



MICHELE COSCIA

IT UNIVERSITY OF COPENHAGEN

THE ATLAS FOR THE ASPIRING NETWORK SCIENTIST

Copyright © 2025 Michele Coscia

MICHELE COSCIA IS EMPLOYED BY THE IT UNIVERSITY OF COPENHAGEN, RUED LANGGAARDS VEJ 7, 2300
COPENHAGEN, DENMARK

TUFTETEX.GOOGLECODE.COM

Licensed under the Apache License, Version 2.0 (the “License”); you may not use this file except in compliance with the License. You may obtain a copy of the License at <http://www.apache.org/licenses/LICENSE-2.0>. Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an “AS IS” BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

First Edition, January, 2021

Second Edition, January, 2025

Contents

| | | |
|------------|------------------------------|------------|
| 1 | Introduction | 7 |
| I | Background Knowledge | 21 |
| 2 | Probability Theory | 22 |
| 3 | Statistics | 39 |
| 4 | Machine Learning | 55 |
| 5 | Linear Algebra | 70 |
| II | Graph Representations | 90 |
| 6 | Basic Graphs | 91 |
| 7 | Extended Graphs | 103 |
| 8 | Matrices | 119 |
| III | Simple Properties | 133 |
| 9 | Degree | 134 |
| 10 | Paths & Walks | 154 |
| 11 | Random Walks | 165 |
| 12 | Density | 179 |
| IV | Centrality | 190 |
| 13 | Shortest Paths | 191 |

| | |
|---|------------|
| 14 Node Ranking | 207 |
| 15 Node Roles | 225 |
| V Synthetic Graph Models | 236 |
| 16 Random Graphs | 237 |
| 17 Understanding Network Properties | 247 |
| 18 Generating Realistic Data | 258 |
| 19 Evaluating Statistical Significance | 274 |
| VI Spreading Processes | 285 |
| 20 Epidemics | 286 |
| 21 Complex Contagion | 299 |
| 22 Catastrophic Failures | 314 |
| VII Link prediction | 326 |
| 23 For Simple Graphs | 327 |
| 24 For Multilayer Graphs | 344 |
| 25 Designing an Experiment | 356 |
| VIII The Hairball | 367 |
| 26 Bipartite Projections | 368 |
| 27 Network Backboning | 381 |
| 28 Uncertainty & Measurement Error | 395 |
| 29 Network Sampling | 412 |
| IX Mesoscale | 429 |
| 30 Homophily | 430 |
| 31 Quantitative Assortativity | 441 |

| | |
|---|------------|
| 32 Core-Periphery | 450 |
| 33 Hierarchies | 462 |
| 34 High-Order Dynamics | 474 |
| | |
| X Communities | 489 |
| 35 Graph Partitions | 490 |
| 36 Community Evaluation | 510 |
| 37 Hierarchical Community Discovery | 531 |
| 38 Overlapping Coverage | 543 |
| 39 Bipartite Community Discovery | 561 |
| 40 Multilayer Community Discovery | 572 |
| | |
| XI Graph Mining | 587 |
| 41 Frequent Subgraph Mining | 588 |
| 42 Shallow Graph Learning | 606 |
| 43 Random Walk Embeddings | 622 |
| 44 Message-Passing & Graph Convolution | 636 |
| 45 Deep Graph Learning Models | 654 |
| | |
| XII Holistic Network Analysis | 667 |
| 46 Graph Summarization | 668 |
| 47 Node Vector Distance | 679 |
| 48 Topological Distances | 697 |
| | |
| XIII Visualization | 711 |
| 49 Node Visual Attributes | 712 |
| 50 Edge Visual Attributes | 727 |
| 51 Network Layouts | 735 |

| | |
|--|------------|
| XIV Useful Resources | 756 |
| 52 Network Science Applications | 757 |
| 53 Data & Tools | 770 |
| 54 Glossary | 789 |
| 55 Most Common Abbreviations | 799 |
| Bibliography | 803 |

1

Introduction

Network science is a way to make sense of complex systems by modeling them as relations between their interacting components. It is immensely useful for two reasons: networks can model most – if not all – complex systems, and understanding complexity is the greatest challenge in front of humanity. This is, in my opinion, the reason why every scientist should be at least a little bit of a network scientist, and what motivated me to write this book.

1.1 Why Complexity is the Key

It is challenging to define what a complex system is¹, but my way to understand complexity is by using this definition:

A complex system is a system whose behavior cannot be reduced by analyzing its interacting parts.

This serves me well, because it can capture many key things about complex systems:

- Emergence: having properties that the parts do not have. There's no neuron that is conscious, no water molecule is wet, no person is a country.
- Chaos²: changing a part can have hard-to-predict effects on the system. Removing a species from an ecosystem can lead to the extinction of another that did not have any relation – direct or indirect – with it, because it is difficult to foresee a chained series of reactions to that removal.
- Self-Organization: the same part behaves differently depending on the context provided by (groups of) other parts. Think about the fact that all your cells have the same DNA, but a muscle cell is radically different from a bone cell.

¹ James Ladyman, James Lambert, and Karoline Wiesner. What is a complex system? *European Journal for Philosophy of Science*, 3:33–67, 2013

² Norman H Packard. Adaptation toward the edge of chaos. *Dynamic patterns in complex systems*, 212:293–301, 1988

And so on.

The diversity of these examples shows how complexity is all around us – and so are networks. At some level, every aspect of reality seems to be made by interconnected parts. Societies are made by people entertaining multiple different types of relations with each other: artist citing each other’s works, people making financial transactions, or developing friendships and enmities. The brains in their skulls are an intertwined web of neurons and synapses, but also machines to make inferences³, which is another way to say connecting stimuli to each other. They’re built with genes connected in a ballet of upregulating and downregulating dynamics. Genes are made with interacting proteins. Chemical compounds are atoms linked by bonds. Feynman’s diagrams⁴ show elementary particles having all sorts of interesting relations. It’s interactions all the way down.

If complexity is all around us, why did I say it is difficult to define? Part of the reason is because it is difficult to quantify. We have decided to describe reality with the language of math, to the point of believing that the math is actually the only thing that is real and objective⁵ but, try as we might, we haven’t been able to quantify complexity. If math is the language of science, then our understanding of complexity is pre-scientific, because we haven’t found a way to fit complexity in our language. So we can’t find the laws of complexity.

The solution, in my opinion, is a change in perspective. We need to develop a new language of complexity and go beyond our simple quantitative approach to science. We need to embrace complexity, not pigeonhole it into formulas. After all, the math is only useful when it hides the complexity away. One can be allured by the beauty of math and how well it describes reality, but math is only beautiful insofar it hides complexity, rather than explaining it. Take for instance the Standard Model of physics. This is a model that succinctly describes every elementary interaction (except gravity). However, if you were trying to use it to simulate a single iron atom in isolation you’d have a computationally intractable problem in your hands.

If you look at the formula behind the Standard Model of physics, you’ll realize why:

$$\begin{aligned}
 & -\frac{1}{2}\partial_\nu g_\mu^a \partial_\nu g_\mu^a - g_s f^{abc} \partial_\mu g_\nu^a g_\mu^b g_\nu^c - \frac{1}{4}g_s^2 f^{abc} f^{ade} g_\mu^b g_\nu^c g_\mu^d g_\nu^e + \\
 & \frac{1}{2}ig_s^2 (\bar{q}_i^\sigma \gamma^\mu q_j^\sigma) g_\mu^a + \bar{G}^a \partial^2 G^a + g_s f^{abc} \partial_\mu \bar{G}^a G^b g_\mu^c - \partial_\nu W_\mu^+ \partial_\nu W_\mu^- - \\
 & M^2 W_\mu^+ W_\mu^- - \frac{1}{2}\partial_\nu Z_\mu^0 \partial_\nu Z_\mu^0 - \frac{1}{2c_w^2} M^2 Z_\mu^0 Z_\mu^0 - \frac{1}{2}\partial_\mu A_\nu \partial_\mu A_\nu - \frac{1}{2}\partial_\mu H \partial_\mu H - \\
 & \frac{1}{2}m_h^2 H^2 - \partial_\mu \phi^+ \partial_\mu \phi^- - M^2 \phi^+ \phi^- - \frac{1}{2}\partial_\mu \phi^0 \partial_\mu \phi^0 - \frac{1}{2c_w^2} M \phi^0 \phi^0 - \beta_h [\frac{2M^2}{g^2} + \\
 & \frac{2M}{g} H + \frac{1}{2}(H^2 + \phi^0 \phi^0 + 2\phi^+ \phi^-)] + \frac{2M^4}{g^2} \alpha_h - ig c_w [\partial_\nu Z_\mu^0 (W_\mu^+ W_\nu^- - \\
 & W_\nu^+ W_\mu^-) - Z_\nu^0 (W_\mu^+ \partial_\nu W_\mu^- - W_\mu^- \partial_\nu W_\mu^+) + Z_\mu^0 (W_\nu^+ \partial_\nu W_\mu^- - \\
 & W_\nu^- \partial_\nu W_\mu^+)] - ig s_w [\partial_\nu A_\mu (W_\mu^+ W_\nu^- - W_\nu^+ W_\mu^-) - A_\nu (W_\mu^+ \partial_\nu W_\mu^- -
 \end{aligned}$$

³ Karl Friston. The free-energy principle: a unified brain theory? *Nature reviews neuroscience*, 11(2):127–138, 2010

⁴ Richard Feynman. The theory of positrons. *Phys. Rev.*, 76:749–759, 1949

⁵ Max Tegmark. The mathematical universe. *Foundations of physics*, 38(2):101–150, 2008

$$\begin{aligned}
& W_\mu^- \partial_\nu W_\mu^+) + A_\mu (W_\nu^+ \partial_\nu W_\mu^- - W_\nu^- \partial_\nu W_\mu^+)] - \frac{1}{2} g^2 W_\mu^+ W_\mu^- W_\nu^+ W_\nu^- + \\
& \frac{1}{2} g^2 W_\mu^+ W_\nu^- W_\mu^+ W_\nu^- + g^2 c_w^2 (Z_\mu^0 W_\mu^+ Z_\nu^0 W_\nu^- - Z_\mu^0 Z_\mu^0 W_\nu^+ W_\nu^-) + \\
& g^2 s_w^2 (A_\mu W_\mu^+ A_\nu W_\nu^- - A_\mu A_\nu W_\mu^+ W_\nu^-) + g^2 s_w c_w [A_\mu Z_\nu^0 (W_\mu^+ W_\nu^- - \\
& W_\nu^+ W_\mu^-) - 2 A_\mu Z_\mu^0 W_\nu^+ W_\nu^-] - g\alpha [H^3 + H\phi^0\phi^0 + 2H\phi^+\phi^-] - \\
& \frac{1}{8} g^2 \alpha_h [H^4 + (\phi^0)^4 + 4(\phi^+\phi^-)^2 + 4(\phi^0)^2\phi^+\phi^- + 4H^2\phi^+\phi^- + \\
& 2(\phi^0)^2 H^2] - g M W_\mu^+ W_\mu^- H - \frac{1}{2} g \frac{M}{c_w^2} Z_\mu^0 Z_\mu^0 H - \frac{1}{2} i g [W_\mu^+ (\phi^0 \partial_\mu \phi^- - \\
& \phi^- \partial_\mu \phi^0) - W_\mu^- (\phi^0 \partial_\mu \phi^+ - \phi^+ \partial_\mu \phi^0)] + \frac{1}{2} g [W_\mu^+ (H \partial_\mu \phi^- - \phi^- \partial_\mu H) - \\
& W_\mu^- (H \partial_\mu \phi^+ - \phi^+ \partial_\mu H)] + \frac{1}{2} g \frac{1}{c_w} (Z_\mu^0 (H \partial_\mu \phi^0 - \phi^0 \partial_\mu H) - \\
& i g \frac{s_w^2}{c_w} M Z_\mu^0 (W_\mu^+ \phi^- - W_\mu^- \phi^+) + i g s_w M A_\mu (W_\mu^+ \phi^- - W_\mu^- \phi^+) - \\
& i g \frac{1-2c_w^2}{2c_w^2} Z_\mu^0 (\phi^+ \partial_\mu \phi^- - \phi^- \partial_\mu \phi^+) + i g s_w A_\mu (\phi^+ \partial_\mu \phi^- - \phi^- \partial_\mu \phi^+) - \\
& \frac{1}{4} g^2 W_\mu^+ W_\mu^- [H^2 + (\phi^0)^2 + 2\phi^+\phi^-] - \frac{1}{4} g^2 \frac{1}{c_w^2} Z_\mu^0 Z_\mu^0 [H^2 + (\phi^0)^2 + 2(2s_w^2 - \\
& 1)^2 \phi^+\phi^-] - \frac{1}{2} g^2 \frac{s_w^2}{c_w} Z_\mu^0 \phi^0 (W_\mu^+ \phi^- + W_\mu^- \phi^+) - \frac{1}{2} i g^2 \frac{s_w^2}{c_w} Z_\mu^0 H (W_\mu^+ \phi^- - \\
& W_\mu^- \phi^+) + \frac{1}{2} g^2 s_w A_\mu \phi^0 (W_\mu^+ \phi^- + W_\mu^- \phi^+) + \frac{1}{2} i g^2 s_w A_\mu H (W_\mu^+ \phi^- - \\
& W_\mu^- \phi^+) - g^2 \frac{s_w}{c_w} (2c_w^2 - 1) Z_\mu^0 A_\mu \phi^+ \phi^- - g^1 s_w^2 A_\mu A_\mu \phi^+ \phi^- - \bar{e}^\lambda (\gamma \partial + \\
& m_e^\lambda) e^\lambda - \bar{v}^\lambda \gamma \partial v^\lambda - \bar{u}_j^\lambda (\gamma \partial + m_u^\lambda) u_j^\lambda - \bar{d}_j^\lambda (\gamma \partial + m_d^\lambda) d_j^\lambda + \\
& i g s_w A_\mu [-(\bar{e}^\lambda \gamma^\mu e^\lambda) + \frac{2}{3} (\bar{u}_j^\lambda \gamma^\mu u_j^\lambda) - \frac{1}{3} (\bar{d}_j^\lambda \gamma^\mu d_j^\lambda)] + \frac{i g}{4c_w} Z_\mu^0 [(\bar{v}^\lambda \gamma^\mu (1 + \\
& \gamma^5) v^\lambda) + (\bar{e}^\lambda \gamma^\mu (4s_w^2 - 1 - \gamma^5) e^\lambda) + (\bar{u}_j^\lambda \gamma^\mu (\frac{4}{3}s_w^2 - 1 - \gamma^5) u_j^\lambda) + \\
& (\bar{d}_j^\lambda \gamma^\mu (1 - \frac{8}{3}s_w^2 - \gamma^5) d_j^\lambda)] + \frac{i g}{2\sqrt{2}} W_\mu^+ [(\bar{v}^\lambda \gamma^\mu (1 + \gamma^5) e^\lambda) + (\bar{u}_j^\lambda \gamma^\mu (1 + \\
& \gamma^5) C_{\lambda\kappa} d_j^\kappa)] + \frac{i g}{2\sqrt{2}} W_\mu^- [(\bar{e}^\lambda \gamma^\mu (1 + \gamma^5) v^\lambda) + (\bar{d}_j^\kappa C_{\lambda\kappa}^\dagger \gamma^\mu (1 + \gamma^5) u_j^\lambda)] + \\
& \frac{i g}{2\sqrt{2}} \frac{m_e^\lambda}{M} [-\phi^+ (\bar{v}^\lambda (1 - \gamma^5) e^\lambda) + \phi^- (\bar{e}^\lambda (1 + \gamma^5) v^\lambda)] - \frac{g}{2} \frac{m_e^\lambda}{M} [H (\bar{e}^\lambda e^\lambda) + \\
& i \phi^0 (\bar{e}^\lambda \gamma^5 e^\lambda)] + \frac{i g}{2M\sqrt{2}} \phi^+ [-m_d^\kappa (\bar{u}_j^\lambda C_{\lambda\kappa} (1 - \gamma^5) d_j^\kappa) + m_u^\lambda (\bar{u}_j^\lambda C_{\lambda\kappa} (1 + \\
& \gamma^5) d_j^\kappa)] + \frac{i g}{2M\sqrt{2}} \phi^- [m_d^\lambda (\bar{d}_j^\lambda C_{\lambda\kappa}^\dagger (1 + \gamma^5) u_j^\kappa) - m_u^\kappa (\bar{d}_j^\lambda C_{\lambda\kappa}^\dagger (1 - \gamma^5) u_j^\kappa)] - \\
& \frac{g}{2} \frac{m_u^\lambda}{M} H (\bar{u}_j^\lambda u_j^\lambda) - \frac{g}{2} \frac{m_d^\lambda}{M} H (\bar{d}_j^\lambda d_j^\lambda) + \frac{i g}{2} \frac{m_u^\lambda}{M} \phi^0 (\bar{u}_j^\lambda \gamma^5 u_j^\lambda) - \frac{i g}{2} \frac{m_d^\lambda}{M} \phi^0 (\bar{d}_j^\lambda \gamma^5 d_j^\lambda) + \\
& \bar{X}^+ (\partial^2 - M^2) X^+ + \bar{X}^- (\partial^2 - M^2) X^- + \bar{X}^0 (\partial^2 - \frac{M^2}{c_w^2}) X^0 + \bar{Y} \partial^2 Y + \\
& i g c_w W_\mu^+ (\partial_\mu \bar{X}^0 X^- - \partial_\mu \bar{X}^+ X^0) + i g s_w W_\mu^+ (\partial_\mu \bar{Y} X^- - \partial_\mu \bar{X}^+ Y) + \\
& i g c_w W_\mu^- (\partial_\mu \bar{X}^- X^0 - \partial_\mu \bar{X}^0 X^-) + i g s_w W_\mu^- (\partial_\mu \bar{X}^- Y - \partial_\mu \bar{Y} X^+) + \\
& i g c_w Z_\mu^0 (\partial_\mu \bar{X}^+ X^+ - \partial_\mu \bar{X}^- X^-) + i g s_w A_\mu (\partial_\mu \bar{X}^+ X^+ - \partial_\mu \bar{X}^- X^-) - \\
& \frac{1}{2} g M [\bar{X}^+ X^+ H + \bar{X}^- X^- H + \frac{1}{c_w^2} \bar{X}^0 X^0 H] + \frac{1-2c_w^2}{2c_w} i g M [\bar{X}^+ X^0 \phi^+ - \\
& \bar{X}^- X^0 \phi^-] + \frac{1}{2c_w} i g M [\bar{X}^0 X^- \phi^+ - \bar{X}^0 X^+ \phi^-] + i g M s_w [\bar{X}^0 X^- \phi^+ - \\
& \bar{X}^0 X^+ \phi^-] + \frac{1}{2} i g M [\bar{X}^+ X^+ \phi^0 - \bar{X}^- X^- \phi^0]
\end{aligned}$$

My thesis is that we need to understand complexity so we find a better language to describe reality.

We already know that, because no one uses that formula to do chemistry. We compartmentalized the fields of knowledge, because we know we can't describe societies via an explanation of quantum interactions of elementary particles, climbing the long and perilous ladder of fields in order of purity⁶. Some phenomena cannot be reduced to the underlying laws⁷. We do need to toss away a good chunk of physics, add a bunch of new tools, to understand this

⁶ <https://xkcd.com/435/>

⁷ Philip W Anderson. More is different. *Science*, 177(4047):393–396, 1972

new field called “chemistry”, because the change in scale causes the emergence of new phenomena.

We have some starting pointers to understand how this compartmentalization of knowledge can help scientific investigation. This is what Hayek called “division of knowledge”⁸, which is a much more powerful concept than Smith’s classical division of labor⁹. If I specialize as a chemist and hone my skills and tools to that specific task, I can be immensely more productive, because I am outsourcing all other knowledge discovery endeavors to other specialists. This is how societies grow their pool of knowledge efficiently. However, the result is that, now, no individual can really fully grasp a well-rounded picture of reality. The collective society can, but not its individual components. It is all deformed by the lens of their specialization.

The resulting irony is that we might not be able to get to the language of complexity because we need it in order to get it. To make an advancement in physics we need teams of tens or hundreds of people. The paper containing the discovery of the Higgs boson¹⁰ has 5,154 authors. The knowledge needed for it was so vast it could not fit in a single brain. Only a collective of interconnected brains could understand it – and that is a complex system.

How do you meaningful coordinate a complex system to make an even vaster scientific discovery? You can only do it if you understand complexity – which is what you need the collective of brains to do! It’s a circular problem. We already know that simple interventions will cause unexpected local optima that are unsatisfactory. Science with its replication crisis¹¹, its publish-or-perish misaligned incentives, its difficulty in dealing with misinformation¹², isn’t going as smoothly as it could. Because we don’t understand complexity. Because we need to understand complexity in order to understand complexity.

But I’ll be damned if I don’t give everything I have to make this understanding of complexity happen. And I think our best shot is via network science, because it is the field that gives us a way to talk about emergence. We need to know how the different fields – physics, chemistry, biology, ... – relate and transform into each other, which is necessary to reconstruct a picture of reality.

Connecting those fields means finding a shared language that can describe the *relations* between the *symbols* they use – make a mental note of this, it’ll come back later. A shared language can then be universal and move across layers and fields. If you understand intelligence as a way of how information is aggregated and manipulated in each part given its interactions, then this description is independent of what the parts actually are, as long as they can perform the same function. You can use this network theory of intelligence to describe

⁸ Friedrich August Hayek. The use of knowledge in society. *The American economic review*, 35(4):519–530, 1945

⁹ Adam Smith. *The Wealth of Nations*. 1776

¹⁰ Georges Aad, Tatevik Abajyan, Brad Abbott, Jalal Abdallah, S Abdel Khalek, Ahmed Ali Abdelalim, R Aben, B Abi, M Abolins, OS AbouZeid, et al. Observation of a new particle in the search for the standard model higgs boson with the atlas detector at the lhc. *Physics Letters B*, 716(1):1–29, 2012

¹¹ John PA Ioannidis. Why most published research findings are false. *PLoS medicine*, 2(8):e124, 2005

¹² Carl T Bergstrom and Jevin D West. *Calling bullshit: The art of skepticism in a data-driven world*. Random House Trade Paperbacks, 2021

not only how individual brains learn, but how collectives made of brains learn.

In summary, I hope I'm wrong when I say that we need to understand complexity in order to understand complexity. I hope network science can bootstrap our understanding. By teaching you network science with this book, I'm trying my darnedest to prove myself wrong. I want you – the collective of all the 25 people who'll read this thing – to understand complexity and save science and society in the process. No pressure.

1.2 *The Creation Myth of Network Science*

I just put a big weight on the shoulders of network science and network scientists. That is because I think this is a promising field, because networks provide a versatile way of representing virtually anything that is made of interacting parts. To understand why, it is necessary to explore the surface of network science's history – and deconstruct its creation myth, which is always a fun thing to do.

Normally, the creation myth of network science starts with Euler's solution to the famous Königsberg Bridge Problem¹³. The problem asks whether it is possible to cross all bridges of Königsberg (now Kaliningrad) without crossing the same bridge twice. Euler realized that the problem didn't require reasoning about any quantities: it was exclusively a problem about the qualitative relationships between islands and bridges, whether you could use the latter to reach the former. So he abstracted quantities away and created the mathematical object of the graphs, which only has relations between the various parts.

This, as every single creation myth ever, is obviously false. Not only because Euler himself credited this idea of qualitative topology to Leibniz in the introduction of his paper¹⁴. Not only because Euler's actual solution has no graph at all but it's instead combinatorial in nature. Not even because it's kind of ridiculous to say anyone invented graphs – it'd be similar to say someone invented sticks, sure someone must have written about them first, but did they really invent them?

These are three great reasons why the creation myth of network science on Euler's desk is false, but there's one even more fundamental. That is, at best, a creation myth of *graph theory*, but *network science* is an entirely different beast. If graphs were born on Euler's desk in 1736, why does 99.9% of network science happened from the 1930s on?

Modern network science is a gift from sociology. Before sociology, graphs were seen as exact and deterministic mathematical objects,

¹³ Leonhard Euler. *Solutio problematis ad geometriam situs pertinentis. Commentarii academiae scientiarum Petropolitanae*, pages 128–140, 1741

¹⁴ Maximilian Schich. Cultural analysis situs. 2019

worthy of exploration through the manipulation of *abstract symbols*. Sociologists saw the value in using these mathematical objects – symbols – to investigate a statistical and stochastic reality. This was the first – fundamental and necessary – explosion in possibilities for a true network science. If we want to have a better creation myth for network science, we should replace Euler’s God-figure with Helen Hall Jennings – who invented the sociograms¹⁵ that were the real beginning of network science.

Jennings had two problems in the 1930s, though: she could only collect and manipulate data manually, and lacked a unified language to represent *all* of reality as an analyzable symbol. She could do ad-hoc representations, tailored for a specific problem, but lacked the representation power necessary to use networks as the tool to understand complexity in general.

What Jennings needed was the computer. The value of the computer is not that it can perform lots of operations quickly, although that certainly helps. One can prove theorems and lemmas without computers. Rather, the revolution of the computer is in its symbolic language. The computer’s zeroes and ones can be a universal language to represent reality – that shared language I alluded to before. Computers seem to be able to allow you to manipulate anything: with spreadsheets you can tame problems in logistics, with XML you can map semantic concepts, with media players you can appreciate art and videos¹⁶. And yet, inside computers you just have a mass of zeroes and ones.

The power of the computer is its ability of seeing everything – anything – as a symbol. Once everything is a symbol, you can understand its relations with other symbols¹⁷. In this sense, the true revolution of the computer was not pioneered by Babbage and von Neumann¹⁸. What they did was an immensely useful mechanical invention, but the revolution we needed was a logical invention for the manipulation of symbols and their interactions. This was gifted us by the first programmer Ada Lovelace¹⁹, and by Wittgenstein’s sharp observations of the relations between symbols and reality²⁰.

The second half of the XX century was the moment when we started connecting symbols together in the same place: in the memory of a computer. Ironically, computers facilitated the emergence of even more networks that were latently waiting to express their potential. For instance, the invention of the Internet via ARPANET – and of e-mail – codified explicitly the relationships between centers of command and of knowledge creation that existed across the world. But it was arguably Berners-Lee²¹ – following in the footsteps of Bush²² – that truly understood how to piece symbols together in computers.

¹⁵ Jacob L Moreno and Helen H Jennings. Statistics of social configurations. *Sociometry*, pages 342–374, 1938

¹⁶ It has been legendarily said that VLC, one of the most popular multimedia software, can open anything – even a can of tuna.

¹⁷ One of the facts that never fails to blow my mind is the realization that a piece of software is, after all, just a very cleverly composed number. Thus you can sum Adobe Photoshop to Google Chrome, although the result won’t probably make much sense. That is also why there exist such a thing as an “illegal number” (https://en.wikipedia.org/wiki/Illegal_number)

¹⁸ John von Neumann. First draft of a report on the edvac. 1945

¹⁹ Ada Lovelace. Notes on menabrea’s “sketch of the analytical engine invented by charles babbage”, 1842

²⁰ Ludwig Wittgenstein. Tractatus logico-philosophicus, 1921

²¹ Timothy J Berners-Lee. Information management: A proposal. Technical report, 1989

²² Vannevar Bush et al. As we may think. *The atlantic monthly*, 176(1):101–108, 1945

Once hypertexts birthed the Web, it was just a formality to get the papers published before modern network science could kick into gear. We finally had both the tools and the data to really understand how universal networks were, and developing the language we could use to talk about them. Thus, in a sense, the XX century planted the seed for the XXI: the great awakening of the world to the pervasive presence of networks – and complexity – in everything we do.

1.3 How This Book Approaches Complexity

There are a ton and a half other books about network science out there. I'd bet that a good chunk of them are better than mine. I do not advocate that this is the only book you should read to understand network science and to be a network analyst. In fact, I advocate the opposite: read diverse takes and use diverse structures to think about networks. I see my book as complementary to those out there.

Given that I am belatedly giving the credit to Jennings and sociologists for the creation of network science, one should consider social network analysis as the foundational approach. In that, Wasserman's and Faust's classic is a must read²³.

Post-1998 network science arguably took off because the innovations from sociology were picked up by physicists^{24,25}, who inspired other fields with their quest for universal laws. It is no wonder that there exists a plethora of books^{26,27,28} and review articles²⁹ taking the physics angle on network science. Among these, a few certainly stand out. Barabási's book³⁰ towers in accessibility and clarity, while Newman's work is probably the most complete and in-depth³¹. Physicists also have a good track record in publishing books for the wider audience^{32,33}, not necessarily with a scientific background in mind.

Given the reliance on computational tools and the need of processing large amounts of data, the computer science angle should not be ignored³⁴. A great favorite of mine combines the computer science methodology with applications in economics³⁵. If you need yet another proof of the breadth of approaches in network science, consider that another major book on the topic was authored by a chemist³⁶.

The natural question now is: if there are already so many network science books and they are all great, what is the need of this one you're reading? Is it just for updating with the newest developments in the field? Not really. I hope you noticed that, when presenting the other network science books, I never introduced them as books written by a network scientist. This was not by accident. My impression is that these books are aimed at introducing people from a variety of disciplines into the skills and tools of network science, rather than

²³ Stanley Wasserman and Katherine Faust. *Social network analysis: Methods and applications*, volume 8. Cambridge university press, 1994

²⁴ Duncan J Watts and Steven H Strogatz. Collective dynamics of 'small-world'networks. *nature*, 393(6684):440, 1998

²⁵ Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *science*, 286(5439):509–512, 1999

²⁶ Guido Caldarelli and Michele Catanzaro. *Networks: A very short introduction*, volume 335. Oxford University Press, 2012

²⁷ Guido Caldarelli. *Scale-free networks: complex webs in nature and technology*. Oxford University Press, 2007

²⁸ Vito Latora, Vincenzo Nicosia, and Giovanni Russo. *Complex networks: principles, methods and applications*. Cambridge University Press, 2017

²⁹ Claudio Castellano, Santo Fortunato, and Vittorio Loreto. Statistical physics of social dynamics. *Reviews of modern physics*, 81(2):591, 2009

³⁰ Albert-László Barabási et al. *Network science*. Cambridge university press, 2016

³¹ Mark Newman. *Networks*. Oxford university press, 2018a

³² Albert-László Barabási. *Linked: The new science of networks*, 2003

³³ Duncan J Watts. *Six degrees: The science of a connected age*. WW Norton & Company, 2004

³⁴ Filippo Menczer, Santo Fortunato, and Clayton A Davis. *A First Course in Network Science*. Cambridge University Press, 2020

³⁵ David Easley and Jon Kleinberg. *Networks, crowds, and markets: Reasoning about a highly connected world*. Cambridge University Press, 2010

³⁶ Ernesto Estrada. *The structure of complex networks: theory and applications*. Oxford University Press, 2012

examining network science from within.

There are now PhD programs for network scientists³⁷, but they are only a handful years old, meaning that there are only a few graduates coming out of them and they do not have yet the time or the experience to write a network science book. Worse still, I believe there are even fewer master and bachelor programs in network science, if any. This means that every book you can find on network science is a sort of “something/network science”, with that something being sociology, physics, computer science, archaeology, or other.

This book has the – probably overambitious – aim of being no “slash something”. It wants to be a pure breed: just a network science book. In other words, the difference between this and the other books is that this book considers “network science” not as something one attaches to another discipline, but rather it is a discipline in itself. People can – and should! – be trained from scratch in it.

I believe my background is as close as it could be to the right mix that network analysis requires. I am a digital humanist, a field pioneered by Busa³⁸ which focuses on the digital processing of content produced by humans. This is to say: the computer-mediated manipulation and analysis of symbols representing different facets of reality – which are not necessarily mathematical – and their connections. If this sounds familiar, it is because this is the exact characterization of network science as the key or representing and understanding complexity that I adopted in this introduction.

By the time I started a PhD, there was no one offering it in network science – or digital humanities – so, formally, I am a computer scientist as well. However, from day one, I immersed myself in network science literature in all its facets – physics, computer science, sociology – armed with my digital humanities toolbox. I designed new network science algorithms and at the same time studied Dante’s *Inferno* as a complex network³⁹; I scouted for laws in complex systems whilst fighting Mexican drug traffic⁴⁰. Everything I did was in an attempt to be an all-round network scientist. And this is the same hat I’m wearing as the author of this book.

If I do so, it is because I am intimately convinced that network science is truly a special field. This is not only because, as I opened this introduction, relations are what I consider being a fundamental way to understand reality. That consideration is only the beginning: it caused network science to have its complex and multifaceted origin story – combining all the fields that I’ve been mentioning so far. By birthing out of many different scientific – and non-scientific! – disciplines, network science is truly a method to grasp emergence.

Connecting to the first section of this introduction, it should now

³⁷ <https://www.networkscienceinstitute.org/phd>

³⁸ Roberto Busa. Index thomisticus sancti thomae aquinatis operum omnium indices et concordantiae in quibus verborum omnium et singulorum formae et lemmata cum suis frequentiis et contextibus variis modis referuntur. 1974

³⁹ Amedeo Cappelli, Michele Coscia, Fosca Giannotti, Dino Pedreschi, and Salvo Rinzivillo. The social network of dante’s inferno. *Leonardo*, 44(3):246–247, 2011

⁴⁰ Michele Coscia and Viridiana Rios. Knowing where and how criminal organizations operate using web content. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, pages 1412–1421, 2012

be clear why I consider network science important, and a truly *network* science book necessary: it is our best shot at building a collective understanding of all human knowledge, and such attempt needs to be approached with the proper humility of those who are not expert in anything else but gluing together the pieces created by the real experts.

That said, I don't want to oversell the importance of network science. If what I said is really true, it means that we can represent any – or at least most – aspects of reality as mathematical symbols and we can manipulate them with the mathematical tools of computer and network science. Which means that a complete understanding of them is necessarily out of reach. Not just because, as Poincaré would put it, “the head of the scientist, which is only a corner of the universe, could never contain the universe entire”⁴¹. Rather, because Gödel taught us that there is a strong bound of what is tractable in a formal system⁴². At some point, even when you represent the entirety of reality as interconnected mathematical symbols, you will need to jump out and look at the loops from the outside⁴³. No book can really give you a scientific road map on how to do so.

1.4 What is in This Book?

This is all fine and dandy but, at the end of the day, what does this book *contain*?

At a general level, it contains the widest possible span of all that is related to network science that I know. It is the result of twelve sixteen⁴⁴ years of experience that I poured on the field. Virtually any concept that I used or that I simply came to know in these fifteen years is represented in at least a sentence in this book.

As you might expect, this is a lot to include and would not fit a book, not even a ~ 800 pages like this one. By necessity, many – if not all – of the topics included in this book are treated relatively superficially. I would not say that this book would provide you what you need to know to be a network scientist. But it would *point* you to what you need to know⁴⁵. To borrow from Rumsfeld⁴⁶: the book provides little to no *known knowns*, but it will provide you with all the *known unknowns* in network science – so that your *unknown unknowns* are aligned with those of everyone else. After internalizing this book, you will know what you don't know; you will be handed all the tools you need to ask meaningful questions about network science in 2025. You can go to the other books or to any other article, and find the answers. Or you can figure out the answer yourself.

That is why I decided to call this book an “Atlas”. It is the map you need to set foot among networks and start exploring. An atlas

⁴¹ Henri Poincaré. *The foundations of science*. 1913

⁴² Kurt Gödel. Über formal unentscheidbare sätze der principia mathematica und verwandter systeme i. *Monatshefte für mathematik und physik*, 38(1):173–198, 1931

⁴³ Douglas R Hofstadter. *Gödel, Escher, Bach*. Harvester press Hassocks, Sussex, 1979

⁴⁴ Boy, writing a version 2 of this book really makes me feel even older.

⁴⁵ Connecting the symbols of network science, maybe?

⁴⁶ <https://archive.defense.gov/Transcripts/Transcript.aspx?TranscriptID=2636>

doesn't do the exploration for you, but you can't explore without an atlas. This is the book I wished I had fifteen years ago.

At a more specific level, the book is divided in fourteen parts.

Part I provides the basics for the background knowledge to be a good network scientist. You need to know probability theory, statistics, machine learning, and linear algebra. That's the basic mental toolbox to handle networks. There are no networks in this part, so if you already have a good foundation in these fields you can skip it.

Part II teaches you what a graph is and how many features to the simple mathematical model were added over the years, to empower our symbols to tame more and more complex real world entities. Finally, it pivots perspectives to show an alternative way of manipulating networks, via matrices.

Part III is a carousel of all the simplest analyses you can operate on a graph. These are either local node properties – how many connections a node has –, or global network ones – how many connections on average I have to cross to go from one node to another. We see that some of these are easy to calculate on the graph structure, while others are naturally solved by linear algebra operations. Shifting perspectives is something you need to get used to, if you want to make it as a network scientist.

Part IV uses some of the tools presented in the previous part to build slightly more advanced analyses. Specifically, it focuses on the question: which nodes are playing which role in the network? And: can we say that a node is more important than another? If you want to answer these questions, you need to relate the entire network structure to a node, i.e. to use fully what Part III trained you to do.

Part V teaches you the main approaches for the creation of synthetic network data. It explores the main reasons why we want to do it. Sometimes, it is because we need to test an algorithm and we need a benchmark. Alternatively, we can use these models to reproduce the properties of real world networks we investigated in the previous parts, to see whether we understand the mechanisms that make them emerge.

Part VI starts considering not just network structures, but events on networks. That is, your nodes and your edges represent real world entities that actually do something, rather than simply connecting to each other. Specifically, Part VI deals with things spreading through a network: when a node is affected by something, it

has a chance to pass it to its neighbors via its connections. This something might be a disease in epidemiological models, or a behavior in sociological ones.

Part VII evolves the idea of dynamic events on networks. In Part VI, we assumed that the network structure was unchanging: it was a timeless snapshot of reality and all nodes and connections are eternally the same. In Part VII, we acknowledge that things change: if two nodes are not connected today, they might connect tomorrow. We investigate which techniques allow us to make a good guess on which connections we will observe in the future, on the basis of the ones we are observing now.

Part VIII performs another leap: up until now, we mostly inhabited an ideal world. We assumed we could model phenomena and cleanly gather insights. Here, we have our first impact with the real world. How do networks look like when you gather them via experiments and/or observations? Often, they don't look at all like the ones from your models. The only expectation that reality meets is its inability to meet expectations. This part trains you in the art of cleaning real world data to obtain something passable that can be fed to your neat theories and analyses.

Part IX opens the Pandora's Box of the level of analysis that is the most interesting and probably the one with which you will struggle most of the time: the mesoscale. The mesoscale is what lies between local node properties and global network statistics. This includes – but is not limited to – questions such as: does my network have a hierarchical structure? Is there a densely connected core surrounded by a sparsely connected periphery? Do nodes consider other nodes' properties to decide whether to connect to them?

Part X continues the exploration of the mesoscale. It needs to be split off Part IX because there is one mesoscale analysis that has dominated all other subfields in network science: community discovery. Community discovery is the network equivalent of what clustering is for machine learning: the task of dividing nodes in a network into groups. Nodes in the same groups are densely connected to each other, more so than with nodes in different groups. Or so people would lead you to believe. The fact that there are literally thousands of papers proposing different algorithms to tackle this problem should be a hint at the fact that things might not be as simple.

Part XI takes a steep turn into the realm of computer science. It deals with graph mining: a collection of techniques that allow you to

discover patterns in your graph structure, even if you are not sure about what these patterns might look like or hint at. It is what we would call “bottom-up” discovery. This is where you’ll find a deep dive on graph neural networks, which is one of the fastest moving subfields of network science at the moment.

Part XII comprises a branch of network science that deals with analyzing the network structure as a whole. This is not simply providing summary statistics about the entire network like you’d find in Part III, but trying to embed the entire structure into a more advanced analysis. For instance, you’d deal with networks defining a complex non-Euclidean space.

Part XIII includes a few tips and tricks for an aspect of network science that is rarely covered in other books: how to browse/explore your network data and how to communicate your results. Specifically, I will show you some best practices in visualizing networks. I am a visual thinker and, sometimes, patterns and ideas about those patterns emerge much more clearly when you see them, rather than scouting through summary statistics. Moreover, network science papers thrive on visual communication and a good looking network has an amazing chance of ending up in the cover of the journal you’re publishing on. It is a mystery to me why you would not spend some time in making sure that your network figures are at least of passable quality. Moreover, even if we are all primed to think dots and lines when it comes to visualizing a graph, you should be aware of the situations in which there are different ways to show your network.

Part XIV is a final collection of miscellanea that can help you to venture out in the real world of network analysis. It contains a quick discussion of the applications of network science I find most interesting, and a repository of tools you might need to kickstart your career.

1.5 What’s New in Version Two

This is the second edition of the book. I decided to update the book because there were many things I found unsatisfactory with the first edition. Here I’m letting you know what’s new besides making the introduction slightly less tongue-in-cheek – so that, if you already read version one, you know where to check for what you have missed.

The first thing I greatly expanded was the introductory part. In version one, I only had a chapter on probability theory. The book

contained some linear algebra and machine learning, but those concepts were scattered around in an unorganized way. Part I of this book is a brand new effort to concentrate all that basic knowledge in the same place, adding some statistics for good measure.

The second major change is the coverage of graph neural networks. In version one, I only had a single chapter that put together shallow and deep learning. The explanations were simplistic and obscure, a necessary evil to convey so much content in so little space. Now that single chapter has exploded into four chapters, from Chapter 42 to Chapter 45. Hopefully, now those concepts are explained in a clearer way and I'm doing them justice.

The final major change is about uncertainty. In the original version of the book, I had only a brief half page section lamenting the fact that network scientists don't really give uncertainty its credit – trusting that their networks are a given truth without measurement error. I wanted to amend that by giving justice to all the work done handling uncertainty and probabilistic connections in network science. Chapter 28 is the result.

There are a number of minor additions. The most significant are: a new discussion of effective resistance (Section 11.4) which then enables to talk about how it is used for network statistics (Section 47.5) and applications (Section 52.8); an improved overview of simplicial complexes for the high-order chapter (Section 34.1); and some extensions in the discussion of node similarities (Section 15.2).

There are also a bunch of minor fixes and corrections, some of them were already made for version one but never appeared in print.

1.6 Acknowledgements

You'd be a complete fool if you trusted me to get this amount of knowledge correct by myself. Any and all scientific endeavors are only as good as the attention they receive by their peers, both in terms of building on top of those results, but also in terms of catching and correcting mistakes. This is in line with what I've been saying in this introduction: network science is too vast and hard for me to grasp, so I need to rely on the help of the experts from all of its subfields. It takes a village – or an extensive social network – to write a textbook. Here I want to thank all those who helped me in this journey.

The person who stood up tall above everybody was Aaron Clauset, who gets my most sincere thanks. Aaron is the only one who reviewed almost the whole version one of the book, all the 650 friggin' pages of it. All while rocking a few months old baby daughter. Aaron is a superhero and should have everyone's deep respect.

Another one who went beyond the call of duty was Andres Gomez-Lievano. Andres and I shared a desk for years and I cherish those as the most fun I had at work. Andres didn't stop at the chapters I asked him to review, but deeply commented on the philosophy and framing of this book. I can see in his comments the spark of the years we spent together.

My other kind reviewers were, in alphabetical order: Alexey Medvedev, Andrea Tagarelli, Charlie Brummitt, Ciro Cattuto, Clara Vandeweerd, Fred Morstatter, Giulio Rossetti, Gourab Ghoshal, Isabel Meirelles, Laura Alessandretti, Luca Rossi, Mariano Beguerisse, Marta Sales-Pardo, Matté Hartog, Petter Holme, Renaud Lambiotte, Roberta Sinatra, Yong-Yeol Ahn, and Yu-Ru Lin.

For version two, I have recruited the additional help of: Giovanni Puccetti, Matteo Magnani, Maria Astefanoaei, Daniele Cassese, and Paul Scherer.

All these people donated hours of their time with no real tangible reward, just to make sure my book graduated from "incomprehensible mess" to "almost passable and not misleading". Thank you.

With their work, some reviewers expressed their intent to support charitable organizations. Specifically, they mentioned TechWomen⁴⁷ – to support the careers of women in STEM fields –, Evidence Action⁴⁸ – to expand our de-worming efforts and reaping the surprisingly high societal payoff –, and Doctors without Borders⁴⁹. You should also consider donating to them.

If there's any value in this book, it comes from the hard work of these people. All the mistakes that remain here are exclusively due to my ineptitude in properly implementing my reviewers' valuable comments. I expect there must be many of such mistakes, ranging from trivial typos and clumsily written sentences, to more fundamental issues of misrepresentation. If you find some, feel free to drop me an email to mcos@itu.dk.

If, for some reason, you only have access to a printed version of this book – or you found the PDF somewhere on the Internet, know that there is a companion website⁵⁰ with data for the exercises, their solutions, and – hopefully in the future – interactive visualizations.

⁴⁷ <https://www.techwomen.org/>

⁴⁸ <https://www.evidenceaction.org/>

⁴⁹ <https://www.doctorswithoutborders.org/>

⁵⁰ <https://www.networkatlas.eu/>

Part I

Background Knowledge

2

Probability Theory

Before even mentioning networks, I need to lay the groundwork for a basic understanding of a few concepts necessary to make you a good network analyst. These concepts are part of four broad subjects: probability theory, statistics, linear algebra, and machine learning. We start with probability theory here, because that is the foundation on top of which this pyramid is built. Then, we deal with more specialized statistical concepts in Chapter 3, with machine learning in Chapter 4, and linear algebra in Chapter 5.

These chapters deal with entire fields of human knowledge that are much more vast and complicated than the extremely simplified picture I present here. As with many other chapters, my coverage of the subject is the bare minimum I can get away with. The wisest course of action for you would be to skip this book part entirely and take entire courses on these subjects and then come back to this book and start directly with Part II. Alas, that's not an option for everybody, and that is why this book part covers the basics you need to know to enjoy the rest of the book.

If you want to dive deep into probability theory, there are good books on the subject you should check out^{1,2}. I will attempt, where possible, to give forward references to later topics in this book, to let you know why understanding probability theory is important for a network scientist.

2.1 Frequentism and Bayesianism

Probability theory is the branch of mathematics that allows you to work with uncertain events. It gives you the tools to make inferences in cases of uncertainty.

Probability theory is grounded in mathematical axioms. However, there are different ways to interpret what we really mean with the term “probability”. With a very broad brush, we can divide the main interpretations into two camps: the frequentist and the Bayesian.

¹ Willliam Feller. *An introduction to probability theory and its applications*, volume 2. John Wiley & Sons, 1968

² Rick Durrett. *Probability: theory and examples*, volume 49. Duxbury Press, 1996

There are more subtleties to this, but since these are the two main approaches we will see in this book, there is no reason to make this picture more complex than it needs to be.

To understand the difference, let's suppose you have Mrs. Frequent and Mr. Bayes experimenting with coin tosses. They toss a coin ten times and six out of ten times it turns heads up. Now they ask themselves the question: what is the probability that, if we toss the coin, it will turn heads up again?

Mrs. Frequent reasons as follows: "An event's probability is the relative frequency after many trials. We had six heads after ten tosses, thus my best guess about the probability it'll come out as heads is 60%". Note that Mrs. Frequent doesn't really believe that ten tosses gave him a perfect understanding of that coin's odds of landing on heads. Mrs. Frequent knows that he will get the answer wrong a certain number of times, that is what confidence intervals are for, but for the sake of this example we need not to go there.

"Hold on a second," Mr. Bayes says, "Before we tossed it, I examined the coin with my Coin Examiner™ and it said it was a fair coin. Of course my Coin Examiner™ might have malfunctioned, but that rarely happens. We haven't performed enough experiments to say it did, but I admit that the data shows it might have. So I think the probability we'll get heads again is 51%". Just like Mrs. Frequent, also Mr. Bayes is uncertain, and he has a different procedure to estimate such uncertainty – in this case dubbed "credible intervals" – which again we leave out for simplicity.

Herein lies the difference between a frequentist and a Bayesian. For a frequentist only the outcome of the physical experiment matters. If you toss the coin an infinite number of times, eventually you'll find out what the true probability of it landing on heads is. For a Bayesian it's all about degrees of beliefs. The Bayesian has a set of opinions about how the world works, which they call "priors". Performing enough new experiments can change these priors, using a standard set of procedures to integrate new data. However a Bayesian will never take a new surprising event at face value if it is wildly off its priors, because those priors were carefully obtained knowledge coherent with how the world worked thus far.

Figure 2.1 shows the difference between the mental processes between a frequentist and a Bayesian. The default mode for this book is taking a frequentist approach. However, here and there, Bayesian interpretations are going to pop up, thus you have to know why we're doing things that way.

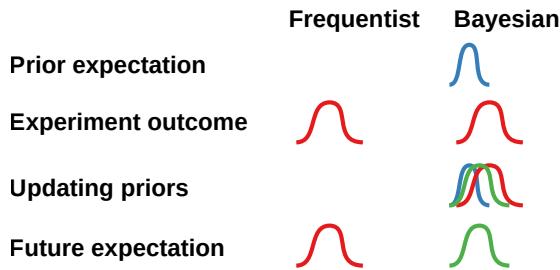


Figure 2.1: Schematics of the mental processes used by a frequentist and a Bayesian when presented with the results of an experiment.

2.2 Notation

Probability theory is useful because it gives us the instruments to talk about uncertain processes. For instance, a process could be tossing a die. The first important thing is to understand the difference between *outcome* and *event*. An *outcome* is a *single* possible result of the experiment. A die landing on 2 is an outcome. An *event* is a *set* of possible outcomes on which we're focusing. In our convention, we use X to refer to outcomes. X is a random variable and it can take many values and forms, and we don't know which of them it will be before actually running the process. As for events, they are the focus of all questions in probability theory: you can sum up probability theory as the set of instruments that allow you to ask and answer questions about events (sets of X) such as: "What is the probability that X , the outcome of the process, is this and/or this but not that and/or that?"

Mathematically one writes such a question as $P(X \in S)$, where S is a set of the values that X takes in our question. X is an outcome, $X \in S$ is an event. For instance, if we were asking about the event "will the die land on an even number?", $S = \{2, 4, 6\}$. So, $P(X \in S)$ asks what's the probability of the "die lands on an even number" event – or for X to take either of the 2, 4, 6 values. Note that elements in S are all possible alternatives: if we write $P(X \in \{2, 4, 6\})$, we're asking about the probability of landing on 2 *or* 4 *or* 6. If you want to have the probability of two events happening simultaneously, you have to explicitly specify it with set notation: $P(X \in \{2, 4, 6\}) \cap P(X \in \{1\})$ asks the probability of landing on an even side and on 1 at the same time.

We also need to consider special questions. For instance, there is the case in which no event happens: $P(X \in \emptyset)$ (here \emptyset refers to the empty set, a set containing no elements). The converse is also important: the probability of any event happening. In the case of the die, there are a total of six possible outcomes. Notation-wise, we define the set of all possible outcomes as $\Omega = \{1, 2, 3, 4, 5, 6\}$. So this is represented as $P(X \in \{1, 2, 3, 4, 5, 6\})$, or $P(X \in \Omega)$. Figure 2.2 shows how the mathematical notation corresponds to our visual

intuition.

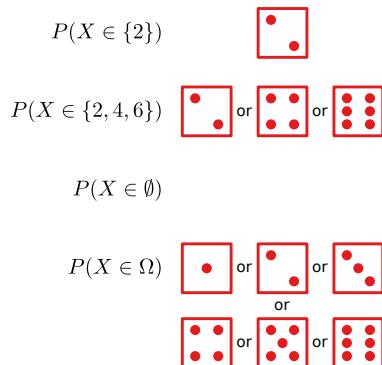


Figure 2.2: A visual shorthand for understanding the mathematical notation of probabilities (left) and the possible outcomes of the “tossing a die” event.

To be more concise, we can skip the explicit reference to the variable X . For instance, we can codify the outcome “the die lands on 3” with the symbol 3. In this way, we can write $P(3)$ to refer to the probability of the die landing on 3, $P(\{2, 4, 6\})$ for the probability of landing on an even number, $P(\{2, 4, 6\}) \cap P(1)$ for landing on an even number and on 1, $P(\emptyset)$ for the probability of nothing happening, and $P(\Omega)$ for the probability of anything possible happening.

2.3 Axioms

When building probability theory we need to establish a set of axioms: unprovable and – hopefully – self-evident statements that allow you to derive all other statements of the theory. Probability theory rests on three of such axioms.

First, the probability of an event is a non-negative number. Or: talking about a “negative probability” doesn’t make any sense. Worst case scenario, an event A is impossible, therefore $P(A) = 0$ – for instance, this is the “nothing happens” case from the previous section when $A = \emptyset$. If A is possible, $P(A) > 0$. In a borderless coin toss, there are only two possible outcomes: heads (H) or tails (T). The coin cannot land on the non-existing rim. Thus, the probability of landing on the rim is zero. It cannot be negative.

Second, certain events occur with probability equal to one. That is, if A is an absolutely certain event, $P(A) = 1$. Using the notation from the previous section: $P(\Omega) = 1$, with $\Omega = \{H, T\}$ for a coin toss. Note that there isn’t anything magical about the number 1, we could have said that the maximum probability is equal to 42, π , or “meh”. It’s just a convenient convention to define your units.

Third, the probability of happening for mutually exclusive events is the sum of their probabilities, or $P(\{H, T\}) = P(H) + P(T)$. A coin cannot land on heads and tails at the same time³, thus the probability

³ Get those Schrödinger coins out of my classroom!

that it lands on heads or tails is the sum of the probability of landing on heads and the probability of landing on tails.

As a corollary, you can also multiply probabilities. If A and B are *independent* events, then $P(A)P(B)$ – their multiplication – tells you the probability of *both* events happening. Independent events are events that have no relation to each other, such as you getting a promotion and the appearance of a new spot on the sun. For *dependent* events, you need to take into account this dependence before applying the multiplication. In a fair die, $P(1) = P(2) = 1/6$, but we know that you can't get a 1 if you are getting a 2, so $P(1)P(2)$ is actually zero, not $1/36$. How to perform this check leads us to the world of conditional probabilities.

2.4 Conditional Probability

Events do not usually happen in isolation. Things that have happened in the past might influence what will happen in the future. There is a certain probability that the coin will land on heads: $P(H)$. But if I know something happened to the coin before the toss – maybe I put some weights in it, event W – then the probability of heads will change. To handle this scenario, we introduce the concept of “conditional probability”. In our scenario, the notation is $P(H|W)$. $P(H|W)$ is the probability of the coin landing on heads – H – given that event W happened.

This view of probability is particularly in line with the Bayesian interpretation, as what you call “prior” is really a synthesis of everything that happened in the past. That is not to say that a frequentist cannot understand conditional probabilities: they can, they just take the usual approach of simply observing what happens before/after something and be done with it.

Conditional probabilities enable you to make a nice set of inferences. Figure 2.3 shows the most basic ones. If you measure $P(H|W)$,

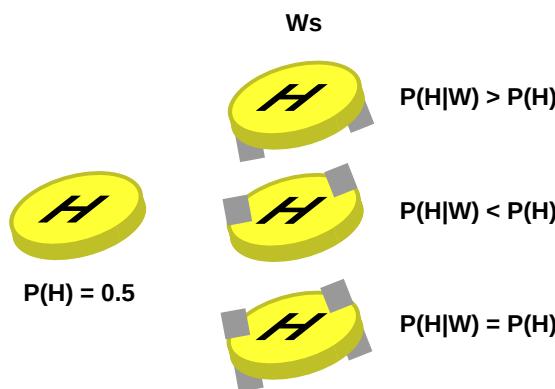


Figure 2.3: The baseline probability of H is 0.5. When you add feet to the coin (W) the coin is more likely to land on the opposite side. Thus, $P(H|W) \neq P(H)$ and the two events are not independent – unless you add feet on both sides as in the bottom example.

you can figure out what event W did to the coin. If $P(H|W) > P(H)$, it means that adding the weight to the coin made it more likely to land on heads. $P(H|W) < P(H)$ means the opposite: your coin is loaded towards tails. The $P(H|W) = P(H)$ case is equally interesting: it means that you added the weight uniformly and the odds of the coin to land on either side didn't change.

