in groups represented by hyperedges in a hypergraph can equally and often more conveniently be represented as a *bipartite network*, also called a *two-mode network* in the sociology literature. In such a network there are two kinds of vertices, one representing the original vertices and the other representing the groups to which they belong. We discussed bipartite networks previously in the context of affiliation networks in Section 3.5 and of recommender networks in Section 4.3.2. For example, we can represent the network of film actors discussed in Section 3.5 as a bipartite network in which the two types of vertex are the actors themselves and the films in which they appear. The edges in a bipartite network run only between vertices of unlike types: in the film network they would run only between actors and films, and each actor would be connected by an edge to each film in which he or she appeared. A small example of a bipartite network is shown in Fig. 6.4b. This example network in fact portrays exactly the same set of group memberships as the hypergraph of Fig. 6.4a; the two are entirely equivalent.

Bipartite networks occur occasionally in contexts other than membership of groups. For example, if we were to construct a network of who is or has been married to whom within a population, that network would be bipartite, the two kinds of vertex corresponding to men and women and the edges between them marriages.[52]

The equivalent of an adjacency matrix for a bipartite network is a rectangular matrix called an *incidence matrix*. If n is the number of people or other participants in the network and g is the number of groups, then the incidence matrix **B** is a $g \times n$ matrix having elements B_{ij} such that

$$B_{ij} = \begin{cases} 1 & \text{if vertex } j \text{ belongs to group } i, \\ 0 & \text{otherwise.} \end{cases}$$

(6.15)

For instance, the 4×5 incidence matrix of the network shown in Fig. 6.4b is

$$\mathbf{B} = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 \end{pmatrix}.$$

(6.16)

Although a bipartite network may give the most complete representation of a particular network it is often convenient to work with direct connections between vertices of just one type. We can use the bipartite network to infer such connections, creating a *one-mode projection* from the two-mode bipartite form. As an example, consider again the case of the films and actors. We can perform a projection onto the actors alone by constructing the *n*-vertex network in which the vertices represent actors and two actors are connected by an edge if they have appeared together in a film. The corresponding one-mode projection onto the films would be the g-vertex network where the vertices represent films and two films are connected if they share a common actor. Figure 6.5 shows the two one-mode projections of a small bipartite network.

When we form a one-mode projection each group in the bipartite network results in a cluster of vertices in the one-mode projection that are all connected to each other—a “clique” in network jargon (see Section 7.8.1). For instance, if a group contains four members in the bipartite network, then each of those four is connected to each of the others in the one-mode projection by virtue of common membership in that group. (Such a clique of four vertices is visible in the center of the lower projection in Fig. 6.5.) Thus the projection is, generically, the union of a number of cliques, one for each group in the original bipartite network. The same goes for the other projection onto the groups.

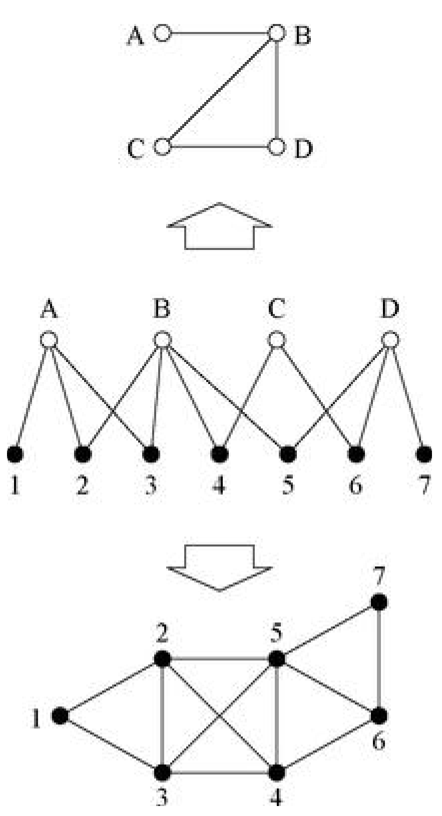


Figure 6.5: The two one-mode projections of a bipartite network. The central portion of this figure shows a bipartite network with four vertices of one type (open circles labeled A to D) and seven of another (filled circles, 1 to 7). At the top and bottom we show the one-mode projections of the network onto the two sets of vertices.

The one-mode projection, as we have described it, is often useful and is widely employed, but its construction discards a lot of the information present in the structure of the original bipartite network and hence it is, in a sense, a less powerful representation of our data. For example, the projection loses any information about how many groups two vertices share in common. In the case of the actors and films, for instance, there are some pairs of actors who have appeared in many films together—Fred Astaire and Ginger Rogers, say, or William Shatner and Leonard Nimoy—and it’s reasonable to suppose this indicates a stronger connection than between actors who appeared together only once.

We can capture information of this kind in our projection by making the projection weighted, giving each edge between two vertices in the projected network a weight equal to the number of common groups the vertices share. This weighted network still does not capture all the information in the bipartite original—it doesn't record the number of groups or the exact membership of each group for instance—but it is an improvement on the unweighted version and is quite widely used.

Mathematically the projection can be written in terms of the incidence matrix **B** as follows. The product $B_{ki}B_{kj}$ will be 1 if and only if i and j both belong to the same group k in the bipartite network. Thus, the total number P_{ij} of groups to which both i and j belong is

$$P_{ij} = \sum_{k=1}^{g} B_{ki}B_{kj} = \sum_{k=1}^{g} B_{ik}^T B_{kj},$$

(6.17)

where B_{ik}^T is an element of the transpose $\mathbf{B}^T$ of **B**. The $n \times n$ matrix $\mathbf{P} = \mathbf{B}^T\mathbf{B}$ is similar to an adjacency matrix for the weighted one-mode projection onto the n vertices. Its off-diagonal elements are equal to the weights in that network, the number of common groups shared by each vertex pair. **P** is not quite an adjacency matrix, however, since its diagonal elements are non-zero, even though the network itself, by definition, has no self-edges. (In this respect **P** is somewhat similar to the cocitation matrix of Section 6.4.1.) The diagonal elements have values

$$P_{ii} = \sum_{k=1}^{g} B_{ki}^2 = \sum_{k=1}^{g} B_{ki},$$

(6.18)

where we have made use of the fact that B_{ki} only takes the values 0 or 1. Thus P_{ii} is equal to the number of groups to which vertex i belongs.

Thus to derive the adjacency matrix of the weighted one-mode projection, we would calculate the matrix $\mathbf{P} = \mathbf{B}^T\mathbf{B}$ and set the diagonal elements equal to zero. And to derive the adjacency matrix of the unweighted projection, we would take the adjacency matrix of the weighted version and replace every non-zero matrix element with a 1.

The other one-mode projection, onto the groups, can be represented by a $g \times g$ matrix $\mathbf{P}' = \mathbf{B}\mathbf{B}^T$, whose off-diagonal element P'_{ij} gives the number of common members of groups i and j, and whose diagonal element P'_{ii} gives the number of members of group i.

One occasionally also comes across bipartite networks that are directed. For example, the metabolic networks discussed in Section 5.1.1 can be represented as directed bipartite networks—see Fig. 5.1a. A variety of more complex types of projection are possible in this case, although their use is rare and we won't spend time on them here. Weighted bipartite networks are also possible in principle, although no examples will come up in this book.

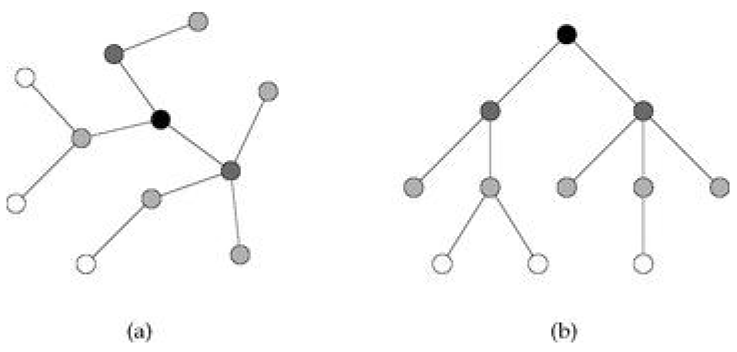


Figure 6.6: Two sketches of the same tree. The two panels here show two different depictions of a tree, a network with no closed loops. In (a) the vertices are positioned on the page in any convenient position. In (b) the tree is a laid out in a “rooted” fashion, with a root node at the top and branches leading down to “leaves” at the bottom.

6.7 TREES

A *tree* is a connected, undirected network that contains no closed loops—see Fig. 6.6a.[53] By "connected" we mean that every vertex in the network is reachable from every other via some path through the network. A network can also consist of two or more parts, disconnected from one another,[54] and if an individual part has no loops it is also called a tree. If all the parts of the network are trees, the complete network is called a *forest*.

Trees are often drawn in a *rooted* manner, as shown in Fig. 6.6b, with a *root node* at the top and a branching structure going down. The vertices at the bottom that are connected to only one other vertex are called *leaves*.[55] Topologically, a tree has no particular root—the same tree can be drawn with any node, including a leaf, as the root node, but in some applications there are other reasons for designating a root. A dendrogram is one example (see below).

Not very many of the real-world networks that we will encounter in this book are trees, although a few are. A river network is an example of a naturally occurring tree (see Fig. 2.6, for instance). Trees do nonetheless play several important roles in the study of networks. In Chapter 12 for instance we will study the network model known as the "random graph." In this model local groups of vertices—the so-called small components in the network—form trees, and we can exploit this property to derive a variety of mathematical results about random graphs. In Section 11.11.1 we introduce the "dendrogram," a useful tool that portrays a hierarchical decomposition of a network as a tree. Trees also occur commonly in computer science, where they are used as a basic building block for data structures such as AVL trees and heaps (see Sections 9.5 and 9.7 and Refs. [8, 81]) and in other theoretical contexts like minimum spanning trees [81], Cayley trees or Bethe lattices [269], and hierarchical models of networks (see Section 19.3.2 and Refs. [70, 179, 322]).

Perhaps the most important property of trees for our purposes is that, since they have no closed loops, there is exactly one path between any pair of vertices. (In a forest there is at most one path, but there may be none.) This is clear since if there were two paths between a pair of vertices A and B then we could go from A to B along one path and back along the other, making a loop, which is forbidden.

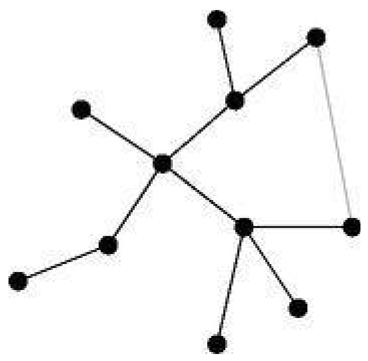
This property of trees makes certain kinds of calculation particularly simple, and trees are sometimes used as a basic model of a network for this reason. For instance, the calculation of a network's diameter (Section 6.10.1), the betweenness centrality of a vertex (Section 7.7), and certain other properties based on shortest paths are all relatively easy with a tree.

Another useful property of trees is that a tree of n vertices always has exactly n— 1 edges. To see this, consider building up a tree by adding vertices one by one. Starting with a single vertex and no edges, we add a second vertex and one edge to connect it to the first. Similarly when we add a third vertex we need at least one edge to connect it one of the others, and so forth. For every vertex we must add at least one edge to keep the network connected. This means that the number of edges must always be at least one less than the number of vertices. In mathematical terms, $n - 1$ is a lower bound on the number of edges.

But it is also an upper bound, because if we add more than one edge when we add a new vertex then we create a loop: the first edge connects the added vertex to the rest of the network and the second then connects together two vertices that are already part of the network. But adding an edge between two vertices that are already connected via the network necessarily creates a loop. Hence we are not allowed to add more than one edge per vertex if the network is to remain free of loops.

Thus the number of edges in a tree cannot be either more or less than $n - 1$, and hence is exactly $n - 1$.

