Chapter 3

Mathematics of Networks

So far we have talked about real-world networks from various contexts, how they operate, and how to measure them. However, just like any other physical system, qualitative description of networks is just the first step towards understanding them better. As discussed in previous chapters, one of the primary reasons to understand real-world networked systems is to be able to control and manipulate them for our benefits. But this is possible only if we could understand them quantitatively i.e using numbers. In fact, when we decide to collect data about any real-world system, the first step is to decide what to measure. This means that we must think of characteristics and properties of the system that can be measured. Often many of these are numerical. Consider a concrete example of collecting data about birds in your city. In this case, you may decide to measure or observe characteristics like the color, size, habitat, amount of food a typical bird from a species needs every day, length of its beak, number of years it survives, how far it goes in search of its food and so on. In this case, the characteristics like size, amount of food, length of beak, its lifespan must be represented by numbers. It may not be obvious that even characteristics like color and habitat could sometimes be represented by numbers, and the experimenter may or may not decide to do that depending on the question of interest.

3.1 Adjacency matrix

What about networks? As the most fundamental quantities, we must find out the number of vertices and edges a given network has [1]. But that's certainly not enough; there could certainly be many networks that have the exact same number of nodes and/or links and still differ substantially from each other. For a network, what matters is how those edges are distributed among the nodes: which node is connected to which other node? Let us start by considering an undirected network. We must first have some labelling for the vertices which will let us identify different vertices uniquely. The most common way of labelling is to assign the vertices numbers $1, 2, 3, \cdots, n$ or $0, 1, 2, \cdots, n-1$. Then we construct a square matrix A of size $n \times n$, called adjacency matrix, so that

$$A_{ij} = \begin{cases} 1 & \text{if} \quad i \text{ is connected to } j \\ 0 & \text{otherwise} \end{cases} \tag{3.1}$$

That is, the $(i, j)^{\text{th}}$ element of the matrix A is 1 if and only if the vertices i and j are connected by an edge, and 0 if they are not connected. Thus, the adjacency matrix for an undirected network is a binary matrix. It should also be obvious that the adjacency matrix for an undirected graph is symmetric i.e. $A^T = A$ because whenever i is connected to j, automatically j is connected to i, and hence $A_{ij} = A_{ji}$.

[1]We will henceforth always represent the number of vertices by n and the number of edges by m

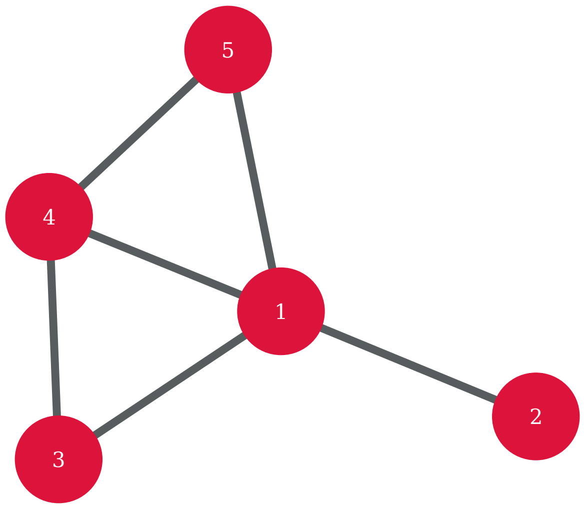


Figure 3.1: A small network with 5 vertices and 6 edges

Fig. 3.1 shows a small network with just 5 nodes, and Eq(3.2) gives its adjacency matrix. (You should verify that the matrix is correct!). Observe the zeros on the main diagonal and the symmetry of the matrix.

$$A = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \end{bmatrix} \tag{3.2}$$

In an undirected network, a pair of vertices need not have at most one link connecting them, and can potentially have more. For example, in a social network, two humans may have more than one relation between them, say friendship and a family relationship. It makes sense to represent such multiple relationships by two edges between the same pair of vertices. Or instead of saying that there are two edges, we can think of this as a special type of edge called **multiedge**. A graph that contains one or more multiedges is called a **multigraph**. Also, sometimes there could be an edge which connects a node to itself. Such edge is called a **self-loop**. A graph that doesn't contain any self-loops or multiedges is called a **simple graph**.

It should be obvious that in simple graph, all the entries on the main diagonal of the adjacency matrix are always zero since no node is connected to itself. In a multigraph, whenever the number of edges between i and j is greater than 1, we can reasonably define the corresponding A_{ij} to be the multiplicity of the edge: if there are 3 edges between i and j, then $A_{ij} = 3$. Self-loops are more tricky: it may sound natural to make $A_{ii} = 1$ whenever i is connected to itself by a self-loop. But this is wrong! We represent a self-loop by setting $A_{ii} = 2$. This convention makes the matrix equations (which we will study throughout the course) for networks consistent so that the same equation is applicable to simple graphs as well as multigraphs. If you are still not convinced , observe that both ends of the same edge are connected to the same node, and hence both the ends should be counted in A_{ii}. Another way of looking at it is that each edge in the network is represented twice in the adjacency matrix: once at (i, j) and once at (j, i). Since self-edge has only one place on the diagonal, it must be represented twice at the same place. This also means that if there are two self-loops attached to the same node, $A_{ii} = 4$ and so on. Fig. 3.2 depicts a multigraph with 5 vertices, and Eq(3.3) gives its adjacency matrix (Again, check it!).