This is a big deal: if you have two events and this equation, then you can conclude that the events are independent – the occurrence of one has no effect on the occurrence of the other⁴. This should be your starting point when testing a hypothesis: the null assumption is that there is no relation between an outcome (landing on heads) and an intervention (adding a weight). “Unless,” Mr. Bayes says, “You have a strong prior for that to be the case.”

Reasoning with conditional probabilities is trickier than you might expect. The source of the problem is that, typically, $P(H|W) \neq P(W|H)$, and often dramatically so. Suppose we’re tossing a coin to settle a dispute. However, I brought the coin and you think I might be cheating. You know that, if I loaded the coin, the probability of it landing on heads is $P(H|W) = 0.9$. However, you can’t see nor feel the weights: the only thing you can do is tossing it and – presto! – it lands on heads. Did I cheat?

Naively you might rush and say yes, there’s a 90% chance I cheated. But that’d be wrong, because the coin already had a 50% chance of landing on heads without any cheating. Thus $P(H|W) \neq P(W|H)$, and what you really want to estimate is the probability I cheated given that the coin landed on heads: $P(W|H)$. How to do so, using what you know about coins ($P(H)$) and what you know about my integrity ($P(W)$), is the specialty of Bayes’ Theorem.

⁴ Note that here I’m talking about *statistical* independence, which is not the same as *causal* independence. Two events could be statistically dependent without being causally dependent. For instance, the number of US computer science doctorates is statistically dependent with the total revenue of arcades (<http://www.tylervigen.com/spurious-correlations>). This is what the mantra “correlation does not imply causation” means: correlation is mere statistical dependence, causation is causal dependence, and you shouldn’t confuse one with the other. You should check [Pearl and Mackenzie, 2018] to delve deeper into this.

2.5 Bayes’ Theorem

Bayes’ Theorem is an almost magical formula that allows you to estimate the probability of an event based on your priors. Keeping the example of cheating on a coin toss, we want to estimate the probability I cheated and rigged the coin so it lands on heads after we tossed it and it indeed landed on heads – in mathematical notation: $P(W|H)$. To do so, you need to have priors. You need to know: what’s the probability of heads for all coins in the world (whether they are rigged or not, $P(H)$), what’s the probability I rigged the coin ($P(W)$), and what is the probability of obtaining heads on a rigged coin ($P(H|W)$). Without further ado, here’s one of the most important formulas in human history:

$$P(W|H) = \frac{P(H|W)P(W)}{P(H)}.$$

Figure 2.4 shows a graphical proof of the theorem. When trying to derive $P(W|H)P(H)$, we realize that's identical to $P(H|W)P(W)$, from which Bayes' theorem follows.

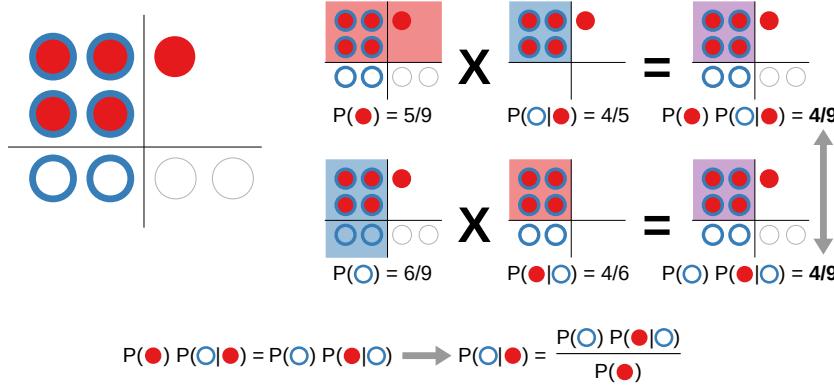


Figure 2.4: The table on the left shows the occurrence of all possible events: red circles (5), blue borders (6), red circles with blue borders (4) and neither (2).

I already told you that I'm a pretty good coin rigger ($P(H|W) = 0.9$). For the sake of the argument, let's assume I'm a very honest person: the probability I cheat is fairly low ($P(W) = 0.3$).

Now, what's the probability of landing on heads ($P(H)$)? $P(H)$ is trickier than it appears, because we're in a world where people might cheat. Thus we can't be naive and saying $P(H) = 0.5$. $P(H)$ is 0.5 if rigging coins is impossible. It's more correct to say $P(H| - W) = 0.5$: a non rigged coin (if W didn't happen, which we refer to as $-W$) is fair and lands on heads 50% of the times. The real $P(H)$ is $P(H| - W)P(-W) + P(H|W)P(W)$. In other words: the probability of the coin landing on heads is the non rigged heads probability if I didn't rig it ($P(H| - W)P(-W)$) plus the rigged heads probability if I rigged it ($P(H|W)P(W)$).

The probability of not cheating $P(-W)$ is equal to $1 - P(W)$. This is because cheating and non cheating are mutually exclusive and either of the two *must* happen. Thus we have $\Omega = \{W, -W\}$. Since $P(\Omega) = 1$ and $P(W) = 0.3$, the only way for $P(W, -W)$ to be equal to 1 is if $P(-W) = 0.7$.

This leads us to: $P(H) = P(H| - W)P(-W) + P(H|W)P(W) = 0.5 \times 0.7 + 0.9 \times 0.3 = 0.62$. Shocking.

The aim of Bayes' theorem is to update your prior about me cheating ($P(W)$) given that, suspiciously, the toss went in my favor ($P(W) \rightarrow P(W|H)$). Plugging in the numbers in the formula:

$$P(W|H) = \frac{0.9 \times 0.3}{0.62} = 0.43.$$

A couple of interesting things happened here. First, since the event went in my favor, your prior about me possibly cheating got updated. Specifically, the event became more likely: from 0.3 to 0.43. Second, even if my success probability after cheating is very high, it is still more likely that I didn't cheat, because your prior about my lack of integrity was low to begin with.

This second aspect is absolutely crucial and it's easy to get it wrong in everyday reasoning. The textbook example is the cancer diagnosing machine. Let's say that 0.1% of people develop a cancer, and we have this fantastic diagnostic machine with an accuracy of 99.9%: the vast majority of people will be diagnosed correctly (positive result for people with cancer and negative for people without). You test yourself and the test is positive. What's your chance of having cancer? 99.9% accuracy is pretty damning, but before working on your last will, you apply Bayes' Theorem:

$$P(C|+) = \frac{0.999 \times 0.001}{0.999 \times 0.001 + 0.001 \times 0.999} = 0.5.$$

The probability you have cancer is *not* 99.9%: it's a coin toss! (Still bad, but not *that* bad).⁵

The real world is a large and scary environment. Many different things can alter your priors and have different effects on different events. The way a Bayesian models the world is by means of a Bayesian network: a special type of network connecting events that influence each other. Exploring a Bayesian network allows you to make your inferences by moving from event to event. I talk more about Bayesian networks in Section 6.4.

2.6 Stochasticity

Colloquially, a stochastic process is one or more random variables that change their values over time. The quintessential stochastic process is Brownian motion. Brown observed very light pollen particles on water changing directions, following a *stochastic* path that seemed governed purely by randomness. Interestingly, this problem was later solved by Einstein in one of his first contributions to science⁶, working off important prior work⁷. He explained the seemingly random changes of direction as the result of collision between the pollen and water molecules jiggling in the liquid.

When you have a stochastic process, there is an almost infinite set of results. The pollen can follow potentially infinite different paths. When you observe an actual grain, you obtain only one of those paths. The observed path is called a realization of the process. Figure 2.5 shows three of such realizations, which should help you visualize

⁵ Of course, in the real world, if you took the test it means you thought you might have cancer. Thus you were not drawn randomly from the population, meaning that you have a higher prior that you had cancer. Therefore, the test is more likely right than not. Bayes' theorem doesn't endorse carelessness when receiving a bad news from a very accurate medical test.

⁶ Albert Einstein. Über die von der molekularkinetischen theorie der wärme geforderte bewegung von in ruhenden flüssigkeiten suspendierten teilchen. *Annalen der physik*, 4, 1905

⁷ Louis Bachelier. Théorie de la spéculation. In *Annales scientifiques de l'École normale supérieure*, volume 17, pages 21–86, 1900

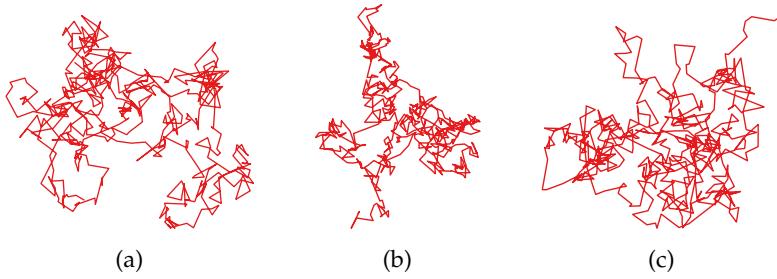


Figure 2.5: Three realizations of a Brownian stochastic motion on a two dimensional plane.

the intrinsic randomness of the change of direction.

Whenever you encounter the word “stochastic” in this book or in a paper, we’re referring to a process governed by these dynamics. For instance, a stochastic matrix is a matrix whose rows and/or columns sum up to one. We call it stochastic, because such matrices are routinely used to describe stochastic processes. By having their rows to sum to one, you can interpret each entry of the row as the *probability* of its corresponding event. The row in which you are tells you the current state of the process, the column tells you the next possible state, and the cell value tells you the probability of transitioning to each of the next possible states (column) given the current state (row). In other words, it is the probability of one possible realization of a single step in a stochastic process. In network science, you normally have stochastic adjacency matrices, which are the topic of Section 8.2.

| | | | | | | | | |
|------|------|------|------|------|-----|------|------|------|
| 0.3 | 0.2 | 0.08 | 0 | 0.07 | 0.1 | 0 | 0.07 | 0.2 |
| 0.1 | 0.3 | 0.04 | 0.04 | 0.05 | 0.2 | 0.04 | 0.09 | 0.1 |
| 0.08 | 0.06 | 0.3 | 0.06 | 0 | 0.2 | 0.06 | 0.06 | 0.1 |
| 0 | 0.08 | 0.08 | 0.2 | 0 | 0.2 | 0.08 | 0.2 | 0.08 |
| 0.07 | 0.07 | 0 | 0 | 0.3 | 0.1 | 0.08 | 0.2 | 0.2 |
| 0.07 | 0.1 | 0.1 | 0.06 | 0.07 | 0.3 | 0.04 | 0.09 | 0.08 |
| 0 | 0.08 | 0.08 | 0.08 | 0.1 | 0.1 | 0.2 | 0.08 | 0.2 |
| 0.05 | 0.09 | 0.04 | 0.1 | 0.1 | 0.1 | 0.04 | 0.3 | 0.1 |
| 0.1 | 0.1 | 0.08 | 0.03 | 0.1 | 0.1 | 0.08 | 0.1 | 0.3 |

Figure 2.6 is a stochastic matrix⁸. The rows tell you your current state and the columns tell you your next state. If you are in the first row, you have a 30% probability of remaining in that state (the value of the cell in the first row and first column is 0.3). You have a 20% probability of transitioning to state two (first row, second column), 8% probability of transitioning to state three, and so on.

Figure 2.6: A right stochastic matrix.

⁸ Specifically, it is a right stochastic matrix: the rows sum to one, although there’s a bit of rounding going on. In a left stochastic matrix, the columns sum to one.

2.7 Markov Processes

It should be clear now that, even if the next state is decided by a random draw, a stochastic process isn’t necessarily uniformly random. In Brownian motion, the next position is determined by your

previous position as well as a random kick. This observation is at the basis of a fundamental distinction between three flavors of stochastic processes, which are the most relevant for network science. The three flavors are: Markov processes, non-Markov processes, and higher-order Markov processes.

In a Markov process, the next state is exclusively dependent on the current state and nothing else. No information from the past is used: only the present state matters. That is why a Markov process is usually called “memoryless”. The stochastic process I described when discussing Figure 2.6 is a typical Markov process. The only thing we needed to know to determine the next state was the current state: in which row are we?

The classical Markov process in network science is the random walk. A random walker simply chooses the next node it wants to occupy, and its options are determined solely by the node it is currently occupying. Rather surprisingly, random walks are one of the most powerful tools in network science and have been applied to practically everything. I’m going to introduce them properly in Chapter 11, but they will pop up throughout the book – for instance, in community discovery (Part X) and in network sampling (Chapter 29). Figure 2.7 shows an example of a random walk. As you can see, we start from the leftmost node. From that state, reaching the two rightmost ones is impossible because the nodes are not connected. Only when you transition to another state, new states become available.

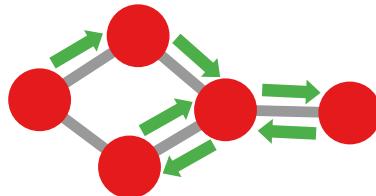


Figure 2.7: A random walk. The green arrows show the state transitions.

A bit more formally, let’s assume you indicate your state at time t with X_t . You want to know the probability of this state to be a specific one, let’s say x . x could be the id of the node you visit at the t -th step of your random walk. If your process is a Markov process, the only thing you need to know is the value of X_{t-1} – i.e. the id of the node you visited at $t - 1$. In other words, the probability of $X_t = x$ is $P(X_t = x | X_{t-1} = x_{t-1})$. Note how $X_{t-2}, X_{t-3}, \dots, X_1$ aren’t part of this estimation. You don’t need to know them: all you care about is X_{t-1} .

On the other hand, a non-Markov process is a process for which knowing the current state doesn’t tell you anything about the next possible transitions. For instance, a coin toss is a non-Markov process. The fact that you toss the coin and it lands on heads tells you nothing

about the result of the next toss – under the absolute certainty that the coin is fair. The probability of $X_t = x$ is simply $P(X_t = x)$: there's no information you can gather from your previous state.

Finally, we have higher-order Markov processes. Higher-order means that the Markov process now has a memory. A Markov process of order 2 can remember one step further in the past. This means that, now, $P(X_t = x|X_{t-1} = x_{t-1}, X_{t-2} = x_{t-2})$: to know the probability of $X_t = x$, you need to know the state value of X_{t-2} as well as of X_{t-1} . More generally, $P(X_t = x|X_{t-1} = x_{t-1}, X_{t-2} = x_{t-2}, \dots, X_{t-m} = x_{t-m})$, with $m \leq t$.

The classical network examples of a higher order Markov process is the non-backtracking random walk (Figure 2.8). In a non-backtracking random walk, once you move from node u to node v , you are forbidden to move back from v to u . This means that, once you are in v , you also have to remember that you came from u . Higher order Markov processes are the bread and butter of higher order network problems, which is the topic of Chapter 34.

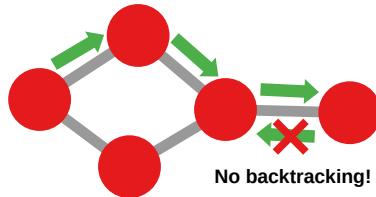


Figure 2.8: A non-backtracking random walk. The green arrows show the state transitions.

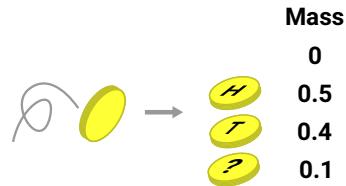
2.8 Alternatives to Probability Theory

When it comes to dealing with uncertainty, probability theory is not the only game in town. Here I briefly present two alternatives. These will be handy when we focus specifically on probabilistic networks in Chapter 28.

Dempster-Shafer's Theory of Evidence

In Dempster-Shafer's theory of evidence (DST) we take the key insight from Bayes and we turn it up to eleven. One thing that underlies Bayesian thought is that probabilities are subjective. If two people have different priors, say A and B such that $P(A) \neq P(B)$, then they will disagree on the probability of event C , which will depend on the priors. This doesn't happen with a frequentist framework, because there are no priors and $P(C)$ is based on objective data available to everyone. However, besides this subjectivity, Bayes still uses the axioms and the rules of probability theory.

DST is a generalization of probability theory, which moves from exact probabilities to probability intervals. Its central parts are *beliefs* and *plausibilities*, and these two things don't necessarily behave like probabilities^{9,10}.



Starting with beliefs, the first thing you need is a *degree of belief*, which estimates your ability to prove a set of beliefs¹¹. This degree of belief is quantified by a function which is conventionally called its Mass function. Figure 2.9 shows a relatively simple example when tossing a coin – slightly loaded on heads. The distinction between Mass in DST and classical probability is that it considers the case “we don't know whether heads or tail” as distinct from “heads” and “tail”. In probability theory, you wouldn't make this distinction, because no other outcome than heads or tails can happen, even if for some reason you don't know the outcome. But in DST you want to model this, because we're talking about the ability of proving our statement, so we need to specifically take into account the situation in which we don't actually know the result – e.g., if the coin rolled under the sofa and we can't see it. In that case, we don't have any evidence to say that the coin landed on heads or tail.

In summary, if $\Omega = \{H, T\}$, then $p(\Omega) = p(H) + p(T) = 1$, but $Mass(\Omega) \neq Mass(H) + Mass(T)$: $Mass(\Omega)$ is less trivial and actually informative – it is the amount of uncertainty we have about the outcome of the event given the imperfection of our evidence. So the Mass function is basically giving all the available evidence a probability and obeys the following two rules:

1. $Mass(\emptyset) = 0$, and

2. $\sum_{X \in 2^\Omega} Mass(X) = 1$.

Here, 2^Ω means all possible subsets of Ω , which in my simple case are: \emptyset , $\{H\}$, $\{T\}$, and $\{H, T\}$. The first property means that it's impossible to prove that nothing happened – we know we tossed the coin, it must have landed on something. That's why the ugly coin toss drawing on the left of Figure 2.9 is necessary: to show we performed the tossing. The second property means that Ω contains all possible answers to our question – so what's actually true must be a subset of Ω .

⁹ Arthur Dempster. Upper and lower probabilities induced by a multi-valued mapping. *Annals of Mathematical Statistics*, 38, 1967

¹⁰ Glenn Shafer. *A mathematical theory of evidence*, volume 42. Princeton university press, 1976

Figure 2.9: An illustration of Mass in DST. After tossing a coin (left) each each subset of Ω obtains a value.

¹¹ Judea Pearl. Reasoning with belief functions: An analysis of compatibility. *International Journal of Approximate Reasoning*, 4(5-6):363–389, 1990

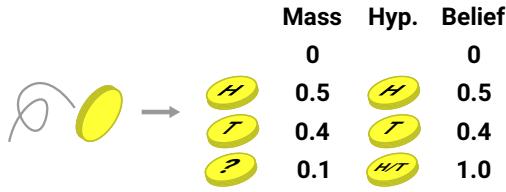


Figure 2.10: An illustration of Belief in DST, based on the Mass from Figure 2.9. Note how Belief of a subset of size 1 is equal to its Mass.

Now that you know the probabilities of all possible outcomes, you must make a hypothesis which is a set of potential outcomes. To estimate your ability to prove your hypothesis, you sum all the Mass values of all the subsets of your hypothesis. This is the Belief function, and you can see how it works in Figure 2.10. If your hypothesis has no support in the gathered evidence then its Belief value is zero, while if it is absolutely certain then Belief evaluates to one. So Belief tells you how likely your hypothesis – or any of its subsets – is to be proven given the available evidence.

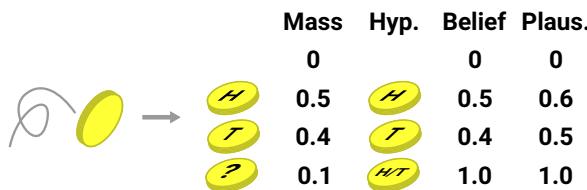


Figure 2.11: An illustration of Plausibility in DST, based on the Mass and Belief Figure 2.10.

DST also allows to compute the Plausibility function, which is an upper bound of Belief. In practice, you can tally up all the evidence of your hypothesis not containing the truth and take the inverse of it. Figure 2.11 shows how it works. Plausibility is basically estimating how much your hypothesis can survive an attempt to prove it false. A handy rule to remember is that $\text{Plausibility}(X) = 1 - \text{Belief}(\bar{X})$, where \bar{X} is the complementary set of X – the plausibility of something is the opposite of your belief of that something being proven false.

We go through the trouble of defining these things because DST has some advantages. For instance, there are some operations you can do with the Belief and Plausibility functions – which we are not going to see here – but allow you to work with different hypotheses in conflict. Classical probability theory is ill suited to handle these cases. For instance, suppose that the ice cream shop has three flavors: chocolate, strawberry, and vanilla. We want to share an ice cream and my preference is 99% chocolate and 1% vanilla, while yours are 99% strawberry and 1% vanilla. We should obviously go for vanilla, and DST agrees, but if we took a probabilistic approach ignoring DST, you might instead say that vanilla is the least likely solution, and we would end up with either chocolate or strawberry, much

to the distress of either of us. For instance by naively aggregating probabilities as 49.5% for each strawberry and chocolate. To be fair, DST also ends up saying weird things occasionally, which has led researchers to formulate ways to turn it into computable functions that are different from the ones I explained¹².

Moreover, probability theory must assign a probability to an event, even if there is no evidence for it, while DST can simply give it zero Mass. For instance, if we have a potentially loaded die, in probability theory using a Bayesian approach we must start by assigning a prior probability of 1/6 to all outcomes *even if we have zero evidence for it*. In DST, you'd give them Mass zero instead, and Mass one to Ω and then start gathering evidence.

Fuzzy Logic

In probability theory, you only deal with boolean events, whose truth values can either be zero or one. Either something is false or it is true. The coin either landed on heads or on tails. In fuzzy logic, you work with something different. Things can have degrees of truthiness, which is to say we assign them a truth value between zero and one. If it is zero, we're certain that a statement is false, if it is one we're certain it is true, and if it is a value in between then there is some vagueness about whether it is true or false¹³.

For instance, at the moment of writing this paragraph I am 39 years old. Is that young or old? Well, you could line up 100 people and ask them this question. Maybe 60 will say that I'm young, 39 will say that I'm old (I'm so insecure I am disrespected even in my thought experiments), and 1 will say something else. In probability theory you could model this as something like: there's a 39% chance a random person will call me old (hey!). But in fuzzy logic you'd do something different. You could say that I belong to both the sets of young people and old people, with different strengths. I'm 60% young and 39% old – I frankly don't know if that's an improvement over the alternative.

The consequences of this difference lead to different outcomes when working with fuzzy logic. We'll see a basic common example¹⁴, but know that there are alternative ways to implement fuzzy logic¹⁵. Let's assume that the degree to which a person belongs to an age class depends on their age, following the function I draw in Figure 2.12.

For the age I highlight, probability theory can say the following things:

- The probability of picking somebody who could call me both young and old (assuming I ask multiple time and people can

¹² Kari Sentz and Scott Ferson. Combination of evidence in dempster-shafer theory. 2002

¹³ Petr Hájek. *Metamathematics of fuzzy logic*, volume 4. Springer Science & Business Media, 2013

¹⁴ Ebrahim H Mamdani. Application of fuzzy algorithms for control of simple dynamic plant. In *Proceedings of the institution of electrical engineers*, volume 121, pages 1585–1588. IET, 1974

¹⁵ Tomohiro Takagi and Michio Sugeno. Fuzzy identification of systems and its applications to modeling and control. *IEEE transactions on systems, man, and cybernetics*, (1):116–132, 1985

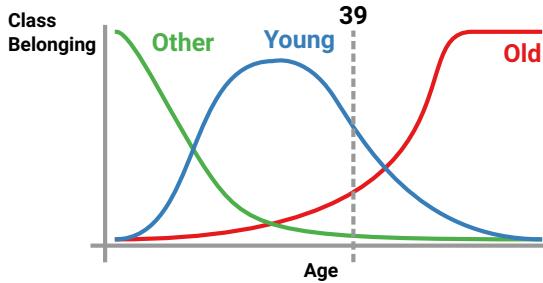


Figure 2.12: An illustration of fuzzy logic. The degrees of belonging (y axis) to different age classes (line color) for a given age (x axis).

change their mind independently from what they said the first time) is $P(Y \cap O) = P(Y)P(O) = 0.234$;

- The probability of picking somebody who will call me either young or old is $P(Y \cup O) = P(Y) + P(O) = 0.99$;
- The probability of somebody not calling me young is $P(\bar{Y}) = 1 - P(Y) = 0.61$.

But in fuzzy logic we have:

- My belonging to the class of people who are both young and old is $P(Y \cap O) = \min(P(Y), P(O)) = 0.39$ – it can't be any higher than my minimum belonging, because to fully belong to the young-old class I must be fully young and fully old;
- My belonging to the class of people who are either young or old is $P(Y \cup O) = \max(P(Y), P(O)) = 0.6$ – it can't be any lower than my maximum belonging, because if I am fully old then I am also fully-young-or-fully-old;
- My belonging to the class of people who are not young is $P(\bar{Y}) = 1 - P(Y) = 0.61$.

2.9 Summary

1. Probability theory gives you the tools to make inferences about uncertain events. We often use a frequentist approach, the idea that an event's probability is approximated by the aggregate past tests of that event. Another important approach is the Bayesian one, which introduces the concept of priors: additional information that you should use to adjust your inferences.
2. Probabilities are non-negative estimates. The set of all possible outcomes has a probability sum of one. Summing two probabilities tells you the probability of either of two independent outcomes to happen.

3. The conditional probability $P(A|B)$ tells you the probability of an outcome A given that you know another outcome B happened. If $P(A|B) \neq P(A)$ then the two outcomes are not independent. Bayes' Theorem allows you to infer $P(A|B)$ from $P(B|A)$.
4. When we track the change over time of one or more random variables, we're observing a stochastic process. Markov processes are stochastic processes whose status exclusively depends on the status of the system in the previous time step.
5. There are alternative to probability theory when working with uncertainty. Dempster-Shafer's theory of evidence allows to work with the degrees of beliefs in specific hypotheses, while fuzzy logic allows for multiple things to be a little bit true at the same time.

2.10 Exercises

1. Suppose you're tossing two coins at the same time. They're loaded in different ways, according to the table below. Calculate the probability of getting all possible outcomes:

| $p_1(H)$ | $p_2(H)$ | H-H | H-T | T-H | T-T |
|----------|----------|-----|-----|-----|-----|
| 0.5 | 0.5 | | | | |
| 0.6 | 0.7 | | | | |
| 0.4 | 0.8 | | | | |
| 0.1 | 0.2 | | | | |
| 0.3 | 0.4 | | | | |

2. 60% of the emails hitting my inbox is spam. You design a phenomenal spam filter which is able to tell me, with 98% accuracy, whether an email is spam or not: if an email is not spam, the system has a 98% probability of saying so. The filter knows 60% of emails are spam and so it will flag 60% of my emails. Suppose that, at the end of the week, I look in my spam box and see 963 emails. Use Bayes' Theorem to calculate how many of those 963 emails in my spam box I should suspect to be non-spam.
3. You're given the string: "OCZ XJMMZXO VINRZM". Each letter follows a stochastic Markov process with the rules expressed by the table at <http://www.networkatlas.eu/exercises/2/3/data.txt>. Follow the process for three steps and reconstruct the correct answer. (Note, this is a Caesar cipher¹⁶ with shift 7 applied three times, because the Caesar cipher is a Markov process).
4. Suppose that we are examining a painting and we're trying to date it with the century when it was produced. Find out the Belief and Plausibility values for all hypotheses given the following Mass

¹⁶ https://en.wikipedia.org/wiki/Caesar_cipher

estimation (note that, by definition $\Omega = \{\text{XIV}, \text{XV}, \text{XVI}\}$ must have Belief and Plausibility equal to one):

| Hypothesis | Mass | Belief | Plausibility |
|------------------------------|------|--------|--------------|
| \emptyset | 0.00 | | |
| XIV | 0.16 | | |
| XV | 0.04 | | |
| XVI | 0.21 | | |
| $\{\text{XIV}, \text{XV}\}$ | 0.34 | | |
| $\{\text{XV}, \text{XVI}\}$ | 0.16 | | |
| $\{\text{XIV}, \text{XVI}\}$ | 0.08 | | |
| Ω | 0.01 | 1 | 1 |

3

Statistics

In Chapter 2 I explained the basic concepts of probability theory you need to understand to be a good network scientist. This chapter focuses on basic statistical concepts that are also necessary to analyze your networks. The disclaimer I put at the beginning of Chapter 2 also holds here: statistics is much more vast and complicated than what I present here. This chapter is emphatically *not* a substitute for a proper statistics textbook^{1,2}, which you could use to study this stuff further. You'll get a sense of how powerful statistics is³, in that it can allow you to support any point. As the saying goes: there are lies, damn lies, and statistics.

The main difference between probability and statistics is that you can do a lot of work in probability theory without actually looking at any data. When data takes the center stage, you enter in the world of statistics. Statistics covers more than simply describing your data: you should think in statistical terms also when collecting, cleaning, validating your data. But here we ignore all that and we focus on the tools that allow you to say something interesting about your data, assuming you did a good job collecting, cleaning, and validating it.

3.1 Summary Statistics

Mean & Median

One common use of statistics is to give a quick description of what is in the data. The most classical task is to try and figure out what are the values you'd expect to find if you were to look directly at the data. For instance, if you want to know the height of the average human, you might want to calculate the mean height: $\mu(H) = \sum_i H_i / |H|$, where H_i is the height of one human. The mean in this case would tell you what you'd expect to see if you were to measure the height of a random person.

However, that works for height because height is normally dis-

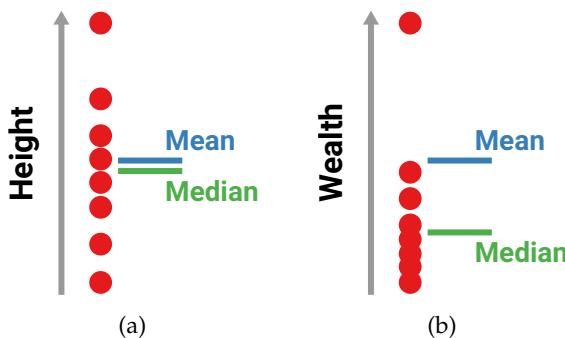
¹ Charles Wheelan. *Naked statistics: Stripping the dread from the data*. WW Norton & Company, 2013

² Richard McElreath. *Statistical rethinking: A Bayesian course with examples in R and Stan*. Chapman and Hall/CRC, 2018

³ Darrell Huff. *How to lie with statistics*. Penguin UK, 1954

tributed – meaning that the average value is actually the mean and values farther from the mean are progressively more rare. We'll see what a normal distribution looks like in Section 3.2. The same section will also tell you that not all variables distribute like that: in some cases the mean is actually not a great approximation of your average (or “typical”) case. Wealth is like that: a tiny fraction of people own vastly more than the majority. In this case, the median gives you a better idea⁴. The median tells you the value that splits the data in two equally populated halves: 50% of the points are below the median and 50% of the points are above.

Figure 3.1 shows that the mean and the median can be quite different. In network science, we use the mean extensively – even though maybe we shouldn't. When we'll talk about the number of connections a node has in a network (Chapter 9), we'll see we routinely take its “average” by calculating its mean. But connection counts in real networks typically do not follow a neat normal distribution. These distributions tend to look more like Figure 3.1(b) than Figure 3.1(a). So, perhaps the mean count of connections is not the most meaningful thing you can calculate.



⁴ David J Sheskin. *Handbook of parametric and nonparametric statistical procedures*. Chapman and hall/CRC, 2003

Figure 3.1: The mean (blue) and median (green) of two different variables (y axis): (a) normally distributed height; (b) skewed distribution of wealth.

This disconnect between the mean and the typical case is true for the arithmetic mean I show here, but there are other types of means – such as geometric or harmonic – which can take into account some special properties of the data⁵.

Variance & Standard Deviation

When we deal with an average observation, we might want to know not only its expected value, but also how much we expect it to differ from the actual average value. Even if the average human height is, let's say, 1.75 meters, we could think of two radically different populations. In the first, almost everyone is more or less 1.75 and heights don't vary much. In the other, the opposite is true: the average is still 1.75, but people could be anything between 1 meter and 2.5 meters. So the heights in this second population vary much more. We

⁵ Philip J Fleming and John J Wallace. How not to lie with statistics: the correct way to summarize benchmark results. *Communications of the ACM*, 29(3):218–221, 1986

need to have a tool allowing us to distinguish these two populations. Since the difference is all about how much the heights *vary*, we call this measure *variance*. Variance (and standard deviation) helps you quantify how dispersed your values are away from the mean.

Variance is almost literally the mean difference from the mean. The only tweak is that we take the square of this difference: $\text{var}(H) = \mu((H - \mu(H))^2)$. We take the square because we don't want values below the mean to cancel out values above the mean. The standard deviation is simply the square root of the variance: $\sigma(H) = \sqrt{\text{var}(H)}$. The advantage of the standard deviation is that it ends up having the same units as the original variable. For example, for heights measured in cm, the variance will be in cm^2 , but the standard deviation will be in cm again. Figure 3.2 shows what it looks like to have different variances for variables with the same mean.

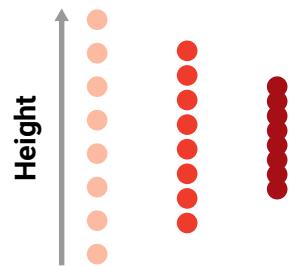


Figure 3.2: The variances of three different populations, going from left (high variance, bright red) to right (low variance, dark red).

The concept of variance will be important when we talk about data dimensionality reduction (Section 5.6) and degree distributions (Section 9.3). We will even see how to modify its definition to create a notion of network variance (Section 47.5)

Distribution & Skewness

Knowing how skewed your data is can be quite important – in wealth distribution, how skewed the data is equals how screwed people are. Variance and standard deviation can help you quantify this. There are different formulas and different terms to talk about skewness⁶, but for our purposes we limit ourselves to a bit of terminology.

First: what actually is a distribution? I've used this term in an intuitive way without really defining it. Let's do it here. When you perform an experiment, or observe a stochastic process, you have many possible outcomes. For instance, you're measuring the heights of all people in a country and so your possible outcomes are all the possible heights a person can have. Sometimes, you're not interested on the frequency of measuring a specific outcome – say 175 cm. Sometimes, you want to study all possible outcomes together, to determine which is more likely, what you could expect when you

⁶ Paul T Von Hippel. Mean, median, and skew: Correcting a textbook rule. *Journal of statistics Education*, 13(2), 2005

measure more people, if there are maximums and minimums you don't expect to ever exceed, and so on. This is the task of a distribution. A distribution is a function that, for each outcome in the set of all possible ones (called the "sample space"), tells you how many times you measured a given outcome. Figure 3.3 shows a vignette on how to interpret a plot showing you a distribution.

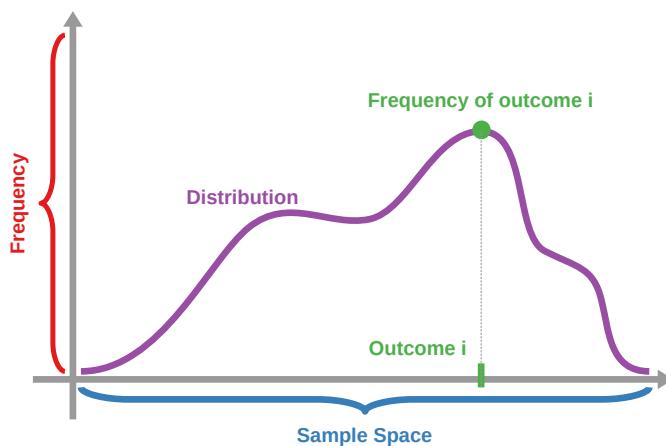


Figure 3.3: A distribution, connecting every possible outcome in the sample space (x axis) to a frequency (y axis).

Skewness is a property of a distribution, it measures its symmetry. A symmetric distribution has no skewness, and any asymmetry will create a non-zero skewness value: positive if the skewedness is on the right and negative if it is on the left – see Figure 3.4.

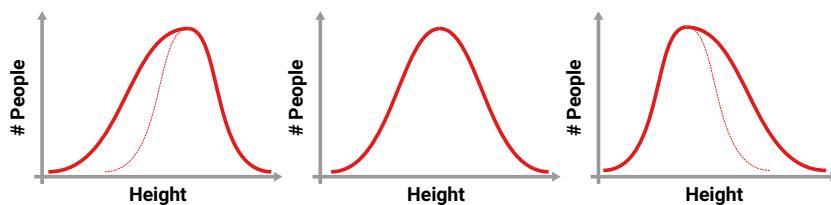


Figure 3.4: Three distributions with different skewness: (a) the values lower than the average are more likely, (b) no skewness, (c) the values higher than the average are more likely.

An important related concept to skewness is the heavy-tailed distribution. There are two types of heavy-tailed distributions that interest network scientists, because they're often observed in real world networks. They are the long tail and the fat tail⁷. In a long tail, you can find arbitrarily large outliers: that means the very highest value can be many times larger than the second-highest one. You might know these from the popular concept of the black swan⁸. In these kinds of distributions, observations can happen that are much more extreme than anything we have seen so far. You'll see a network example in Chapter 9, when we'll see it is common for nodes in real network to have a long tail in the number of connections attached to them. With a fat tail, you still have outliers that can be many times

⁷ In case you were wondering: yes, statisticians body-shame distributions.

⁸ Nassim Nicholas Taleb. *The black swan: The impact of the highly improbable*, volume 2. Random house, 2007

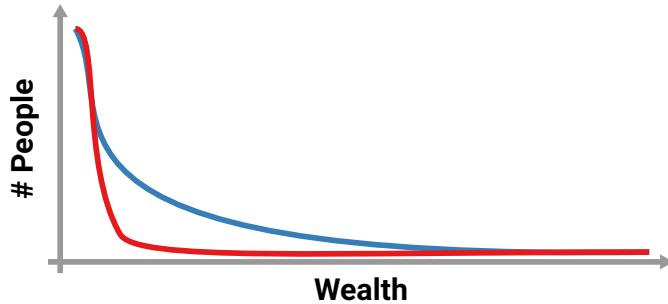


Figure 3.5: A long tail (red) and a fat tail (blue).

over the average value. However, these outliers are more common and less extreme. Figure 3.5 shows a graphical example.

If we're talking wealth, a long tail world is a world with a single Jeff Bezos and everybody else works in an Amazon warehouse. In a fat tail world, Jeff might not be quite as rich, but there are a few billionaire friends to keep him company.

3.2 Important Distributions

Chapter 9 will drill in your head how important distributions are for network science, so it pays off to become familiar with a few of them. First, let's make an important distinction. There are two kinds of distributions, depending on the kinds of values that their underlying variables can take. There are discrete distributions – for instance, the distribution of the number of ice cream cones different people ate on a given day. And there are continuous distributions – for instance the distances you rode on your bike on different days. The difference is that the former has specific values that the underlying variable can take (you may have eaten two or three ice cream cones, but 2.5 is not an option), the latter can take any real value as an outcome. In the first discrete case, we call the distribution a "mass function". In the second case, we call it a "density function".

Figure 3.6 shows some stylized representations of the most important distributions you should pay attention to, which are:

- **Uniform:** in this distribution each event is equally likely. This distribution can be both discrete or continuous. In the discrete case, if you have n possible events, each occurs with probability $p = 1/n$. You get a discrete uniform distribution if you look at the number on a ball extracted from an urn, where all balls in the urn are identified by distinct, progressive numbers without gaps. A continuous uniform distribution could be the amount of time you have to wait for the next drop to come from a leaky faucet that drips once a minute: if you haven't seen the last drop falling, the wait time could be any time between 0 and 60 seconds.

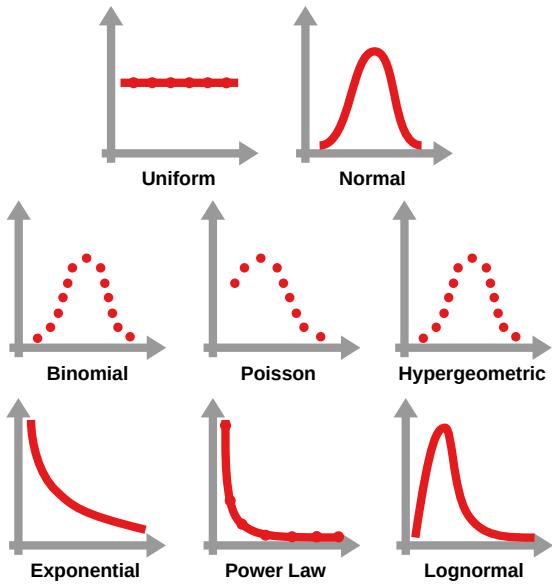


Figure 3.6: A stylized representation of the most common distributions you'll encounter as a network scientist. Solid lines show continuous distributions, while dots show discrete ones – note that some distributions can be both (dotted lines).

- **Normal (or Gaussian):** this is a very common distribution for continuous variables. The classical example is the distribution of people's heights: most people are of average height, and larger and larger deviations from the average get steadily less likely.
- **Binomial:** this is a discrete distribution, in which you do n experiments, each with success probability p , and you calculate the probability of having n' successes. For instance, suppose you take five balls from an urn containing 50 red and 50 green balls – each time putting the ball you extracted back into the urn. Let's say we count getting a green ball as a “success” here. The number of times you got 0, 1, 2, 3, 4 or 5 green balls over a bunch of trials would follow a binomial distribution.
- **Hypergeometric:** this is yet another discrete probability function. It is very similar to a binomial distribution. Where the binomial described the number of “successes” in an extraction-with-replacement urn game, the hypergeometric describes the more common case of extraction-without-replacement. When you extract a ball from the urn, you don't put it back. It is mathematically less tractable, but much more useful. This is used especially for the task of network backboning (Chapter 27).
- **Poisson:** this is another discrete distribution, which is the number of successes in a given time interval, assuming that each success arrives independently from the previous ones. For instance, the number of meteorites impacting on the moon each year will have a Poisson distribution. Interestingly, many examples commonly

mentioned for explaining a Poisson distribution (number of admittances to a hospital in an hour, number of emails written in an hour, and so on) aren't actually Poisson distributions, because of the "burstiness" of human behavior⁹.

- **Exponential:** the exponential distribution is a continuous distribution modeling cases in which the probability of something happening is not dependent of how much time has passed since you starting observing the phenomenon. For instance, if there's an epidemics out, the amount of time you have been infection-free bears no weight in determining your probability of being infected, if exposed. This is the reason why this distribution is sometimes called "memoryless" or that it "doesn't age".
- **Power law:** a power law can be both a discrete or a continuous distribution. It describes the relationship between two quantities, the second quantity changes as a power of the first. One practical consequence is that, if you were given a power law plot without axis labels, you would not be able to tell where you are in the distribution, because the slope of the line always looks the same no matter how much you zoom in or out, or whether you're on the head or the tail. An example of discrete power law is Zipf's law¹⁰ recording the frequency of words in a document against their frequency rank. We'll see more than you want to know about power laws when talking about fitting degree distributions in Section 9.3.
- **Lognormal:** a lognormal distribution is the distribution of a continuous random variable whose logarithm follows a normal distribution – meaning the logarithm of the random variable, not of the distribution. This is the typical distribution resulting from the multiplication of two independent random positive variables. If you throw a dozen 20-sided dice and multiply the values of their faces up, you'd get a lognormal distribution. It's very tricky to tell this distribution apart from a power law, as we'll see.

Sometimes, rather than looking at the mass/density functions, it's more useful to look at their cumulative versions. In practice, you want to ask yourself what is the number of – say – x or *fewer* successes. Each distribution changes in predictable ways, as Figure 3.7 shows.

For instance, a cumulative uniform distribution is a line that goes straight up, because each event adds the same value to the cumulative sum. A cumulative normal distribution has an the shape of a flattened "S". In the power law case, as we'll see, we actually want to see the complement of the cumulative distribution ($1 - \text{CDF}$). This is, interestingly, also a power law.

⁹ Albert-Laszlo Barabasi. The origin of bursts and heavy tails in human dynamics. *Nature*, 435(7039):207, 2005

¹⁰ Mark EJ Newman. Power laws, pareto distributions and zipf's law. *Contemporary physics*, 46(5):323–351, 2005b

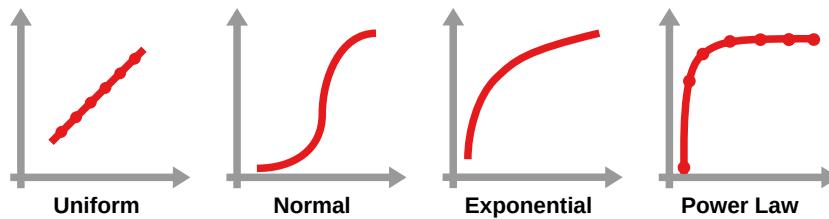


Figure 3.7: A stylized representation of a few cumulative distributions. Same legend as Figure 3.6.

3.3 *p*-Values

One of the key tasks of statistics is figuring out whether what you’re observing – a natural phenomena or the result of an experiment – can tell us something bigger about how the world works. The way this is normally done is making an hypothesis, for instance that a specific drug will cause weight loss. To figure out whether it is true, we need to prove that taking the drug actually does something rather than nothing. “The drug does nothing” is what we call the *null hypothesis*, which is what we’d expect – after all, most drugs don’t cause weight loss. This is what we colloquially call “burden of proof”: the person making the claim that something exists needs to prove that it does, because if we haven’t proven that something exists yet there is no reason to believe it does.

We call what you want to prove – “the drug causes weight loss” – the *alternative hypothesis*, because it’s the alternative to the null hypothesis.

p-values are among of the most commonly used tools to deal with this problem¹¹. The interpretation of *p*-values is tricky and it is easy to get it wrong. The “*p*” stands for “probability”. Suppose that you give your drug to a bunch of people and, after a few weeks, you see that their weight decreased by 5kg. The *p*-value tells you the probability that you would be observing an effect this strong – a loss of 5kg – if the null hypothesis was true – i.e. if the drug actually did nothing. Lower *p*-values mean there is stronger evidence against the null hypothesis. What the *p*-value does not tell you (but might trick you into thinking it does) is:

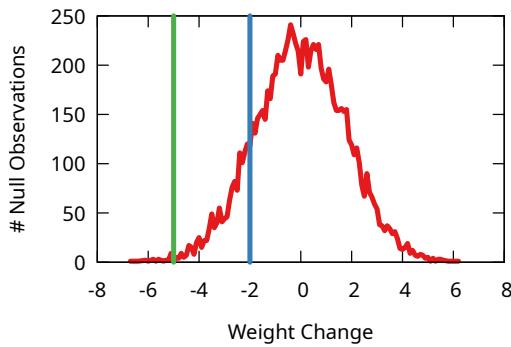
- The *p*-value does **NOT** tell you how likely you are to be right;
- The *p*-value does **NOT** tell you how strong the effect is;
- The *p*-value does **NOT** tell you how much evidence you have against your hypothesis.

Etch these bullet points into your brain, because it is so easy to fool yourself. The last of them means that a high *p*-value does not

¹¹ Ronald L Wasserstein and Nicole A Lazar. The asa statement on p-values: context, process, and purpose, 2016

mean that the null hypothesis is true. A high p-value just means that the observations we have are compatible with a world where the null hypothesis is true. But it could also mean that our sample is not big enough to draw firm conclusions. Given how tricky it is to get them right, some researchers have called for not using p-values altogether¹².

Figure 3.8 shows a graphical way to understand the p-value. You have a distribution of values that would be produced by the null hypothesis, you pit your measurements against those values, and the more unusual your observations look compared to those that would be produced by the null hypothesis, the lower the p-value. Exactly how to produce this null hypothesis distribution is not something we'll cover here, because it is not so close to the core of network science – although we'll see something similar in Section 19.1.



¹² Raymond Hubbard and R Murray Lindsay. Why p values are not a useful measure of evidence in statistical significance testing. *Theory & Psychology*, 18(1):69–88, 2008

Figure 3.8: In red we have the number of observations (y axis) with a given weight change (x axis) under the null hypothesis. In blue we have an observation of 2kg weight loss after taking the drug ($p = 0.14$). In green we have an observation of 5kg weight loss after taking the drug ($p = 0.003$).

One thing is worth mentioning, though. You will often see some magical p-value thresholds that people use as standards – the most common being $p < 0.05$ and $p < 0.01$. These p-value thresholds say something about the strength of the evidence we want to see before we are willing to reject the null hypothesis of no effect. Beware of these. Not only because – as I said before – they don't mean what you think they mean, but also because of Goodhart's Law¹³. If we say $p < 0.01$ is the gold standard we need to achieve to publish a paper, the natural tendency would be to try and repeat/modify experiments until we get $p < 0.01$. This is known in the literature as p-hacking¹⁴ and has led researchers to publish a flurry of false results.

If the p-value tells you the probability of observing a given result under the null hypothesis, then if you repeat your experiments 100 times the probability that at least one of them will leave to a p-value ≤ 0.01 is actually 63%!¹⁵ That is because the probability of getting a $p > 0.01$ is 99% – with this standard of evidence, one percent of the time, you will reject the null hypothesis by accident. You try 100 times, so the formula to know how likely a $p \leq 0.01$ is

¹³ "When a measure becomes a target, it ceases to be a good measure."

¹⁴ Megan L Head, Luke Holman, Rob Lanfear, Andrew T Kahn, and Michael D Jennions. The extent and consequences of p-hacking in science. *PLoS biology*, 13(3):e1002106, 2015

¹⁵ <https://xkcd.com/882/>

becomes $1 - (0.99^{100})$. Sometimes, of course, you do need to run more than one test, and look at more than one p-value. A couple of common options to deal with this are applying the Bonferroni^{16,17} or the Holm-Bonferroni¹⁸ corrections to your p-values, systematically lowering the significance threshold to make up for “cheating” by running multiple tests.

p-values are important for network science because we will often have to create null models to figure out whether some property we observe in a network is actually interesting (Section 19.1). Another case is figuring out whether an edge has a weight significantly different from zero (i.e. it actually exists) or not (Chapter 27).

3.4 Correlation Coefficients

So far we have worked with a single variable and we have done our best to describe how it distributes. However, more often than not, you have more than one variable and you want to know something interesting about how one relates to the other. For instance, you might want to describe how the weight of a person tends to be related to their height.

In general, the taller a person is, the more we expect them to weigh. In this sense the two variables **vary together**. Therefore, we call this concept “**covariance**¹⁹.” The formula is pretty simple, it is the mean of the product of the deviations from the mean of both variables, so: $cov(H, W) = \mu((H - \mu(H))(W - \mu(W)))$.

If a person is both a lot taller than the mean and a lot heavier than the mean, the product of their height and weight deviations will be big, and they will contribute a large positive value to the covariance calculation. If a person is both a lot shorter than the mean and a lot lighter than the mean, the product of their height and weight deviations, both negative, will again be a large positive. So, they will also make the covariance turn out bigger. If someone is both tall and light, their positive height deviation and negative weight deviation will be multiplied to become a large negative number, pulling the covariance down. The covariance will be large in total if we observe many tall/heavy and short/light people, and not so many tall/light or short/heavy people (see Figure 3.9, the covariance values are in the figure’s caption, first row under $cov(H, W)$).

One problem is that the covariance depends on the units of your variables. That means the size of the covariance can be difficult to interpret – is a covariance of 5.5 high or low? To make sense of it, it helps to normalize it. This is what the Pearson correlation coefficient tries to do²⁰. It divides the covariance by the variances of both variables. That means, for example, that if start measuring

¹⁶ Carlo Bonferroni. Teoria statistica delle classi e calcolo delle probabilità. *Pubblicazioni del R Istituto Superiore di Scienze Economiche e Commerciali di Firenze*, 8:3–62, 1936

¹⁷ Olive Jean Dunn. Multiple comparisons among means. *Journal of the American statistical association*, 56(293): 52–64, 1961

¹⁸ Sture Holm. A simple sequentially rejective multiple test procedure. *Scandinavian journal of statistics*, pages 65–70, 1979

¹⁹ John A Rice. *Mathematical statistics and data analysis*. Cengage Learning, 2006

²⁰ Francis Galton. Typical laws of heredity. Royal Institution of Great Britain, 1877

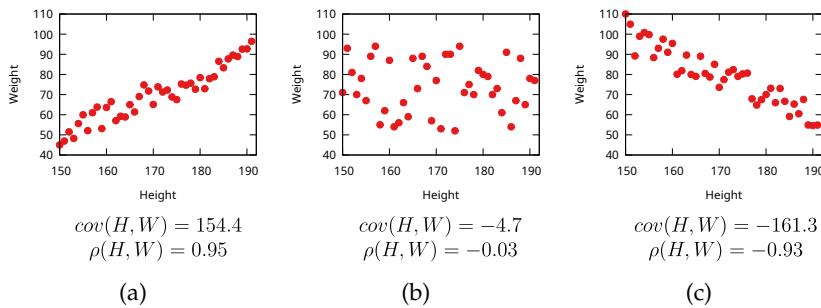


Figure 3.9: Three examples of datasets with (a) positive, (b) no, and (c) negative covariance between height and weight. Covariance ($cov(H, W)$) and correlation ($\rho(H, W)$) values below the scatter plots.

height in meters rather than centimeters, its covariance with weight will change, but its Pearson correlation with weight will stay the same. The Pearson correlation can only take values between -1 (perfect anticorrelation) and $+1$ (perfect correlation).

The Pearson correlation coefficient is nice and it is used for many things – including in network science to predict links (Section 23.7), project bipartite networks (Section 26.2), estimate assortativity (Section 31.1), ... you get the idea. There are a couple of problems with the Pearson correlation. The first is that it only estimates how monotone a relationship is: this means that it wants variables to always vary together in the same general direction. Figure 3.10 shows a couple of non-monotone relationships – the classic “U” and “inverted U” shapes. In this case, even if there clearly is a relationship, Pearson will return zero and there isn’t much you can do about it.

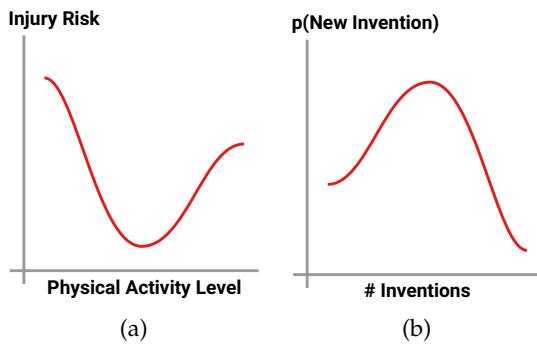


Figure 3.10: Two non monotone relationships (a) U-shaped: if you never exercise you’re frail and prone to injury, and if you exercise too much you have more chances to injury yourself. (b) Inverted U-shaped (or A-shaped): if there are no inventions innovation is hard, if there are too many innovations there’s nothing left to innovate.

The second issue is that, even if the variables have a roughly monotone relationship, Pearson only measures it accurately if this relationship is *linear*. Pearson will give us the wrong idea if increases in one variable are associated with changes in another variable, but less and less so. For example, we visit our friends less often if they live further away from us, but moving 20 kilometers away if they lived next door before could make a big difference, whereas moving 20 extra kilometers away if they already lived in another country is not so meaningful.

One neat trick you can often do to fix this is to do a log-transformation

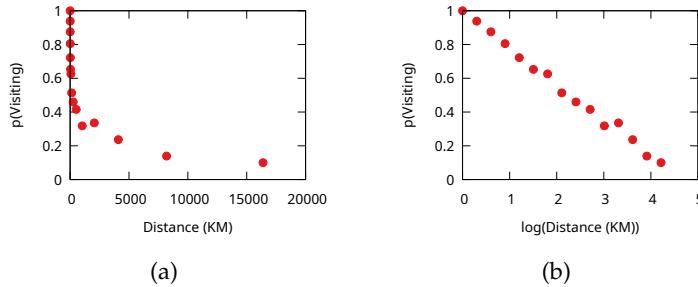


Figure 3.11: Data with skewed distributions across various orders of magnitude. (a) Linear plot. (b) Same data, with the x axis log-transformed.

of your data before you calculate a correlation. In Figure 3.11(a) you can see a pair of variables with a non-linear relationship – the probability of deciding to visit a friend living at a given distance. Once you take the logarithm of the distance (Figure 3.11(b)), a linear relationship with the visit probability comes to the surface. Often, you can log both variables.