The reverse is also true, that any connected network with n vertices and $n - 1$ edges is a tree. If such a network were not a tree then there must be a loop in the network somewhere, implying that we could remove an edge without disconnecting any part of the network. Doing this repeatedly until no loops are left, we would end up with a tree, but one with less than $n - 1$ edges, which cannot be. Hence we must have had a tree to begin with. As a corollary, this implies that the connected network on n vertices with the minimum number of edges is always a tree, since no connected network has less than $n - 1$ edges and all networks with $n - 1$ edges are trees.



Adding an extra edge (gray) between any two vertices of a tree creates a loop.

6.8 PLANAR NETWORKS

A *planar network* is a network that can be drawn on a plane without having any edges cross.[56] Figure 6.7a shows a small planar network. Note that it is in most cases possible to find a way to draw a planar network so that some edges do cross (Fig. 6.7b). The definition of planarity only specifies that at least one arrangement of the vertices exists that results in no crossing.

Most of the networks we will encounter in this book are not planar, either because there is no relevant two-dimensional geometry to which the network is confined (e.g., citation networks, metabolic networks, collaboration networks), or else there is but there is nothing to stop edges from crossing on it (e.g., the Internet, airline route maps, email networks). However, there are a few important examples of networks that are planar. First of all, all trees are planar. For some trees, such as river networks, this is obvious. Rivers never cross one another; they only flow together. In other cases, such as the trees used in computer data structures, there is no obvious two-dimensional surface onto which the network falls but it is planar nonetheless.

Among non-tree-like networks, some are planar for physical reasons. A good example is a road network. Because roads are confined to the Earth's surface they form a roughly planar network. It does happen sometimes that roads meet without intersecting, one passing over the other on a bridge, so that in fact, if one wishes to be precise, the road network is not planar. However, such instances are rare (in the sense that there are far more places where roads intersect than there are bridges where they don't) and the network is planar to a good approximation.

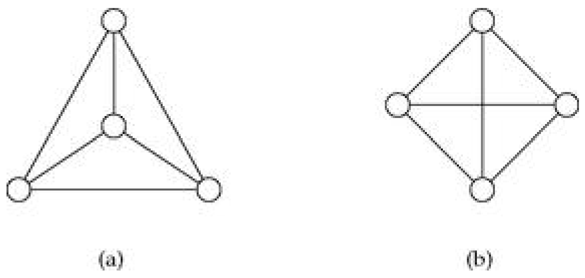


Figure 6.7: Two drawings of a planar graph. (a) A small planar graph with four vertices and six edges. It is self-evident that the graph is planar, since in this depiction it has no edges that cross. (b) The same graph redrawn with two of its edges crossing. Even though the edges cross, the graph is still planar—a graph is planar if it *can* be drawn without crossing edges.

Another example is the network of shared borders between countries, states, or provinces—see Fig. 6.8. We can take a map depicting any set of contiguous regions, represent each by a vertex, and draw an edge between any two that share a border. It is easy to see that the resulting network can always be drawn without crossing edges provided the regions in question are formed of contiguous landmasses.

Networks of this type, representing regions on a map, play an important role in the *four-color theorem*, a theorem that states that it is possible to color any set of regions on a two-dimensional map, real or imagined, with at most four colors such that no two adjacent regions have the same color, no matter how many regions there are or of what size or shape.[57] By constructing the network corresponding to the map in question, this problem can be converted into a problem of coloring the vertices of a planar graph in such a way that no two vertices connected by an edge have the same color. The number of colors required to color a graph in this way is called the *chromatic number* of the graph and many mathematical results are known about chromatic numbers. The proof of the four-color theorem—the proof that the chromatic number of a planar graph is always four or less—is one of the triumphs of traditional graph theory and was first given by Appel and Haken [20-22] in 1976 after more than a hundred years of valiant effort within the mathematics community.[58]

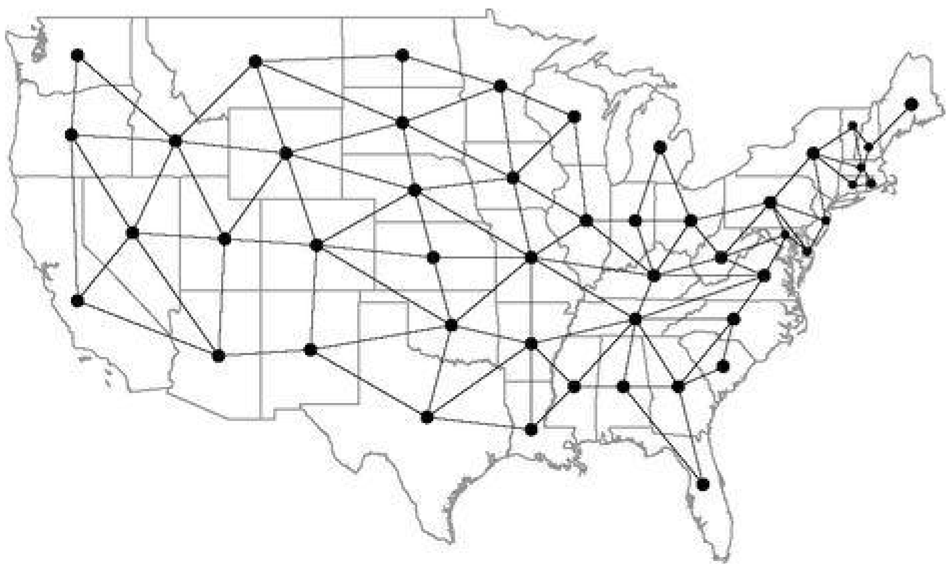
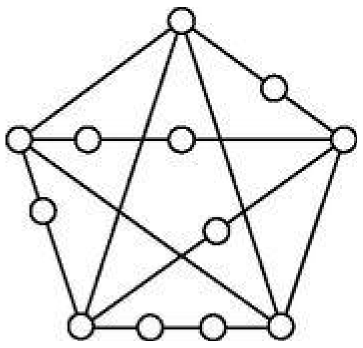


Figure 6.8: Graph of the adjacencies of the lower 48 United States. In this network each of the lower 48 states in the US is represented as a vertex and there is an edge between any two vertices if the corresponding states share a border. The resulting graph is planar, and indeed any set of states, countries, or other regions on a two-dimensional map can be turned into a planar graph in this way.

An important question that arises in graph theory is how to determine, given a particular network, whether that network is planar or not. For a small network it is a straightforward matter to draw a picture and play around with the positions of the vertices to see if one can find an arrangement in which no edges cross, but for a large network this is impractical and a more general method of determining planarity is needed. Luckily a straightforward one exists. We will only describe the method here, not prove why it works, since the proof is long and technical and not particularly relevant to the study of real-world networks. For those interested in seeing a proof, one is given by West [324].

Figure 6.9 shows two small networks, conventionally denoted K_5 and UG, that are definitely not planar.[59] Neither of these networks can be drawn without edges crossing. It immediately follows that any network that contains a subset of vertices, or subgraph, in the form of K_5 or UG, is also not planar.



An expansion of K_5.

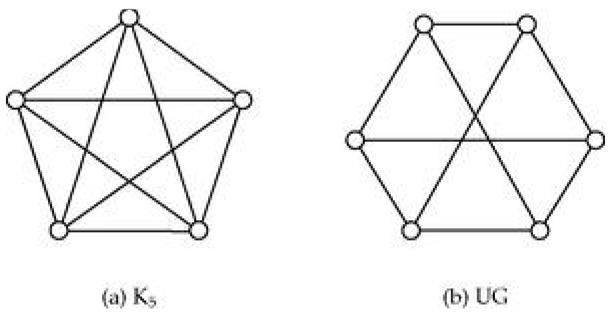


Figure 6.9: The fundamental non-planar graphs $K_5$and UG employed in Kuratowski's theorem. These two small graphs are non-planar and Kuratowski's theorem states that any non-planar graph contains at least one subgraph that is an expansion of K_5 or UG.

An *expansion* is a network derived by adding extra vertices in the middle of edges in another network. No such added vertices, however numerous, will ever make a non-planar network planar, so it is also the case that any expansion of K_5 or UG is non-planar, and hence that any network containing an expansion of K_5 or UG, is also non-planar.

Kuratowski's theorem (sometimes also called the *Kuratowski reduction theorem*) states that the converse is also true:

Every non-planar network contains at least one subgraph that is an expansion of K_5 or UG.

"Expansion" should be taken here to include the null expansions, i.e., the graphs K_5 and UG themselves.

This theorem, first proved by Pontryagin in 1927 but named after Kuratowski who gave an independent proof a few years later,[60] provides us with a way of determining whether a graph is planar. If it contains a subgraph that is an expansion of K_5 or UG it is not, otherwise it is.

Kuratowski's theorem is not, however, particularly useful for the analysis of real-world networks, because such networks are rarely precisely planar. (And if they are, then, as in the case of the shared border network of countries or states, it is usually clear for other reasons that they are planar and hence Kuratowski's theorem is unnecessary.) More often, like the road network, they are very nearly planar, but have a few edge crossings somewhere in the network. For such a network, Kuratowski's theorem would tell us, correctly, that the network was not planar, but we would be missing the point.

What we would really like is some measure of the degree of planarity of a network, a measure that could tell us, for example, that the road network of a country is 99% planar, even though there are a few bridges or tunnels here and there. One possible such measure is the minimum number of edge crossings with which the network can be drawn. This however would be a difficult measure to determine since, at least in the simplest approach, its evaluation would require us to try every possible way of drawing the network. Perhaps another approach would be to look at the number of subgraphs in a network that are expansions of K_5 or UG. So far, however, no widely accepted metric for degree of planarity has emerged. If such a measure were to gain currency it might well find occasional use in the study of real-world networks.

6.9 DEGREE

The *degree* of a vertex in a graph is the number of edges connected to it. We will denote the degree of vertex *i* by k_i. For an undirected graph of *n* vertices the degree can be written in terms of the adjacency matrix as

$$k_i = \sum_{j=1}^{n} A_{ij}.$$

(6.19)

Every edge in an undirected graph has two ends and if there are *m* edges in total then there are 2*m* ends of edges. But the number of ends of edges is also equal to the sum of the degrees of all the vertices, so

$$2m = \sum_{i=1}^{n} k_i,$$

(6.20)

or

$$m = \tfrac{1}{2}\sum_{i=1}^{n} k_i = \tfrac{1}{2}\sum_{ij} A_{ij},$$

(6.21)

a result that we will use many times throughout this book.
The mean degree *c* of a vertex in an undirected graph is

$$c = \frac{1}{n}\sum_{i=1}^{n} k_i,$$

(6.22)

and combining this with Eq. (6.20) we get

$$c = \frac{2m}{n}.$$

(6.23)

This relation too will come up repeatedly throughout the book.

The maximum possible number of edges in a simple graph (i.e., one with no multiedges or self-edges) is $\binom{n}{2} = \frac{1}{2}n(n-1)$. The *connectance* or *density* ρ of a graph is the fraction of these edges that are actually present:

$$\rho = \frac{m}{\binom{n}{2}} = \frac{2m}{n(n-1)} = \frac{c}{n-1},$$

(6.24)

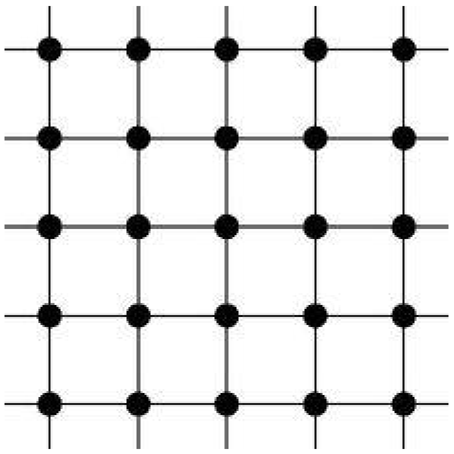
where we have made use of Eq. (6.23).[62] The density lies strictly in the range $0 \le \rho \le 1$. Most of the networks we are interested in are sufficiently large that Eq. (6.24) can be safely approximated as $\rho = c/n$.