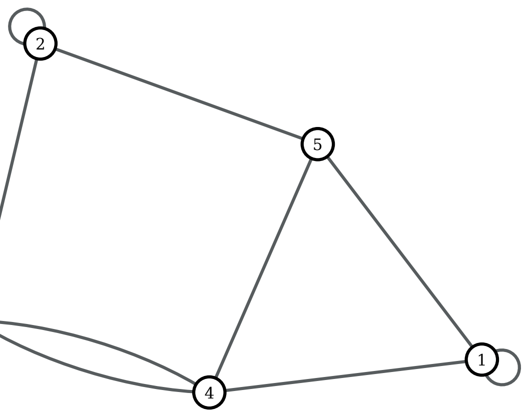


Figure 3.2: An undirected multigraph with 5 nodes. Observe the self-loops on vertex 1 and vertex 2.

$$A = \begin{bmatrix} 2 & 0 & 0 & 1 & 1 \\ 0 & 2 & 1 & 0 & 1 \\ 0 & 1 & 0 & 2 & 0 \\ 1 & 0 & 2 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \end{bmatrix} \tag{3.3}$$

In many real-world networks, edges have directions: person i may know person j, but not the other way round or webpage i may have a hyperlink to go to page j but j may not have a hyperlink to i. We can represent directions on such edges by drawing a small arrow at the end of the edge (i.e. near j if the direction is from i to j). A network with directed edges is called a **directed network**. Fig. 3.3 shows a visual of a small directed graph.

For a directed network, the adjacency matrix is defined so that $A_{ij} = 1$ if and only if **vertex** j **is connected to vertex** i. We could also have defined this the other way round (i to j), but several matrix equations simplify if we adopt this convention. In a directed network, if the vertex i points to vertex j, there is no guarantee that j is also connected to i by another directed edge, and hence in general $A_{ij} \neq A_{ji}$. Thus, the adjacency matrix of a directed graph is in general asymmetric. Similar to the undirected cases, multiedges in the directed graph are represented in the adjacency matrix by setting the matrix elements equal to their multiplicities. But a self-loop in the directed graph are represented by setting $A_{ij} = 1$, not 2 (Think about this).

Along with being directed, many networks contain edges that have numerical values called **weights**. In a social network, an edge with higher weight may be used to represent a stronger friendship or in a transport network it may represent amount of traffic on the highway. A network with weighted edges is naturally called **weighted network**. Notice that a given network may (and often is) directed and weighted at the same time. Visually we may represent edge weights by making widths of edges proportional to them. We can think of an unweighted network as a weighted network where all the edges have the same weights that are scaled to 1. The element A_{ij} of the adjacency matrix for a weighted graph is thus equal to the weight of the edge between i and j.

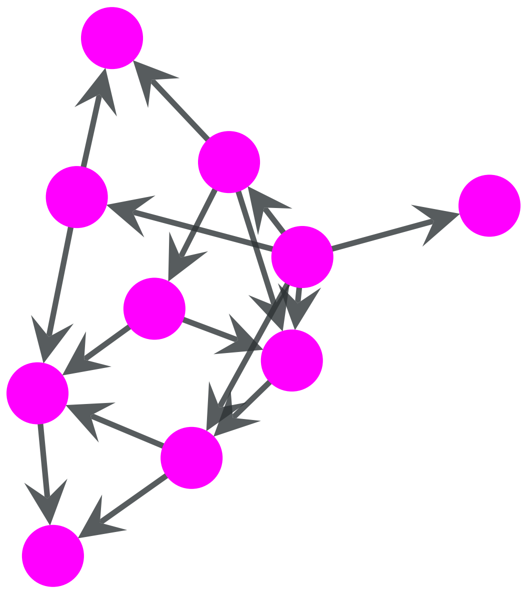


Figure 3.3: A directed network

3.2 Hypergraphs and Bipartite networks

So far, we have been thinking of an edge as something that connects two nodes in a network. However, it is common to have an interaction between more than two nodes in many networks. For example, when three friends are sitting in a restaurant, chatting, each of them interacts with the other two simultaneously. Although we can represent these interactions using three separate links (between A and B, B and C, and A and C), this representation misses the information about the fact that the nodes in question were interacting *together*. Hence, it would be much more appropriate to represent their interaction by a single edge that connects all three. In other words, here we are generalizing the definition of an edge to an object that can connect more than two nodes. Visually we can represent this edge say by drawing a loop around the nodes which are part of the common interaction. Such networks in which edges are generalized objects that can connect more than two nodes are called as **hypergraphs** and the edges in hypergraphs are called **hyperedges**. You can think of other such interactions in various networks. Families can be represented in social networks by hyperedges because everybody in a family bears the same type of relation with others, that of being a member of that family. When more than two proteins interact with each other two form protein complexes, their interaction can be represented by a hyperedge. Although this looks like an important representation to account for group interactions in networks, fortunately we don't have to keep thinking in terms of hyperedges since a much better representation of group interactions exists in the form of what are known as **bipartite networks**.

