6
Basic Graphs

6.1 Simple Graphs

Every story should start from the beginning and, in this case, in the beginning was the graph[1,2,3,4]. To explain and decompose the elements of a graph, I'm going to use the recurrent example of social networks. The same graph can represent different networks: power grids, protein interactions, financial transactions. Hopefully, you can effortlessly translate these examples into whatever domain you're going to work.

Let's start by defining the fundamental elements of a social network. In society, the fundamental starting point is you. The person. Following Euler's logic that I discussed in the introduction, we want to strip out the internal structure of the person to get to a node. It's like a point in geometry: it's the fundamental concept, one that you cannot divide up into any sub-parts. Each person in a social network is a node – or vertex; in the book I'll treat these two terms as synonyms. We can also call nodes "actors" because they are the ones interacting and making events happen – or "entities" because sometimes they are not actors: rather than making things happen, things happen to them. "Actor" is a more specific term which is not an exact synonym of "node", but we'll see the difference between the two once we complicate our network model just a bit[5], in Section 7.2.

To add some notation, we usually refer to a graph as G. V indicates the set of G's vertices. Since V is the set of nodes, to refer to the number of nodes of a graph we use $|V|$ – some books will use n, but I'll try to avoid it. Throughout the book, I'll tend to use u and v to indicate single nodes.

So far, so good. However, you cannot have a society with only one individual. You need more than one. And, once you have at least two people, you need interactions between them. Again, following Euler, for now we forget about everything that happens in the internal structure of the communication: we only remember that an interaction is

[1] John Adrian Bondy, Uppaluri Siva Ramachandra Murty, et al. *Graph theory with applications*, volume 290. Citeseer, 1976

[2] Douglas Brent West et al. *Introduction to graph theory*, volume 2. Prentice hall Upper Saddle River, 2001

[3] Reinhard Diestel. *Graph theory*. Springer Publishing Company, Incorporated, 2018

[4] Jonathan L Gross and Jay Yellen. *Graph theory and its applications*. CRC press, 2005

[5] The understatement of the century.

taking place. We will have plenty of time to make this model more complicated. The most common terms used to talk about interactions are "edge", "link", "connection" or "arc". While some texts use them with specific distinctions, for me they are going to be synonyms, and my preferred term will always be "edge". I think it's clearer if you always are explicit when you refer to special cases: sure, you can decide that "arc" means "directed edge", but the explicit formula "directed edge" is always better than remembering an additional term, because it contains all the information you need. (What the hell are "directed edges"? Patience, everything will be clear)

Again, notation. E indicates the set of G's edges and $|E|$ is the number of edges – some books will use m as a synonym for $|E|$. Usually, when talking about a specific edge one will use the notation (u, v), because edges are pairs of nodes – unless we complicate the graph model. Now we have a way to refer to the simplest possible graph model: $G = (V, E)$, with $E \subseteq V \times V$. A graph is a set of nodes and a set of edges – i.e. node pairs – established among those nodes.

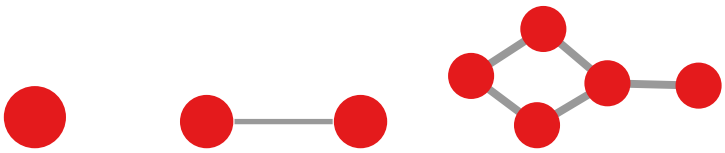


Figure 6.1: (a) A node. (b) An edge. (c) A simple graph.

We're going to talk about how to visualize networks much later in Part XIII, but it's better to introduce some visual elements now, otherwise how are we supposed to have figures before then? Nodes are usually represented as dots, or circles – Figure 6.1(a). Edges are lines connecting the dots – Figure 6.1(b). When all you have is nodes and edges, then you have a simple graph – Figure 6.1(c). Note that these visual elements are basic and widely used, but they are by no means the only way to visualize nodes and edges. In fact, when you want to convey a message about a network of non-trivial size, they're usually not a great idea.

The first famous graph in history is Euler's Königsberg graph, which I show in Figure 6.2. In the graph, each node represents a landmass and each edge represents a bridge connecting two landmasses. Since there were multiple bridges connecting the same landmasses, we have multiple edges between the same two nodes. This seemingly trivial fact is actually rather interesting.

"Simple graph" means *literally* simple: nothing more than nodes and edges – no attributes, no possibility of having multiple connections between the same two nodes. If you add any special feature, it's not a simple graph any more. Under this light, we discover that

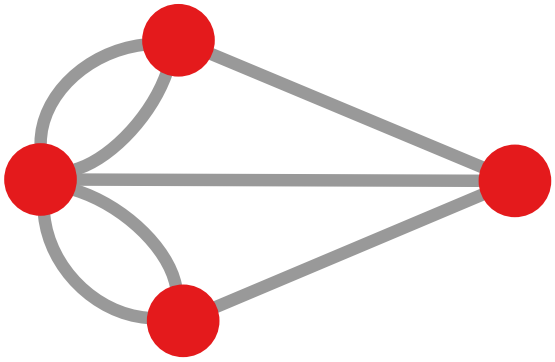
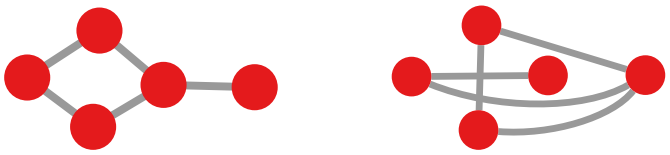


Figure 6.2: The famous Königsberg graph Euler used.

Euler's first graph wasn't simple after all. It allowed for parallel edges: multiple edges between the same two nodes. Euler's first graph was a multigraph. That's so non-standard that we're not even going to talk about it in this chapter: you'll have to wait for the next one, specifically for Section 7.2.

In our simple graph we also assume there are no self loops, which are edges connecting a node with itself. Our assumption is that we aren't psychopaths: everybody is friend with themselves, so we don't need to keep track of those connections.



(a) (b)

Figure 6.3: (a) A simple graph. (b) Its complement.

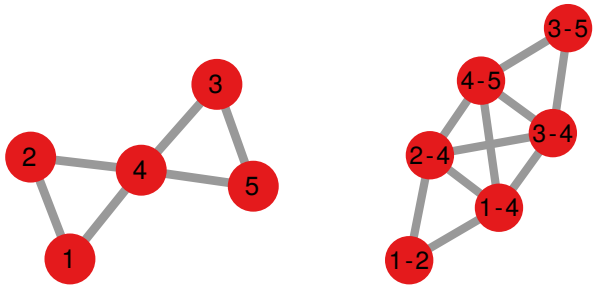
When you have a simple graph G, you can derive a series of special simple graphs related to G. For instance, you can derive the complement of G. This is equivalent to remove all of the original edges of G, and then connect all the unconnected pairs of nodes in G. Figure 6.3 shows an example.

This operation basically views G as a set of edges. If you take this perspective, you can define many operations on graphs as sets. Given two graphs G' and G'', you can calculate their union, intersection, and difference, which are the union, intersection, and difference of their edge sets. The union of G' and G'' is a graph G that has the edges found in either G' or G''; the intersection of G' and G'' is a graph G that has the edges found in both G' and G''; and the difference of G' and G'' is a graph G that has the edges found in G' but not in G''.

Another important special graph is the line graph[6,7]. The line graph of G represents each of G's edges as a node. Two nodes in the line graph are connected to each other if the edges they represent

[6] Hassler Whitney. Congruent graphs and the connectivity of graphs. *American Journal of Mathematics*, 54(1):150–168, 1932

[7] József Krausz. Démonstration nouvelle d'une théoreme de whitney sur les réseaux. *Mat. Fiz. Lapok*, 50(1):75–85, 1943



(a) (b)

Figure 6.4: (a) A graph. (b) Its linegraph version.

are attached to the same node in G. Figure 6.4 shows an example of line graph. We'll see how you can use line graphs to represent high order relationships in Chapter 34, to find overlapping communities in Chapter 38, and to estimate similarities between networks in Chapter 48.

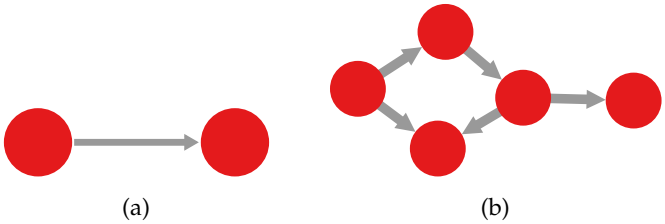
6.2 *Directed Graphs*

Simple graphs are awesome. They allow you to represent a surprising variety of different complex systems. But they are not the end all be all of network theory. There are many phenomena out there that cannot be simply reduced to a set of nodes interacting through a set of edges. Sometimes you really need to complicate stuff. In this and in the next section we're going to see two ways to enhance the simple graph models. They all work in the same way: by slightly modifying the definition of an edge. We're going to see even more fundamental reworkings of the simple graph model in Chapter 7.

The first thing we will do is realizing that not all relations are reciprocal. The fact that I consider you as my friend – and I do, my dear reader – doesn't necessarily mean that you also consider me as your friend – wow, this book is getting very real very fast. We can introduce this asymmetry in the graph model. So far we said that (u, v) is an edge and we implicitly assumed that (u, v) is the same as (v, u). Directed graphs[8] are graphs for which $(u, v) \neq (v, u)$.

