Chapter 15

Dynamics and dynamic networks

Some networks, many in fact, vary with time. They may grow in size, gaining nodes and links. Or they may shrink, losing links and becoming sparser over time. Sitting behind many networks are drivers that change the structure, predictably or not, leading to dynamic networks that exhibit all manner of changes. The focus of this chapter is describing and quantifying such dynamic networks, recognizing the challenges that dynamics bring and finding ways to address those challenges. Dynamic network data brings along practical issues as well, and we discuss working with date and time data and file formats.

15.1 Dynamic networks and dynamics *on* networks

Let's distinguish two sources of dynamics. The first, which plays a role in network analysis but is not our focus here (we discuss it shortly), considers dynamical processes that run on the network topology. Imagine, for example, assigning a time-dependent variable $x_i(t)$ to each node i in the network. This variable tracks a dynamical process across the network where the neighbors of i play a role in how x_i changes. That is, $x_i(t+1)$ is a function not only of $x_i(t)$ but also, among other things, $x_j(t)$, for each node j which is a neighbor of i. The network structure affects the dynamics of x by dint of the neighborhoods. Examples of such dynamics mediated by a network structure include the Kuramoto model [2], voter model [438], and many other models that fall in the class of interacting particle systems [274, 275] and, more generally, dynamical systems [375].

The second source of dynamics is that of the network structure itself. Here the graph representing the network changes with time: $G(t) = (V(t), E(t))$, meaning the node and edge sets are both time-dependent or *indexed* in time.[1] Nodes can appear

[1] Note that time can be discrete or continuous. For the discrete case, time increments in steps, with each timestep $t = 1, 2, \ldots$. This makes it easy to consider changes to the network. Continuous time, where t is now real-valued instead of integer-valued, is more challenging, as now we need to consider mathematically an uncountable index set.

https://doi.org/10.1017/9781009212601.018 Published online by Cambridge University Press

or disappear, as can edges. Evolving network structures and organizing patterns can then manifest: triangles and other motifs can emerge and disintegrate, communities can appear, growing or shrinking over time. The overall density of the network may vary, perhaps periodically, or not.

A dynamic network is also called a *temporal network*.

Many questions arise due to the network's dynamics:

1. Does the network change slowly over time, or drastically? Can we predict what the network will look like in the future, or is its evolution too chaotic?

2. What kinds of changes occur? Are changes due to changes in the node set or the edge set? Both?

3. Do many nodes appear, leading to a growing network? Do nodes disappear, leading to a shrinking network? Are nodes exclusively added over time, exclusively removed, or do both occur together?

4. How do edges changes? Do edges appear or disappear? Both? Do new edges tend to form between existing nodes? Perhaps the node set is static in time and only the edges between those nodes change?

5. How best to quantify or represent the network dynamics? Is it meaningful to simplify the dynamic network with a static network?

6. Can we use information from the dynamics of the network structure to better understand the network?

15.2 Representations

The first challenge with a dynamic network structure is how to represent it. The two most common and complementary approaches are edge-event sequences (*event representation*) and network snapshots (*snapshot representation*). A third, the *signals representation*, appears when time series are available for nodes but the network structure is absent and needs to be inferred.

15.2.1 Event representation

Here we treat the network as a sequence of edge events, where each edge event is a tuple $(u, v, t, \Delta t)$. (We assume $\Delta t > 0$.) This tuple tells us that edge (u, v) appeared at time t and disappeared at time $t + \Delta t$ (which may be the end of our data window if the edge never disappeared). The full dynamics of the network comes from a sequence of the form

$$\text{events} = \{(u_i, v_i, t_i, \Delta t_i), i = 1, 2, \ldots\}\,. \tag{15.1}$$

https://doi.org/10.1017/9781009212601.018 Published online by Cambridge University Press

Of course, this is quite general. We may be studying data where duration is fixed across all events, for example, in which case we can omit Δt. A useful bookkeeping device is to decompose the full set of events (Eq. (15.1)) by edge or by node:

$$\begin{aligned} \text{events}_{uv} &= [(t_i, \Delta t_i) \mid (u, v, t_i, \Delta t_i) \in \text{events}] , \\ \text{events}_{u} &= [(t_i, \Delta t_i) \mid (u, v_i, t_i, \Delta t_i) \in \text{events}] . \end{aligned} \tag{15.2}$$

(We use $[\cdot]$ to denote these as sequences, not sets, ordered in time.)

An equivalent way to consider the event representation is to define a network G containing every edge that appears in at least one event, then define an edge attribute that lists all the time periods $(t, \Delta t)$ when that edge occurred. The length of this attribute can vary from edge to edge, as some edges may appear in more events than others.