We have already seen many examples of bipartite networks. A bipartite network consists of two types of nodes and edges in the network connect only the nodes of different types. For example, a network of customers who buy different products from a store and the products they buy form a network in which customers as well as products are nodes. A link between a customer and a product represents the fact that the customer has bought the product at some point in the past. The two types of nodes in a bipartite network are also sometimes called "nodes" and "groups" since usually a bipartite network represents a set of nodes and their memberships (think of students in a college and their friend circles: a student usually belongs to several friend circles). Although this terminology is somewhat confusing, it is so common that we will also use it here. Suppose then that we have a bipartite network with g groups and n groups. We can mathematically represent this network using a $n_1 \times n_2$ matrix instead of a square matrix. If the node j in the network belongs to to the group i in this network, then we make $(i, j)^{\text{th}}$ element of this matrix 1. If not, then the corresponding element is represented by 0. Such matrix used to represent a bipartite network is called an **incidence matrix**.

3.2.1 Projection of a bipartite network

Although a bipartite network is a better representation of many networked systems, it is also somewhat inconvenient because of two types of vertices. Also, often researchers are interested in only one of the two types of vertices in a bipartite graph. In this case, another network with only one of the two types of vertices can be constructed that contains structural information about the original bipartite network, but that is more convenient to study. Such network is called projection of the original bipartite network. Since there are two types of vertices, we actually get two types of projections: projections on the set of vertices, and projection on the set of groups.

Let J denote an incidence matrix of a bipartite graph with g groups and n vertices. Suppose that we want to project this graph on to the set of vertices. If the vertices j and k both belong to group i, then both A_{ij} and A_{ik} would be 1. Hence the product $A_{ij}A_{ik}$ would be 1 if and only if both j and k belong to i. Hence the total number of groups to which both j and k belong is given by:

$$P_{jk} = \sum_{i=1}^{g} A_{ij}A_{ik} = \sum_{i=1}^{g} A_{ji}^{T}A_{ik}$$

Here A_{ji}^{T} denotes $(j,i)^{\text{th}}$ element of the transpose of the adjacency matrix. This can be written in the matrix form as:

$$\boldsymbol{P} = \boldsymbol{A}^T\boldsymbol{A}$$

The matrix $\boldsymbol{P}$ is an adjacency matrix of a weighted network in which the vertices are the vertices from the original bipartite graph, and weighted edges represent the number of common groups between given two vertices. The only problem is that the diagonal elements of $\boldsymbol{P}$ are not zero, and hence it can't be completely regarded as an adjacency matrix of a graph. In fact:

$$P_{jj} = \sum_{i=1}^{g} A_{ij}A_{ij} = \sum_{i=1}^{g} A_{ij}^2 = \sum_{i=1}^{g} A_{ij} = k_j$$

where k_j is the degree of vertex j and we have used the fact that square of any element of the incidence matrix is same as the value of the element because each element is either 0 or 1. Thus, the diagonal elements in $\boldsymbol{P}$ are degree values of the vertices. We can easily set all the diagonal entries to zero, and then $\boldsymbol{P}$ can be regarded as a true adjacency matrix of a weighted projection. This can be mathematically represented as:

$$\boldsymbol{P} = \boldsymbol{A}^T\boldsymbol{A} - diag(\boldsymbol{A}^T\boldsymbol{A}) \tag{3.4}$$

where $diag(\boldsymbol{A}^T\boldsymbol{A})$ denotes the diagonal matrix (matrix with zero entries everywhere except the main diagonal) with same diagonal entries as $\boldsymbol{A}^T\boldsymbol{A}$. Often, however, one wants to work with an unweighted version of the projection, in which case we can convert the adjacency matrix P into the adjacency matrix of the corresponding unweighted network by replacing each nonzero element by 1. A similar analysis would show that the projection on the groups can be represented by matrix $\boldsymbol{A}\boldsymbol{A}^T$

Although projections are easier to work with than the original bipartite networks, we must keep in mind that they throw away lot of potentially useful information. For example, an unweighted projection on vertices does not contain the information about the number of common groups between given two vertices. Weighted projections do contain this information but the information about exactly which groups are common is still lost.

3.3 Converting directed networks to undirected networks

Directed networks are usually more difficult to analyze because the directions of the edges also need to be taken into account. But sometimes, depending upon the problem at hand, the detailed information about the direction of each edge in the network may not be required. In such situations, it makes sense to convert a given directed network to an undirected network keeping only the required information about the directions. Perhaps the simplest strategy is to ignore directions for all the edges and treat them as undirected. However, this implies complete removal of the information stored in the edge directions, and hence this is probably not a good strategy. There exist two better strategies to convert a directed network to an undirected network: **cocitation network** and **bibliographic coupling network**.