Most of the times you can stop here, but when this fails you still have one tool that allows to calculate correlations of data related in just about any way. As long as a change in one variable monotonically corresponds to a change in the other (so, no U- or A-shapes where the direction of the relationship changes midway), you can summarize this with the Spearman rank correlation²¹. You do not assume anything at all about *how much* each variable changes as the other changes. The Spearman rank correlation is still normalized to take values between -1 and $+1$ with the exact same meaning as Pearson.

How can you achieve this? It's actually rather simple. You still calculate a Pearson correlation. But rather than calculating it directly on the values of your variable, you do it on their *ranks*. The observation with the highest value becomes a 1, the second largest becomes a 2, and so on. Then, you calculate the Pearson correlation of these ranks. That is why this is called the Spearman *rank* correlation.

Figure 3.12 provides some examples. You can see what a monotonic but not linear relationship looks like (Figure 3.12(a)), that for data fitting the Pearson's assumptions Spearman returns comparable values (Figure 3.12(b)), and the robustness of Spearman to outliers (Figure 3.12(c)).

²¹ Charles Spearman. The proof and measurement of association between two things. *Am J Psychol*, 15:72–101, 1904

3.5 Mutual Information

Mutual Information (MI) is a key concept in information theory. It is another measure of how related two variables are. You can see it as a sort of a special correlation. Formally, MI quantifies how much information you obtain about one variable if you know the value of the other variable. For example, if we know how tall someone is,

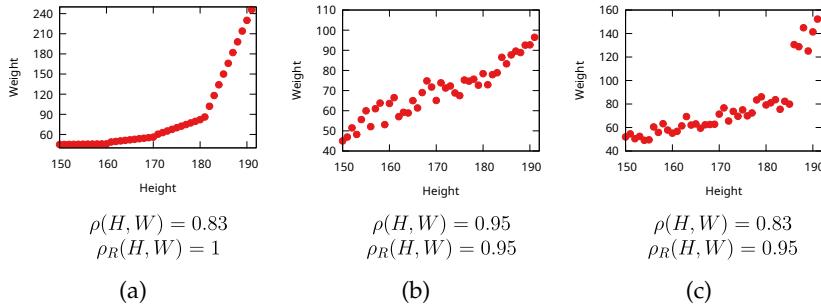


Figure 3.12: Three pairs of variables showing the difference between the Pearson ($\rho(H, W)$) and the Spearman ($\rho_R(H, W)$) correlation coefficients.

does this allow us to make a better guess at their weight? How much better? This “amount of information” is usually measured in bits.

To understand MI, we need to take a quick crash course on information theory^{22,23}, which starts with the definition of information entropy. It is a lot to take in, but we will extensively use these concepts when it comes to link prediction and community discovery in Parts VII and X, thus it is a worthwhile effort.

Consider Figure 3.13. The figure contains a representation of a vector of six elements that can take three different values. The first thing we want to know is how many bits of information we need to encode its content.

| X | | X | |
|---|------|----|------------------|
| | = 0 | 0 | 9 bits |
| | | 0 | 6 values |
| | | 0 | |
| | = 10 | 10 | = 9 / 6 |
| | | 10 | = 1.5 bits/value |
| | = 11 | 11 | |

We can be smart and use the shortest codes for the elements that appear most commonly, in this case the red square. Every time we see a red square, we encode it with a zero. If we don't see a red square, we write a one, which means that we need to look at a second bit to know whether we saw a blue or a green square. If it was a blue square, we write a zero, if it was green we write another one. With these rules, we can encode the original vector using nine bits, i.e. we use 1.5 bits per element.

This is close to – but not exactly – the definition of information entropy. In information entropy, we weigh the probability of an event by its logarithm^{24,25}.

²² Thomas M Cover. *Elements of information theory*. John Wiley & Sons, 1999

²³ David JC MacKay. *Information theory, inference and learning algorithms*. Cambridge university press, 2003

Figure 3.13: A simple example to understand information entropy. From left to right: the vector x has six elements taking three different values. We can encode each value with a sequence of zeros and ones. Doing so allows us to transmit x 's six elements using nine bits of information. This means that the number of bits per value is 1.5.

²⁴ Claude Elwood Shannon. A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423, 1948

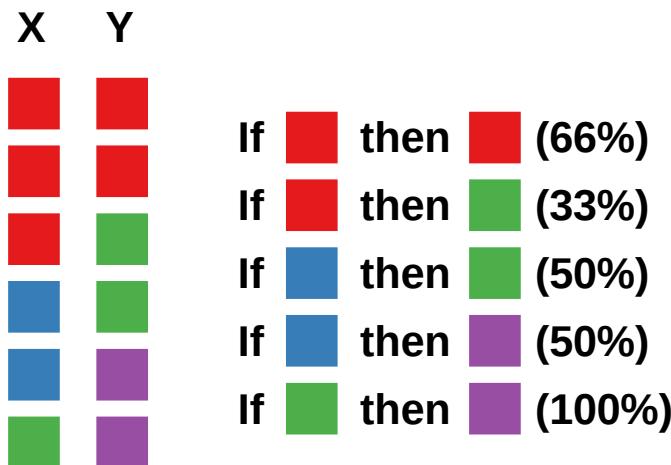
²⁵ Andrei N Kolmogorov. Three approaches to the quantitative definition of information'. *Problems of information transmission*, 1(1):1–7, 1965

Consider flipping a coin. Once you know the result, you obtain one bit of information. That is because there are two possible events, equally likely with a probability p of 50%. Generalizing to all possible cases, every time an event with probability p occurs, it gives you $-\log_2(p)$ bits of information for... reasons²⁶. So, the total information of an event is the amount of information you get per occurrence times the probability of occurrence: $-p \log_2(p)$. Summed over all possible events i in x : $H_x = -\sum_i p_i \log_2(p_i)$, which is Shannon's information entropy – how many bits you need to encode the occurrence of all events.

Mutual information is defined for two variables. As I said, it is the amount of information you gain about one by knowing the other, or how much entropy knowing one saves you about the other. Consider Figure 3.14. It shows the relationship between two vectors, x and y . Note how y has equally likely outcomes: each color appears three times. However, if we observe a green square in x , we know with 100% confidence that the corresponding square in y is going to be purple. This means that, knowing x 's values gives us information about y 's value. Mathematically speaking, mutual information is the amount of information entropy shared by the two vectors.

It would take $-\log_2(1/3) \sim 1.58$ bits to encode y on its own (it is a random coin with three sides). However, knowing x 's values makes you able to use the inference rules we see in Figure 3.14. Those rules are helpful: note how our confidence is almost always higher than 33%, which is the probability of getting y 's color right without any further information. The rules will save you around 0.79 bits, which is x and y 's mutual information.

The exact formulation of mutual information is similar to the formula of entropy:



²⁶The amount of information of an event is a function that only depends on the probability p of the event to happen, e.g. $i_a = f(p_a)$ for event a . If we have two events, a and b , happening with probability p_a and p_b , the event c defined as a and b happening has probability $p_c = p_a p_b$. Now, each event also gives you an amount of information, namely i_a and i_b . When c happens, it means that both a and b happened, thus you got both pieces of information, or $i_c = i_a + i_b$. What we just said can be rewritten as $f(p_c) = f(p_a) + f(p_b)$, given the equation at the beginning. Since $p_c = p_a p_b$, then we can also rewrite the equation as $f(p_a p_b) = f(p_a) + f(p_b)$. The only function f that we can possibly plug into this equation maintaining it true is the logarithm. Since probabilities are lower than 1, the logarithm would be lower than zero, which would be nonsense – you cannot get negative information. Thus we take the negated logarithm: $i_a = -\log(p_a)$.

Figure 3.14: An illustration of what mutual information means for two vectors. Vector y has equal occurrences for its values (there is one third probability of any colored square). However, if we know the value of x we can usually infer the corresponding y value with a higher than chance confidence.

$$MI_{xy} = \sum_{j \in y} \sum_{i \in x} p_{ij} \log \left(\frac{p_{ij}}{p_i p_j} \right),$$

where p_{ij} is the joint probability of i and j . The meat of this equation is in comparing the joint probability of i and j happening with what you would expect if i and j were completely independent. If they are, then $p_{ij} = p_i p_j$, which means we take the logarithm of one, which is zero. But any time the happening of i and j is not independent, we add something to the mutual information. That something is the number of bits we save.

3.6 Summary

1. If you want to know what an average observation looks like in a variable, you might want to calculate the mean or the median.
2. The average observation will deviate from the mean: to estimate how far from the mean it will be on average, you can calculate the variance or the standard deviation.
3. Data can be skewed, meaning it distributes asymmetrically around the mean. There could be long or fat tails, depending how extreme and uncommon outliers are. The more skewed your data is, the more mean and median will disagree.
4. There are important distributions one should know: uniform, where all values are equally likely; normal, where values cluster around the mean; and various skewed distributions such as exponential, lognormal, and power law.
5. A cumulative distribution tells you the probability of observing a values equal to or smaller than a given threshold.
6. The p-value tells you the probability of making a given observation under the null hypothesis (no change, no effect, nothing interesting is happening). A low p-value can prove the null hypothesis wrong but a high p-value does not prove the null hypothesis is true.
7. If you do multiple experiments, you need to correct the p-values you get otherwise you are going to misinterpret them.
8. When you have two variables, covariance tells you how much the two change together. Correlation coefficients are normalized covariances that do not change depending on the scale of the data. If your variables have a non-linear relationship, you need to use a correlation coefficient that can handle it.

9. Another approach to measure of how related two random variables are is mutual information. It tells you how many bits of information you gain about the status of one variable by knowing the other.

3.7 Exercises

1. Calculate the mean, median, and standard deviation of the two variables at <http://www.networkatlas.eu/exercises/3/1/data.txt> (one variable per column).
2. Make a scatter plot of the variables used in the previous exercise – with one variable on the x axis and the other on the y axis. Do you think that they are skewed or not? Calculate their skewness to motivate your answer.
3. Draw the mass function and the cumulative distribution of the following outcome probabilities:

| Outcome | p |
|---------|------|
| 1 | 0.1 |
| 2 | 0.15 |
| 3 | 0.2 |
| 4 | 0.21 |
| 5 | 0.17 |
| 6 | 0.09 |
| 7 | 0.06 |
| 8 | 0.02 |

4. Which correlation coefficient should you use to calculate the correlation between the variables used in the exercise 2? Motivate your answer by calculating covariance, and the Pearson and Spearman correlation coefficients (and their p-values). Does the Spearman correlation coefficient agree with the Pearson correlation calculated on log-transformed values?
5. How many bits do we need to independently encode v_1 and v_2 from <http://www.networkatlas.eu/exercises/3/5/data.txt>? How much would we save in encoding v_1 if we knew v_2 ?

4

Machine Learning

Machine learning is a collection of data analysis techniques that aim at discovering patterns in data. The main difference between statistics and machine learning is that statistics is “top-down” and machine learning is “bottom-up”. By “top-down” I mean that, in statistics, you usually have a theory or hypothesis, you impose it on the data from above, and then you see if it fits. On the other hand, machine learning’s “bottom-up” means that you instead scout for patterns and correlations in the data that you didn’t necessarily know about, and you try to let them arise from the data.

Here’s the usual disclaimer – and I can tell you’re already sick of it -: if you already know your machine learning, you can skip this chapter. If, instead, you want to truly learn machine learning, you should check out specialized texts^{1,2} that don’t cram superficial explanations in a rushed chapter.

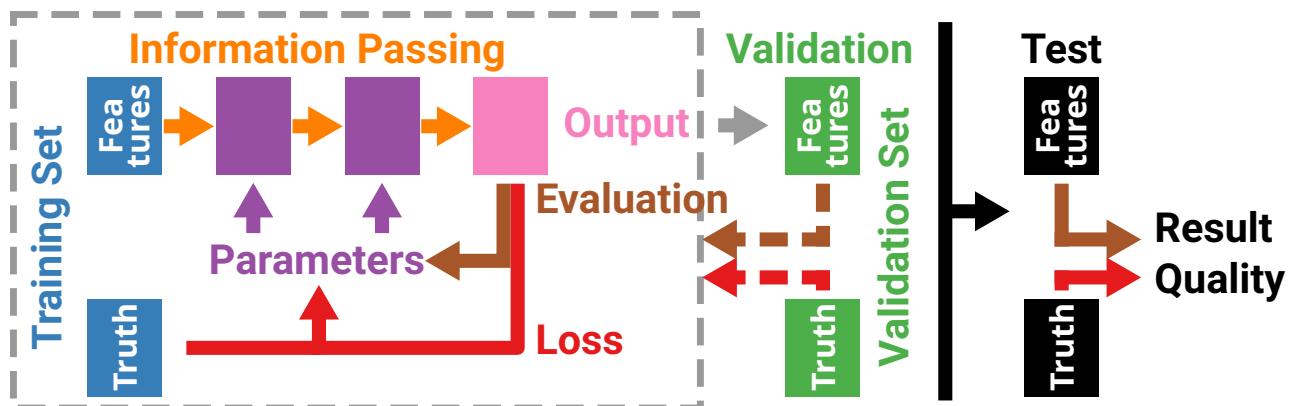
This chapter will not include one part of machine learning, which are evaluation measures: the functions telling you how well you did in your prediction task. There are a couple of reasons why. First, because they fit better in the link prediction part (Part VII), since they are used mostly for that specific task in network science. The second reason is to highlight their difference with loss functions, which are instead treated here. Ostensibly, evaluation measures and loss functions do the same thing: given an objective, they tell you how close you are to it. However, loss functions are used for *training* while evaluation measures are used for *testing*. These two phases of machine learning – as we’ll see shortly – are different, they should be as separate as possible, and that is why we should always have different measures and functions to drive them, which is why I create this big divide between evaluation measures and loss functions.

¹ Ethem Alpaydin. *Introduction to machine learning*. 2020

² Jiawei Han, Jian Pei, and Hanghang Tong. *Data mining: concepts and techniques*. Morgan kaufmann, 2022

4.1 General Structure

To explain machine learning, I'm going to rely on a generic architecture I depict in Figure 4.1. This is a bit of a simplification, but for a network science book it will do. It also looks daunting, but it's just a bunch of pipes, and I'm going to make a plumber out of you now.



At a bird's eye view, you have three phases: **training**, **validation**, and **test**. Here, we deal only with the first two, leaving the test phase for Chapter 25 about link prediction.

The objective of the **training** phase is to make your algorithm learn the patterns in your data. The way you do it is by passing the **data** into your algorithm governed by a bunch of **parameters**, possibly through a pipeline of several different operations – **passing** the processed information –, and getting out an **output**. Then you have to figure out how good your output is and there are two ways to do it.

If you already have the right answers for a given problem, you can use them to drive the learning process: the algorithm should learn from that truth and you can estimate how much it is currently **getting it wrong**. This is *supervised learning*, because we can supervise the algorithm and drive what it learns by using the correct answers.

Otherwise, you might just have a (bunch of) generic quality function(s) you want to **maximize/minimize**. This is *unsupervised* learning, because the algorithm can roam freely through the parameter space in search of the patterns. Since there are no hard truth to reach in supervised learning, the evaluation functions could look rather different from the loss functions you use for training! One classical case is large language models: your evaluation could be how spooked humans are when reading the output, which is quite difficult to use as a loss function in the training phase – given that you don't want to

Figure 4.1: A generic machine learning pipeline. Arrows indicate the flow of information. The red arrows and the truth boxes are only present for supervised learning.

ask humans to evaluate millions of outputs.

Then there is the **validation** phase. This looks like the evaluation we did in the training phase. The key difference is that the **data** we use in the validation phase was never seen by the algorithm in the training phase. We do the validation step to reduce the problem of overfitting³. Figure 4.2 shows a classical case of overfitting. It's possible for the algorithm to be getting better and better and better on the training data, but starting to give very poor predictions for new data. If we observe a much reduced quality / much higher errors in the validation set than in the training set, it means the algorithm cannot generalize and we should probably revise something about its architecture.

Errors



³ Xue Ying. An overview of overfitting and its solutions. In *Journal of physics: Conference series*, volume 1168, page 022022. IOP Publishing, 2019

Figure 4.2: Errors (y axis) as a function of training iterations (x axis) in the training (blue) and validation (green) sets. The red arrows highlights the moment in which we stop making progress on the validation set, a classical sign of overfitting.

It is important to know that we never optimize parameters over the validation set data, which is only used once the training phase is over. That is why I box the training phase in the gray outline in Figure 4.1. If the validation set tells us we're overfitting, we revise the training phase in isolation and once we're done with training we try again to validate. If you were to accidentally leak information from the validation set in the training phase, you might think you're solving overfitting, but you're actually making the problem worse.

Once you're happy about having fixed your overfitting, you can move to the **test** phase. Note that no information ever makes it back from here: once you get to testing, it's do or die. Which means that the data in the test set must be separated from both training *and* validation. The performance reports you get from the test phase are final.

There are alternatives to the supervised/unsupervised approach I described. One could do *semi-supervised* learning⁴, where you have a small portion of annotated data which you can use to kickstart

⁴ O. Chapelle, B. Scholkopf, and A. Zien. *Semi-Supervised Learning*. Adaptive Computation and Machine Learning series. MIT Press, 2010. ISBN 9780262514125. URL <https://books.google.dk/books?id=TtWMEAAQBAJ>

a first phase of supervised learning, before moving to the bulk of unlabeled data and use unsupervised learning that is guided by what you learned in the supervised phase. Another big paradigm is *reinforcement learning*^{5,6}, but that matters most for AI and agents and we skip it here.

Ok, so now I gave you a lot of empty labels that need filling.

- By data we mean anything we can store in a bunch of numerical vectors. Think of describing a human being as its height, weight, running speed, ... There is no distinction in quality between the data in the [training set](#) or the one in the [validation set](#) – except that they must be kept separate at all times. We'll see in Part XI that it's tricky to do this with networks. In Chapter 42 we'll see methods to reduce complex network structures to numeric vectors, while in Chapter 44 we'll figure out how to do the machine learning directly on the network structure.
- The [parameters](#) are the things regulating how the algorithm works. If you're trying to predict how weight influences height, your algorithm could be simply to multiply the height with a given number. That number is the parameter.
- By [information passing](#) we mean the mechanism to inform a phase of your pipeline with the results of the previous phase – or to sum up the results in an [output](#). This is the job of activation functions, which we see in Section 4.2.
- The errors can be estimated using a loss function, the topic of Section 4.3. Loss functions can help both if you have [errors on your true answers](#), or you're trying to [minimize/maximize](#) a given outcome.

4.2 Activation Functions

Activation functions take the output of an arbitrary analysis and transform it so that you can do either of two things with it. You can either summarize all the information you have to make a choice, or you can pass on information for further processing. We need activation functions to ensure that the data passing through them assumes some convenient properties that will facilitate either of those two tasks⁷. We'll see a couple of examples that fall in either class.

Summarizing Information

To summarize information, an activation function should take an arbitrary real number as an input and return a number that you

⁵ Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996

⁶ Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018

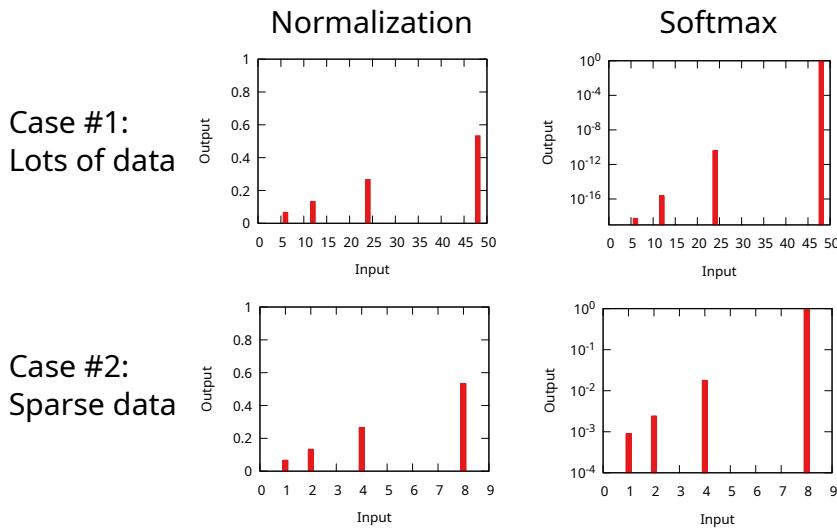
⁷ Chigozie Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. Activation functions: Comparison of trends in practice and research for deep learning. *arXiv preprint arXiv:1811.03378*, 2018

can interpret, roughly as the probability or degree of belonging to a certain class. To understand what that means, let's take a look at a popular option, the softmax function⁸:

$$\text{softmax}(X)_i = \frac{e^{x_i}}{\sum_{x_j \in X} e^{x_j}},$$

where X is a vector of numbers and we want normalize its i th entry.

The softmax function is useful because it can transform arbitrary vectors of real numbers into probabilities, which is convenient if you're doing a classification task with multiple possible outcomes. But why do we need such a complex function? Why all the exponentiations? Can't we just divide X by its sum and call it a day? Well, let's compare what happens if we try to do that for a network task with both softmax and this normalization approach – which I show in Figure 4.3.



⁸ John S Bridle. Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. In *Neurocomputing: Algorithms, architectures and applications*, pages 227–236. Springer, 1990

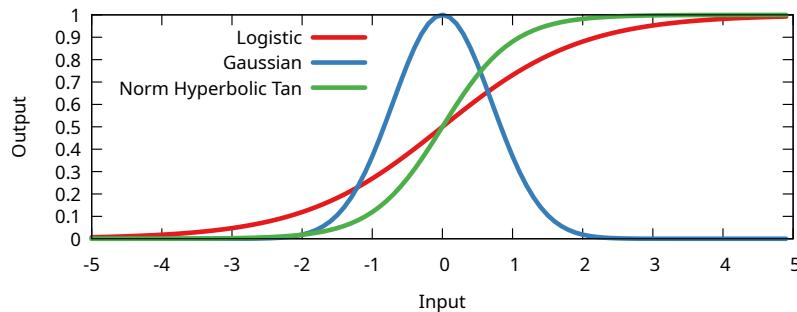
Figure 4.3: Different activations for plain normalization (left column) and softmax (right columns) for different scenarios with data richness (top row) or sparsity (bottom row). Each plot shows the activation function value (y axis) for a given input (x axis).

Suppose you want to classify people into social circles or communities (Chapter 35). A totally legit way to do it is by counting the number of friends you have who we already sorted in a group. The more of your friends belong to community c_1 the more likely it is that you're part of c_1 yourself. If you have four communities, for a given person v you might have these membership counts: [48, 24, 12, 6] – meaning you have 48 friends in the first group, and so on. A simple normalization would give you a probability of 53% of belonging to the first community, while softmax will be almost certain at more than 99.9% – this is the top row of Figure 4.3.

The reason is that simple normalization applies a frequentist approach, while softmax is more Bayesian (see Section 2.1): each friend in the first community is additional evidence you’re part of that community. Consider what happens if we decrease all counts: [8, 4, 2, 1]. For the normalization approach nothing changes, because the proportions are the same. But this person has much fewer friends, so the Bayesian will admit more ignorance and give “only” a 97.8% probability of belonging to the first community. This is the bottom row of Figure 4.3 – and pay attention to the differences in the y axis scales.

Another argument for softmax over normalization is that softmax can take a vector with negative numbers and still return probabilities. For instance passing [4, 1, -1, -4] through softmax gives (approximately) [0.946, 0.047, 0.006, 0.001]. Passing the same vector to a plain sum normalization returns instead [W, T, F, ?].

Softmax is nice for classification tasks with multiple options, but more often than not you just want to know whether an output is or isn’t a particular thing – i.e. you’re doing a binary classification. For instance, in link prediction (Chapter 23), either a link exists or it doesn’t. Doing softmax in this case is an overkill and you can use other popular activation functions such as the logistic⁹ (or sigmoid), the Gaussian, and the hyperbolic tangent¹⁰, which I show in Figure 4.4. Logistic is actually equivalent to softmax if softmax gets only two options. Note that, in the figure, I renormalize the hyperbolic tangent to be between 0 and 1, rather than its normal -1 to +1 interval.

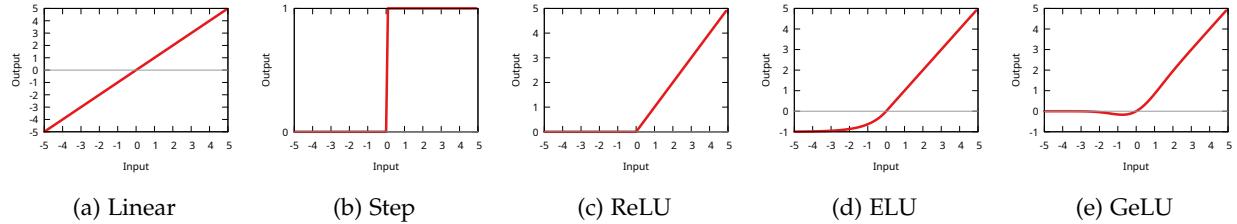


⁹ Neil A Gershenfeld. *The nature of mathematical modeling*. Cambridge university press, 1999

¹⁰ Incidentally, “hyperbolic tangent” also describes well my writing style.

Figure 4.4: Different activation functions (line color): the activation function value (y axis) for a given input (x axis).

For the logistic, the higher the value going in, the more likely it is to be an example of a positive class. For the Gaussian, we instead want to be close to a given value, usually zero – which can be interpreted as “the class is positive if its distance from a reference point is low”. The hyperbolic tangent is not interpretable as a probability, but you can still normalize it if you want – as I did just to plot this figure.



Passing Information

Not to go onto another hyperbolical tangent, I'll keep this section barebone and mention that the activation functions we're dealing with here want to transform their input to make it more useful for a further analysis. In this case, activation functions should augment the informative character of their input and/or prevent problems down your analytic pipeline – fancy stuff that has esoteric names used to evoke a sense of wonder and mystery such as "vanishing gradients"¹¹. Let's focus on a handful of activation functions, which I show in Figure 4.5.

The Linear activation function does nothing at all: what you pass in goes out. The Step activation function tests whether the input is positive or negative, and returns a binary output. If you want to go all fancy and impress people at cocktail parties, you can call it the Heaviside Step Function – and enjoy the *uuuhs* and *aaahs*.

The Rectified Linear Unit¹² – ReLU for its friends – is the star of the bunch and probably the most used one¹³, since it works so well in neural networks¹⁴. If the input is negative it returns zero, and if it is positive ReLU is like the Linear function and it does nothing. In practice, $\text{ReLU}(x) = \max(0, x)$. ELU¹⁵ and GeLU¹⁶ are slight modifications of ReLU with more complex formulas, but they follow ReLU's approach.

4.3 Loss Functions

Whenever you're training your predictor, you're going to get things wrong. If you're never wrong during training, either your problem is so trivial you don't need machine learning, or something fishy is going on and there must be a mistake somewhere. The role of a loss function is to drive the training process to minimize its mistakes. Once you have a loss function, you can compare two potential updates in the training phase to pick the best one. Training means nothing else than trying something, calculate the loss function, try something else and, if the loss now is lower, keep the change and keep going in that direction. It follows that the loss function is as

Figure 4.5: Five examples of activation functions passing information in a machine learning framework. Input of the function on the x axis and output on the y axis. I draw a horizontal gray line to show where the zero output is, to contextualize and compare the outputs.

¹¹ Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02):107–116, 1998

¹² Kunihiko Fukushima. Visual feature extraction by a multilayered network of analog threshold elements. *IEEE Transactions on Systems Science and Cybernetics*, 5(4):322–333, 1969

¹³ Prajit Ramachandran, Barret Zoph, and Quoc V Le. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017

¹⁴ Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323. JMLR Workshop and Conference Proceedings, 2011

¹⁵ Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015

¹⁶ Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016

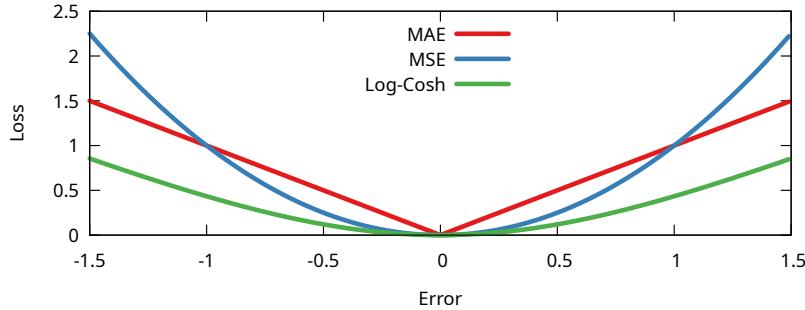


Figure 4.6: Different loss functions (line color): the loss function value (y axis) for a given error (x axis).

important as the activation function that propagates information. I show you a few examples of loss functions in Figure 4.6.

The reason why you have many loss functions is because you might want to treat differently different types of errors. For instance, you can see how the MSE function in Figure 4.6 – which I’m about to define – shoots up for large errors. This means it is sensitive to outliers, which are expected to produce large errors. An alternative, such as Log-Cosh will give less weight to these outliers, producing a lower loss for the same error level¹⁷.

Again, be reminded of the difference between loss and errors. Loss is something you care about minimizing during training, and loss functions should be generic enough not to lead to overfitting. Errors are an evaluation in test phase on which you cannot and should not minimize, only hope you get them as small as possible. When it comes to error functions, you can be as specific as you want, because you already passed the phase in which overfitting was a concern.

Mean Errors

Two classical examples are mean absolute error (MAE) and mean squared error (MSE). In MAE, if y_i is the real outcome and \bar{y}_i is what your method says, then you average the absolute value of the difference¹⁸:

$$MAE(\bar{Y}) = \sum_i |y_i - \bar{y}_i| / |\bar{Y}|,$$

where \bar{Y} is your vector with all your answers and Y is the vector with the corresponding correct answers.

It should be clear why we need the absolute value: if you were to make symmetric errors (for each overestimation you also make, for a different observation, an equal underestimation) your loss would be zero – confusing symmetric errors with no errors at all. In MSE you solve the same issue of symmetric errors by taking not the absolute

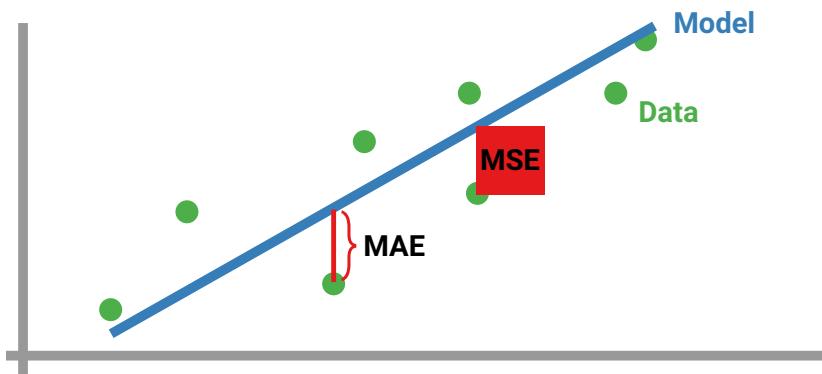
¹⁷ Shruti Jadon. A survey of loss functions for semantic segmentation. In 2020 IEEE conference on computational intelligence in bioinformatics and computational biology (CIBCB), pages 1–7. IEEE, 2020

¹⁸ Cort J Willmott and Kenji Matsuura. Advantages of the mean absolute error (mae) over the root mean square error (rmse) in assessing average model performance. *Climate research*, 30(1): 79–82, 2005

value, but the squared error¹⁹ (which is always positive, also for underestimates):

$$MSE(\bar{Y}) = \sum_i (y_i - \bar{y}_i)^2 / |Y|.$$

If you find it distasteful to take the square because then you have an error in different units than your observation, you can always take the square root of the result and you have the root mean squared error – just like you can take the root of the variance to get the standard deviation (Section 3.1). Figure 4.7 gives you a graphical representation of how the errors are calculated.



¹⁹ Peter J Bickel and Kjell A Doksum. *Mathematical statistics: basic ideas and selected topics, volumes I-II package*. Chapman and Hall/CRC, 2015

Figure 4.7: Given a model (blue) for data points (green), we have two ways to interpret the errors (red).

In Figure 4.7, for MAE, only the length of the segment matters. For MSE, you instead consider the entire area of the square.

(Log) Likelihood

The likelihood function is a function that tells you what is the probability of observing an event given a (set of) parameter(s) regulating such an event²⁰. Notation-wise, we use \mathcal{L} to indicate the likelihood function, x is the outcome of the event, and θ is the set of parameters.

Let's make a simple example. Suppose that we are tossing a coin three times. The sides on which it lands, let's say heads twice and tails once, is our $x = \{H, H, T\}$. θ tells us the parameter regulating the coin. A coin is a fairly simple system, so it has only one parameter: the probability of the coin landing on heads – which is 50% when we know nothing about the coin and whether it is fair. So we can say $\theta = \{p_H = 0.5\}$. At this point we can estimate $\mathcal{L}(\theta, x)$ – which, in our case, is $\mathcal{L}(\{p_H = 0.5\}, \{H, H, T\})$ – which is the likelihood of the coin being fair given that we observe that given event. In this case, to know the \mathcal{L} value given the event for any p_H value we need to evaluate the formula $p_H^2(1 - p_H)$ – because we got two heads and one tails

²⁰ Anthony William Fairbank Edwards. Likelihood. In *Time Series and Statistics*, pages 126–129. Springer, 1972

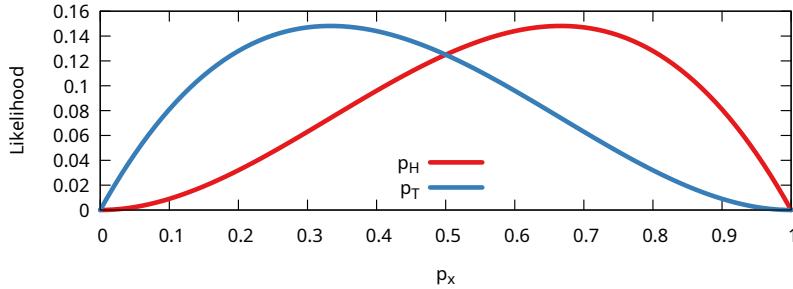


Figure 4.8: The likelihood (y axis) of the parameter p_H (x axis) for the $\{H, H, T\}$ event.

in entirely independent events and so we multiply the probabilities (Section 2.3). This is what Figure 4.8 shows in red.

So the likelihood of $p_H = 0.5$ is 0.125, because that's the value of $0.5^2(1 - 0.5)$. That's not the most likely value, since we got two heads. The most likely value is $p_H = 0.66$, or 0.2178. Note that this is all symmetric to p_T since H and T are independent, mutually exclusive, and one is one minus the other. The figure also shows the likelihood of p_T , in blue, for the same data, which is symmetric to p_H , being $p_T(1 - p_T)^2$.

Note that \mathcal{L} is NOT the probability that the coin is fair given the experiment result, because to reach that conclusion you should apply Bayes's theorem (Section 2.5). What you observe is $P(x|\theta)$ – the probability of the event given the fairness –, but that is NOT equivalent to $P(\theta|x)$ – the probability of the fairness given the event. So don't make this mistake when looking at the results of the likelihood function.

You can see from Figure 4.8 why likelihood is useful: by looking at its value you can compare different parameter values and \mathcal{L} will tell you which one is more likely to be accurate – given the data you have. So you can use \mathcal{L} to drive your training phase. Of course, real world problems are much harder than figuring out how loaded a coin is, so it is not this trivial to fine tune your parameters with the likelihood function – also because you might have many parameters and so you're trying to explore a multidimensional space, which here is monodimensional, since we only have p_H to play with.

For practical reasons, it is common to use $\log(\mathcal{L})$, i.e. to take the logarithm of the likelihood function rather than \mathcal{L} itself. The reason is that, as you saw above, to calculate the likelihood we need to make a lot of multiplications. If we instead have log-likelihood we only have to do sums, because the logarithm of the product of two quantities is the sum of their logarithms. Sums are faster than products, and so this is convenient computationally. Figure 4.9 shows

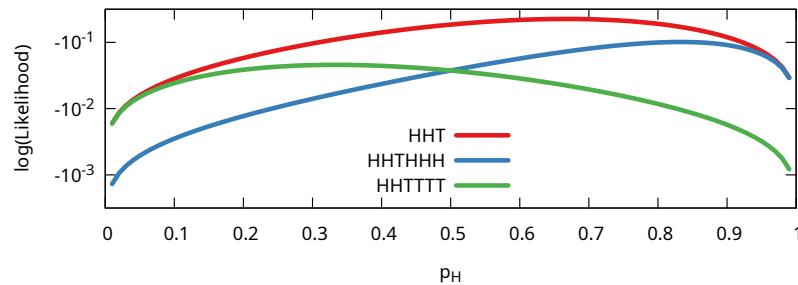


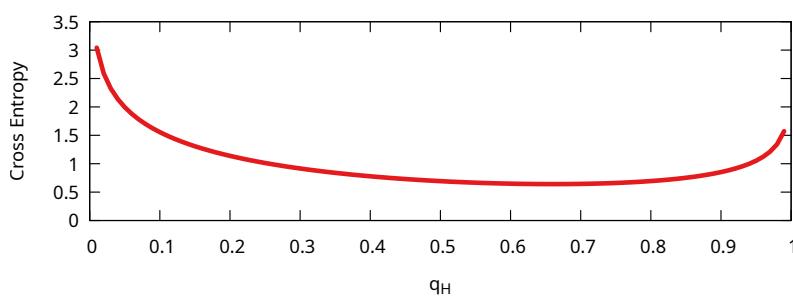
Figure 4.9: The log-likelihood (y axis) of the parameter p_H (x axis) for different events (line color).

a couple of log-likelihood functions for different events with our coin.

Since we're taking the logarithm of the likelihood function, which is always below 1, the log-likelihood is always negative. In the figure, we can see that, when we get additional data, the log-likelihood function changes – humping before for $p_H = 0.5$ if we get a lot of tails (meaning heads is less likely), or after if we keep getting heads.

Cross Entropy

In Section 3.5 I introduced the concept of information entropy, which is how many bits you need to encode the occurrence of all events. The formula is $H_x = -\sum_i p_i \log_2(p_i)$, with p_i being the probability of event outcome i . If we have two different probability distributions, let's call them p and q , we can calculate their cross entropy as $H_x = -\sum_i p_i \log_2(q_i)$. This can be used as a loss function^{21,22}, because you can assume that p and q are describing the same events, but p describes the real probabilities you're trying to predict and q describes instead the probabilities you get out of your model. Figure 4.10 shows how cross entropy works in our simplified example of guessing the p_H of a loaded coin.



²¹ Zhilu Zhang and Mert Sabuncu. Generalized cross entropy loss for training deep neural networks with noisy labels. *Advances in neural information processing systems*, 31, 2018

²² Anqi Mao, Mehryar Mohri, and Yutao Zhong. Cross-entropy loss functions: Theoretical analysis and applications. In *International Conference on Machine Learning*, pages 23803–23828. PMLR, 2023

Figure 4.10: The cross entropy (y axis) of our q_H guess (x axis), given that the real $p_H = 0.66$.

For the figure, I set $p_H = 0.66$. Then our classifier needs to spit out the q_H it believes being the closest to the real p_H . The closest q_H is to 0.66, the lower cross entropy is. You can see how cross entropy shoots up the farther we get from our mark, while being tolerant of smaller mistakes – the function is pretty flat around 0.66.

4.4 Dealing with Computational Complexity

One last important thing I should point out about machine learning is that it is normally done on a truckload of data. If you don't have a lot of observations, the appeal of machine learning is not that great. The more data points you have, the more likely it is you're going to discover whatever pattern lurks in them. Otherwise, the patterns might be overpowered by whatever other random fluctuation affects the observations. This introduces the problem of computational complexity. If you're going to do machine learning, you need to do it quickly, to process large amounts of information in a short time.

This is a problem because, even if the operations you perform are simple, you need to do them for each observation, potentially billions of times, which will take time. There are two main solutions to this issue: sampling and batching.

Sampling

In sampling, you decide not to perform your operation on all data points, but on a selection of them. That selection is a sample. The key problem to solve here is to get a representative sample: if all data points in your sample are “weird” in the same way, you might discover a pattern that is not present in your overall dataset. There are a few techniques to ensure that your sample is representative, but I'm not covering them here in details. That doesn't mean that you should avoid learning how to sample, since it's quite important²³. What I will say, though, is that sampling complex networks is its own variation of the problem, one we will look at in Chapter 29.

Of particular interest is a specific type of sampling called *negative sampling*²⁴. Here a network example is helpful, let's look together at Figure 4.11. I know that I haven't formally introduced networks yet, but hopefully you can follow with some intuition.

Suppose we want to figure out what causes connections between nodes in Figure 4.11. We should collect various features and compare connections with non-connections. The problem is that the number of non-connections dwarfs the number of connections. We have 10 connections, but – trust me on this – a whopping 45 pairs of nodes that are not connected. And this is a super tiny network! Imagine what

²³ Chao-Yuan Wu, R Manmatha, Alexander J Smola, and Philipp Krahenbuhl. Sampling matters in deep embedding learning. In *Proceedings of the IEEE international conference on computer vision*, pages 2840–2848, 2017

²⁴ Zhen Yang, Ming Ding, Chang Zhou, Hongxia Yang, Jingren Zhou, and Jie Tang. Understanding negative sampling in graph representation learning. In *ACM SIGKDD*, pages 1666–1676, 2020

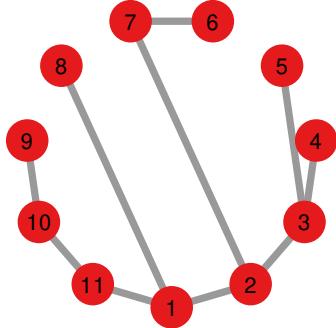


Figure 4.11: A network with fewer connections than non-connections.

would happen if you had an actually large one. You won't be able to look at all the *negative* instances – the non-connections. That is why you need to *sample* them, usually to have as many negative samples as observations – so here you'd pick 10 random non-connected node pairs. This is negative sampling.

Batching

Ok, now we've cut the potentially gigantic number of negative observations with negative sampling. Are we good? Not really. What you have now is twice the number of data points – for each data point you have a companion generated via negative sampling. This can still be pretty huge, if you have a lot of data points – as you should. What now?

Now, batching²⁵. In many common machine learning infrastructures, it is much faster – and it requires less memory – to run the training process on a small portion of the data at a time. Once you learn your parameters with this first chunk, you restart the learning process, updating the parameters with another portion of the data. Rinse and repeat until you have used all of your data. These chunks are the batches. Sometimes, it makes sense to use really small batches, and we have a cute pet name for them: minibatches²⁶.

The advantages of batching are faster computation and less memory required. This might come at a cost of accuracy, so batching should be done with care – specifically when choosing the batch size.

To remember the difference between batching and sampling, you can summarize them in your head as follows. If you only do sampling, you train once on part of the data. If you do only batching, you train multiple times on all your data, one chunk at a time.

Of course, you might end up doing both sampling and batching.

²⁵ Nils Bjorck, Carla P Gomes, Bart Selman, and Kilian Q Weinberger. Understanding batch normalization. *Advances in neural information processing systems*, 31, 2018

²⁶ Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017

4.5 Summary

1. Machine learning is a set of techniques to discover patterns from data without necessarily knowing in advance what they might be. Generally, you have an algorithm with some parameters and you learn them through a training phase.
2. In the training phase, you might have the right answers you're interested in finding and your algorithm can learn from them. This is supervised learning. If you don't have them, you'll be doing unsupervised learning.
3. During training you might overfit, i.e. learn the odd peculiarities of the training set rather than the general patterns in the data. Having a separate validation set can help you to spot overfitting.
4. To update the parameters and producing an output you need to pass information about the data. This is the job of the activation functions: they take a generic signal and return one that satisfies some properties we care about – e.g. to make it into an interpretable probability.
5. Loss functions drive your training phase by telling you how far from the objective your method is. They should not be confused with error functions, which are used in testing phase to tell you how wrong your final answers are – and, since they are final, they cannot be changed, unlike training outputs.
6. Machine learning operates on large datasets. To increase efficiency you can sample your data points, learning only on a part of them. You can also do batching, updating your parameters in the training phase on one small chunk of your data at a time.

4.6 Exercises

1. Generate a random vector with 100 normally distributed random values (with zero average and standard deviation of one). Implement the softmax function. Plot the result with the original vector on the x axis and the softmax output on the y axis.
2. Generate a random vector with 100 normally distributed random values (with zero average and standard deviation of one). Implement the ReLU function. Plot the result with the original vector on the x axis and the ReLU output on the y axis.
3. Generate a random vector with 100 normally distributed random values (with zero average and standard deviation of one). Imple-

ment the MAE and MSE functions and compare their outputs when applied to the vector, by plotting each of them.

4. Plot the likelihood function for p_H and p_T for the events $\{H, H, T, H, T\}$.

5

Linear Algebra

The final piece of groundwork we need to complete the foundations of network science is linear algebra. You will learn in Chapter 8 that there are many different ways to represent a network as a matrix. Linear algebra is necessary to understand what sort of operations you can perform on those matrices, and what they mean.

Normally, in chapter preambles I provide you some generic academic references that cover the topic as a whole, pointing out that they are a much better and more complete resources than what I write. While that's certainly the case here as well^{1,2,3}, I'd say that my strongest recommendation by far does not come from academia, but from YouTube. The linear algebra series from 3blue1brown⁴ is, quite possibly, the best and most intuitive introduction to linear algebra that I had the pleasure to consume.

5.1 Vectors

The reason why linear algebra is called linear algebra is because it is a collection of algebraic techniques to solve systems of linear equations. However, we're not going to use this perspective, because using a **spatial** perspective is more intuitive and closer to what we're actually going to do with linear algebra – which is to deal with networks.

The starting point of linear algebra is the vector. A vector is a list of numbers. So for instance $[3, 1]$ is a vector. The number of entries in the vector is the dimension of the space it lives in. The example I just gave is a vector embedded in a two dimensional space, often we say it is a 2D vector, although this habit will get us in a bit of terminology trouble down the road. A three dimensional vector could be $[4, 1, 7]$. In the spatial interpretation of linear algebra, you can interpret these numbers as coordinates in space. By convention, the origin of the space, its central point, is a vector full of zeroes – so $[0, 0]$ is the origin of a two dimensional space. While the vector is a point in space,

¹ Gilbert Strang. *Introduction to linear algebra*. Wellesley-Cambridge Press
Wellesley, MA, 1993

² Carl D Meyer. *Matrix analysis and applied linear algebra*, volume 71. Siam, 2000

³ <https://ocw.mit.edu/courses/mathematics/18-06-linear-algebra-spring-2010/>

⁴ https://www.youtube.com/playlist?list=PLZHQB0WTQDPD3MizzM2xVFitgF8hE_ab

sometimes it is convenient to think of it as an arrow – so that's why I use both representations in Figure 5.1. The tail of the arrow is always at the origin and the point of the arrow is always at the coordinates specified by the vector.

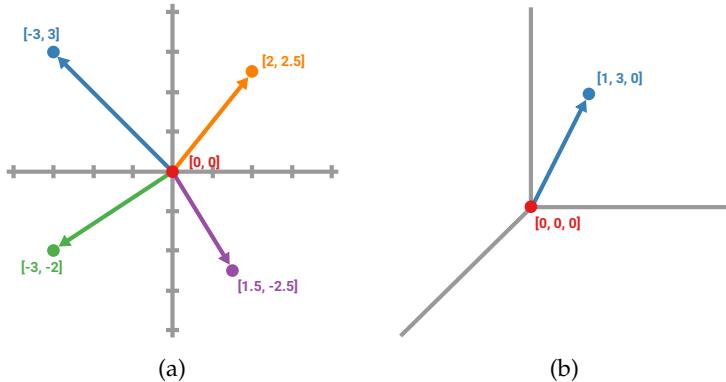


Figure 5.1: (a) Several vectors on a 2D space. (b) Two vectors on a 3D space.

From the figure you can see that it is a bit cumbersome to represent 3D vectors on a piece of 2D paper, so that's why I will stick to 2D examples when making figures. But there is no need to stop at 2D. In fact there is no need to stop at 3D either: all operations in linear algebra work the same in an arbitrary number of dimensions. They just become a little harder to picture in your head.

Notation-wise, normally a vector is written as \vec{v} , but I'm skipping the arrow and writing them as simple lowercase letters: v . This is a bit ambiguous, but in general it will be clear from the context when I'm talking about vectors or not.

One useful property of a vector is its length. The length of the vector is literally the length of the arrow we draw: it is the Euclidean (straight line) distance between the point identified by the vector and the origin. To calculate the Euclidean distance we can realize that a vector is nothing more than the hypotenuse of a special right triangle. Starting from the origin, we can first travel along the x axis until we get to the first coordinate of the vector, then we walk up parallel to the y axis to make up the second coordinate of the vector. So the $[3, 4]$ vector in Figure 5.2 is three steps along the x axis and four steps parallel to the y axis.

Pythagoras teaches us that the length of the hypotenuse is the square root of the sum of the squares of the catheti lengths⁵. Putting it into mathematical form, our $[3, 4]$ vector is $\sqrt{3^2 + 4^2} = 5$ long.

More generally, any d dimensional vector v is $\sqrt{\sum_{k=1}^d v_k^2}$ long.

What operations can you do with vectors? Fundamentally, we need two: **sum** and **multiplication**.

The **sum** of two vectors is pretty straightforward. Say you have $[1, 3]$ and $[2, 2]$. What do you think the result of their sum should be?

⁵ https://en.wikipedia.org/wiki/Pythagorean_theorem, Wikipedia will have to do, since Pythagoras never put a bibtex out...

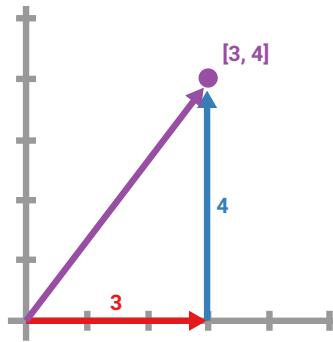


Figure 5.2: Decomposing the vector into its dimension coordinates to apply Pythagoras's theorem to find its length.

It will be the element-wise sum of their entries. So $[1, 3] + [2, 1] = [3, 4]$. Using a more abstract notation, “element-wise” means that, if you have two dimensional vectors u and v , then their sum is $[u_1 + v_1, u_2 + v_2]$. You sum the first element of u with the first element of v , then second with second, and so on (if you have more dimensions).

The result of the sum of two vectors is another vector with the same dimensions. Visually, this means that we’re taking the second vector and move its tail to the point of the first vector. The point where we end up is the new vector, the result of the sum. This is why it is sometimes easier to think of vectors as arrows rather than points, because this operation would be a bit less intuitive if we only have points. But arrows make it easy to see what we’re doing, which is what Figure 5.3 shows.

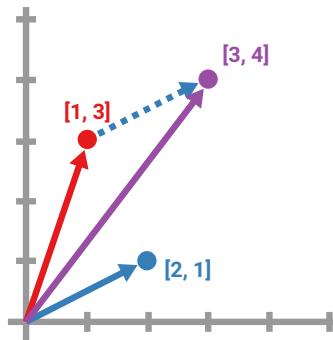


Figure 5.3: An example of vector sum.

This definition of the sum operation is consistent with how we defined vectors in the first place. Any vector could be considered the sum of itself to the origin point. If we have $[1, 2]$, that is equivalent to $[0, 0] + [1, 2]$ – i.e. you place the tail of $[1, 2]$ on the point of $[0, 0]$ and then you see where you end up. You can also think of vector sum as a sequence of movements in space. The first vector tells you the first movement. When you sum another vector to it, you take another movement, starting from where you ended up with the previous movement.

For the **multiplication**, for now we're only going to deal with the case of multiplying a number to a vector. As you can guess from the sum example, this means to multiply all the elements of the vector with that number: $2v = [2v_1, 2v_2]$. Visually, that means we extend the vector by that factor – as you can see in Figure 5.4. De facto, that means stretching the arrow's length while remaining on the same line – we have to flip the direction if we're multiplying by a negative number.

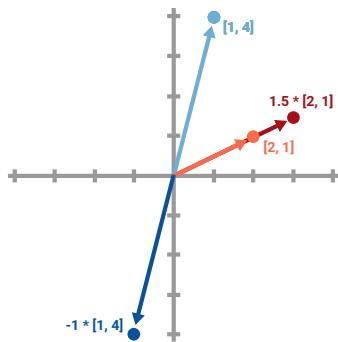


Figure 5.4: Two examples of vector multiplications by a number (scalar). The light color version is the original vector, and the darker version is the result of the multiplication (the scaling).

Multiplying by 2 means to end up with an arrow twice as long. Multiplying by a negative number means to reverse the direction before doing the stretching. And since stretching means to change the length of the arrow – i.e. to “scale” it – that’s the reason sometimes one will refer to numbers as “scalars”.

5.2 Matrices

Basic Properties and Operations

If vectors are lists of numbers, matrices are lists of lists of numbers. While you can write a vector as $[4, 1, 7]$, for a matrix you need to do something slightly more complicated:

$$\begin{pmatrix} 0 & 4 \\ 5 & 1 \end{pmatrix}.$$

Like vectors, matrices too have dimensions. The matrix above has two rows and two columns, so we say it is a 2×2 matrix. Since the number of rows and columns is the same, the matrix looks like a square and so we call it a square matrix. But nothing stops matrices from having a *different* number of rows and columns. For instance, a 3×2 matrix is a non-square (rectangular) matrix, a totally valid object. Notation wise, M_{ij} means to look at the value in the cell at the i th row and the j th column.

Before getting into what matrices are and what they are for, I want to list a few important things about matrices. First, square matrices have a *diagonal*, which goes from the top left element down to the bottom right. Second, there's the concept of symmetry. A square matrix is *symmetric* if you can mirror it along the diagonal and you get the same matrix. Mathematically, this means that is is always the case that $M_{ij} = M_{ji}$.

Note how non-square matrices do not have a diagonal and cannot be symmetric. *Transposing* a matrix M means that all M_{ij} values become M_{ji} and viceversa. In this book, for convention, M^T will be the transpose of M . Figure 5.5 shows the case of a squared non symmetric matrix transpose. In practice, transposing is like placing a mirror on the diagonal.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 |
| 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |

(a) M

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |

(b) M^T

If your matrix is squared and symmetric transposing has no effect: $M^T = M$. However, for non symmetric matrices, $M^T \neq M$. Moreover, for non square matrices, transposing an $n \times m$ matrix results into an $m \times n$ one, the dimensions flip.

Figure 5.5: A matrix and its transpose. Note how the (i,j) entries of M equal to one transposed to the (j,i) entries of M^T , for instance (1,3) to (3,1).

Vector-Matrix Multiplication

To keep our spatial interpretation of linear algebra, in the case of matrices we need to jump right to the multiplication of a matrix with a vector. If you have matrix M and vector v , then $Mv = w$: multiplying vector v with matrix M results a new vector w . The new vector w will have different coordinates from v – with few interesting and useful exceptions we'll get to later. In practice, M is moving v so that it ends up in w . Alternatively, you can say that M is changing the coordinate system of v : w is still the same as v , but in a different coordinate system.

So far this isn't helping much, it's still pretty abstract. What does the matrix I showed you before *really means*? Each column of the matrix tells you how to change each coordinate in isolation. This is the same as stretching a vector that is equal to one in that given dimension, and zero everywhere else. So, since the first column of the matrix is $[0, 5]$ then we know that the unit vector $[1, 0]$ will end up

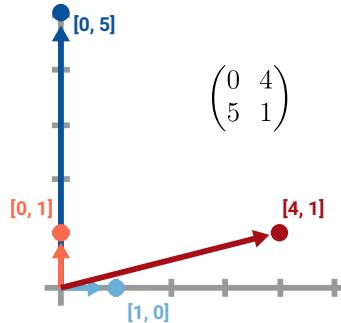


Figure 5.6: A visualization of a matrix. The light color vectors are the original unit vectors, and their darker version is the result of the coordinate transformation applied by the matrix.

in $[0, 5]$. The second column tells you that the second unit vector $[0, 1]$ will end up in $[4, 1]$. Figure 5.6 shows you this transformation.