15.2.2 Snapshot representation

Another way to treat the dynamics of the network is to divide time into specific periods, sometimes called *windows* (see also Sec. 10.5.4), and aggregate all the edge events in a given period into a single network. This gives us a sequence of networks $G(T_1), G(T_2), \ldots$ for periods $T_1, T_2, \ldots$. We call these *snapshots* or *snapshot networks*. This representation can be derived from an event sequence by taking all the nodes and edges that occur during[2] a time period,

$$\begin{aligned} V(T) &= \bigcup_{t_i \in T} \{u_i, v_i\}, \\ E(T) &= \{(u_i, v_i) \mid t_i \in T\} , \end{aligned} \tag{15.3}$$

and constructing a corresponding network $G(T) = (V(T), E(T))$. (Here $E(T)$ uses the set notation we describe in Ch. 4.)

A word of caution. To properly include nodes and links in the snapshot networks, you need to ensure that the period when a given node or link is present in the network intersects with the time period of the snapshot:

> **Account for event duration when building snapshots.** An event i that begins at time t_i and lasts until time $t_i + \Delta t_i$ may be present in more than one snapshot depending on the duration Δt_i. This can be lost if only using t_i from Eq. (15.1) to define the snapshot graphs.

While this sounds obvious, care may be needed. For example, it is naive to assume that a node or link is present in the window if the earliest time a node or link appears is before the time window and the latest time a node or link appears is after the time window—a node or link may be present for multiple periods before and after the window but not during it.

The sequence $G(T_1), G(T_2), \ldots$ invites us to consider a sequence of corresponding adjacency matrices $\mathbf{A}(T_1), \mathbf{A}(T_2), \ldots$ where $A_{ij}(t) = 1$ if i and j are neighbors at time

[2] We use the notation $t_i \in T$ to denote this, but a more proper notation would consider the intersection of the interval $[t_i, t_i + \Delta t_i]$ with the interval T.

https://doi.org/10.1017/9781009212601.018 Published online by Cambridge University Press

t and zero otherwise, or even an adjacency "tensor" A_{ijt} where a third index has been added to capture time. And we could also consider the sequence as a single multilayer network, where each layer represents one period. While fundamentally the same, such different viewpoints on the network can help with different problems.

Of course, dividing the network into snapshots raises an important question: how to define the periods $T_1, T_2, \ldots$? Sometimes it may be obvious. In a social network, it may make sense to have one snapshot per day, or perhaps one snapshot per week. In a financial network, perhaps business quarters would justify four snapshots per year. In other data it may not be so clear, perhaps there are no obvious cycles to base your snapshots on.[3] One choice in this situation would be to base snapshots on the data: for example, define a fixed number of events n and then choose periods such that each period contains n events. Fundamentally, what has occurred is that the snapshot representation returns us to the *upstream task* (Ch. 7), and with a dynamic network we should take the same care with our snapshots as we did when extracting a static network.

15.2.3 Signals representation

Yet another way to consider a dynamic network is from a multivariate time series, or a set of N time series, one per node. This sounds like the dynamics on networks discussed above where each node i has an associated time-varying quantity $x_i(t)$, but it differs in that we do not know the network—we only know the time series. The most common example is in neuroscience, where functional imaging data gives us activity levels for different regions of the brain but nothing on how those regions are connected. From the time series, we can devise an $N \times N$ similarity matrix, using a measure such as correlation between every pair of time series. We can extract a static network from this matrix, as we discussed in Sec. 10.5, but we can also in principle extract a time-dependent network.

A snapshot representation can be extracted fairly easily: divide the time series into snapshot periods, compute a similarity matrix for each period, and extract a static network from each similarity matrix. When doing so we may want to use overlapping snapshots, creating a sliding-window effect, that may be more robust. Many other approaches exist, utilizing ideas from statistical inference and signal processing. See Dong et al. [133] and Masuda and Lambiotte [297] for overviews.

15.2.4 From dynamic to static

Often a dynamic network is too much to bear, and you may find yourself wishing to have a static network. Maybe you want to apply a particular data processing algorithm which only works for a static network. Or you'd rather begin with something simple and easy to work with before investigating the dynamics. Or you may simply be limited in capacity, with a computer that cannot handle the full dynamic network (this is

[3] Even in more clear situations, you may need to consider less obvious possibilities. For example, a social network that describes a workplace, such as an office building or factory, may need to be divided into snapshots based on work shifts, not work days. Likewise, you may want to make two snapshots for each week, separating weekdays and weekends. This choice leads to snapshots of varying durations, something to keep in mind.

https://doi.org/10.1017/9781009212601.018 Published online by Cambridge University Press

increasingly less of an issue as computers improve in memory capacity and computing power, but networks can get big). Keep in mind that information is necessarily lost when neglecting the dynamics, and you will need to judge whether a static representation is either appropriate or sufficient for your research—it may be neither.

Probably the simplest way to generate a static network from a dynamic one is to simply take the *cumulative network* by accumulating all nodes and links.[4] This cumulative network is defined such that every node and edge that ever appeared in the dynamic network is present in the static network. This accumulation may be fine for simple networks that change slightly, but it has significant potential downsides too. For instance, paths may exist in the static network that are impossible in reality, as the edges of those paths were never present simultaneously.