3.3.1 Cocitation network

Cocitation of vertices i and j in an unweighted directed network is the number of other vertices which point to *both* i and j. The idea is that if many such vertices exist, then there is something similar about i and j. Note that the cocitation does not depend upon whether there is an edge between i and j or not. Now imagine a network in which we have the same set of vertices as the original directed network, but the edges are undirected and each edge has weight equal to the cocitations of the vertices which it connects. If the cocitation is zero, the corresponding edge simply does not exist. This network is called the cocitation network corresponding to the original directed network. The information about the directions is indirectly present in the network in the form of edges and their weights. Sometimes we can even set all the nonzero weights to 1 so that the cocitation graph would be unweighted. In that case, the interpretation of an edge connecting i and j in this cocitation graph would be that there exists at least one vertex in the original directed graph which points to both i and j.

Consider a vertex k that points to both i and j. Then by definition the entries A_{ik} and A_{jk} of the adjacency matrix would be equal to 1. Hence the product $A_{ik}A_{jk}$ would be 1. If k points to only i or only j or neither, then the product would be zero. Hence the cocitation between i and j can be written as:

$$C_{ij} = \sum_{k=1}^{n} A_{ik}A_{jk} = \sum_{k=1}^{n} A_{ik}A^T_{kj} = (\boldsymbol{A}\boldsymbol{A}^T)_{ij}$$

Here A^T_{ik} denotes the $(i,k)^{\text{th}}$ element of the transpose of matrix $\boldsymbol{A}$. We can't immediately consider the matrix $\boldsymbol{C}$ as the adjacency matrix of the cocitation network because the diagonal entries of the matrix are not zero. In fact we have:

$$C_{ii} = \sum_{k=1}^{n} A_{ik}A^T_{ik} = \sum_{k=1}^{n} A^2_{ik} = \sum_{k=1}^{n} A_{ik} = k^{in}_i$$

Here k^{in}_i is the in-degree of vertex i and we have used the fact that the square of any element of the adjacency matrix is the elment itself (since every element is either 0 or 1). Hence, we can regard the matrix obtained by setting all the main-diagonal elements of matrix $\boldsymbol{C}$ to 0 as the adjacency matrix of the cocitation network.

3.3.2 Bibliographic coupling network

Bibliographic coupling of vertices i and j in an unweighted directed network is the number of vertices *to which* both i and j point to. Just like the cocitation, this number does not depend upon whether i and j themselves are connected or not. Again the logic is that if i and j point to many vertices in common, then there is something similar in i and j. This may help us, for example, to discover similar pages in the World Wide Web for example, without knowing their actual content! Now consider a weighted undirected network in which the set of vertices is same as the original directed network but where the weight of an edge between i and j represents the bibliographic coupling of i and j. This network is called the bibliographic coupling network of the original directed network.

Consider a vertex k to which both i and j point to. Then (and only then) the product $A_{ki}A_{kj}$ would be equal to 1. Hence the bibliographic coupling between i and j can be written as:

$$B_{ij} = \sum_{k=1}^{n} A_{ki}A_{kj} = \sum_{k=1}^{n} A^T_{ik}A_{kj} = (\boldsymbol{A}^T\boldsymbol{A})_{ij}$$

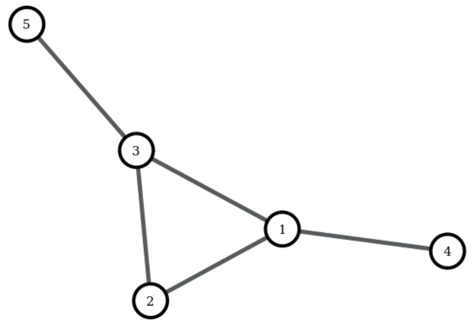
Just like in the case of cocitation network, we must manually set the diagonal entries B_{ii} to zero so that the matrix $\boldsymbol{B}$ can be regarded as an adjacency matrix of the bibliographic coupling network. You may want to verify that the diagonal entries in this case would be the out-degrees of the vertices.

3.3.3 Cocitation vs Bibliographic coupling

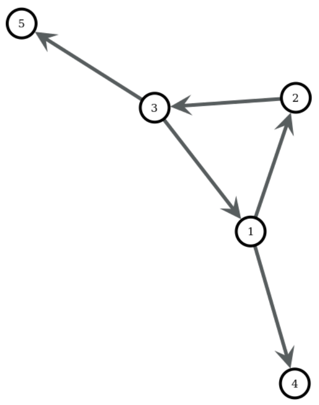
Although both cocitation and bibliographic coupling contain information about the edge directions in the original directed network, in practice one must consider some factors which affect comparative usefulness of these representations. If the directed network that we are interested in is also evolving in time, this becomes particularly important. One such network is the citation network of scientific papers. At any point in time, a published paper usually cites some papers from the past. This "list of references" is fixed and cannot be changed. Thus, if we use bibliographic coupling, once an edge appears in the constructed undirected network, its weight will not change when new published papers are added to the network. However, a little thought would show that this is not the case with cocitation. In fact, for two newly published papers, cocitation is always zero, and only after several months or years, there will appear papers which will cite both of them. Hence, although the bibliographic coupling can be computed as soon as two papers are published, calculation of cocitation involves waiting for years. Moreover, cocitation is useful only when many other vertices cite both the papers and hence is usually useful only for influential papers. For all these reasons, for converting evolving directed networks, bibliographic coupling is preferred over cocitation.

