1 Representing and describing networks

Because networks are a *representation* of the structure of a complex system, all of network analysis and modeling relies on first specifying what we can represent, and how we represent it mathematically or computationally. In this lecture, we will introduce the three main ways to represent a network, and then explore the standard ways we can use that representation to describe (summarize) a network's structure.

1.1 Representing networks

There are three main ways to represent a network: an adjacency matrix, an adjacency list, or an edge list. As a running example for each representation, consider the following network:

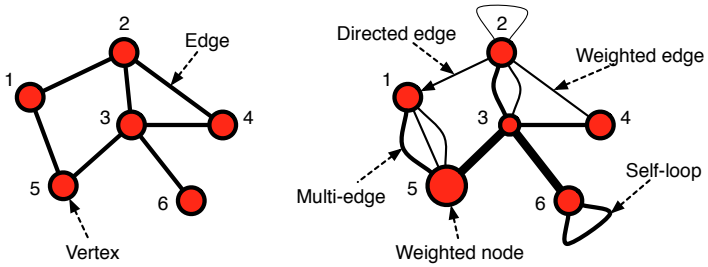

Figure 1: A simple network (left) and a network with more exotic graph properties (right).

1.1.1 The adjacency matrix

An **adjacency matrix** is an $n \times n$ matrix, typically denoted A, defined as

$$A_{ij} = \begin{cases} w_{ij} & \text{if nodes } i \text{ and } j \text{ are connected} \\ 0 & \text{otherwise,} \end{cases}$$

where $n = |V|$ is the number of nodes, and $w_{ij} \in \mathbb{R}$ is the *weight* of the connection between nodes i and j.[1] If A represents an *unweighted network*, then edges have "unit" weight, defined as $w_{ij} = 1$.

In Figure 1, an edge's thickness is proportional to its weight w_{ij}. The corresponding adjacency

[1] In some systems, a connection (i, j) may exist but have $w_{ij} = 0$. Such situations require special handling.

matrices are

$$A_{\text{simple}} = \begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix} \qquad A_{\text{exotic}} = \begin{pmatrix} 0 & 0 & 0 & 0 & \{1,1,2\} & 0 \\ 1 & \frac{1}{2} & \{2,1\} & 1 & 0 & 0 \\ 0 & \{1,2\} & 0 & 2 & 3 & 3 \\ 0 & 1 & 2 & 0 & 0 & 0 \\ \{1,2,1\} & 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 & 2 \end{pmatrix}.$$

Notice that for the simple graph, (i) the diagonal is all zeros (no self-loops), (ii) all entries are "binary" (0 or 1, and hence unweighted), and (iii) if $A_{ij} = 1$ then also $A_{ji} = 1$ (undirected, implying that A is symmetric across the diagonal). For the non-simple graph, none of these properties holds: it is a directed, weighted, multigraph, with self-loops.

Adjacency matrices are most commonly used in mathematical expressions, e.g., to describe mathematically what a network algorithm or network calculation does.[2]

Sometimes, adjacency matrices are used in network algorithms, because an operation on the matrix A is more efficient (takes less time) than the same operation on another representation, e.g., determining whether some edge exists, $(i, j) \in E$, is fastest in A because it requires only a constant-time $O(1)$ lookup. The tradeoff of fast access to each adjacency A_{ij}, however, is that A takes a quadratic amount $\Theta(n^2)$ of memory to store, because it explicitly stores a value for every pair of nodes $i, j \in V$. As a result, on a modern computer, a simple network with only $n = 100{,}000$ nodes will take about 8 bytes $\times$ $(10^5)^2/2^{30} = 74.5$ GB of space to store.

When we say an empirical network is **dense**, we mean that most of A's elements are non-zero. In this case, an adjacency matrix is the most space-efficient way to represent the edges. When we define edges using pairwise correlations or similarity scores, we get dense matrices because every pair produces a score. When we say an empirical network is **sparse**, we mean that roughly $c \cdot n$, for a "small" integer c, of A's entries are non-zero. Crucially, most empirical networks are sparse in this sense, which makes an adjacency matrix a very inefficient representation for so small number of non-zero elements.[3]

[2] For this reason, many techniques from linear algebra have applications in network analysis and modeling. Here, we will not assume the reader has familiarity with matrix multiplication or spectral techniques, but will try to highlight such connections for those who do.

[3] Mathematical definitions of "sparse" and "dense" networks are asymptotic, e.g., a dense network has $\Omega(n^2)$ connections, which requires assuming a model by which to "grow" the matrix in the limit of large n. However, real-world networks are always finite objects, and we usually do not know the underlying data generating process. Hence, "sparse" and "dense" for empirical networks are only loose descriptive terms, and refer mainly to whether the average degree is closer to 1 or n.