In a message passing game, (u, v) – or $u \rightarrow v$ – means that node u can pass a message to node v, but v cannot send it back to u. Directed graphs introduce all sorts of intricacies when it comes to finding paths in the network, a topic we're going to dissect in Chapter 10. The use of the arrow is a pretty straightforward metaphor to indicate the lack of reciprocity: relationships flow from the tail to the head of the arrow, not the other way around. It comes as no surprise, then, that we can use the arrow to indicate a directed edge, as we do in Figure 6.5(a). If E contains directed edges, we have a directed

[8] Frank Harary, Robert Zane Norman, and Dorwin Cartwright. *Structural models: An introduction to the theory of directed graphs*. Wiley, 1965



(a) (b)

Figure 6.5: (a) A directed edge. (b) A directed graph.

graph – Figure 6.5(b).

Note that, in a directed graph (or digraph) representation, an edge always has a direction. If two nodes have a reciprocal relationship, convention dictates that we draw two directed edges pointing in the two directions, to make such relationship explicit.

In general, when you have a directed graph G, you can calculate its reverse graph by flipping all edge directions.

6.3 *Weighted Graphs*

Another way to make edges more interesting is realizing that two connections are not necessarily equally important in the network. One of the two might be much stronger than another. We are all familiar with the concepts of "best friend" and "Facebook friend". One is a much more tightly knit connection than the other.

For this reason, we can add weights to the edges[9,10]. A weight is simply an additional quantitative information we add to the connection. A possible notation could be (u, v, w): nodes u and v connect to each other with strength w. So our graph definition now changes to $G = (V, E, W)$, where W is our set of possible weights. W is practically always included in the set of real numbers, and most of the times in the set of real positive numbers – i.e. $W \subseteq R^+$. Now we have a weighted graph.

[9] Alain Barrat, Marc Barthelemy, Romualdo Pastor-Satorras, and Alessandro Vespignani. The architecture of complex weighted networks. *Proceedings of the national academy of sciences*, 101(11): 3747–3752, 2004a

[10] Mark EJ Newman. Analysis of weighted networks. *Physical review E*, 70 (5):056131, 2004a

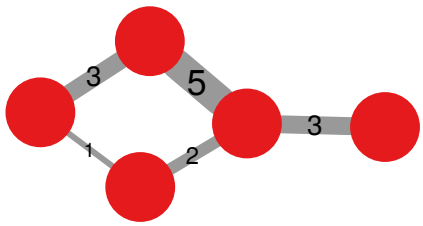


Figure 6.6: A weighted graph. The weight of the edge dictates its label and thickness.

Graphically, we usually represent the weight of a connection either by labeling the edge with its value, or simply by using visual elements such as the line thickness. I do both things in Figure 6.6.

Edge weights can be interpreted in two opposite ways, depending on what the network is representing. They can be considered the

proximity between the two nodes or their *distance*. This can and will influence the results of many algorithms you'll apply to your graph, so this semantic distinction matters. For instance, if you're looking for the shortest path (see Chapter 13) in a road network, your edge weight could mean different things. It could be a distance if it represents the length of the trait of road: longer traits will take more time to cross. Or it can be a proximity: it could be the throughput of the trait of road in number of cars per minute that can pass through it – or the number of lanes. If the weight is a distance, the shortest path should avoid high edge weights. If the weight is a proximity, it should do its best to include them.

To sum up, "proximity" means that a high weight makes the nodes closer together; e.g. they interact a lot, the edge has a high capacity. "Distance" means that a high weight makes the nodes further apart; e.g. it's harder or costly to make the nodes interact.

Edge weights don't have to be positive. Nobody says nodes should be friends! Examples of negative edge weights can be resistances in electric circuits or genes downregulating other genes. This observation is the beginning of a slippery slope towards signed networks, which is a topic for another time (namely, for Section 7.2, if you want to jump there).

The network in Figure 6.6 has nice integer weights. In this case, the edge weights are akin to counts. For instance, in a phone call network, it could be the number of times two people have called each other. Unfortunately, not all weighted networks look as neat as the example in Figure 6.6. In fact, most of the weighted networks you might work with will have continuous edge weights. In that case, many assumptions you can make for count weights won't apply – for instance when filtering connections, as we will see in Chapter 27.

By far, the most common case is the one of correlation networks. In these networks, the nodes aren't really interacting directly with one another. Instead, we are connecting nodes because they are similar to each other, for some definition of similarity. For instance, we could connect brain areas via cortical thickness correlations[11], or currencies according to their exchange rate[12], or correlating the taxa presence in different biological communities[13].

These cases have more or less the same structure. I provide an example in Figure 6.7. In this case, nodes are numerical vectors, which could represent a set of attributes, for instance. We calculate a correlation between the vectors, or some sort of attribute similarity – for instance mutual information (Section 3.5). We then obtain continuous weights, which typically span from -1 to 1. And, since every pair of nodes have a similarity (because any two vectors can be correlated, minus extremely rare degenerate cases), every node is connected to

[11] Boris C Bernhardt, Zhang Chen, Yong He, Alan C Evans, and Neda Bernasconi. Graph-theoretical analysis reveals disrupted small-world organization of cortical thickness correlation networks in temporal lobe epilepsy. *Cerebral cortex*, 21(9):2147–2157, 2011

[12] Takayuki Mizuno, Hideki Takayasu, and Misako Takayasu. Correlation networks among currencies. *Physica A: Statistical Mechanics and its Applications*, 364:336–342, 2006

[13] Jonathan Friedman and Eric J Alm. Inferring correlation networks from genomic survey data. *PLoS computational biology*, 8(9):e1002687, 2012

every other node. So, when working with similarity networks, you will have to filter your connections somehow, a process we call "network backboning" which is far less trivial that it might sound. We will explore it in Chapter 27.

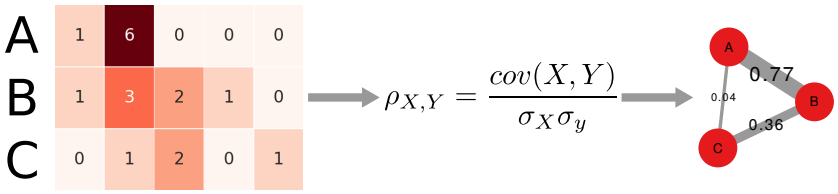


Figure 6.7: A typical workflow for correlation networks: (left to right) from nodes represented as some sort of vectors, to a graph with a similarity measure as edge weigth.

6.4 Network Types

Now that you know more about the various features of different network models, we can start looking at different types of networks. I'm going to use a taxonomy for this section. I find this way of organizing networks useful to think about the objects I work with.

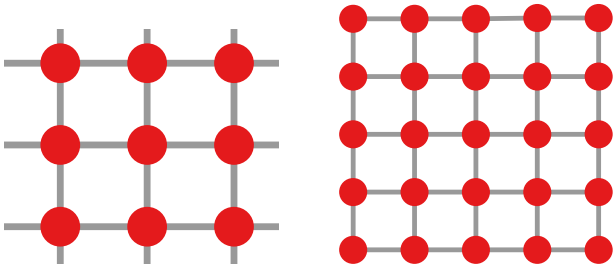
Simple Networks

The first important distinction between network types is between *simple* and *complex* networks. A simple network is a network we can fully describe analytically. Its topological features are exact and trivial. You can have a simple formula that tells you everything you need to know about it. In complex networks that is not possible, you can only use formulas to approximate their salient characteristics.

The difference between a simple network and a complex network is the same between a sphere and a human being. You can fully describe the shape of a sphere with a few formulas: its surface is $4\pi r^2$, its volume is $\frac{4}{3}\pi r^3$. If you know r you know everything you need to know about the sphere. Try to fully describe the shape of a human being, internal organs included, starting from a single number. Go on, I have time.

What do simple networks look like? I think the easiest example conceivable is a square lattice. This is a regular grid, in which each node is connected to its four nearest neighbors. Such lattice can either span indefinitely (Figure 6.8(a)), or it can have a boundary (Figure 6.8(b)). Their fundamental properties are more or less the same. Knowing this connection rule that I just stated allows you to picture any lattice ever. That is why this is a simple topology.

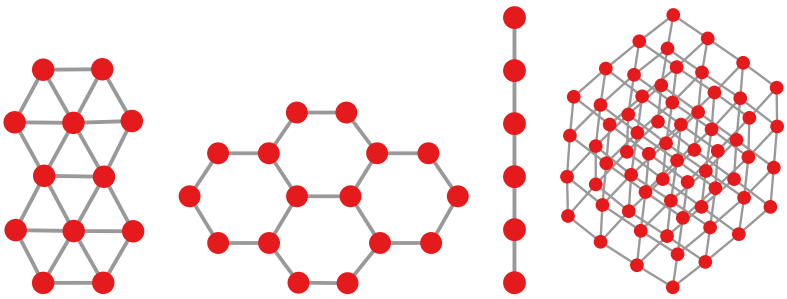
Regular lattices can come in many different shapes besides square, for instance triangular (Figure 6.9(a)) or hexagonal (Figure 6.9(b)).



(a) (b)

Figure 6.8: (a) An infinite lattice without boundaries. (b) A finite lattice with 25 nodes and 40 edges.