3.4 Walks and Paths

A **walk** is a sequence of vertices such that every two consecutive vertices in the sequence are connected by an edge. It is defined for both undirected and directed networks. In directed networks, while considering consecutive vertices, we must also take into account the directions of edges: vertices i and j are consecutive if there is an edge *from* i *to* j.



(a) Undirected graph



(b) Directed graph

Figure 3.4

In Fig. 3.4a, we have an undirected graph on five vertices. Some walks on this graph are listed below:

$$2 \to 1 \to 3 \to 1 \to 4$$
$$5 \to 3$$
$$3 \to 2 \to 1 \to 3 \to 2$$

Notice how repetition is allowed; a walk may visit a given vertex multiple times. The **length of a walk** is defined as the number of edges in the walk. Thus the walks listed above have lengths 5, 1, and 5 respectively.

Similarly, in Fig. 3.4b, we have a directed graph on five vertices. Below are listed some walks on this directed graph:

$$1 \to 2 \to 3 \to 5$$
$$2 \to 3 \to 1 \to 4$$
$$2 \to 3 \to 1 \to 2 \to 3 \to 5$$

Here notice how edge directions are important while considering walks. Of course the lengths of the walks listed above are 3, 3, and 5 respectively. A **cycle** or a **loop** is a walk that starts and ends at the same vertex.

An important type of walk is a **path**. It is defined as a *self-avoiding* path. This means that a path is a walk that is not allowed to visit any vertex more than once, or it is not allowed to traverse any edge more than once. An example of a path in Fig. 3.4a is $2 \to 1 \to 3 \to 5$. Can you list all the paths in Fig. 3.4b.

It is possible to compute the total number of walks of a given length in a graph. To do this, observe that any walk of length two from vertex j to vertex i must contain an intermediate vertex k so that from j we can first go to k in one step and then from k we can go to i in the second step. For this to happen, both A_{kj} and A_{ik} must be 1, or their product must be 1. Note that if either A_{kj} or A_{ik} is zero, the product is zero. Hence the total number of walks of length two from j to i could be computed by summing this product over all intermediate vertices kk:

$$N_{ij}^{(2)} = \sum_{k=1}^{n} A_{ik}A_{kj} = (\boldsymbol{A}^2)_{ij}$$

Here $N_{ij}^{(2)}$ represents the number of walks from vertex j to vertex i, and $\boldsymbol{A}^2$ is the square of the adjacency matrix. You may also want to verify that this is true independent of whether the network is undirected or directed. It is not difficult to see that this argument can be easily extended to walks of higher length: to consider walks of length r from j to i, we must have $r-1$ intermediate vertices $k_1, k_2, \cdots, k_{r-1}$ (not necessarily distinct), and hence $N_{ij}^{(r)}$ is given by:

$$N_{ij}^{(r)} = \sum_{k_1=1}^{n} \sum_{k_2=1}^{n} \cdots \sum_{k_{r-1}=1}^{n} A_{ik_1}A_{k_1k_2} \cdots A_{k_{r-2}k_{r-1}}A_{k_{r-1}j} = (\boldsymbol{A}^r)_{ij} \tag{3.5}$$

Hence the total number of walks of length r is:

$$N^{(r)} = \sum_{i,j} (\boldsymbol{A}^r)_{ij} \tag{3.6}$$

Since a cycle is walk that starts and end at the same vertex, to get the total number of cycles of length r, we only need to sum the diagonal elements of $\boldsymbol{A}^r$ i.e. the total number of cycles of length r is same as the trace of this matrix. This quantity can also be expressed in terms of the eigenvalues of the adjacency matrix. If $\lambda_1, \lambda_2, \cdots, \lambda_n$ represent the eigenvalues of $\boldsymbol{A}$, then:

$$N^{(r)} = \sum_{i=1}^{n} \lambda_i^r \tag{3.7}$$

This expression is true for both undirected and directed networks.

3.4.1 Geodesic paths (Shortest paths)

A path between the vertices i and j that has the smallest value of the length among all the paths between these two vertices is called a **geodesic path** or a **shortest path** between them. The length of a shortest path between i and j is called the **distance** between these vertices. Of course several shortest paths can exist between a given two vertices. The maximum value of the distance in a network is called its **diameter**. In other words, the diameter of a network is the maximum of the distance taken over all the vertex pairs in the network.

Shortest paths are important for several reasons. The *small-world* effect that we have discussed before is defined in terms of geodesic paths. Shortest paths also determine how fast things like information, rumours, a computer virus, or a disease can spread on a network. These are relevant for the World Wide Web, the Internet, transportation networks, social networks, and so on.