1.1.2 The adjacency list

An **adjacency list** stores only the non-zero elements of the adjacency matrix, using an array of (unordered) lists, one list for each of the n vertices, and the ith list stores the names of the neighbors of node i. Optionally, edge annotations w_{ij} can be stored together with the neighbor name j in a tuple, e.g., (j, w_{ij}). Hence, an adjacency list is similar to a hash table of adjacencies. For the two networks in Figure 1, their adjacency list representations are

$A_{\text{simple}} =$

[1]	→	(2, 5)
[2]	→	(3, 1, 4)
[3]	→	(2, 5, 4, 6)
[4]	→	(2, 3)
[5]	→	(1, 3)
[6]	→	(3)

$A_{\text{exotic}} =$

[1]	→	((5,1),(5,1),(5,2))
[2]	→	((1,1),(2,$\frac{1}{2}$),(3,2),(3,1),(4,1))
[3]	→	((2,1),(2,2),(4,2),(5,3),(5,3))
[4]	→	((2,1),(3,2))
[5]	→	((1,1),(1,2),(1,1),(3,3))
[6]	→	((3,3),(6,2))

In the simple case, because the network is undirected, each of the $m = 7$ edges appears twice in the adjacency list, once as j in i's list, and once as i in j's list. The adjacencies for a given vertex are typically stored as a linked list, or in a more efficient data structure, like a self-balancing binary tree (e.g., a red-black or a splay tree). This form of representation is what is meant by a "sparse matrix" data structure, and is used in most mathematical programming languages.

The popular GML file format[4] for storing networks is an adjacency list representation, written out in text. By only storing the non-zero entries of the adjacency matrix, an adjacency list takes space proportional to the number of nodes and edges $O(n + m)$: $O(n)$ space for the array of n nodes and $O(m)$ space to store the m edges. In this representation, checking whether an edge exists, $(i, j) \in E$, is slower than in an adjacency matrix, taking $O(m/n) = O(\langle k \rangle)$ time, where $\langle k \rangle$ is the average number of edges in an adjacency's list.

1.1.3 The edge list

An *edge list* stores only the edges themselves: (i, j) for unweighted and (i, j, w_{ij}) for weighted edges, without directly listing the nodes. Node indices, the presence of nodes with no edges attached, and whether the network is directed, weighted, bipartite, etc. must all be inferred from the edge list's contents. For the two networks in Figure 1, their edge lists are

$$\begin{aligned} A_{\text{simple}} &= \{(1,2),(1,5),(2,3),(2,4),(3,5),(3,6)\} \\ A_{\text{exotic}} &= \{(1,5,1),(1,5,1),(1,5,2),(2,1,1),(2,3,2),(2,2,1/2),(2,3,1), \\ &\quad (2,4,1),(3,2,1),(3,2,2),(3,4,2),(3,5,3),(3,5,3),(4,2,1), \\ &\quad (4,3,2),(5,1,1),(5,1,2),(5,1,1),(5,3,3),(6,3,3),(6,6,2)\} \ . \end{aligned}$$

[4] See https://en.wikipedia.org/wiki/Graph_Modelling_Language.

In a simple network, an edge may appear only as (i, j), implying both that it is unweighted and that its reciprocal edge (j, i) also exists. In a non-simple network, edges can be directed, and hence seeing (i, j) should not imply that (j, i) also exists. This presents an ambiguity that cannot be resolved without external knowledge about whether the edge list represents a directed or undirected network. Furthermore, because only edges are listed, nodes without edges ("singletons") are not listed, which can cause additional ambiguities.

Edge lists may be a convenient and compact way to store a network, but the ambiguity of how to interpret their contents in the absence of additional information make them problematic. In contrast, formats like GML take nearly as little space, but avoid ambiguities.

2 Summarizing networks using statistics

Regardless of which representation we use, networks are sufficiently complicated objects that we almost always need to use statistical methods in order to gain an intuitive sense of their shape and variations, or to describe their patterns, or test whether these patterns are meaningful. In this section, we will survey some of the most basic of these descriptive statistics. These statistics fall largely into three categories:

1. **connectivity** measures are directly or indirectly related to the number of connections a node has, and to the overall degree structure of the network,
2. **motif** measures count the frequency of specific subgraphs in a network,
3. **positional** measures are related to where in the network a node sits, and to the distances between nodes in the network.

Many of these network measures come in two flavors:

- **node-level** or "local" measures, which we calculate for an individual node i, and
- **network-level** or "global" measures, which we calculate across the entire network.

Sometimes, we can obtain the corresponding network-level statistic of a particular quantity by simply computing the average node-level quantity, but not always.

Because network representations can vary, so too do the algorithms for calculating specific network statistics. As a result, before beginning any network analysis, we need to first identify the graph properties of representation being used: Is the network simple? Are edges directed? Are edges weighted? Do nodes have metadata? Does the graph have one or several components? Etc.