Significant information can be lost in this accumulation as well. An edge that appeared and disappeared many times and an edge that appeared only once are on equal footing in the cumulative network. One obvious way to address this is with *weights*: assign to each link in the static network a weight counting how many times the link appeared (how many events) over the course of the data or, if appropriate, the total duration of time ($\sum \Delta t$) the link was present. When extracting this static network, it may need further processing. For example, many rare edges may occur. Consider using a thinning technique such as backbone extraction (Ch. 10). Many such techniques exist tailored specifically to dynamic networks.

15.3 Quantifying dynamic networks

Many network statistics discussed in previous chapters can be applied to dynamic networks. But many also need to be modified because they are either ill-suited or entirely ill-defined for dynamic networks. On top of that, we also have information on event times, which we may wish to quantify. Let's discuss some examples:

Degree We can count node degree several ways. First, we can take node i's degree to be the number of unique nodes $j \neq i$ that have at least one event with i: $k_i = |\{j \mid i, j \in \text{events}\}|$. But we may want to define a weighted sum to count for how often j is a neighbor of i. This can distinguish a node with many fleeting neighbors from one with many long-term neighbors.

We may also want to consider the time-dependent degree $k_i(t)$ by counting the number of neighbors of i during some time period denoted t. If we aggregate the events into network snapshots, we can take the degree of the node in each snapshot, essentially defining the degree as the number of edges connecting the node that activate at least once during a snapshot period. One issue here is if we are looking at many snapshots, or otherwise aggregating over small time periods, k_i may be small or even zero most of the time. A solution is to define a time-weighted average degree, like a rolling mean of degree, across snapshots.

[4] This is equivalent to the snapshot representation with a single snapshot covering all events; see the upper left of Fig. 10.1 for an example.

https://doi.org/10.1017/9781009212601.018 Published online by Cambridge University Press

Clustering Clustering considers triangles, and now we need to consider not just whether nodes a, b, c form a triangle but whether they do so at the same point in time, a concept known as *temporal coherence*. To define the clustering coefficient for a node, we need to consider every time period where a two-path exists with that node in the center, and then ask if and for how long during that period do the three nodes form a cycle. Let T_{uv} be the set of all time periods where edge u, v exists, which can be computed from Eq. (15.1). Then the temporal clustering coefficient [263] of a node u is

$$C(u) = \frac{\sum_{v_1<v_2; v_1, v_2 \neq u} \left| T_{uv_1} \cap T_{uv_2} \cap T_{v_1 v_2} \right|}{\sum_{v_1<v_2; v_1, v_2 \neq u} \left| T_{uv_1} \cap T_{uv_2} \right|}, \tag{15.4}$$

where the sums run over every pair of nodes v_1, v_2 such that neither node is u and not separately counting both v_1, v_2 and v_2, v_1. From this we can define a measure like transitivity or an average of C over the nodes.[5]

Paths Temporal versions of shortest paths and related quantities such as betweenness can be defined by accounting for the time-ordered existences of edges as we move along a particular path. One subtlety is that paths in the cumulative (one-snapshot) network always over-count the number of temporally valid paths. In the cumulative network, we assume all the edges on a path exist simultaneously, but this may not be and often isn't true. A path between nodes i and j is temporally valid over a period T if we can start at node i during $t \in T$ and follow edges from i to j forward in time such that those edges exist at the times we need to move on them. We call such paths *temporal paths* and we say i and j are *temporally connected* or that j is *reachable* from i if a temporal path exists from i to j. (Note that, even for undirected networks, temporally valid paths are not necessarily symmetric, we may be able to reach j from i during T but not vice versa.) We can define the temporal path length as the number of edges traversed or, sometimes, the total duration $t_j - t_i$, where t_j is the earliest time we could arrive at node j if we leave node i at time t_j. Shortest path lengths are usually defined based on such "travel times"; several definitions exist [297].

Finding temporal paths allows us to define temporal connected components. The temporal component of i is the largest set of reachable nodes j. This gets complicated quickly, as reachability is not only not symmetric but also not transitive: Just because we can reach node v_2 from v_1 and we can reach v_3 from v_2 does not necessarily mean we can reach v_3 from v_1. It may be that the path between v_2 and v_3 occurs *before* the path between v_1 and v_2. By the time we reach v_2 from v_1, the path onward to v_3 is gone. These effects make connected components more complicated in temporal networks—for example, a node can belong to multiple connected components at the same time![6] And finding temporal components

[5] For an average clustering, we recommend taking an average of $C(u)$ weighted by how often u is present in the network, which can again be calculated from Eq. (15.1). Doing so ensures the contribution of u to the average depends on how often u is present in the network.