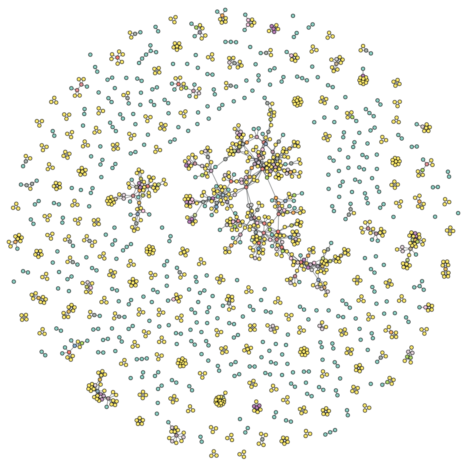


Figure 3.5: An undirected network with many components

3.4.2 Eulerian path

An Eulerian path is a path in a network which traverses every edge in the network exactly once. This means that an Eulerian path neither leaves out any edge nor traverses any edge more than once. It is possible for an Eulerian path to visit a given vertex multiple times, the restriction is only about the edges. In fact, if a vertex has degree greater than two, then that vertex must be visited multiple times so as to traverse all the edges connected to it. A necessary (but not sufficient) condition for an Eulerian path to exist is that each vertex in the network must have an even degree value, or should contain exactly two vertices with odd degrees. Also, it is perfectly possible for many Eulerian paths to exist in a given network. Eulerian path is related to the famous Konigsberg bridge problem that we have discussed before.

3.4.3 Hamiltonian path

A Hamiltonian path is a path in a network which visits every vertex exactly once. Thus, a Hamiltonian path neither leaves out any vertex, nor visits any vertex multiple times. Similar to the case of Eulerian path, a network may contain several Hamiltonian paths. Hamiltonian paths are important in many important real-world applications.

3.5 Components

A network can be in the form of several "disconnected parts". These are called components of the network. As an example, the network shown in Fig. 3.5 can be seen to have many components of various sizes including one much larger than the others. Components need to be defined differently for undirected and directed networks. Let us discuss both these separately.

3.5.1 Components in undirected networks

Technically, a component in an undirected network is defined as the subset of vertices the network that satisfies the following two properties:

1. There is a path from any vertex in the network to any other vertex
2. The property above cannot be preserved by adding any other vertex to the subset

The first of these properties is probably obvious. To understand the second property, consider a network of size n in which every vertex is directly connected to every other vertex by an edge (such graph is also known as a *complete graph*). Now take any subset of $n-1$ vertices in this graph. Obviously the first property is satisfied for this subset. However, if we enlarge this set by including the remaining vertex, then the property will hold true even for this enlarged set. Hence the subset with only $n-1$ vertices in this graph is not a component. It is also trivial to see that in this graph the subset of $n-1$ vertices is not disconnected by the remaining vertex. (Such sets of vertices which satisfy some property but cannot be enlarged further by preserving the property are called **maximal** sets).

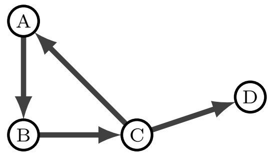
The size of a component is the number of vertices it contains (not the number of edges). An isolated vertex in a network is considered as a component of size *one*.

A network is called **connected** if all its vertices belong to the same component i.e. if there is only one component in the network. If there are more than one components, the vertices in the network can always be labelled so that the adjacency matrix of the network is block diagonal. To do this, one just needs to use consecutive labels for all the vertices in each component.

3.5.2 Components in directed networks

The notion of components in directed networks is more involved than in the undirected case since one also needs to take into account the edge directions. A trivial way to define components in a directed network is to simply ignore the edge directions for all the edges and treat the network as undirected. The components we get this way are just the components in the corresponding undirected graph, and are known as **Weakly Connected Components**. Arguably, these are not very interesting or useful.

A better way to think about components in a directed networks is the following. We say that vertex i and vertex j are **strongly connected**, if there exists a path from i to j, as well as a path from j to i. Then a **Strongly Connected Component** is a set of vertices in the network that are strongly connected to each other. Of course just like the undirected case, we impose a restriction that the set is *maximal.* Let us note a few important things about strongly connected components (or SSCs for brevity). First, each vertex belongs to exactly one SSC (Can you see why?). Second, unlike the undirected networks, it is possible for a vertex to be connected to other vertices still be part of a SCC of size one. To understand this, consider the following graph:

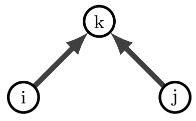


In this graph, it is easy to see that the vertices A, B, and C belong to the same SCC. However, since the vertex D does not point to any other vertex, it does not belong to the SCC of the other three vertices. Hence, it belongs to a SCC which contain no other vertices. Thus, it is possible for two SCCs to have an edge between them. This is unlike the case of directed networks where adding a link between two components immediately merges them.

Apart from weakly and strongly connected components we can also define out-components and in-components. An **out-component** of a vertex i is the set of all vertices which can be reached from i. In other words, if you imagine that you are standing on vertex i, and you are allowed to traverse directed edges only in the direction in which they point, the out-component is the set of vertices which you can visit this way. Thus, unlike other types of components that we have discussed, an out-component is a property of the structure of the network as well as the starting vertex.

Now consider a vertex j which is in the SCC of i. By definition of SCC, j is reachable from i, and hence also belongs to the out-component of i. Hence all the vertices in the strongly connected component of i are automatically in the out-component of j. Of course the reverse need not be true since there can exist vertices which can be reached from i but it may not be possible to come by to i from them. Taking the same argument a little further, we see that all the vertices that can be reached from i should also be reachable from vertex j if it belongs to the SCC of i. This is true because from j you can reach i because they are part of the same SCC, and then from there you can reach to any vertex in the out-component of i. This shows that the out-components of all the vertices in a given SCC are in fact identical! Hence, it would be more appropriate to say that out-components actually belong to strongly connected components instead of saying that they belong to individual vertices. Each SCC in a directed network has its own out-component. It also means that a SCC is a subset of its out-component.