Once you know this, you know how to move any two vectors, because any vector is a combination of these two unit vectors. To know where any arbitrary vector $v = [v_1, v_2]$ ends – to calculate w – you need to look at the matrix first by rows: the first row tells you the contribution of the matrix to the first entry of w . Then you look at columns: the first column tells you the effect of v_1 , the second of v_2 , and so on. To sum up in general terms, $Mv = w$ means that $w_1 = M_{1,1}v_1 + M_{1,2}v_2$ and $w_2 = M_{2,1}v_1 + M_{2,2}v_2$. You can see an example in Figure 5.7.

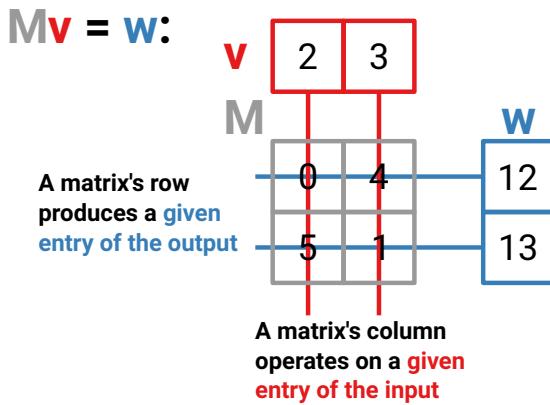


Figure 5.7: A example of vector-matrix multiplication. Each entry of the result vector depends on the corresponding row of the matrix. Each entry of the input vector is handled by the corresponding column of the matrix.

It's important to remember that M cannot change the coordinate system in any arbitrary way. We're still in *linear* algebra, so M can only change coordinates linearly. That is, if you have a line in a coordinate system, after applying M that is still going to be a line – maybe longer or shorter, maybe pointing in another direction, but it won't curve. We also want to maintain the origin – in the case of 2D space $[0, 0]$ – in its place.

You might be wondering what happens when we use a non-square matrix to perform the transformation. What does it mean to multiply $[1, 0]$ to a 3×2 matrix? Well, you just learned that the first column of

the matrix gives you the coordinates of where $[1, 0]$ lands. If a matrix is 3×2 , it means that $[1, 0]$ will land on a 3D space, because it will have three coordinates. So non-square matrices allow you to move between spaces with a different number of dimensions.

In our $Mv = w$, besides w_1 and w_2 , we also have $w_3 = M_{3,1}v_1 + M_{3,2}v_2$. More generally: $w_i = \sum_{k=1}^n M_{ik}v_k$. This formula sneakily tells you one important thing: you cannot multiply any vector-matrix combination. The matrix must have the same number of columns as the number of entries in the vector. Otherwise you either have some entries of v you can't transform, or portions of M doing nothing. In general, multiplying a d dimensional vector with a $n \times d$ matrix will result in an n dimensional vector. Figure 5.8 shows you this operation, and highlights how this coordinate change moves to a completely new 2D space from the original 3D one.

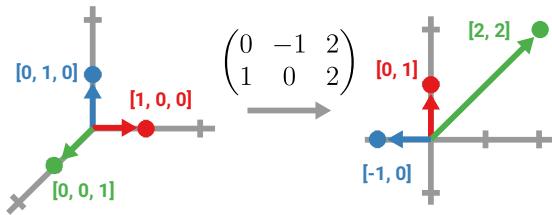


Figure 5.8: A visualization of a non-square matrix. Each unit vector in the 3D space gets mapped to a new coordinate system in a 2D space.

Possibly the most useful matrix for this book is the identity matrix, which we call I . This is defined as a matrix that has ones on its main diagonal – the one going from top left to bottom right, and zero everywhere else:

$$\begin{pmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{pmatrix}.$$

Can you guess what it does? Each column in this matrix is exactly the unit vector for that specific dimension. Since it tells you what the unit vector becomes after the transformation, what this means is that the unit vector will not change. If it does not change, nothing will! The identity matrix will preserve the coordinate system exactly as it is. Mathematically, for any v : $Iv = v$.

Matrix Multiplication

It is often the case that you might want to multiply two matrices together, rather than a matrix and a vector. It's worth explaining what that operation means intuitively, because that will make it easy to understand why we achieve it the way we do.

If a matrix is a change in the coordinate system, the multiplication of two matrices A and B is the effect of making first the A transformation and then transform the result with B . The concatenation – or composition – of these two transformations is by itself another transformation, so it must be a matrix too, C . So, matrix multiplication is an operation that produces a matrix C from two matrices A and B .

Figure 5.9 shows an example. The first coordinate change moves $[1, 0]$ to $1[1, 2] + 0[2, -1] = [1, 2]$, then the second change moves $[1, 2]$ to $1[2, 0] + 2[1, 1] = [4, 2]$, so $[4, 2]$ must be the first column of the result.

$$\begin{pmatrix} 2 & 1 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 2 \\ 2 & -1 \end{pmatrix} = \begin{pmatrix} 4 & 3 \\ 2 & -1 \end{pmatrix}$$

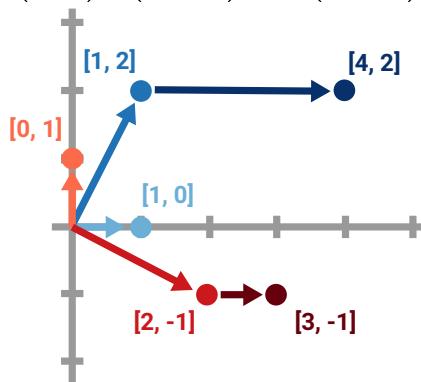


Figure 5.9: A visualization of a matrix multiplication. The lightest color vectors are the original unit vectors. The midway dark version is the application of the first coordinate change, the darkest vectors apply the second coordinate change on top of that.

With formal notation, what we're saying here is that first we transform v with A as Av . Then we transform that result with B : BAv . That must equal Cv . So it is saying: $BAv = Cv$. And, at that point, you can simplify as $BA = C$, since this will work for any v . Note how we read this right to left: BA means applying A and then applying B to the result. In most cases, $BA \neq AB$.

The advantage of thinking of this in terms of concatenation of transformations is that it immediately reduces it to the case of multiplying a vector to a matrix, since it's the same thing. The first column of A tells you where $[1, 0]$ lands, then the first column of B tells you where that results lands in turn. If you expand this mathematically, you get that each C_{ij} entry of C is equal to the sum of the products of all entries in the i th row of A and the j th column of B . Formally:

$$C_{ij} = \sum_{k=1}^m A_{ik} B_{kj}.$$

Just like in the case of vector-matrix multiplication, you also cannot multiply any two matrices together. A and B must have one dimension of equal size. So a $n \times m$ matrix can be multiplied by a $m \times x$ matrix to generate a $n \times x$ matrix. In this case, the common dimension "disappears". Figure 5.10 shows a graphical representation of the algorithm to multiply two non-square matrices – which also

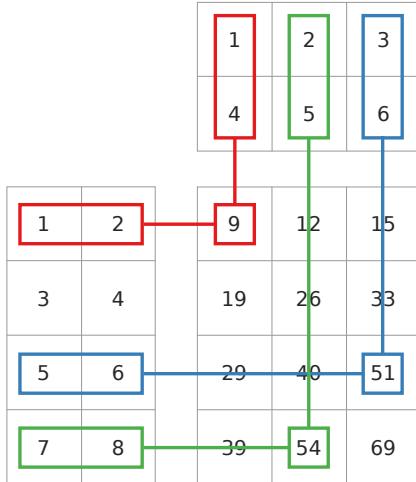


Figure 5.10: An example of matrix multiplication. Each cell is the result of the combination of the rows/columns of the corresponding color, whose element-wise products are summed. So the red cell equals to 9 because it is the sum of 1×1 (the product of the first elements) plus 2×4 (the product of the second elements).

generalizes to multiplying square matrices.

5.3 Tensors

Finally, we get to the concept of tensor. The tensor is the generalization of vectors and matrices. Here we clash a bit in terminology, because we already defined what a dimension is in linear algebra: the number of entries in a vector. Abusing terminology a bit, we can use the term “dimension” to mean another thing. You can think of vectors – with any number of entries – as one-dimensional lists of numbers, because they look like a line: $[1, 3, 6, 3, 8, 2, \dots]$. On the other hand, matrices look like two-dimensional rectangles:

$$\begin{pmatrix} 1 & 3 & 4 & \dots & 3 \\ 8 & 5 & 4 & \dots & 9 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & 2 & 7 & \dots & 8 \end{pmatrix}.$$

The tensor generalizes this intuition. A vector is a one dimensional tensor, it only needs one index to identify its entries v_i . A matrix is a two dimensional tensor, needing two indices for its entries M_{ij} . Then you can have three dimensional tensors, with three indices like T_{ijk} , four dimensional with four indices (T_{ijkl}) and so on. In fact, you can also generalize in a different direction. To indicate a number, you need no index. So a number is a zero dimensional tensor.

This latter generalization also shows you how much it makes sense to make vector sums the way I explained in Section 5.1. The normal sum you’re used to is defined the same way, for zero dimensional tensors. When we say $2 + 3 = 5$, what we mean is that we first move to the end of the first zero dimensional tensor to go from 0 to 2, then

we apply the movement from the second zero dimensional tensor 3 to end up in the final zero dimensional tensor 5. Figure 5.11 shows you exactly this.

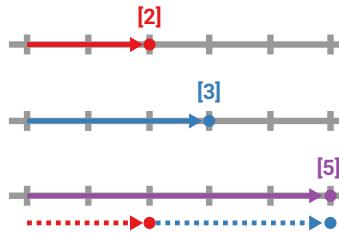


Figure 5.11: An example of sum for two zero dimensional tensors.

Note that the dimension of the tensor has no relation with the dimension of the space in which the tensor lives! A vector is a one dimensional tensor, but can live in a 3D space, if it has three entries. Vice versa, a 2D space can contain a three dimensional tensor, for instance $[[[1, 2], [2, 4]], [[3, 1], [2, 5]]]$ is an example of such a tensor. Confusing, I know!

5.4 Vector Products

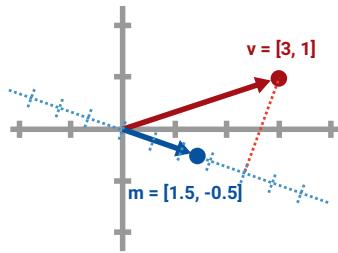
Dot Product

The fact that both vectors and matrices are tensors suggests a profound thing: there is no qualitative distinction between a vector and a matrix. This is indeed true, and one cool repercussion is that vectors, just like matrices, are *also* coordinate shifts. That is because any d dimensional vector is also a $d \times 1$ rectangular matrix. So it can play a role in the vector-matrix and matrix-matrix multiplications I explained in Section 5.2.

This is the dot product. Let's say you have two d dimensional vectors: v and m . We can decide that m is actually a $d \times 1$ matrix and we use it to perform a vector-matrix multiplication with v . Since m is a rectangular matrix, by now you know that it will transport you to a space with a different number of dimensions. In this case, you'll end up with only one dimension. A one-dimensional vector is a number.

What this means in our spatial perspective is that you're projecting v onto the line defined by the direction pointed by m . This is because you're bringing v into the 1D space of m , which is a number line. You will also have to stretch v 's projection proportionally to the length of m , just like entries in a matrix M will stretch or squish the space if they are different than 1. Figure 5.12 shows you how this looks like spatially.

Note how in Figure 5.12 the red vector v lands on the fourth tick-mark of the number line determined by the direction of vector m .



In fact, the dot product of $v = [3, 1]$ and $m = [1.5, -0.5]$ is exactly four, because it so happens that m does not stretch anything on this number line – it is its unit vector. However, if we were to do the dot product with another vector on that same number line – say $m' = [3, -1]$ – we will have to do some stretching. In this case, since m' is twice as long as m , we will have to multiply the result of the projection by two. In fact, the dot product between v and m' is exactly 8.

Mathematically, you're not doing anything differently than what you already learned in vector-matrix multiplication, only that now you have only one column to worry about. The dot product between v and m is still $w = \sum_{k=1}^n m_k v_k$, which as you can see is a single number.

Notation-wise, what you're doing with the dot product is $m^T v$. I want to point out one thing that will be useful down the road. You can dot product a vector with itself: $v^T v$. This is normally known as the “quadratic sum” because, if you expand this mathematically, that's what it is: $v^T v = \sum_{k=1}^n v_k^2$. Looks familiar? It's the same sum to get the vector's length. So another way to calculate v 's length is $\sqrt{v^T v}$. This actually generalizes to calculating the distance between two vectors: you simply need to take their difference and put it into the same formula. So the distance between u and v is $\sqrt{(u - v)^T (u - v)}$. The difference between u and v is simply the vector sum of u to $-1v$, both things you know the definition of from Section 5.1.

Outer Product

There's another type of product that is useful in several parts of this book. It is the cousin of the dot product: the outer product. When we decided to multiply two $d \times 1$ vectors, we decided we were collapsing the d dimension to get a 1 dimensional vector, a number. But that's not the only option. We could have instead collapsed the 1 dimension and the result would be... can you guess it? A $d \times d$ matrix! In fact, if we decide to collapse the 1 dimension, it actually doesn't matter

Figure 5.12: An example of dot product. The blue dashed line is the number line defined by the dark blue vector m . The red dashed line shows the projection of the dark red v vector onto the number line.

whether u and v have the same dimension. Any two vectors can have an outer product. In this book I'll write the outer product as $u \otimes v$.

Formally, this looks like:

$$\mathbf{u} \otimes \mathbf{v} = \mathbf{u}\mathbf{v}^T = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} \begin{bmatrix} v_1 & v_2 & v_3 \end{bmatrix} = \begin{pmatrix} u_1v_1 & u_1v_2 & u_1v_3 \\ u_2v_1 & u_2v_2 & u_2v_3 \\ u_3v_1 & u_3v_2 & u_3v_3 \\ u_4v_1 & u_4v_2 & u_4v_3 \end{pmatrix}.$$

In practice, the outer product of two vectors u and v is a $|u| \times |v|$ matrix, whose (i, j) entry is the multiplication of u_i to v_j .

Positive (Semi)Definite Matrices

Positive definiteness is an interesting property for a matrix which will come in handy in Chapter 47, so it's worthwhile to introduce it here. Suppose you have a vector v of length m and a $n \times m$ matrix M . It follows from the properties of matrix multiplication that you can always multiply them – because a vector of length m is a $m \times 1$ matrix. Hopefully, you know what the result would be: a vector of length n – remember: the common dimension “disappears”. For the very same reason, you can always multiply a vector with the transpose of itself. $v^T v$ is a legit operation, as we just saw when talking about the dot product, and produces a number. If we put together what we discovered in the previous two paragraphs: if M is a square matrix, then $v^T M v$ is a scalar.

Now, some matrices M are special. For these special matrices, it doesn't matter what you put in v , as long as it is a vector of real numbers: the result of $v^T M v$ is always going to be greater than zero. We call these special matrices “positive definite”. Relaxing the concept a bit, if $v^T M v \geq 0$ for any real number v , then M is positive semi-definite – “semi” because we allow the result to be zero sometimes.

The identity matrix I that I introduced a while ago is positive semi-definite. This means that any $v^T I v$ is going to be zero or positive. In fact, by properties of the identity matrix, $v^T I v = v^T v$: the identity matrix does nothing, remember? So why did I mention this fact? Well, we can expand our formula of the Euclidean distance between two vectors as $\sqrt{(u - v)^T I (u - v)}$ without fear, because we know everything checks out. This gave some people some ideas. All it matters for a matrix to be a good fit in this formula and replace I is that they have to be positive semi-definite. That's because the formula is under a square root and we don't want negatives there. The consequence is that any positive semi-definite matrix replacing I will also define a

special distance in a non-Euclidean space! (Admittedly, this excites me much more than it should)

This is the exact thing that you do, for instance, when calculating a Mahalanobis distance – which is a smart Euclidean that takes into account the correlation between the vectors, see Section 47.1. The Mahalanobis distance is $((p - q)^T \text{cov}(p, q)^{-1} (p - q))^{1/2}$, where $\text{cov}(p, q)^{-1}$ is the inverse of the covariance matrix between p and q . The covariance matrix is a matrix whose element in the i, j position is the covariance (Section 3.4) between the i -th and j -th elements of p and q . Surprise surprise, $\text{cov}(p, q)^{-1}$ is positive semidefinite.

This will be immensely useful in Chapter 47 when I will show how there are secret identity matrices hidden in many formulas that can be replaced with matrices representing your network, to expand many classical statistical measures into network statistical measures. It'll be uber cool, I pinky promise!

5.5 Eigenvalues and Eigenvectors

Two absolutely key concepts for network science are eigenvalues and eigenvectors. They are used almost everywhere in network science, so we need to define them here. Consider Figure 5.13. Given a vector v , we learned we can apply an arbitrary matrix transformation M to it – as long as it has the correct dimensions. We then obtain a new vector $w = Mv$. Any transformation M has special vectors: M scales these special vectors without altering their *directions*. In practice, the transformation M simply multiplies the elements of such vectors by the same scalar λ : $w = Mv = \lambda v$. We have a name for this: v is M 's eigenvector and λ is its associated eigenvalue. Mathematically, we represent this relation as $Mv = \lambda v$.

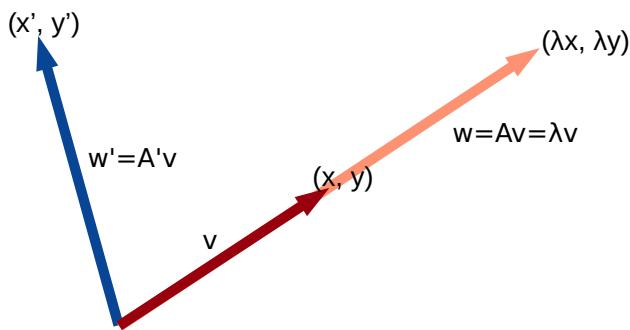


Figure 5.13: A graphical depiction of an eigenvector.

The formula we just introduced is the one for *right* eigenvectors, because the vector multiplies the matrix *from the right*. If M is square, there are also *left* eigenvectors, which multiply the matrix *from the left*: $vM = v\lambda$. Right and left eigenvectors are different, have different

values, but their corresponding eigenvalues are the same. From now on, when I mention eigenvectors, I refer to the *right* eigenvectors.

Right eigenvectors are the *default*, and when I refer to *left* eigenvectors I will explicitly acknowledge it.

A square matrix with n rows and columns also has n eigenvalues. By convention, we sort eigenvalues by their value, sometimes in increasing order, sometimes in decreasing order, depending on the application.

A key term you need to keep in mind is “multiplicity”. The multiplicity of an eigenvalue is the number of eigenvectors to which it is associated. If you have an $n \times n$ matrix, but only $d < n$ distinct eigenvalues, some eigenvalues are associated to more than one eigenvector. Thus their multiplicity is higher than one.

5.6 Matrix Factorization

In some cases, you might want to express a matrix as the result of the multiplication of other matrices. This can be useful because the matrices you use to reconstruct your observed matrix might be made of pieces you can more easily interpret. We call this decomposition of a matrix “factorization”, because we divide the matrix into its “factors”, its building blocks. There are countless ways to factorize a matrix. Here we examine only the ones that you’re most likely to encounter in network analysis. I divide them in two classes: the ones operating on regular bi-dimensional matrices, and the ones which work on scary and confusing multidimensional matrices (i.e. tensors).

Matrix Decomposition

One of the easiest ways to perform matrix factorization is what we call “eigendecomposition”. A square matrix M can always be decomposed as $M = \Phi\Lambda\Phi^{-1}$. Rather than being the left and right eyes of a really pissed frowny face, Φ is the matrix we obtain piling all eigenvectors next to each other, and Λ is a diagonal matrix with the eigenvalues on its main diagonal and zeros everywhere else:

$$\Lambda = \begin{pmatrix} \lambda_0 & \dots & 0 \\ 0 & \ddots & 0 \\ 0 & \dots & \lambda_n. \end{pmatrix}$$

We are mostly interested in eigendecomposition as the special case of the more general Singular Value Decomposition (SVD) – which can be applied to any matrix, even non-square ones. In SVD, we simply replace Φ and Λ with generic matrices. In other words, we say that we can reconstruct M with the following operation:

$M = Q_1 \Sigma Q_2^T$. Like Λ , also Σ is a diagonal matrix. The difference is that Σ contains the singular values of M , rather than its eigenvalues. While there is only one valid Σ to solve this equation – that is why it is called “singular” – there could be multiple Q_1 and Q_2 matrices that you could plug in, as long as they’re both unitary matrices. A unitary matrix Q is a matrix whose transpose is also its inverse: $QQ^{-1} = I = QQ^T$, with I being the identity matrix. SVD is especially useful for estimating node distances on networks (Section 47.2).

Along with eigendecomposition, the two most common and useful matrix decomposition tools are the Principal Component Analysis (PCA) and the Non-Negative Matrix Factorization (NMF).

To understand PCA, suppose that your matrix is just a set of observations and variables. Each row of the matrix is an observation and each column is a variable. PCA, like NMF, is used to summarize this matrix of data. If two columns/variables are correlated it means they contain redundant information. Thus, you’re after a way to describe your data in such a way that each variable has no redundant information.

Figure 5.14 shows an example: each row is a day and each column is some measurement taken in that day – the temperature, wind speed, the millimeters of rain/snow that fell that day, etc. You might expect that some of these variables might be correlated. For instance, it is very difficult to have a single millimeter of snow if the temperature is above a certain value. Rather than describing a day by all variables, you want to describe it by its similarity with an “archetypal” day: is this a snow day or a rain day?

| Day | Temp (°C) | Wind (km/h) | Sunlight (%) | Rain (mm) | Snow (mm) |
|-----|-----------|-------------|--------------|-----------|-----------|
| 1 | 27 | 10 | 80 | 2 | 0 |
| 2 | 26 | 1.2 | 95 | 1 | 0 |
| 3 | 32 | 7.6 | 100 | 0 | 0 |
| 4 | 12 | 2.3 | 12 | 20 | 0 |
| 5 | 14 | 3.8 | 8 | 25 | 0 |
| 6 | 6 | 0.2 | 24 | 40 | 1 |
| 7 | 4 | 0.1 | 2 | 8 | 30 |
| 8 | 2 | 0.9 | 4 | 1 | 40 |
| 9 | -1 | 1.1 | 4 | 0 | 80 |

This is the aim of PCA. Let’s repeat the previous paragraph mathematically: you want to transform your correlated vectors in a set of uncorrelated, or orthogonal, vectors which we call “principal components”. Each component is a vector that explains the largest possible amount of variance (Section 3.4) in your data, under the condition of being orthogonal with all the other components. You can have as many components as you have variables, but usually you want much

Figure 5.14: A table recording in a matrix the characteristics of some days.

fewer – for instance two, so you can plot the data. That is because the first component explains the most variance in the system, the second a bit less, and so on, until the last few components which are practically random. Thus you want to stop collecting components after you've taken the first n , setting n to your delight. In Figure 5.15, I collect the first two – they're there for illustrative purposes so don't be shocked if you realize they're not really orthogonal.

| Day | PC1 | PC2 |
|-----|-------|-------|
| 1 | 0.2 | -0.05 |
| 2 | -0.05 | 0.1 |
| 3 | -0.1 | -0.1 |
| 4 | 2.6 | -0.01 |
| 5 | 2.9 | 0 |
| 6 | 3.2 | 0.35 |
| 7 | 0.3 | 2.4 |
| 8 | 0.1 | 2.6 |
| 9 | -0.05 | 2.8 |

PCA is extremely helpful when performing data clustering. Suppose that we're looking only at the first two principal components of our matrix describing our days. We can make a two dimensional scatter plot of the system, with one point per day. It might look like Figure 5.16. This seems successful, because we can clearly see three clusters: days dominated by the first component (in blue), days dominated by the second component (in green), and days which have low values in both (in red). When we look at the original data, we might recognize that the first class of days had high rain precipitation, the second high snow precipitation, and the third group was mostly sunny days. In this sense, PCA aided us in finding our archetypal days: the first component describes the archetypal rainy day, while the second component describes the archetypal snowy day.

PCA has no restrictions in the way it builds the principal compo-

Figure 5.15: The first two principal components of the matrix in Figure 5.14.

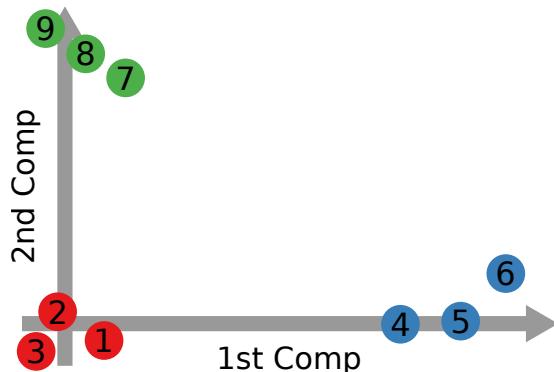


Figure 5.16: A scatter plot with the first principal component of the matrix in Figure 5.14 on the x axis and the second component on the y axis.

nents, besides the fact that all these components must be orthogonal with each other. This means that you might end up with components with negative values, as we do in Figure 5.15. This might not be ideal. What does it mean for a day to be a “negative rainy day”? PCA is interpretable, but sometimes the interpretation can be a bit... confusing.

Non-Negative Matrix Factorization solves this problem. Without going into technical details, NMF is PCA with the additional constraint that no component can have a negative entry – hence the “Non-Negative” part in the name. At a practical level, if there were no negative entries in Figure 5.15, then the two components in that figure could be results of NMF. This additional constraint comes at the expense of some precision: PCA can fit the data better because it does not restrict its output space. However, usually, NMF components are more easy to interpret.

Given their links to data clustering, both PCA and NMF are extensively used when looking for communities in your networks (Part X).

Tensor Decomposition

Let’s assume you have a three dimensional tensor. You can think of a tensor as a cuboid and a slice of it is a matrix. If you find it difficult to picture this in your head, don’t worry: you’re not alone. That is why many researchers put effort into finding ways to decompose tensors in lower-dimensional representations that can sum up their main properties. This process is generally known as “tensor decomposition”.

Tensor decomposition is a general term encompassing many techniques to express a tensor as a sequence of elementary operations (addition, multiplication, etc) on other, simpler tensors. For instance, you can represent a 3D tensor as a combination of three vectors, one per dimension. Or as a matrix and a vector. You want to do this to solve complex network analyses on multilayer networks – whatever the hell this means, Section 8.1 will enlighten you – by taking the full dimensionality into account at the same time, rather than performing the analysis on each layer separately and then merge the results somehow. Examples of applications of tensor decomposition range from node ranking (Chapter 14), to link prediction (Part VII), to community discovery (Part X).

I am going to mention very briefly only two of these techniques: tensor rank decomposition and Tucker decomposition. You should look elsewhere for a more complete treatment of the subject⁶. There also exists a tensor SVD^{7,8}, but it is relatively similar to a special case

⁶ Tamara G Kolda and Brett W Bader. Tensor decompositions and applications. *SIAM review*, 51(3):455–500, 2009

⁷ Lieven De Lathauwer, Bart De Moor, and Joos Vandewalle. A multilinear singular value decomposition. *SIAM journal on Matrix Analysis and Applications*, 21(4):1253–1278, 2000

⁸ Elina Robeva and Anna Seigal. Singular vectors of orthogonally decomposable tensors. *Linear and Multilinear Algebra*, 65(12):2457–2471, 2017

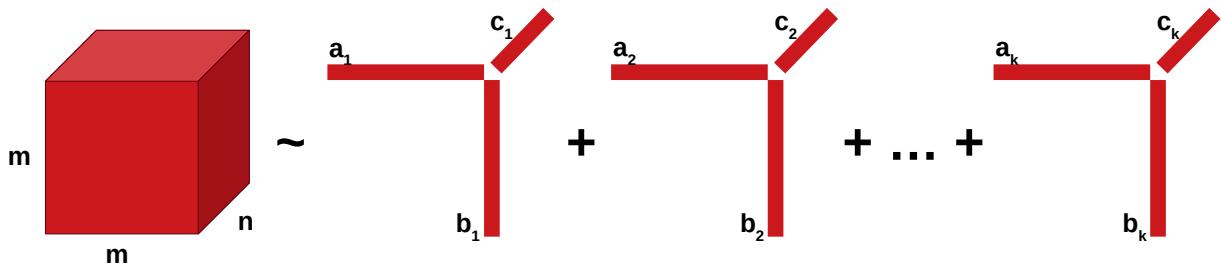
of Tucker decomposition, so I will not cover it.

Tensor rank decomposition is the oldest of the two⁹ and has historically been referred to as PARAFAC¹⁰ or CANDECOMP¹¹.

Let's say you have your nice 3D tensor T . This is a three dimensional matrix of dimensions $n \times n \times m$ – for simplicity here we assume that the slices of the tensor are square matrices, thus two dimensions are the same, however the method also works if all three dimensions are different. Tensor rank decomposition tells you that there is a way to decompose T as the following combination:

$$T \sim \sum_k \lambda_k a_k \otimes b_k \otimes c_k.$$

Here, $T = a \otimes b \otimes c \rightarrow T_{ijk} = a_i b_j c_k$ which means that \otimes represents the outer product. a and b are vectors of length n and c is a vector of length m . Finally, λ_k is just a scaling factor that tells us how much to count the k th element of the sum. The convention is to call $\lambda_k a_k \otimes b_k \otimes c_k$ a component, while the vectors are the factors.



If you find difficult to understand what's going on by just looking at the formula, take inspiration from Figure 5.17. What this operation does is to find the right set of one-dimensional vectors a , b and c such that, once they are scaled by factors λ , they can best represent the full tensor T . At that point, you are working in a lower dimensional space and all the rest of linear algebra starts making sense again.

How many components does this sum have? Or, in other words, how big should k be to approximate T ? That depends on the rank of the tensor. Unfortunately, calculating the rank of a tensor isn't as easy as calculating the rank of a matrix. The rank of a matrix is the number of columns (or rows) that are linearly independent from each other. The definition is the same for a tensor but, in this case, there is no straightforward algorithm to determine it¹². What happens is that, to find the rank of a tensor, you would literally apply the rank decomposition with different k values and find the one that works the best.

Tucker decomposition¹³ takes a different approach. It decomposes our tensor T into a smaller core tensor and a set of matrices. If we

⁹ Frank L Hitchcock. The expression of a tensor or a polyadic as a sum of products. *Journal of Mathematics and Physics*, 6(1-4):164–189, 1927

¹⁰ Richard A Harshman et al. Foundations of the parafac procedure: Models and conditions for an "explanatory" multimodal factor analysis. 1970

¹¹ J Douglas Carroll and Jih-Jie Chang. Analysis of individual differences in multidimensional scaling via an n -way generalization of "eckart-young" decomposition. *Psychometrika*, 35(3):283–319, 1970

Figure 5.17: A schema of tensor rank decomposition.

¹² Joseph B Kruskal. Three-way arrays: rank and uniqueness of trilinear decompositions, with application to arithmetic complexity and statistics. *Linear algebra and its applications*, 18(2):95–138, 1977

¹³ Ledyard R Tucker. Some mathematical notes on three-mode factor analysis. *Psychometrika*, 31(3):279–311, 1966

keep our simplified case of a 3D tensor representing an adjacency matrix, mathematically speaking the Tucker factorization does:

$$T \sim \mathcal{T} \times X \times Y \times Z.$$

Here, \mathcal{T} is the core tensor, whose dimensions are smaller than T 's. X, Y , and Z are matrices which have one dimension in common with T and the other in common with \mathcal{T} – so that the matrix multiplication of them with \mathcal{T} reconstructs a tensor with T 's dimensions.

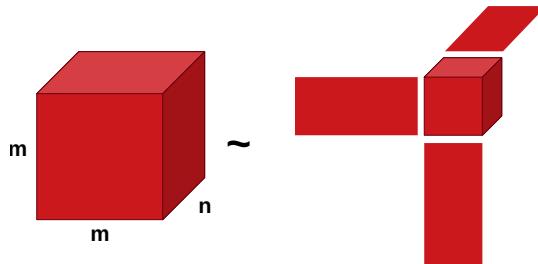


Figure 5.18: A schema of Tucker decomposition.

Again, for the visual thinkers, Figure 5.18 might come in handy. In Tucker decomposition you have the freedom to choose the dimensions of the core tensor \mathcal{T} . Smaller cores tend to be more interpretable, because they defer most of the heavy lifting to X, Y , and Z . However, they also tend to make the decomposition less precise in reconstructing T .

5.7 Summary

1. Vectors are lists of numbers, which specify the coordinates of a point in space. The number of entries in the list tells you the number of dimensions of its space. The length of a vector is its Euclidean distance to the origin.
2. Summing two vectors means to go to the point of the first vector and then move according to the coordinates of the second vector, but starting from where you landed rather than from the origin. So the result is another vector.
3. Matrices are coordinate transformations: multiplying a vector to a matrix means to find the coordinates of the vector in the new space transformed by the matrix. Multiplying two matrices gives another matrix, which performs both transformations one after the other.
4. Transposing means mirroring the matrix on its main diagonal. It can be used to flip the direction of edges in a directed network,

or looking at two different modes of connections in a bipartite network.

5. A matrix's eigenvector is a vector that gets stretched by a factor (the eigenvalue) but does not change direction when multiplied by that matrix.
6. In Principal Component Analysis, we deconstruct a matrix in its “principal components”: uncorrelated vectors that express most of the variation in the (correlated) values of the matrix. If no value in these vectors can be negative, we call it “Non-Negative Matrix Factorization”.
7. Tensor decomposition is an operation expressing a multidimensional matrix as the result of the sum/product or other, simpler, tensors.

5.8 Exercises

1. What is the length of the vector you obtain by summing $[0, 4]$ to $[5, 1]$?
2. Suppose:

$$A = \begin{pmatrix} 1 & 0 \\ 0 & 2 \end{pmatrix} \quad B = \begin{pmatrix} 3 & 0 \\ 0 & -1 \end{pmatrix}$$

Are these two transformations commutative? Does applying A first and B second lead to the same transformation as applying B first and A second?

3. Calculate the eigenvalues and the right and left eigenvectors of the matrix from <http://www.networkatlas.eu/exercises/5/3/data.txt>. Make sure to sort the eigenvalues in descending order (and sort the eigenvectors accordingly). Only take the real part of eigenvalues and eigenvectors, ignoring the imaginary part.
4. Perform the eigendecompositions of the matrices from exercise 2, showing that you can reconstruct the originals from their eigenvalues and eigenvectors.

Part II

Graph Representations

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 bit⁵, 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 Ramchandra 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.

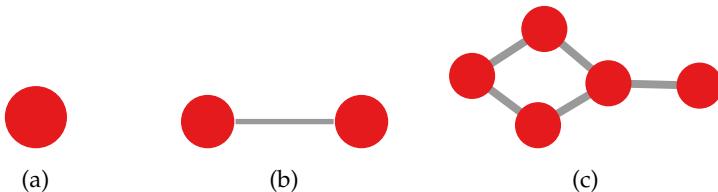


Figure 6.1: (a) A node. (b) An edge. (c) A simple graph.

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

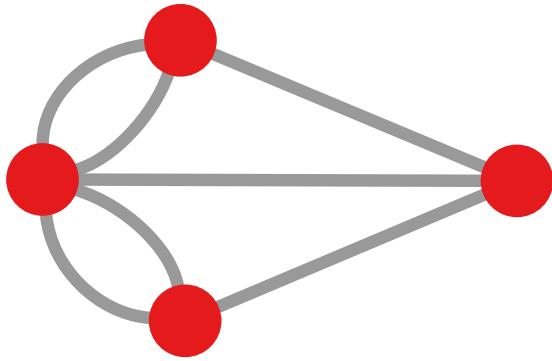


Figure 6.2: The famous Königsberg graph Euler used.

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.

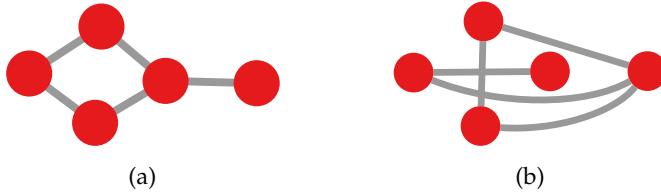


Figure 6.3: (a) A simple graph.
(b) Its complement.

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

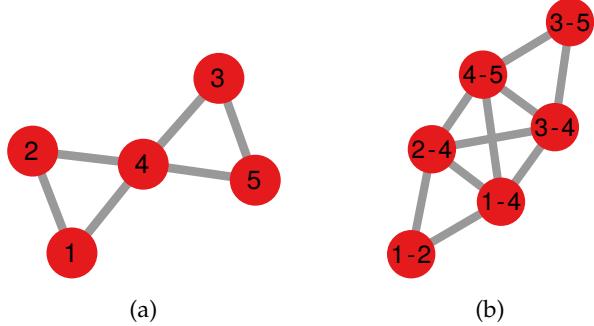


Figure 6.4: (a) A graph. (b) Its linegraph version.

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 graphs⁸ 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

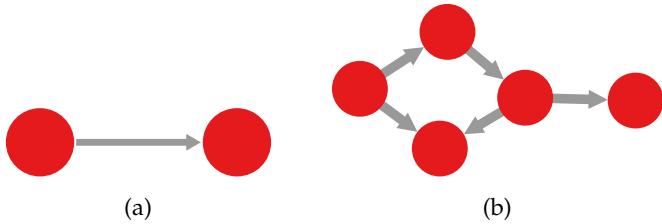


Figure 6.5: (a) A directed edge.
 (b) A directed graph.

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, Rómulo 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

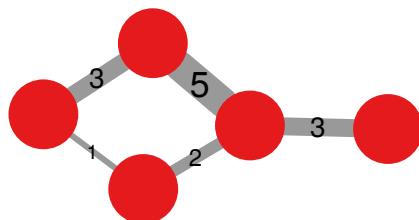


Figure 6.6: A weighted graph.
The weight of the edge dictates
its label and thickness.

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 correlations¹¹, or currencies according to their exchange rate¹², or correlating the taxa presence in different biological communities¹³.

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 than it might sound. We will explore it in Chapter 27.

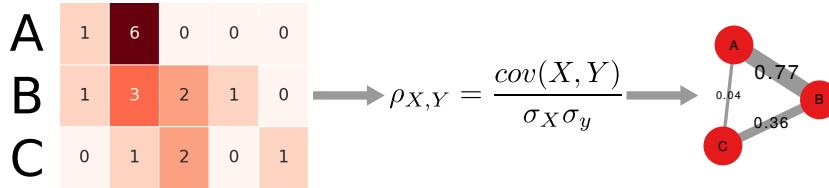


Figure 6.7: A typical workflow for correlation networks: (left to right) from nodes represented as some sort of vectors, to a graph with a similarity measure as edge weight.

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)).

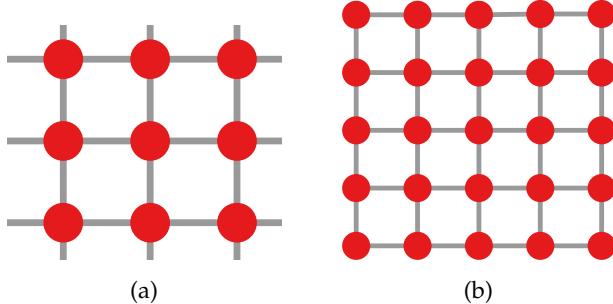


Figure 6.8: (a) An infinite lattice without boundaries. (b) A finite lattice with 25 nodes and 40 edges.

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.

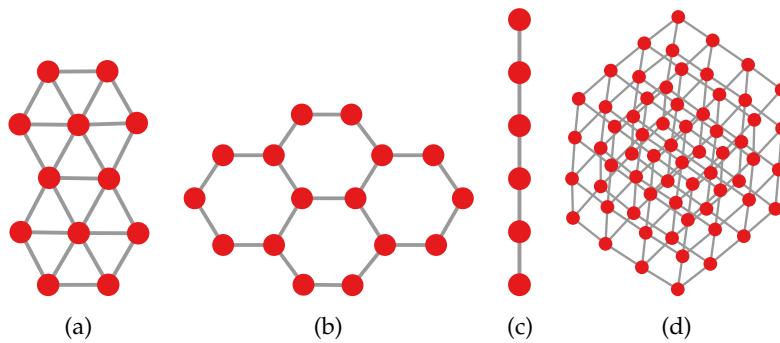


Figure 6.9: Different lattice types. (a) Triangular. (b) Hexagonal. (c) One dimensional. (d) Three dimensional cube.

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 graph¹⁴ (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

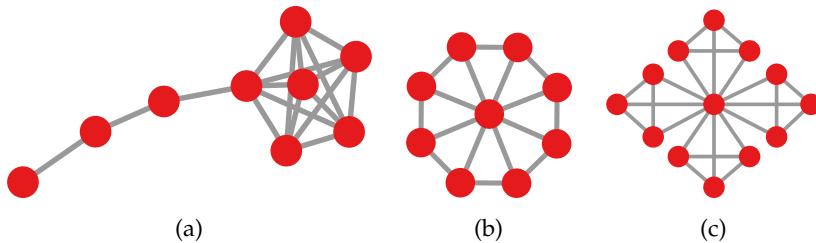


Figure 6.10: Different simple networks. (a) Lollipop graph. (b) Wheel graph. (c) Windmill graph.

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

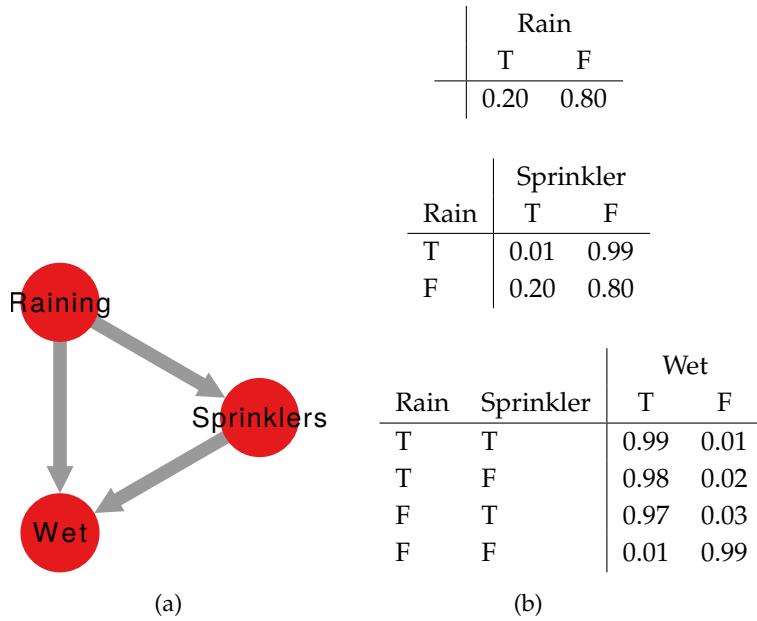


Figure 6.11: (a) A Bayesian network. (b) The conditional probability tables for the node states. The tables are referring to, from top to bottom: Rain, Sprinkler, Wet.

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.

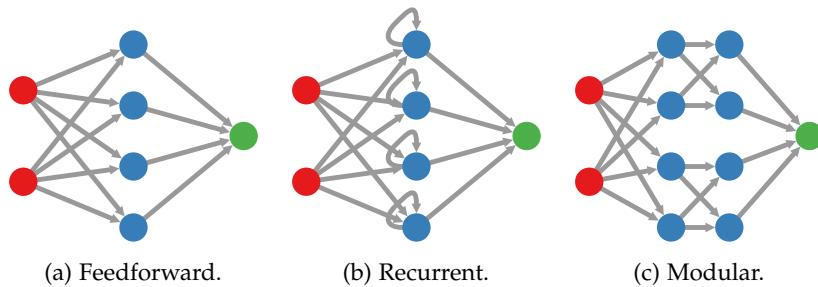


Figure 6.12: Different neural networks. The node color determines the layer type: input (red), hidden (blue), output (green).

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

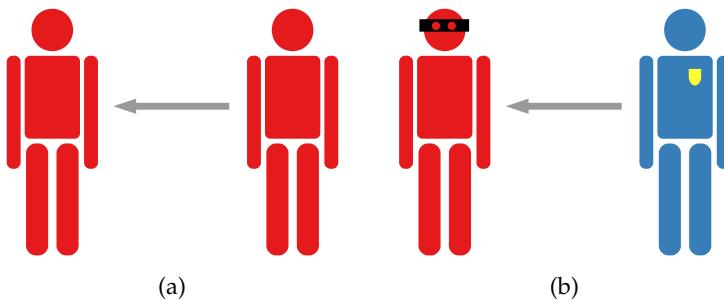
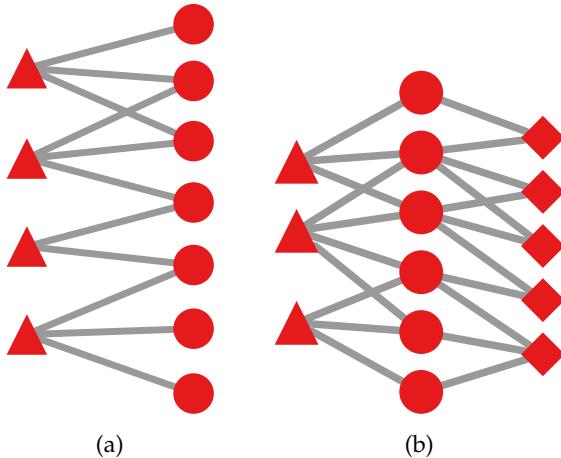


Figure 7.1: (a) A simple graph representing a social network with no additional constraints. (b) A cop-thief bipartite network: nodes can be either a cop or a thief, and cops can only catch (connect to) thieves.



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 export³, hosts to guest in symbiotic relationships⁴, users to the social media items they tag⁵, bank-firm relationships in financial networks⁶, players-bands in jazz⁷, listener-band in music consumption⁸, plant-pollinators in ecosystems⁹, 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

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

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 network¹⁰, or the aid organization-country-issue structure¹¹.

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 analysis¹². 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

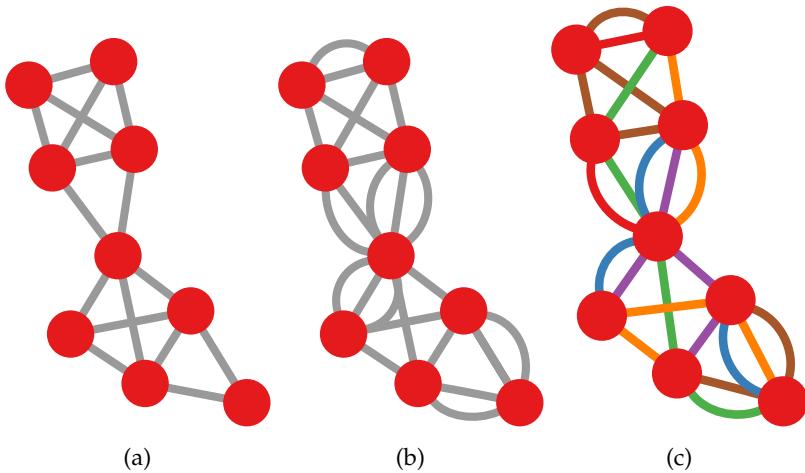


Figure 7.3: (a) A simple graph. (b) A multigraph, with multiple edges between the same node pairs. (c) A multilayer network, where each edge has a type (represented by its color).

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 complementary¹⁹. 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.

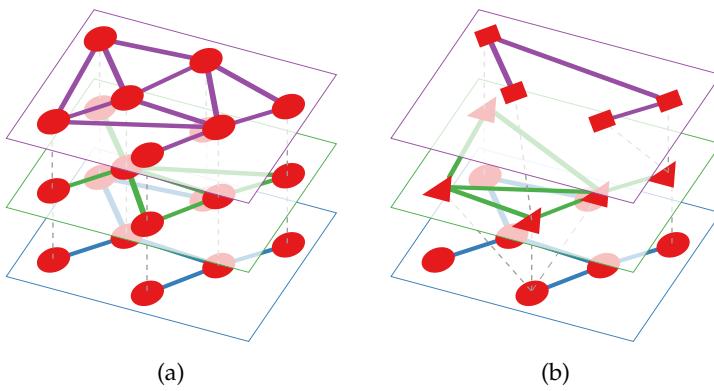


Figure 7.4: The extended multilayer model. Each slice represents a different layer of the network. Dashed grey lines represent the inter-layer coupling connections. (a) A multilayer network with trivial one-to-one coupling. (b) A multilayer network with complex interlayer coupling.

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).

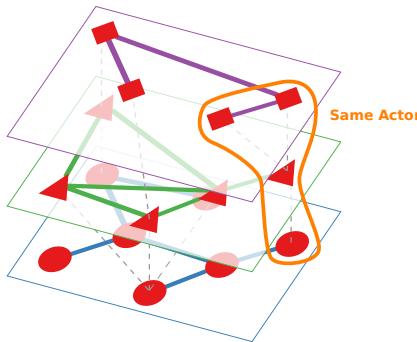


Figure 7.5: An actor in a many-to-many coupled multilayer network. The orange outline surrounds nodes with coupling edges connecting them.

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.

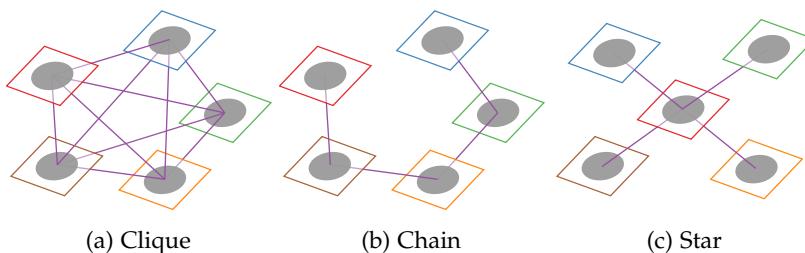


Figure 7.6: Different coupling flavors for your multilayer networks. Showing a network with a single actor and a single node per actor per layer (represented by the border-colored polygon). I color the coupling edges in purple.

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}.

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 subject²³.

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”

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

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

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.

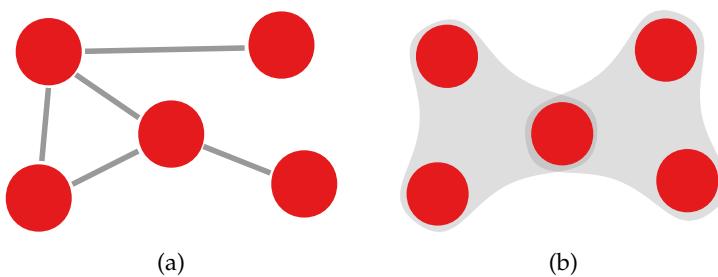
Positive and negative relationships have different dynamics. For instance, in a seminal study looking at interactions between players in a massively multiplayer online game²⁴, the authors studied the different degree distributions (Section 9.2) for each type of relationship. They uncovered that positive 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

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-

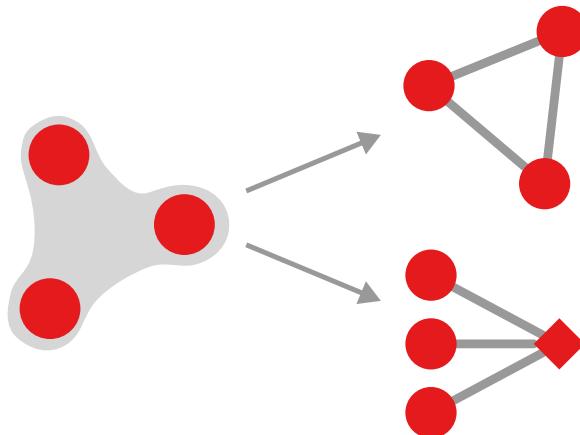
cause that's the number of players in the team. In this case, we call the resulting structure a "uniform hypergraph", and have all sorts of interesting properties²⁷. In general, when simply talking about hypergraphs we have no such constraint.

It is difficult to work with hypergraphs²⁸. 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.

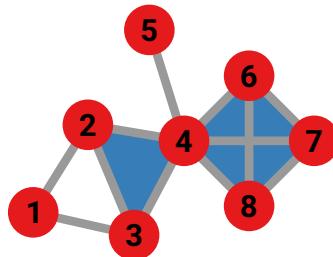
Figure 7.8: The two ways to convert a hyperedge into simpler forms. A hyperedge connecting three nodes can become a triangle (top right), or a bipartite network (bottom right).

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.

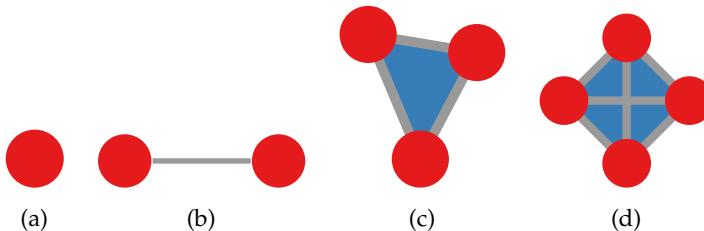


Figure 7.10: The smallest simplices a simplicial complex can have. (a) 0-simplex, (b) 1-simplex, (c) 2-simplex, (d) 3-simplex.

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 Barthélémy, 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

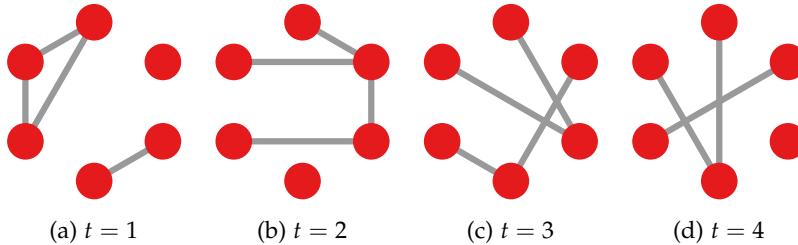


Figure 7.11: An example of dynamic network. Each figure represents the same network, observed at different points in time.

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.

