Chapter 13

Visualizing networks

Visualization is a powerful tool. As shown by Anscombe's quartet and similar exercises [299], visualization can reveal a wide range of patterns not easily inferred from statistics alone. Therefore, when working with network data, it is almost always a good idea to visualize the network and just take a look. For instance, suppose the network consists of 10 disconnected components. Drawing the network using a simple visualization can quickly reveal the fact that the network consists of these disconnected components. Of course, this can be learned by computing the network's connected clusters. However, if you don't visualize it, or think to perform this check, it may take a while until you realize that the network is disconnected and there are 10 components. You may even perform a variety of analyses with the wrong assumption that the network is connected. It is plausible that one may miss this important fact until all analysis is complete, if the network has not been visualized.

Although network visualization can be done in many ways, the vast majority of network visualizations take the form of 2D "ball-and-stick" (or "node-link") diagrams, where each node is represented as a symbol (e.g., a circle, a ball, etc.) and they are connected by lines that represent edges. It is intuitive and familiar (you've seen many in this book already) and thus often ideal—especially for small networks—at visualizing a network. Thus, our primary focus will be on the node-link diagram, although we will briefly talk about alternatives at the end.

In network visualization, size matters a lot. When the size of your network is small, many methods will simply work well and there are not many factors that should be considered. However, when the network is large (say more than 10,000 nodes), it becomes challenging—both computationally and aesthetically—to create a useful and informative visualizations. It is also not just the size—density also plays a role. High density means more edges (lines) that can obscure the visualization. A network where each node is connected to only 1 or 2 neighbors will be much easier to visualize than a network where each node is connected to 1,000 neighbors. Network visualizations of the latter, often humorously referred to as "*hairballs*" or "*ridiculograms*," are a real sticking point when trying to visualize a network.

https://doi.org/10.1017/9781009212601.016 Published online by Cambridge University Press

Large or dense networks are difficult to visualize, both to compute the visualization and to read or interpret it.

Thus, one can say that the major challenge in network visualization occurs when the network becomes large and dense. That's when we need to apply some methods to simplify the network or take more sophisticated approaches. In addition, or more importantly, we need to clarify what types of information we want to get out of the visualization. Is it about seeing the large-scale structure of the network? Is it about distinguishing different "regions" or "areas" of the network? Perhaps it is not about a single static network but about demonstrating a change between networks or within a network over time (Ch. 14)? Depending on the main objectives, drastically different visualization methods may be used. Do you just want to know the component structure? Maybe then you don't need to worry too much about creating a hairball as long as you clearly see disconnected components. Do you want to see how network communities are organized? Then it may be useful to perform community detection and use the results to simplify the visualization, for instance by considering each community as a "supernode" and considering the connections between those supernodes.

In this chapter, we will delve into these questions and discuss practical ways to visualize networks from the perspectives of network analysis, visual perception, design principles, and so on. We will begin with a discussion of the basics of network visualization, from graph layout algorithms to drawing network elements. We then discuss ways to customize the visualization, tuning it for different purposes such as for exploring the data or communicating discoveries. Researchers at times must confront networks that cannot be easily visualized—some may need significant preparation and at times it may be best to approach visualization from a non-traditional avenue. We discuss advanced visualizations techniques and non-traditional approaches that can help, as can network preprocessing steps undertaken to improve or clarify a visualization. Lastly, we cover some tools available for generating network visualizations.

13.1 Standard network visualization

We have encountered many typical network visualizations already (Figs. 1.3, 1.4, and 10.1, for example). Networks are usually visualized using a "ball-and-stick" approach (a.k.a. "node-link diagram"). Nodes are represented graphically with circles, squares, or other shapes, and edges are drawn as lines or curves between those shapes. To visualize a network then requires: (i) laying out the nodes in a visually informative (and pleasing) manner, and (ii) determining how nodes and edges are drawn, including whether nodes and edges with different properties or attributes are drawn differently from one another.

13.1.1 Layout algorithms

The graph layout problem is that of determining the graphical positions of a network's nodes. Layouts can be determined by hand (check out the "sociograms" in Fig. 1.3) but this rapidly becomes tedious and impractical for all but the smallest networks so

https://doi.org/10.1017/9781009212601.016 Published online by Cambridge University Press

computer algorithms are used instead. Graph layout algorithms have been the subject of research in computer science for decades. Taking a graph, especially a dense graph, and projecting it into a 2D or sometimes 3D plane for visualization is a challenging computational problem (and is not always possible when some constraints must be satisfied; Sec. 13.1.3). Fundamentally, this projection problem is one of determining the spatial coordinates of the nodes, generally with an eye towards placing nodes in a pleasing and informative way. It is of course impossible to review all graph layout algorithms, thus we will discuss a few of the most common ones, most of which share the common idea that connected nodes should be placed close together.

A *layout algorithm* is one that determines how nodes are placed within the visualization. Ideally nodes should be arranged so they don't overlap and that edges cross each other as little as possible.

Some layout algorithms are simple and largely ignore the rationale of placing connected nodes close together. *Circular layouts*, for instance, arrange nodes along the perimeter of a circle of a given radius. The layout algorithm then determines the ordering of the nodes along the circle. Sometimes, nodes are arranged in descending order by a network property such as their degree. Other times, nodes are arranged along the circle to keep them close to their neighbors. Edges are then drawn as lines or arcs connecting the nodes.

Many other layout algorithms take a physics-inspired approach to place connected nodes close together. One simple solution is to consider each edge as a spring (or any medium of force that pull nodes together), so that connected nodes are attracted to each other while avoiding being too close. Imagining this attractive force and using a physical simulation to obtain a layout is the key idea behind the "force-directed" layout algorithm.

Usually, there are other ingredients that improve the algorithm's output. The first is repulsive forces that keep nodes from overlapping. Although this spring force prevents connected nodes from overlapping, it does not prevent unrelated nodes from overlapping. Therefore, the layout algorithm usually also considers the repulsion force between nodes by treating each node as a point charge that repels others. Second, many algorithms implement "gravity" toward the center of the visualization to keep the nodes from floating away. Finally, the algorithm can also be configured to consider the strength (weight) of the edges and allow fine-tuning of the strengths of these forces. Eventually, so the idea goes, if these forces are well tuned—neither too strong nor too weak—the graph will settle down into a pleasing and informative layout.