We must note two important things here. First, it is perfectly possible for a vertex to be part of more than one out-component. In the graph shown below, vertex k is reachable from both i and j and hence belongs to out-components of both.



Because of this, although SCCs in a directed network are always disjoint, their out-components overlap in general. Second, a SCC and its out-component can be exactly same. This would happen when there are no vertices from which it is not possible to come back inside the SCC. A special case of this a vertex with only one in-link and no out-links. This vertex is a part of SCC of size one and its out-component is composed of only itself.

Once we understand the notion of out-component properly, it is easy to define an in-component and work out its properties. An **in-component** of vertex i is the set of all vertices which i can be reached. Similar to the out-component, it is easy to see that the in-components of all the vertices in the SCC of i are identical, and hence an in-component actually belongs to the whole SCC. Also, in-components of different SCCs can overlap. Finally, a SCC and its in-component may be identical but in general the SCC is a proper subset of its in-component.

Thus, we now see that a directed network has a rich component structure. A directed network can be decomposed into disjoint strongly connected components. Each of these SCCs has an associated in-component and an associated out-component and each SCC is actually an intersection of its in and out components. The in and out components of different SCCs can overlap with each other. It is even possible (and in fact is common) that a particular SCC is a part of the out-component or the in-component of some other SCC.

3.6 Trees

A **tree** is a connected undirected network which does not contain any *self-avoiding loops.* A self-avoiding loop is a loop such that no edge in the loop is traversed more than once. There is exactly one path between given two vertices in a tree. To see this, assume that vertex i and j are connected by more than one path. Then it would be possible to go from i to j along one path, and then come back to i via another path. This means that the network contains a self-avoiding loop and then it can't be a tree. An undirected network containing more than one component is called a **forest** if all its components are trees. For example, the method called 'Random Forest' in the field of machine learning uses a collection of 'decision trees'.

A tree on n vertices contains exactly $n-1$ edges. To see why this is true, imagine constructing a tree by adding one vertex at a time to the network. With exactly one vertex (call it vertex 1), we need zero edges to keep the network connected. When another vertex (vertex 2) is added to the network, we need to connect it to vertex 1 to keep the network connected. Hence with two vertices, the tree has one edge. Observe that we can't add another edge between these two vertices because that would create a self-avoiding loop in the network. Now we add vertex 3 to the network. We must connect it to either vertex 1 or vertex 2 to keep the network connected, and hence with three vertices we would have two edges. However, we cannot connect vertex 3 to both 1 and 2 since that creates a loop in the network. Continuing this way, with n vertices, we would have exactly $n-1$ edges in the tree.

The converse is also true. A connected network containing exactly $n-1$ edges is necessarily a tree. To prove this, suppose that is not the case, and hence the network contains a self-avoiding loop somewhere in it. Such loop of size n also contains exactly n edges and we can remove one of those to break the loop. Note that this doesn't disconnect the network because all the vertices in the loop are connected by two paths, and

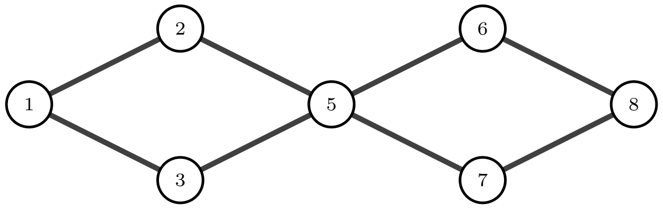


Figure 3.6

after removing an edge, one path is still available. But that means that now we have a connected network with $n-2$ edges! As we have seen above, this is impossible; we need at least $n-1$ edges to keep the network connected. Hence, the network must have been a tree to start with.

Trees are of fundamental importance in the field of networks as well as other areas such as computer science and machine learning. They form one of the most important data structures in computer science, and many methods in machine learning rely on the notion of trees. In the theoretical study of complex networks, trees arise in the study of random graph models, they also are starting point of many network calculations like network spectra, first passage time, and so on. Some real-world networks take the form of trees, at least approximately. For example, the actual tree (branches as edges, their intersection as nodes), the river network, etc.

3.7 Independent paths

A particularly important type of paths in networks is 'independent paths'. In a nutshell, an independent path is a path that does not intersect itself. Two paths are called **'edge independent'** if they don't share any edge (although they can share a vertex). Similarly, two paths are said to be **'vertex independent'** if they don't share any vertex except for the starting and ending vertices. A vertex independent path is always edge independent but the reverse is not true. In the network shown below, the paths $1-2-5-6-8$ and $1-3-5-7-8$ are not vertex independent because they share the vertex 5. But they are edge independent because they don't share any edges.

It should be noted that for given two vertices i and j, the set of independent paths can in general be chosen in several different ways. In the network shown above, we could have chosen the set $\{(1-2-5-7-8),(1-3-5-6-8)\}$ as the set of two independent paths between 1 and 8 instead of the set $\{(1-2-5-6-8),(1-3-5-7-8)\}$. The size of the set containing largest number of independent paths between i and j is called their **connectivity**. The connectivity tells us how strongly given two vertices are connected.