A network for which the density ρ tends to a constant as $n \to \infty$ is said to be *dense*. In such a network the fraction of non-zero elements in the adjacency matrix remains constant as the network becomes large. A network in which $\rho \to 0$ as $n \to \infty$ is said to be *sparse*, and the fraction of non-zero elements in the adjacency matrix also tends to zero. In particular, a network is sparse if c tends to a constant as n becomes large. These definitions of dense and sparse networks can, however, be applied only if one can actually take the limit $n \to \infty$, which is fine for theoretical model networks but doesn't work in most practical situations. You cannot for example take the limit as an empirical metabolic network or food web becomes large—you are stuck with the network nature gives you and it can't easily be changed.

In some cases real-world networks do change their sizes and by making measurements for different sizes we can make a guess as to whether they are best regarded as sparse or dense. The Internet and the World Wide Web are two examples of networks whose growth over time allows us to say with some conviction that they are best regarded as sparse. In other cases there may be independent reasons for regarding a network to be sparse or dense. In a friendship network, for instance, it seems unlikely that the number of a person's friends will double solely because the population of the world doubles. How many friends a person has is more a function of how much time they have to devote to the maintenance of friendships than it is a function of how many people are being born. Friendship networks therefore are usually regarded as sparse.

In fact, almost of all of the networks we consider in this book are considered to be sparse networks. This will be important when we look at the expected running time of network algorithms in Chapters 9 to 11 and when we construct mathematical models of networks in Chapters 12 to 15. One possible exception to the pattern is food webs. Studies comparing ecosystems of different sizes seem to show that the density of food webs is roughly constant, regardless of their size, indicating that food webs may be dense networks [102, 210].

Occasionally we will come across networks in which all vertices have the same degree. In graph theory, such networks are called *regular graphs*. A regular graph in which all vertices have degree *k* is sometimes called *k-regular*. An example of a regular graph is a periodic lattice such as a square or triangular lattice. On the square lattice, for instance, every vertex has degree four.



An infinite square lattice is an example of a 4-regular graph.

Vertex degrees are more complicated in directed networks. In a directed network each vertex has two degrees. The *in-degree* is the number of ingoing edges connected to a vertex and the *out-degree* is the number of outgoing edges. Bearing in mind that the adjacency matrix of a directed network has element $A_{ij} = 1$ if there is an edge from j to i, in- and out-degrees can be written

$$k_i^{\text{in}} = \sum_{j=1}^{n} A_{ij}, \qquad k_j^{\text{out}} = \sum_{i=1}^{n} A_{ij}.$$

(6.25)

The number of edges m in a directed network is equal to the total number of ingoing ends of edges at all vertices, or equivalently to the total number of outgoing ends of edges, so

$$m = \sum_{i=1}^{n} k_i^{\text{in}} = \sum_{j=1}^{n} k_j^{\text{out}} = \sum_{ij} A_{ij}.$$

(6.26)

Thus the mean in-degree c_{in} and the mean out-degree c_{out} of every directed network are equal:

$$c_{\text{in}} = \frac{1}{n}\sum_{i=1}^{n} k_i^{\text{in}} = \frac{1}{n}\sum_{j=1}^{n} k_j^{\text{out}} = c_{\text{out}}.$$

(6.27)

For simplicity we will just denote both by *c*, and combining Eqs. (6.26) and (6.27) we get

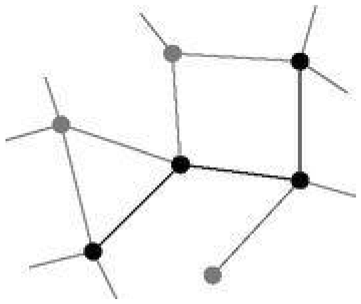
$$c = \frac{m}{n}.$$

(6.28)

Note that this differs by a factor of two from the equivalent result for an undirected network, Eq. (6.23).

6.10 PATHS

A *path* in a network is any sequence of vertices such that every consecutive pair of vertices in the sequence is connected by an edge in the network. In layman′s terms a path is a route across the network that runs from vertex to vertex along the edges of the network. Paths can be defined for both directed and undirected networks. In a directed network, each edge traversed by a path must be traversed in the correct direction for that edge. In an undirected network edges can be traversed in either direction.



A path of length three in a network.

In general a path can intersect itself, visiting again a vertex it has visited before, or even running along an edge or set of edges more than once. Paths that do not intersect themselves are called *self-avoiding paths* and are important in some areas of network theory. Geodesic paths and Hamiltonian paths are two special cases of self-avoiding paths that we will study in this book.

The *length* of a path in a network is the number of edges traversed along the path (not the number of vertices). Edges can be traversed more than once, and if they are they are counted separately each time they are traversed. Again in layman's terms, the length of a path is the number of "hops" the path makes from vertex to adjacent vertex.

It is straightforward to calculate the number of paths of a given length *r* on a network. For either a directed or an undirected simple graph the element A_{ij} is 1 if there is an edge from vertex *j* to vertex *i*, and 0 otherwise. (We will consider only simple graphs for now, although the developments generalize easily to non-simple graphs.) Then the product $A_{ik}A_{kj}$ is 1 if there is a path of length 2 from *j* to *i* via *k*, and 0 otherwise. And the total number $N_{ij}^{(2)}$ of paths of length two from *j* to *i*, via any other vertex, is

$$N_{ij}^{(2)} = \sum_{k=1}^{n} A_{ik}A_{kj} = \left[\mathbf{A}^2\right]_{ij},$$

(6.29)

where $[\ldots]_{ij}$ denotes the ijth element of a matrix.

Similarly the product $A_{ik}A_{kl}A_{lj}$ is 1 if there is a path of length three from j to i via l and k, in that order, and 0 otherwise, and hence the total number of paths of length three is

$$N_{ij}^{(3)} = \sum_{k,l=1}^{n} A_{ik}A_{kl}A_{lj} = [\mathbf{A}^3]_{ij}.$$

(6.30)

Generalizing to paths of arbitrary length r, we see that

$$N_{ij}^{(r)} = [\mathbf{A}^r]_{ij}.$$

(6.31)

A special case of this result is that the number of paths of length r that start and end at the same vertex i is $[\mathbf{A}^r]_{ii}$. These paths are just loops in the network, what we called “cycles” in our discussion of acyclic graphs in Section 6.1. The total number L_r of loops of length r anywhere in a network is the sum of this quantity over all possible starting points i:

$$L_r = \sum_{i=1}^{n}[\mathbf{A}^r]_{ii} = \mathrm{Tr}\,\mathbf{A}^r.$$

(6.32)

Note that this expression counts separately loops consisting of the same vertices in the same order but with different starting points.64 Thus the loop 1 → 2 → 3 → 1 is considered different from the loop 2 → 3 → 1 → 2. The expression also counts separately loops that consist of the same vertices but traversed in opposite directions, so that 1 → 2 → 3 → 1 and 1 → 3 → 2 → 1 are distinct.

Equation (6.32) can also be expressed in terms of the eigenvalues of the adjacency matrix. Let us consider the case of an undirected graph first. In this case, the adjacency matrix is symmetric, which means that it has n real non-negative eigenvalues, the eigenvectors have real elements, and the matrix can always be written in the form $\mathbf{A} = \mathbf{UKU}^T$, where $\mathbf{U}$ is the orthogonal matrix of eigenvectors and $\mathbf{K}$ is the diagonal matrix of eigenvalues. Then $\mathbf{A}^r = (\mathbf{UKU}^T)^r = \mathbf{UK}^r\mathbf{U}^T$ and the number of loops is

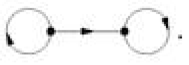
$$\begin{aligned} L_r &= \mathrm{Tr}(\mathbf{U}K^r\mathbf{U}^T) = \mathrm{Tr}(\mathbf{U}^T\mathbf{U}K^r) = \mathrm{Tr}\,K^r \\ &= \sum_i \kappa_i^r, \end{aligned}$$

(6.33)

where κ_i is the ith eigenvalue of the adjacency matrix and we have made use of the fact that the trace of a matrix product is invariant under cyclic permutations of the product.

For directed networks the situation is more complicated. In some cases the same line of proof works and we can again demonstrate that Eq. (6.33) is true, but in other cases the proof breaks down. Recall that directed graphs have, in general, asymmetric adjacency matrices, and some asymmetric matrices cannot be diagonalized.[65] An example is the matrix

$$\begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}, \quad \text{which describes the graph}$$



This matrix has only a single (right) eigenvector (1, 0), and thus one cannot form an orthogonal matrix of eigenvectors with which to diagonalize it. Nonetheless Eq. (6.33) is still true even in such cases, but a different method of proof is needed, as follows.

Every real matrix, whether diagonalizable or not, can be written in the form $\mathbf{A} = \mathbf{QTQ}^T$, where $\mathbf{Q}$ is an orthogonal matrix and $\mathbf{T}$ is an upper triangular matrix. This form is called the *Schur decomposition* of **A** [217].

Since **T** is triangular, its diagonal elements are its eigenvalues. Furthermore those eigenvalues are the same as the eigenvalues of **A**. To see this, let **x** be a right eigenvector of **A** with eigenvalue κ. Then $\mathbf{QTQ}^T\mathbf{x} = \mathbf{Ax} = \kappa\mathbf{x}$, and multiplying throughout by $\mathbf{Q}^T$, bearing in mind that **Q** is orthogonal, gives

$$\mathbf{TQ}^T\mathbf{x} = \kappa\mathbf{Q}^T\mathbf{x},$$

(6.34)

and hence $\mathbf{Q}^T\mathbf{x}$ is an eigenvector of **T** with the same eigenvalue κ as the adjacency matrix.[66] Then

$$\begin{aligned} L_r &= \operatorname{Tr}\mathbf{A}^r = \operatorname{Tr}(\mathbf{QT}^r\mathbf{Q}^T) = \operatorname{Tr}(\mathbf{Q}^T\mathbf{QT}^r) = \operatorname{Tr}\mathbf{T}^r \\ &= \sum_i \kappa_i^r, \end{aligned}$$

(6.35)

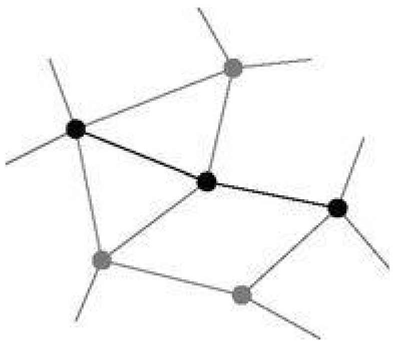
the final equality following because the diagonal elements of any power of a triangular matrix **T** are **T**′s diagonal elements raised to the same power.

This demonstration works for any graph, whatever the properties of its adjacency matrix, and hence Eq. (6.35) is always true. We used this result in Eq. (6.14) to show that the graph described by a nilpotent adjacency matrix (i.e., a matrix whose eigenvalues are all zero) must be acyclic. (All such matrices are non-diagonalizable, so one must use Eq. (6.35) in that case.)

Since the adjacency matrix of a directed graph is, in general, asymmetric it may have complex eigenvalues. But the number of loops L_r above is nonetheless always real, as it must be. The eigenvalues of the adjacency matrix are the roots of the characteristic polynomial $\det(\kappa\mathbf{I} - \mathbf{A})$, which has real coefficients, and all roots of such a polynomial are either themselves real or come in complex-conjugate pairs. Thus, while there may be complex terms in the sum in Eq. (6.33), each such term is complemented by another that is its complex conjugate and the sum itself is always real.

6.10.1 GEODESIC PATHS

A *geodesic path*, also called simply a *shortest path*, is a path between two vertices such that no shorter path exists:



A geodesic path of length two between two vertices.

The length of a geodesic path, often called the *geodesic distance* or *shortest distance* , is thus the shortest network distance between the vertices in question. In mathematical terms, the geodesic distance between vertices i and j is the smallest value of r such that $[\mathbf{A}^r]_{ij} > 0$. In practice however there are much better ways of calculating geodesic distances than by employing this formula. We will study some of them in Section 10.3.

It is possible for there to be no geodesic path between two vertices if the vertices are not connected together by any route though the network (i.e., if they are in different "components"—see Section 6.11). In this case one sometimes says that the geodesic distance between the vertices is infinite, although this is mostly just convention—it doesn't really mean very much beyond the fact that the vertices are not connected.