They also don't necessarily have to be two dimensional as the examples I made so far: you can have 1D (Figure 6.9(c)) and 3D (Figure 6.9(d)) lattices – the latter might be a bit hard to see, but it is a cube of with four nodes per side.



(a) (b) (c) (d)

Figure 6.9: Different lattice types. (a) Triangular. (b) Hexagonal. (c) One dimensional. (d) Three dimensional cube.

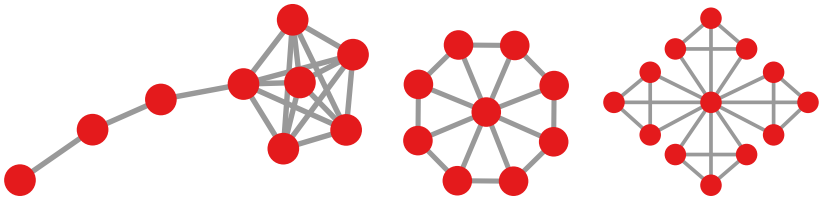
Even if deceptively simple, lattices can be extremely useful and are used as starting point for many advanced tasks. For instance, they are at the basis of the small-world graph generator (Section 17.2) and of our understanding of epidemic spread in society (Chapter 20).

Lattices are not the only simple network out there. There is a wide collection of other network types. These are usually developed as the simplest illustrative examples for explaining new problems or algorithms. A few of my favorites (yes, I'm the kind of person who has favorite graphs) are the lollipop graph[14] (a set of n nodes all connected to each other plus a path of m nodes shooting out of it, Figure 6.10(a)), the wheel graph (which has a center connected to a circle of m nodes, Figure 6.10(b)), and the windmill graph (a set of n graphs with m nodes and all connections to each other, also all connected to a central node, Figure 6.10(c)). Once you figure out what rule determines each topology, you can generate an arbitrary set of arbitrary size of graphs that all have the same properties.

[14] Graham Brightwell and Peter Winkler. Maximum hitting time for random walks on graphs. *Random Structures & Algorithms*, 1(3):263–276, 1990

Complex Networks

If simple networks were the only game in town, this book would not exist. That is because, as I said, you can easily understand all



(a) (b) (c)

Figure 6.10: Different simple networks. (a) Lollipop graph. (b) Wheel graph. (c) Windmill graph.

their properties from relatively simple math. That is not the case when the network you're analyzing is a complex network. Complex networks model complex systems: systems that cannot be fully understood if all you have is a perfect description of all their parts. The interactions between the parts let global properties emerge that are not the simple sum of local properties. There isn't a simple wiring rule and, even knowing all the wiring, some properties can still take you by surprise.

Personally, I like to divide complex networks into two further categories: complex network *with fundamental metadata* and *without fundamental metadata*. We saw that you can have edge metadata, the direction and weight. In Chapter 7 we'll see you can have even more, attached to both nodes and edges. The difference I'm trying to make is that, if the metadata are fundamental, they change the way you interpret some or all the metadata themselves.

To understand *non-fundamental* metadata, think about the fact that social networks, infrastructure networks, biological networks, and so on, model different systems and have different metadata attached to their nodes and edges. They can be age/gender, activation types, up- and down-regulation. However, the algorithms and the analyses you perform on them are the same, regardless of what the networks represent. They have nodes and edges and you treat them as such. You perform the Euler operation: you forget about all that is unnecessary so you can apply standardized analytic steps.

That is emphatically not true for networks with *fundamental* metadata. In that case, you need to be aware of what the metadata represent, because they change the way you perform the analysis and you interpret the results. A few examples:

- *Affiliation networks*. These are networks that, for instance, connect individuals to the groups they belong to. Here it is clear that one node type includes the other – the group includes the individual. This is fundamentally different when you have node types at an equal level – for instance if you connect people to the products they buy.
- *Interdependent networks*. These usually model some sort of physical

system, for instance computers connected to the power plants they control. Edges express the dependencies of one node on the other they connect to. In this case, the removal of one node in one layer has immediate and non-trivial repercussions on all the layers depending into it, often with catastrophic consequences (see Section 22.4) – which may not be true for other networks.

- *Correlation networks*. We saw a glimpse of these networks when we looked at weighted graphs. Here we have constraints on the edge weights, which can also be negative. The interpretation of such edge weights is different from what you would have in regular weighted networks. For instance, edges with very low weights are important here, because a strong negative correlation is interesting, even if its value (-1) is lower than no correlation at all (0).

A special mention for this class of networks should go to Bayesian networks[15,16,17]. In a Bayesian network, each node is a variable and directed edges represent dependencies between variables. If knowing something about the status of variable u gives you information about the status of variable v, then you will connect u to v with a directed (u, v) edge.

In the classical example, you might have three variables: the probability of raining, the probability of having the sprinklers on, and the probability that the grass is wet. Clearly, rain and sprinklers both might cause the grass to be wet, so the two variables point to them. Rain also might influence the sprinklers, because the automatic system to save water will not turn them on when it's raining, since it would be pointless. Obviously, the fact that the sprinklers are on will have no effect on whether it will rain or not.

We can model this system with the simple Bayesian network in Figure 6.11(a) and the corresponding conditional probability tables in Table 6.11(b). Bayesian networks are usually the output of a machine learning algorithm. The algorithm will learn the best network that fits the observations. Then, you can use the network to predict the most likely probability of the state of a variable given a new observation of a subset of variables.

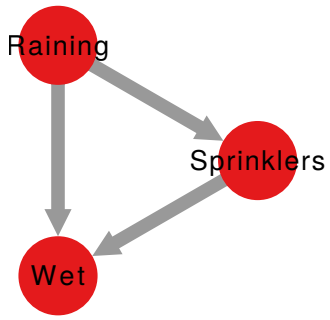
Simple examples like this might seem boring, but when you start having hundreds of variables you can find interesting patterns by applying some of the techniques you will learn later on. For instance, you might discover sets of variables that are independent of each other, even if, at first glance, it might be difficult to tell.

A not so distant relative of Bayesian networks are neural networks, the bread and butter of machine learning these days. Notwithstanding their amazing – and, sometimes, mysterious – power, neural networks are actually much more similar to simple networks than to

[15] Finn V Jensen et al. *An introduction to Bayesian networks*, volume 210. UCL press London, 1996

[16] Nir Friedman, Dan Geiger, and Moises Goldszmidt. Bayesian network classifiers. *Machine learning*, 29(2-3): 131–163, 1997

[17] Nir Friedman, Lise Getoor, Daphne Koller, and Avi Pfeffer. Learning probabilistic relational models. In *IJCAI*, volume 99, pages 1300–1309, 1999



(a)

	Rain	
	T	F
	0.20	0.80

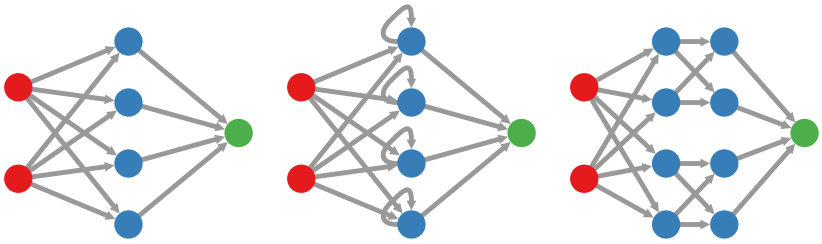
	Sprinkler	
Rain	T	F
T	0.01	0.99
F	0.20	0.80

		Wet	
Rain	Sprinkler	T	F
T	T	0.99	0.01
T	F	0.98	0.02
F	T	0.97	0.03
F	F	0.01	0.99

(b)

Figure 6.11: (a) A Bayesian network. (b) The conditional probability tables for the node states. The tables are referring to, from top to bottom: Rain, Sprinkler, Wet.

complex ones. Differently from Bayesian networks, the wiring rules of neural networks – of which I show some examples in Figure 6.12 – are usually rather easy to understand.



(a) Feedforward. (b) Recurrent. (c) Modular.

Figure 6.12: Different neural networks. The node color determines the layer type: input (red), hidden (blue), output (green).

The way they work is that the weight on each node of the output layer is the answer the model is giving. This weight is directly dependent on a combination of the weights of the nodes in the last hidden layer. The contribution of each hidden node is proportional to the weight of the edge connecting it to the output node. Recursively, the status of each node in the hidden layer is a combination of all its incoming connections – combining the edge weight to the node weight at the origin. The first hidden layer will be directly dependent on the weights of the nodes in the input layer, which are, in turn, determined by the data.

What the model does is simply finding the combination of edge weights causing the output layer's node weights to maximize the desired quality function.

6.5 Summary

1. The mathematical representation of a network is the graph: a collection of nodes – the actors of the network –, and edges – the connections among those actors. In a simple graph, no additional feature can be added, and there is only one edge between a pair of nodes.

2. If connections are not symmetric, meaning that if you consider me your friend I don't necessarily consider you mine, then we have directed graphs. In directed graphs, edges have a direction so relations flow one way, unless there is a reciprocal edge pointing back.

3. In weighted graphs, connections can be more or less strong, indicated by the weight of the edge, a numerical quantity. It doesn't have to be a discrete number, nor necessarily positive: for instance in correlation networks you can have negative continuous weights.