[6] See how difficult temporal networks are? We can understand the appeal of ignoring time!

https://doi.org/10.1017/9781009212601.018 Published online by Cambridge University Press

becomes computationally challenging. Unlike in a static network where BFS can find a connected component in $O(M)$ time, finding all temporal components is known to be an NP-complete problem [297].

Centralities Temporal paths allow us to adapt betweenness centrality (Eqs. (12.23) and (12.24)) and closeness centrality (Eq. (12.22)). Another measure, *temporal efficiency*, similar to closeness centrality, is

$$\text{eff}(i) = \frac{1}{N-1} \sum_{j \neq i} \frac{1}{\ell(i,j)}, \tag{15.5}$$

where $\ell(i, j)$ is, for example, the shortest (or fastest) path length observed at any time. Efficiency is sometimes used in place of closeness centrality as it is less biased by disconnected nodes (which appear more often in temporal than static networks) [297].

Many other temporal centralities can be used, including time-dependent eigenvector centralities. One interesting approach is to build a "supra"-matrix that aggregates all the $N \times N$ matrices (such as $\mathbf{A}(t)$) from the snapshots into one larger matrix. The aggregation process can also be interpreted as a multilayer network and spectral properties of the supra-matrix can be used for centralities and ranking. See Taylor et al. [454] and Masuda and Lambiotte [297] for more details.

Communities Communities are especially interesting in dynamic networks. Just as the events lead to evolving sets of nodes and edges, a dynamic network can have communities that grow, shrink, appear or disappear. Different communities can merge into one, and one community can split in two or more. A temporal community detection method should not only find communities but connect together related communities based on their changes, even building an "ancestry structure" relating "parent" and "child" communities.

Methods to find dynamic communities involve adjusting modularity (Eq. (12.15)) to account for time or by using a multilayer representation aggregating the snapshots. We can also treat the snapshots as separate networks, find the communities in each, and use a "matching" algorithm to find which communities in snapshot T_s are closest to which communities in snapshot T_{s-1}. Other approaches include extending matrix factorization to tensor factorization, and applying it to the adjacency tensor with elements A_{ijt}, or extending the stochastic block model to temporal networks and performing inference of block membership. For a detailed overview of such methods, see Masuda and Lambiotte [297].

15.3.1 Change measures and change detection

With network statistics, even simple ones like degree, now being time-dependent, we may want to explore such measures as time series. For example, what does $k_i(t)$ look like over time? In effect, this question points us to a time series exploratory data analysis (Ch. 11), where we look at the networks and their structural measures (Ch. 12) over time.

https://doi.org/10.1017/9781009212601.018 Published online by Cambridge University Press

Further, suppose the network is relatively static until some particular time τ when it suddenly changes. Can we detect this *changepoint* from our data? If the network suddenly became very dense, this would be evident in a sudden increase in $k(t)$, but what about other structural changes? For those we may wish to use a network comparison measure (Ch. 14) and compare adjacent snapshots.

Changepoint detection—and the related question of anomaly detection—is often formulated as a statistical inference problem. The strategy is to build a statistical model whose parameters θ depend on a "switchpoint": $\theta(t) = \theta_1$ if $t \leq \tau; \theta_2$ if $t > \tau$. We can then perform a hypothesis test on whether $\theta_1 \neq \theta_2$ or we can build a Bayesian model and look at the posterior distributions for θ_1 and θ_2 (the posterior of τ even tells us when the change occurs). Masuda and Lambiotte [297] discuss network changepoint detection in more detail.

15.3.2 Time-dependent attributes—dynamics on networks

For the most part, we are focusing on networks that are themselves dynamic, the nodes and edges change. But we can also consider dynamics on top of a network. Consider a dynamical process $x_i(t)$ associated each node i that evolves in time. Often this is something network scientists consider as a mathematical model; $x(t)$ could come from a simulated epidemic running over the network, for example. But we could also have data for such a process. How best to consider it?

For a static network, we recommend considering $x(t)$ as a node attribute (Ch. 9) that varies with time. We can treat each node's $x_i(t)$ separately, and quantify it using any time series or other appropriate statistics. We can also compare between nodes $x_i(t)$ and $x_j(t)$. We can make this comparison at specific times or across time using any manner of summary statistics and comparison measures, such as the correlation coefficient between x_i and x_j. If we focus on nodes that are adjacent, we arrive at an assortativity measure (Sec. 12.5). We can either measure the correlation between x_i and x_j at each time t, giving a time-dependent assortativity across edges, or we can devise a time-independent similarity between x_i and x_j using all times and then incorporate that similarity into an assortativity.

An especially interesting, but also challenging, situation is when we have both dynamics on networks and dynamics of the network. Now we need to understand how $x(t)$ evolves in time on a network that is itself dynamic, no mean feat.

15.4 Null models

Chapter 11 describes ways to understand a static network structure by comparing it to null models, often randomizing the original network while preserving its degree using monte carlo (the edge exchange method) or by building a new network from scratch using the degrees of the original network (the configuration model). Such methods are applicable to dynamic networks as well, but we can do even better by accounting for the temporality of the network in our null.