Although force-directed layout algorithms are probably the most common, there are useful alternatives. In particular, let us talk about approaches that leverage useful vector representations of nodes. The first is to determine node coordinates using spectral information. A matrix representation $\mathbf{M}$ of the network structure is derived, typically the graph Laplacian (Ch. 25) and the elements of its eigenvectors are used as cartesian coordinates to place nodes. In other words, if $\mathbf{u}$ and $\mathbf{v}$ are two eigenvectors of $\mathbf{M}$, then each node i is placed at coordinate (u_i, v_i). Typically, the eigenvectors used correspond to the two (or three, for 3D layouts) largest or smallest eigenvalues, depending on which matrix representation is employed. Spectral graph layouts share many similarities with spectral data clustering [478].

https://doi.org/10.1017/9781009212601.016 Published online by Cambridge University Press

Depending on the matrix used, spectral layouts also possess underlying physical interpretations. For example, the graph Laplacian corresponds to a discrete diffusion operator, and so its eigenvectors form the basis of the graph diffusion equation's solution space. This means that information about the time it takes a particle to randomly diffuse across different parts of the graph is captured by the eigenvectors and some of that "spatial" information can be displayed visually in plots of those eigenvectors. We discuss the spectral theory of networks in greater detail in Ch. 25.

Another more recent alternative is to use (neural) "graph embedding" methods (see Ch. 26). As we will discuss in the later chapters, graph embedding methods aim to find a low-dimensional vector-representation of the graph (usually the nodes). Once we have this representation, a dimensionality reduction method can be applied to obtain a even lower-dimensional (2D or 3D) representation. This approach has become particularly appealing thanks to the development of sophisticated methods for graph embedding and dimensionality reduction (e.g., t-SNE or UMAP). This approach can be faster than the force-directed layout, which is essentially a large physical simulation.

Finally, there are many layout algorithms that are specialized to specific types of networks. For instance, when the network is a directed tree or DAG (directed acyclic graph) or close to a DAG, the "dot" algorithm (part of GraphViz) provides useful "hierarchical" layouts [145].

13.1.2 Drawing the network

Given the spatial coordinates of nodes, found from a layout algorithm, drawing the network becomes straightforward. Suppose node i is assigned 2D position (x_i, y_i) (or similarly in 3D) in the plotting coordinate system of your visualization. Then, i can be drawn using a graphical command, such as $\texttt{CIRCLE}(x_i, y_i, r_i, c_i)$, where r_i is a radius for the node and c_i is its color. Likewise, edge (i, j) between nodes i and j can be drawn with a basic line command: $\texttt{LINE}(x_i, y_i; x_j, y_j; w_{ij})$, where w_{ij} is the width of the line representing the edge.[1,2]

From this, we realize two things. First, the fundamental challenge is the graph layout: any basic plotting library can be used to draw a network once a layout is determined. Call the library's scatter plot functionality to lay down the nodes and draw edges with line plots. Of course, specialized tools (Sec. 13.6) can make this more convenient, and provide additional features such as interactive layouts, but fundamentally the challenging step is not the drawing but computing the graph layout. Second, we see that the graphical attributes used to draw nodes and edges do not have to be the same for each node. For instance, we can choose radius r_i to visualize a feature such as i's importance, and then this radius can vary for different nodes. Using graphical attributes to reinforce network properties in your visualization is very powerful, as we discuss below.

[1] Typically the edges are either drawn before the nodes or drawn at a lower "z" order so that the edges do not occlude the nodes that they connect.

[2] Edges can also be visualized with arcs or splines; see below.

https://doi.org/10.1017/9781009212601.016 Published online by Cambridge University Press

13.1.3 When good network visualizations go bad

Because we must project the network down to 2D (or 3D, on occasion), there can be a significant loss of information. Fundamentally, only planar graphs—those that can be drawn in a 2D plane without any edges crossing one another—can be guaranteed to visualize without information loss in two dimensions. For other cases, if the network is too large, too dense, or both, it becomes almost impossible to create informative visualization in two dimensions.[3]

In our experience, network visualizations tend to lose utility when the number of nodes or the number of edges per node becomes too large. If the network is very sparse, the visualization will not be too dense, but if there are tens of millions of nodes, it still be hard to read despite being sparse. A scatter plot (visualizing a network without any edges is essentially drawing a scatter plot) suffers in much the same way: too many points and you cannot distinguish them. Likewise, if the number of nodes is reasonable (say, under a few thousand at most) but the number of edges is very high, the final visualization will be dense and hard to parse. That said, more advanced visualizations, including techniques we discuss below, can overcome some of these limitations. Your own judgment is an important final arbiter: are you able to convey the information you need to convey with the visualization or not?

What to do with a hairball? When confronted with a network that defies easy visualization, what options are available to you? Generally you need to either transform or subset the network in some way or use a non-network or non-traditional network visualization. Properly transforming or subsetting can reduce the complexity of the network such that a readable visualization may now be feasible. As mentioned in Ch. 10, removing dangling nodes, k-core filtering, or applying network backbone algorithms may reduce the network enough to make a useful visualization feasible. Another way is *coarse-graining* your visualization by applying community detection (Ch. 12). Consider communities as super nodes and visualize only the community-level network. Another option is to focus on the layout of the nodes and not draw edges, or treat edges in ways (e.g., making them more transparent) so that they don't obscure the overall visualization. An appropriately chosen visualization alternative can still achieve your specific goals. We describe some non-traditional visualization techniques in Sec. 13.3 and discuss manipulating network data for visualizations in Sec. 13.4.

Fixes for networks that are hard to visualize include thinning the network, aggregating nodes into super-nodes, or choosing a non-traditional visualization.