4. Weights can have two meanings: proximity – the edge is the strength of a friendship –, or distance – the edge is a cost to pay to cross from one node to another. Different semantics imply that some algorithms' results should be interpreted differently.

5. Simple networks are networks whose topology can be fully described with simple rules. For instance, in regular lattices you place nodes uniformly in a space and you connect them with their nearest neighbors.

6.6 Exercises

1. Calculate $|V|$ and $|E|$ for the graph in Figure 6.1(c).

2. Mr. A considers Ms. B a friend, but she doesn't like him back. She has a reciprocal friendship with both C and D, but only C considers D a friend. D has also sent friend requests to E, F, G, and H but, so far, only G replied. G also has a reciprocal relationship with A. Draw the corresponding directed graph.

3. Draw the previous graph as undirected and weighted, with the weight being 2 if the connection is reciprocal, 1 otherwise.

4. Draw a correlation network for the vectors in `http://www.networkatlas.eu/exercises/6/4/data.txt`, by only drawing edges with positive weights, ignoring self loops.

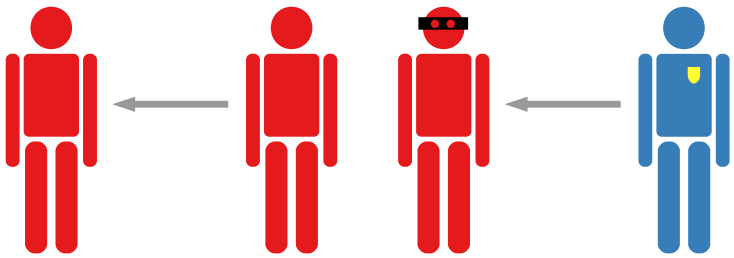
7
Extended Graphs

The world of simple graphs is... well... simple. The only thing complicating it a bit so far was adding some information on the edges: whether they are asymmetric – meaning $(u, v) \neq (v, u)$ – and whether they are strong or weak. Unfortunately, that's not enough to deal with everything reality can throw your way. In this chapter, I present even more graph models, which go beyond the simple addition of edge information.

7.1 Bipartite Graphs

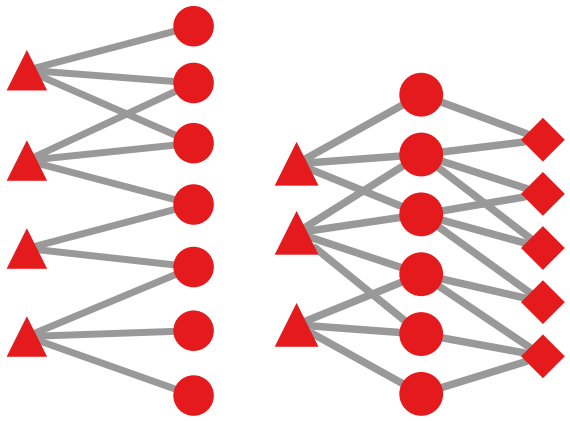
So far we have talked about networks in which relations run between peers: nodes are all the same to us. But nodes might belong to two distinct classes. And connections can only be established between members of different classes. Figure 7.1 provides an example. In a social network without node attributes nor types, anybody can be friend with anybody else and there isn't much to distinguish two nodes. But if we want to connect cops with the thieves they catch, then we are establishing additional connecting rules. Thieves don't catch each other. And, hopefully, cops aren't thieves. Another example could be connecting workers to the buildings hosting their offices.

Stripping down the model to a minimum, bipartite networks are



(a) (b)

Figure 7.1: (a) A simple graph representing a social network with no additional constraints. (b) A cop-thief bipartite network: nodes can be either a cop or a thief, and cops can only catch (connect to) thieves.



(a) (b)

Figure 7.2: (a) An example of a bipartite network. (b) An example of a tripartite network.

networks in which nodes must be part of either of two classes (V_1 and V_2) and edges can only be established between nodes of unlike type[1,2]. Formally, we would say that $G = (V_1, V_2, E)$, and that E can only contain edges like (v_1, v_2), with $v_1 \in V_1$ and $v_2 \in V_2$. Figure 7.2(a) depicts an example.

Bipartite networks are used for countless things, connecting: countries to the products they export[3], hosts to guest in symbiotic relationships[4], users to the social media items they tag[5], bank-firm relationships in financial networks[6], players-bands in jazz[7], listener-band in music consumption[8], plant-pollinators in ecosystems[9], and more. You get the idea. Bipartite networks pop up everywhere.

However, by a curious twist of fate, the algorithms able to work directly on bipartite structures are less studied than their non-bipartite counterparts. For instance, for every community discovery algorithm that works on bipartite networks you have a hundred working on non-bipartite ones. The distinction is important, because the standard assumptions of non-bipartite community discovery do not hold in bipartite networks, as we will see in Part X.

Why would that be the case? Because practically everyone who works on bipartite networks projects them. Most of the times, you are interested only in one of the two node types. So you create a unipartite version of the network connecting all nodes in V_1 to each other, using some criteria to make the V_2 count. The trivial way is to connect all V_1 nodes with at least a common V_2 neighbor. This is so widely done and so wrong that I like to call it the Mercator bipartite projection, in honor of the most used and misunderstood map projection of all times. We'll see in Chapter 26 why that's not very smart, and the different ways to do a better job.

Why stopping at bipartite? Why not go full n-partite? For instance, a paper I cited before actually builds a tri-partite network (Figure

[1] Armen S Asratian, Tristan MJ Denley, and Roland Häggkvist. *Bipartite graphs and their applications*, volume 131. Cambridge University Press, 1998

[2] Jean-Loup Guillaume and Matthieu Latapy. Bipartite structure of all complex networks. *Information processing letters*, 90:Issue–5, 2004

[3] César A Hidalgo and Ricardo Hausmann. The building blocks of economic complexity. *Proceedings of the national academy of sciences*, 106(26):10570–10575, 2009

[4] Brian D Muegge, Justin Kuczynski, Dan Knights, Jose C Clemente, Antonio González, Luigi Fontana, Bernard Henrissat, Rob Knight, and Jeffrey I Gordon. Diet drives convergence in gut microbiome functions across mammalian phylogeny and within humans. *Science*, 332(6032):970–974, 2011

[5] Renaud Lambiotte and Marcel Ausloos. Collaborative tagging as a tripartite network. In *International Conference on Computational Science*, pages 1114–1117. Springer, 2006

[6] Luca Marotta, Salvatore Micciche, Yoshi Fujiwara, Hiroshi Iyetomi, Hideaki Aoyama, Mauro Gallegati, and Rosario N Mantegna. Bank-firm credit network in japan: an analysis of a bipartite network. *PloS one*, 10(5): e0123079, 2015

[7] Pablo M Gleiser and Leon Danon. Community structure in jazz. *Advances in complex systems*, 6(04):565–573, 2003

[8] Renaud Lambiotte and Marcel Ausloos. Uncovering collective listening habits and music genres in bipartite networks. *Physical Review E*, 72(6):066107, 2005

7.2(b) depicts an example): users connect to the social media they tag and with the tags they use. However, the gains you get from a more precise data structure quickly become much lower than the added complexity of the model. Even tripartite networks are a rarity in network science. A couple of examples are the recipe-ingredient-compound structure of the flavor network[10], or the aid organization-country-issue structure[11].

7.2 *Multilayer Graphs*

In this section I describe models you can use to analyze multilayer networks. This should not be confused with the similarly-sounding, but actually completely different, multilevel network analysis[12]. This is a whole different way to analyze social network data using multilevel analysis, of which I know little and I will attempt to cover in future versions of this book.

One-to-One

Traditionally, network scientists try to focus on one thing at a time. If they are interested in analyzing your friendship patterns, they will choose one network that closely approximates your actual social relations and they will study that. For instance, they will download a sample of the Facebook graph. Or they will analyze tweets and retweets.

However, in some cases, that is not enough to really grasp the phenomenon one wants to study. If you want to predict a new connection on Facebook, something happening in another social media might have influenced it. Two people might have started working in the same company and thus first connected on Linkedin, and then became friends and connected on Facebook. Such scenario could not be captured by simply looking at one of the two networks. Network scientists invented multilayer networks[13,14,15,16,17,18] to answer this kind of questions.

There are two ways to represent multilayer networks. The simpler is to use a multigraph. Remember Euler's parallel edges in the Königsberg graph from Figure 6.2? That's what makes a multigraph. Differently from a simple graph (Figure 7.3(a)), in which every pair of nodes is forced to have at most one edge connecting them, in a multigraph (Figure 7.3(b)) we allow an arbitrary number of possible connections.