Geodesic paths are necessarily self-avoiding. If a path intersects itself then it contains a loop and can be shortened by removing that loop while still connecting the same start and end points, and hence self-intersecting paths are never geodesic paths.

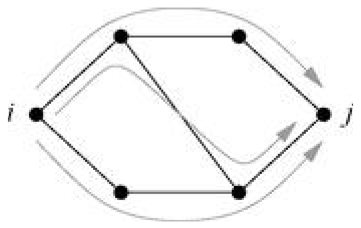


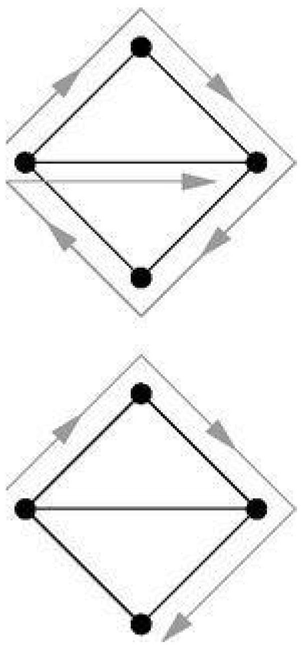
Figure 6.10: Vertices i and j have three geodesic paths between them of length three.

Geodesic paths are not necessarily unique, however. It is perfectly possible to have two or more paths of equal length between a given pair of vertices. The paths may even overlap along some portion of their length—see Fig. 6.10.

The *diameter* of a graph is the length of the longest geodesic path between any pair of vertices in the network for which a path actually exists. (If the diameter were merely the length of the longest geodesic path then it would be formally infinite in a network with more than one component if we adopted the convention above that vertices connected by no path have infinite geodesic distance. One can also talk about the diameters of the individual components separately, this being a perfectly well-defined concept whatever convention we adopt for unconnected vertices.)

6.10.2 EULERIAN AND HAMILTONIAN PATHS

An *Eulerian path* is a path that traverses each edge in a network exactly once. A *Hamiltonian path* is a path that visits each vertex exactly once. A network can have one or many Eulerian or Hamiltonian paths, or none. A Hamiltonian path is by definition self-avoiding, but an Eulerian path need not be. Indeed if there are any vertices of degree greater than two in a network an Eulerian path will have to visit those vertices more than once in order to traverse all their edges.



Examples of Eulerian and Hamiltonian paths in a small network.

Eulerian paths form the basis of one of the oldest proofs in graph theory, which dates from 1736. Around that time the great mathematician Leonard Euler became interested the mathematical riddle now known as the *Königsberg Bridge Problem*. The city of Königsberg (now Kaliningrad) was built on the banks of the river Pregel, and on two islands that lie midstream. Seven bridges connected the land masses, as shown in Fig. 6.11a. The riddle asked, “Does there exist any walking route that crosses all seven bridges exactly once each?” Legend has it that the people of Königsberg spent many fruitless hours trying to find such a route, before Euler proved the impossibility of its existence.[67] The proof, which perhaps seems rather trivial now, but which apparently wasn’t obvious in 1736, involved constructing a network (technically a multigraph) with four vertices representing the four land masses and seven edges joining them in the pattern of the Königsberg bridges (Fig. 6.11b). Then the bridge problem becomes a problem of finding an Eulerian path on this network (and indeed the Eulerian path is named in honor of Euler for his work on this problem). Euler observed that, since any Eulerian path must both enter and leave every vertex it passes through except the first and last, there can at most be two vertices in the network with odd degree if such a path is to exist. Since all four vertices in the Königsberg network have odd degree, the bridge problem necessarily has no solution.

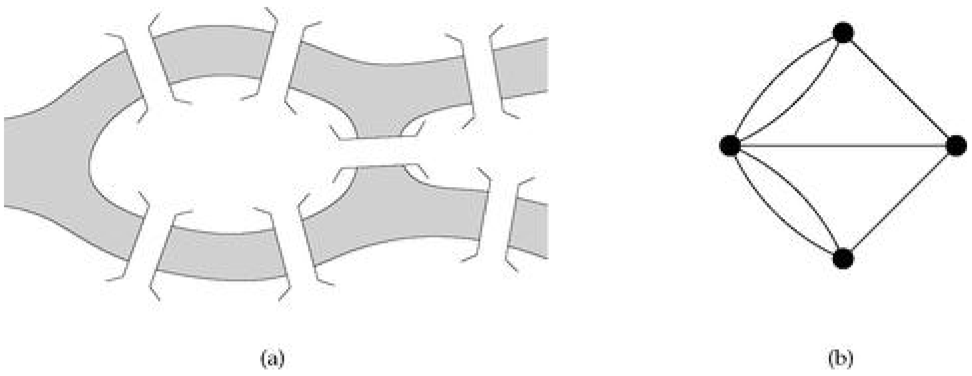


Figure 6.11: The Königsberg bridges. (a) In the eighteenth century the Prussian city of Königsberg, built on four landmasses around the river Pregel, was connected by seven bridges as shown. (b) The topology of the landmasses and bridges can be represented as a multigraph with four vertices and seven edges.

More precisely a network can have an Eulerian path only if there are exactly two or zero vertices of odd degree—zero in the case where the path starts and ends at the same vertex. This is not a sufficient condition for an Eulerian path, however. One can easily find networks that satisfy it and yet have no Eulerian path. The general problem of finding either an Eulerian or Hamiltonian path on a network, or proving that none exists, is a hard one and significant work is still being done on particular cases.

Eulerian and Hamiltonian paths have a number of practical applications in computer science, in job sequencing, "garbage collection," and parallel programming [81]. A Hamiltonian path problem was also, famously, the first problem solved using a DNA-based computer [7].

6.11 COMPONENTS

It is possible for there to be no path at all between a given pair of vertices in a network. The network shown in Fig. 6.12, for example, is divided into two subgroups of vertices, with no connections between the two, so that there is no path from any vertex in the left subgroup to any vertex in the right. For instance, there is no path from the vertex labeled A to the vertex labeled B. A network of this kind is said to be *disconnected*. Conversely, if there is a path from every vertex in a network to every other the network is *connected*.

The subgroups in a network like that of Fig. 6.12 are called *components*. Technically a component is a subset of the vertices of a network such that there exists at least one path from each member of that subset to each other member, and such that no other vertex in the network can be added to the subset while preserving this property. (Subsets like this, to which no other vertex can be added while preserving a given property, are called *maximal subsets*.) The network in Fig. 6.12 has two components of three and four vertices respectively. A connected network necessarily has only one component. A singleton vertex that is connected to no others is considered to be a component of size one, and every vertex belongs to exactly one component.

The adjacency matrix of a network with more than one component can be written in block diagonal form, meaning that the non-zero elements of the matrix are confined to square blocks along the diagonal of the matrix, with all other elements being zero:

$$\mathbf{A} = \begin{pmatrix} \boxed{} & 0 & \cdots \\ 0 & \boxed{} & \cdots \\ \vdots & \vdots & \ddots \end{pmatrix}.$$

(6.36)

Note, however, that the vertex labels must be chosen correctly to produce this form. The visual appearance of blocks in the adjacency matrix depends on the vertices of each component being represented by adjacent rows and columns and choices of labels that don’t achieve this will produce non-block-diagonal matrices, even though the choice of labels has no effect on the structure of the network itself. Thus, depending on the labeling, it may not always be immediately obvious from the adjacency matrix that a network has separate components. There do, however, exist computer algorithms, such as the “breadth-first search” algorithm described in Section 10.3, that can take a network with arbitrary vertex labels and quickly determine its components.

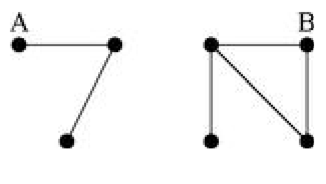


Figure 6.12: A network with two components. This undirected network contains two components of three and four vertices respectively. There is no path between pairs of vertices like A and B that lie in different components.

6.11.1 COMPONENTS IN DIRECTED NETWORKS

When we look at directed networks the definition of components becomes more complicated. The situation is worth looking at in some detail, because it assumes some practical importance in networks like the World Wide Web. Consider the directed network shown in Fig. 6.13. If we ignore the directed nature of the edges, considering them instead to be undirected, then the network has two components of four vertices each. In the jargon of graph theory these are called *weakly connected components*. Two vertices are in the same weakly connected component if they are connected by one or more paths through the network, where paths are allowed to go either way along any edge.

In many practical situations, however, this is not what we care about. For example, the edges in the World Wide Web are directed hyperlinks that allow Web users to surf from one page to another, but only in one direction. This means it is possible to reach one web page from another by surfing only if there is a directed path between them, i.e., a path in which we follow edges only in the forward direction. It would be useful to define components for directed networks based on such directed paths, but this raises some problems. It is certainly possible for there to be a directed path from vertex A to vertex B but no path back from B to A. Should we then consider A and B to be connected? Are they in the same component or not?

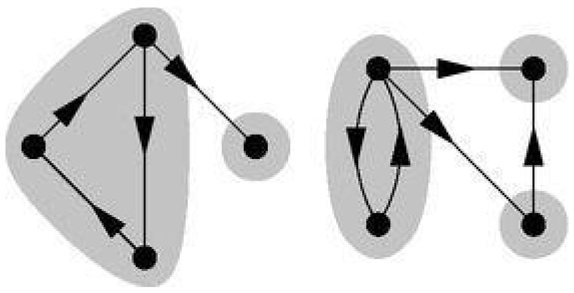


Figure 6.13: Components in a directed network. This network has two weakly connected components of four vertices each, and five strongly connected components (shaded).

Clearly there are various answers one could give to these questions. One possibility is that we define A and B to be connected if and only if there exists both a directed path from A to B and a directed path from B to A. A and B are then said to be *strongly connected*. We can define components for a directed network using this definition of connection and these are called *strongly connected components*. Technically, a strongly connected component is a maximal subset of vertices such that there is a directed path in both directions between every pair in the subset. The strongly connected components of the network in Fig. 6.13 are highlighted by the shaded regions. Note that there can be strongly connected components consisting of just a single vertex and, as with the undirected case, each vertex belongs to exactly one strongly connected component. Note also that every strongly connected component with more than one vertex must contain at least one cycle. Indeed every *vertex* in such a component must belong to at least one cycle, since there is by definition a directed path from that vertex to every other in the component and a directed path back again, and the two paths together constitute a cycle. (A corollary of this observation is that acyclic directed graphs have no strongly connected components with more than one vertex, since if they did they wouldn’t be acyclic.)

Strongly and weakly connected components are not the only useful definitions of components in a directed network. On the Web it could be useful to know what pages you can reach by surfing from a given starting point, but you might not care so much whether it's possible to surf back the other way. Considerations of this kind lead us to define the *out-component*, which is the set of vertices that are reachable via directed paths starting at a specified vertex A, and including A itself.

An out-component has the property that edges connecting it to other vertices (ones not in the out-component) only point inward towards the members of component, and never outward (since if they pointed outward then the vertices they connected to would by definition be members of the out-component).

Note that the members of an out-component depend on the choice of the starting vertex. Choose a different starting vertex and the set of reachable vertices may change. Thus an out-component is a property of the network structure and the starting vertex, and not (as with strongly and weakly connected components) of the network structure alone. This means, among other things, that a vertex can belong to more than one different out-component. In Fig. 6.14, for instance, we show the out-components of two different starting vertices, A and B. Vertices X and Y belong to both.

A few other points are worth noticing. First, it is self-evident that all the members of the strongly connected component to which a vertex A belongs are also members of A's out-component. Furthermore, all vertices that are reachable from A are necessarily also reachable from all the other vertices in the strongly connected component. Thus it follows that the out-components of all members of a strongly connected component are identical. It would be reasonable to say that out-components really "belong" not to individual vertices, but to strongly connected components.