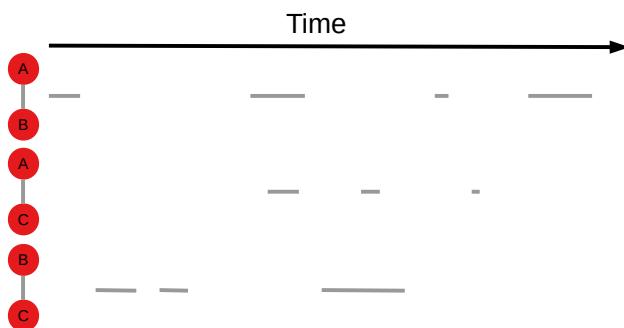
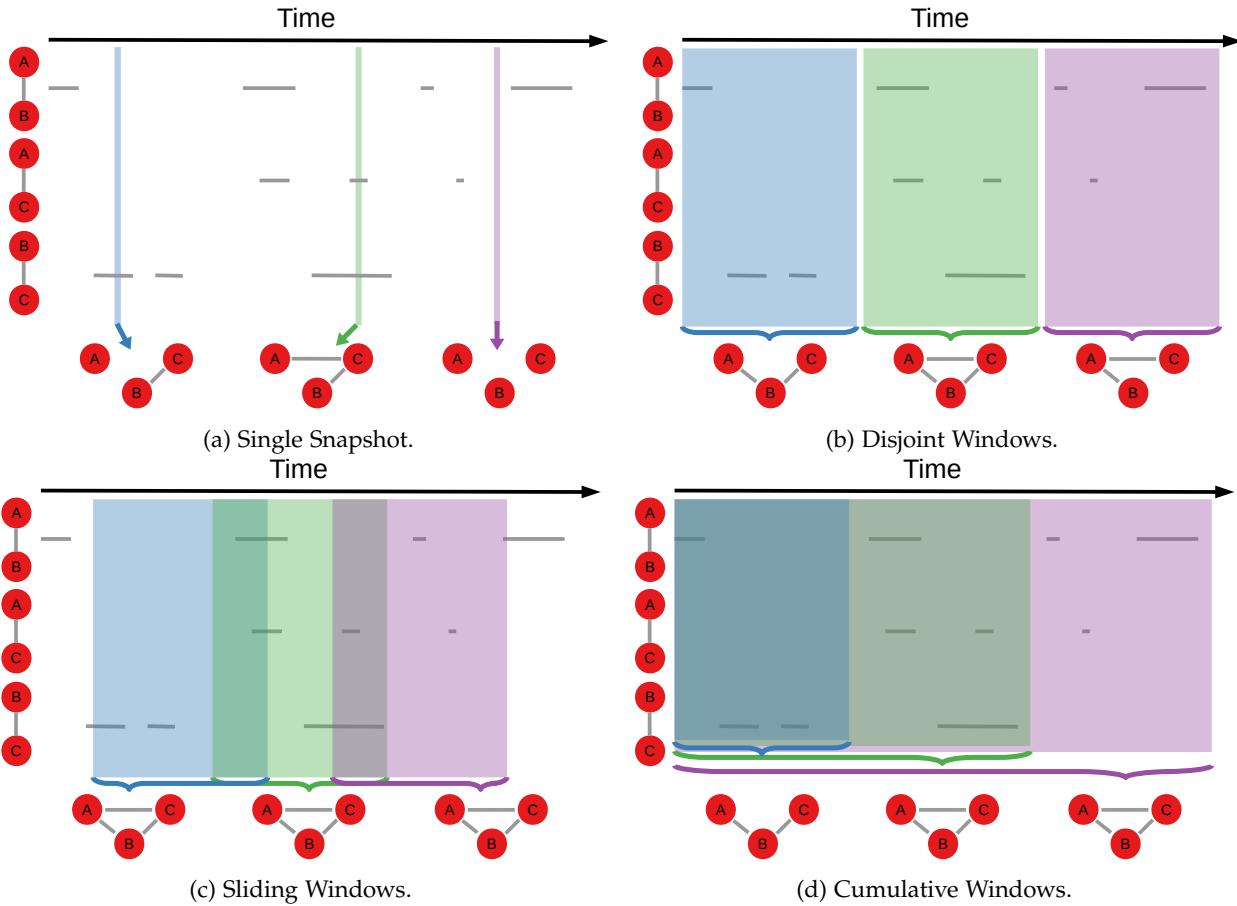


Figure 7.12: An example of dynamic edge information. Time flows from left to right. Each row represents a possible potential edge between nodes A, B, and C. The moments in time in which each edge is active are represented by gray bars.

More formally, our graph can be represented as $G = (G_1, G_2, \dots, G_n)$, 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.



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

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

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!

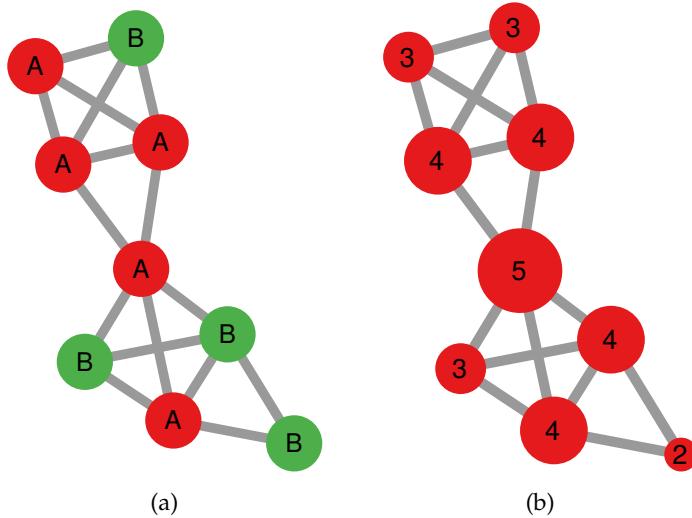


Figure 7.14: (a) A network with qualitative node attributes, represented by node labels and colors. (b) A network with quantitative node attributes, represented by node labels and sizes.

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: $v = (a_1, a_2, a_3, \dots)$. 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, \dots). 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.

8

Matrices

Graphs, with their fancy nodes and edges, are not the only way to represent a network. One can do so also by using matrices. In fact, ask some people and they will tell you that everything is a matrix. What's a number if not a zero-dimensional tensor (Section 5.3)? I mean, come on!

Unfortunately, I am not one of those people, so this chapter will contain only the bare minimum for you to smile and nod while talking to them.

The reason of having this chapter is because sometimes operations are more natural to understand with the graph models, and sometimes they are just matrix operations. Which perspective is more useful – graph vs matrix – often depends on the perspective used by the researcher(s) discovering a given property of developing a given tool. So in the book I'll often switch back and forth between these two representations, and this chapter is your map not to get lost once I start rambling about the “supra adjacency matrix”, whatever the hell that means.

In each of the sections of this chapter we will see a different way of making a matrix that has a meaningful correspondence to more and more advanced network structures. These will enable interesting operations via the linear algebra techniques I introduced in Chapter 5.

8.1 Adjacency Matrix

Basics

The adjacency matrix is a deceptively simple object. Suppose that you have a group of friends and you want to keep a tally of who's friend with whom. You can make a table with one friend per row and one friend per column. If two people say they know each other, you can just put a cross in the corresponding cell, as my example shows in

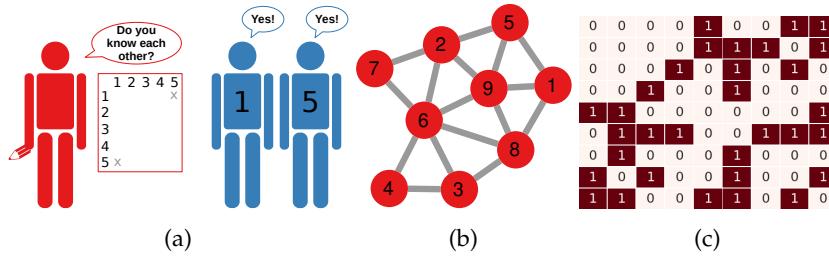


Figure 8.1(a). Well, that's it. That's an adjacency matrix.

The adjacency matrix is the basic representation of a graph as a matrix. Each row/column corresponds to a node. Each cell represents an edge, set to one if the edge exists, and zero otherwise. If the graph is undirected, each edge sets two cells to one. If the edge connects nodes u and v both the A_{uv} and the A_{vu} entries are equal to one. Figure 8.1(b) shows a graph and Figure 8.1(c) shows its adjacency matrix – in the graph view I labeled the nodes with the order as they appear in the adjacency matrix: the first row/column represents node 1, the second row/column is for node 2, and so on.

In Figures 8.1(b) and 8.1(c) we have the simplest graph possible: the unweighted undirected graph. In this case, the adjacency matrix carries a few properties. For instance, the graph has no self-loops – edges connecting a node to itself. For this reason, the diagonal of the adjacency matrix contains zeros. We like to keep it that way, because we'll use the diagonal for all sorts of interesting stuff later on – for instance later on when dealing with the graph Laplacian. The adjacency matrix is also square, meaning that it has the same number of rows and columns. Moreover, it is symmetric, meaning that $\forall u, v A_{uv} = A_{vu}$. The diagonal divides the matrix into two identical triangular halves.

You can calculate the complement of any graph by simply calculating $1 - A$, with 1 being a matrix full of ones – although you might want to fill its diagonal with zeros to avoid self loops, which are usually ignored in complement graphs.

A few interesting properties of binary adjacency matrices. The sum of the rows – and of the columns in a symmetric matrix – is equal to the node's number of connections, which we call the degree (and will be the topic of Chapter 9). If the graph is directed the rows/columns give you the the number of arrow tails and heads connected to the node. The sum of the entries of the matrix is 2 times the number of edges (undirected) or the number of edges (directed).

Figure 8.1: (a) A vignette of how one would construct an adjacency matrix. (b) An example graph. (c) The adjacency matrix of (b). Rows and columns are in the same order as the node ids (so the first row/column refers to node 1, the second to node 2, etc).

Directed, Weighted, Bipartite Adjacency

We can adapt the adjacency matrix to deal with all the complications we introduced in the graph model in Chapters 6 and 7. For instance, we can represent a directed graph by breaking the symmetry property we just enunciated. If A_{uv} is allowed to be zero when A_{vu} is one, then it means that we just introduced directionality in the matrix, as Figure 8.2 shows. Transposing A in a directed graph means to reverse all edge directions. Note that different authors/papers might follow different conventions. Some will represent the $u \rightarrow v$ edge as A_{uv} and some as A_{vu} . So make sure you identify the convention before you start working your way through the paper!

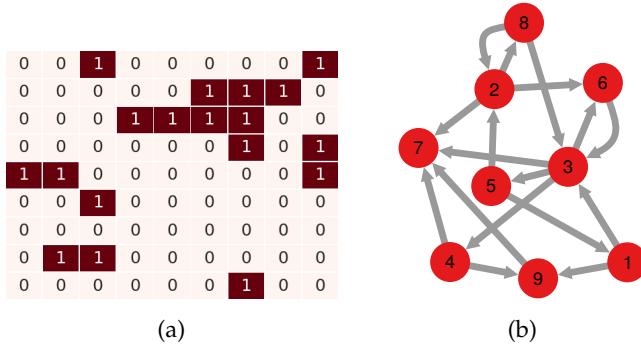


Figure 8.2: (a) A non-symmetric adjacency matrix. (b) The corresponding directed graph.

If we want edge weights to exploit the power of linear algebra also on weighted graphs (Section 6.3), we can allow values different than one for the cells representing edges. Now we can have an arbitrary real value in the cells, representing the connection's strength – see Figure 8.3.

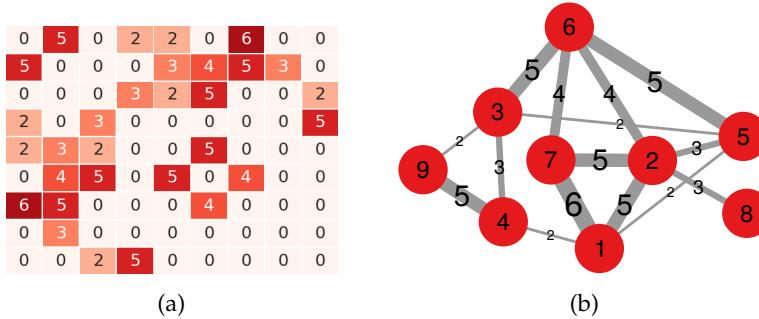
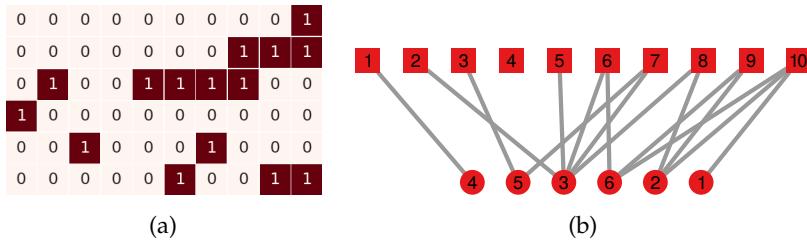


Figure 8.3: (a) A non-binary adjacency matrix. (b) The corresponding weighted graph.

What else? We can make the adjacency matrix not square if we need to represent a bipartite network. The different numbers of rows and columns allow us to use one dimension to represent the nodes in V_1 and the other to represent the nodes in V_2 . Figure 8.4 depicts an example. The downside is that we lose the power of the diagonal we had in the adjacency matrix – which doesn't seem like a big deal now, because at the moment I'm being all hush hush about what this power really is.



Of course, it's possible to have a square adjacency matrix for a bipartite network if $|V_1| = |V_2|$. You can also "squarify" a bipartite adjacency matrix by dividing it in four blocks. The blocks on the main diagonal contain zeros, while the blocks in the other diagonal contain the original adjacency matrix. Such a construct is a $(|V_1| + |V_2|) \times (|V_1| + |V_2|)$ matrix, and they can be useful. Figure 8.5 shows an example. Note that transposing the adjacency matrix of a bipartite network with V_1 and V_2 node types will change its shape, from being a $|V_1| \times |V_2|$ matrix to a $|V_2| \times |V_1|$ matrix.

$$\begin{matrix} & |V_1| & |V_2| \\ |V_1| & \left[\begin{array}{cc} 0 & A \\ A^T & 0 \end{array} \right] \\ |V_2| & \end{matrix}$$

Figure 8.4: (a) A non-square adjacency matrix. (b) The corresponding bipartite graph.

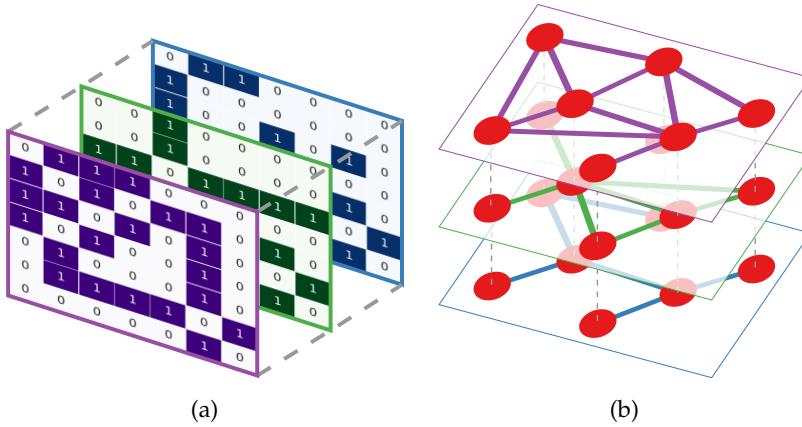
Figure 8.5: A way to build a $(|V_1| + |V_2|) \times (|V_1| + |V_2|)$ square matrix starting from A , a non-square bipartite adjacency matrix.

Multilayer Adjacency

Finally we can – and do – represent even multilayer networks with matrices. We have two options to do so. We can either use **tensors** or the **supra adjacency matrix**.

I introduced **tensors** in Section 5.3 as generalized vectors. As a refresher, a vector can be seen as a monodimensional array: a list of values. A matrix could be said to be a two-dimensional array. A tensor is a multidimensional array: we can have as many dimensions as we want.

A one-to-one coupled multilayer network can be represented with a three-dimensional vector. The first two dimensions – rows and columns – are the nodes, and the third dimension is the layers. Mathematically, the A_{uvl} entry in the tensor tells you the relationship between nodes u and v in layer l . Figure 8.6 provides an intuitive example. Note that we are assuming that the nodes are sorted in the same way across the third dimension, thus the inter-layer couplings



(see Section 7.2) are implicit.

Tensors start getting into trouble when you want to deal with many-to-many couplings. In this case, my suggestion would be to find a different job to use the **supra adjacency matrix**¹. To build this matrix you apply a strategy that is relatively similar to the squarification of a bipartite adjacency matrix. In practice, you start by taking the simple adjacency matrices for each layer independently. You put them those as blocks in the diagonal of the supra adjacency matrix.

Then you can use the rest of the matrix to represent the inter-layer couplings. This can work because each entry u in the supra adjacency matrix stands for a node in a given layer. A A_{uv} entry outside the diagonal blocks tells you if the node-in-layer represented by the u th row in the matrix is coupled with the node-in-layer represented by the v th column. Figure 8.7 shows you a many-to-many multilayer network and its corresponding supra adjacency matrix.

Figure 8.6: (a) A three dimensional tensor. (b) The corresponding multilayer graph.

¹ Mason A Porter. What is... a multilayer network. *Notices of the AMS*, 65(11):1419–1423, 2018

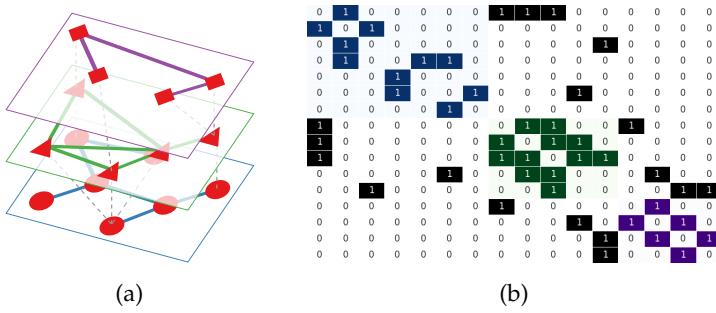


Figure 8.7: (a) A multilayer network with many-to-many interlayer couplings. Each slice represents a different layer of the network. Dashed grey lines represent the inter-layer coupling connections. (b) The supra adjacency matrix of (a).

As you can see, there is no problem if layers don't have the same number of nodes, as blocks are allowed to have different sizes and their off-diagonal parts are still able to represent the inter-layer couplings.

8.2 Stochastic

Adjacency matrices are nice, but I think most of the times you'll see them transformed in various ways to squeeze out all the possible analytic juice. The simplest makeover we can give to the adjacency matrix is to convert it into a stochastic matrix. This means that we normalize it, dividing each entry by the sum of its corresponding row – this means that each of its rows sums to one. If nodes u and v are connected, and u has 5 connections, the A_{uv} entry will be $1/5 = 0.2$. Figure 8.8 shows an example of this stochastic transformation.

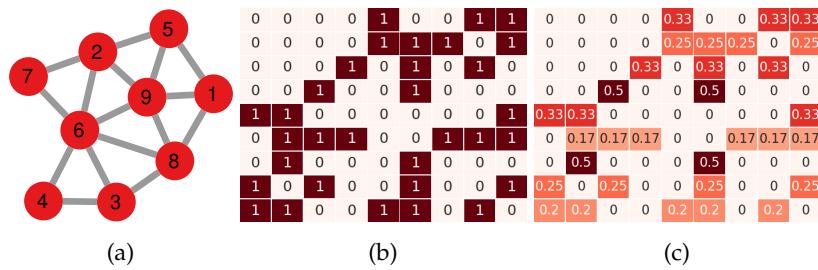


Figure 8.8: (a) The original graph. (b) The adjacency matrix of (a). (c) The corresponding stochastic version.

What's the usefulness of the stochastic adjacency matrix? The first direct use we can make of it is to calculate transition probabilities. This is literally what it contains: each entry is the probability that a random walker (see Chapter 11) on a given node (row) will cross that edge. Since non-edges have value zero, it is impossible to follow them. In Figure 8.8(a) we see that node 9 has degree equal to five. This corresponds to having five entries set to one in Figure 8.8(b). If we close our eyes and pick one of these at random, each one has a probability of 0.2 to be picked. That is the value in Figure 8.8(c).

Suppose we picked node 6 and that we repeat the exercise. Picking one of node 6's neighbors at random has a probability of 0.17, and we end up – for instance – on node 3. We might want to know what was the likelihood of ending in node 3 starting from node 9 and doing exactly two random jumps. The probability is not simply the product of the two jumps – as we would do naively for independent events (see Section 2.3) – because there is an alternative route. We could have visited node 8 first and *then* moved to 3. We have to keep track of all possible alternative paths, and this becomes really unwieldy when we start considering longer random walks.

Luckily, we don't have to do it. The stochastic matrix has the power of telling us what we want. It's literally its *power*. Say A is our stochastic matrix. We just saw how A is just the probability of transitioning from one node to another. In other words, it gives us the probability of all transitions for random walks of length 1 – the length of a walk is the number of edges we crossed or the number of steps we took (we'll talk in depth about this in Chapter 10). Let's

now write this matrix as A^1 , which is the same thing as A . Let's say this again: A^1 is the probability of all transitions for random walks of length 1. Could it be, then, that A^2 is the probability of all transitions for random walks of length 2? And that A^n is the probability of all transitions for random walks of length n ? Yes, they are!

From the matrix multiplication crash course I gave you in Section 5.2 you know why: A^2 's uv entry is, as the formula I wrote there shows, the sum of the multiplication of probabilities of all nodes k that are connected to both u and v , and thus can be used in a path of length 2. Multiplying A_{uk} to A_{kv} means asking the probability of going from u to k and from k to v . Summing $A_{uk_1}A_{k_1v}$ to $A_{uk_2}A_{k_2v}$ means asking the probability of passing through k_1 or k_2 . See Chapter 2 for a refresher on what multiplying and summing probabilities mean.

| | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|------|------|------|------|------|------|------|------|-------|-------|-------|-------|------|-----|-------|-------|------|-------|-------|-------|-------|------|-------|-------|------|-----|---|
| 0 | 0 | 0 | 0 | 0.33 | 0 | 0 | 0.33 | 0.33 | 0.3 | 0.2 | 0.08 | 0 | 0.07 | 0.1 | 0 | 0.07 | 0.2 | 0.080 | 0.090 | 0.040 | 0.05 | 0.2 | 0.1 | 0.07 | 0.2 | 0.2 | |
| 0 | 0 | 0 | 0 | 0.25 | 0.25 | 0.25 | 0 | 0.25 | 0.1 | 0.3 | 0.040 | 0.040 | 0.05 | 0.2 | 0.040 | 0.09 | 0.1 | 0.060 | 0.090 | 0.070 | 0.04 | 0.1 | 0.2 | 0.1 | 0.1 | 0.2 | |
| 0 | 0 | 0 | 0.33 | 0 | 0.33 | 0 | 0.33 | 0 | 0.080 | 0.06 | 0.3 | 0.08 | 0 | 0.2 | 0.060 | 0.06 | 0.1 | 0.04 | 0.1 | 0.08 | 0.1 | 0.07 | 0.2 | 0.06 | 0.2 | 0.1 | |
| 0 | 0 | 0.5 | 0 | 0 | 0.5 | 0 | 0 | 0 | 0 | 0.080 | 0.08 | 0.2 | 0 | 0.2 | 0.08 | 0.2 | 0.08 | 0.080 | 0.09 | 0.2 | 0.060 | 0.04 | 0.3 | 0.050 | 0.07 | 0.1 | |
| 0.33 | 0.33 | 0 | 0 | 0 | 0 | 0 | 0 | 0.33 | 0.070 | 0.07 | 0 | 0 | 0.3 | 0.1 | 0.08 | 0.2 | 0.2 | 0.2 | 0.070 | 0.020 | 0.08 | 0.1 | 0.040 | 0.09 | 0.2 | 0 | 0 |
| 0 | 0.17 | 0.17 | 0.17 | 0 | 0 | 0.17 | 0.17 | 0.17 | 0.07 | 0.1 | 0.1 | 0.060 | 0.07 | 0.3 | 0.040 | 0.090 | 0.08 | 0.06 | 0.1 | 0.1 | 0.07 | 0.1 | 0.2 | 0.09 | 0.1 | 0.2 | 0 |
| 0 | 0.5 | 0 | 0 | 0 | 0.5 | 0 | 0 | 0 | 0 | 0.080 | 0.080 | 0.08 | 0.1 | 0.1 | 0.2 | 0.08 | 0.2 | 0.1 | 0.2 | 0.080 | 0.050 | 0.06 | 0.3 | 0.040 | 0.09 | 0.1 | |
| 0.25 | 0 | 0.25 | 0 | 0 | 0.25 | 0 | 0 | 0.25 | 0.050 | 0.090 | 0.04 | 0.1 | 0.1 | 0.1 | 0.04 | 0.3 | 0.1 | 0.1 | 0.1 | 0.1 | 0.040 | 0.06 | 0.2 | 0.050 | 0.08 | 0.2 | 0 |
| 0.2 | 0.2 | 0 | 0 | 0.2 | 0.2 | 0 | 0.2 | 0 | 0.1 | 0.1 | 0.080 | 0.03 | 0.1 | 0.1 | 0.08 | 0.1 | 0.3 | 0.1 | 0.2 | 0.060 | 0.04 | 0.1 | 0.2 | 0.04 | 0.1 | 0.1 | |

(a) A^1 (b) A^2 (c) A^3

Let's take a closer look at A^2 and A^3 in Figures 8.9(b) and 8.9(c). First, they are stochastic matrices, and Section 2.6 taught you that the rows of a stochastic matrix always sum to 1. So each entry in the matrix is still a transition probability. This time, though, it's not the transition probability of the direct connection, but of a path of length 2 and 3, respectively.

Second, the diagonal is not zero any more. That is because, with a random walk of length 2, there is a chance to select the same edge twice, and therefore returning to the point of origin. So the A_{vv}^2 entry tells you the likelihood of starting from v and returning back to v in two steps. That is because – as the matrix multiplication section showed you mathematically – A_{vv}^2 is the combination of the probabilities of going from any of v 's neighbors to v , weighted by the probability of having reached each of v 's neighbors from v itself.

Third, while A^2 still has zero entries, A^3 does not. This is because some node pairs are farther than two edges away, so the probability of a random walker to reach them in two hops is zero – check Figure 8.8(a) again if you don't believe me! On the other hand, no pair of nodes is farther than three hops away, and thus there is always a path of length three between any node pair, no matter how unlikely.

Finally, stochastic matrices – either A or any of its powers – are not symmetric any more, even if the “raw” adjacency matrix of an

Figure 8.9: Different powers of the stochastic adjacency matrix of the graph in Figure 8.8(b).

undirected graph is. This is because the likelihood of ending in u from v isn't necessarily the same as the other way around: if v has better connected neighbors, the random walkers are more likely to be led astray. For instance, the probability to go from node 4 to 5 in three random steps is 0.04, while the other way around is 0.02. That is because node 5 has an extra connection, which can lead the walker to be unable to reach 4 in two additional hops.

This last property means that, if you transpose a stochastic adjacency matrix, $A^T \neq A$ even for undirected graphs! Since you normalized by row sum, the A_{uv} entry can be different from the A_{vu} : the only thing you know is that they're both non-zero.

There is one surprise hidden in the folds of A^n for a suitably large n . This surprise is waiting for you in Section 11.1.

Note that there are two valid stochastic adjacency matrices for a bipartite network. If you normalize by row sum, the stochastic A tells you the probability of going from a V_1 node to a V_2 node. Normalizing by column sum, which is equivalent of taking the stochastic A^T (i.e., transposing A beforehand), then the matrix tells you the probability of going from a V_2 node to a V_1 node. In fact, bipartite matrices allow us to make use of another linear algebra operation I talked about: the transpose.

Suppose you have a bipartite network and what you really want to know is not which node of type V_1 connects to nodes in type V_2 , but how similar nodes in V_1 are, because they connect to the same V_2 nodes. This is practically the subject of Chapter 26 but, to make it simple for this example, you want the probability of going from a V_1 node to another V_1 node, passing via V_2 nodes.

Dividing the bipartite adjacency matrix by its row sum gets you the probability to go from a V_1 node to a V_2 node – as we just said. However, we don't know how to go back: in that case we should normalize by column sum! That could be achieved by normalizing A^T , A 's transpose, by its row sum. Since A is a $|V_1| \times |V_2|$ matrix and A^T is a $|V_2| \times |V_1|$ matrix, you can multiply one by the other: AA^T is in fact a $|V_1| \times |V_1|$ which is exactly what you need.

But wait, what does AA^T actually mean? What's the result of such an operation? Well, following Figure 5.10, the (v_1, v_2) cell is the sum of the probability of going from v_1 to any V_2 node times the probability of arriving to v_2 from any V_2 node. Which is what we wanted!

Without linear algebra, you'd have to represent the adjacency by sets of neighbors, and then calculate intersections and dividing various scalars in isolation. But transposes and matrix multiplications are such standard operations that many libraries will have implemented them very efficiently, with the result of being blazingly fast to calcu-

late and extremely easy to incorporate in your code. Moreover, if you use sparse matrix representations you can also be memory efficient, meaning you can go big with your matrices!

Coming back to eigenvalues and eigenvectors (Section 5.5), the largest eigenvalue of a stochastic adjacency matrix is always equal to one – we also call it the “leading” eigenvalue, or λ_1 . This takes a special value: any stochastic adjacency matrix you can come up with will always have $\lambda_1 = 1$. No eigenvalue will ever be greater.

As we saw in Figure 5.13, each eigenvalue has a corresponding eigenvector. We call the eigenvector associated to the largest eigenvalue the “largest” or “leading” eigenvector, for convenience. The point of looking at eigenvectors is that there is a relationship between the v -th entry in the i -th eigenvector of an adjacency matrix and node v 's relationship with the entire graph.

For instance, the multiplicity of the largest eigenvalue of the stochastic adjacency matrix is important. It can happen that second largest eigenvalue λ_2 of A could be equal to the first. And, actually, also the third, fourth, fifth, ... could be equal to the first. This is related to the first application of linear algebra to network analysis, which we will fully appreciate when it will be time to discuss about connected components (Section 10.4).

8.3 Incidence

In general, an incidence matrix is a matrix telling you what are the relations between two classes of objects. For instance, you can have an incidence matrix telling you for which company a person works. Since the two classes might have a different number of members, incidence matrices are not necessarily square. In fact, you could say that the adjacency matrix of a bipartite network is an incidence matrix.

However, when performing network analysis, there is one type of incidence matrix that is widely used, and thus “owns” this term. The vast majority of times, if you read a paper talking about the “incidence matrix”, you’ll see the same object: a matrix that has nodes on the rows, edges on the columns, and it has an entry equal to one if the node and the edge are connected to each other. Figure 8.10(b) shows the incidence matrix of the graph at Figure 8.10(a).

Incidence matrices have interesting properties, some more trivial than others. For instance, you know that, in the incidence matrix of a simple graph, each column sums to two because each edge only connects two nodes. Only in the incidence matrix of a hypergraph a column can sum to a number larger than 2. You can use the incidence matrix to construct other special matrix representations. For instance,

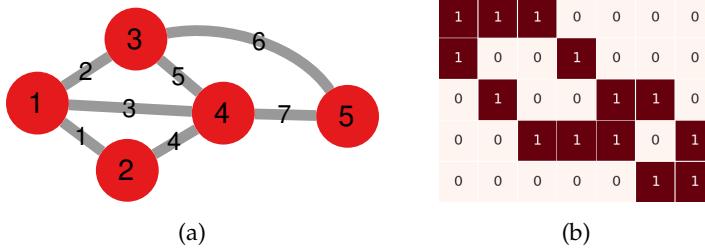


Figure 8.10: (a) A graph with nodes and edges labeled with their ids. (b) The incidence matrix of (a), with nodes on the rows and edges on the columns.

you can construct the adjacency matrix of the line graph of G by calculating $B^T B - 2I$, assuming that B is the incidence matrix, and I is a $|E| \times |E|$ identity matrix.

An incidence matrix can also be oriented. In an oriented incidence matrix, the columns sum to zero. For every edge, one of the two non zero entries – the nodes to which it is attached – is equal to 1 and the other is equal to -1. It doesn't really matter which of the two you pick, as long as you make sure all columns sum to zero. If B is an oriented incidence matrix, you can use it to construct the Laplacian as BB^T . The Laplacian is a super cool matrix and I'll focus on it now.

8.4 Laplacian

Basic Laplacian

The stochastic adjacency matrix is nice, but the real superstar when it comes to matrix representations of networks is the Laplacian. To know what that is, we need to introduce the concept of Degree matrix D – which is a very simple animal. It is what we call a “diagonal” matrix. A diagonal matrix is a matrix whose nonzero values are exclusively on the main diagonal. The other off-diagonal entries in the matrix are equal to zero. In D the diagonal entries are the degrees of the corresponding nodes. Figure 8.11 shows an example of a degree matrix.

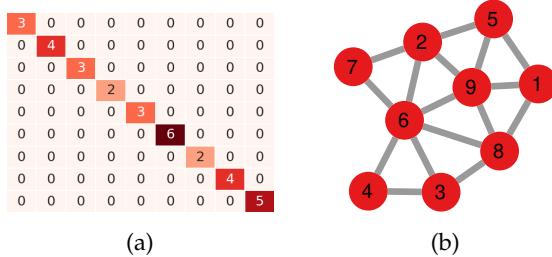
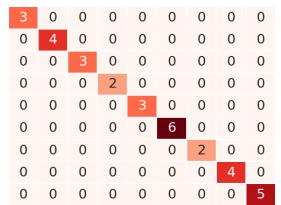
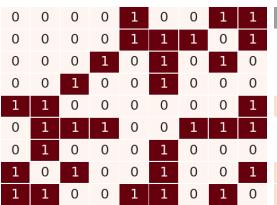
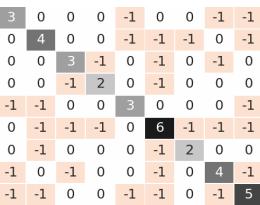


Figure 8.11: The degree matrix (a) of the sample graph (b).

The Laplacian version of the adjacency matrix – which we call L – is the result of a simple operation: $L = D - A$. In practice, we take the degree matrix D and we subtract A from it. L is a matrix that has the node degree in the diagonal, -1 for each entry corresponding to

an edge in the network, and zero everywhere else. Figure 8.12 depicts the operation. Why is this matrix interesting? It was originally developed to represent something very physical: it captures the relation between voltages and currents between resistors² – represented by the edges of the graph.

However, you don't need to care about electric circuits to find a use for the Laplacian. L has a number properties that are useful in general, regardless of what your graph represents. Some of the most obvious ones are that, since it has the degree of the node on the diagonal and -1 for each of the node's connection, the sums of all rows and columns are equal to zero.

| | | |
|---|---|--|
|  |  |  |
| (a) D | (b) A | (c) L |

The Laplacian of a connected undirected graph is part of the positive semi-definite club (Section 5.4), which will come in handy in Section 47.2.

Just like for A , we are also interested in the eigenvectors of L . We need to make some adjustments, though. Instead of looking at the largest eigenvalues, we focus on the smallest ones – meaning that now we use λ_1 to refer to the *smallest* eigenvalue. This takes a special value like for A but, for L , $\lambda_1 = 0$. Besides doing some of the same things you can do with the eigenvector of the adjacency matrix, the Laplacian has a few more tricks up its sleeve. For instance, one can use it to solve the normalized cut problem, which is useful for community discovery and we will discuss it in detail in Section 11.5.

Another connection between stochastic and Laplacian matrices is on multiplicity. The multiplicity of the smallest eigenvalue of the Laplacian plays the exact same role as the one of the largest eigenvalue of the stochastic matrix – a role that you will appreciate in Section 10.4 when we will study connected components.

We will see what makes the Laplacian so important in Chapter 11, when I'll show you how many things you can do with its quasi-mystical properties.

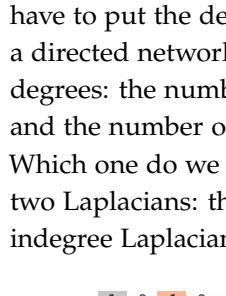
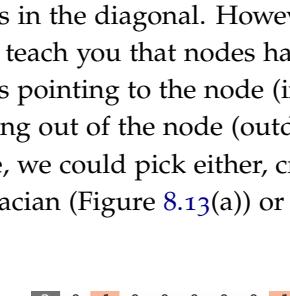
Special Laplacians

The way the Laplacian is built makes it non trivial how to generalize it for different types of networks. Here I focus on two cases: **directed** and **signed** graphs.

² Gustav Kirchhoff. Ueber die auflösung der gleichungen, auf welche man bei der untersuchung der linearen vertheilung galvanischer ströme geführt wird. *Annalen der Physik*, 148(12):497–508, 1847

Figure 8.12: The operation producing the Laplacian matrix L (c), subtracting the adjacency matrix A (b) from the degree matrix D (a).

For **directed** networks the problem comes from the fact that we have to put the degrees of the nodes in the diagonal. However, in a directed networks, Chapter 9 will teach you that nodes have two degrees: the number of arrow heads pointing to the node (indegree) and the number of arrow tails coming out of the node (outdegree). Which one do we pick? In principle, we could pick either, creating two Laplacians: the outdegree Laplacian (Figure 8.13(a)) or the indegree Laplacian (Figure 8.13(b)).

| | |
|---|---|
|  |  |
| (a) Indegree Laplacian | (b) Outdegree Laplacian |

However, this deceptively simple operation will make you lose some nice properties the Laplacian has. For instance, the Laplacian normally has real-valued eigenvalues. Unfortunately, as soon as you add a single directed link to the graph, the eigenvalues become complex, i.e. they contain a $i = \sqrt{-1}$ portion. Oops.

Signed networks are a special case of multilayer networks I introduced in Section 7.2. Also in this case you need to take some care, because you can't simply sum the adjacency matrix to make the degree matrix D . You need to sum the absolute value of A to make D . If you do so, you get the signed Laplacian³ – which I show in Figure 8.14(b). You can also make an unsigned Laplacian – Figure 8.14(c) – by ignoring this issue and summing positive and negative values alike⁴. However, that means you might end up with a zero entry on the diagonal.

The eigenvectors of the signed Laplacian are as cool as the ones of the regular one, and you'll briefly encounter them in Chapter 24.

8.5 Summary

1. You can represent a graph with an adjacency matrix. The matrix has a row/column per node, and cells are equal to one if the two nodes are connected, zero otherwise.

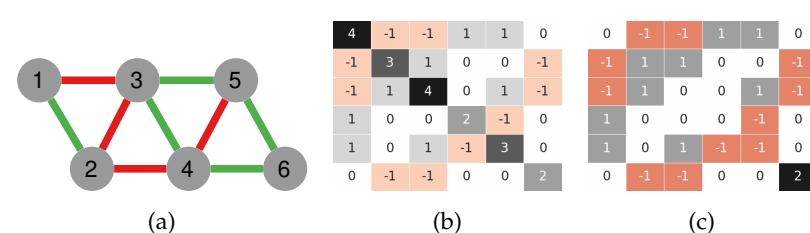


Figure 8.13: The two Laplacians for the directed graph in Figure 8.2.

³ Yu Tian and Renaud Lambiotte. Spreading and structural balance on signed networks. *SIAM Journal on Applied Dynamical Systems*, 23(1):50–80, 2024

⁴ Guodong Shi, Claudio Altafini, and John S Baras. Dynamics over signed networks. *SIAM Review*, 61(2):229–257, 2019

Figure 8.14: (a) A signed graph, with positive edges in green and negative edges in red, and its two Laplacians. (b) The signed Laplacian of (a). (c) The unsigned Laplacian of (a)

2. Special graph types will have special adjacency matrices. A directed graph has an asymmetric adjacency, a bipartite graph has a non-square one, and multilayer graphs have tensors and supra adjacencies.
3. The stochastic adjacency matrix is a row-normalized (or column-normalized depending on the convention) adjacency matrix, whose rows (or columns) sum to one. It describes transition probabilities from one node to another.
4. The incidence matrix is a $|V| \times |E|$ matrix telling you if a given node is connected to a given edge.
5. The degree matrix D is a matrix having the degree of the node in the main diagonal and zero everywhere else. If you subtract the adjacency matrix from the degree matrix you obtain the graph Laplacian, which is widely used in many network applications.
6. For directed networks, there are two Laplacians depending on whether you use the in- or out-degree for the diagonal. For signed networks, you can have two alternative Laplacians: signed and unsigned.

8.6 Exercises

1. Calculate the adjacency matrix, the stochastic adjacency matrix, and the graph Laplacian for the network in <http://www.networkatlas.eu/exercises/8/1/data.txt>.
2. Given the bipartite network in <http://www.networkatlas.eu/exercises/8/2/data.txt>, calculate the stochastic adjacency matrix of its projection. Project along the axis of size 248. (Note: don't ignore the weights)
3. Calculate the eigenvalues and the right and left eigenvectors of the stochastic adjacency of the network at <http://www.networkatlas.eu/exercises/8/2/data.txt>, using the same procedure applied in the previous exercise. Make sure to sort the eigenvalues in descending order (and sort the eigenvectors accordingly). Only take the real part of eigenvalues and eigenvectors, ignoring the imaginary part.
4. Generate the indegree and outdegree Laplacians of the directed graph at <http://www.networkatlas.eu/exercises/8/4/data.txt>. Calculate their eigenvalues as well as the eigenvalue of the undirected version of the graph.

5. Generate the signed and unsigned Laplacians of the signed graph at <http://www.networkatlas.eu/exercises/8/5/data.txt> – the third column contains the sign. Calculate their eigenvalues as well as the eigenvalue of the version of the graph ignoring edge signs.

Part III

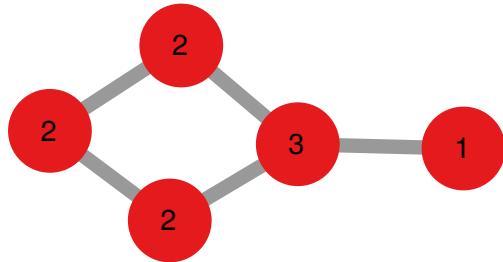
Simple Properties

9

Degree

So far we just described graph representations. We haven't actually done anything with them. Here, we start probing their properties, to say things about their nodes, edges, and structures. In this chapter we deal with the simplest possible statistics of a node: its degree. This is such a basic concept that I actually already mentioned it multiple times without defining it. It's hard not to, when talking about networks.

The intuition behind the degree is easy to understand in a social network. What is the simplest way for you to know how well you're doing in a society? Well, you could look around you, see the friends you have, and count them. This is the degree. I've got my decent couple of hundreds Facebook friends, like almost everyone else. But some people are superstars, and count the number of their acquaintances in the thousands. How many people does Brad Pitt know? Probably a couple orders of magnitude more than me. Those are the kinds of differences you can quantify by calculating the degree of all nodes in your network.



The degree of a node is simply the number of edges incident to it¹, as Figure 9.1 shows. Or the number of its connections. Or – given some assumption that I'll break later on – the number of its neighbors. It is the first, most fundamental measure of structural importance of a node. A node of high degree ought to be an important node in the network. If it were to disappear, many nodes would lose

Figure 9.1: A network where I labeled each node with its degree.

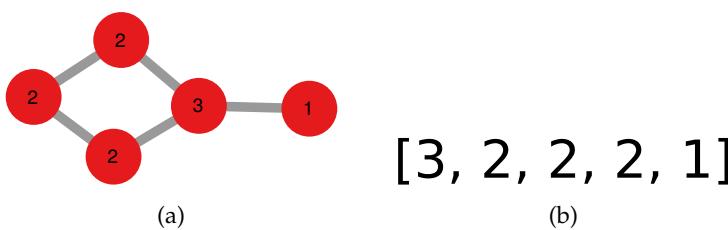
¹ Reinhard Diestel. *Graph theory*. Springer Publishing Company, Incorporated, 2018

a connection.

The degree is a property of a node. Let's call k_v the degree of node v . We can aggregate the degrees of all nodes in a network to get a "global" information about its connectivity. The most common way to do it is by calculating the average degree of a network. This would be $\bar{k} = \sum_{v \in V} k_v / |V|$, however it's much simpler to remember that $\bar{k} = 2|E| / |V|$. The average degree of a network is twice the number of edges divided by the number of nodes. Why twice? Because each edge increases by one the degree of the two nodes it connects.

In a social network, this is how many friends people have on average. What would that number be in your opinion? If we have a social network including two billion people, what's the average degree? It turns out that this number is usually ridiculously lower than one would expect, because – as we'll see in Section 12.1 – real networks are sparse².

We call a node with zero degree, a person without friends, an isolated node, or a singleton. A node with degree one is a "leaf" node: this term comes from hierarchies, where nodes at the bottom – the leaves of the tree – can only have one incoming connection without outgoing ones. The sum of all degrees is $2|E|$, which implies that any graph can only have an even number of nodes with odd degree³ – otherwise the sum of degrees would be odd and thus it cannot be two times something.



A degree sequence is the list of degrees of all nodes in the network^{4,5}. Typically, we sort the nodes in descending degree, so you always start with the node with maximum degree and you go down until you reach the node with the lowest degree. Figure 9.2 shows an example.

Note that not all lists of integers are valid degree sequences. Some lists cannot generate a valid graph. The easiest case to grasp is if they contain an odd number of odd numbers. As we just saw, the degree sequence must sum to an even number ($2|E|$), thus a sequence summing to an odd number cannot describe a simple undirected graph⁶. We call all valid sequences "graphic". We'll see that there are other, more subtle, requirements for a graphic sequence.

² Anna D Broido and Aaron Clauset. Scale-free networks are rare. *Nature communications*, 10(1):1017, 2019

³ Leonhard Euler. Solutio problematis ad geometriam situs pertinentis. *Commentarii academiae scientiarum Petropolitanae*, pages 128–140, 1741

Figure 9.2: A graph and its degree sequence.

⁴ Michael Molloy and Bruce Reed. The size of the giant component of a random graph with a given degree sequence. *Combinatorics, probability and computing*, 7(3):295–305, 1998

⁵ Béla Bollobás, Oliver Riordan, Joel Spencer, and Gábor Tusnády. The degree sequence of a scale-free random graph process. *Random Structures & Algorithms*, 18(3):279–290, 2001

⁶ Gerard Sierksma and Han Hoogeveen. Seven criteria for integer sequences being graphic. *Journal of Graph theory*, 15(2):223–231, 1991

9.1 Degree Variants

Of course, the degree definition I just gave only makes sense in the world of undirected, unweighted, unipartite, monolayer networks. We had two whole chapters detailing when such a simple model doesn't work in complex real scenarios. We need to extend the definition of degree to take into account all different graph models we might have to deal with.

Directed

As we saw in Section 6.2, edges can have a direction, meaning that the edge going from u to v doesn't necessarily point back from v to u . Such is life. In directed graphs you can keep counting the degree as simply the number of connections of a node, but there is a more helpful way to think about it. You might want to distinguish the people who send a lot of connections – but don't necessarily see them reciprocated –, and those who are the target of a lot of friends requests – whether they accept them or not.

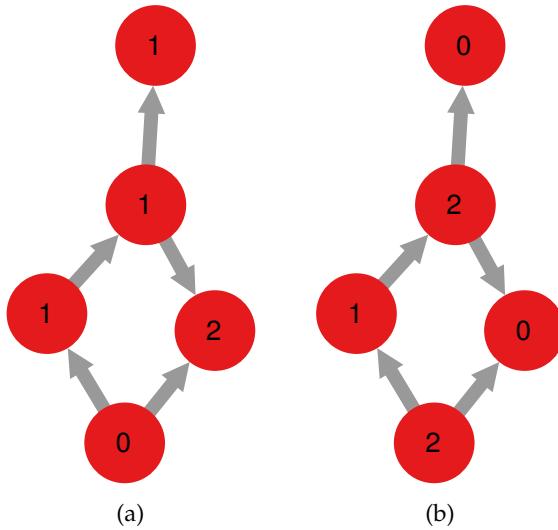


Figure 9.3: (a) A network where I labeled each node with its in-degree. (b) A network where I labeled each node with its out-degree.

So we split the concept in two parts, helpfully named in-degree and out-degree^{7,8}. As one can expect, the in-degree is the number of incoming connections. If we represent a directed edge as an arrow, the in-degree is the number of arrow heads attached to your node. See Figure 9.3(a) for a helpful representation. The out-degree is the number of outgoing connections, the number of arrow tails attached to your node. I show the out-degree of the nodes in my example in Figure 9.3(b).

A directed graph's degree sequence is now a list of tuples. The first element of the tuple tells you the indegree, while the second

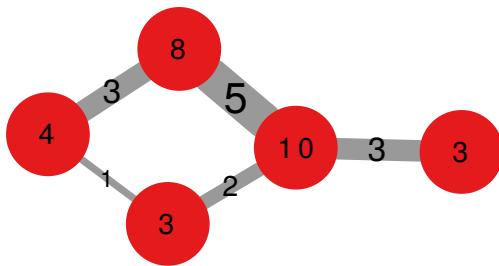
⁷ Frank Harary, Robert Zane Norman, and Dorwin Cartwright. *Structural models: An introduction to the theory of directed graphs*. Wiley, 1965

⁸ Jørgen Bang-Jensen and Gregory Z Gutin. *Digraphs: theory, algorithms and applications*. Springer Science & Business Media, 2008

element tells you the outdegree. Or you can have two sequences, but you need to make sure that the n th positions of the two sequences refer to the same node. If the two sequences are the same, meaning that every node has the same in- and out-degree, we have a “balanced” graph.

Weighted

Most of the time, people do not change the definition of degree when dealing with weighted networks. Many network scientists like how the standard definition works in weighted graphs, and keep it that way. The degree is simply the number of connections a node has.



Other people don't⁹. In the case of weighted networks, one might be interested in the total weight incident into a node. We would call such quantity the “weighted degree” or “node strength”¹⁰. Node strengths are key concepts in investigating propagating failures on networks (Section 22.3) and in some network backboning techniques (Section 27.5).

Node strengths work exactly how you would expect them to do: to get v 's weighted degree you sum the weights of all the edges incident to v . Figure 9.4 shows an example. The advantage of this definition is that it reduces to the classical degree definition if your network is unweighted – that is to say that all of G 's edge weights are equal to one.

By separating the unweighted count of connections (degree) from the weighted sum of connections (weighted degree), we capture two distinct notions of connectivity. One can have a node with enormous strength but low degree – a core router on the internet with few high-bandwidth connections – and a “peripheral” router on your street – which has a large number of low-bandwidth connections.

The reasons to do so are many. For instance, if you're looking a road graph, each edge represents a trait of road. It might be weighted with the number of cars passing through it per unit of time. Nodes, in this case, are road intersections. A weighted degree will tell you how many cars per unit of time want to clear that particular intersec-

Figure 9.4: A weighted network where I labeled each edge with its weight and each node with its weighted degree.

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

¹⁰ Tore Opsahl, Filip Agneessens, and John Skvoretz. Node centrality in weighted networks: Generalizing degree and shortest paths. *Social networks*, 32(3):245–251, 2010

tion. If the number is too high, you might be in trouble!

Bipartite

The bipartite case doesn't need too much treatment: the degree is still the number of connections of a node. It doesn't matter much that for V_1 nodes it is gained exclusively via connections to V_2 nodes and viceversa. However, there's a little change when one uses a matrix representation that it's worthwhile to point out. Assuming A as a binary adjacency matrix (not stochastic), in the regular case the degree is the sum of the rows: the sum of first row tells you the degree of the first node, and so on.

| | |
|---------------------|---------------------|
| 0 0 1 0 0 0 0 0 1 | 0 0 0 0 0 1 1 1 0 |
| 0 0 0 0 0 1 1 1 0 | 0 0 0 0 0 1 1 1 0 |
| 0 0 0 1 1 1 1 0 0 | 1 0 0 0 0 0 1 0 1 0 |
| 0 0 0 0 0 0 0 1 0 1 | 0 0 1 0 0 0 0 0 0 0 |
| 1 1 0 0 0 0 0 0 1 | 0 1 1 1 0 0 0 0 0 0 |
| 0 0 1 0 0 0 0 0 0 | 0 1 1 1 0 0 0 0 0 1 |
| 0 0 0 0 0 0 0 0 0 | 0 1 0 0 0 0 0 0 0 0 |
| 0 1 1 0 0 0 0 0 0 | 1 0 0 1 1 0 0 0 0 0 |
| 0 0 0 0 0 0 0 1 0 0 | |

(a) A

| | |
|---------------------|---------------------|
| 0 0 0 0 1 0 0 0 0 | 1 0 0 0 1 0 0 1 0 |
| 0 0 0 0 1 0 0 1 0 | 0 0 1 0 0 0 0 0 0 |
| 1 0 0 0 0 0 1 0 1 0 | 0 0 1 0 0 0 0 0 0 |
| 0 0 1 0 0 0 0 0 0 | 0 1 1 1 0 0 0 0 0 |
| 0 0 0 1 0 0 0 0 0 | 0 1 1 1 0 0 0 0 0 1 |
| 0 1 1 0 0 0 0 0 0 | 0 1 0 0 0 0 0 0 0 0 |
| 1 0 0 1 1 0 0 0 0 0 | |

(b) A^T

In a bipartite network that will only tell you the degree of the V_1 nodes. You won't know anything about the V_2 nodes if you only look at row sums. You can fix the problem in two, equivalent, ways. You can either looking at the column sums, or you can look at the row sums of A^T , the transpose of A . A^T 's rows are A 's columns and vice versa, so the equivalence between these two approaches should be self-evident – if it isn't, try to play with Figure 9.5.

Just like directed graphs, also bipartite graphs have two degree sequences, one for V_1 nodes and the other for V_2 nodes. They both sum to the same value: $|E|$, implying that, in this case, you can have an odd number of odd degree nodes in each node type¹¹.

Multigraph

When I introduced the degree I said that it can be the number of a node's connections or the number of its neighbors. These two were assumed to be interchangeable, because each edge in a simple graph will bring you to a distinct neighbor. Say that k_u is u 's degree, and N_u the set of its neighbors. In a simple graph, $k_u = |N_u|$ – assuming there are no self-loops or, if there are, that N_u can contain u itself.

That is not the case in a multigraph. Since we allow parallel edges, you can follow two distinct connections and end up in the same neighbor. So we need to solve this ambiguity. The way I saw most commonly accepted is to keep the degree (k_u) as the number of

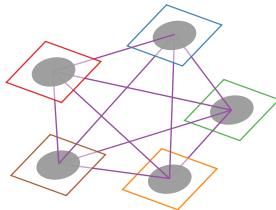
Figure 9.5: Calculating the degree of a bipartite network via its adjacency matrix A and its transpose A^T . The first V_1 node has degree equal to two (the sum of the first row is two). The first V_2 node has degree equal to one, which you can calculate either by summing the first column of A , or by summing the first row of A^T .

¹¹ Armen S Asratian, Tristan MJ Denley, and Roland Häggkvist. *Bipartite graphs and their applications*, volume 131. Cambridge University Press, 1998

connections of a node. The number of neighbors of a node ($|N_u|$) will be just that: the number of neighbors. So, in a multigraph $k_u \neq |N_u|$ or, to be more precise, $k_u \geq |N_u|$.

Multilayer

The multilayer case is possibly the most complex of them all. At first, it doesn't look too bad. The degree is still the number of connections a node has. Then you realize that there are some connections you shouldn't count. For instance, no one – that I know of – counts the interlayer coupling connections as part of the degree. It's easy to see why: these are not connections that lead you to a neighbor in a proper sense. They lead you to... a different version of yourself.



Even if we want to ignore this quirk, counting these connections won't really give you meaningful information. If you have a one-to-one multilayer network in which all nodes are part of all layers, they are all going to have the same number of inter-layer couplings. Sometimes, this number can be quite high. If you have five layers and you connect all identities of the same actor across layers, you effectively have a clique (see Section 12.3) of inter-layer couplings. If you count those as part of the degree, this actor would have a degree starting from ten – as I show in Figure 9.6 –, which would be unreasonable. You could have fewer inter-layer coupling using different coupling strategies, but that wouldn't change the substance.

Since each layer is a network on its own, it is natural to want to have a measure telling us the degree of a node in a particular layer. So an actor can have many degrees: one per layer, and a general one, which we can define as the sum of each layer's degree. However, things can get complicated with a many-to-many mapping. In that case, the actor can "own" more than one node in a layer. Each node has its own degree, but how much do they contribute to the actor's degree? The answer might vary, depending on what you're interested in calculating.

One can also combine layers to do all sorts of interesting stuff. I'm going to give you some examples from a paper of mine¹², with the caveat that the space of actual possibilities is much vaster than this^{13,14}. What follows is also very related to the multilayer concept

Figure 9.6: Should we really say that the degree of this isolated node is ten just because there are five layers in the network and we couple them with each other? Eight out of ten cats say "no".