If that is all, there wouldn't be much difference between multigraphs and weighted networks. If all parallel edges are the same, we could have a single edge with a weight proportional to the number

[9] Colin Campbell, Suann Yang, Réka Albert, and Katriona Shea. A network model for plant–pollinator community assembly. *Proceedings of the National Academy of Sciences*, 108(1):197–202, 2011

[10] Yong-Yeol Ahn, Sebastian E Ahnert, James P Bagrow, and Albert-László Barabási. Flavor network and the principles of food pairing. *Scientific reports*, 1:196, 2011

[11] Michele Coscia, Ricardo Hausmann, and César A Hidalgo. The structure and dynamics of international development assistance. *Journal of Globalization and Development*, 3(2):1–42, 2013a

[12] Emmanuel Lazega and Tom AB Snijders. *Multilevel network analysis for the social sciences: Theory, methods and applications*, volume 12. Springer, 2015

[13] Mikko Kivelä, Alex Arenas, Marc Barthelemy, James P Gleeson, Yamir Moreno, and Mason A Porter. Multilayer networks. *Journal of complex networks*, 2 (3):203–271, 2014

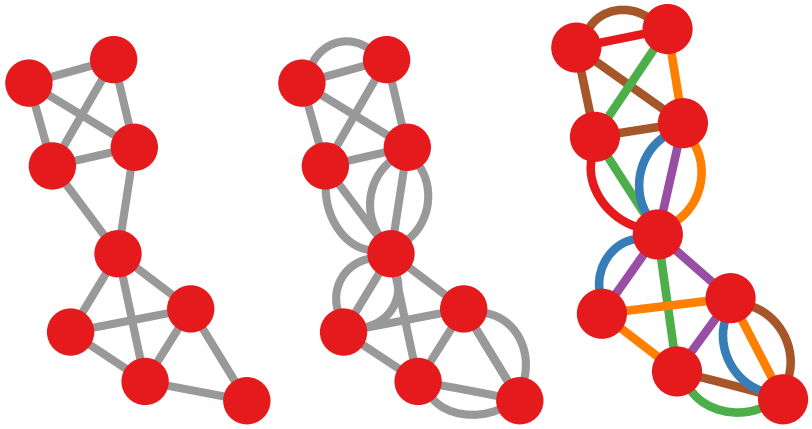
[14] Manlio De Domenico, Albert Solé-Ribalta, Emanuele Cozzo, Mikko Kivelä, Yamir Moreno, Mason A Porter, Sergio Gómez, and Alex Arenas. Mathematical formulation of multilayer networks. *Physical Review X*, 3(4):041022, 2013

[15] Stefano Boccaletti, Ginestra Bianconi, Regino Criado, Charo I Del Genio, Jesús Gómez-Gardenes, Miguel Romance, Irene Sendina-Nadal, Zhen Wang, and Massimiliano Zanin. The structure and dynamics of multilayer networks. *Physics Reports*, 544(1):1–122, 2014

[16] Michele Berlingerio, Michele Coscia, Fosca Giannotti, Anna Monreale, and Dino Pedreschi. Multidimensional networks: foundations of structural analysis. *WWW*, 16(5-6):567–593, 2013a

[17] Mark E Dickison, Matteo Magnani, and Luca Rossi. *Multilayer social networks*. Cambridge University Press, 2016

[18] Matteo Magnani and Luca Rossi. The ml-model for multi-layer social networks. In *ASONAM*, pages 5–12. IEEE, 2011



(a) (b) (c)

Figure 7.3: (a) A simple graph. (b) A multigraph, with multiple edges between the same node pairs. (c) A multilayer network, where each edge has a type (represented by its color).

of connections between the two nodes. However, in this case, we can add a "type" to each connection, making them *qualitatively* different: one edge type for Facebook, one for Twitter, one for Linkedin (Figure 7.3(c)), etc.

In practice, every edge type – or label – represents a different layer of the network. A pair of nodes can establish a connection in any layer, even at the same time. Each layer is a simple graph. In this book – and generally in computer science – the most used notation to indicate a multilayer network is $G = (V, E, L)$. V and E are the sets of nodes and edges, as usual. L is the set of layers – or labels. An edge is now a triple $(u, v, l) \in E$, with $u, v \in V$ as nodes, and $l \in L$ as the layer. This might seem similar to the notation used for weighted edges – which was (u, v, w). The key difference is that w is a quantitative information, while l is a qualitative one: a class, a type. We can make the two co-exist in weighted multigraphs, by specifying an edge as (u, v, l, w).

The model that we introduce in Figure 7.3(c) is but the simplest way to represent multilayer networks. This strategy rests on the assumption that there is a one-to-one node mapping between the layers of the network. In other words, the entities in each layer are always the same: you are always you, whether you manage your Facebook account or your Linkedin one. Such simplified multilayer networks are sometimes called multiplex networks.

Studies have shown how layers in a multiplex network could be complementary[19]. This means that a single layer in the network might not show the typical statistical properties you would expect from a real world network – the types of things we'll see in this book. However, once you stack enough layers one on top of the other, the resulting network does indeed conform to our structural expectations.

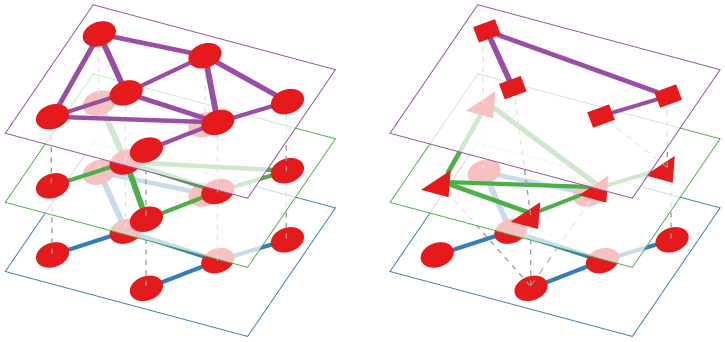
[19] Alessio Cardillo, Jesús Gómez-Gardenes, Massimiliano Zanin, Miguel Romance, David Papo, Francisco Del Pozo, and Stefano Boccaletti. Emergence of network features from multiplexity. *Scientific reports*, 3:1344, 2013

In other words, multilayer networks have *emerging* properties.

Multiplex networks, don't necessarily cover all application scenarios: sometimes a node in one layer can map to multiple nodes – or none! – in another. This is what we turn our attention to next.

Many-to-Many

To fix the insufficient power of multiplex networks to represent true multilayer systems we need to extend the model. We introduce the concept of "interlayer coupling". In this scenario, the node is split into the different layers to which it belongs. In this case, your identity includes multiple personas: you are the union of the "Facebook you", the "Linkedin you", the "Twitter you". Figure 7.4(a) shows the visual representation of this model: each layer is a slice of the network. There are two types of edges: the intra-layer connections – the traditional type: we're friends on Facebook, Linkedin, Twitter –, and the inter-layer connections. The inter-layer edges run between layers, and their function is to establish that the two nodes in the different layers are really the same node: they are *coupled* to – or *dependent* on – each other.



(a) (b)

Figure 7.4: The extended multilayer model. Each slice represents a different layer of the network. Dashed grey lines represent the inter-layer coupling connections. (a) A multilayer network with trivial one-to-one coupling. (b) A multilayer network with complex interlayer coupling.

Formally, our network is $G = (V, E, L, C)$. V is still the set of nodes, but now we split the set of edges in two: E is the set of classical edges, the intra-layer one – connections between different people on a platform –; and C is the set of coupling connections, the inter-layer one, denoting dependencies between nodes in different layers.

Having a full set of coupling connections enables an additional degree of freedom. We can now have nodes in one layer expressing coupling with multiple nodes in other layers. In our social media case, we are now allowing you to have multiple profiles in one platform that still map on your single profile in another. For instance, you can run as many different Twitter accounts as you want, and they are still coupled with your Facebook account. To get a visual sense on what this means, you can look at Figure 7.4(b).

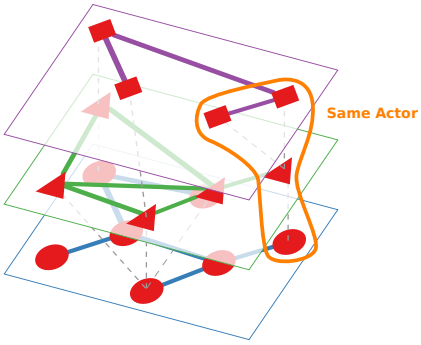
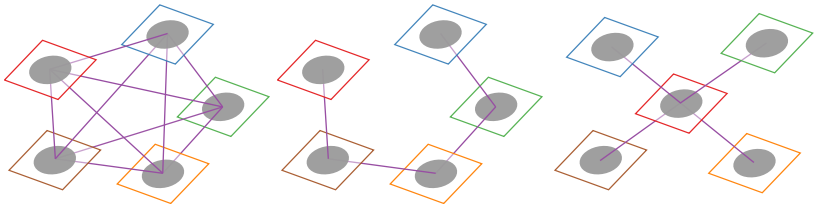


Figure 7.5: An actor in a many-to-many coupled multilayer network. The orange outline surrounds nodes with coupling edges connecting them.

This new freedom comes to a cost. While in the one-to-one mapping it is easy to identify a node among layers, because all identities of a node are concentrated in a single point in a layer, in the many-to-many coupling that is not true any more. So we introduce the term "actor", which is the entity behind all the multiple identities across layers and within a layer. In practice, the actor is a connected component (see Section 10.4), when only considering inter-layer couplings as the possible edges. If my three Twitter profiles all refer to the same person, with maybe two Flickr accounts and one Facebook profile, all these identities belong to the same actor: me. Figure 7.5 should clarify this definition.

Note that there can be many ways to establish inter-layer couplings between the different nodes belonging to the same actor. As far as I know, when analyzing networks people usually use a "cliquey" approach: every node belonging to the same actor is connected to every other node as, for instance, in Figure 7.6(a). This effectively creates a clique of inter-layer coupling connections – for more information about what a clique is, see Section 12.3.