3.7.1 Cut sets and Menger's theorem

The connectivity of two vertices i and j can be related to *bottlenecks* between them. If there are many independent paths between two vertices, then failure of some of them doesn't disconnect them. On the other hand, if there are only a few, then depending upon whether we are talking about edge independent paths or vertex independent paths, some edges or vertices along these paths act as bottlenecks. For example, in the graph shown above, vertex 5 is such bottleneck for paths between 1 and 8: removing 5 will completely disconnect them. This leads us to the idea of cut sets.

A **vertex cut set** for i and j is a set of vertices whose removal from the network completely disconnects them. For a graph shown in Fig. 3.6, some vertex cut sets for vertices 1 and 8 are $\{2,3\},\{5\},\{2,5\},\{3,5,7\}$. An **edge cut set** for i and j is similarly defined as the set of edges whose removal would disconnect them. Again for the same graph and for the vertices 1 and 8, some edge cut sets are:

$$\{(2,5),(3,5)\},\{(1,3),(2,5)\},\{(3,5),(5,6),(7,8)\}$$

.

A **minimum cut set** is defined as a cut set with smallest size for a given pair of vertices. In the example above, the minimum vertex cut set between 1 and 8 is $\{5\}$ while the sets $\{(1,2),(1,3)\}$ and $\{(1,3),(2,5)\}$ are minimum edge cut sets (Can you find all minimum edge cut sets for 1 and 8?).

Menger's theorem relates the size of a minimum cut set and the number of independent paths. It states that:

> *"The number of independent paths between vertices i and j is equal to the size of the minimum cut set between them."*

The theorem applies both to vertex independent paths as well as edge independent paths. At first sight, the theorem may look like a trivial statement; isn't it obvious that two disconnect i and j, shouldn't we remove one edge from each edge independent path between them? To understand this, observe that we must remove *correct* edges (or vertices in case of vertex independent paths). For example, the number of edge independent paths between 1 and 8 is two but we can't remove two arbitrary edges to disconnect them. Removing edges $(1,2)$ and $(5,6)$ will not disconnect them but removing $(2,5)$ and $(3,5)$ will. In fact the proof of the Menger's theorem for a general network is sufficiently complicated that we won't attempt to prove it.

3.7.2 Max-flow/min-cut theorem

Suppose something, say water, is flowing on a network along the edges. Define **flow** along an edge in the network as the amount that flows between them per unit time. If we assume that the edges have a certain value of the maximum capacity so that they cannot carry more than certain amount of fluid per unit time, let us call that as the maximum flow along an edge. In this case, what is the maximum flow from some vertex i to some other vertex j (not necessarily directly connected to i by an edge)? The **max-flow/min-cut** theorem tells that the maximum flow between vertices i and j, in units of the maximum flow along a single edge, is equal to the number of edge independent paths between them. That is, assuming that every edge in the network has the same value for the maximum flow, and assuming that that value is 1, the maximum flow between any two vertices in the network is same as the number of edge independent paths between them.

Thus, now if we combine the *Menger's theorem* and the *max-flow/min-cut theorem*, we see that for any two vertices in a network, the following three quantities are equal:

- Edge connectivity (i.e. the number of edge independent paths)
- Size of the minimum edge cut set
- Maximum flow

Max-flow/min-cut in weighted networks

Now let us assume that each edge in a network has a different capacity so that the maximum flow is different for different edges. This is relevant to many real-world networks: capacity of roads in a transport network is different from each other, and so is the capacity of the links in the Internet. In such situation, we would like to know the value of the maximum flow between given two vertices. This would in effect decide how many vehicles can go from point A to point B per unit time in a transport network, and how many data packets can go from one router to another per unit time which in turn decides the speed of our internet connection!

First let us try to understand minimum edge cut set when the network where edges are weighted because maximum flow along an edge can be thought of as its edge weight. Similar to unweighted case, an edge cut set for vertices i and j in a weighted network is again a set of edges whose removal disconnects them. However, the size of an edge cut set is now the sum of the weights of the edges in the cut set, not the number of edges in it. With this definition of the size, the definition of the minimum edge cut set is same as that in the unweighted case (as a cut set with minimum size). This means that a minimum cut set may contain more edges than another cut set, but the weights of the edges in it is always smallest. The max-flow/min-cut theorem in weighted networks then states that the maximum flow between two vertices is equal to the size of the minimum edge cut set between them. Everything else that we have said above for unweighted case is now equally applicable to weighted networks also.

This has an interesting consequence for real-world networks like transport networks. First consider a vertex pair with only one path between them. Because the maximum flow between two vertices is entirely decided by the size of the minimum cut set, the maximum flow in this case would be entirely decided by the edge with the minimum weight along the path. Thus, increasing the capacity of other edges (say by widening the roads in transportation network) along the path has absolutely no effect on the flow between these two vertices! Such insights are very useful while actually modifying the real-world networks.

Finally, notice that the whole discussion related to independent paths in this section nowhere assumed that a network is undirected. Hence all the developments and theorems are equally applicable to directed networks also.