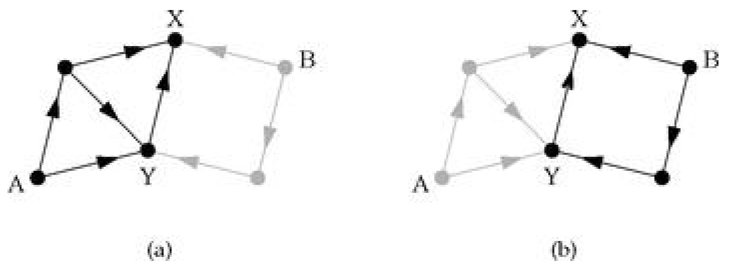
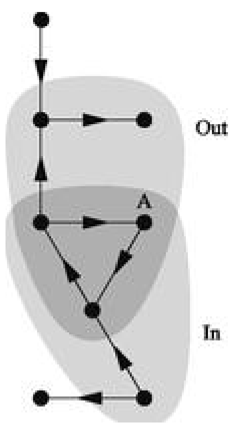


Figure 6.14: Out-components in a directed network. (a) The out-component of vertex A, which is the subset of vertices reachable by directed paths from A. (b) The out-component of vertex B. Vertices X and Y belong to both out-components.

Very similar arguments apply to vertices *from which* a particular vertex can be reached. The *in-component* of a specified vertex A is the set of all vertices from which there is a directed path to A, including A itself. In-components depend on the choice of the specified vertex, and a vertex can belong to more than one in-component, but all vertices in the same strongly connected component have the same in-component. Furthermore, the strongly connected component to which a vertex belongs is a subset of its in-component, and indeed a vertex that is in both the in- and out-components of A is necessarily in the same strongly connected component as A (since paths exist in both directions) and hence A's strongly connected component is equal to the intersection of its in- and out-components.



The in- and out-components of a vertex A in a small directed network.

6.12 INDEPENDENT PATHS, CONNECTIVITY, AND CUT SETS

A pair of vertices in a network will typically be connected by many paths of many different lengths. These paths will usually not be independent however. That is, they will share some vertices or edges, as in Fig. 6.10 for instance (page 140). If we restrict ourselves to independent paths, then the number of paths between a given pair of vertices is much smaller. The number of independent paths between vertices gives a simple measure of how strongly the vertices are connected to one another, and has been the topic of much study in the graph theory literature.

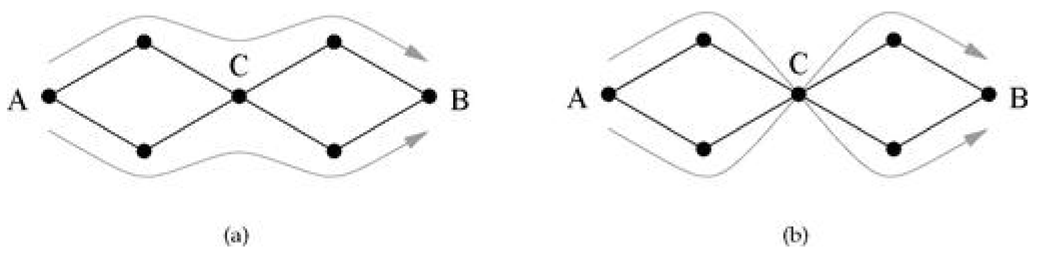


Figure 6.15: Edge independent paths. (a) There are two edge-independent paths from A to B in this figure, as denoted by the arrows, but there is only one vertex-independent path, because all paths must pass through the center vertex C. (b) The edge-independent paths are not unique; there are two different ways of choosing two independent paths from A to B in this case.

There are two species of independent path: edge-independent and vertex-independent. Two paths connecting a given pair of vertices are *edge-independent* if they share no edges. Two paths are *vertex-independent* (or *node-independent*) if they share no vertices other than the starting and ending vertices. If two paths are vertex-independent then they are also edge-independent, but the reverse is not true: it is possible to be edge-independent but not vertex-independent. For instance, the network shown in Fig. 6.15a has two edge-independent paths from A to B, as denoted by the arrows, but only one vertex-independent path—the two edge-independent paths are not vertex-independent because they share the intermediate vertex C.

Independent paths are also sometimes called *disjoint paths*, primarily in the mathematical literature. One also sees the terms *edge-disjoint* and *vertexdisjoint* , describing edge and vertex independence.

The edge- or vertex-independent paths between two vertices are not necessarily unique. There may be more than one way of choosing a set of independent paths. For instance Fig. 6.15b shows the same network as Fig. 6.15a, but with the two paths chosen a different way, so that they cross over as they pass through the central vertex C.

It takes only a moment's reflection to convince oneself that there can be only a finite number of independent paths between any two vertices in a finite network. Each path must contain at least one edge and no two paths can share an edge, so the number of independent paths cannot exceed the number of edges in the network.

The number of independent paths between a pair of vertices is called the *connectivity* of the vertices.[68] If we wish to be explicit about whether we are considering edge- or vertex-independence, we refer to *edge* or *vertex connectivity* . The vertices A and B in Fig. 6.15 have edge connectivity 2 but vertex connectivity 1 (since there is only one vertex-independent path between them).

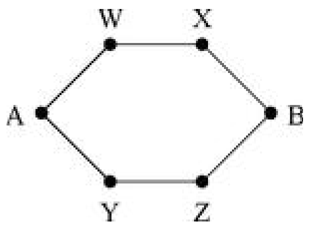
The connectivity of a pair of vertices can be thought of as a measure of how strongly connected those vertices are. A pair that have only a single independent path between them are perhaps more tenuously connected than a pair that have many paths. This idea is sometimes exploited in the analysis of networks, for instance in algorithmic methods for discovering clusters or communities of strongly linked vertices within networks [122].

Connectivity can also be visualized in terms of “bottlenecks” between vertices. Vertices A and B in Fig. 6.15, for instance, are connected by only one vertex-independent path because vertex C forms a bottleneck through which only one path can go. This idea of bottlenecks is formalized by the notion of cut sets as follows.

Consider an undirected network. (In fact the developments here apply equally to directed ones, but for simplicity let us stick with the undirected case for now.) A *cut set*, or more properly a *vertex cut set*, is a set of vertices whose removal will disconnect a specified pair of vertices. For example, the central vertex C in Fig. 6.15 forms a cut set of size 1 for the vertices A and B. If it is removed, there will be no path from A to B. There are also other cut sets for A and B in this network, although all the others are larger than size 1.

An *edge cut set* is the equivalent construct for edges—it is a set of edges whose removal will disconnect a specified pair of vertices.

A *minimum cut set* is the smallest cut set that will disconnect a specified pair of vertices. In Fig. 6.15 the single vertex C is a minimum vertex cut set for vertices A and B. A minimum cut set need not be unique. For instance, there is a variety of minimum vertex cut sets of size two between the vertices A and B in this network:



{W,Y}, {W,Z}, {X,Y}, and {X,Z} are all minimum cut sets for this network. (There are also many different minimum edge cut sets.) Of course all the minimum cut sets must have the same size.

An important early theorem in graph theory addresses the size of cut sets. *Menger’s theorem* states:

> If there is no cut set of size less than *n* between a given pair of vertices, then there are at least *n* independent paths between the same vertices.

The theorem applies both to edges and to vertices and was first proved by Karl Menger [216] for the vertex case, although many other proofs have been given since. A simple one can be found in Ref. [324].

To understand why Menger’s theorem is important, consider the following argument. If the minimum vertex cut set between two vertices has size *n*, Menger’s theorem tells us that there must be at least *n* vertex-independent paths between those vertices. That is, the number of vertex-independent paths is greater than or equal to the size of the minimum cut set. Conversely, if we know there to be exactly *n* vertex-independent paths between two vertices, then, at the very least, we have to remove one vertex from each path in order to disconnect the two vertices, so the size of the minimum cut set must be at least *n*. We thus conclude that the number of vertex-independent paths must be both greater than or equal to *and* less than or equal to the size of the minimum cut set, which can only be true if the two are in fact equal. Thus Menger’s theorem implies that:

> The size of the minimum vertex cut set that disconnects a given pair of vertices in a network

is equal to the vertex connectivity of the same vertices.

Given that Menger's theorem also applies for edges, a similar argument can be used to show that the same result also applies for edge cut sets and edge connectivity.

The edge version of Menger's theorem has a further corollary that will be of some importance to us when we come to study computer algorithms for analyzing networks. It concerns the idea of *maximum flow*. Imagine a network of water pipes in the shape of some network of interest. The edges of the network correspond to the pipes and the vertices to junctions between pipes. Suppose that there is a maximum rate *r*, in terms of volume per unit time, at which water can flow through any pipe. What then is the maximum rate at which water than can flow through the network from one vertex, A, to another, B? The answer is that this maximum flow is equal to the number of edge-independent paths times the pipe capacity *r*.

We can construct a proof of this result starting from Menger's theorem. First, we observe that if there are *n* independent paths between A and B, each of which can carry water at rate *r*, then the network as a whole can carry a flow of at least *nr* between A and B, i.e., *nr* is a lower bound on the maximum flow.

At the same time, by Menger's theorem, we know that there exists a cut set of *n* edges between A and B. If we push the maximum flow (whatever it is) through the network from A to B and then remove one of the edges in this cut set, the maximum flow will be reduced by at most *r*, since that is the maximum flow an edge can carry. Thus if we remove all *n* edges in the cut set one by one, we remove at most *nr* of flow. But, since the cut set disconnects the vertices A and B, this removal must stop all of the flow. Hence the total capacity is at most *nr*, i.e., *nr* is an upper bound on the maximum flow.

Thus *nr* is both an upper and a lower bound on the maximum flow, and hence the maximum flow must in fact be exactly equal to *nr*.

This in outline is a proof of the *max-flow/min-cut theorem*, in the special case in which each pipe can carry the same fixed flow. The theorem says that the maximum flow between two vertices is always equal to the size of the minimum cut set times the capacity of a single pipe. The full max-flow/min-cut theorem applies also to weighted networks in which individual pipes can have different capacities. We look at this more general case in the following section.

In combination, Menger's theorem for edges and the max-flow/min-cut theorem show that for a pair of vertices in an undirected network three quantities are all numerically equal to each other: the edge connectivity of the pair (i.e., the number of edge-independent paths connecting them), the size of the minimum edge cut set (i.e., the number of edges that must be removed to disconnect them), and the maximum flow between the vertices if each edge in the network can carry at most one unit of flow. Although we have stated these results for the undirected case, nothing in any of the proofs demands an undirected network, and these three quantities are equal for directed networks as well.

The equality of the maximum flow, the connectivity, and the cut set size has an important practical consequence. There are simple computer algorithms, such as the augmenting path algorithm of Section 10.5.1, that can calculate maximum flows quite quickly (in polynomial time) for any given network, and the equality means that we can use these same algorithms to quickly calculate a connectivity or the size of a cut set as well. Maximum flow algorithms are now the standard numerical method for connectivities and cut sets.

6.12.1 MAXIMUM FLOWS AND CUT SETS ON WEIGHTED NETWORKS

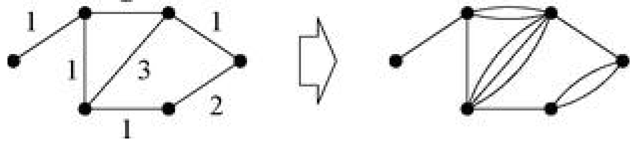
As discussed in Section 6.3, networks can have weights on their edges that indicate that some edges are stronger or more prominent than others. In some cases these weights can represent capacities of the edges to conduct a flow of some kind. For example, they might represent maximum traffic throughput on the roads of a road network or maximum data capacity of Internet lines. We can ask questions about network flows on such networks similar to those we asked in the last section, but with the added twist that different edges can now have different capacities. For example, we can ask what the maximum possible flow is between a specified pair of vertices. We can also ask about cut sets. An edge cut set is defined as before to be a set of edges whose removal from the network would disconnect the specified pair of vertices. A *minimum* edge cut set is defined as being a cut set such that the sum of the weights on the edges of the set has the minimum possible value. Note that it is not now the number of edges that is minimized, but their weight. Nonetheless, this definition is a proper generalization of the one we had before—we can think of the unweighted case as being a special case of the weighted one in which the weights on all edges are equal, and the sum of the weights in the cut set is then simply proportional to the number of edges in the set.