¹² Michele Berlingerio, Michele Coscia, Fosca Giannotti, Anna Monreale, and Dino Pedreschi. Multidimensional networks: foundations of structural analysis. *WWW*, 16(5-6):567–593, 2013a

¹³ 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

¹⁴ Federico Battiston, Vincenzo Nicosia, and Vito Latora. Structural measures for multiplex networks. *Physical Review E*, 89(3):032804, 2014

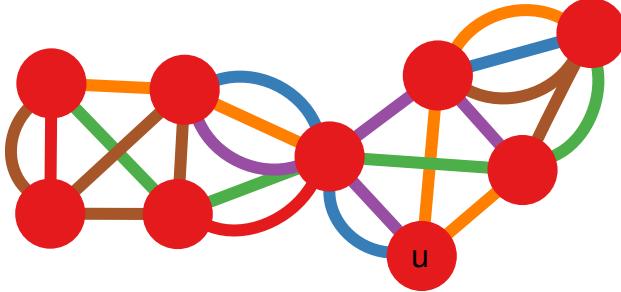


Figure 9.7: A multigraph representation of a multilayer network with one-to-one mapping, where the edge color encodes layer in which it appears.

of “versatility”: the ability of a node to be a relevant central node in different layers^{15,16,17}.

Consider Figure 9.7. Let’s call u the bottom node, the one with two orange edges, a blue and a purple one. We can see that its degree is four (four edges), and its neighbor set is of size three: u has three neighbors, $|N_u| = 3$.

Now, we can count the size of the neighbor set per layer too, or $N_{u,l}$. In the orange layer u has two neighbors ($|N_{u,l}| = 2$), in the blue and purple one it has only one ($|N_{u,l}| = 1$). There is a difference between the neighbors in the orange layer and the ones in the other layers. If u wants to communicate with them, it has to use the orange layer: there is no alternative. On the other hand, if the blue layer were to disappear, u could still use the purple one, and vice versa.

This observation is at the basis of the definition of the “exclusive neighbor” set, or $N_{u,l}^{XOR}$. Given a node u and a layer l , the $N_{u,l}^{XOR}$ contains those neighbors of u that can be reached exclusively via l . If there is an alternative path, those neighbors are not part of $N_{u,l}^{XOR}$. So $|N_{u,l}^{XOR}| = 2$, if l is the orange layer, but $|N_{u,l}^{XOR}| = 0$ in the other two cases. So the exclusive neighbor gives us a rather intuitive measure: how many neighbors would u lose if layer l were to disappear?

We can use $N_{u,l}$ and $N_{u,l}^{XOR}$ to establish some generalized degree definitions, establishing the importance of l for u . For instance, the Layer Relevance of l for u is the fraction of u ’s neighbors that u can reach through l , or $|N_{u,l}| / |N_u|$. In Figure 9.7 that’s 2/3 for the orange layer, and 1/3 for both the blue and the purple layers. The exclusive variant of Layer Relevance is the fraction of u ’s neighbors that u can reach through l and l alone: $|N_{u,l}^{XOR}| / |N_u|$. In Figure 9.7 that’s still 2/3 for the orange layer, but it turns to zero for both the blue and the purple layers.

We can also have a normalized version of Layer Relevance such that it always sums to one for all nodes. In this version, for every pair of connected nodes (u, v) , each layer in which this connection appears does not contribute one to the sum, but $1 / |L_{u,v}|$, where $|L_{u,v}|$ is the number of layers in which u and v are neighbors of each

¹⁵ Manlio De Domenico, Albert Solé-Ribalta, Elisa Omodei, Sergio Gómez, and Alex Arenas. Ranking in interconnected multilayer networks reveals versatile nodes. *Nature communications*, 6: 6868, 2015d

¹⁶ Manlio De Domenico, Albert Solé-Ribalta, Sergio Gómez, and Alex Arenas. Navigability of interconnected networks under random failures. *Proceedings of the National Academy of Sciences*, 111(23):8351–8356, 2014

¹⁷ Federico Battiston, Vincenzo Nicosia, and Vito Latora. Efficient exploration of multiplex networks. *New Journal of Physics*, 18(4):043035, 2016

other. In my example, the normalized Layer Relevance of the orange dimension for u is still $2/3$, but it turns to $1/6$ for the blue and the purple one, because they have to share the remaining $1/3$ of u 's neighbors.

Hyper

As one might expect, allowing edges to connect an arbitrary number of nodes – rather than just two – does unspeakable things to your intuition of the degree. We can still keep our usual definition: the degree in a hypergraph is the number of hyperedges to which a node belongs – or: the number of its hyper-connections^{18,19}. However, if you take any step further, all hell breaks loose. The number of neighbors has no relationship whatsoever with the number of connections: with a single hyperedge you can connect a node with the entirety of the network. Also the average degree is something tricky to calculate. Forget about $\bar{k} = 2|E|/|V|$: if a single hyperedge can connect the entire network, then $|E| = 1$, but $\bar{k} = |V|$.

Things are a bit less crazy for uniform hypergraphs – where we force hyperedges to always have the same number of nodes. Which might explain why they're a much more popular thing to study, rather than arbitrary hypergraphs. I'll deal with the generalization of the degree for simplicial complexes in Section 34.1, because it opens possibilities much more vast than the space I can allow them to have here.

9.2 Degree Distributions

The degree of a node only gives you information about that node. The average degree of a network gives you information about the whole structure, but it's only a single bit of data. There are many ways for a network to have the same average degree. It turns out that looking at the whole degree distribution can shed light on surprising properties of the network itself. Since degree distributions can be so important, generating and looking at them is a second nature for a network scientist. As a consequence, there are a lot of standardized procedures you want to follow, to avoid confusing your reader by breaking them.

Let's break down all the components of a good degree distribution plot. First, the basics. What's a degree distribution? At its most simple, it is just a degree scatter plot: the number of nodes with a particular degree. The degree should be on the x axis and the number of nodes on the y axis, just as I do in Figure 9.8. Commonly, one would normalize the y axis by dividing its values by the number of

¹⁸ Paul Erdős and Miklós Simonovits. Super-saturated graphs and hypergraphs. *Combinatorica*, 3(2):181–192, 1983

¹⁹ Alain Bretto. Hypergraph theory: An introduction. *Mathematical Engineering*. Cham: Springer, 2013

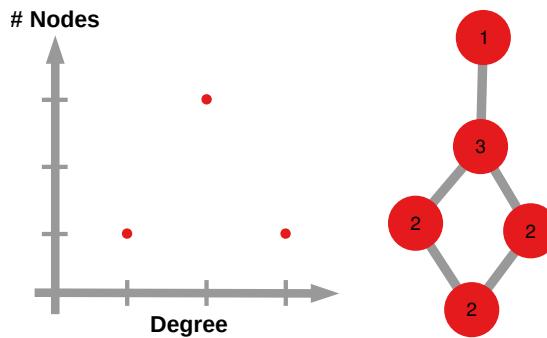


Figure 9.8: The degree scatter plot (left) of the graph on the right.

nodes in the network. At this point, the y axis is the probability of a node to have a degree equal to k , not simply the node count. That makes it easier to compare two networks with a different node count.

Figure 9.9(a) shows you the degree distribution of protein-protein interaction for the *Saccharomyces Cerevisiae*, the beer bug. An interesting pattern is that there are lots of nodes with few interactions, and few nodes with many. As a consequence, we end up with all our datapoints concentrated in the same part of the plot, and it's difficult to appreciate both the low- and the high-degree structure. These degree patterns are more evident and easy to see when represented on a log-log scale, as Figure 9.9(b) shows, which stretches out the low-degree area while compressing the high-degree one.

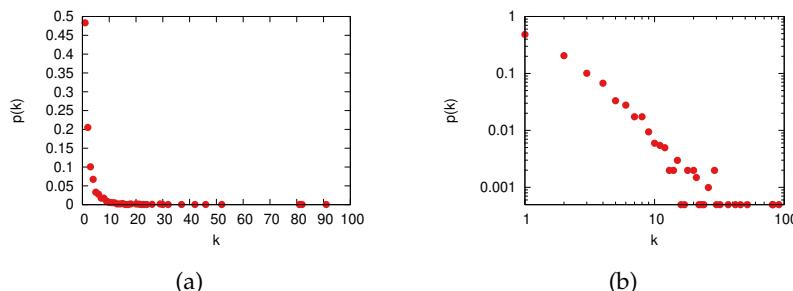


Figure 9.9: The degree distribution of the protein-protein interaction network. The distributions are the same, but in (a) we have a linear scale for the x and y axes, which is replaced in (b) by a log-log scale.

So, we just discovered that this protein-protein interaction network has something peculiar. The baseline assumption would be that nodes connect at random. If that were the case, we would expect the degree to distribute normally, in a nice bell-shape – see Chapter 16. But Figure 9.9(b) is not what a normal distribution looks like. The vast majority of nodes have a very low degree, and a few giant hubs have a degree much larger than average. Is this common?

Yes it is. Most real world networks would show such a broad distribution: email exchanges²⁰, synapses in the brain²¹, internal cell interactions²². Take a look at the degree distribution zoo in Figure 9.10. To put it simply: in most networks we have many orders of

²⁰ Holger Ebel, Lutz-Ingo Mielsch, and Stefan Bornholdt. Scale-free topology of e-mail networks. *Physical review E*, 66(3):035103, 2002

²¹ Victor M Eguiluz, Dante R Chialvo, Guillermo A Cecchi, Marwan Baliki, and A Vania Apkarian. Scale-free brain functional networks. *Physical review letters*, 94(1):018102, 2005

²² Reka Albert. Scale-free networks in cell biology. *Journal of cell science*, 118(21):4947–4957, 2005

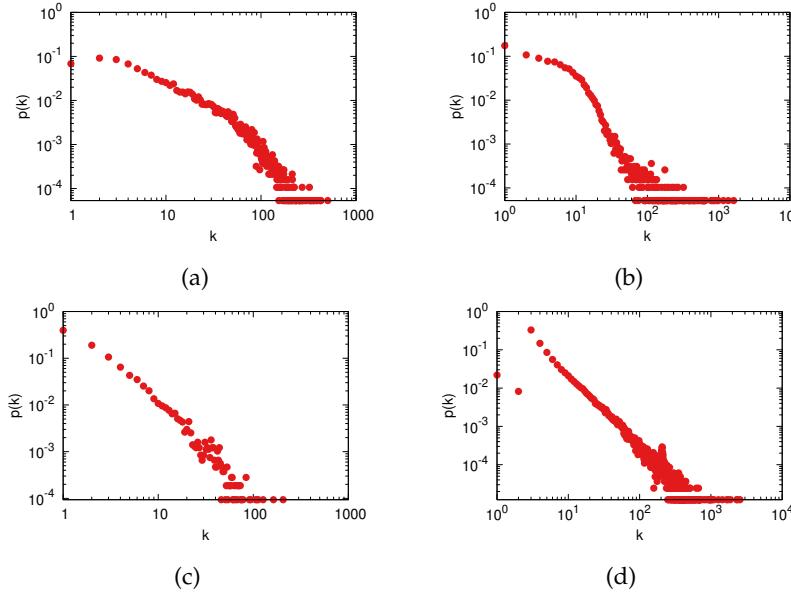


Figure 9.10: The degree distributions of many real-world networks: (a) coauthorship in scientific publication [Leskovec et al., 2007b]; (b) coappearance of characters in the same comic book [Alberich et al., 2002]; (c) interactions of trust between PGP users [Boguñá et al., 2004]; (d) connections through the Slashdot platform [Leskovec et al., 2009].

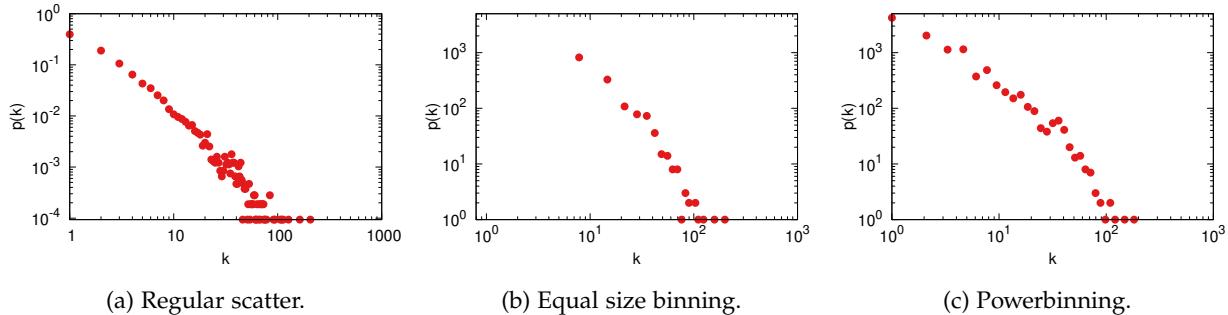
magnitude between the minimum and the maximum degree (x axis), and between the most and least popular degree value (y axis). This is not what scientists initially expected. And when things are not as we expected, we all get excited and start wonder why.

Before exploring these questions we need to finish our deep dive into how to generate and visualize a proper degree distribution. The disadvantages of the degree scatter plots is that they're a bit messy. The physicists in the audience would want a true functional form. But one cannot do that if we have such a broad scatter, especially for high degree values: the wide range of degree values carried only by a node in the network generate what we call a fat tail (Section 3.1).

There are two ways to do it. The first is to perform a power-binning of your x axis²³. Rather than drawing a point for each distinct degree value you have in your network, you can lump together values into larger bins. Using equally-sized bins – with the same increment for the entire space – doesn't work very well: for low degree values you're putting together very populated bins, while for high degree values usually the distribution is so dispersed that you aren't actually grouping together anything. See Figure 9.11(b) for an example. In Figure 9.11(b) we completely lost the head of the distribution – the low degree values are all lumped together – and, while less prominent, the fat tail is still there.

That's why you do power binning. You start with a small bin size, usually equal to 1. Then each bin becomes progressively larger, by a constant multiplicative factor. At first, the bins are still small. But, as you progress, the bins start to be large enough to group a significant

²³ Staša Milojević. Power law distributions in information science: Making the case for logarithmic binning. *Journal of the American Society for Information Science and Technology*, 61(12):2417–2425, 2010



portion of your space²⁴. A good power bin choice can make the plot clearer, as the one in Figure 9.11(c). In Figure 9.11(c) we saved the head of the distribution and further reduced the fat tail.

One can do better than Figure 9.11(c), that's why in network papers you rarely see power-binned distributions. An issue of power binning is that it forces you to make a choice: to determine the bin size function. Having a choice is a double-edged sword: it opens you to the possibility of tricking yourself into seeing a pattern that is not there.

The most common way to visualize degrees is by drawing cumulative distributions (CDF), or – to be more aligned with the convention you'll see everywhere – the complement of a cumulative distribution (CCDF). We can transform a degree histogram into a CCDF by changing the meaning of the y-axis. Rather than being the probability of finding a node of degree equal to k , in a CCDF this is the probability of finding a node of degree k or higher. This is not a scattergram any more, but a function, which helps us when we need to fit it. Figure 9.12 shows an example, where we go from a degree histogram to its equivalent CCDF.

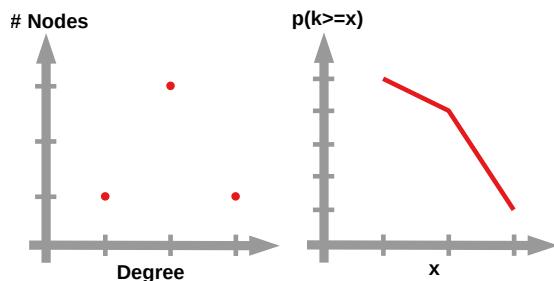


Figure 9.11: The degree distribution of the PGP trust network.

²⁴ An example of power binning, starting with size 1 and increasing the bin size by 10% at each step: [1, 2, 3, 5, 6, 8, 9, 11, 14, ..., 1410, 1552, 1709, 1881, 2070, 2278, ...]

We can see the relationship of our protein-protein network more clearly in Figure 9.13. It appears that, in log-log space, the relationship between degree and the number of nodes with a given degree is fixed. This relationship can be approximated with a straight line – at least asymptotically: in Figure 9.13(b) you can see that the head

Figure 9.12: The degree scatter plot (left) and its corresponding complement of the cumulative distribution (CCDF).

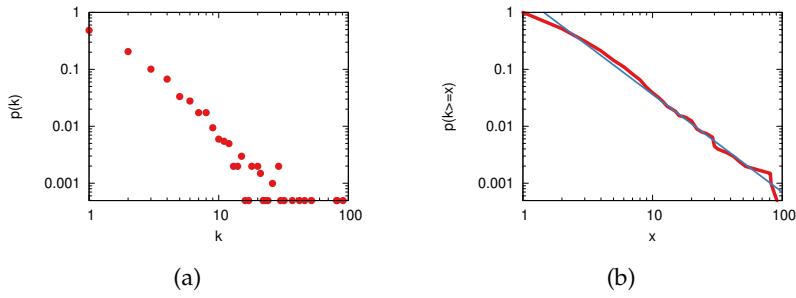


Figure 9.13: The degree distribution of the protein-protein interaction network. The distributions are the same and are both in log-log scale, but in (a) we have the degree histogram, and in (b) we show the CCDF version (with the best fit in blue).

doesn't really fit. Is this a coincidence, or does it have meaning? To answer this question we need to enter in the wonderful world of power-law degree distributions and scale free networks.

9.3 Power Laws and Scale Free Networks

In statistics, a power law is a functional relationship between two quantities, where a relative change in one quantity results in a proportional relative change in the other quantity. The relation is independent of the initial size of those quantities: one quantity varies as a power of another. I show what I mean in Figure 9.14: each time you move on the x-axis by a specific increment, you also always move on the y-axis by a fixed function of that x increment, no matter where you are in the distribution (head, tail, or middle).

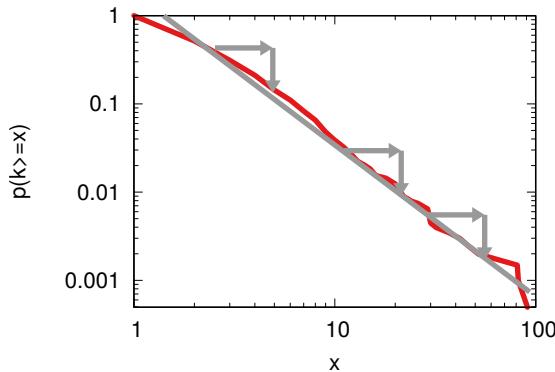
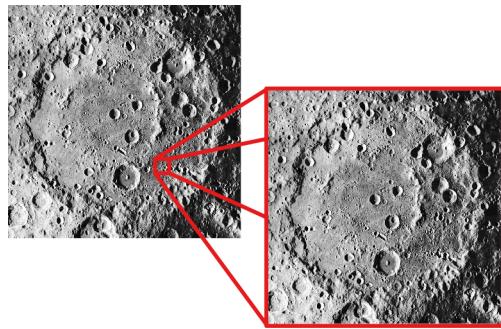


Figure 9.14: An example of power law, showing how the red line always goes down by the same proportion as its right movement, no matter if we look a head, middle or tail.

You can find power laws in nature in many places: the frequencies of words in written texts, the distribution of earthquake intensities, etc... To grasp the concept you need a visual example, and my favorite is moon craters²⁵. You can see in Figure 9.15 there are a lot of tiny craters caused by small debris and a huge one. This is fractal self-similarity: if the moon were an infinite plane, you could zoom in and out the picture and the size distributions would be the same. This is the scale invariance I'm talking about: no matter the zoom, the

²⁵ Mark EJ Newman. Power laws, pareto distributions and zipf's law. *Contemporary physics*, 46(5):323–351, 2005b



picture looks the same – obviously in reality it doesn't, because the moon isn't an infinite plane, and you cannot zoom in infinitely many times (in fact, whether finite systems can actually generate power laws is a controversial topic²⁶).

This applies to networks too! There are many studies showing how some networks possess this sort of self-similar structure at different scales^{27,28} – i.e. they are fractals. This is not necessarily the same thing as looking at the degree distribution²⁹ – although classifying a network as "scale free" by looking at its degree distribution is a common operation in the literature and it is also the stance I'll adopt from now on in the book.

In Figure 9.16, I show the usual CCDF of the protein-protein network: there we see that 50% of the nodes have a degree of 2 or more. This means that 50% of the nodes have degree equal to one. A formula you'll see everywhere links the probability of a node having degree k to k to the power of a constant α . Mathematically speaking, the scale free network master equation is:

$$p(k) \sim k^{-\alpha}.$$

In this formula, we call α the scaling factor. Its value is important, because it determines many properties of the distribution. In general,

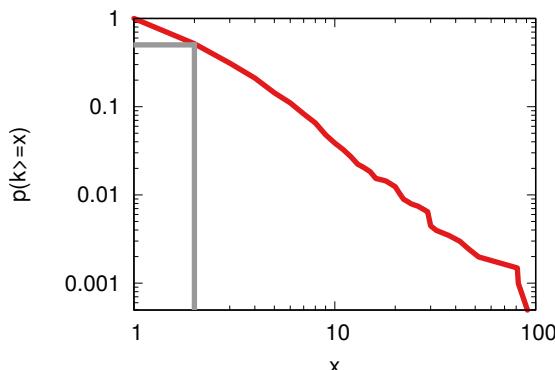


Figure 9.15: The distribution of crater sizes in the moon is an example of quasi-power law, with many tiny ones and a huge one spanning the entire picture. If the moon were an ideal infinite plane and we could zoom in indefinitely, the picture we would get would be equivalent to the original one, i.e. the scale at which we're observing the moon would not influence the observation result.

²⁶ Michael PH Stumpf and Mason A Porter. Critical truths about power laws. *Science*, 335(6069):665–666, 2012

²⁷ Chaoming Song, Shlomo Havlin, and Hernan A Makse. Self-similarity of complex networks. *Nature*, 433(7024):392–395, 2005

²⁸ M Angeles Serrano, Dmitri Krioukov, and Marián Boguná. Self-similarity of complex networks and hidden metric spaces. *Physical review letters*, 100(7):078701, 2008

²⁹ Jin Seop Kim, Kwang-Il Goh, Byungnam Kahng, and Doochul Kim. Fractality and self-similarity in scale-free networks. *New Journal of Physics*, 9(6):177, 2007

Figure 9.16: An example of power law in a CCDF. The vertical gray bar shows that the point in the distribution is associated with degree equal to two. The horizontal gray bar shows that this degree correspond to a probability of around 0.5. This means that half of the network has a degree equal to or greater than two. Or, in other words, that the other half of the network has degree equal to one.

if a real world network has a power law degree distribution, α tends to be low ($\alpha \sim 2$, and for a majority $\alpha < 3$, although you can find networks with higher α s). This is rather unfortunate, because it means that the degree distribution has a well defined mean, but not a well defined variance (Section 3.4). This implies that the average degree has meaning, but it's not very useful to do anything more than a superficial description of the network.

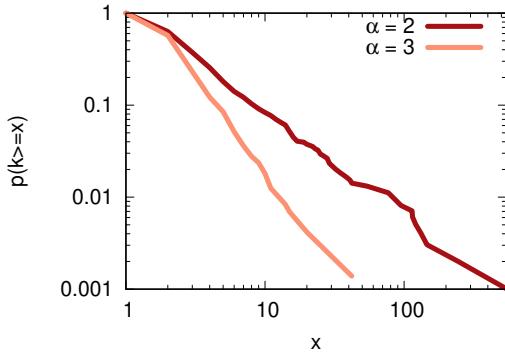


Figure 9.17: The CCDF degree distributions of two random networks with different α exponents.

This is all well and good, but what does it mean exactly to have $\alpha = 2$ or $\alpha = 3$? How do two networks with these two different coefficients look like? I provide an example of their degree distributions in Figure 9.17, and I show two very simple random networks with such degree distributions in Figure 9.18 – obviously, systems this small are a very rough approximation. From Figure 9.17 you see that α determines the slope of the degree distribution, with a steeper slope for $\alpha = 3$. This means that the hubs in $\alpha = 3$ are “smaller”, they do not have a ridiculously high degree.

Figure 9.18 confirms this: in Figure 9.18(a) you see that, for $\alpha = 2$, you have only one obvious hub that is head and shoulders above the

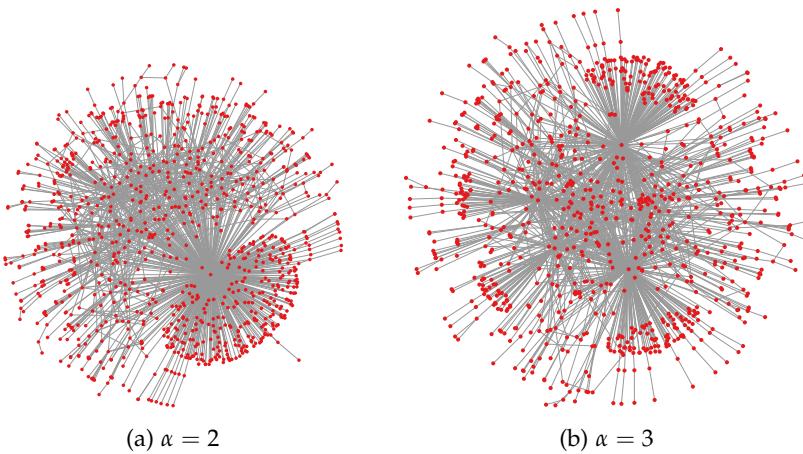
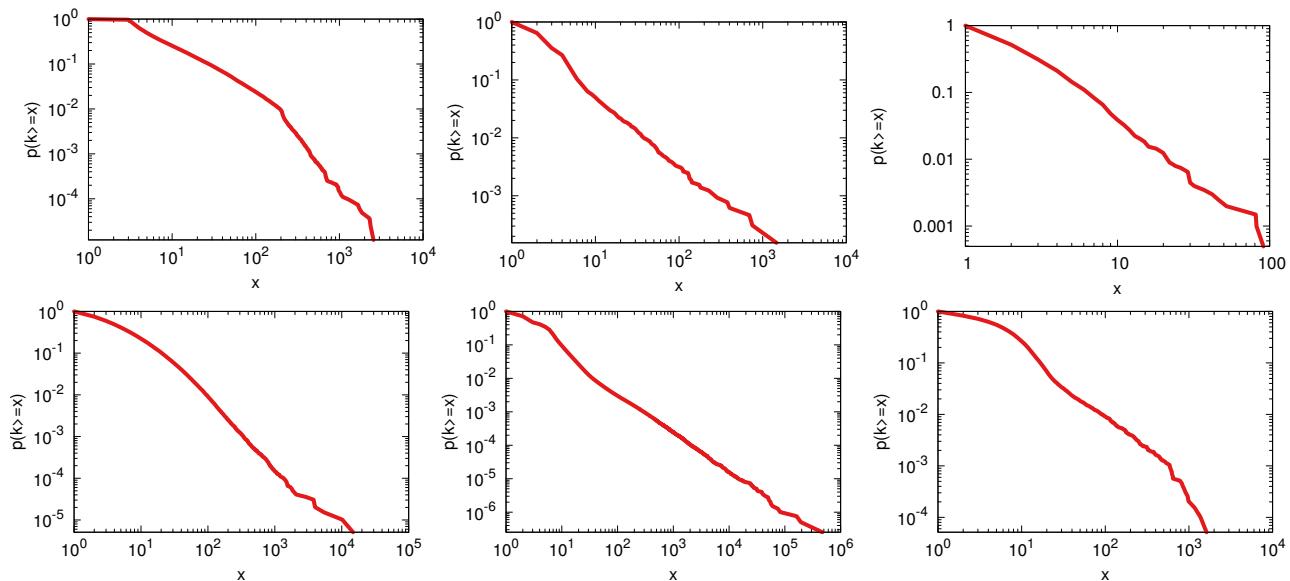


Figure 9.18: An example of two networks with scale free degree distributions, with different α exponents.

rest, practically connected to the entire network. In Figure 9.18(b), instead, you still have a clear winner catching your eye (in the top), but it is much closer to the second best hub.

The average degree is heavily influenced by the outliers with thousands of connections. For instance, in Figure 9.16 the average degree is equal to three, meaning that around 70% of nodes are below average. This is well illustrated by the stadium example: you have a stadium with 79,999 individuals sampled at random from the US population. If you calculate their average net worth you'll obtain a value – it's difficult to be precise, but let's say it's around \$100,000. So their total net worth is ~ 8 billion dollars. However, the 80,000th person entering the stadium is our outlier hub: Jeff Bezos. His net worth alone is 192 billion dollars³⁰. The new average is 200 billion divided by 80 thousand people: 2.5 million dollars. The average shifted dramatically: 2.5 million is *very* different from 100 thousand. This is because net worth distributes broadly and thus has a crazy variance, which causes tremendous shifts in the average. This makes it incorrect to apply to this kind of distribution traditional statistics that are based on variance and standard deviation – such as regression analysis, as we'll see in the next section.



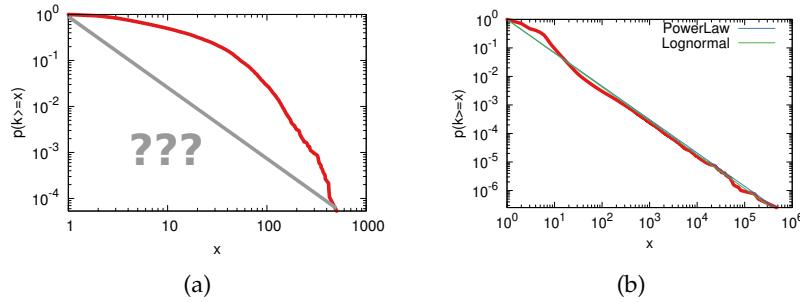
Early works have found power law degree distributions in many networks, prompting the belief that scale free networks are ubiquitous. In fact, this seems true. Figure 9.19 shows the CCDFs of many networks: protein interactions, PGP, Slashdot, DBpedia concept network, Gowalla, Internet autonomous system routers.

But we need to be aware of our tendency of seeing patterns when

³⁰ Bear with me, I know it has probably doubled by the time you read this paragraph.

they aren't there – after all, as Feynman says, the easiest person you can fool is yourself. So in the next section I'll give you an arsenal to defend yourself from your own eyes and brain.

9.4 Testing Power Laws



Often, people will just assume that any degree distribution is a power law, calling “power laws” things that are not even deceptively looking like power laws. I’ve seen distributions as the one in Figure 9.20(a) passing as power laws and that’s just... no. However, I don’t want to pass as one perpetuating the myth that “everything that looks like a straight line in a log-log space is a power law”. That is equally wrong, even if more subtle and harder to catch.

Seeing the plot in Figure 9.20(b), you might be tempted to perform a linear fit in the log-log space. This more or less looks like fitting the logged values with a $\log(p(x)) = \alpha \log(x) + \beta$. Transforming this back into the real values, the slope α becomes the scaling factor, and β is the intercept, in other words: $\log(p(x)) = \alpha \log(x) + \beta$ is equivalent to $p(x) = 10^\beta x^\alpha$ – assuming you logged to the power of ten.

A small aside: if you were to do this on the distributions from Figure 9.17, you would expect to recover $\alpha \sim 2$ and $\alpha \sim 3$, because I told you I generated the degree distributions with those exponents. Instead, you will obtain $\alpha \sim 1$ and $\alpha \sim 2$, respectively. That is because, in Figure 9.17, I showed you the CCDF of the degree distribution, not the distribution itself. The CCDF of a power law is also a power law, but with a different exponent³¹. If you’re doing the fit on the CCDF, you have to remember to add one to your α to recover the actual exponent of the degree distribution.

Back to parameter estimation. If you perform a simple linear regression, you’ll get an unbelievably high R^2 associated to a super-significant p value. Well, of course: you’re fitting a straight line over a straight-ish line. Does that mean you’re looking at a power law? Not really.

Just because something looks like a straight line in a log-log plot,

Figure 9.20: (a) An example of a CCDF that is most definitely NOT a power law, but that a researcher with a lack of proper training might be fooled into thinking it is. (b) Fitting a power law (blue) and a lognormal (green) on data (red) can yield extremely similar results.

³¹ Heiko Bauke. Parameter estimation for power-law distributions by maximum likelihood methods. *The European Physical Journal B*, 58(2):167–173, 2007

it doesn't mean it's a power law. You need a proper statistical test to confirm your hypothesis. The reason is that other data generating processes, such as the ones behind a lognormal distribution, can generate plots that are almost indistinguishable from a power law. Figure 9.20(b) shows an example. You cannot really tell which of the two functions fits the data better.

What you need to do is to fit both functions and then estimate the likelihood (Section 4.3) of each model to explain the observed data³². This can be done with, for instance, the `powerlaw` package^{33,34} – available for Python. However, be prepared for the fact that having a significant difference between the power law and the lognormal model is extremely hard.

In most practical scenarios, you'll have to argue that your network is a power law. How could you do it? Well, in complex networks power law degree distributions can arise by many processes, but one in particular has been observed time and time again: cumulative advantage. Cumulative advantage in networks says that the more connections a node has, the more likely it is that the new nodes will connect to it. For instance, if you write a terrific paper which gathers lots of citations this year, next year it will likely gain more citations than the less successful papers³⁵.

This is the same mechanism behind – for instance – Pareto distributions and the 80/20 rule. Pareto says that 80% of the effects are generated by 20% of the causes³⁶. For instance, 20% of people control 80% of the wealth. And, given that it takes money to make money, they are likely to hold – or even grow – their share, given their ability to unlock better opportunities. In fact, the Pareto distribution is a power law. Similar to this is Zipf's Law, the observation that the second most common word in the English language occurs half of the time as the most common, the third most common a third of the time, etc^{37,38,39}. In practice, the n th word occurs $1/n$ as frequently as the first, or $f(n) = n^{-1}$, which is a power law with $\alpha = 1$.

This is opposed to the data generating process of a lognormal distribution. To generate a lognormal distribution you simply have to multiply many random and independent variables, each of which is positive. A lognormal distribution arises if you multiply the results of many ten-dice rolls. You can see that there is no cumulative advantage here: scoring a six on one die doesn't make a six more likely on any other die – nor influences subsequent rolls.

So, to sum up, to test for a power law you have to do a few things. First, make sure that your observations cannot be explained with an exponential. Confusion between a power law and some other distribution such as an exponential is hard. If you think a distribution might be an exponential, then it's definitely not a power law. Second,

³² Aaron Clauset, Cosma Rohilla Shalizi, and Mark EJ Newman. Power-law distributions in empirical data. *SIAM review*, 51(4):661–703, 2009

³³ Jeff Alstott and Dietmar Plenz Bullmore. powerlaw: a python package for analysis of heavy-tailed distributions. *PloS one*, 9(1), 2014

³⁴ <https://github.com/jeffalstott/powerlaw>

³⁵ Derek de Solla Price. A general theory of bibliometric and other cumulative advantage processes. *Journal of the American society for Information science*, 27(5):292–306, 1976

³⁶ Vilfredo Pareto. *Manuale di economia politica con una introduzione alla scienza sociale*, volume 13. Società editrice libraria, 1919

³⁷ Jean-Baptiste Estoup. *Gammes sténographiques: méthode et exercices pour l'acquisition de la vitesse*. Institut sténographique, 1916

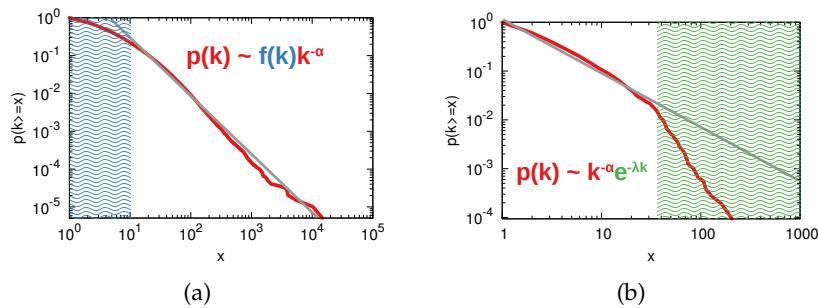
³⁸ Felix Auerbach. Das gesetz der bevölkerungskonzentration. *Petermanns Geographische Mitteilungen*, 59:74–76, 1913

³⁹ GK Zipf. The psycho-biology of language. 1935

try to see if you can statistically prefer a power law model over a log-normal. In the likely event of you not being able to mathematically do so, you should look at your data generating process. If you have the suspicion that it could be due to random fluctuations, then you might have a lognormal. Otherwise, if you can make a convincing argument of non-random cumulative advantage, go for it.

There are a few more technicalities. Pure power laws in nature are – as I mentioned earlier – rare⁴⁰. Your data might be affected by two impurities. Your power law could be shifted⁴¹, or it could have an exponential cutoff⁴². In a shifted power law, the function holds only on the tail. In an exponential cutoff the power law holds only on the head.

Shifted power laws have an initial regime where the power law doesn't hold. Formally, the power law function needs a slowly growing function on top that will be overwhelmed by the power law for large values of k – as I show in Figure 9.21(a). So we modify our master equation as: $p(k) \sim f(k)k^{-\alpha}$, with $f(k)$ being an arbitrary but slowly growing. Slowly growing means that, for low values of k it will overwhelm the $k^{-\alpha}$ term, but for high values of k , the latter would be almost unaffected. In power law fitting, this means to find the k_{min} value of k such that, if $k < k_{min}$ we don't observe a power law, but for $k > k_{min}$ we do.



Shifted power laws practically mean that “Getting the first k_{min} connections is easy”. If you go and sign up for Facebook, you generally already have a few people you know there. Thus we expect to find fewer nodes with degree 1, 2, or 3 than a pure power law would predict. The main takeaway is that, in a shifted power law, we find fewer nodes with low degrees than we expect in a power law.

Truncated power laws are typical of systems that are not big enough to show a true scale free behavior. There simply aren't enough nodes for the hubs to connect to, or there's a cost to new connections that gets prohibitive beyond a certain point. This is practically a power law excluding its tail, that's why we call them “truncated”. Mathematically speaking, this is equivalent to having an

⁴⁰ Whether this holds true also for networks is the starting point of a surprisingly hot debate, see for instance [Broido and Clauset, 2019] and [Voitakov et al., 2018].

⁴¹ Gudlaugur Jóhannesson, Gunnlaugur Björnsson, and Einar H Gudmundsson. Afterglow light curves and broken power laws: a statistical study. *The Astrophysical Journal Letters*, 640(1):L5, 2006

⁴² Aaron Clauset, Cosma Rohilla Shalizi, and Mark EJ Newman. Power-law distributions in empirical data. *SIAM review*, 51(4):661–703, 2009

Figure 9.21: (a) An example of shifted power law. The area in which the power law doesn't hold is shaded in blue. (b) An example of truncated power law: a power law with an exponential cutoff. The area in which the power law doesn't hold is shaded in green.

exponential cutoff added to our master equation: $p(k) \sim k^\alpha e^{-\lambda k}$. The exponential function is dominated by the power law function for low values of k , but it becomes dominant for high values of k . See Figure 9.21(b) for an example.

Truncated power laws practically mean that “Getting the last connections is hard”: the biggest superstar on Twitter has a lot of followers, but relatively speaking they are not that many more as the second biggest superstar on Twitter. Thus its degree is not as big as we would expect. The main takeaway is that, in a truncated power law, the hubs have lower degrees than we expect in a power law.

At the end of the day, it doesn’t matter too much if your network has an exponential, lognormal or power law degree distribution. On one thing the brotherhood of network scientists can agree: the vast majority of networks have broad degree distributions, spanning multiple orders of magnitude. Most nodes have below-average degree and hubs lie many standard deviations above the average. Even if they are not power laws at all, that’s still pretty darn interesting.

9.5 Summary

1. The degree is the number of edges connected to a node and it’s probably a node’s most important and basic feature. It tells us how well connected and how structurally important the node is.
2. In more complex graph models (directed, weighted, bipartite, multilayer), the degree measure becomes itself more complex. For instance, in directed networks you have both in- and out-degree, depending on the direction of the edge.
3. The degree distribution is a plot telling you how many nodes have a specific degree value in the network, and it is one of the network’s most important properties.
4. When plotting degree distributions, the standard choice is the complement of the cumulative distribution, shown in a log-log scale.
5. Many networks have a power law degree distribution, but rarely this is a pure power law: it is often shifted or truncated. Fitting a power law and finding the correct exponent is tricky and you should not do it using a linear regression: you should use specialized tools.
6. Moreover, determining whether the degree follows a power law is useful for modeling and theory, but it isn’t crucial empirically. The interesting thing is that networks have broad and unequal degree

distributions. You can describe them with statistics that are easier to get right than the tricky business of fitting power laws.

9.6 Exercises

1. Write the in- and out-degree sequence for the graph in Figure 9.3(a). Are there isolated nodes? Why? Why not?
2. Calculate the degree of the nodes for both node types in the bipartite adjacency matrix from Figure 9.5(a). Find the isolated node(s).
3. Write the degree sequence of the graph in Figure 9.7. First considering all layers at once, then separately for each layer.
4. Plot the degree distribution of the network at <http://www.networkatlas.eu/exercises/9/4/data.txt>. Start from a plain degree distribution, then in log-log scale, finally plot the complement of the cumulative distribution.
5. Estimate the power law exponent of the CCDF degree distribution from the previous exercise. First by a linear regression on the log-log plane, then by using the `powerlaw` package. Do they agree? Is this a shifted power law? If so, what's k_{min} ? (Hint: `powerlaw` can calculate this for you)
6. Find a way to fit the truncated power law of the network at <http://www.networkatlas.eu/exercises/9/6/data.net>. Hint: use the `scipy.optimize.curve_fit` to fit an arbitrary function and use the functional form I provide in the text.

10

Paths & Walks

So far, we adopted a static vision of a network. We have a structure and we ask simple questions about the structure as it is. Does it have directed edges? What's the degree of the nodes? Those are interesting questions, but a network really shines when you use it for what it is for: exploring its connections.

To understand what I mean, let's come back to the social network example. Nodes are people and edges connect friends. Suppose you want to send a message to a person you are not connected to. Maybe you want to sell them a new shampoo. How do you do it? Well, you could tell the message to a friend, and instruct them to pass the message on. This means having a packet of information travel through the network, exploiting its edges.

There are many ways to cross a network using its edges, depending on which restrictions you want to put on your exploration. I'm going to define a few technical terms (following graph theory literature¹) but, if you find it simpler, you can call all of them "paths" plus a qualifier specifying what type of path it is. I'm going to present both conventions and you simply use the one you find most natural – as everybody does in network analysis.

The most basic way to explore a graph is by performing a **walk**, or "path with repeating nodes". A walk is a sequence of nodes with the property that each node in the sequence is adjacent to the node next to it. In a walk you're not imposing any rule in your exploration. You can go back and forth between the same two nodes by using the same edge as many times as you want. The **length** of a walk is the number of times you're using the edges in your walk. If you use the same edge n times, this will increase the walk's length by n . Figure 10.1(a) shows an example of a walk of length 6 in a network.

In a walk the choice of the next edge to explore is yours. You can have a slightly more constrained definition of a walk, where you put rules to choose the next edge to traverse. For instance, in the random walk you impose to make this choice completely at random.

¹ Reinhard Diestel. *Graph theory*. Springer Publishing Company, Incorporated, 2018

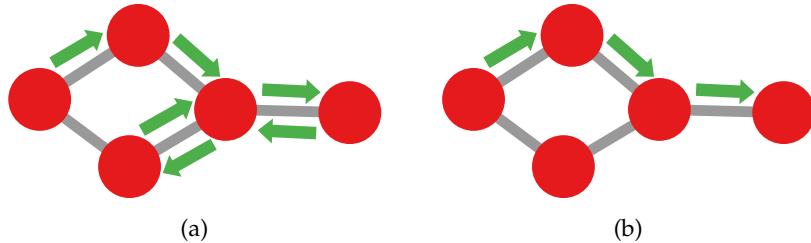


Figure 10.1: (a) An example of a walk of length six in the network, following the green arrows. (b) An example of a path of length three in the network.

We already saw a way to calculate node exploration probabilities via a random walk using powers of the adjacency matrix in Section 8.1. We'll see that random walks are a phenomenally powerful way to explore your network's properties and are at the basis of countless methods: Chapter 11 will be but a superficial introduction.

When you impose even more constraints on your walks, then you can generate a **path**, or "simple path" (Figure 10.1(b)). This is a walk that does not repeat nodes nor edges. Again, you can put more qualifiers on your path to make it special. For instance, recalling the seven bridges problem, an Eulerian path is a path that travels through all edges of a connected graph – since it is a path, not only it has to visit each edge, but it also has to do it exactly once. A cousin of the Eulerian path is the Hamiltonian path, which instead wants to visit each node – not edge – exactly once. More interestingly, you can try to find the *shortest* path between two nodes. That will be the topic of Chapter 13.

Similarly to a walk, a path has a length as well. This is again defined as the number of edges the path crosses. Since no edge can be used twice in a path, this is also the number of distinct edges used.

10.1 Walks and Matrices

In Chapter 8 I showed you that taking powers of the stochastic matrix is fun, because it tells us the probability of a random walker going from nodes u to v . But looking at ye olde regular adjacency matrix can be insightful. If A is binary and its diagonal is set to zero, then A^n can tell us lots of interesting things.

First, if $A_{uv}^n = 0$, then there are no walks of length n going from u to v . In Figure 10.2(b) you see that $A_{1,7}^2$ is zero, because node 7 is only connected to node 6. Node 6 isn't connected to node 1 so there's no way to go from 1 to 7 in two hops. If, instead, $A_{uv}^n > 0$, then the A_{uv}^n is exactly the number of such walks! There are two ways to go from node 1 to node 3 in two hops in Figure 10.2(b): one via node 2 and one via node 4, since they're both connected to node 3.

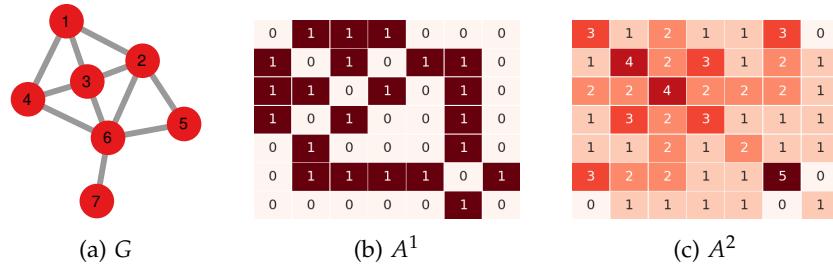


Figure 10.2: (a) A graph. (b-c) Different powers of its binary adjacency matrix A .

It doesn't end here: you can set the diagonal of A to 1 by summing to it the identity matrix I . Then, $(I + A)^n$ tells you the number of walks of length n or less. The reason is that the diagonal represents self loops. If it is set to 1, the walker in u can choose to follow the self loop to u an arbitrary number of times before reaching v .

Every time you calculate A^n , the resulting diagonal is interesting. Specifically, A_{uu}^n is the number of loops or closed walks of length n – walks starting and ending in the same node – to which u participates. For $n = 2$ we have a special case: this value is equal to the degree – or $A_{uu}^2 = k_u$. Check the diagonal of Figure 10.2(b) if you don't believe me! This means that you could consider A_{uu}^n as a sort of generalized degree.

10.2 Cycles

You can make a walk and a path in any graph, no matter its topology. There is a special path that you cannot always do, though. That is the cycle. Picking up the social network example as before, now you're not happy just by reaching somebody with your message. You want the message you originally sent to come back to you. Also, you don't want anybody to hear it twice. If you manage to do so, then you have found a cycle in your social network.

A **cycle** is a path that begins and ends with the same node. Note that I said "path", so we don't have any repeated nodes nor edges – except the origin, of course. Figure 10.3(a) shows an example of a cycle in the network. The cycle's length is the number of edges you use in your cycle. Given its topological constraints, that is also the number of its nodes.

Imposing cycles to be paths make them a non trivial object to have in your network. We can easily see why there might be nodes that participate in no cycles. If a node has degree equal to one, you can start a path from it, but you can never go back to complete a cycle. Doing so would force you to re-use the only connection they have. Thus a cycle is impossible for such nodes.

In fact, we can go further. We can imagine a network structure

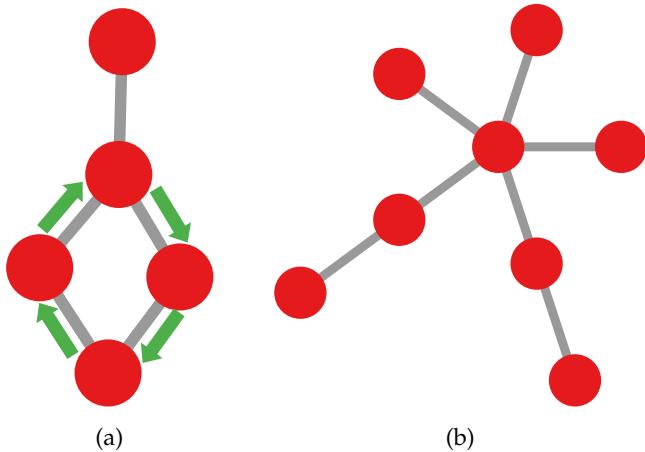


Figure 10.3: (a) An example of a cycle in the network, following the green arrows. (b) A tree.

that has no cycles at all! I draw one such structure in Figure 10.3(b). No matter how hard you squint, you're never going to be able to draw a cycle there. We have a special name for such structures: **trees**. Trees are simple graphs with no cycles. In a tree you cannot get your message back, unless somebody hears it twice. Given their lack of cycles, some even call them **acyclic graphs**.

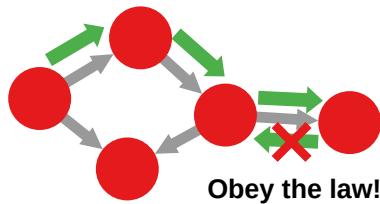


Figure 10.4: An example of a directed walk, where we cannot explore an edge if we do not respect its direction.

Directed networks add some spice to these concepts. In a directed social network, people are only willing to pass messages to their friends. If the friendship is not reciprocated, the receiver will not pass the message back to the sender. In practice, a walk – and a path, and a cycle – cannot use edges pointing to the direction opposite of the one they want to go. Figure 10.4 shows a naughty walk trying to do exactly that.

In the undirected case, cycles create only two types of networks: those who have them (cyclic networks) and those who don't (acyclic networks). Instead, in the directed case, we can create a larger zoo of different directed structures. Figure 10.5 showcases them. Figure 10.5(a) is the basic case: we have a cycle connecting the four nodes at the bottom, thus it is a **directed cyclic graph**. There's no such cycle present in Figure 10.5(b), thus we call it a **directed acyclic graph**.

There *could* be a cycle in Figure 10.5(b), if we were to ignore edge directions. That is why we create a category for those directed graphs that would be acyclic if we were to ignore the edge directions. These

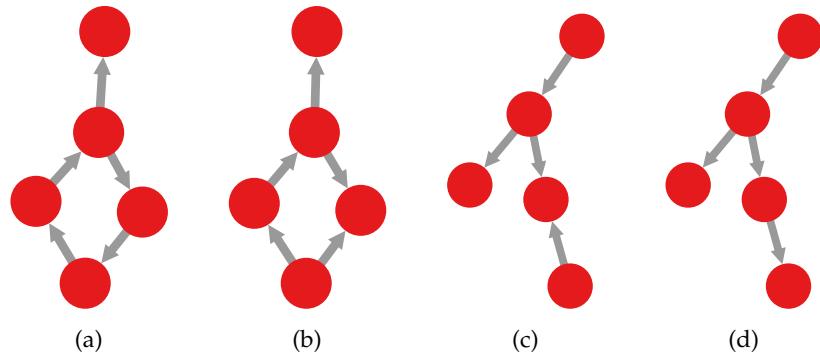


Figure 10.5: (a) A directed cyclic graph. (b) A directed acyclic graph. (c) A directed tree. (d) An arborescence.

are **directed trees**, and Figure 10.5(c) provides an example.

If you – like me – have even a mild case of self-diagnosed OCD, you’ll probably be as irritated as I am about Figure 10.5(c). There’s a natural flow to that directed tree, except for that little pesky edge at the bottom, going into the opposite direction. To restore sanity to the network world, we decided to create a final definition for directed graphs: **arborescences**. This is French for “tree”, and in fact the two terms are often used interchangeably. But, technically speaking, an arborescence is a directed tree in which all nodes have in-degree of one, except the root. In an arborescence, the root is a special node: the only one with in-degree of zero. An arborescence must have one and only one root. Figure 10.5(d) fixes Figure 10.5(c) to be compliant to the definition of arborescence, and it is a work of art. So satisfying.

10.3 Reciprocity

Directed networks allow for a special type of path. In an undirected network without parallel edges, each path using two edges will necessarily bring you to a third node. You simply cannot use the same edge to go back to your origin. This is actually possible in a directed network. That is because the edge bringing you from u to v is not the same edge that brings you back to u from v – by definition of what a directed edge is.

In the social network case, this is about replying messages, or considering as a friend somebody who also consider you as their friend. So these are cycles of length two, or containing two nodes and two edges.

In a social network, it is interesting to know the probability that, if I consider you my friend, you also consider me your friend – which hopefully is 100%, but it rarely is so. This is an important quantity in network analysis, and we give it a name. We call it **reciprocity**, because it is all about reciprocating connections.

To calculate reciprocity we count the number of connected pairs of

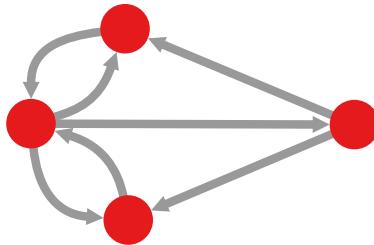


Figure 10.6: An example of a directed network with some reciprocal edges.

the network: pairs of nodes with at least one edge between them. In Figure 10.6, we have five connected pairs. Then we count the number of connected pairs that have both possible edges between them: the ones reciprocating the connection. In Figure 10.6, we have two of them. Reciprocity is simply the second count over the first one. So, for the example in Figure 10.6, we conclude that reciprocity is $2/5$, or that the probability of a connection to be reciprocated is 40%. Sad.

10.4 Connected Components

Walks and paths can help you uncover some interesting properties in your network. Let's pick up our game of message-passing. In this scenario, we might end up in a situation where there is no way for a message to reach some of the people in the social network. The people you can reach with your message do not know anybody who can communicate to your intended targets. In this scenario, it is natural to divide people into groups that *can* talk to each other. These are the network's "components".

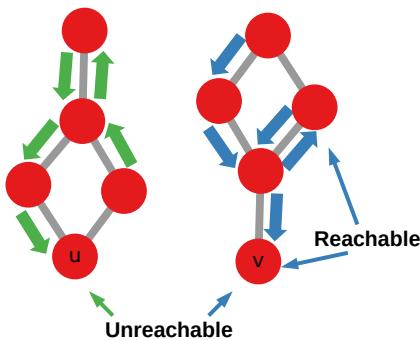


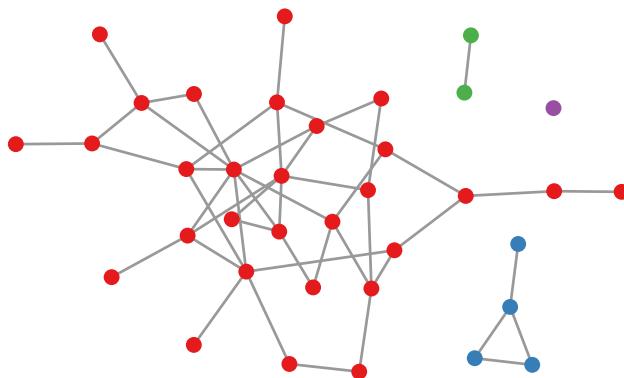
Figure 10.7: A network with two connected components, each with five nodes. No matter how long you try, you can never find a path starting from u and ending in v .

I can translate what I just said in terms of walks and paths. If two nodes cannot be connected by a walk, then they are on different connected components. Connected components are subgraphs whose nodes can be reached from one another by following the edges of the network. The network in Figure 10.7 has two connected components: the nodes reachable with green-like walks, and the ones reachable by blue-like walks.²

² One nice thing about graph theorists is that they are less bad than average scientists in naming things. For instance, if you have a network made by different connected components and each of those components is a tree, then you can call that a "forest". 'cause it's made of trees. You get it? Anyone?

A network with multiple connected components is usually bad news. The whole point of a network is to connect nodes together so that they are in the same shared structure. However, when you have multiple connected components, you effectively have two – or more – separate networks which cannot talk to each other. That's a bummer.