If we suspect time plays an important role in our network, a temporal null model, chosen appropriately, can dampen or remove the effects of time through randomization.

https://doi.org/10.1017/9781009212601.018 Published online by Cambridge University Press

Considering the event representation, we can randomize events across links or randomize the order events occur on links. And while doing so we may wish to preserve various quantities, such as the number of events on a given link or the time between events.

Some possible null models:

Random times A simple randomization is to distribute the same number of events but uniformly at random over the full time span of the network, call it the interval $[0, t_{\max}]$ or, if we wish to preserve the times when edge u, v exists, the interval $[\min(\text{events}_{uv}), \max(\text{events}_{uv})]$. This preserves the number of events, and the time span of the events but otherwise breaks all other statistics. For one, the distribution of *interevent times*, or waiting time between events, is not the same.[7]

Interevent shuffle Here we explicitly control for the times between events by shuffling the interevent times. If we think of the time span of the edge, $[\min(\text{events}_{uv}), \max(\text{events}_{uv})]$, as being divided up by the events into time segments that add up to the full time span, we simply shuffle those segments. Now the interevent times are fixed, but when those interevent times begin and end is random. Note that doing this requires fixing the first and last interevent time, which preserves the first two and last two events. If there are four or fewer events, you won't be able to carry out this shuffle.

Across-edge shuffle Here we preserve when events occur but randomize which edge they belong to. One way to do this is to build two lists from the events, a list of edges and a list of times, randomly permute the list of times, then put the list of edges and times back together to create a random events list:

$$\text{events}_{\text{null}} = \left\{(u_i, v_i, t_{\sigma(i)}, \Delta t_{\sigma(i)}), i = 1, 2, \ldots\right\}, \tag{15.6}$$

where $\sigma(i)$ in the subscript of t represents a random permutation across all event indexes $i = 1, 2, \ldots$.

Edge exchange The temporal analog of edge exchange, here we exchange the endpoints of edge pairs in the sequence of events, and randomly distribute the original edges' events between the new edges.

Complicating matters further, we can also apply multiple randomizations one after the other, testing even more the role of temporality by destroying it further in our null. In fact, a broader mathematical framework classifying these and other temporal nulls as different levels of a graph *ensemble* (Ch. 22) was recently proposed [175].

What about snapshots? While we've discussed null models in terms of the event representation, where it's generally easier to envision the nulls, the same procedures can be applied to the snapshots by first randomizing the events, aggregating them into the snapshots $G_{\text{null}}(T_1), G_{\text{null}}(T_2), \ldots$ and then proceeding with further analysis. Depending

[7] In fact, this randomization turns the event sequence into a Poisson process, where events occur at random and without memory, meaning the probability for a new event to happen does not change given when previous events happened.

https://doi.org/10.1017/9781009212601.018 Published online by Cambridge University Press

on what the null preserved, we can even compare the real and random snapshots on a per-period basis ($G(T)$ vs. $G_{\text{null}}(T)$) using some techniques outlined in Ch. 14. That said, if you only have the snapshot networks and not the event times, you can still randomize the $G(T)$ individually but information such as interevent times will not be available, so not every null model can be used.

15.5 Visualization

15.5.1 Dynamic and static network visualization

Visualization can show how a dynamic network changes. One option is to create an animation (Sec. 13.2.3), drawing the network and then changing the drawing as the network changes, but animated visualizations are limited to certain contexts such as webpages or slideshows. A static network visualization can still be used on dynamic networks in a few ways. One is to build a larger network that contains one or more snapshots and visualize those. One can draw several network snapshots next to one another, with common node positions across time—essentially, treating time as the layers in a multilayer network and drawing a prism plot (Sec. 13.2.3). This may work well for a small network but a viewer may struggle to scan across snapshots if the network is too large.

Another way to incorporate temporal information into a static visualization is by mapping time-related quantities to graphical attributes. For example, node color can be used to represent age, the time when the node first appeared in the network, with, say, blue nodes being very young, and red nodes being old. Link age can also be mapped likewise. We can even combine these ideas with the previous approach by visualizing multiple snapshots using graphics for various temporal attributes. Some creativity can go a long way.

Consider using node and link attributes based on temporal quantities in a static visualization of a dynamic network.

As you can imagine, however, it's easy to overdo it. Static visualizations can struggle when too much is happening. The network could be very large, or change very quickly, or have lots of different changes occurring simultaneously—it can be hard to capture when nodes or edges are removed as well as added. As with most visualizations, we have to fight against overloading the graphics and rendering them unreadable.