[3] Occasionally, the network is so simple, like a lattice or a collection of almost entirely disconnected nodes, that a visualization will be hardly useful. It will be unlikely to earn back the space it consumes and time put into making it. However, even then, a quick visualization to explore the network may help you understand it better.

https://doi.org/10.1017/9781009212601.016 Published online by Cambridge University Press

13.2 Customizing your visualization

Visualization is an iterative process. The first draft visualization, direct from a graph layout algorithm or application, may be not as informative as it can be, and can be a bit boring. It's useful for quick check-ins during exploratory work (Ch. 11) such as examining how well connected the network is as different filtering thresholds are employed. But it doesn't convey much information, at least, not as much as it could.

What are some useful actions you can take to improve both the information density and readability of a standard network visualization? You can incorporate various node and edge attributes (Ch. 9) as graphical elements. You can add diagramming elements such as callouts and other annotations on top of the network. While graph layout algorithms often produce great visualizations, sometimes a small bit of hand tweaking to node placement can greatly improve the layout (although this should never been done to mislead the viewer). You can also extend the "ball-and-stick" visuals by drawing edges as curves or splines instead of straight lines.

By iteratively improving a basic visualization along these lines, a network visualization can become rich, informative, and memorable (Fig. 13.1).

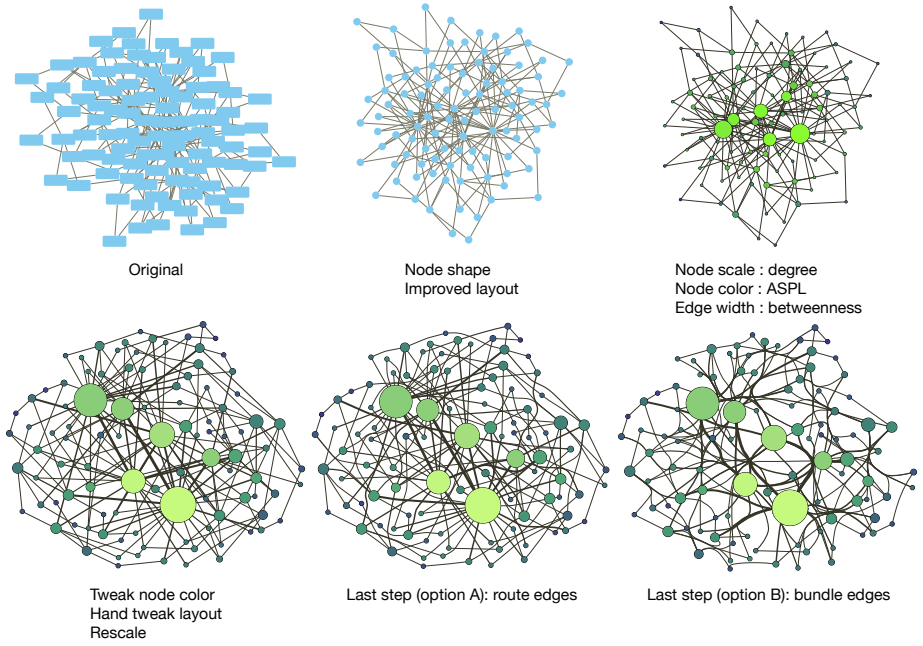

Figure 13.1 Iteratively refining a basic network visualization. An informative and memorable network visualization can be made by taking a basic network visualization, adding on the node and edge attributes as graphical elements, and, sometimes, improving the layout with small manual adjustments.

https://doi.org/10.1017/9781009212601.016 Published online by Cambridge University Press

13.2.1 Reinforcing your information hierarchy

One of the simplest and yet most powerful ways to use a network visualization is as a means to convey graphically the information about the network elements. This information takes the form of node and edge attributes (Ch. 9). The graphical attributes of the nodes and edges in the visualization can be mapped onto some or all of the node and edges attributes, providing a means to convey that information directly to the viewer.

Use the nodes and edges in your visualization as graphical elements to "hang" additional information from. Node size, node color, edge thickness, and more are all available to represent further network properties or attributes.

Node attributes Generally the nodes in the network are drawn as shapes. Visually, besides its location within the visualization, these shapes are defined by the type of shape (square, circle, pentagon, and so forth), the color of the shape, the shape's size, a line defining the boundary of the same (and that line's color, thickness, and so on). These graphical attributes can then be related to the node attributes in various ways. For instance, a node may fall into a particular type or group (Ch. 12). Different shapes can be used to represent different groups. Color may also be used for node attributes. In a social network of politicians, for example, the political affiliations of the nodes could be represented by color. Color can also be used to convey an ordinal or numeric quantity by using a color scale. For instance, political ideology could range in strength from right-wing through centrist to left-wing, and a color gradient fading from, say, red to blue could be applied to the nodes' fill colors to show this information.

Edge attributes In concert with node attributes, edges, usually drawn as lines or curves (Sec. 13.2.3), also provide opportunities for visualizing edge attributes. Lines are defined by their thickness and color, both of which can be associated with numeric quantities (or categories) by defining a suitable mapping. For the widths of edges, a basic function $w = f(x)$ can map the numeric edge attribute x to the corresponding line width w. A linear map, $w = ax + b$, uses constants a and b to tune the minimum and maximum thickness of lines. Nonlinear maps can also be used, such as $w = ax^b + c$ or $w = a\log(x) + b$. These maps may overly exaggerate the distribution of x, however, so care should be taken to ensure the data are represented accurately.

Color can also be used to map edge attributes. A "colormap" should be defined, for instance a gradient fading from blue to red as a numeric x goes from small to large. A categorical x can also be illustrating with a mapping of unique colors to distinct categories. Categorical colors can represent the types of edges, or the types of nodes (e.g., by using the colors of the connected nodes). Another aspect is the transparency of the color. By adjusting transparency, one can potentially mitigate edges obscuring the nodes, especially in dense graphs. The transparency can be also tied to edge attributes. But care should be taken when relying on color. When edge lines are too thin, color may be hard to perceive. Some color combinations are not accessible to individuals with color blindness, and may be lost when media do not support color, such as the

https://doi.org/10.1017/9781009212601.016 Published online by Cambridge University Press

Table 13.1 Common node and edge graphical attributes and example mappings with network attributes.