As you might expect, real world networks tend to have multiple components. Reality always comes in the way of a good story. However, there is a silver lining. The vast majority of real networks host most of their nodes in a single connected component. In practice, networks have what we call a “giant connected component” (GCC). One of the components of the network is usually ridiculously larger than all the others^{3,4}, as is the case in Figure 10.8.



I mentioned earlier in this section that there is a relationship between some matrix operations and random walks. If you recall Section 8.1, raising the stochastic adjacency matrix to the power of n tells you the probability of reaching a node with a random walk. So you might expect that there are also some matrix operations related to connected components.

Indeed, there are. We are interested in the eigenvalues of the stochastic adjacency matrix – a more in-depth explanation of why this is the case will come in Section 11.1. In Section 8.2 I said that the largest eigenvalue of the stochastic adjacency is equal to one. However, I also mentioned that the second eigenvalue λ_2 could also be equal to one – and so could λ_3 and so on.

It turns out that the number of eigenvalues equal to one is the number of components in the graph. The reason is that you can consider the adjacency matrix as two adjacency matrices pasted into the same. They are disjoint matrices and each has as a maximum eigenvalue of one. I show an example in Figure 10.9.

If you can use the leading eigenvalues to count the number of connected components (Figure 10.9), the leading eigenvectors tell you to which component the nodes belong. If the network has two

³ Svante Janson, Donald E Knuth, Tomasz Łuczak, and Boris Pittel. The birth of the giant component. *Random Structures & Algorithms*, 4(3):233–358, 1993

⁴ Sergey N Dorogovtsev, José Fernando F Mendes, and Alexander N Samukhin. Giant strongly connected component of directed networks. *Physical Review E*, 64(2):025101, 2001

Figure 10.8: A network with a giant connected component. The node color codes the component containing the node.

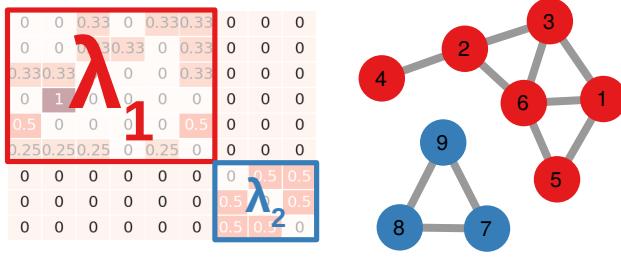


Figure 10.9: The stochastic adjacency matrix of a disconnected graph looks like two different adjacency matrices pasted on the diagonal. Thus, they both have a (different) leading eigenvector equal to one.

components, the nodes belonging to one will have a non-zero value in the eigenvector, while the nodes which do not belong to that component will have a zero – see Figure 10.10. As you might have already deduced, if there is only one component then the leading eigenvector contains the same non-zero value. Similar properties hold for the Laplacian. The smallest eigenvectors of L play the very same role as did the largest eigenvector of A : they are vectors telling us to which component the node belongs.

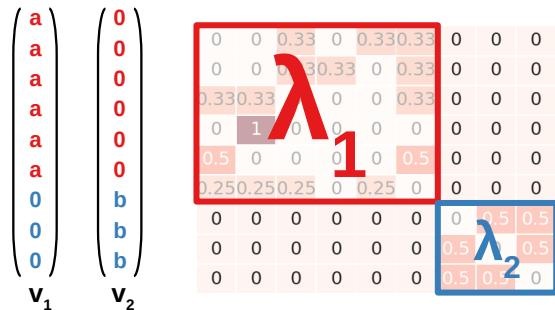


Figure 10.10: If your graph has two components, the eigenvectors associated with the largest two eigenvalues of the stochastic adjacency matrix will tell you to which component the node belongs, by having a non-zero value.

Strong & Weak Components

So far, we saw that a measure of a network's usefulness is *connectedness*. If there is no way to follow the edges of the network from node u to node v , then u has no way to influence – or communicate to – v . However, we dealt only with the case of undirected graphs. What if our edges are not symmetric, but have a direction?

In that case we have two different scenarios. In the first scenario, which we call *strong*, we want to ensure the same ability that the undirected network endowed us: u must be able to contact v , and vice versa. Figure 10.11(a) shows an example of such a component. No matter where we choose to start our path, we can always go back. Therefore, any u can reach any v , and vice versa. Since this strong requirement is satisfied, we call these “strongly connected components” (SCC).

It is not a surprise to reveal that strongly connected components contain cycles. By definition, if you can find a pair of nodes that

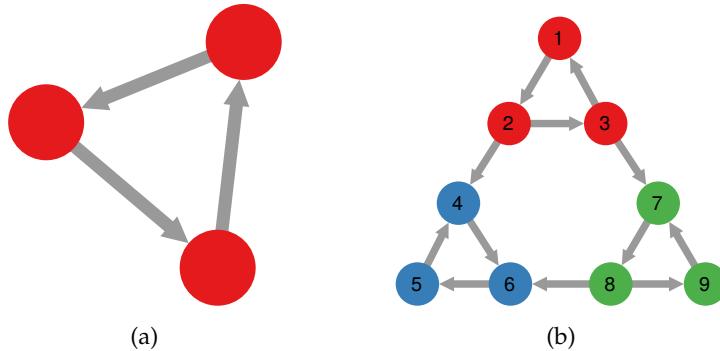


Figure 10.11: (a) A strongly connected component. (b) A network with multiple strongly connected components, coded by different node colors.

you cannot join with a cycle – meaning starting from u and passing through v makes it impossible to go back to u – then those nodes are not part of the same strongly connected component. SCCs are important: if you are playing a message-passing game where messages can only go in one direction, you can always hear back from the players in the same strongly connected component as you.

Popular algorithms to find strongly connected components in a graph are Tarjan's⁵, Nuutila's⁶, and others that exploit parallel computation⁷.

The definition of SCC leaves the door open for some confusion. Even by visually inspecting a network that appears to be connected in a single component, you will find multiple different SCCs – as in Figure 10.11(b). In the figure, there is no path that respects the edge directions and leads from node 1 to node 7 and back. The best one could do is $1 \rightarrow 2 \rightarrow 3 \rightarrow 7 \rightarrow 8 \rightarrow 6 \rightarrow 5 \rightarrow 4$.

However, it *feels* like this network should have one component, because we can see that there are no cuts, no isolated vertices. If we were to ignore edge directions, Figure 10.11(b) would really look like a connected component in an undirected network. This feeling of uneasiness led network scientists to create the concept of “weakly connected components” (WCC). WCCs are exactly what I just wrote: take a directed network, ignore edge directions, and look for connected components in this undirected version of it. Under this definition, Figure 10.11(b) has only one weakly connected component.

In & Out Components

Not all weakly connected components are created equal. In large networks, one can find any sort of weird things. Suppose you are working in an office. The core of the office works on documents together, by passing them to each other multiple times and giving them the core's stamp of approval. This is by definition a strongly connected component.

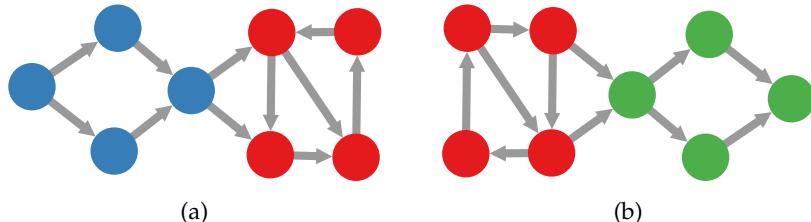
⁵ Robert Tarjan. Depth-first search and linear graph algorithms. *SIAM journal on computing*, 1(2):146–160, 1972

⁶ Esko Nuutila and Eljas Soisalon-Soininen. On finding the strongly connected components in a directed graph. *Inf. Process. Lett.*, 49(1):9–14, 1994

⁷ Sungpack Hong, Nicole C Rodia, and Kunle Olukotun. On fast parallel detection of strongly connected components (scc) in small-world graphs. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, pages 1–11, 2013

But you're not part of the core of the office, you are in a weakly connected component. Your job is simply to receive a document, stamp it, and pass it to the next desk. Since you are in a WCC, you know you're never going to see the same document twice. That would imply that there is a cycle, and thus that you are in a strongly connected component with someone. However, what you see in the document can be radically different. The document might arrive to you with or without the core's stamp of approval. These two scenarios are quite different.

If you are in the first scenario, it means your WCC is positioned "before" the core. Documents pass through it and they are put *in* the core. The flow of information originates from you or from some other member of the weakly connected component, and it is poured *into* the core. This is the scenario in Figure 10.12(a): you are one of the four leftmost nodes. In this paragraph I highlighted the word *in* because we decided to call these special WCCs *in*-components.



If you are in the second scenario, it means your WCC is positioned "after" the core. The core does its magic on the documents, and then *outputs* them into your weakly connected component. The flow of information originates from the core and it is poured *out* to your WCC. This is the scenario in Figure 10.12(b): you are one of the four rightmost nodes. In this paragraph I highlighted the word *out* because we decided to call these special WCCs *out*-components.

Figure 10.12: (a) An in-component (in blue), composed by the four leftmost nodes. (b) A out-component (in green), composed by the four rightmost nodes.

10.5 Summary

1. A walk is a sequence of nodes you can visit by following edges in the network. Its length is the number of edges you use. A path is a walk in which you never visit the same node or edge twice.
2. Cycles are paths which start and end in the same node. Acyclic graphs are graphs without cycles. An undirected acyclic graph is called a tree – a graph with $|V|$ nodes and $|V| - 1$ edges. Otherwise, you can have directed acyclic graphs which are not trees.
3. A directed acyclic graph with $|V|$ nodes and $|V| - 1$ edges is a directed tree. If all nodes in a directed tree have in-degree of one,

except one node with in-degree zero, then that directed tree is also an arborescence.

4. Reciprocal edges in directed networks are edges between two nodes pointing at each other. They allow cycles of length two. The number of reciprocated connections over the number of connected pairs is the reciprocity of the directed network.
5. A connected component is a set of nodes that can all reach each other by following walks on the edges. Real world networks usually have one giant connected component which contains the vast majority of nodes in the network.
6. You can count the number of connected components in a graph by counting the number of eigenvalues equal to one of its stochastic adjacency matrix. The non-zero entries in the corresponding eigenvectors tell you which nodes are in which connected component.
7. In directed networks you can have strong components: components of nodes that can reach each other respecting the direction of the edges. You can also have weak components, which ignore the edge direction.

10.6 Exercises

1. Write the code to perform a random walk of arbitrary length on the network in <http://www.networkatlas.eu/exercises/10/1/data.txt>.
2. Find all cycles in the network in <http://www.networkatlas.eu/exercises/10/2/data.txt>. Note: the network is directed.
3. What is the average reciprocity in the network used in the previous question? How many nodes have a reciprocity of zero?
4. How many weakly and strongly connected component does the network used in the previous question have? Compare their sizes, in number of nodes, with the entire network. Which nodes are in these two components?

11

Random Walks

11.1 Stationary Distribution

Remember the stochastic adjacency matrix from Section 8.2? Figure 11.1 provides a refresher. Here we have the stochastic adjacency matrix of a graph, A , raised to different powers: A^1 (Figure 11.1(a)), A^2 (Figure 11.1(b)), and A^3 (Figure 11.1(c)).

| |
|--|
| 0 0 0 0 0.33 0 0 0.33 0.33 0.3 0.2 0.08 0 0.07 0.1 0 0.07 0.2 0.08 0.09 0.04 0.05 0.2 0.1 0.07 0.2 0.2 |
| 0 0 0 0 0.25 0.25 0.25 0 0.25 0.1 0.3 0.04 0.04 0.05 0.2 0.04 0.09 0.1 0.06 0.09 0.07 0.04 0.1 0.2 0.1 0.1 0.2 |
| 0 0 0 0.33 0 0.33 0 0.33 0 0.08 0.06 0.3 0.06 0 0.2 0.06 0.06 0.1 0.04 0.1 0.08 0.1 0.07 0.2 0.06 0.2 0.1 |
| 0 0 0.5 0 0 0.5 0 0 0 0 0.08 0.08 0.2 0 0.2 0.08 0.2 0.08 0.08 0.09 0.2 0.06 0.04 0.3 0.05 0.07 0.1 |
| 0.33 0.33 0 0 0 0 0 0.33 0.07 0.07 0 0 0.3 0.1 0.08 0.2 0.2 0.2 0.2 0.07 0.02 0.08 0.1 0.04 0.09 0.2 |
| 0 0.17 0.17 0.17 0 0 0.17 0.17 0.17 0.07 0.1 0.1 0.06 0.07 0.3 0.04 0.09 0.08 0.06 0.1 0.1 0.1 0.07 0.2 0.09 0.1 0.2 |
| 0 0.5 0 0 0 0.5 0 0 0 0 0.08 0.08 0.08 0.1 0.1 0.2 0.08 0.2 0.1 0.2 0.08 0.05 0.06 0.3 0.04 0.09 0.1 |
| 0.25 0 0.25 0 0 0.25 0 0 0.25 0.05 0.09 0.04 0.1 0.1 0.04 0.3 0.1 0.1 0.1 0.04 0.06 0.2 0.05 0.08 0.2 |
| 0.2 0.2 0 0 0.2 0.2 0 0.2 0 0.1 0.1 0.08 0.03 0.1 0.1 0.08 0.1 0.3 0.1 0.2 0.06 0.04 0.1 0.2 0.04 0.1 0.1 |

(a) A^1

(b) A^2

(c) A^3

There's one curious thing if we look at the columns of the A^1 and A^3 matrices. In the first case, the minimum is zero and the maximum can be up to 0.5. But in A^3 the minimum is higher than zero, and the maximum is just 0.3. It seems that the values are somehow converging. So what happens if we take A^{30} or – gasp! – A^∞ ?

| | | | | | | | | |
|------|-----|------|------|------|-----|------|-----|-----|
| 0.09 | 0.1 | 0.09 | 0.06 | 0.09 | 0.2 | 0.06 | 0.1 | 0.2 |
| 0.09 | 0.1 | 0.09 | 0.06 | 0.09 | 0.2 | 0.06 | 0.1 | 0.2 |
| 0.09 | 0.1 | 0.09 | 0.06 | 0.09 | 0.2 | 0.06 | 0.1 | 0.2 |
| 0.09 | 0.1 | 0.09 | 0.06 | 0.09 | 0.2 | 0.06 | 0.1 | 0.2 |
| 0.09 | 0.1 | 0.09 | 0.06 | 0.09 | 0.2 | 0.06 | 0.1 | 0.2 |
| 0.09 | 0.1 | 0.09 | 0.06 | 0.09 | 0.2 | 0.06 | 0.1 | 0.2 |
| 0.09 | 0.1 | 0.09 | 0.06 | 0.09 | 0.2 | 0.06 | 0.1 | 0.2 |
| 0.09 | 0.1 | 0.09 | 0.06 | 0.09 | 0.2 | 0.06 | 0.1 | 0.2 |
| 0.09 | 0.1 | 0.09 | 0.06 | 0.09 | 0.2 | 0.06 | 0.1 | 0.2 |

Figure 11.2 shows the result. We see now that the columns are constant vectors. These numbers have a specific meaning. When we calculate A^∞ , what we're doing is basically asking the probability of being in a node after a random walk of infinite length. Since

Figure 11.1: Different powers of the stochastic adjacency matrix of the graph in Figure 8.8(b).

Figure 11.2: The result when raising the stochastic A to the power of 30.

the length is infinite, it does not really matter from which node you originally started. That's why all rows of A^∞ are the same – remember that the row indicates the starting point while the column indicates the ending point.

This row vector – you can pick any of them, since they're all the same – is so important that we give it a name. We call it the “stationary distribution” – or π , for short. π tells us that, if you have a path of infinite length, the probability of ending up on a destination is only dependent on the destination's location and not on your point of origin. In practice, if you apply the transition probability (A) to the stationary distribution (π), you still obtain the stationary distribution: $\pi A = \pi$. Having a high value in the stationary distribution for a node means that you are likely to visit it often with a random walker – by the way, this is almost exactly what PageRank estimates, plus/minus some bells and whistles, see Section 14.4.

Note that it is not necessary to calculate A^∞ to know the stationary distribution. At least for undirected networks, π is quite literally the normalized degree of the nodes: the degree divided by the sum of all degrees ($2|E|$).

But... wait! This stationary distribution formula is oddly familiar: $\pi A = \pi$. Haven't we seen something similar to it? This kind of looks like our eigenvector specification ($Av = \lambda v$, see Section 5.5), with a few odd parts. First, where's the eigenvalue? Well, we can always multiply a vector to 1 and we won't change anything in the equation. So: $\pi A = \pi 1$. This is cool, because we already know that 1 is the largest eigenvalue (λ_1) of a stochastic matrix. Second, the vector π is *on the left*, not *on the right*. Putting these things together: the stationary distribution π is the vector associated with the largest eigenvalue, if multiplied on the left of A . Therefore: π is the leading left eigenvector.

If you're dealing with an undirected graph, there is a relationship between right and left eigenvectors. If you were to transpose the stochastic adjacency matrix, that is making it column-normalized instead of row-normalized, the left and right eigenvectors would swap. In different words: the left eigenvectors of A are exactly the same as the right eigenvectors of A^T . Thus the vector of constant π are the right and left leading eigenvectors of A , and they swap roles in A^T .

What do you do if your graph is not connected? No matter how many powers of A you take, how infinitely long your walks are, some destinations are unreachable from some origins. We end up with two stationary distributions, one for one component, and one for the other. Figure 11.3 shows an example. These two stationary distributions are not directly comparable one with the other. They are

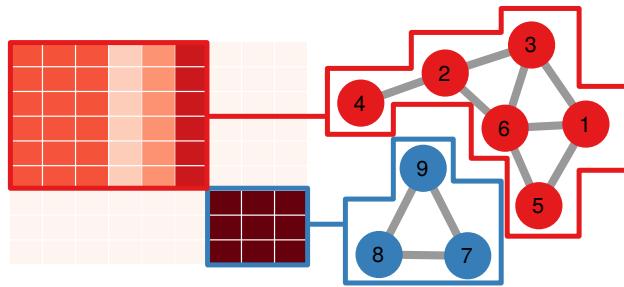


Figure 11.3: The result when calculating the stationary distribution for an unconnected graph.

effectively telling you something about two different networks: one made by the first component, and the other composed by the second one.

This makes it clear why the eigenvector contains zeros for the entries corresponding to the nodes that are not part of the connected component we're looking at. The eigenvector contains the probability of ending up in a node after a random walk of infinite length. The probability of ending up in those nodes via a random walk – no matter how long – is zero, because there is no edge that you can use.

11.2 Non-Backtracking Random Walks

This is a good place to mention that there is an almost infinite number of different matrix representations you can build for a graph. Each of those representations are useful to describe some sort of process on the graph. Since we're in the random walk section, I will mention another useful matrix representing random walks: the non-backtracking matrix. However, be aware that this is only one arbitrary choice about the many possible, you can have: cycle matrices^{1,2}, cut-set matrices³, path and distance matrices, modularity matrices⁴, to name a few.

So what's a non-backtracking matrix? As the name suggests, it's a matrix describing non-backtracking walks. A walk is non-backtracking when we forbid the walker to re-use the same edge twice in a row in its walk. Figure 11.4 shows an example.

If we want to represent such a process with a matrix, we need to build it quite differently from an adjacency matrix. Rather than

¹ Prabhaker Mateti and Narsingh Deo. On algorithms for enumerating all circuits of a graph. *SIAM Journal on Computing*, 5(1):90–99, 1976

² Frank Harary. Graphs and matrices. *SIAM Review*, 9(1):83–90, 1967

³ Jørn Vatn. Finding minimal cut sets in a fault tree. *Reliability Engineering & System Safety*, 36(1):59–62, 1992

⁴ Mark EJ Newman. Modularity and community structure in networks. *Proceedings of the national academy of sciences*, 103(23):8577–8582, 2006b

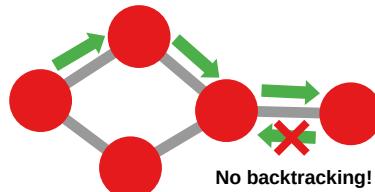


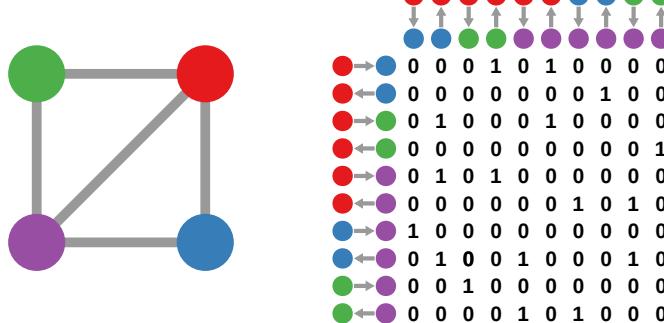
Figure 11.4: A non-backtracking random walk. The green arrows show the state transitions.

having a row and a column per node, we instead have two rows and columns per edge⁵ – or one row/column per edge direction if we have a directed graph, meaning that we treat an undirected graph as a directed one with perfect reciprocity. Each cell contains a one if we can use the edge direction for our non-backtracking walk, zero otherwise. Formally:

$$NB_{uv,vz} = \begin{cases} 1 & \text{if } u \neq z \\ 0 & \text{otherwise.} \end{cases}$$

So you see what's going on here: we can only transition to node z from v only if we got into v via u and u is not the same node as z . If you're a graphical thinker, Figure 11.5 might help you. The non backtracking matrix is not symmetric: if you go from red to blue (first column) you can go from blue to purple (first column, seventh row equals to 1). But if you go from blue to purple (seventh column) you cannot go from red to blue (seventh column, first row equals to 0).

This breaks the symmetry.



As the figure shows, the non backtracking matrix has zero on the block diagonal, because the block diagonal contains all edges to themselves. Thus, a one in the diagonal is exactly a backtracking move: you used the u, v edge to get to v and then you use it again to get to u . Naughty backtracking walker!

On the other hand, if we go to the blue node from the red node, the only legal move is to go to the purple node. If we did the opposite move, from blue to red, we could reach either the green or the purple node. Using these rules, you can figure out each entry in the matrix. Of course, you can have a directed non-backtracking matrix, in which you need to respect the direction of the edge as well.

Non-backtracking matrices are useful for a bunch of applications: fixing eigenvector centrality degeneration⁶ (Section 14.4); helping with community detection⁷ (Part X); describing percolation processes⁸ (Chapter 22); and even helping with counting motifs in

⁵ Ki-ichiro Hashimoto. Zeta functions of finite graphs and representations of p -adic groups. In *Automorphic forms and geometry of arithmetic varieties*, pages 211–280. Elsevier, 1989

Figure 11.5: A non-backtracking matrix.

⁶ Travis Martin, Xiao Zhang, and Mark EJ Newman. Localization and centrality in networks. *Physical review E*, 90(5):052808, 2014

⁷ Florent Krzakala, Cristopher Moore, Elchanan Mossel, Joe Neeman, Allan Sly, Lenka Zdeborová, and Pan Zhang. Spectral redemption in clustering sparse networks. *Proceedings of the National Academy of Sciences*, 110(52):20935–20940, 2013

⁸ Brian Karrer, Mark EJ Newman, and Lenka Zdeborová. Percolation on sparse networks. *Physical review letters*, 113(20):208702, 2014

graphs⁹ (Chapter 41).

11.3 Hitting Time

The stationary distribution allows you to calculate the probability of visiting a node given an infinite length random walk. Another important thing you might be interested in discovering is how long you have to wait before a node gets visited by a random walker. Of course you have an intuition: if it is very likely to visit the node – high value in the stationary distribution – then probably it won't take long before we "hit" that node. But the two quantities, probability and average hitting time, are not the same. Especially since the hitting time of node v depends from the starting point u .

We use $H_{u,v}$ to indicate the expected hitting time of v for a random walk starting in u . How can we calculate $H_{u,v}$? Well, if we want to reach v from u , first we have to go to a neighbor of u . Let's call it z . Once we are in z , it will take us $H_{z,v}$ steps to reach v , by definition. The probability of ending up in z from u is one over u 's degree (k_u) since we picked it at random. Thus the formula expands to:

$$H_{u,v} = 1 + \frac{1}{k_u} \sum_{z \in N_u} H_{z,v}.$$

Applying this formula naively wouldn't lead you very far, as you'll find yourself needing to calculate $H_{u,v}$ in order to find out $H_{u,v}$'s value. There is a way to find $H_{u,v}$ using – what else? – the eigenvectors and eigenvalues of the adjacency matrix¹⁰. The exact mathematical derivation is not for the faint of heart and can be appreciated by the brave readers who will dare to look at this obscene footnote¹¹, and ends with the following formula:

$$H_{u,v} = 2|E| \sum_{n=2}^{|V|} \frac{1}{1 - \lambda_n} \left(\frac{w_{n,v}^2}{k_v} - \frac{w_{n,u} w_{n,v}}{\sqrt{k_u k_v}} \right),$$

where $|V|$ and $|E|$ are the number of nodes and edges. λ and w are eigenvalues and eigenvectors of a special decomposition of A , namely $N = \Delta^{1/2} A \Delta^{1/2}$. $\Delta = D^{-1}$ is the diagonal matrix with the inverse of the degree on the diagonal – much like D is the diagonal matrix with the degree on the diagonal, we met it when calculating the graph Laplacian. λ_n is the n th eigenvalue – sorted in descending order –; $w_{n,u}$ is the u th entry of the n th eigenvector. Finally, k_u is the degree of node u .

The formula looks threatening, but it ought not to be. Let's look at the stupidest example possible, in Figure 11.6. Here we have a simple chain graph. Its degrees k are $(1, 2, 1)$, as the second node has two neighbors and the other nodes have only one. The eigenvalues

⁹ Leo Torres, Pablo Suárez-Serrato, and Tina Eliassi-Rad. Non-backtracking cycles: length spectrum theory and graph mining applications. *Applied Network Science*, 4(1):41, 2019

¹⁰ László Lovász et al. Random walks on graphs: A survey. *Combinatorics, Paul erdos is eighty*, 2(1):1–46, 1993

¹¹ So, here we go. The main formula can be rewritten in matrix form: $H = J + AH - F$, with J being the matrix of ones, and A the stochastic adjacency. What's that F , though? It's a diagonal matrix we have to remove from H because, by definition, the diagonal of H must be zero: the hitting time of the origin from the origin is zero, it is *not* the time it takes for a random walker to go somewhere and coming back, which is what you'd get if you didn't take F out. So did we just make it worse by adding something more we don't know? Actually, we can derive F . Let's rewrite the equation as $F = J + AH - H$. Let's multiply the stationary distribution to both sides: $F\pi = J\pi + H(A - I)\pi$ (I grouped the H terms). Note that $A\pi = \pi$ by definition of a stationary distribution and that $I\pi = \pi$ by definition of an identity matrix. So the whole $H(A - I)\pi$ term disappears, leaving $F\pi = J\pi$. Again by definition of π , a matrix of ones times π equals the scalar one, giving us $F\pi = 1$. So F is a diagonal matrix with $1/\pi_u = 2|E|/k_u$ on its u th diagonal entry. We use D to specify the diagonal matrix with the degree on the diagonal, giving us the equation $H(I - A) = J - 2|E|D^{-1}$. So we can derive H easily now, right? Ahahah. Wrong. $(I - A)$ is singular, thus non-invertible: you can't calculate $(I - A)^{-1}$. So we need to multiply the left and the right side by something that will make $(I - A)$ disappear. That something is the special matrix $Z = (I - A + A^\infty)^{-1}$. This gives us $H = (I - A + A^\infty)^{-1}(J - 2|E|D^{-1})$. To cut a long story short, we can decompose A using its eigenvectors, obtaining the formula in the main text. It took me one day to write this footnote. I'm not paid nearly enough for this.

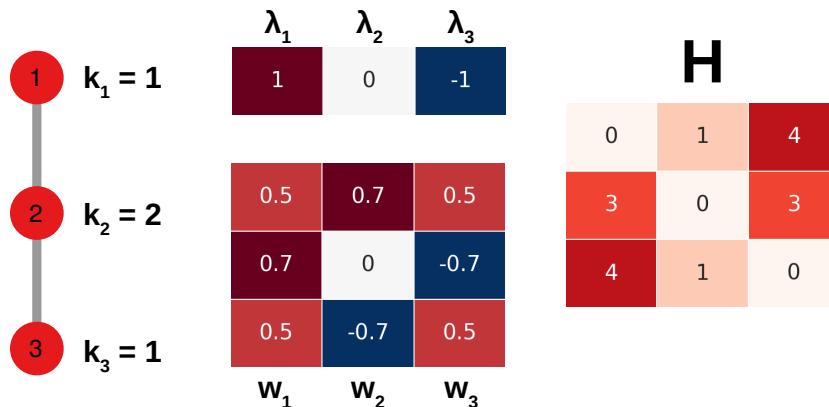


Figure 11.6: The elements needed to calculate the hitting time of a graph. From left to right: the graph and its degree vector, the eigenvalues and eigenvectors of N , the resulting hitting time matrix H .

of N are $(1, 0, -1)$ – we know the largest must be one because N is stochastic. w shows the corresponding eigenvectors. Note how, in $w_1, w_{1,u} = (1/\sqrt{2|E|})\sqrt{k_u}$, which you can derive following what you know about the stationary distribution of A and the way we derived N from A .

If you want to know $H_{1,3}$, you need to do two things. First, for $n = 2$, $\frac{1}{1-\lambda_2} = 1$ and $\left(\frac{0.7^2}{1} - \frac{0.7 \times -0.7}{\sqrt{1}}\right) = 1$. Then, for $n = 3$, $\frac{1}{1-\lambda_3} = 1/2$ and $\left(\frac{0.5^2}{1} - \frac{0.5 \times 0.5}{\sqrt{1}}\right) = 0$. So, for $n = 2$ the part on the right side of the sum evaluates to 1 and for $n = 3$ it evaluates to zero. Thus the total sum is one. Multiplied to $2|E|$ you obtain four.

This is super intuitive. How long does it take to get from node 1 to node 2? Well, node 1 has only one connection and it goes to node 2, so it will always take one step. But to get from node 2 to node 1, you only have a 50% chance of doing it in one step. The other 50% of the times the random walker will go to node 3. It will always come back after another step, and then we'll have another 50% chance to go to node 1. You sum that to infinity, and you get an expected hitting time of three.

From the formula and the example it is easy to see that H is asymmetric, given that one of its parts is dependent on the degree of the destination k_v and not on k_u , the degree of the origin. For this reason, another object of interest is the commute matrix C : the time it takes to go from u to v and back to u . This is C , and it is simply defined as: $C_{u,v} = H_{u,v} + H_{v,u}$, trivially symmetric.

There's one fun connection with the stationary distribution here. We call this the "Random Target Lemma". Suppose you start from node u and you pick destinations v at random. What's the expected hitting time? This is basically averaging $H_{u,v}$ over all possible destinations v . If you do that, you'll find out that the result is:

$$\sum_{v \in V} \pi_v H_{u,v} = \sum_{n=2}^{|V|} \frac{1}{1 - \lambda_n}.$$

Noticing something weird? The right hand side has no trace of u . This means that the average time to hit something doesn't depend on your starting point u , exactly like, in the stationary distribution, the probability of ending your random walk somewhere didn't depend on from where you started.

Note that this is a very short and incomplete treatment of the subject, just to let you know how to calculate hitting and commute times. You really should check out Lovász's paper (footnote 10) for a full treatment of the subject.

11.4 Effective Resistance

The hitting time matrix H is useful specifically when you want to exploit its asymmetry: you want to start from u and know when you will hit v . However, if you're only interested in the symmetric commute time, then there is an easier way to calculate the commute matrix C . This calculation involves the effective resistance matrix Ω – which is one of the awesomest matrices in network science and that's why you should know about it.

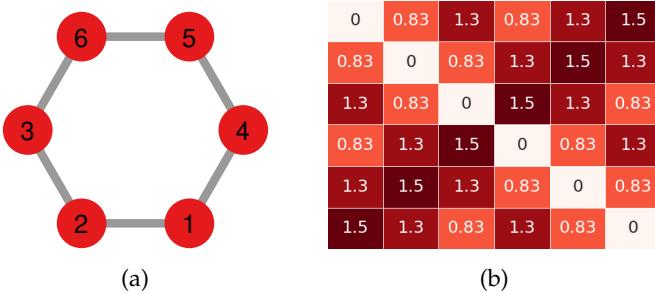


Figure 11.7: (a) A ring graph, (b) its effective resistance matrix.

Mathematically, $C = 2|E|\Omega$, which means that Ω is a sort of normalized commute time. It is the commute time, ignoring the overall size of the graph – which is given by the $2|E|$ factor. You can see an example graph and its effective resistance matrix in Figure 11.7. The original definition of Ω is physical: you assume G is an electrical network and each edge is a resistor of one Ohm. Then the effective resistance between nodes u and v $\Omega_{u,v}$ is the literal electric resistance you'd measure. If you remember, when I introduced the Laplacian (Section 8.4) I said that it was originally used to describe electric circuits, so you might expect it to pop up here. In fact, here's the formula to calculate effective resistance^{12,13}: $\Omega_{u,v} = \Gamma_{u,u} + \Gamma_{v,v} - 2\Gamma_{u,v}$, where:

¹² D Babić, Douglas J Klein, István Lukovits, Sonja Nikolić, and N Trinajstić. Resistance-distance matrix: a computational algorithm and its application. *International Journal of Quantum Chemistry*, 90(1):166–176, 2002

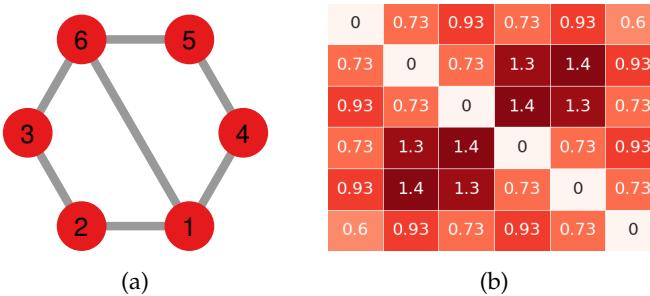
¹³ Wenjun Xiao and Ivan Gutman. Resistance distance and laplacian spectrum. *Theoretical chemistry accounts*, 110:284–289, 2003

$$\Gamma = \left(L + \frac{1}{|V|} \mathbb{1} \right)^{\dagger}.$$

Here, L is the Laplacian, $|V|$ is the number of nodes, and $\mathbb{1}$ is a $|V| \times |V|$ matrix filled with ones. In practice, inside the parentheses we have a matrix that is L plus $1/|V|$ in all its entries. The \dagger symbol means that we want to invert this matrix. If you remember, the Laplacian tells you how electricity flows in the network, so we need to invert it to estimate the resistances. The problem is that the Laplacian is a singular matrix, which means it cannot be inverted.

This is why we use \dagger instead of -1 : rather than inverting we take the Moore-Penrose pseudoinverse^{14,15,16}, which is basically the inverse, but shhh don't tell the Laplacian or it will get mad. To get the Moore-Penrose pseudoinverse, the first step is to perform the singular value decomposition (SVD) of L . SVD is one of the many ways to perform matrix factorization (Section 5.6). In SVD, we want to find the elements for which this equation holds: $Q_1 \Sigma Q_2^T = L$. The important part here is Σ , which is a diagonal matrix containing L 's singular values. We can easily build a Σ^{-1} matrix, containing in its diagonal the reciprocals of L 's singular values. Then $Q_2 \Sigma^{-1} Q_1^T = L^\dagger$ is L 's Moore-Penrose pseudoinverse. It holds that $LL^\dagger L = L$ and that $L^\dagger LL^\dagger = L^\dagger$.

What makes effective resistance so awesome is that it defines a proper analogue to the Euclidean distance in a graph¹⁷. By far the most popular alternative is using shortest path distances – we'll see how to calculate this distance in Chapter 13. However, differently from shortest paths, Ω is a proper metric, which means you can do a bunch of things (most of which we'll see in Chapter 47 when we'll talk about network distances) that would lead to mathematical nonsense if you were to use shortest paths instead.



¹⁴ Eliakim H Moore. On the reciprocal of the general algebraic matrix. *Bulletin of the american mathematical society*, 26: 294–295, 1920

¹⁵ Arne Bjerhammar. Application of calculus of matrices to method of least squares: with special reference to geodetic calculations. (*No Title*), 1951

¹⁶ Roger Penrose. A generalized inverse for matrices. In *Mathematical proceedings of the Cambridge philosophical society*, volume 51, pages 406–413. Cambridge University Press, 1955

¹⁷ Karel Devriendt. Effective resistance is more than distance: Laplacians, simplices and the schur complement. *Linear Algebra and its Applications*, 639: 24–49, 2022

Figure 11.8: (a) A graph, (b) its effective resistance matrix.

Moreover, by being the result of a process of diffusion through the entirety of the graph – current flows as it was presented – Ω is also more resistant to random fluctuations. The removal or introduction of a single edge can radically change the shortest path distance between two nodes, but Ω will change less abruptly. Figure 11.8

shows an example where I added an edge between nodes 1 and 6 to the graph in Figure 11.7. The shortest path distance reduces by a factor of three by adding a single edge, while effective resistance changes from 1.5 to 0.6 – a factor of 2.5.

The difference here is small because this is a tiny toy didactic example, but it can get quite significant. For instance, via a simulation, for a graph with $|V| = 1,000$ and $|E| = 3,000$ I could find nodes connected by a shortest path of length 7. When connecting them, the shortest path decreases by a factor of 7, but their effective resistance went down by a factor of less than 3 (from 1.9 to 0.65). This is important, because network data is noisy and contains errors (Chapter 28) and you don't want a small chance fluctuation to dramatically change your results.

11.5 Mincut Problem

I already said in Section 10.4 that the smallest eigenvector of L isn't so special after all. It plays the very same role as the largest eigenvector of A : it is a vector of constant, telling us to which component the node belongs. The reason why L is interesting lies with the second smallest eigenvector (or the eigenvector associated to the smallest non-zero eigenvalue, if the graph has multiple components).

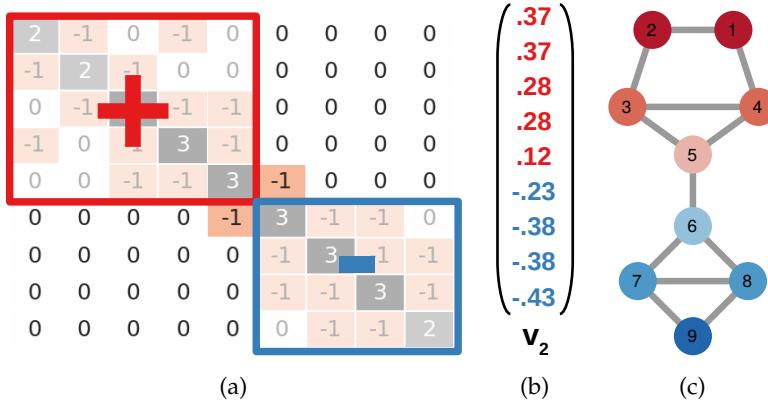


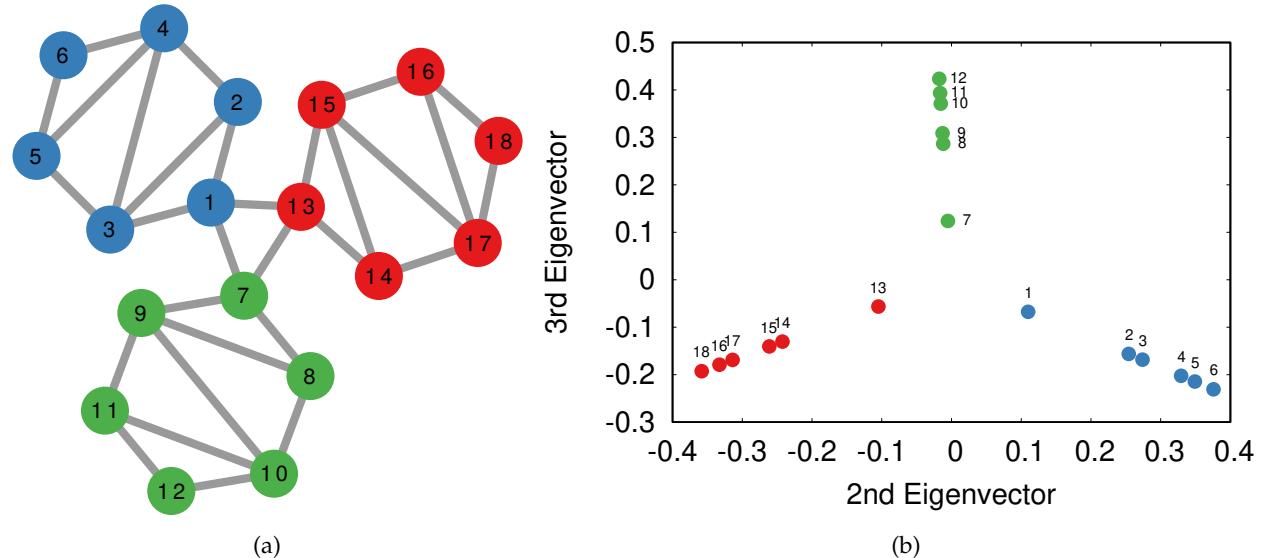
Figure 11.9: (a) The Laplacian matrix of a graph. I show the 2-cut solution for this graph with the red and blue blocks. (b) The second smallest eigenvector of (a). (c) The graph view of (a) – I color the nodes according to the value attached to them in the (b) vector.

One classical problem in graph theory is to find the minimum (or normalized) cut of a graph: how to divide nodes in two disjoint groups such that the number of edges running across groups is minimized. Turns out that the second smallest eigenvector of the Laplacian is a very good approximation to solve this problem¹⁸. How? Consider Figure 11.9. In Figure 11.9(a) I show the Laplacian matrix of a graph. I arranged the rows and columns of the matrix so that the 2-cut solution is evident: by dividing the matrix in two diagonal blocks there is only one edge outside our block structure that needs to be cut.

¹⁸ Miroslav Fiedler. Laplacian of graphs and algebraic connectivity. *Banach Center Publications*, 25(1):57–70, 1989

Now, why did I label the two blocks as “+” and “-”? The reason lies in the magical second smallest eigenvector of the Laplacian – also known as the Fiedler vector –, which is in Figure 11.9(b). We can see that the top entries are all positive (in red) and the bottom are all negative (in blue). This is where L shines: by looking at the sign of the value of a node in its second smallest eigenvector we know in which group the node has to be to solve the 2-cut problem!

Not only that, but the values in Figure 11.9(b) are clearly in descending order. If we look at the graph itself – in Figure 11.9(c) – and we use these values as node colors, we discover that there is much more information than that in the eigenvector. The absolute value tells us how embedded the node is in the group, or how far from the cut it is. Node 5 is right next to it, while node 9 is the farthest away.



Now – you’re thinking – you’re itching to tell me you can use this eigenvector to solve all the k -cut problems, for any k larger than two. But you can’t, because the Fiedler vector is just a simple monodimensional vector. Or can I? True: the second smallest eigenvector cannot solve, by itself, the arbitrary k -cut problem, finding the minimum cuts to divide the graph in k parts. That’s why we have all the other eigenvectors of the Laplacian.

Solving the 3-cut problem involves looking at the eigenvectors in a two dimensional space. I show an example of this in Figure 11.10. Figure 11.10(b) is a 2D representation of the second and third smallest eigenvectors of the Laplacian of the graph in Figure 11.10(a). We can see that there is a clear pattern: each node takes a position in this space on a different axis, depending on the block to which it belongs. Farther nodes on the axis are more embedded in the block,

Figure 11.10: (a) A graph in which the node colors represent the best solution to the 3-cut problem. (b) The eigenspace of the Laplacian of the (a) graph. I plot the second smallest eigenvector in the x-axis, and the third smallest eigenvector on the y-axis. Data point color corresponds to the node color in (a). I label each data point with its corresponding node ID.

while nodes closer to the cuts are nearby the origin $(0,0)$. You can imagine that we could solve the 4-cut problem looking at a 3D space, and the k -cut problem looking at a $(k - 1)$ D space. I can't show it right now because, although it's truly remarkable, the margin of my Möbius paper is too small to contain it.

Of course, at the practical level, real world networks are not amenable to these simple solutions. Most of the times, the best way to solve the 2-cut problem is to put in one group a node with degree equal to one and put all other nodes of the network in the other group. If you want to find non-trivial k -cuts of the network that are meaningful for humans... well... you have to do community discovery (and jump to Part X).

11.6 Random Walks and Consensus

One thing that might be left in your head after reading the previous section is: why? Why do the eigenvectors of the Laplacian help with the mincut problem? What's the mechanism? To sketch an answer for this question we need to look at what we call "consensus dynamics". This is a subclass of studying the diffusion of something (a disease, word-of-mouth, etc) on a network – which we'll see more in depth in Part VI. This section is sketched from a paper¹⁹ that you should read to have a more in-depth explanation of the dynamics at hand. Consensus dynamics were originally modeled this way by DeGroot²⁰.

In this section I'm going to use the stochastic adjacency matrix of the graph, but what I'm saying also holds for the Laplacian. The difference between the two – as I also mention in Section 34.3 – is that the stochastic adjacency matrix describes the discrete diffusion over a network. In other words, you have a clock ticking and nothing happens between one tick of the clock and the other. The Laplacian, instead, describes continuous diffusion: time flows without ticks in a clock, and you can always ask yourself what happens between two observations. Besides this difference, the two approaches could be considered equivalent for the level of the explanation in this section.

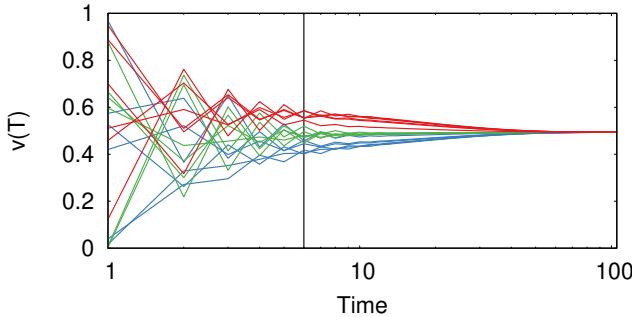
How does the stochastic adjacency help us in studying consensus over a network? Let's suppose that each node starts with an opinion, which is simply a number between 0 and 1. We can describe the status of a network with a vector x of $|V|$ entries, each corresponding to the opinion of each node. One valid operation we could do is multiplying x with A , the stochastic adjacency matrix, since A is a $|V| \times |V|$ matrix.

What does this operation mean? Mathematically, from Section 5.2, the result is a vector x' of length $|V|$ defined as $x'_v = \sum_{u=1}^{|V|} x_u A_{uv}$. In

¹⁹ Michael T Schaub, Jean-Charles Delvenne, Renaud Lambiotte, and Mauricio Barahona. Structured networks and coarse-grained descriptions: A dynamical perspective. *Advances in Network Clustering and Blockmodeling*, pages 333–361, 2019

²⁰ Morris H DeGroot. Reaching a consensus. *Journal of the American Statistical Association*, 69(345):118–121, 1974

practice, the formula tells you that node v is updating its opinion by averaging the opinion of its neighbors. Non-neighbors do not contribute anything because $A_{uv} = 0$, and this is an average because we know that the rows of the adjacency matrix sum to 1 – thus each x_u is weighted equally and x'_v will still be between 0 and 1.



We can reach a consensus by multiplying x with A an infinite number of times. If you do that, x will converge to a vector of constant – which is the average of the initial x , in this case 0.5 since values were extracted uniformly at random between 0 and 1. Figure 11.11 shows an example process on the network from Figure 11.10. Each node starts with a uniform random value and the line tracks this value over time.

Now the connection with eigenvectors: from Section 11.1 you remember that the vector of constant is the leading eigenvector of A . This operation showed you why: if a network has separate connected components, it cannot reach a unique consensus; every connected component will reach its own consensus independently because there's no exchange of information across components.

The second eigenvector, instead, tells you *how quickly* the nodes will reach the consensus. In the figure, the line color tells you the community of the node. You might notice that nodes bundle up with their community mates before reaching the network's final consensus. This is described by the inverse of the second eigenvalue of the Laplacian. For that network, it is $1/\lambda_2 \sim 5.13$. This tells you that the nodes are expected to converge to their community's opinion between step 5 and 6, which is the step highlighted in Figure 11.11. You might notice that one blue and one red node don't seem to converge to their community, but that's because they are nodes 1 and 13 and, as you can see from Figure 11.10, they are in between communities, i.e. they are close to the cut.

So the reason why the Fiedler vector allows you to solve the mincut is because it tells you how much it will take for the node to converge to the consensus. Nodes farther from the cut will take

Figure 11.11: The value of x (y-axis) for each node in the graph from Figure 11.10 over time (x-axis). At each time step, I update x 's values by multiplying them to A . The line color tells you to which community the node belongs. The black vertical line highlights step 6.

longer time. Moreover, you can use the sign to know on which side of the cut you are, because the nodes will first tend to converge to the value of their own community, which is the opposite of the value in the other community.

11.7 Summary

1. Raising the stochastic adjacency matrix to high powers will make it converge to the stationary distribution, which is the probability of ending in a node in the network after an infinite length random walk. This is also the leading left eigenvector, or the normalized degree.
2. A non-backtracking matrix is a matrix describing a non-backtracking random walk, with a row/column per edge telling you whether you can move from one edge to another. It has zero on its main diagonal.
3. The hitting time is the number of expected steps you need to take in a random walk to reach one node starting from another. It is related to a special eigenvector decomposition of the adjacency matrix.
4. The commute time of u and v is the hitting time from u to v plus the hitting time from v to u . It is related to the effective resistance, which can be calculated by inverting the Laplacian. The effective resistance is a proper metric on a graph, less affected by random edge fluctuations than the shortest path distance.
5. The normalized cut problem aims to find the way to partition the network in n balanced parts such that we “cut” the minimum number of edges (the ones flowing from one group to another). You can approximate the solution by looking at the $n - 1$ smallest eigenvectors of the Laplacian (skipping the first).
6. This is because the eigenvectors of the Laplacian tell you when a node will reach a consensus, and to which intermediate value it will converge before doing so. Nodes on the same side of a cut will converge to the same intermediate value.

11.8 Exercises

1. Calculate the stationary distribution of the network at <http://www.networkatlas.eu/exercises/11/1/data.txt> in three ways: by raising the stochastic adjacency to a high power, by looking at the leading left eigenvector, and by normalizing the degree. Verify that they are all equivalent.

2. Calculate the non-backtracking matrix of the network used for the previous question. (The network is undirected)
3. Calculate the hitting time matrix of the network at <http://www.networkatlas.eu/exercises/11/3/data.txt>. Note: for various reasons, a naive implementation in python using numpy and scipy might lead to the wrong result. I would advise to try and do this in Octave (or Matlab).
4. Calculate the effective resistance matrix of the network at <http://www.networkatlas.eu/exercises/11/3/data.txt> and prove it is equal to the commute time divided by $2|E|$. Note: differently from above, the effective resistance matrix can be calculated in python without an issue. But the second part of the exercise might fail if not done in Octave (or Matlab).
5. Draw the spectral plot of the network at <http://www.networkatlas.eu/exercises/11/5/data.txt>, showing the relationship between the second and third eigenvectors of its Laplacian. Can you find clusters?

12

Density

12.1 Density & Real Networks

In Chapter 9, we saw that there is a quick way to get a sense of how well connected the nodes of your network are. You can calculate their average degree, that is to say the average number of edges each node uses to connect to its neighbors. However, in the same chapter, we also saw that the degree distribution is usually extremely broad, which makes the average degree an incomplete information. Moreover, depending on the size of your network, the same value of average degree could mean different things. A network with three nodes and average degree equal to two is fully connected. A network with the same average degree, but ten thousand nodes is quite sparse. See Figure 12.1 for a graphical example.

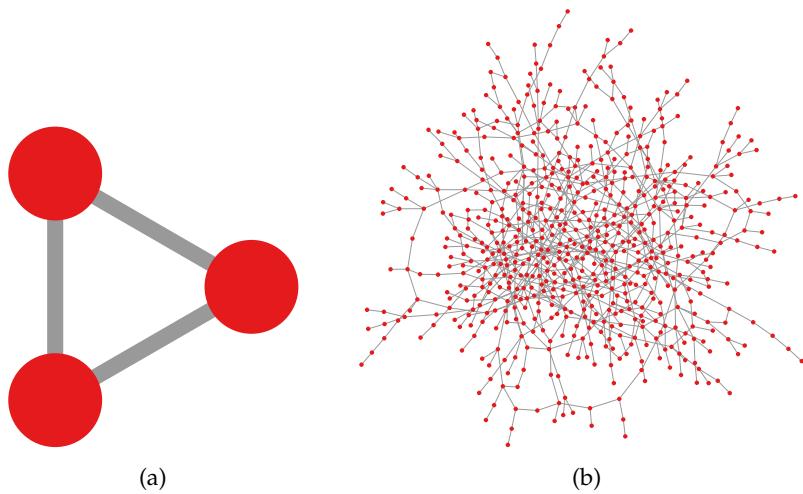


Figure 12.1: (a) A network with three nodes and average degree equal to two. (b) A network with around 650 nodes and average degree equal to two.

To quantify the difference between the two cases, network scientists defined the concept of network density. Informally, this is the probability that a random node pair is connected. Or, the number of edges in a network over the total possible number of edges that can exist given the number of nodes. We can estimate this latter quantity

quite easily – from now on, I'll just assume the network is undirected, unipartite and monolayer, for simplicity.

Here's the problem, rephrased as a question: how many edges do we need to connect $|V|$ nodes? Well, let's start by connecting one node, v , to every other node. We will need $|V| - 1$ edges – we're banning self loops. Now we take a second node, u . We need to connect it to the other nodes in V minus itself – seriously, no self loops! – and v , because we already added the (u, v) edge at the previous step. So we add $|V| - 2$ edges. If you go on and perform the sum for all nodes in V , you'll obtain that the number of possible edges connecting $|V|$ nodes is $|V|(|V| - 1)/2$. In other words, you need $|V| - 1$ edges to connect a node in V with all the other nodes, and you divide by two because each edge connects two nodes. In fact, the number of possible edges in a directed network is simply $|V|(|V| - 1)$, because you need an edge for each direction, as $(u, v) \neq (v, u)$.

Now we can tell the difference between the networks in Figures 12.1(a) and 12.1(b). The first one has three nodes. We would expect $3 * 2 / 2 = 3$ edges, and that's exactly what we have. Its density is the maximum possible: 100%. The network on the right, instead, contains 650 nodes. Since the average degree of the network is also two, we know it also contains 650 edges. This is a far cry from the $650 * 649 / 2 = 210,925$ we'd require. Its density is just 0.31%, more than three hundred times lower than the example on the left!

With all this talk about real world networks having low degree, we should expect them to be quite sparse. They are, in fact, even sparser than you think. A few examples – the numbers are a bit dated, they refer to the moment in which these networks were studied in a paper¹.

The network connecting the routers forming the backbone of the Internet? It contains $|V| = 192,244$ nodes. So the possible number of edges is $|V|(|V| - 1)/2 = 18,478,781,646$. How many does it have, really? 609,066, which is just 0.003% of the maximum. How about the power grid? The classical studied structure has 4,941 nodes, which means – potentially – 12,204,270 edges. Yet, it only contains 6,594 of them, or 0.05%. Well, compared to the Internet that's quite the density! Final example: scientific paper citations. A dataset from arXiv contains 449,673 papers. The theoretical maximum number of citations is 101,102,678,628. Physicists, however, are quite stingy: they only made 4,689,479 citations, or 0.004% of the theoretical maximum.