One possibility to overcome this is to tailor a *changepoint* visualization. Pick two time periods[8] and look at the difference in the networks. How many nodes exist in the first period but not the second? How many nodes exist in the second period but not the first? How many were in both? Then draw the network as the union of the network structure in the two periods, showing all nodes, including those that were born and those that died. Pick some visual attributes to map the node status (born, died, survived, etc.). From the visuals[9] we can then see how much change may have occurred and

[8] It may be possible to extend this beyond two periods but comparison becomes difficult and it may be too hard for the viewer to interpret the final product. Care is needed when balancing complexity and readability.

[9] And visuals can and often should be complemented with quantitative analysis.

https://doi.org/10.1017/9781009212601.018 Published online by Cambridge University Press

if that change followed a pattern. Perhaps neighbors of newly created nodes tend to die, indicating a spreading process or at least some network association between the addition and removal of nodes. Perhaps nodes that survive the longest tend to be high degree, or low. By focusing not necessarily on the full dynamics, but on one point, we can narrow the complexity of the visualization and tailor it for what is (hopefully) the most important moment in the network's dynamics.

As with the snapshot representation itself, the key to this changepoint visualization is determining the two time periods. As a researcher you may have a specific hypothesis in mind that can clue you into which periods to look at. But otherwise you may wish to do some exploration first. For example, vary the duration of the two periods and slide them along your data window. As you do, examine a network property such as the density of the network in the two periods or how much the networks differ. Changepoint detection can also help you find the best snapshots to use. Network differences can be measured by looking at similarities in nodes and edges; see also Ch. 14 on comparing networks. See Ch. 10 for techniques specific to windowing data.

15.5.2 Other visualizations

Beyond creating node and edge attributes related to time and generating a static network visualization, non-network visualizations are also often helpful. Any kind of time series visualization may be appropriate, especially if working with the signals representation. Most network statistics are now time-dependent, so even basic plots of $\langle k \rangle$, number of nodes or edges, and more as functions of time may be helpful. Visualizing the distribution of Δt may be useful, or even the number of events over time. Some more involved visualizations can help certain analyses. For instance, if you have found temporal communities, then an "alluvial" diagram (also called a Sankey diagram), showing the communities as flows, can highlight their sizes over time.

Space-time matrices We have found one uncommon non-network visualization to be quite helpful, which we call, somewhat humorously, the *space-time matrix*. For a dynamic network consisting of $T_{\max}$ snapshots, $G(1), G(2), \ldots, G(T_{\max})$, let M and N be the numbers of edges and nodes that appear in at least one snapshot:

$$M = \left| \cup_{t=1}^{T_{\max}} E(t) \right|, \quad N = \left| \cup_{t=1}^{T_{\max}} V(t) \right|. \tag{15.7}$$

The space-time edge matrix $\mathbf{E}$ is an $M \times T_{\max}$ matrix with one row per edge and one column per snapshot. Likewise, the space-time node matrix $\mathbf{N}$ is the $N \times T_{\max}$ matrix with one row per node and one column per snapshot. The entries E_{eT} and N_{vT} of these matrices equal the number of events involving edge e or node v, respectively, during snapshot period T:

$$\begin{aligned} E_{eT} &= |\{\text{events}_{uv} \mid t_i \in T; (u, v) = e\}|, \\ N_{vT} &= |\{\text{events}_{v} \mid t_i \in T\}|. \end{aligned} \tag{15.8}$$

In other words, the matrices tell us how much activity is happening across the elements of the network over different time periods.

https://doi.org/10.1017/9781009212601.018 Published online by Cambridge University Press

We find plotting these matrices, especially **E**, to be fascinating. Let's look at them for the Malawi Sociometer Network, a temporal focal network.

This undirected network came from contact observations between participants wearing sociometers, small sensors that detect and record when they are in proximity with other sensors. Some data filtering was performed by Ozella et al. [353] after data collection and we are left with observations of edge events at $\Delta t = 20$s over a 14-day period. These data follow the form of Eq. (15.1) without Δt readings. For the most part, until now, when working with this network, we have reduced it to a static, weighted network by creating a snapshot representation with a single period. But now, let's explore its temporality.

We construct daily snapshots and proceed to build **E** and **N**. Looping over the edge events per snapshot, we count the number of events involving edge u, v and node u, populating the entries of **E** and **N**. We order the rows of these matrices by the first appearance times of the corresponding edges and nodes: $\min(\text{events}_{uv})$ for edge u, v and $\min(\text{events}_u)$ for node u so the first edge that appears is the first row of **E**, the second edge that appears is the second row, and so forth (and likewise for **N**). We similarly order the columns of **E** and **N** by snapshot period (day). Figure 15.1 shows the final matrices.