Visual attribute	Variable types	Example network attributes
Node position	$\mathbb{R}^2$, $\mathbb{R}^3$	Geographic or spatial position
Node label	Categorical	Name or ID of node, group name
Node color	Categorical, continuous	Type of node, node centrality
Node size[4,5]	Continuous	Node degree, node centrality
Node shape	Categorical	Node type, group
Edge color	Continuous	Edge betweenness, edge weight
Edge thickness	Continuous	Edge weight, edge betweenness
Edge style (solid, dashed, dot-dashed)	Categorical (not recommended)	
Edge adornment	Categorical	Arrowheads for directed networks

printed page. Colormaps, if not properly designed, can introduce artifacts not present in the underlying data. Therefore, color is best used sparingly and in concert with another graphical attribute such as line thickness (Sec. 13.2.1).

Table 13.1 summarizes some common mappings between graphical and network attributes.

Choosing attributes A graph visualization almost always benefits from adding attributes even if attributes are not present in the data. For instance, there may be no node attributes but you can always use the size of nodes to represent a network property such as their degree. This will lead to a visualization that emphasizes the degree distribution of the network, making any highly connected hubs jump out of the visualization. Or maybe instead you wish to emphasize structural bottlenecks, in which case drawing edges with high betweenness as very thick lines will immediately draw your viewer's attention. Consider also reinforcing these points by combining multiple visual attributes. Make hubs both larger and more red, emphasizing degree with size and color. Make central edges both thicker and brighter in color. Once you have chosen the message you wish to convey, you can then choose the appropriate graphical attributes to emphasize that message.

Reinforce visual attributes with redundancy. If you use both size and color to represent the same thing, it makes the point even clearer.

[4] It is best to map the area to avoid misleading the viewer about the trends of a numeric quantity.

[5] The size of any node text labels can also be scaled to convey information, generally the same information as the node size itself.

https://doi.org/10.1017/9781009212601.016 Published online by Cambridge University Press

13.2.2 Affordances for the viewer

Seeing a network visualization for the first time can be overwhelming to the viewer so you should take care to make their entry into your graphic as easy as possible. Options to ease the viewer's introduction will depend on the medium the visualization is shown in. In a talk or video, for instance, you can begin by showing a small piece of the network, taking a moment to describe the meaning of nodes and edges, then zoom out to show the full network, either through a sequence of slides or an animation. On the other hand, in a static medium, such as a figure in a paper, you may need to provide guidance through other means.[6] In papers, a descriptive figure caption is crucial.

Take care to avoid relying on visual features that are not accessible. Color blindness, for example, should be considered. Likewise, avoid overwhelming the reader with too many representations of too many attributes.

Legends Legends provide convenient affordances for a viewer of your network visualization. With a legend, you have the opportunity to emphasize the meaning of nodes and links. You can also provide quantitative scales to interpret the attributes you added to your visualization. A color scale to represent node centrality, for instance, can be indicated with an appropriately labeled colorbar. Likewise, if the thickness of lines was used to represent a numeric edge attribute, then the legend can include a collection of lines of increasing thickness labeled with interpretable numeric values. Indeed, as you expand the number of graphical attributes in the visualization, the legend becomes more important for reinforcing your information hierarchy (Sec. 13.2.1).

Network attributes that are reinforced with multiple visual properties, such as using both size and color, should also be combined in the corresponding legend entry. Remember that the legend entries should also be easy to read and understand. We illustrate some example legends in Fig. 13.2.

Callouts and graphical emphasizers Remember that the goal of network visualization, like any visualization, is to communicate a message via illustration. A plot can make a point clear, as can a network diagram. But don't consider yourself limited to only the elements of the plot itself or the network diagram itself. Consider adding additional text and graphics to highlight or emphasize certain aspects of the visualization. For a network diagram, consider drawing a box around the portion of the network most relevant to your point. Add a text label with some arrows calling out particular nodes or edges. Often for large networks there isn't room to label every node, but labeling a select number can be an effective way to emphasize important nodes. You can even devise a scheme to choose which nodes to label, for instance labeling the top-k most central nodes.

[6] Although it may still be possible using multiple figures to first show a cartoon of the network then display the full visualization, perhaps as multiple panels of a single figure.

https://doi.org/10.1017/9781009212601.016 Published online by Cambridge University Press

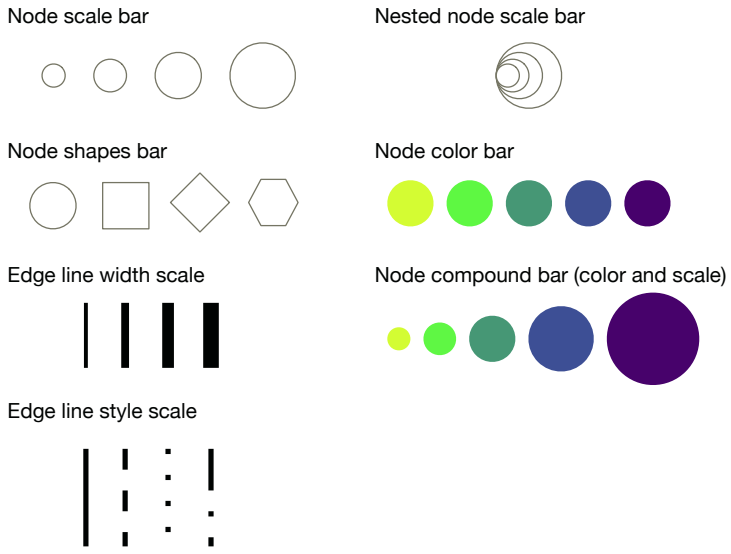

Figure 13.2 Some examples of node and link attribute legends. Figure 1.4, back in Ch. 1, shows an example of a network visualization legend.

13.2.3 Advanced visualization methods

A basic "ball-and-stick" network diagram is a great vehicle for graphical customizations and the node shapes and edge lines can be easily "overloaded" to represent node and edge attributes. Here we discuss some more advanced forms of network visualization and customization that take this even further.

Edge routing and bundling The standard network diagram uses line segments between nodes to draw edges. Straight line segments are easy to follow and often perfectly sufficient, especially when the network layout can minimize most edge crossings. But sometimes a better, more readable layout is possible by using curved lines for edges. Curved edges increase the complexity of drawing the network, as new computations will be needed to place the edges down compared to straight lines which are simply placed at the centers of the adjacency nodes once the node layout is computed.