(a) Clique (b) Chain (c) Star

Figure 7.6: Different coupling flavors for your multilayer networks. Showing a network with a single actor and a single node per actor per layer (represented by the border-colored polygon). I color the coupling edges in purple.

However, this is usually too cumbersome to draw. So, for illustration purposes, the convention is to use a "chainy" approach (Figure 7.6(b)): you sort your layers somehow, and you simply place a line representing your coupling connections piercing through the layers. We don't really have to stop there. One could imagine using a "starry" approach: defining one layer as the center of the system, and connecting all nodes belonging to that actor to the node in the cen-

tral layer. To see what I mean, look at Figure 7.6(c). Using different coupling flavors can be useful for computational efficiency: when you start having dozens or even hundreds of layers, creating cliques of layers can add a significant overhead.

Such many-to-many layer couplings are often referred to in the literature as "networks of networks", because each layer can be seen as a distinct network, and the interlayer couplings are relationships between different networks[20,21,22].

[20] Jacopo Iacovacci, Zhihao Wu, and Ginestra Bianconi. Mesoscopic structures reveal the network between the layers of multiplex data sets. *Physical Review E*, 92 (4):042806, 2015

[21] Gregorio D'Agostino and Antonio Scala. *Networks of networks: the last frontier of complexity*, volume 340. Springer, 2014

[22] Dror Y Kenett, Matjaž Perc, and Stefano Boccaletti. Networks of networks–an introduction. *Chaos, Solitons & Fractals*, 80:1–6, 2015

Aspects

Do you think we can't make this even more complicated? Think again. These aren't called "complex networks" by accident. To fully generalize multilayer networks, adding the many-to-many interlayer coupling edges is not enough. To see why that's the case, consider the fact that, up to this point, I considered the layers in a multilayer network as interchangeable. Sure, they represent different relationships – Facebook friendship rather than Twitter following – but they are fundamentally of the same type. That's not necessarily the case: the network can have multiple aspects.

For instance, consider time. We might not be Facebook friends now, but that might change in the future. So we can have our multilayer network at time t and at time $t + 1$. These are two aspects of the same network. All the layers are present in both aspects and the edges inside them change. Another classical example is a scientific community. People at a conference interact in different ways – by attending each other talks, by chatting, or exchanging business cards – and can do all of those things at different conferences. The type of interaction is one aspect of the network, the conference in which it happens is another.

I can't hope to give you here an overview of how many new things this introduces to graph theory. So I'm referring you to a specialized book on the subject[23].

[23] Ginestra Bianconi. *Multilayer Networks: Structure and Function*. Oxford University Press, 2018

Signed Networks

Signed networks are a particular case of multilayer networks. Suppose you want to buy a computer, and you go online to read some reviews. Suppose that you do this often, so you can recognize the reviewers from past reviews you read from them. This means that you might realize you do not trust some of them and you trust others. This information is embedded in the edges of a signed network: there are positive and negative relationships.

Signed networks are not necessarily restricted to either a single positive or a single negative relationship – e.g. "I trust this person"

or "I don't trust this person". For instance, in an online game, you can have multiple positive relationships like being friend or trading together; and multiple reasons to have a negative relationship, like fighting each other, or putting a bounty on each other heads.

A key concept in signed networks is the one of structural balance. Since this is mostly related to the link prediction problem, I expand on this in Section 24.1.

Positive and negative relationships have different dynamics. For instance, in a seminal study looking at interactions between players in a massively multiplayer online game[24], the authors studied the different degree distributions (Section 9.2) for each type of relationship. They uncovered that positive relationships have a marked exponential cutoff, while negative relationships don't. You'll become more accustomed to what a degree distribution is and all the lingo related to it in Chapter 9. For now, the meaning of what I just said is: there is a limit to the number of people you can be friends with, but there is no limit to the number of people that can be mad at you.

[24] Michael Szell, Renaud Lambiotte, and Stefan Thurner. Multirelational organization of large-scale social networks in an online world. *Proceedings of the National Academy of Sciences*, 107 (31):13636–13641, 2010

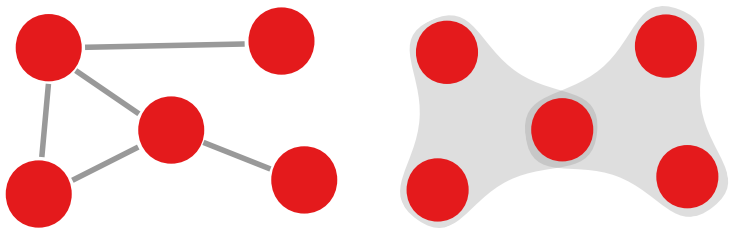
7.3 *Many-to-Many Relationships*

Hypergraphs

In the classical definition, an edge connects two nodes – the gray lines in Figure 7.7(a). Your friendship relation involves you and your friend. If you have a second friend, that is a different relationship. There are some cases in which connections bind together multiple people at the same time. For instance, consider team building: when you do your final project with some of your classmates, the same relationship connects you with all of them. When we allow the same edge to connect more than two nodes we call it a *hyperedge* – the gray area in Figure 7.7(b). A collection of hyperedges makes a *hypergraph*[25,26].

[25] Vitaly Ivanovich Voloshin. *Introduction to graph and hypergraph theory*. Nova Science Publishers Hauppauge, 2009

[26] Alain Bretto. Hypergraph theory: An introduction. *Mathematical Engineering. Cham: Springer*, 2013



(a) (b)

Figure 7.7: (a) Classical Graph. (b) Hypergraph.

To make them more manageable, we can put constraints to hyperedges. We could force them to always contain the same number of nodes. In a soccer tournament, the hyperedge representing a team can only have eleven members: not one more nor one less, be-

cause that's the number of players in the team. In this case, we call the resulting structure a "uniform hypergraph", and have all sorts of interesting properties[27]. In general, when simply talking about hypergraphs we have no such constraint.

It is difficult to work with hypergraphs[28]. Specialized algorithms to analyze them exist, but they become complicated very soon. In the vast majority of cases, we will transform hyperedges into simpler network forms and then apply the corresponding simpler algorithms.

There are two main strategies to simplify hypergraphs. The first is to transform the hyperedge into the simple edges it stands for. If the hyperedge connects three nodes, we can change it into a unipartite network in which all three nodes are connected to each other. In the project team example, the new edges simply represent the fact that the two people are part of the same team. The advantage is a gain in simplicity, the disadvantage is that we lose the ability to know the full team composition by looking at its corresponding hyperedge: we need to explore the newly created structures.

The second strategy is to turn the hypergraph into a bipartite network. Each hyperedge is converted into a node of type 1, and the hypergraph nodes are converted into nodes of type 2. If nodes are connected by the same hyperedge, they all connect to the corresponding node of type 1. In the project team example, the nodes of type 1 represent the teams, and the nodes of type 2 the students. This is an advantageous representation: it is simpler than the hypergraph, but it preserves some of its abilities, for instance being able to reconstruct teams by looking at the neighbors of the nodes of type 1. However, the disadvantage with respect to the previous strategy is that there are fewer algorithms working for bipartite networks than with unipartite networks.

Figure 7.8 provides a simple example on how to perform these two

[27] Shenglong Hu and Liqun Qi. Algebraic connectivity of an even uniform hypergraph. *Journal of Combinatorial Optimization*, 24(4):564–579, 2012

[28] Source: I tried once.

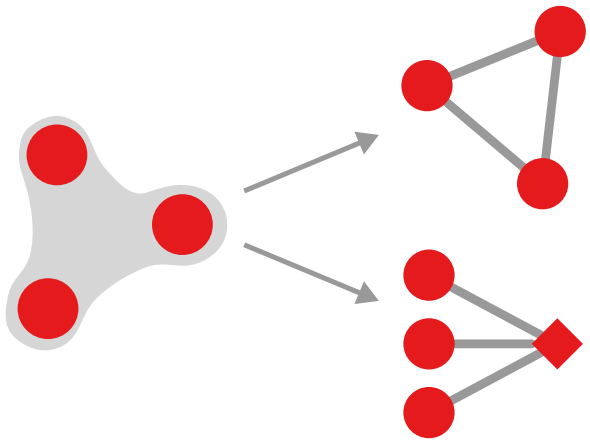


Figure 7.8: The two ways to convert a hyperedge into simpler forms. A hyperedge connecting three nodes can become a triangle (top right), or a bipartite network (bottom right).

conversion strategies on a simple hyperedge connecting three nodes.

When it comes to notation, the network is still represented by the classical node and edge sets: $G = (V, E)$. However, the E set now is special: its elements are not forced to be tuples any more. They can be triples, quartuplets, and so on. For instance, (u, v, z) is a legal element that can be in E, with $u, v, z \in V$.

Simplicial Complexes

Simplicial complexes[29,30,31] are related to hypergraphs. A simplicial complex is a graph containing simplices. Simplicial complexes are like hyperedges in that they connect multiple nodes, but they have a strong emphasis on geometry. Graphically, we normally represent simplicial complexes as fills in between the sides that compose the simplex – as I show in Figure 7.9.