Information about the network immediately jumps out at us in Fig. 15.1. We see that about one-third of edges, all appearing on day 1, often frequently reappear throughout the remaining snapshots. The remaining two-thirds of edges, meanwhile, which tend to

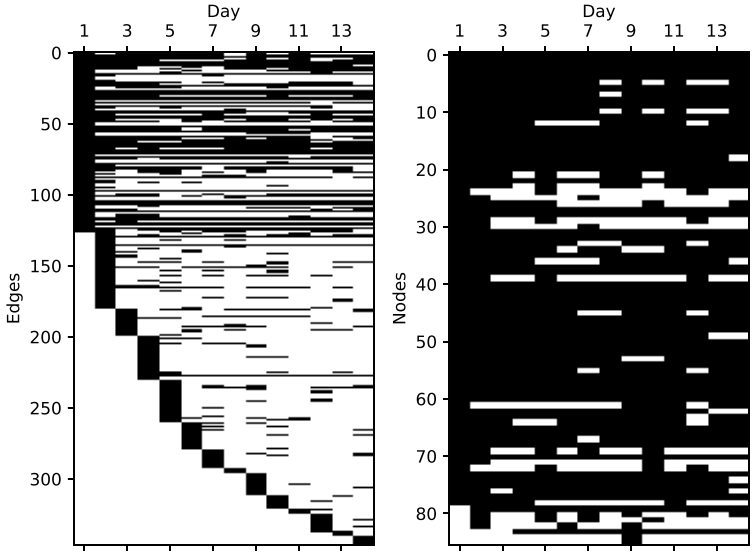

Figure 15.1 "Space-time" matrices for the Malawi Sociometer Network. Note that we have binarized the matrices into zero and nonzero entries, for ease of printing. A bit more information is available using a color scale for the matrix element values, but even the binarized view is very informative.

https://doi.org/10.1017/9781009212601.018 Published online by Cambridge University Press

appear after day 1, tend not to reoccur on later days. In other words, **E** clearly shows a kind of temporally "unrolled" core–periphery structure. Fascinating!

Figure 15.1 also shows **N**. The view we see is less exciting, but still important. Most nodes appear immediately on day 1 and persist, appearing in most subsequent snapshots. This tells us that the fleeting edges in **E** generally lie between preexisting nodes; if **N** showed a pattern like that in **E**, the network would be growing over time.

Of course, this view doesn't capture all possible temporal information, how could it? For one, among the two-thirds of fleeting (peripheral) edges, it is not obvious which nodes they connect to just from the matrix. This could be investigated. In general, we should expect many other patterns to appear in these data. Visualizations and quantitative analyses using some of the techniques described in this and other chapters, can extract such information.[10]

15.6 Further considerations

Some practical details specific to dynamic networks and their data warrant further discussion.

15.6.1 Storing dynamic network data

Many data formats can store a dynamic network (see also Ch. 8). We recommend a simple format, a temporal edge list. Each line of a temporal edge list takes the form

```
node <coldelim> node <coldelim> timestamp <rowdelim>
```

where `<coldelim>`, usually a comma or tab character, separates columns, `<rowdelim>`, usually a newline, separates rows, and `timestamp` indicates when the link occurred.

When links exist over durations we recommend using a start timestamp and either a stop timestamp to indicate the duration, or the duration itself. (Links that appear and disappear multiple times would have multiple rows in the file, with an associated start, stop duration for each appearance.) This is equivalent to recording Eq. (15.1).

Notice in this representation that augmenting the edge list with time information maps nicely to the edge attributes[11] discussed in Ch. 9. Essentially, time is another source of edge attributes with which we can describe the network.

15.6.2 Times and timestamps

Working with network data requires working with underlying data, and that often includes times and timestamps. Time data are no fun to deal with. How can you tell in a computer code that "12 Dec 2012," "December 12th, 2012," and "12/12/12" are all the same date? How can you deal with time zones? Daylight saving time?

Timestamps, written recordings of dates and times, usually take one of two forms. The first, ***epoch-based***, defines an "epoch" reference time and a unit of duration, then each time becomes a count of how many time units since the epoch. The UNIX "Epoch"

[10] Interested? We apply a statistical model, the *edge observer model*, to these data in Ch. 23.

[11] Or, more precisely, multi-edge attributes.

https://doi.org/10.1017/9781009212601.018 Published online by Cambridge University Press

time is one example, measuring the number of seconds that have elapsed since 00:00:00 UTC, 1 January 1970.[12] If you see timestamps in your data that are recorded only as numbers, you probably have an epoch-based timestamp.[13] The second form of timestamp, ***language-based***, follows natural language records of time, such as the "December 12th, 2012" or "12/12/12" examples we discussed.

An epoch-based timestamp format is easy to deal with computationally once you know the epoch time and the duration unit. But these two facts are not readily contained in a list of numbers. Does "34095" mean 34 thousand seconds or 34 thousand minutes? Hopefully, from context you can rule out some possibilities, but ideally the data you are working with properly document these two facts.

A language-based timestamp, on the other hand, is more challenging to work with computationally, as you need to parse the text of the timestamp into a data structure used by your code. All modern programming languages come with support for parsing timestamps and performing operations on dates and times. The challenge really comes when the language-based timestamps are messy, not following a single written form. If every timestamp is written in the form "12/12/12" you may be fine (assuming you know it's always day first or month first). But when different timestamp formats show up, which often happens with data produced manually by people or data coming from different sources, you may need to produce code that handles multiple formats. Some libraries exist for inferring timestamp formats in an automatic fashion, but it is best not to rely on them if you can.