Edge-routing methods allow edges to curve so they can move around nodes in such a way that edges cross less often. Edge routes can be computed through a variety of algorithms, often treating the problem as an optimization seeking to minimize the length of edge curves and the number of edge crossings. Edge bundling is another approach where certain groups of edges are drawn together into bundles to reduce the visual clutter of the graph while preserving its overall character. Although there are multiple methods, the most common edge-bundling method is force-directed one. To compute edge bundles, it models edges as a chain of point charges and springs, but unlike the repulsive charges used to model nodes in a force-directed layout, charges from nearby edges attract one another. These attractive forces pull together or "bundle" the edges,

https://doi.org/10.1017/9781009212601.016 Published online by Cambridge University Press

reducing clutter in the visualization. A key to visually appealing edge bundling is to pick which edges should interact with one another. Edge pairs that are far apart, or nearly perpendicular to one another, or where one edge is much longer than the other usually lead to unpleasant visuals if bundled together. Generally, a collection of heuristic expressions is used to determine pairwise which edges to bundle [223].

Visualizing multi-networks Some networks consist of multiple networks. These include multilayer and multiplex networks, and networks-of-networks. One approach to visualizing multilayer networks is with a 3D stacking of 2D visualizations, sometimes called a *prism plot*. Each network layer is drawn as a separate graphic, sometimes with a foreshortened perspective added or implied to emphasize a flat planar appearance. These graphs are then arranged top-to-bottom or left-to-right to capture the notion of a stack of networks. When drawing the layers it is important to arrange them based on a meaningful ordering of the network layers. For example, an infrastructure network may have layers of increasing complexity and you can emphasize this in your visualization by stacking the low-level, concrete layers underneath the more involved abstract layers. Usually in these drawings the nodes are placed at the same relative coordinates within a given layer, so the viewer can trace changes in the network structure between the different layers more easily and without having to hunt for nodes between the layers. Both layer placement and node placement within the layers are vehicles for you to emphasize the important facets of your data and to make the viewer's job easier when deciphering your message.

Multiplex networks can also be drawn as stacked graphs, with each layer ℓ corresponding to the edges (and incident nodes) in that layer. However, it is often more common to visualize edge types using other edge graphical attributes (Sec. 13.2.1) such as edge color.

Temporal networks (Ch. 15) are sometimes visualized as multilayer networks, where each layer captures the network during a corresponding time period.

Networks-of-networks, where each node in a super-network is itself a network, are typically visualized by drawing super-nodes themselves as network visualizations. We discuss this idea below.

Of course, the fundamental challenge when visualizing multi-networks is the size of the networks. Visualizing one network is difficult if it's too big. A multilayer network consisting of 5 or 10 layers might contain 5 or 10 times more information, which may be impossible to visualize. So unless the overall network is not too big, implying that each layer of the network is not too big, it may be better to avoid visualizing the full network.

Nodes as graphical elements While not common, on occasion, a great technique for enhancing a network visualization is to use more complex graphical elements for the nodes, instead of the basic squares, circles, and other shapes. One approach is to use a small pie chart for each node. Suppose the network was course-grained so each node in the visualization represents a community or group of nodes in the original network (Sec. 13.4). Pie charts placed at the locations of the visualization's super-nodes could then illustrate properties of the underlying group, for instance the proportions of

different political party affiliations of the group members. While pie charts are generally a poor choice for visualizing a categorical distribution [110], they are very compact, so they might actually work well for nodes in a network diagram.

Besides pie charts, other graphics may be suitable for network nodes. In fact, small network visualizations can themselves be used as nodes. This may be helpful for visualizing a hierarchy or network-of-networks dataset or even illustrating relationships between small graph motifs (Ch. 12). Of course, drawing many networks can lead to a busy, hard-to-parse visualization, so this approach may be quite limited, but it can work well when there are not too many nodes in either the visualization or within the node themselves. Small motifs for instance, can sometimes be drawn quite well, as they typically only consist of a few nodes.

Animation Animated network visualizations are powerful for showing the dynamics of a network. These dynamics can take the form of changes to the nodes or edges, such as a birth–death process or the gains and losses of edges, or they can visualize changes to any network attributes. For example, the set of edges may be fixed in time but edge weights associated with edges may vary over time. An animation could visualize this by drawing the network in each animation frame but varying the edge line widths, which represent edge weights, across frames.

Of course, animating a network is more computationally complex than preparing a single static visualization. Each frame of the animation must be drawn. You can either fix the positions of the nodes in advance and simply show the changes as each frame, or animate the whole layout process and network dynamics at the same time. The latter can be implemented as running the physical (force-directed) simulation and recording the layout at each frame while changing the network itself in real time.

For more on dynamic networks, see Ch. 15

13.3 Alternatives to "ball-and-stick" diagrams

On occasion, a network visualization task may be better accomplished using a non-traditional visualization.

One situation is when capturing dynamics in the network. While it is possible to animate the network, redrawing it over time to show how it changes, this can be done only for media that support animation, such as video. Yet in a static visualization, dynamics can still be visualized by other means. For instance, *stream diagrams*, sometimes also called *phase plots*, can illustrate the temporal dimension of the network explicitly. Here time becomes a plot axis and the network is reduced to a mostly one-dimensional representation orthogonal to the time axis. Edges are then drawn connecting points along the "network axis" for each point in time. Stream diagrams are great for illustrating specific patterns but have scalability problems. For one, a large network is generally not amenable to a pseudo-one-dimension visualization. Second, for a reasonably sized diagram, only a limited number of time periods can be drawn along the time axis. One solution is to use the stream diagram to capture the community structure of the network and how it evolves. Here one can even visualize fissions and fusions of groups with links along the time axis.

https://doi.org/10.1017/9781009212601.016 Published online by Cambridge University Press

Networks as matrices Every network structure can be mapped to a matrix representation such as the adjacency matrix $\mathbf{A}$ or an $N \times N$ similarity matrix $\mathbf{S}$. We can visualize the matrix itself, for example as an $N \times N$ grid of black and white squares,[7] which may be useful in some circumstances but has problems. Most networks are rather sparse, meaning $\mathbf{A}$ and often also $\mathbf{S}$ will be mostly empty. But beyond that, drawing the matrix as a grid requires developing an ordering of the nodes so that rows and columns can be laid out in a meaningful way. Not all networks possess a natural ordering and the same network under different orderings can produce visually very different matrix drawings. One approach to computing a consistent ordering is through hierarchical clustering. Here nodes are grouped into clusters, those clusters are grouped into super-clusters, and so forth. A tree called a dendrogram (Sec. 12.8) can be drawn to visualize this hierarchy, and ordering the leaves of the tree to prevent crossing branches can be used to arrange the rows and columns of the original matrix. Often the dendrogram and the matrix are drawn together in a visualization.