Maximum flows and minimum cut sets on weighted networks are related by the max-flow/min-cut theorem in its most general form:

> The maximum flow between a given pair of vertices in a network is equal to the sum of the weights on the edges of the minimum edge cut set that separates the same two vertices.

We can prove this theorem using the results of the previous section.[69]

Consider first the special case in which the capacities of all the edges in our network are integer multiples of some fixed capacity *r*. We then transform our network by replacing each edge of capacity *kr* (with *k* integer) by *k* parallel edges of capacity *r* each. For instance, if *r* = 1 we would have something like this:



It is clear that the maximum flow between any two vertices in the transformed network is the same as that between the corresponding vertices in the original. At the same time the transformed network now has the form of a simple unweighted network of the type considered in Section 6.12, and hence, from the results of that section, we can immediately say that the maximum flow in the network is equal to the size *in unit edges* of the minimum edge cut set.

We note also that the minimum cut set on the transformed network must include either all or none of the parallel edges between any adjacent pair of vertices; there is no point cutting one such edge unless you cut all of the others as well—you have to cut all of them to disconnect the vertices. Thus the minimum cut set on the transformed network is also a cut set on the original network. And it is a minimum cut set on the original network, because every cut set on the original network is also a cut set with the same weight on the transformed network, and if there were any smaller cut set on the original network then there would be a corresponding one on the transformed network, which, by hypothesis, there is not.

Thus the maximum flows on the two networks are the same, the minimum cuts are also the same, and the maximum flow and minimum cut are equal on the transformed network. It therefore follows that the maximum flow and minimum cut are equal on the original network.

This demonstrates the theorem for the case where all edges are constrained to have weights that are integer multiples of r. This constraint can now be removed, however, by simply allowing r to tend to zero. This makes the units in which we measure edge weights smaller and smaller, and in the limit $r \to 0$ the edges can have any weight—any weight can be represented as a (very large) integer multiple of r—and hence the max-flow/min-cut theorem in the form presented above must be generally true.

Again there exist efficient computer algorithms for calculating maximum flows on weighted networks, so the max-flow/min-cut theorem allows us to calculate minimum cut weights efficiently also, and this is now the standard way of performing such calculations.[70]

6.13 THE GRAPH LAPLACIAN

Section 6.2 introduced an important quantity, the adjacency matrix, which captures the entire structure of a network and whose matrix properties can tell us a variety of useful things about networks. There is another matrix, closely related to the adjacency matrix but differing in some important respects, that can also tell us much about network structure. This is the graph Laplacian.

6.13.1 DIFFUSION

Diffusion is, among other things, the process by which gas moves from regions of high density to regions of low, driven by the relative pressure (or partial pressure) of the different regions. One can also consider diffusion processes on networks, and such processes are sometimes used as a simple model of spread across a network, such as the spread of an idea or the spread of a disease. Suppose we have some commodity or substance of some kind on the vertices of a network and there is an amount ψ_i of it at vertex i. And suppose that the commodity moves along the edges, flowing from one vertex j to an adjacent one i at a rate $C(\psi_j - \psi_i)$ where C is a constant called the *diffusion constant*. That is, in a small interval of time the amount of fluid flowing from j to i is $C(\psi_j - \psi_i)\,\mathrm{d}t$. Then the rate at which ψ_i is changing is given by

$$\frac{\mathrm{d}\psi_i}{\mathrm{d}t} = C\sum_j A_{ij}(\psi_j - \psi_i).$$

(6.37)

The adjacency matrix in this expression insures that the only terms appearing in the sum are those that correspond to vertex pairs that are actually connected by an edge. Equation (6.37) works equally well for both undirected and directed networks, but let us focus here on undirected ones.[71] We will also consider our networks to be simple (i.e., to have at most a single edge between any pair of vertices and no self-edges).

Splitting the two terms in Eq. (6.37), we can write

$$\begin{aligned}\frac{\mathrm{d}\psi_i}{\mathrm{d}t} &= C\sum_j A_{ij}\psi_j - C\psi_i\sum_j A_{ij} = C\sum_j A_{ij}\psi_j - C\psi_i k_i \\ &= C\sum_j (A_{ij} - \delta_{ij}k_i)\psi_j,\end{aligned}$$

(6.38)

where k_i is the degree of vertex i as usual and we have made use of the result $k_i = \sum_j A_{ij}$—see Eq. (6.19). (And δ_{ij} is the Kronecker delta, which is 1 if $i = j$ and 0 otherwise.)

Equation (6.38) can be written in matrix form as

$$\frac{\mathrm{d}\boldsymbol{\psi}}{\mathrm{d}t} = C(\mathbf{A} - \mathbf{D})\boldsymbol{\psi},$$

(6.39)

where ψ is the vector whose components are numbers ψ_i, **A** is the adjacency matrix, and **D** is the diagonal matrix with the vertex degrees along its diagonal:

$$\mathbf{D} = \begin{pmatrix} k_1 & 0 & 0 & \cdots \\ 0 & k_2 & 0 & \cdots \\ 0 & 0 & k_3 & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix}.$$

(6.40)

It is common to define the new matrix

$$\mathbf{L} = \mathbf{D} - \mathbf{A},$$

(6.41)

so that Eq. (6.38) takes the form

$$\frac{\mathrm{d}\psi}{\mathrm{d}t} + C\mathbf{L}\psi = 0,$$

(6.42)

which has the same form as the ordinary diffusion equation for a gas, except that the Laplacian operator ∇^2 that appears in that equation has been replaced by the matrix **L**. The matrix **L** is for this reason called the *graph Laplacian*, although its importance stretches much further than just diffusion processes. The graph Laplacian, as we will see, turns up in a variety of different places, including random walks on networks, resistor networks, graph partitioning, and network connectivity.[72]

Written out in full, the elements of the Laplacian matrix are

$$L_{ij} = \begin{cases} k_i & \text{if } i = j, \\ -1 & \text{if } i \neq j \text{ and there is an edge } (i,j), \\ 0 & \text{otherwise,} \end{cases}$$

(6.43)

so it has the degrees of the vertices down its diagonal and a −1 element for every edge. Alternatively we can write

$$L_{ij} = \delta_{ij} k_i - A_{ij}.$$

(6.44)

We can solve the diffusion equation (6.42) by writing the vector ψ as a linear combination of the eigenvectors $\mathbf{v}_i$ of the Laplacian thus:

$$\psi(t) = \sum_i a_i(t)\, \mathbf{v}_i,$$

(6.45)

with the coefficients $a_i(t)$ varying over time. Substituting this form into (6.42) and making use of $\mathbf{L}\mathbf{v}_i = \lambda_i \mathbf{v}_i$, where λ_i is the eigenvalue corresponding to the eigenvector $\mathbf{v}_i$, we get

$$\sum_i \left(\frac{\mathrm{d}a_i}{\mathrm{d}t} + C\lambda_i a_i \right) \mathbf{v}_i = 0.$$

(6.46)

But the eigenvectors of a symmetric matrix such as the Laplacian are orthogonal, and so, taking the dot product of this equation with any eigenvector $\mathbf{v}_j$, we get

$$\frac{\mathrm{d}a_i}{\mathrm{d}t} + C\lambda_i a_i = 0,$$

(6.47)

for all i, which has the solution

$$a_i(t) = a_i(0)\, \mathrm{e}^{-C\lambda_i t}.$$

(6.48)

Given an initial condition for the system, as specified by the quantities $a_i(0)$, therefore, we can solve for the state at any later time, provided we know the eigenvalues and eigenvectors of the graph Laplacian.

6.13.2 EIGENVALUES OF THE GRAPH LAPLACIAN

This is the first of many instances in which the eigenvalues of the Laplacian will arise, so it is worth spending a little time understanding their properties. The Laplacian is a symmetric matrix, and so has real eigenvalues. However, we can say more than this about them. In fact, as we now show, all the eigenvalues of the Laplacian are also non-negative.

Consider an undirected network with n vertices and m edges and let us arbitrarily designate one end of each edge to be end 1 and the other to be end 2. It doesn't matter which end is which, only that they have different labels.

Now let us define an $m \times n$ matrix **B** with elements as follows:

$$B_{ij} = \begin{cases} +1 & \text{if end 1 of edge } i \text{ is attached to vertex } j, \\ -1 & \text{if end 2 of edge } i \text{ is attached to vertex } j, \\ 0 & \text{otherwise.} \end{cases}$$

(6.49)

Thus each row of the matrix has exactly one +1 and one −1 element.

The matrix **B** is called the *edge incidence matrix*. It bears some relation to, but is distinct from, the incidence matrix for a bipartite graph defined in Section 6.6.

Now consider the sum $\sum_k B_{ki} B_{kj}$. If $i \neq j$, then the only non-zero terms in the sum will occur if both B_{ik} and B_{jk} are non-zero, i.e., if edge k connects vertices i and j, in which case the product will have value −1. For a simple network, there is at most one edge between any pair of vertices and hence at most one such non-zero term, so the value of the entire sum will be −1 if there is an edge between i and j and zero otherwise.

If $i = j$ then the sum is $\sum_k B_{ki}^2$, 2 which has a term +1 for every edge connected to vertex i, so the whole sum is just equal to the degree k_i of vertex i.

Thus the sum $\sum_k B_{ki} B_{kj}$ is precisely equal to an element of the Laplacian $\sum_k B_{ki} B_{kj} = L_{ij}$—the diagonal terms L_{ii} are equal to the degrees k_i and the off-diagonal terms L_{ij} are—1 if there is an edge (i, j) and zero otherwise. (See Eq. (6.43).) In matrix form we can write

$$\mathbf{L} = \mathbf{B}^T \mathbf{B},$$

(6.50)

where $\mathbf{B}^T$ is the transpose of **B**.

Now let $\mathbf{v}_i$ be an eigenvector of **L** with eigenvalue λ_i. Then

$$\mathbf{v}_i^T \mathbf{B}^T \mathbf{B} \mathbf{v}_i = \mathbf{v}_i^T \mathbf{L} \mathbf{v}_i = \lambda_i \mathbf{v}_i^T \mathbf{v}_i = \lambda_i,$$

(6.51)

where we assume that the eigenvector $\mathbf{v}_i$ is normalized so that its inner product with itself is 1.

Thus any eigenvalue λ_i of the Laplacian is equal to $(\mathbf{v}_i^T\mathbf{B}^T)(\mathbf{B}\mathbf{v}_i)$. But this quantity is itself just the inner product of a real vector $(\mathbf{B}\mathbf{v}_i)$ with itself. In other words, it is the sum of the squares of the (real) elements of that vector and hence it cannot be negative. The smallest value it can have is zero:

$$\lambda_i \geq 0$$

(6.52)

for all i.

This is an important physical property of the Laplacian. It means, for instance, that the solution, Eq. (6.48), of the diffusion equation on any network contains only decaying exponentials or constants and not growing exponentials, so that the solution tends to an equilibrium value as $t \to \infty$, rather than diverging.[73]

While the eigenvalues of the Laplacian cannot be negative, they can be zero, and in fact the Laplacian always has at least one zero eigenvalue. Consider the vector $\mathbf{1} = (1,1,1,\ldots)$. If we multiply this vector by the Laplacian, the ith element of the result is given by

$$\begin{aligned}\sum_j L_{ij} \times 1 &= \sum_j (\delta_{ij}k_i - A_{ij}) = k_i - \sum_j A_{ij} = k_i - k_i \\ &= 0,\end{aligned}$$

(6.53)

where we have made use of Eqs. (6.19) and (6.44). In vector notation, $\mathbf{L} \cdot \mathbf{1} = 0$. Thus the vector $\mathbf{1}$ is always an eigenvector of the graph Laplacian with eigenvalue zero.[74] Since there are no negative eigenvalues, this is the lowest of the eigenvalues of the Laplacian. Following convention, we number the n eigenvalues of the Laplacian in ascending order: $\lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n$. So we always have $\lambda_1 = 0$.

Note that the presence of a zero eigenvalue implies that the Laplacian has no inverse: the determinant of the matrix is the product of its eigenvalues, and hence is always zero for the Laplacian, so that the matrix is singular.