[29] Vsevolod Salnikov, Daniele Cassese, and Renaud Lambiotte. Simplicial complexes and complex systems. *European Journal of Physics*, 40(1):014001, 2018

[30] Jakob Jonsson. *Simplicial complexes of graphs*, volume 3. Springer, 2008

[31] Ginestra Bianconi. *Higher-order networks*. Cambridge University Press, 2021

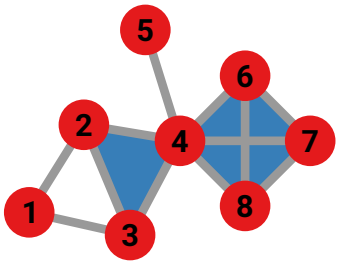


Figure 7.9: An example of simplicial complex. The blue shades represent the two simplices in the complex.

You can think of simplices as hyperedges on steroids. While a hyperedge including, say, four nodes is "just" a group of four nodes all interacting with each other, a simplex of four logically also contains all lower-level simplices, which are taken into account in the analysis. Notation-wise, a node is a 0-simplex, an edge is a 1-simplex, then a 2-simplex connects three nodes, and you can see where this is going. A simplex connecting $n + 1$ nodes is a n-simplex. Figure 7.10 shows you the first entries of the simplicial complex zoo.

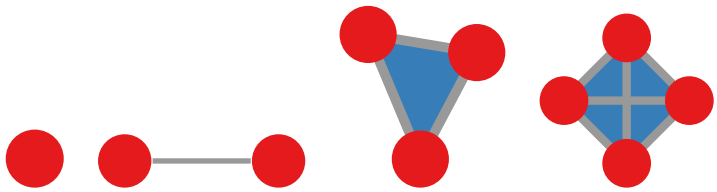


Figure 7.10: The smallest simplices a simplicial complex can have. (a) 0-simplex, (b) 1-simplex, (c) 2-simplex, (d) 3-simplex.

What does it mean for a simplicial to contain all its lower level versions? Consider the 3-simplex in Figure 7.10(d). That simplex contains four 0-simplices – the nodes –, six 1-simplices – the edges –, and four 2-simplices. These are called the *faces* of the simplex – think of them as the faces of a solid in geometry, because that's

what they are. On the other hand, a hyperedge with four nodes only contains those four nodes, it does not logically contain any three-node hyperedge – unless that specific three-node hyperedge is explicitly coded as part of the data, but it is a wholly separate entity. In the paper writing example, four people writing a paper make a four-node hyperedge, but only a *different* paper with three of those authors will generate a hyperedge contained in it – while a 3-simplex will naturally contain all 2-simplices with no extra paper.

A simplicial complex – a network with simplices – has a *dimension*: the largest dimension of the simplices it contains. The simplicial complex in Figure 7.9 has dimension 3. A *facet* of a simplicial complex is a simplex that is not a face of any other larger simplex. The facets in the simplicial complex of Figure 7.9 are: $[\{1,2\},\{1,3\},\{2,3,4\},\{4,5\},\{4,6,7,8\}]$. The sequence of facets of a simplicial complex fully distinguishes it from any other simplicial complex. Just like with uniform hypergraphs, we can also have pure simplicial complexes, which are complexes that contain only facets of the same dimension. Figure 7.9 is not pure because it has facets of dimension three, two and one. Figure 7.10(d) is a 3-pure simplicial complex, because it only contains a simplex of dimension three. If you were to ignore all simplices and analyze the network without them, you'd be working with the *skeleton* of the simplicial complex. In practice, any network is a skeleton of one or more simplicial complexes.

Simplicial complexes are one of the main ways to analyze high-order interactions in networks, and so we're going to look at them extensively in Chapter 34.

7.4 Dynamic Graphs

Most networks are not crystallized in time. Relationships evolve: they are created, destroyed, modified over time by all parties involved. Every time we use a network without temporal information on its edges, we are looking at a particular slice of it, that may or may not exist any longer.

For many tasks, this is ok. For others, the temporal information is a key element. Imagine that your network represents a road graph. Nodes are intersections, and edges are stretches of the street connecting them. Roadworks might cut off a segment for a few days. If your network model cannot take this into account, you would end up telling drivers to use a road that is blocked, creating traffic jams and a lot of discomfort. That is why you need dynamic – or temporal – networks[32,33,34,35,36].

Consider Figures 7.11(a) to (d) as an example. Here, we have a

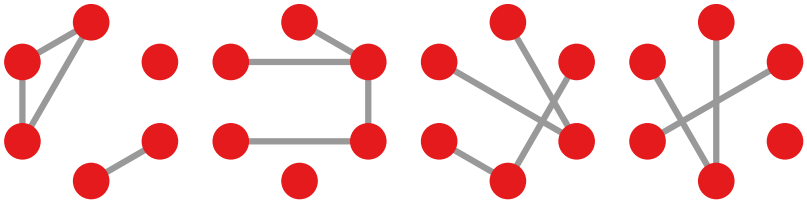
[32] Soon-Hyung Yook, Hawoong Jeong, A-L Barabási, and Yuhai Tu. Weighted evolving networks. *Physical review letters*, 86(25):5835, 2001

[33] Alain Barrat, Marc Barthélemy, and Alessandro Vespignani. Weighted evolving networks: coupling topology and weight dynamics. *Physical review letters*, 92(22):228701, 2004b

[34] Petter Holme and Jari Saramäki. Temporal networks. *Physics reports*, 519 (3):97–125, 2012

[35] Vincenzo Nicosia, John Tang, Cecilia Mascolo, Mirco Musolesi, Giovanni Russo, and Vito Latora. Graph metrics for temporal networks. In *Temporal networks*, pages 15–40. Springer, 2013

[36] Naoki Masuda and Renaud Lambiotte. *A Guidance to Temporal Networks*. World Scientific, 2016



(a) $t = 1$ (b) $t = 2$ (c) $t = 3$ (d) $t = 4$

Figure 7.11: An example of dynamic network. Each figure represents the same network, observed at different points in time.

social network. People are connected only when they are actually interacting with each other. We have four observations, taken at four different time intervals. Suppose that you want to infer if these people are part of the same social group – or community. Do they? Looking at each single observation would lead us to say *no*. In each time step there are individuals that have no relationships to the rest of the group. Adding the observations together, though, would create a structure in which all nodes are connected to each other. Taking into account the dynamic information allows us to make the correct inference. Yes, these nodes form a tightly connected group.

In practice, we can consider a dynamic network as a network with edge attributes. The attribute tells us when the edge is active – or inactive, if the connection is considered to be "on" by default, like the road graph. Figure 7.12 shows a basic example of this, with edges between three nodes.

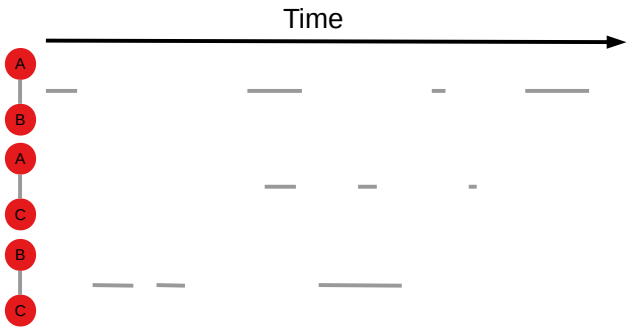
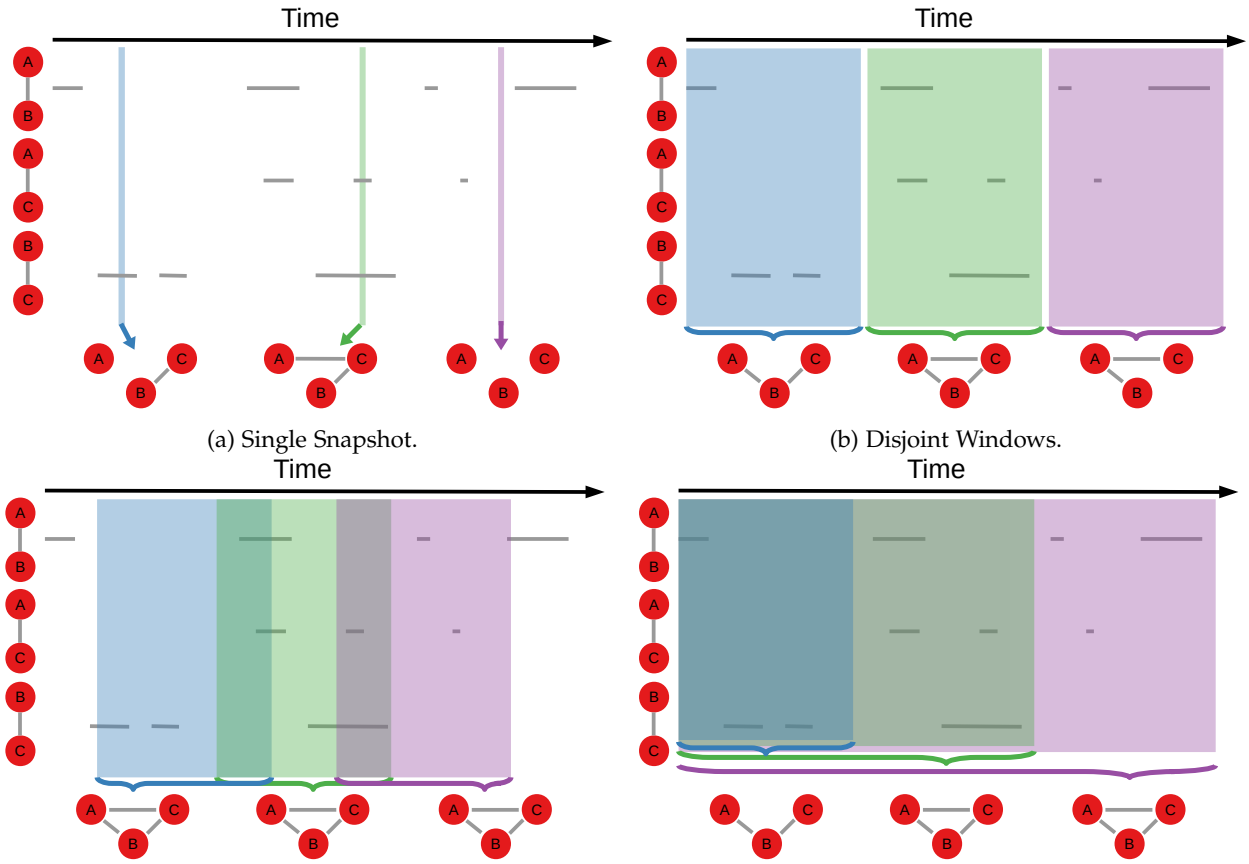