Network portraits One particular matrix, a network "portrait," is helpful for illustrating networks as it overcomes the ordering problem. The portrait is defined based on shortest paths (Ch. 12) between all pairs of nodes. Consider a node i, called a starting node. Following along shortest paths, count how many other nodes are one step away from i (its neighbors), how many nodes are two steps away, three steps, and so forth. Now repeat these counts from each node, treating every node as the starting node in turn. The network portrait $\mathbf{B} = [B_{\ell k}]$ is defined by aggregating these counts:

$$B_{\ell k} = \text{number of starting nodes that have } k \text{ other nodes } \ell \text{ steps away.} \qquad (13.1)$$

The portrait is packed with useful summaries of network structure. The number of nodes is encoded in $B_{01} := N$, as nodes are zero steps away from themselves. The degree distribution is encoded in B_{1k}. The network diameter is captured by the number of nonzero rows of $\mathbf{B}$. Both dimensionality and regularity of the underlying network are captured by the growth and spread of nonzero values across the rows of $\mathbf{B}$. Measures for comparing networks G_i and G_j can be defined as functions of their respective portraits, $\mathbf{B}_i$ and $\mathbf{B}_j$ (Ch. 14). And most importantly for visualization purposes, the matrix is defined entirely based on the network structure and does not require an ordering of rows or columns unlike typical matrix representations.[8] This uniqueness lets network portraits serve as a stable visual representation—heatmaps of $\mathbf{B}$ are an interpretable alternative to traditional network visualizations. Even when a traditional ball-and-stick visualization is used, a portrait can be included to provide a quantitative supplemental visualization. See Fig. 13.3 for some examples of small networks and their portraits.

Multilayer alternatives In some circumstances, it may be best to eschew the network structure itself and focus the visualization on other properties. Suppose you wish to visualize a large multilayer network with several thousand nodes per layer and 10 or more layers. It will be too much information to illustrate all those nodes and provide

[7] You can see some examples of this in Ch. 25.

[8] It is known as a *graph invariant*.

https://doi.org/10.1017/9781009212601.016 Published online by Cambridge University Press

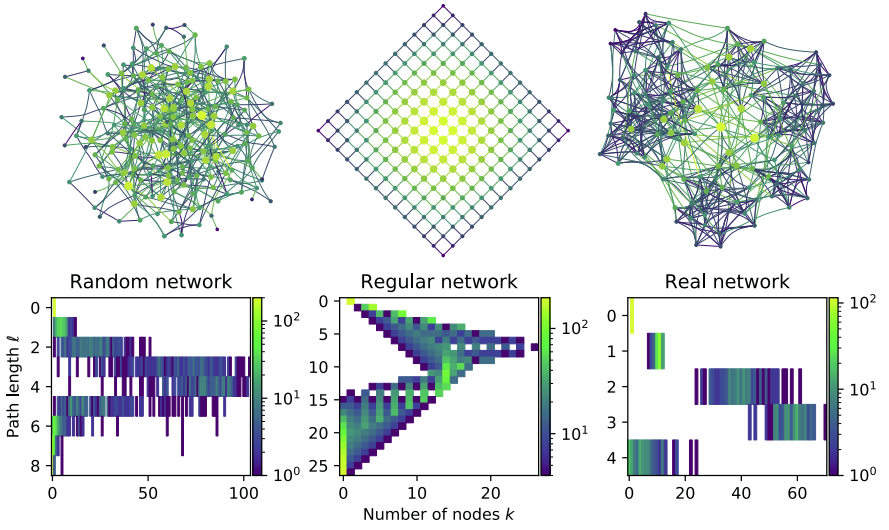

Figure 13.3 Example networks and their portraits. The random network is an Erdős-Rényi graph while the real network is the NCAA Division I football network [360]. Matrix brightness denotes the values of the portrait matrix $\mathbf{B}$ (Eq. (13.1); white indicates $B_{\ell k} = 0$). Figure from [27].

visual space for a viewer to recognize and compare different layers. But perhaps you wish to focus on some specific attributes of the network. Let's say each node in a layer belongs to a category, there are 5 or 10 categories overall, and you wish to understand how the distribution of categories varies across the network layers. You can reduce each layer to a stacked bar plot that captures the category distribution over nodes in that layer, but let's take it a step further. Let's wrap those bars around one another, transform the set of stacked bar charts into concentric rings capturing the network layers from bottom to top. Now, if there are n_ℓ nodes in layer ℓ let's draw ring ℓ as a collection of n_ℓ arcs, each colored according to the node's category. Lastly, let's arrange the arcs using a physics-based layout algorithm (Sec. 13.1.1) but with node positions constrained to move only along their arcs. This layout step will arrange the category colors such that proximity captures network structure, and can reveal patterns in both the distribution of node categories across different layers and what interplay may exist between category and connectivity.

13.4 Processing data for visualization

Network data must often be processed before it can be visualized. Typically this happens when the network itself is too dense for a 2D projection to be visually informative, but often judicious preprocessing can draw out the salient information and enable a readable final visual. Here we discuss options for filtering or transforming the network to improve its visualizability.