6.13.3 COMPONENTS AND THE ALGEBRAIC CONNECTIVITY

See the discussion of block diagonal matrices in Section 6.11.

Suppose we have a network that is divided up into c different components of sizes n_1, n_2, ..., n_c. To make the notation simple let us number the vertices of the network so that the first n_1 vertices are those of the first component, the next n_2 are those of the second component, and so forth. With this choice the Laplacian of the network will be block diagonal, looking something like this:

$$\mathbf{L} = \begin{pmatrix} \boxed{} & 0 & \cdots \\ 0 & \boxed{} & \cdots \\ \vdots & \vdots & \ddots \end{pmatrix}.$$

(6.54)

What is more, each block in the Laplacian is, by definition, the Laplacian of the corresponding component: it has the degrees of the vertices in that component along its diagonal and −1 in each position corresponding to an edge within that component. Thus we can immediately write down c different vectors that are eigenvectors of **L** with eigenvalue zero: the vectors that have ones in all positions corresponding to vertices in a single component and zero elsewhere. For instance, the vector

$$\mathbf{v} = (\underbrace{1,1,1,\ldots}_{n_1 \text{ ones}}, \underbrace{0,0,0,\ldots}_{\text{zeros}}),$$

(6.55)

is an eigenvector with eigenvalue zero.

Thus in a network with c components there are always at least c eigenvectors with eigenvalue zero. In fact, it can be shown that the number of zero eigenvalues is always exactly equal to the number of components [324]. (Note that the vector **1** of all ones is just equal to the sum of the c other eigenvectors, so it is not an independent eigenvector.) An important corollary of this result is that the second eigenvalue of the graph Laplacian λ_2 is non-zero if and only if the network is connected, i.e., consists of a single component. The second eigenvalue of the Laplacian is called the *algebraic connectivity* of the network.[75] It will come up again in Section 11.5 when we look at the technique known as spectral partitioning.

6.14 RANDOM WALKS

Another context in which the graph Laplacian arises is in the study of random walks on networks. A *random walk* is a path across a network created by taking repeated random steps. Starting at some specified initial vertex, at each step of the walk we choose uniformly at random between the edges attached to the current vertex, move along the chosen edge to the vertex at its other end, and repeat. Random walks are normally allowed to go along edges more than once, visit vertices more than once, or retrace their steps along an edge just traversed. *Self-avoiding walks*, which do none of these things, are also studied sometimes, but we will not discuss them here.

Random walks arise, for instance, in the random walk sampling method for social networks discussed in Section 3.7 and in the random walk betweenness measure of Section 7.7.

Consider a random walk that starts at a specified vertex and takes t random steps. Let $p_i(t)$ be the probability that the walk is at vertex i at time t. If the walk is at vertex j at time $t - 1$, the probability of taking a step along any particular one of the k_j edges attached to j is $1/k_j$, so on an undirected network $p_i(t)$ is given by

$$p_i(t) = \sum_j \frac{A_{ij}}{k_j} p_j(t-1),$$

(6.56)

or $\mathbf{p}(t) = \mathbf{A}\mathbf{D}^{-1}\mathbf{p}(t-1)$ in matrix form where $\mathbf{p}$ is the vector with elements p_i and, as before, $\mathbf{D}$ is the diagonal matrix with the degrees of the vertices down its diagonal.

There are a couple of other useful ways to write this relation. One is to define $\mathbf{D}^{1/2}$ to be the matrix with the square roots $\sqrt{k_i}$ of the degrees down the diagonal, so that

$$\mathbf{D}^{-1/2}\mathbf{p}(t) = [\mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}][\mathbf{D}^{-1/2}\mathbf{p}(t-1)].$$

(6.57)

This form is convenient in some situations because the matrix $\mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$ is a symmetric one. This matrix is called the *reduced adjacency matrix* and has elements equal to $1/\sqrt{k_i k_j}$ if there is an edge between i and j and zero otherwise. Equation (6.57) tells us that the vector $\mathbf{D}^{-1/2}\mathbf{p}$ gets multiplied by one factor of the reduced adjacency matrix at each step of the random walk, and so the problem of understanding the random walk can be reduced to one of understanding the effects of repeated multiplication by a simple symmetric matrix.

For our purposes, however, we take a different approach. In the limit as $t \to \infty$ the probability distribution over vertices is given by setting $t = \infty$: $p_i(\infty) = \sum_j A_{ij} p_j(\infty)/k_j$, or in matrix form:

$$\mathbf{p} = \mathbf{A}\mathbf{D}^{-1}\mathbf{p}.$$

(6.58)

Rearranging, this can also be written as

$$(\mathbf{I} - \mathbf{A}\mathbf{D}^{-1})\mathbf{p} = (\mathbf{D} - \mathbf{A})\mathbf{D}^{-1}\mathbf{p} = \mathbf{L}\mathbf{D}^{-1}\mathbf{p} = 0.$$

(6.59)

Thus $\mathbf{D}^{-1}\mathbf{p}$ is an eigenvector of the Laplacian with eigenvalue 0.

On a connected network, for instance—one with only a single component—we know (Section 6.13.3) that there is only a single eigenvector with eigenvalue zero, the vector whose components are all equal. Thus, $\mathbf{D}^{-1}\mathbf{p} = a\mathbf{1}$, where a is a constant and $\mathbf{1}$ is the vector whose components are all ones. Equivalently $\mathbf{p} = a\mathbf{D}\mathbf{1}$, so that $p_i = ak_i$. Then on a connected network the probability that a random walk will be found at vertex i in the limit of long time is simply proportional to the degree of that vertex. If we choose the value of a to normalize p_i properly, this gives

$$p_i = \frac{k_i}{\sum_j k_j} = \frac{k_i}{2m},$$

(6.60)

where we have used Eq. (6.20).

The simple way to understand this result is that vertices with high degree are more likely to be visited by the random walk because there are more ways of reaching them. We used Eq. (6.60) in Section 3.7 in our analysis of the random-walk sampling method for social networks.

An important question about random walks concerns the *first passage time*. The first passage time for a random walk from a vertex u to another vertex v is the number of steps before a walk starting at u first reaches v. Since the walk is random, the first passage time between two vertices is not fixed; if we repeat the random walk process more than once it can take different values on different occasions. But we can ask for example what the mean first passage time is.

To answer this question, we modify our random walk slightly to make it into an *absorbing random walk*. An absorbing walk is one that has one or more absorbing states, meaning vertices that the walk can move to, but not leave again. We will consider just the simplest case of a single absorbing vertex v. Any walk that arrives at vertex v must stay there ever afterwards, but on the rest of the network the walk is just a normal random walk. We can answer questions about the first passage time by considering the probability $p_v(t)$ that a walk is at vertex v after a given amount of time, since this is also the probability that the walk has a first passage time to v that is less than or equal to t. And the probability that a walk has first passage time exactly t is $p_v(t) - p_v(t - 1)$, which means that the mean first passage time τ is

$$\tau = \sum_{t=0}^{\infty} t\big[p_v(t) - p_v(t-1)\big].$$

(6.61)

To calculate the probability $p_v(t)$ we could apply Eq. (6.56) (or (6.58)) repeatedly to find $\mathbf{p}(t)$ and substitute the result into Eq. (6.61). Note, however, that since the random walk can move *to* vertex v but not away from it, the adjacency matrix $\mathbf{A}$ has elements $A_{iv} = 0$ for all i but A_{vi} can still be non-zero. Thus in general $\mathbf{A}$ is asymmetric. Although we can work with such an asymmetric matrix, the computations are harder than for symmetric matrices and in this case there is no need. Instead we can use the following trick.

Consider Eq. (6.56) for any $i \neq v$:

$$p_i(t) = \sum_j \frac{A_{ij}}{k_j} p_j(t-1) = \sum_{j \neq v} \frac{A_{ij}}{k_j} p_j(t-1),$$

(6.62)

where the second equality applies since $A_{iv} = 0$ and hence the terms with $j = v$ don′t contribute to the sum. But if $i \neq v$ then there are no terms in A_{vj} in the sum either. This allows us to write the equation in the matrix form

$$\mathbf{p}'(t) = \mathbf{A}'\mathbf{D}'^{-1}\mathbf{p}'(t-1),$$

(6.63)

where $\mathbf{p}'$ is $\mathbf{p}$ with the vth element removed and $\mathbf{A}'$ and $\mathbf{D}'$ are $\mathbf{A}$ and $\mathbf{D}$ with their vth row and column removed. Note that $\mathbf{A}'$ and $\mathbf{D}'$ are symmetric matrices, since the rows and columns containing the asymmetric elements have been removed. Iterating Eq. (6.63), we now get

$$\mathbf{p}'(t) = [\mathbf{A}'\mathbf{D}'^{-1}]^t\mathbf{p}'(0).$$

(6.64)

Since we have removed the element p_v from the vector $\mathbf{p}$, we cannot calculate its value directly using this equation, but we can calculate it indirectly by noting that $\sum_i p_i(t) = 1$ at all times. Thus

$$p_v(t) = 1 - \sum_{i \neq v} p_i(t) = 1 - \mathbf{1} \cdot \mathbf{p}'(t),$$

(6.65)

where again $\mathbf{1}$ = (1, 1, 1, ...). Using Eqs. (6.61), (6.64), and (6.65) we then have a mean first passage time of

$$\tau = \sum_{t=0}^{\infty} t\, \mathbf{1} \cdot [\mathbf{p}'(t-1) - \mathbf{p}'(t)] = \mathbf{1} \cdot \left[\mathbf{I} - \mathbf{A}'\mathbf{D}'^{-1}\right]^{-1} \cdot \mathbf{p}'(0),$$

(6.66)

where $\mathbf{I}$ is the identity matrix and we have made use of the result that

$$\sum_{t=0}^{\infty} t\,(\mathbf{M}^{t-1} - \mathbf{M}^{t}) = [\mathbf{I} - \mathbf{M}]^{-1},$$

(6.67)

for any matrix $\mathbf{M}$ (assuming the sum actually converges).
We can simplify Eq. (6.66) by writing

$$[\mathbf{I} - \mathbf{A}'\mathbf{D}'^{-1}]^{-1} = \mathbf{D}'[\mathbf{D}' - \mathbf{A}']^{-1} = \mathbf{D}'\mathbf{L}'^{-1},$$

(6.68)

so that

$$\tau = \mathbf{1} \cdot \mathbf{D}'\mathbf{L}'^{-1} \cdot \mathbf{p}'(0),$$

(6.69)

where the symmetric matrix $\mathbf{L}'$ is the graph Laplacian with the vth row and column removed. $\mathbf{L}'$ is called the *vth reduced Laplacian*. Note that, even though, as we noted in Section 6.13.2, the Laplacian has no finite inverse, the reduced Laplacian can have an inverse. The eigenvector (1, 1, 1, ...) whose zero eigenvalue causes the determinant of the Laplacian to be zero is, in general, not an eigenvector of the reduced matrix.

For convenience, we now introduce the symmetric matrix $\mathbf{\Lambda}^{(v)}$, which is equal to $\mathbf{L}'^{-1}$ with a vth row and column reintroduced having elements all zero:

$$\Lambda_{ij}^{(v)} = \begin{cases} 0 & \text{if } i = v \text{ or } j = v, \\ \left[\mathbf{L}'^{-1}\right]_{ij} & \text{if } i < v \text{ and } j < v, \\ \left[\mathbf{L}'^{-1}\right]_{i-1,j} & \text{if } i > v \text{ and } j < v, \\ \left[\mathbf{L}'^{-1}\right]_{i,j-1} & \text{if } i < v \text{ and } j > v, \\ \left[\mathbf{L}'^{-1}\right]_{i-1,j-1} & \text{if } i > v \text{ and } j > v. \end{cases}$$

(6.70)

Then we observe that for a walk starting at vertex u, the initial probability distribution $\mathbf{p}'(0)$ has all elements 0 except the one corresponding to vertex u, which is 1. Thus, combining Eqs. (6.69) and (6.70), the mean first passage time for a random walk from u to v is given by

$$\tau = \sum_i k_i \Lambda_{iu}^{(v)} ,$$

(6.71)

where we have made use of the fact that the non-zero elements of the diagonal matrix $\mathbf{D}'$ are the degrees k_i of the vertices. Thus if we can calculate the inverse of the vth reduced Laplacian then a sum over the elements in the uth column immediately gives us the mean first passage time for a random walk from u to v. And sums over the other columns give us the first passage times from other starting vertices to the same target vertex v—we get n first passage times from a single matrix inversion.