Figure 7.12: An example of dynamic edge information. Time flows from left to right. Each row represents a possible potential edge between nodes A, B, and C. The moments in time in which each edge is active are represented by gray bars.

More formally, our graph can be represented as $G = (G_1, G_2, ..., G_n)$, where each G_i is the i-th snapshot of the graph. In other words, $G_i = (V_i, E_i)$, with V_i and E_i being the set of nodes and edges active at time i.

How do we deal with this dynamic information when we want to create a static view of the network? There are a four standard techniques.

- *Single Snapshot* – Figure 7.13(a). This is the simplest technique. You choose a moment in time and your graph is simply the collection of nodes and edges active at that precise instant. This strategy works well when the edges in your network are "on" by default.



(a) Single Snapshot.

(b) Disjoint Windows.

(c) Sliding Windows.

(d) Cumulative Windows.

Figure 7.13: Different strategies for converting dynamic edges into a graph view.

It risks creating an empty network when edges are ephemeral and/or there are long lulls in the connection patterns, for instance in telecommunication networks at night.

- *Disjoint Windows* – Figure 7.13(b). Similar to single snapshot. Here we allow longer periods of time to accumulate information. Differently from the previous technique, no information is discarded: when a window ends, the next one begins immediately. Works well when it's not important to maintain continuity.

- *Sliding Windows* – Figure 7.13(c). Similar to disjoint windows, with the difference that we allow the observation periods to overlap. That is, the next window starts before the previous one ended. Works well when it is important to maintain continuity.

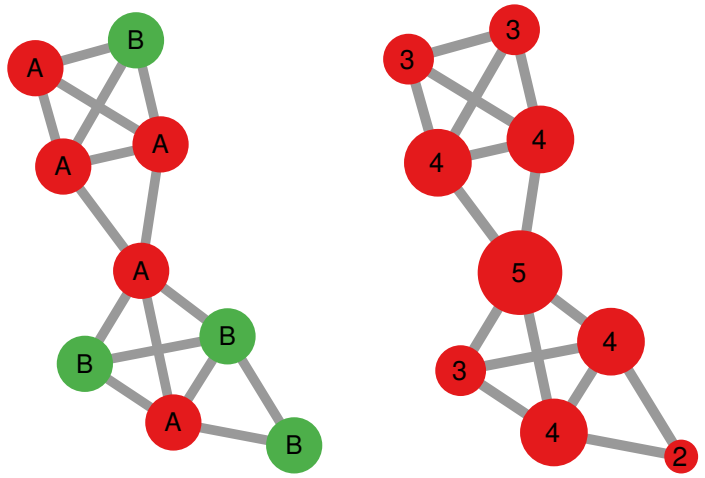
- *Cumulative Windows* – Figure 7.13(d). Similar to sliding windows, but here we fix the beginning of each window at the beginning of the observation period. Information can only accumulate: we never discard edge information, no matter how long ago it was firstly generated. Each window includes the information of all

previous windows. Works well when the effect of an edge never expires, even after the edge has not been active for a long time.

Note how these different techniques generate radically different "histories" for the network in Figure 7.13(a) to (d), even when the edge activation times are identical.

7.5 *Attributes on Nodes*

Earlier I defined what a bipartite network is: a network with two node types and edges connecting exclusively nodes of unlike type. You could consider the node type as a sort of binary attribute on the node. Once you make the step of adding some metadata to the nodes, why stopping at just two values? And why constraining how edges can connect nodes depending on their attributes? Welcome to the world of node attributes!



(a) (b)

Figure 7.14: (a) A network with qualitative node attributes, represented by node labels and colors. (b) A network with quantitative node attributes, represented by node labels and sizes.

Here we do not have the requirement of only establishing edges between nodes with unlike attribute values. Moreover the attributes don't have to be binary. They also don't have to be qualitative at all (as in Figure 7.14(a)): they can be quantitative, as in Figure 7.14(b). For instance, the number of times a user logged into their social media profile. Finally, nodes can have an arbitrary number of attributes attached to them, not just one.

Consider for instance a trade network. The nodes in this network are the various countries. They connect together if one country exports goods to another. We can have multiple quantitative attributes on each country. For instance, it can be its GDP per capita, its population, its total trade volume. On the other hand, we can also put countries in different categories: in which world region are they lo-

cated? Are they democracies or not? Of which trade agreement are they part of?

In this case, our graph changes form again: $G = (V, E, A)$. We can see each $v \in V$ not as a simple entity, but as a vector of attribute values: $v = (a_1, a_2, a_3, ...)$. In this representation, a_1 is the value for v of the first attribute in A. a_1 can be a real, integer, or a category.

Node attributes are important because nodes might have tendencies of connecting – or refusing to connect – to nodes with similar attribute values. We'll explore this topic in the forms of "homophily" in Chapter 30 for qualitative attributes, and "assortativity" in Chapter 31 for quantitative attributes. This is different from bipartite networks because in bipartite networks edges between nodes with the same attribute value are *forbidden*, while in these cases edges are simply *correlated* with attribute values. Moreover, bipartite networks are only defined for qualitative attributes, not quantitative.

To wrap up, no one forces you to use a single of these more complex graph models at a time. You can merge them together to fit your analytical needs. For instance, you can create this monster graph type: $G_n = (V_1, V_2, E, L, W, A)$: a bipartite graph with V_1 and V_2 nodes, each with attributes in A, which is weighted (W) multilayer with $|L|$ layers and – for good measure – is also a hypergraph, allowing edges in E with more than two nodes. And, of course, you can observe it at multiple time intervals (G_1, G_2, ...). Yikes.

7.6 Summary

1. Bipartite networks are networks with two node types. Edges can only connect two nodes of different types. You can generalize them to be n-partite, and have n node types.

2. In multigraphs we allow to have multiple (parallel) edges between nodes. We can have labeled multigraphs when we attach labels to nodes and edges. Labels on nodes can be qualitative or quantitative attributes.

3. If we only allow one edge with a given label between nodes we have a multiplex or multilayer network: the edge label informs us about the layer in which the edge appears.

4. Multilayer networks are networks in which different nodes can connect in different ways. To track which node is "the same" across layers we use inter-layer couplings. Couplings can connect a node in a layer to multiple nodes in another, making a many-to-many correspondence.

5. Signed networks are a special type of multilayer network with two layers: one positive (e.g. friendship) and one negative (e.g. enmity).

6. Hypergraphs are graphs whose (hyper)edges can connect more than two nodes at the same time. You can consider hyperedges as cliques or bipartite edges.

7. Simplicial complexes, like hypergraphs, allow nodes to connect in many-to-many relationships called simplices. Simplices are more powerful than hyperedges because a simplex of 4 nodes logically contain all of its smaller simplices – called "faces".

8. Dynamic graphs are graphs containing temporal information on nodes and edges. This information tells you when the node/edge was present in the network. There are many ways to aggregate this information to create snapshots of your evolving system.

7.7 Exercises

1. The network in `http://www.networkatlas.eu/exercises/7/1/data.txt` is bipartite. Identify the nodes in either type and find the nodes, in either type, with the most neighbors.

2. The network in `http://www.networkatlas.eu/exercises/7/2/data.txt` is multilayer. The data has three columns: source and target node, and edge type. The edge type is either the numerical id of the layer, or "C" for an inter-layer coupling. Given that this is a one-to-one multilayer network, determine whether this network has a star, clique or chain coupling.

3. The network in `http://www.networkatlas.eu/exercises/7/3/data.txt` is a hypergraph, with a hyperedge per line. Transform it in a unipartite network in which each hyperedge is split in edges connecting all nodes in the hyperedge. Then transform it into a bipartite network in which each hyperedge is a node of one type and its nodes connect to it.

4. The network in `http://www.networkatlas.eu/exercises/7/4/data.txt` is dynamic, the third and fourth columns of the edge list tell you the first and last snapshot in which the edge was continuously present. An edge can reappear if the edge was present in two discontinuous time periods. Aggregate it using a disjoint window of size 3.