One can generally group data manipulations into four "verbs": filter, transform, ag-

https://doi.org/10.1017/9781009212601.016 Published online by Cambridge University Press

gregate, and sort [490]. The final verb, sorting operations, means rearranging the data in different ways. Sorting here makes sense to describe different aspects of the visualization itself, such as rearranging nodes into a more pleasing layout, but is generally not applicable to preprocessing for visualization. When specifically preparing network data prior to visualization, we can group various operations based on the remaining verbs:

Filter Remove spurious or weak links; remove singleton or low-degree nodes; window the data—remove nodes and links outside a given time period; focus on the giant connected component; focus on nodes or links meeting certain criteria such as a particular attribute value.

Transform Project bipartite network; split multilayer network; remove or ignore link directionality (if warranted); look at the graph dual.

Aggregate Merge groups of nodes into a single node (edge bundling is a kind of visual aggregation); collapse or partially collapse multilayer or temporal network.

We discuss some example scenarios.

Transform: bipartite or not Many networks follow a bipartite formulation (Ch. 1), such as a movie–actor or gene–disease network. Often it is best to visualize the network in this original form, using different shapes for the two classes of nodes. However, sometimes an effective visualization can be made by projecting the network down onto one of the two sides, usually by applying the one-mode projection. In other words, this would generate, for example, an actor–actor network where two actors are linked if they costar in a movie, or a movie–movie network where two movies are linked if they both featured at least one actor in common. When to project? Generally, if the network is not too dense following projection, and the nature of unipartite edges is more easily understood, then visualizing the projection can work well. On the other hand, if the projected network becomes very dense—which often happens—it may be best to avoid projection or find alternative ways to reduce the density of the projected network. Note that projecting a bipartite network creates a dense, clique-based network. For instance, if we project an actor–movie network onto an actor–actor network, each movie creates a clique of actors who are all connected together. Thus, it may need to be further reduced by filtering out edges. Projection will also reduce the number of nodes, and this is something to keep in mind; fewer nodes may lead to a smaller, more readable network even if the new network is more dense.

Filter: edges Suppose you have a weighted network. You try drawing every edge, but even with lines of varying thicknesses, this makes the network too dense to read. In such instances, you may use the edge weights to eliminate edges from your visualization. One approach is to define a hard threshold w_{thr} and visualize only edges (i, j) that satisfy $w_{ij} > w_{\text{thr}}$. This requires exploring different values of the threshold, of course, and defining a single threshold may not be meaningful.[9] We discuss more advanced

[9] One approach to defining a single threshold is to look at percentiles of the quantity you wish to threshold. For example, you may decide to retain only edges with weights at or above the 90th percentile of the weight distribution.

approaches to filtering edges in Ch. 10 that are also suitable here for preparing data for visualization.

Filter: nodes Like edges, nodes can be filtered to reduce the complexity of the visualization. In principle, any network property or node attribute can be used to define a filtering criterion, but in practice some are more useful than others. Suppose nodes fall into two groups, A and B. Perhaps only A is worth visualizing, so you draw a network with all the B nodes (and their links) removed. Or perhaps instead you filter nodes by degree, and remove all nodes with only a single neighbor (degree $k = 1$); see the k-core decomposition in Ch. 10. Yet another approach is to filter nodes based on whether they exist in the largest connected (giant) component; drawing only the giant component is often helpful if there exist many isolates that do not need to be visualized.

Aggregate: super-nodes Some networks are very large and it's not meaningful to draw every individual node. Instead, we can aggregate nodes into groups, then make a group–group network and visualize this new network. Now we have a drawing of "super-nodes" representing the groups of the original network. The most common approach to grouping nodes is community detection (Ch. 12). Suppose a community detection method has been used and each node i in G is assigned to a group c_i. The group–group network is then defined by taking all the nodes in G that share a group, $c_i = c_j, i \neq j$, and merging them into a single node.

Edges within a group can be ignored in the visualization while edges between nodes in different groups merged into a single edge.[10] Now we have a much smaller network to visualize. Interestingly, this "supernode" approach is at the heart of some of the most widely-used community detection methods and strategies (Ch. 12).

13.5 Emphasizing your network question in your visualization

Your visualization serves a purpose and it is worth keeping this purpose in mind as you prepare the visualization. Exploring the data and getting a rough macro-level understanding of the network by drawing it is one such purpose. Or at least attempting to, if it turns out to be too large or dense. In contrast, after you've got a good handle on the data, a visualization becomes a means for communication, and you can craft the features of the visual to emphasize the points you wish to make.

To explore a network through visualization requires keeping in mind the structural properties that can be apparent in a drawing of the network and seeing if those properties reveal themselves. One example we discussed above was whether the network is disconnected or not. When drawing the network, the connected component distribution

[10] Information about the original network can be retained as node and edge attributes. For example, each group (supernode) can have an associated attribute which is the number of nodes from the original network that were assigned to that group. Likewise, each group–group edge can have a weight counting how many links in the original graph were between nodes in those two groups. Either or both attribute can be used in the visualization: larger groups can be drawn as larger nodes and more heavily interconnected groups can have thicker edges.

becomes clear almost immediately: do multiple components exist and, if so, what are the sizes of the connected components? That is, is there one component containing the majority of nodes (a giant component; Ch. 12) with few (or many) very small components left over? Or is there a more even distribution of component sizes, with most components being roughly the same size? Yes, the component size distribution can be investigated from the network structure itself. But a drawing can let you quickly explore that property along with many other properties.

Properties other than connectedness can also appear in quick exploratory network visualizations. High-degree hub nodes often pop out of the drawing due to how many links they have, assuming the network is not so dense as to mask their appearance. Are there many hubs or few? And do the hubs tend to connect to other high-degree nodes or not? Whether hubs tend to link to other hubs is actually sometimes difficult to see from a visualization, as hubs tend to be uncommon and their links can be hard to trace in the presence of many other links, unless hub-links are specifically emphasized in their appearance. In contrast, it is more often easy to tell if hubs tend to connect to low-degree nodes, as they will typically (but not always) be surrounded by a fringe of low-degree or even singleton (degree one) nodes.