6.14.1 RESISTOR NETWORKS

There are interesting mathematical connections between random walks on networks and the calculation of current flows in networks of resistors. Suppose we have a network in which the edges are identical resistors of resistance *R* and the vertices are junctions between resistors, as shown in Fig. 6.16, and suppose we apply a voltage between two vertices *s* and *t* such that a current *I* flows from *s* to *t* through the network. What then is the current flow through any given resistor in the network?

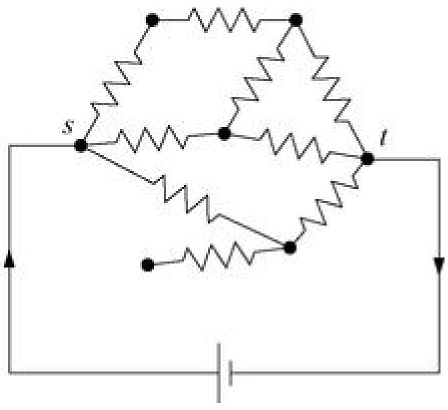


Figure 6.16: A resistor network with applied voltage. A network in which the edges are resistors and the vertices are electrical junctions between them, with a voltage applied between vertices *s* and *t* so as to generate a total current *I*.

The currents in the network obey Kirchhoff's current law, which is essentially a statement that electricity is conserved, so that the net current flowing in or out of any vertex is zero. Let V_i be the voltage at vertex *i*, measured relative to any convenient reference potential. Then Kirchhoff's law says that

$$\sum_j A_{ij} \frac{V_j - V_i}{R} + I_i = 0,$$

(6.72)

where I_i represents any current injected into vertex *i* by an external current source. In our case this external current is non-zero only for the two vertices *s* and *t* connected to the external voltage:

$$I_i = \begin{cases} +I & \text{for } i = s, \\ -I & \text{for } i = t, \\ 0 & \text{otherwise.} \end{cases}$$

(6.73)

(In theory there's no reason why one could not impose more complex current source arrangements by applying additional voltages to the network and making more elements I_i non-zero, but let us stick to our simple case in this discussion.)

Noting that $\sum_j A_{ij} = k_i$, Eq. (6.72) can also be written as $k_i V_i - \sum_j A_{ij} V_j = RI_i$ or

$$\sum_j (\delta_{ij} k_i - A_{ij}) V_j = RI_i,$$

(6.74)

which in matrix form is

$$\mathbf{LV} = R\mathbf{I},$$

(6.75)

where **L** = **D - A** is once again the graph Laplacian.

As discussed in Section 6.13.2, the Laplacian has no inverse because it always has at least one eigenvalue that is zero, so we cannot simply invert Eq. (6.75) to get the voltage vector **V**. We can, however, solve for **V** by once again making use of the reduced Laplacian of Section 6.14.

The reason why we cannot invert Eq. (6.75) is that the equation does not in fact fix the absolute value of the voltages V_i. We can add any multiple of the vector **1** = (1, 1, 1, ...) to the solution of this equation and get another solution, since **1** is an eigenvector of **L** with eigenvalue zero:

$$\mathbf{L}(\mathbf{V} + c\mathbf{1}) = \mathbf{LV} + \mathbf{L1} = \mathbf{LV} = R\mathbf{I}.$$

(6.76)

In physical terms these different solutions correspond to different choices of the reference potential against which we measure our voltages. The actual currents flowing around the system are identical no matter what reference potential we choose. If we fix our reference potential at a particular value, then we will fix the solution for the voltages as well, and our equation for **V** will become solvable.

Let us choose, arbitrarily, to set our reference potential equal to the potential at the target vertex *t* where the current exits the network. (We could choose any other vertex just as well, but this choice is the simplest.) That is, the voltage at this vertex is chosen to be zero and all others are measured in terms of their potential difference from vertex *t*. But now we can remove the element $V_t = 0$ from **V** in Eq. (6.75), along with the corresponding column *t* in the Laplacian, without affecting the result, since they contribute zero to the matrix multiplication anyway. And we can also remove row t from both sides of the equation, since we already know the value of V_t, so there's no need to calculate it. That leaves us with a modified equation $\mathbf{L}'\mathbf{V}' = R\mathbf{I}'$, with $\mathbf{L}'$ being the tth reduced Laplacian, which in general has a well-defined inverse. Then

$$\mathbf{V}' = R\mathbf{L}'^{-1}\mathbf{I}',$$

(6.77)

and once we have the voltages we can calculate in a straightforward manner any other quantity of interest, such as the current along a given edge in the network.

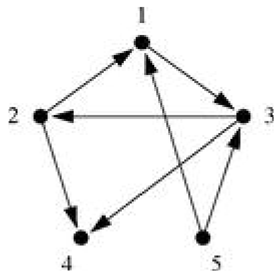
Note that, for the simple case discussed here in which current is injected into the network at just one vertex and removed at another, $\mathbf{I}'$ has only one non-zero element. (The other one, I_t, has been removed.) Thus the vector $\mathbf{V}'$ on the left-hand side of Eq. (6.77) is simply proportional to the column of the inverse reduced Laplacian corresponding to vertex s. To use the notation of Section 6.14, if $\mathbf{\Lambda}^{(t)}$ is the inverse of the tth reduced Laplacian with the tth row and column reintroduced having elements all zero (see Eq. (6.70)), then

$$V_i = RI\Lambda_{is}^{(t)} .$$

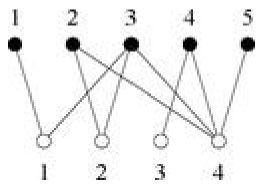
(6.78)

PROBLEMS

6.1 Consider the following two networks:



(a)



(b)

Network (a) is a directed network. Network (b) is undirected but bipartite. Write down:

a. the adjacency matrix of network (a);
b. the cocitation matrix of network (a);
c. the incidence matrix of network (b);
d. the projection matrix (Eq. (6.17)) for the projection of network (b) onto its black vertices.

6.2 Let **A** be the adjacency matrix of an undirected network and **1** be the column vector whose elements are all 1. In terms of these quantities write expressions for:

a. the vector **k** whose elements are the degrees k_i of the vertices;
b. the number m of edges in the network;
c. the matrix **N** whose element N_{ij} is equal to the number of common neighbors of vertices i and j;
d. the total number of triangles in the network, where a triangle means three vertices, each connected by edges to both of the others.

6.3 Consider an acyclic directed network of n vertices, labeled $i = 1 \ldots n$, and suppose that the labels are assigned in the manner of Fig. 6.3 on page 119, such that all edges run from vertices with higher labels to vertices with lower.

a. Find an expression for the total number of edges ingoing to vertices $1 \ldots r$ and another for the total number of edges outgoing from vertices $1 \ldots r$, in terms of the in- and out-degrees k_i^{in} and k_i^{out} of the vertices.
b. Hence find an expression for the total number of edges running to vertices $1 \ldots r$ from vertices $r + 1 \ldots n$.
c. Show that in any acyclic network the in- and out-degrees must satisfy

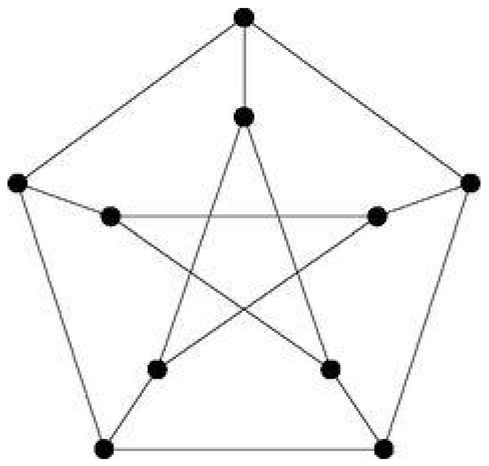
$$k_r^{\text{out}} \leq \sum_{i=1}^{r-1} (k_i^{\text{in}} - k_i^{\text{out}}),$$

for all r.

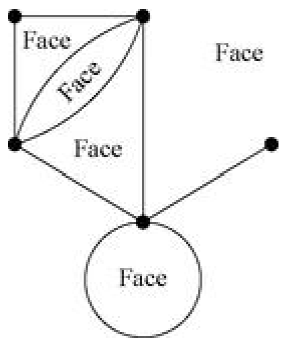
6.4 Consider a bipartite network, with its two types of vertex, and suppose that there are n_1 vertices of type 1 and n_2 vertices of type 2. Show that the mean degrees c_1 and c_2 of the two types are related by

$$c_2 = \frac{n_1}{n_2} c_1.$$

6.5 Using Kuratowski's theorem, prove that this network is not planar:



6.6 Consider a connected planar network with n vertices and m edges. Let f be the number of "faces" of the network, i.e., areas bounded by edges when the network is drawn in planar form. The "outside" of the network, the area extending to infinity on all sides, is also considered a face. The network can have multiedges and self-edges:



a. Write down the values of *n*, *m*, and *f* for a network with a single vertex and no edges.
b. How do *n*, *m*, and *f* change when we add a single vertex to the network along with a single edge attaching it to another vertex?
c. How do *n*, *m*, and *f* change when we add a single edge between two extant vertices (or a self-edge attached to just one vertex), in such a way as to maintain the planarity of the network?
d. Hence by induction prove a general relation between *n*, *m*, and *f* for all connected planar networks.
e. Now suppose that our network is simple (i.e., it contains no multiedges or self-edges). Show that the mean degree *c* of such a network is strictly less than six.

6.7 Consider the set of all paths from vertex *s* to vertex *t* on an undirected graph with adjacency matrix **A**. Let us give each path a weight equal to α^r, where *r* is the length of the path.

a. Show that the sum of the weights of all the paths from *s* to *t* is given by Z_{st} which is the *st* element of the matrix $\mathbf{Z} = (\mathbf{I} - \alpha\mathbf{A})^{-1}$, where **I** is the identity matrix.
b. What condition must α satisfy for the sum to converge?
c. Hence, or otherwise, show that the length ℓ_{st} of a geodesic path from *s* to *t*, if there is one, is

$$\ell_{st} = \lim_{\alpha\to 0} \frac{\partial \log Z_{st}}{\partial \log \alpha}.$$

6.8 What is the difference between a 2-component and a 2-core? Draw a small network that has one 2-core but two 2-components.

6.9 In Section 5.3.1, we gave one possible definition of the trophic level x_i of a species in a (directed) food web as the mean of the trophic levels of the species' prey, plus one.

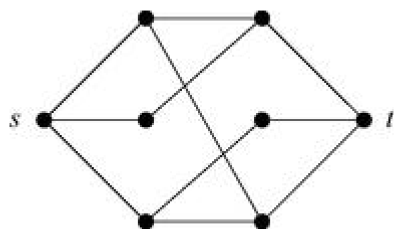
a. Show that x_i, when defined in this way, is the *i*th element of the vector

$$\mathbf{x} = \mathbf{D} - \mathbf{A}^{-1}\mathbf{D} \cdot \mathbf{1},$$

where **D** is the diagonal matrix of in-degrees, **A** is the (asymmetric) adjacency matrix, and $\mathbf{1} = (1, 1, 1, ...)$.
b. This expression does not work for autotrophs—species with no prey—because the

corresponding vector element diverges. Such species are usually given a trophic level of one. Suggest a modification of the calculation that will correctly assign trophic levels to these species, and hence to all species.

6.10 What is the size *k* of the minimum vertex cut set between *s* and *t* in this network?



Prove your result by finding one possible cut set of size *k* and one possible set of *k* independent paths between *s* and *t*. Why do these two actions constitute a proof that the minimum cut set has size *k*?