If you are producing data with timestamps, we recommend taking a hard stance and always use the international standard for times and dates, called *ISO 8601*. This standard records dates as `YYYYMMDD` or `YYYY-MM-DD` using four-digit years and zero-padded two-digit months and days (example: `2012-12-01`), and times as `Thhmmss` or `Thh:mm:ss` using a "T" to denote time and a 24-hour clock (example: `T16:20:00`). Dates and times are combined using `<date><time>` (example: `2012-1201T16:20:00`). Time zones can be included as time offsets from universal time (UTC) (example: `2012-12-01T16:20:00-05:00`). While verbose compared to some formats, this standard is generally the most supported and least ambiguous. It also has the attractive property that alphabetically sorting timestamps in this format will also sort them chronologically.

15.7 Summary

Dynamic or temporal networks have special circumstances both in how you prepare the data and in what you do with the generated network. Note also the distinction between a network whose structure is changing (dynamics *of* networks) and a network with a static structure where a dynamic process is running on top of that structure (dynamics *on* networks). It can be acceptable to neglect temporality, avoiding much complexity and nasty issues, but sometimes we have no choice but to confront the mess of time if we want to truly understand our network.

[12] Wonderfully, UNIX time excludes leap seconds. What could possibly go wrong?

[13] Assuming they really are timestamps and aren't confused with elapsed times or durations.

https://doi.org/10.1017/9781009212601.018 Published online by Cambridge University Press

Steps to deal with temporal networks include quantifying the dynamics or finding static representations. The former remains a developing field with many approaches but no broadly accepted standards; the latter is often necessary when you wish to apply a network analysis method that does not account for temporality.

Many dynamic networks come from time series data and some time series may require processing natural language date and time information. Use libraries to assist with this processing as much as possible; when generating your own datestamps and timestamps, favor the ISO 8601 standard.

Bibliographic remarks

Thorough treatments of dynamic networks, particularly theoretical aspects, are given by Holme and Saramäki [221, 222] and Masuda and Lambiotte [297].

For those interested in learning more about time series analysis in general, we recommend Kirchgässner et al. [243], Shumway [430], and, for time series forecasting, Hyndman and Athanasopoulos [227].

Exercises

15.1 Given two snapshot networks, $G(T_s)$ and $G(T_{s+1})$ define a simple comparison measure to quantify how similar they are. (*Hint*: unlike most of the comparison measures discussed in Ch. 14, here we can safely assume that nodes are consistent between the networks.) Briefly describe the intuition behind your comparison, what it measures and why.

15.2 (**Focal network**) Devise daily snapshots of the Malawi Sociometer Network network.

(a) Which snapshot is densest?

(b) Which snapshot changed the most compared to the preceding snapshot?

15.3 (**Focal network**) Compute the *total appearance time* for each node in the Malawi Sociometer Network, defined as the elapsed time between the earliest and latest appearance of that node.

(a) How many nodes are "long-lived" and how many are "short-lived"?

(b) What is the mean and median appearance time for nodes?

(c) What does the distribution of total appearance time look like? Make a histogram or ECDF and interpret.

(d) Total appearance time may not be the best measure for tracking a node's lifetime. What information is lost when using total appearance time and what may be a better measure?

15.4 (**Focal network**) Repeat Ex. 15.3 but for the total appearance time of *edges* instead of nodes. Compare node and edge quantities.

https://doi.org/10.1017/9781009212601.018 Published online by Cambridge University Press

15.5 (**Focal network**) *Link prediction I*: Divide the Malawi Sociometer Network into two snapshots, G_1 and G_2, where G_1 covers the first half of the network's time span and G_2 covers the second. Use the following baseline link prediction rule: Predict that a link exists in the second snapshot if it existed in the first snapshot; otherwise, predict it does not exist.

(a) How many predicted links were actually present in G_2? How many predicted links were absent? How many links not predicted to be in G_2 were actually present?

(b) Interpret the "predictability" of the network based on how well or how badly the baseline link prediction rule works.

15.6 (**Focal network**) *Link prediction II*: Divide the Malawi Sociometer Network into G_1 and G_2 as in Ex. 15.5.

(a) Use the first snapshot to compute a link prediction scoring function (Sec. 10.6.2). Apply the scoring function to each pair of nodes.

(b) Use the second snapshot to validate (how?) the scoring function.

(c) Repeat your analysis but with two snapshots, where the first snapshot covers every day but the last day, and the second snapshot is the last day. How do your results change compared to the 50/50 split?

Provide a brief write-up describing the details of your work on this exercise.

https://doi.org/10.1017/9781009212601.018 Published online by Cambridge University Press