How else might one customize a network visualization to capture a specific question? Perhaps we are interested in associations between network properties and structure. For instance, suppose we are examining a political affiliation network, where associated with each node is a numeric left-wing/right-wing polarity score. Do hubs tend to be more right-wing or more left-wing? If we visualize the network using node size (area) proportional to degree and node color mapped from polarity score, we can at a glance tell if the hubs, large in size compared to other nodes, also tend to be similar colors. Following along these lines, suppose we hypothesize that the network is modular, with dense groups that share more links with members of their in-group than their out-group. Do we see these groups as clusters in the network drawing? If so, do nodes in clusters tend to be similar in polarity? This clustering and shared polarity can be visually easy to spot by looking for clusters of the same or similar colors.

13.6 Visualization tools

The task of generating network visualizations is common enough that a variety of software tools have been created over the years to assist us. Some tools are simple, bare-bones programs that generate a graphic only, while others provide a large graphical user interface for us to manipulate the visualization in various ways. Some visualization tools are actually larger network analysis platforms or informatics toolboxes that just happen to also provide useful visualization functions.

As with most software, the landscape of specific visualization tools is constantly in flux. Graphviz[11] is a venerable collection of command-line graph drawing tools and both NetworkX[12] and igraph[13] network libraries provide graph drawing functions. For graphical tools, those with a "click-and-drag" visual interface, at the time of writing

[11] https://graphviz.org
[12] https://networkx.org
[13] https://igraph.org/

https://doi.org/10.1017/9781009212601.016 Published online by Cambridge University Press

(March 2023), we have had good luck with Gephi[14] and Cytoscape.[15] Cytoscape, while actually a bioinformatics application, provides high-quality network visualizations and is relatively easy to use. Many layout algorithms can be used and its "Viz Mapper" functionality provides a lot of choices to reinforce your information hierarchy, although it could stand to provide better legend affordances (Sec. 13.2.2).

Beyond specific applications, here are some useful criteria when comparing different visualization tools. The best tools should provide a variety of layout options (Sec. 13.1.1), allow mapping of network attributes to graphical attributes (Sec. 13.2.1), offer a variety of graphical export formats for saving visualization files, and ideally should allow us to manipulate the placement of nodes within the visualization, in case we need to (Sec. 13.2). Another criteria to judge a tool is performance relative to your data. If your network is large and the tool is slow, you may not be able to draw or manipulate the network. Lastly, old software rots over time, and so well-supported and actively maintained tools, ideally open source tools, are always the best to incorporate into your workflow.

13.7 Summary

This chapter toured the use of visualization techniques for network data. The common "ball-and-stick" drawing of a network is straightforward, but the key to a visualization is laying out the nodes properly and choosing the mapping between network and visual properties. Both require iteration to fine-tune the visualization.

Often networks are too dense to be drawn well, in which case you may want to filter or aggregate the network. In general, follow an iterative, back-and-forth workflow when visualizing a network. Try different layout methods and filtering steps to see what best shows the structure of the network while keeping your original questions and goals in mind.

Just as networks are a powerful organizer for connecting data attributes (Ch. 9), visualization can show those attributions with the right mapping. Consider using node size, color, or any other properties to visualize additional node or link attributes. Even network attributes such as centralities can be visualized in this way, compounding the expressiveness of the visualization by reinforcing the message.

Finally, remember that visualization is not always the endpoint of a network analysis. It can also be a useful step in the middle of an exploratory data analysis pipeline, in much the same way that traditional (statistical) visualization powers non-network data exploration. As you're working to understand a network dataset, perhaps by exploring different preprocessing steps, consider making visualizations to see the network you are getting. Even as drafts that are soon discarded, such visuals are useful for both debugging and gaining new insights into the data you are studying.

[14] https://gephi.org
[15] https://cytoscape.org

https://doi.org/10.1017/9781009212601.016 Published online by Cambridge University Press

Bibliographic remarks

Graphs have been drawn within scientific illustrations for hundreds of years, although modern graph drawings did not appear at the origin of graph theory in the works of Euler [255]. Moreno referred to these drawings as "sociograms" [317].

Graph layout algorithms trace back to Tutte [466]. Practical interest arose at Bell Labs as communications engineers, grappling with the increasing complexity of the public telephone system, needed tools to help map out the wiring diagram (literally, in this case). This early work became the "graphviz" library, still being developed today. As with UNIX and the C programming language, science has greatly benefitted from the software developed at Bell Labs that could not be commercialized due to agreement with the United States Government.

Spectral layout methods trace back to Hall [199], while the use of physical heuristics for graph layouts originates in the work of Eades [139], then was extended by Fruchterman and Reingold into the now-classic "force-directed" algorithm. Eades considered only spring forces for edges and thought of nodes as small metal rings upon which edges-as-springs were attached. Fruchterman and Reingold [171] extended this by introducing the use of both repulsive and attractive forces. The Fruchterman–Reingold algorithm is a staple of graph drawing and remains a useful starting point for computing graph layouts.

Marai et al. [291] is a good resource for biologists looking for specific tips for network visualizations in their field.

Exercises

13.1 (**Focal network**) Make a visualization of the plant–pollinator network, using a tool described in Sec. 13.6 (or another one you've found on your own). Use different node symbols to distinguish plants from pollinators. Try several graph layout algorithms, briefly describing the results of each and report which layout seemed to best visualize the network.

13.2 (**Focal network**) Take the Malawi Sociometer Network, write a short computer program that deletes any singleton (degree 1) nodes, repeating this process if new single nodes are created until none remain, then:

(a) Draw this network using a circular layout. Draw nodes as circles large enough to be seen in your visualization but not so large they obscure other elements of the network.

(b) Redraw the network using a physics-based or force-directed layout. Use the size and color of nodes to represent their degree.

13.3 Draw a network of interest using a force-directed layout, but use the size of nodes to represent their degree or another centrality measure (e.g., betweenness centrality; see Ch. 12).

13.4 (**Advanced**) Take a network of interest and apply a community detection algorithm to find a new network where nodes represent communities in the original

network and weighted edges represent the number of edges between nodes in different communities. Draw this community-level network using a physics-based or force-directed layout. Use size and color of nodes to represent community size and use thickness to represent edge weight.

https://doi.org/10.1017/9781009212601.016 Published online by Cambridge University Press