You might have spotted a pattern there. The density of a network seems to go down as you increase the number of nodes. While not an ironclad rule, you might be onto something. The problem is that the

¹ Albert-László Barabási et al. *Network science*. Cambridge university press, 2016

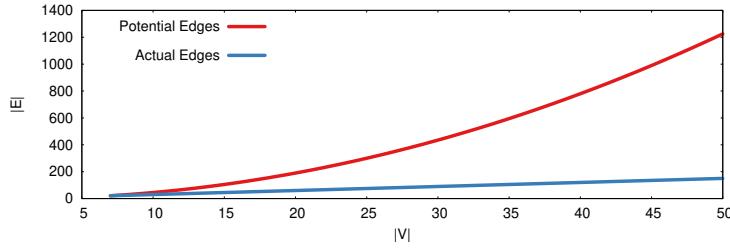


Figure 12.2: The red line shows the number of possible edges ($(|V|(|V| - 1)/2)$) in a network with $|V|$ nodes (x axis). The blue line shows the number of actual edges, assuming the average degree being $\bar{k} = 3$.

denominator of the density formula grows quadratically. Each v you add to V allows $|V|(|V| - 1)$ to grow pretty rapidly. Figure 12.2 shows that as a red line. The numerator, instead, grows practically *linearly* – the blue line in Figure 12.2. Each added node will bring only few edges – three in Figure 12.2 –, as we know that the average degree of real world networks is low.

12.2 Clustering Coefficient

Global Clustering

Density doesn't solve all ambiguities you had in the case of the average degree. Two networks can have the same density and the same number of nodes, but end up looking quite different from each other. That is why the ever industrious network scientists created yet another measure to distinguish between different cases: the clustering coefficient.

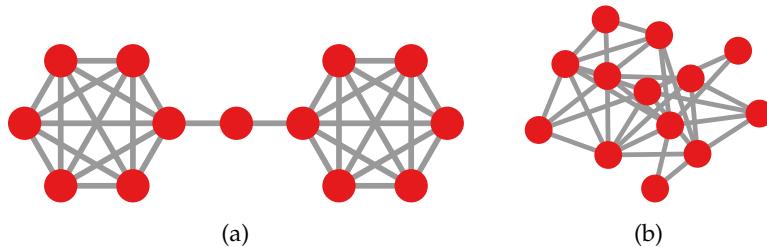


Figure 12.3: Both networks have 13 nodes and 32 edges. However, their topologies are different: in the example to the left the edges are more "clustered" together.

Consider Figure 12.3: it contains two networks with the same number of nodes and the same number of edges – thus the same density. However, they look quite different. There's more a sense of "order" in Figure 12.3(a). That is because, in Figure 12.3(a), the edges are more "clustered" together. That is what the clustering coefficient aims at estimating quantitatively.

I can sum up the intuition behind the clustering coefficient as the old adage "The friend of my friend is my friend". If you have two friends it's overwhelmingly likely that they know each other, because they have something in common: you. You might invite them – even by accident – to the same event. This sort of dynamics

in network is called “triadic closure”. A set of three connected nodes is a triad – Figure 12.4(a). If one member of the triad is connected to the other two, more often than not the triad will “close”, meaning that the other two nodes will connect to each other to form a triangle. “Triangle” is the name we give to a specific network pattern: three nodes that are all connected to each other, and I show one in Figure 12.4(b).

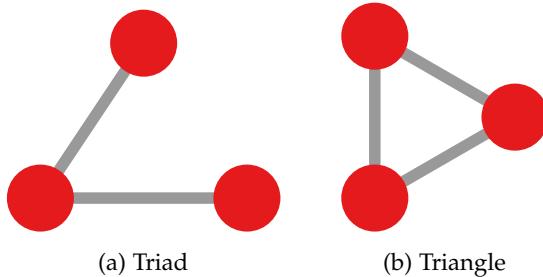


Figure 12.4: The two possible connected network patterns involving three nodes.

A note for computer scientists: do not confuse the clustering coefficient with the operation you know as “clustering”: the process of grouping similar rows of a matrix. The latter is a process that in network science is called community detection – or discovery – and will be the subject of Part X. The clustering coefficient is simply a number you return that describes quantitatively how “clustered” a network looks. A way to dispel the confusion is to use the term “transitivity”. This takes inspiration from the transitive property: if u is connected to v and v is connected to z , then u is connected to z too.

To sum up, the clustering coefficient answers the question: how often does a triad close down into a triangle? What’s the likelihood that the “friend of my friend is my friend” rule holds in the network? To answer this question we have to count the number of triads and the number of triangles in the network. Then the global clustering coefficient (CC) is simply²: $CC = 3 \times \#Triangles / \#Triads$. Why do we have to multiply the number of triangles by three?

Consider Figure 12.5(a). I highlighted a triangle in there. From the perspective of the node highlighted in blue. The triad is closed by the edge highlighted in blue. The same holds for the nodes highlighted in green and purple. That single triangle is closing three triads, that is the reason why we multiply the number of triangles in the network by three.

Counting triads is a bit more confusing, but in the end it’s going to be easy to remember, because it connects with something you already know. I provide a representation of the process in Figure 12.5(b). In there, I count the number of triads centered on node v , meaning that we only count the triads that have v connected to both of its members. This makes it easier, because we only have to look

² Paul W Holland and Samuel Leinhardt. Transitivity in structural models of small groups. *Comparative group studies*, 2(2):107–124, 1971

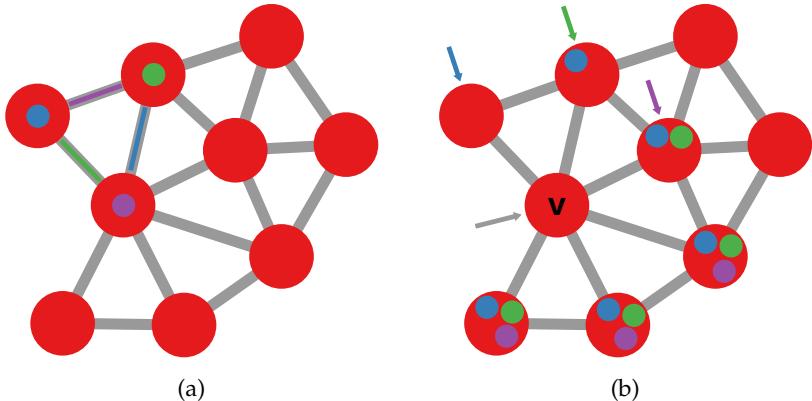


Figure 12.5: (a) Counting the number of triads closed by a triangle. From each node's perspective, a different triad is closed by the same triangle. (b) Counting the number of triads in the network.

at v 's neighbors. I start by selecting its first neighbor, with the blue arrow. How many triads do v and the blue neighbor generate? Well, one for each of the remaining neighbors of v , so I add a blue dot to each of the neighbors. When I move to the neighbor highlighted by the green arrow, I perform the same operation adding the green dot. I don't have to add one to the neighbor with the blue arrow, because I already counted that triad.

Sounds familiar? That is because this is the very same process you apply when you have to count the number of possible edges of a graph. The number of triads centered on node v is nothing more than the number of possible edges among k_v nodes, with k_v being v 's degree. So, if we want to know the number of triads in a graph, we simply need to add $k_v(k_v - 1)/2$ for every v in the graph.

Note that, as you might expect, the clustering coefficient takes different values for weighted³/⁴ and directed⁵ graphs.

I primed you to expect that many statistical properties can be derived via matrix operations. This is true also for the clustering coefficient. It is done via the powers of the binary adjacency matrix – see Section 10.1. Triangles are closed paths of length 3, while triads are paths of length 2. The number of closed walks of length 3 centered on u is A_{uu}^3 , while the number of walks of length 2 passing through u is A_{uv}^2 , with $u \neq v$, which results in the formula:

$$CC = \frac{\sum_u A_{uu}^3}{\sum_{u \neq v} A_{uv}^2}.$$

So, let's calculate the global clustering coefficient for the graph in Figure 12.6(a). We know how many triads there are in the graph. How many triangles are there? Here I made my life easier, because it is rather trivial to count the number of triangles in a planar graph – a graph you can draw on a 2D plane without intersecting any edges. There are eight triangles and 48 triads in the network. Thus the

³ Jukka-Pekka Onnela, Jari Saramäki, János Kertész, and Kimmo Kaski.

Intensity and coherence of motifs in weighted complex networks. *Physical Review E*, 71(6):065103, 2005

⁴ Jari Saramäki, Mikko Kivelä, Jukka-Pekka Onnela, Kimmo Kaski, and Janos Kertesz. Generalizations of the clustering coefficient to weighted complex networks. *Physical Review E*, 75(2):027105, 2007

⁵ Giorgio Fagiolo. Clustering in complex directed networks. *Physical Review E*, 76(2):026107, 2007

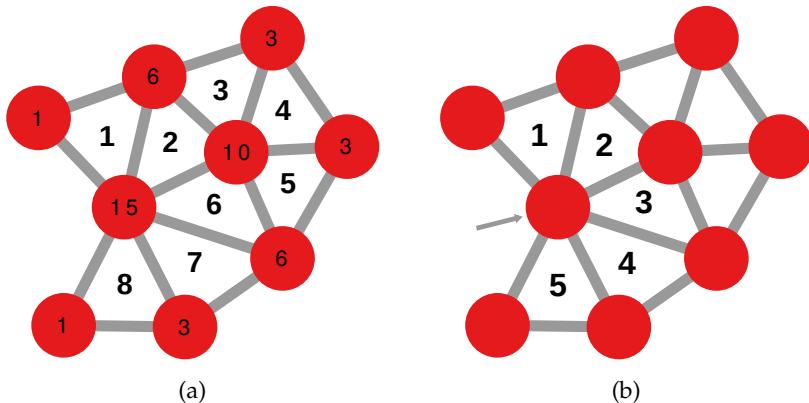


Figure 12.6: (a) Estimating the global clustering coefficient of a graph. Each node is labeled with the number of triads centered on it, and the numbers among the edges count the triangles. (b) Estimating the local clustering coefficient of a node, counting the triangles to which it belongs.

global clustering coefficient of the network is $CC = 3 \times 8 / 48 = 0.5$. Half of the triads in the network close to form a triangle.

Local & Average Clustering

The fact that the previous subsection was called “global” clustering should tip you off about the existence of a “local” clustering coefficient. Its definition is rather similar, but it is focused on a single node. It is the number of triangles to which the node belongs over the number of triads centered on it. We do not multiply by three the numerator, because we’re focusing exclusively on the triads of v , that can then be closed only in one way.

Looking at Figure 12.6(b), let’s try to calculate the local clustering coefficient of the node highlighted by the arrow. Again, we use our planar graph to have an easy time counting triangles: there are five that include node v . We already counted the number of triples before: it was 15. Thus, the local clustering coefficient of v is $CC_v = 5/15 = 0.3$.

And, hopefully, that’s it. Oh, who am I kidding. Of course there’s more. Once you have a local clustering coefficient, one might get curious and desire to calculate the average local clustering coefficient of the network. This is simply $CC_{avg} = \frac{1}{|V|} \sum_{v \in V} CC_v$. You would hope that $CC_{avg} = CC$. No such luck. For the network in Figure 12.7, we know that $CC = 0.5$. Unfortunately, if you average the CC_v values I used to label each node, you’ll find out that $CC_{avg} = 0.648$. So you have to remember that the global and the average clustering coefficient are two different things, and not to report one as the other.

I closed the previous section with a mantra – “real world networks are sparse” –, so I want to do it again. However, there’s a surprise here. If average degrees are low and networks are sparse, wouldn’t you expect real world networks to have a low clustering too? Instead, the opposite holds: real world networks are clustered. The power

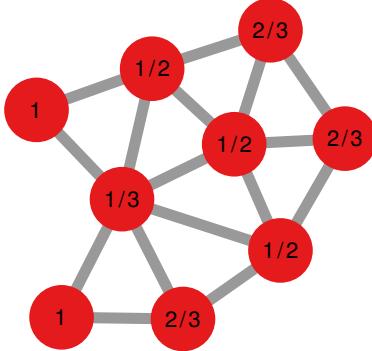


Figure 12.7: I label each node in the network with its local clustering coefficient CC_v value.

grid example I used before? It has a CC of 0.1032, which is 150 times higher than you would expect if its edges were distributed randomly. The scientific paper citations? It has $CC = 0.318$, more than 200 times higher than expected.

This means that these systems might have few connections per node, but these connections tend to be clustered in the same neighborhood. Nodes tend to close triangles. This is especially true for social systems. In fact, the protein-protein network I used in Chapter 9 has a clustering of 0.0236, which is still higher than expected. But in this biological case, we only have a factor of 16, a far cry from the factor of 200 in the social paper citation system.

Structural Holes & Redundancy

High global or local clustering coefficients in social networks imply that we expect triangles to close. It follows that, when they don't, we consider that weird. Every time there are common neighbors of a node v that do not connect with each other we call that a structural hole⁶. Figure 12.8 shows a network with several structural holes. In practice, if $CC_v < 1$ then we know there are structural holes around v . A high CC_v means there are a lot of ways to get around node v , so information can spread freely even if v were to disappear. A low CC_v means that v is fundamental to spread information, since its common neighbors are hardly connected to each other.

Redundancy⁷ is a measure that is relatively similar to the local

⁶ Ronald S Burt. Structural holes: The social structure of competition. *University of Illinois at Urbana-Champaign's Academy for Entrepreneurial Leadership Historical Research Reference in Entrepreneurship*, 1992

⁷ Stephen P Borgatti. Structural holes: Unpacking burt's redundancy measures. *Connections*, 20(1):35–38, 1997

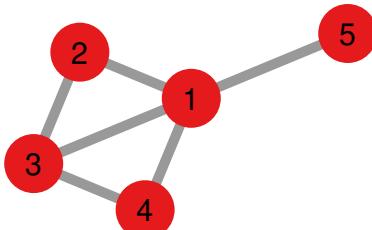


Figure 12.8: A network with several structural holes, for instance between nodes: 2 and 4, 2 and 5, and 4 and 5.

clustering coefficient and can measure the presence of structural holes around a node. The redundancy of node v is the average number of edges from a neighbor of v to the other neighbors of v . Considering again Figure 12.8, one can calculate the redundancy of node 1. It has 4 neighbors, so that's the denominator. Nodes 2 and 4 are connected with only one of node 1's neighbors. Node 3 is connected with two neighbors of node 1. Finally, node 5's degree with node 1's neighbors is zero. Putting everything together: $(1 + 1 + 2 + 0)/4 = 1$.

De facto, local clustering is a normalized redundancy. If R_v is the redundancy, then $CC_v = R_v/(k_v - 1)$ – redundancy can be at most $k_v - 1$ because a neighbor of v can only connect to that many other neighbors of v , it cannot connect with itself, hence the minus one.

12.3 Cliques

When it comes to clustering and density, you cannot do any better than having all possible edges among the nodes in your network. A network will contain many subsets of nodes for which this is true: you pick k nodes in the network and all possible connections among them are present. This happens often in social networks: these complete graphs – as we call them – represent tightly knit groups of friends. They also get a special name: **cliques**.

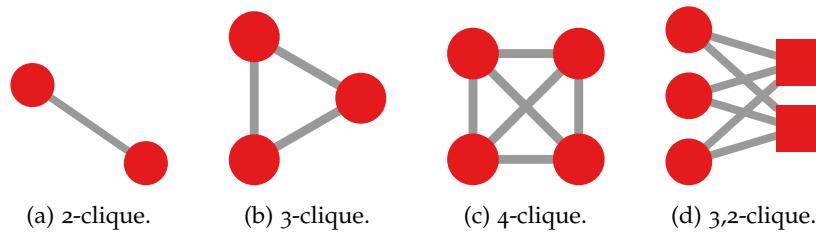


Figure 12.9: The clique zoo.

Formally, the definition of a clique is a subgraph of k nodes and $k(k - 1)/2$ edges – assuming we're in the usual case of undirected unipartite networks. The most general way to refer to a clique of size k is by calling it a k -clique. So an edge, which is a clique of size two, is a 2-clique (Figure 12.9(a)). A clique with three nodes is a 3-clique (Figure 12.9(b)), with four nodes we have a 4-clique (Figure 12.9(c)) and, hopefully, you can generalize from that. A few cliques get special names too, given to their popularity. We already named the 3-clique in the previous section as a triangle.

The case of bipartite networks is a peculiar one worth mentioning. As we know from Section 7.1, in bipartite graphs we're not allowed to connect nodes of the same type. So, when we define a clique as a subset of nodes where “all possible connections among them are present”, we mean something radically different in a bipartite network. Here, we simply mean that all nodes of V_1 type are connected

to all nodes of V_2 type. So we call this structure a **biclique**. Figure 12.9(d) shows an example of biclique made of three nodes of type 1 and two nodes of type 2. We need to modify the way to refer to specific instances of a biclique: from k -clique to n, m -clique. So the one in Figure 12.9(d) is a $3,2$ -clique.

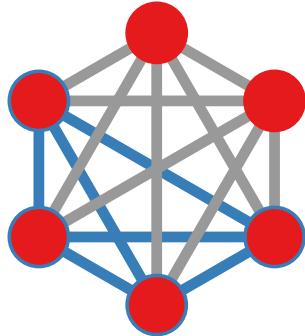


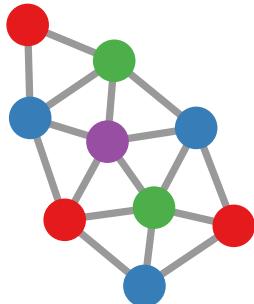
Figure 12.10: An example of maximal 6-clique containing a non-maximal 4-clique, highlighted with the blue outline.

It follows from the definition of a clique that any k -clique will contain many $(k - 1)$ -cliques, down to 2-cliques. If you have all possible edges between six nodes, you can pick any five (four, three, ...) of those six nodes and they're all connected to each other – by definition. Figure 12.10 provides a depiction of this reasoning. We want to distinguish between cliques that are contained in other cliques, and cliques that aren't. We call the latter **maximal cliques**: the set of nodes to which you cannot add any other node and still make it a clique.

12.4 Independent Sets

Just like matter has anti-matter, also cliques have anti-cliques. By definition, a clique is a set of nodes all connected to each other. The anti-clique, which we call “independent set” is a set of nodes none of which are connected to each other⁸. Figure 12.11 shows a possible subdivision of a graph into independent sets.

Note that, in Figure 12.11, I force each node to have a color, i.e. to



⁸ Chris Godsil and Gordon F Royle. *Algebraic graph theory*, volume 207. Springer Science & Business Media, 2013

Figure 12.11: A graph with several independent sets, represented by the color of the nodes.

belong to at least an independent set. I could find a larger independent set, for instance the purple node could make an independent set with the two red nodes it isn't connected to – since they are not connected to each other. The task of finding an independent set coverage of a graph is called graph coloring and it's a classical graph theory problem⁹. Solving graph coloring tells you, for instance, how many colors you need to use for your map such that no two neighboring countries share the same hue – in this case you represent countries as nodes and connect two countries if they share a land border.

When it comes to independent sets, you should not confuse the *maximal* independent set with the *maximum* independent set. A maximal independent set, just like a maximal clique, is an independent set to which you cannot add any node and still make it an independent set. The green set in Figure 12.11 is maximal because the two green nodes are connected to all other nodes in the graph.

On the other hand, the maximum independent set is simply the largest possible independent set you can have in your network. In Figure 12.11, the red set is the maximum independent set, or at least one of the many possible independent sets of size 3. Finding the largest possible independent set is an interesting problem, because it tells you something about the connectivity of the graph. It also has applications in graph mining – see Section 41.5.

12.5 Summary

1. We define a network's density as the number of its edges divided by the total possible amount of edges, which is a different number depending whether your network is directed or not.
2. Real world networks tend to be sparse, meaning that the density is usually lower than a few percentage points.
3. A way to estimate a local density by looking at the neighborhood of a node is the clustering coefficient: the number of common neighbors around node u connected to each other. There are global, local, and average versions of the clustering coefficient, sometimes known as transitivity.
4. Differently from density, many real world networks have very high clustering.
5. A (sub)graph with density equal to one is a clique: a set of nodes all connected to each other. Bipartite networks have bicliques.
6. The opposite of a clique is an independent set: a group of nodes none of which connects to any other member of the group.

⁹ Tommy R Jensen and Bjarne Toft.
Graph coloring problems, volume 39. John Wiley & Sons, 2011

12.6 Exercises

1. Calculate the density of hypothetical undirected networks with the following statistics: $|V| = 26, |E| = 180; |V| = 44, |E| = 221; |V| = 8, |E| = 201$. Which of these networks is an impossible topology (unless we allow it to be a multigraph)?
2. Calculate the density of hypothetical directed networks with the following statistics: $|V| = 15, |E| = 380; |V| = 77, |E| = 391; |V| = 101, |E| = 566$. Which of these networks is an impossible topology (unless we allow it to be a multigraph)?
3. Calculate the global, average and local clustering coefficient for the network in <http://www.networkatlas.eu/exercises/12/3/data.txt>.
4. What is the size in number of nodes of the largest maximal clique of the network used in the previous question? Which nodes are part of it?
5. What is the size in number of nodes of the largest independent set of the network used in the previous question? (Approximate answers are acceptable) Which nodes are part of it?

Part IV

Centrality

13

Shortest Paths

The degree (Chapter 9) is the most direct measure of importance of a node in a network. The more connections a node has, the more important it is. However, there are alternative ways to estimate the importance of a node. Sometimes, it doesn't matter how many connections you have, but how many people someone can reach by passing through you. Normally, the two are correlated – more connections mean more possibilities – but that's not always the case. We explore these differences in this part of the book.

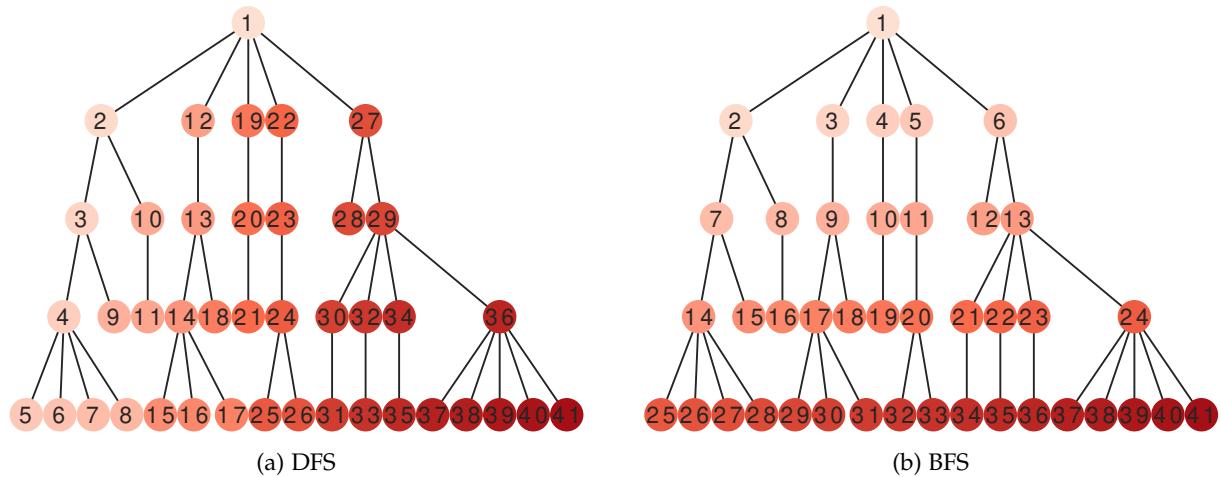
Before we start ranking nodes in Chapter 14 we need to lay down some groundwork. A significant chunk of measures of node importance are based on the concept of shortest paths, which is what we're exploring in this chapter. We start by defining how to explore a graph and we build up from there.

13.1 Graph Exploration

When we first encounter a graph, how do we know its topology and properties? Humans can “see” and parse simple graphs, but how does a computer do it? If we start from a node, how do we access information about its connections, its neighbors, and the connections among them? We need to perform a graph exploration – or graph traversal. There are two main ways to do it: Depth First Search (DFS) and Breadth First Search (BFS).

Depth & Breadth First

In Depth First Search (DFS), you start by picking a root node. Then you put its neighbors in a Last-In-First-Out queue. You pick the last neighbor you added (you “pop” the queue) and you perform the same operation: you add its neighbors to the queue, making sure you don't add nodes you already explored. You continue until you have explored all nodes in the graph. Figure 13.1(a) provides an example,



showing the exploration order. Since you’re using a Last-In-First-Out queue, the very first neighbor of the root node will be the last node to be explored – unless you encounter it again as the neighbor of some other node down the exploration tree, of course.

Breadth First Search (BFS) is practically speaking the exact same algorithm as DFS, with a tiny change. Rather than putting the neighbors of the root node in a Last-In-First-Out queue, you put them into a First-In-First-Out queue. This changes fundamentally the way you explore the graph: you explore all neighbors of the root node before passing to the first neighbor of the first neighbor of the root node. Figure 13.1(b) provides an example, showing the exploration order.

DFS tends to make a gradient over followed paths until it backtracks because it explored the entire neighborhood – in the example from Figure 13.1(a) it backtracks, for instance, from the eighth explored node back to the third explored node. BFS tends to make a gradient from the origin node to the farthest nodes in the network.

Random Node/Edge Access

In day to day computing, you might find yourself exploring the graph in two other ways. These are dependent on the way we store graphs on a computer’s hard disk. We can call these two methods random edge access and random node access.

Random edge access is when you read the file containing your graph one line at a time, and each line contains an edge. We call this type of graph storing format an “edge list”, because it’s a list of one edge per line. In this case, you may or may not have sorted the edges in a particular way, but the baseline assumption is that they’re in a random order. See Figure 13.2(a) for an example.

Random node access is the same, but the file records, in each line, the complete list of a node’s neighbors. We call this type of

Figure 13.1: Exploring a graph by DFS and BFS. In both cases, I label each node with the order in which it gets explored. I use the node color to encode the same information, from light (early explored) to dark (late explored).

| Src | Trg | Node | Neighbors |
|-----|-----|------|-----------|
| 1 | 2 | 1 | 2,3 |
| 2 | 4 | 2 | 1,4 |
| 5 | 3 | 5 | 3,4 |
| 1 | 3 | 3 | 1,5 |
| 4 | 5 | 4 | 2,5 |

(a) Edge list.

(b) Adjacency list.

graph storing format an “adjacency list”, because it’s a list of the adjacencies of one node per line. Also in this case, you may or may not have sorted the nodes – and their neighbors – in a particular way, but the baseline assumption is that they’re in a random order. See Figure 13.2(b) for an example.

13.2 Finding Shortest Paths

Recall that in Chapter 10 we have defined the length of a walk and of a path as the number of edges one crosses to complete the walk/path. The concept of length is crucial when we want to talk about optimality – which is to find the shortest (smallest length, fewest edges used) path between nodes u and v .

The problem of exploring the graph via BFS or DFS is that they are not optimal, they cannot find the “best” (shortest) way to go from v to u . Well, they can, but only in very specific cases under very specific assumptions. For instance, BFS finds shortest paths only for undirected unweighted graphs. This can be useful, for instance, to find the shortest path out of a maze^{1,2,3}. But we still need a better, more general way.

To see why finding “best paths” is important, suppose you have to deliver a letter to a person, as I show in Figure 13.3. If you know them, no problem: you just give it to them. What if you don’t? You might know one of their friends, and pass through them. Or you might know that one of your friends knows one of theirs. But if none of this is true, you have to know the shape of the entire social network – the part in gray in the figure – and discover what’s the least amount of people you have to bother to get your letter to the recipient. This we call the “shortest path problem” in networks.

The formal specification of the shortest path problem is the following. You’re given a start node v and a target node u . You have to find the path going from v to u crossing the fewest possible number of edges. Figure 13.4 provides a visualization of a shortest path between two nodes. In fact, the figure highlights a feature of this problem: it provides not one but two solutions. That is because, in unweighted undirected graphs, it is quite common to find multiple shortest paths

Figure 13.2: Two different ways to store graphs on disk.

¹ Shimon Even. *Graph algorithms*. Cambridge University Press, 2011

² Edward F Moore. The shortest path through a maze. In *Proc. Int. Symp. Switching Theory*, 1959, pages 285–292, 1959

³ Konrad Zuse. *Der Plankalkül*. Number 63. Gesellschaft für Mathematik und Datenverarbeitung, 1972

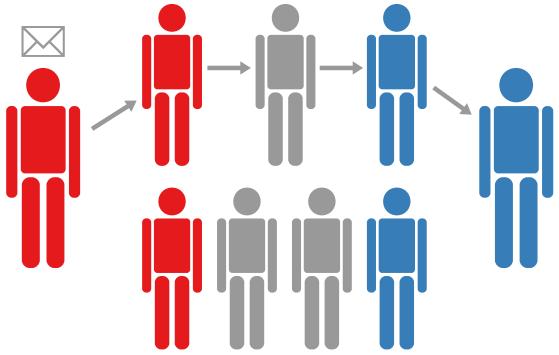


Figure 13.3: A vignette representing the problem of delivering a letter through acquaintances: how do you know the best path is at the top since you're unaware of the existence of the people in gray?

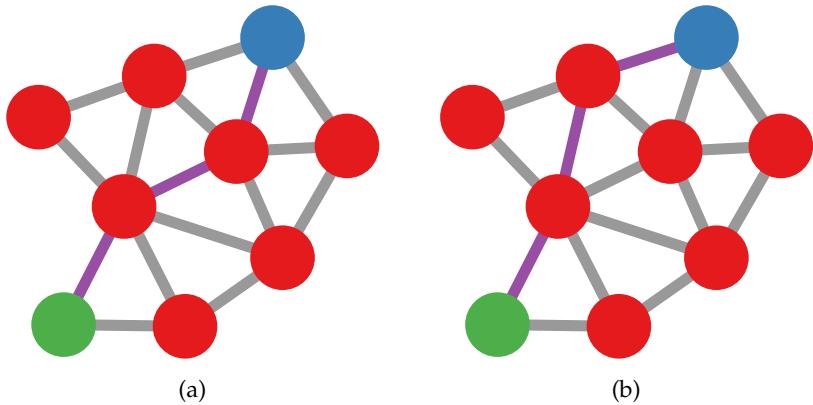


Figure 13.4: Finding the shortest path – edges colored in purple – between the start node (in blue) and the target node (in green). Note that (a) and (b) are both valid shortest paths which have the same length.

between a given origin-destination pair. In other words, the solution to the shortest path problem is not necessarily unique.

This is for the case of undirected, unweighted networks. If you have directed networks you obviously have to respect the edge directions – see Figure 13.5(a). If you have weighted networks, you might want to minimize the weight (as in Figure 13.5(b)), assuming that the edge weight represents its traversal cost. If your edge weights represent proximities rather than costs, for instance they are the capacity of a trait of road as explained in Section 6.3, you'd do the opposite.

How do we find the shortest path? Depending on the properties

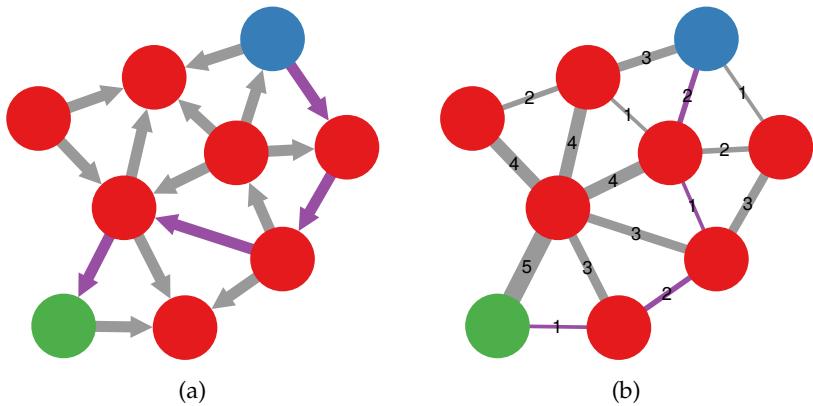


Figure 13.5: Finding the shortest path – edges colored in purple – between the start node (in blue) and the target node (in green). (a) Directed network. (b) Weighted network, where edge weights represent the cost of traversal.

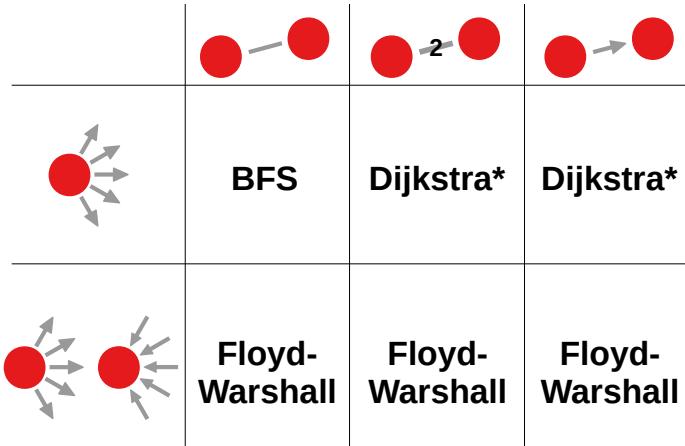


Figure 13.6: A quick reference of the most well known algorithms used to solve specific shortest path problems. Columns (from left to right): simple, weighted, directed. Rows (top to bottom): single-origin, all pairs shortest path. Dijkstra is marked with a star because variants of the base algorithm can outperform it in special cases and they are used in most real world scenarios.

of the graph (e.g. direct/undirected, weighted/unweighted), there are different algorithms for finding shortest paths. We also need to know if we just want to find a path from v to u (single-origin single-destination shortest path), or from v to all other nodes (single-origin shortest path), or between every single pair of nodes.

Figure 13.6 provides you with a quick reference on which algorithm to use given each use case. For instance, as mentioned before, if you have an undirected unweighted network and you are interested in single-origin shortest paths, you can find them by performing a simple BFS exploration.

If you still have a single-origin in mind but your network contains directions and/or weights, you'll probably use one of the many flavors of the classical Dijkstra's algorithm⁴. Dijkstra's algorithm works as follows. You start by your origin, which you mark as your "current node".

1. You look at all the unvisited neighbors of the current node and calculate their tentative distances through the current node.
2. Compare this tentative distance to the current assigned value and assign the smallest one.⁵
3. When you are done considering all of the unvisited neighbors of the current node, mark the current node as visited. You will never check a visited node twice.
4. If the current node, the one you're marking as visited, is the destination node, you can stop. Otherwise, you can continue by selecting the unvisited node that is marked with the smallest tentative distance, as your new current node. Then go back to step 1.

I cannot include in the book the best visual representation of the

⁴ Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959

⁵ For example, if the current node u is at distance of 6 from the source, and the edge connecting it with a neighbor v has length 2, then the distance to v through u is $6 + 2 = 8$. If you previously marked v with a distance greater than 8, you will change it to 8. Otherwise you will keep the current value.

Dijkstra algorithm I know, because it is an animated GIF⁶.

Faster variations of the Dijkstra algorithm^{7,8,9,10} use clever data structures and a few optimizations – often under assumptions about the edge weights – that are of no interest here.

The only other algorithm in the hall of fame of shortest path algorithms we consider here is Floyd-Warshall^{11,12,13,14}. That is because it is the most used algorithm for the all-pairs shortest path problem, when you're not limiting yourself to a single origin and/or a single destination – or to specific constraints on topology and/or edge weights. The algorithm uses recursive programming which, to this day, I still consider borderline magic.

Suppose you have a function $sp(u, v, K)$ that calculates the shortest path between u and v using only nodes in the set K . K is a special set, it contains all nodes of the network with id equal to or lower than K . Obviously, if $K = 0$, then it is an empty set. Then $sp(u, v, 0)$ simply returns the weight of the edge between u and v – if they are connected –, because we're not using any node in the path:

$$sp(u, v, 0) = A_{uv}.$$

If $K > 0$ it means that we are adding a node as a possible member of the shortest path. When we do it, either of two things can happen: (i) adding the extra node allowed us to find a better (shorter) path, or (ii) it didn't. So:

$$sp(u, v, K) = \min(sp(u, k, K) + sp(k, v, K), sp(u, v, K - 1)).$$

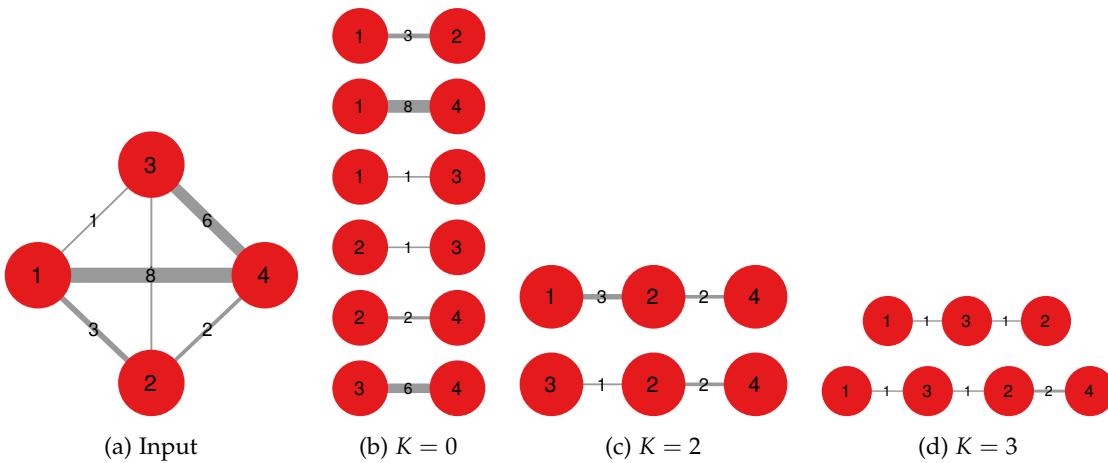


Figure 13.7 shows an example run. Figure 13.7(a) is an hypothetical input. At the first step, $K = 0$, we can only consider directly connected origins and destinations, setting the edge weights as the length – Figure 13.7(b). For $K = 1$ (not pictured) nothing happens:

⁶ https://upload.wikimedia.org/wikipedia/commons/5/57/Dijkstra_Animation.gif

⁷ Robert B Dial. Algorithm 360: Shortest-path forest with topological ordering [h]. *Communications of the ACM*, 12(11):632–633, 1969

⁸ Ravindra K Ahuja, Kurt Mehlhorn, James Orlin, and Robert E Tarjan. Faster algorithms for the shortest path problem. *Journal of the ACM (JACM)*, 37(2):213–223, 1990

⁹ Rajeev Raman. Recent results on the single-source shortest paths problem. *ACM SIGACT News*, 28(2):81–87, 1997

¹⁰ Mikkel Thorup. On ram priority queues. *SIAM Journal on Computing*, 30(1):86–109, 2000

¹¹ Bernard Roy. Transitivité et connexité. *Comptes Rendus Hebdomadaires Des Séances De L Académie Des Sciences*, 249(2):216–218, 1959

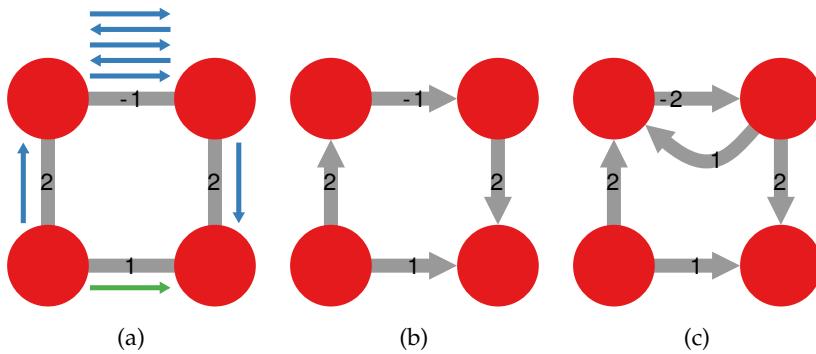
¹² Stephen Warshall. A theorem on boolean matrices. *Journal of the ACM (JACM)*, 9(1):11–12, 1962

¹³ Robert W Floyd. Algorithm 97: shortest path. *Communications of the ACM*, 5(6):345, 1962

¹⁴ Bernard Roy, who actually discovered the algorithm first, for mysterious reasons gets no naming rights.

Figure 13.7: (a) The input for the Floyd-Warshall algorithm. (b-d) The temporary shortest paths at each step of the algorithm.

node 4 cannot use node 1 to go anywhere, because their edge is very costly, and nodes 2 and 3 have low cost connections to node 1, but they are already directly connected by the minimum weight in the network. For $K = 2$ (Figure 13.7(c)) we're also allowed to use node 2 for our paths. Both node 1 and node 3 use it to get to node 4, given that their direct connection to node 4 is costly. For $K = 3$ (Figure 13.7(d)) we can also use node 3 in our paths. The path $1 \rightarrow 3 \rightarrow 2$ is the sum of two paths we already know from Figure 13.7(b): $1 \rightarrow 3$ and $3 \rightarrow 2$. It costs less than $1 \rightarrow 2$, so we select it. To go from node 1 to node 4 we sum two paths we already know: $1 \rightarrow 3$ (from Figure 13.7(b)) and $3 \rightarrow 2 \rightarrow 4$ (from Figure 13.7(c)). We discover then that the actual distance between the nodes 1 and 4 is four, rather than five – as we thought in Figure 13.7(c) – or eight – as we thought in Figure 13.7(b).



A final word about negative weights. As presented earlier, there's no shame if your network contains them (see Section 6.3). However, you need to be careful when computing shortest paths. The reason is evident, as one can see from Figure 13.8(a). The problem with negative weights is that we might think that it is trivial to find a shortest path (in green in the figure), but by going back and forth over a negative weight we can find an equivalent path. At that point, we can be stuck in an infinite loop of shorter and shorter paths without ever reaching the destination (in blue in the figure).

Directed networks can allow negative weights, because you're not allowed to follow the edge against its direction, as in Figure 13.8(b). However, if there is a negative cycle – see Figure 13.8(c) – you are in the same situation as before. A negative cycle is a cycle whose total edge weight sum is lower than zero.

If you're writing shortest path algorithms, you have to take care of these situations. Usually, you have to explicitly say that you're looking for *paths*, not *walks*. In paths, you cannot re-use the same edge twice (see Chapter 10), no matter how cool it would make your path length.

Figure 13.8: (a) A weighted network with negative weights which results in degenerate shortest paths – in blue – over preferred non-shortest paths – in green. (b) A directed weighted network with negative weights but without the infinite negative weight problem. (c) A directed weighted network with negative cycles.

13.3 Path Length Distribution

Just like with the degree, knowing the length distribution of all shortest paths in the network conveys a lot of information about its connectivity. A tight distribution with little deviation implies that all nodes are more or less at the same distance from the average node in the network. A spread out distribution implies that some nodes are in a far out periphery and others are deeply embedded in a core.

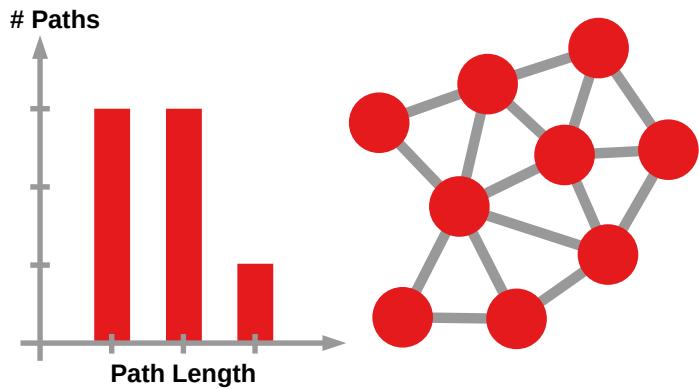


Figure 13.9: The path length distribution (left) of a graph (right). Each bar counts the number of shortest paths of length one, two and three, which is the maximum length in the network.

To generate a path length distribution you perform the same operation you used to get the degree distribution: you have the path length on the x axis and the number of paths of a given length on the y axis. See Figure 13.9 for an example. I'm not going to go on a tangent on log-log spaces and power laws like last time because usually path lengths distribute quasi-normally: you'll find a lot of classical bell shapes.

Some values in the distribution are fixed. For instance, the number of paths of length one is twice the number of edges, because each edge is used for two paths of length one ($u \rightarrow v$, and $v \rightarrow u$). It goes without saying that things are different in directed networks. The number of total shortest paths is $|V|(|V| - 1)$, because each origin has to reach each destination, minus one because we don't count the paths of length zero, from the origin to the origin.

Diameter

The rightmost column of the histogram in Figure 13.9 is important. It records the number of shortest paths of maximum length. These are the “longest shortest paths”. Since this is an important concept, such a long mouthful name won’t do. We’re busy people and we got places to be. So we use a different name for them or, to be more precise, to their length. We call it the **diameter** of the network.

Why do we care about the diameter? Because that’s the worst case

for reachability in the network. The diameter is the measure of the maximum possible separation between two nodes. A long diameter means that the problem of finding a shortest path for some pairs of nodes might be too hard because there are too many hypothetical paths and splits to consider. With a small diameter, everybody is reachable in one or two hops. With a large diameter, a full traversal of the graph might be impossible, especially if we only have local information about our neighborhood.

Let's go over a few values of diameter, just to get a grasp of the concept:

- Diameter = 1 → You know everyone;
- Diameter = 2 → Your friends know everyone;
- Diameter = 3 → Your friends know someone who knows everyone;
- ...

It's now easy to see that a network with diameter equal to three is easy to navigate. As the diameter grows, the number of people to rely on for a full traversal starts becoming unwieldy.

If your network has multiple connected components (Section 10.4), we have a convention. Nodes in different components are unreachable, and thus we say that their shortest path length is infinite. Thus, a network with more than one connected component has an infinite diameter. Usually, in these cases, what you want to look at is the diameter of the giant connected component.

Average

The diameter is the worst case scenario: it finds the two nodes that are the most far apart in the network. In general, we want to know the typical case for nodes in the network. What we calculate, then, is not the longest shortest path, but the typical path length, which is the average of all shortest path lengths. This is the expected length of the shortest path between two nodes picked at random in the network.

If P_{uv} is the path to go from u to v and $|P_{uv}|$ is its length, then the average path length of the network is $APL = \frac{\sum_{u,v \in V} |P_{uv}|}{|V|(|V| - 1)}$. Figure 13.10 shows that, even in a tiny graph, the diameter and the APL can take different values, with the former being more than twice the length of the latter.

With APL, we can fix the origin node. For instance, in a social network, you can calculate your average separation from the world.

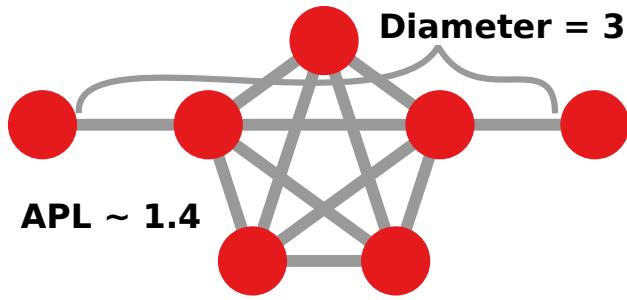


Figure 13.10: The diameter and the APL in a graph can be quite different.

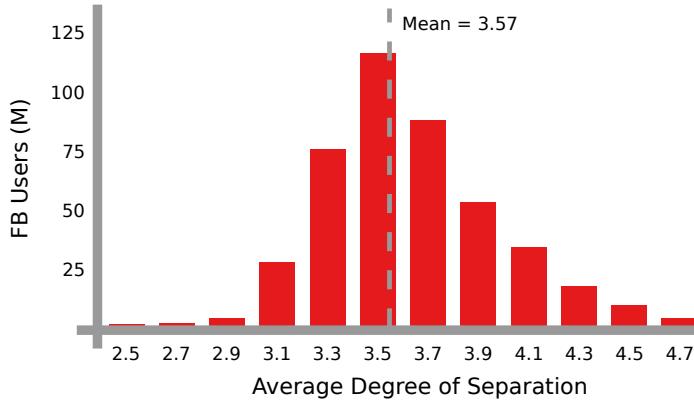


Figure 13.11: The path length distribution for Facebook in 2012.

This would be an APL_v , the average path length for all paths starting at v . Then you can generate the distribution of all lengths for all origins. How does this APL_v distribution look like for a real world network? One of the most famous examples I know comes from Facebook¹⁵. I show it in Figure 13.11. The remarkable thing is how ridiculously short the paths are even in such a gigantic network.

This is in line with classical results of network science, showing that the diameter and APL typically grow sublinearly in terms of number of nodes in the network¹⁶. In other words, there are diminishing returns to path lengths: each additional person contributes less and less to the growth of the system in terms of reachability. In fact, some researchers have found that adding people might even shrink the diameter^{17,18}: as people join a social network, they create shortcuts and new paths that bring close together people that were previously far apart.

The most notorious enunciation of the surprising small average path length in large networks is the famous “six degrees of separation”. This concept says that, on average, you’re six handshakes away from meeting any person in the world, being a fisherman in Cambodia or an executive in Zimbabwe. People used this concept to describe the famous – failed – Milgram experiment.

In 1967, Milgram published a paper¹⁹ detailing the travels of a

¹⁵ Sergey Edunov, Carlos Diuk, Ismail Onur Filiz, Smriti Bhagat, and Moira Burke. Three and a half degrees of separation. *Research at Facebook*, 2016

¹⁶ Mark EJ Newman. The structure and function of complex networks. *SIAM review*, 45(2):167–256, 2003b

¹⁷ Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graphs over time: densification laws, shrinking diameters and possible explanations. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 177–187. ACM, 2005a

¹⁸ Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graph evolution: Densification and shrinking diameters. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 1(1):2, 2007b

¹⁹ Stanley Milgram. The small world problem. *Psychology today*, 2(1):60–67, 1967

series of envelopes. He handed a destination address to people in the Midwest of the United States. The destination was in Boston, Massachusetts. The idea was that each recipient needed to attempt to have the letter reach its final destination. However, they could not mail it directly: they needed to hand it over to a person they knew on a first name basis. So they needed to figure out who in their acquaintances was most likely to know somebody (who knew somebody, who knew somebody, ...) in Massachusetts. Each handler of the envelope would have to write their name on it. When the envelope reached the destination, counting the names in it would give an approximation of the degrees of separation between the origin and destination individuals.

The number turned out to be 5.5 on average, which gave fuel to the “six degrees of separation” urban legend. However, the experiment was arguably a failure given that, of the more than 400 letters sent, less than a hundred actually arrived at the destination. The problem is that obviously there is no way to account for the fact that a letter might not successfully reach its target because some people in the chain were unreliable, rather than unconnected with the destination. Fascinating as it is, this theory might be wrong because the degrees of separation could be lower than six: people have proposed four²⁰, as we see in Facebook (Figure 13.11).

Diameter and average path length are only the two most famous and most used measures derived from the shortest path length distribution. There is a collection of other measures you might find in network science papers and books. Two other examples are the eccentricity of a node and the radius of a network. You can think of the eccentricity as a node-level diameter. It is the longest shortest path leading from node u to the farthest possible node v in the network. Thus, by definition the diameter is equal to the highest eccentricity among the nodes of the network. The radius of a network is, conversely, equal to the smallest eccentricity in the network.

13.4 Spanning Trees & Other Filtered Graphs

I conclude this chapter with a look at spanning trees and other ways to filter down a graph. These methods are usually deployed to reduce a network to its minimum terms and finding its fundamental structure in a way that is parsable by humans. They are also at the basis of some network backboning techniques (Chapter 27).

A spanning tree of an undirected graph is a subgraph that: (i) is a tree (see Section 10.2), and (ii) it includes all of the vertices of the graph. In practice, it is that subgraph that can connect all nodes of the graph with the minimum number of edges, and no cycles.

²⁰ Lars Backstrom, Paolo Boldi, Marco Rosa, Johan Ugander, and Sebastiano Vigna. Four degrees of separation. In *Proceedings of the 4th Annual ACM Web Science Conference*, pages 33–42. ACM, 2012

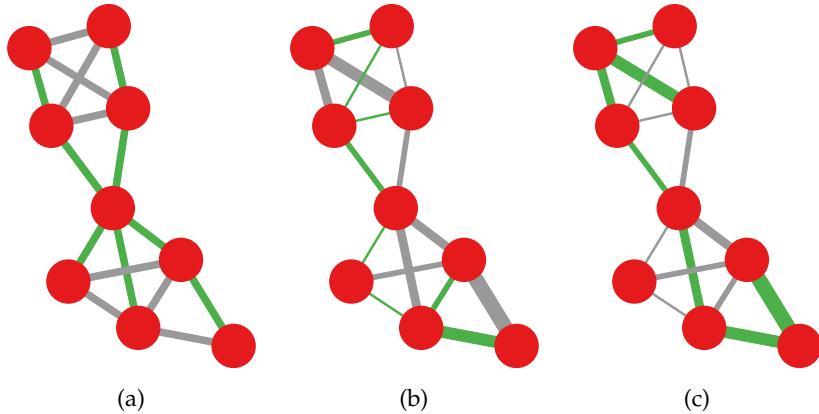


Figure 13.12: (a) A graph with one of its possible spanning trees highlighted in green. (b) The minimum spanning tree of a weighted graph, with the edge width proportional to its weight. (c) The maximum spanning tree of a weighted graph.

Figure 13.12(a) shows you an example of a spanning tree inside a graph. If your graph has multiple connected components you cannot find a single spanning tree, because you don't have a way to connect nodes in different components. However, you can make a spanning forest, by finding the spanning trees of each component separately.

Spanning trees are nice, but they get used mostly in weighted networks. In that case, you have to distinguish between weights as proximities and weights as distances (Section 6.3): is an edge with a high weight expressing the cost of going from u to v , or is it saying how much u and v interact? In the first case we have a "distance" weight: we want to minimize costs. Imagine finding the tree connecting all your road intersections that minimizes driving distance – the cost of an edge.

When your weights are distances you want a minimum spanning tree²¹: the spanning tree among all spanning trees of a graph that has the minimum possible total edge weight. Figure 13.12(b) shows an example. When your weights are proximities – maybe because they tell you the capacity of the road – then you want the maximum spanning tree: the spanning tree among all spanning trees of a graph that has the maximum possible total edge weight. Figure 13.12(c) shows an example.

Of course, the algorithm to find the minimum and the maximum spanning tree is the same, you just flip the sign of the comparison. There's a good range of algorithms, from classical ones to more modern which use special data structures: Borůvka²², Prim²³, Kruskal²⁴, Chazelle²⁵. They are usually all implemented in standard network analysis libraries.

Note that finding the minimum spanning tree doesn't really solve the traveling salesman problem²⁶, although it sounds like it should. A quick recap: the traveling salesman problem is the quest to find the shortest possible route that visits each city and returns to the

²¹ Ronald L Graham and Pavol Hell. On the history of the minimum spanning tree problem. *Annals of the History of Computing*, 7(1):43–57, 1985

²² Otakar Borůvka. O jistém problému minimálním. 1926

²³ Robert Clay Prim. Shortest connection networks and some generalizations. *Bell system technical journal*, 36(6):1389–1401, 1957

²⁴ Joseph B Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical society*, 7(1):48–50, 1956

²⁵ Bernard Chazelle. A minimum spanning tree algorithm with inverse-ackermann type complexity. *Journal of the ACM (JACM)*, 47(6):1028–1047, 2000

²⁶ Eugene L Lawler, Jan Karel Lenstra, AHG Rinnooy Kan, David Bernard Shmoys, et al. *The traveling salesman problem: a guided tour of combinatorial optimization*, volume 3. Wiley New York, 1985

origin city, given a list of cities and the distances between each pair of cities. We can represent the problem as a weighted graph, with city distances as edge weights. The minimum spanning tree doesn't solve the problem: it creates a tree, which has no cycles. Thus, to get back to the origin city, you have to backtrack all the way through the tree – not ideal.

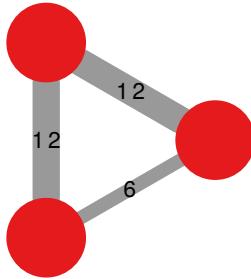


Figure 13.13: An example of a weighted graph with a non unique minimum spanning tree.

Another thing to keep in mind is that rarely minimum/maximum spanning trees are unique: a weighted network can and will have multiple alternative minimum/maximum spanning trees. Consider the graph in Figure 13.13. Suppose we want to find its minimum spanning tree. The first choice is obvious: we use the edge of weight 6. Then, we have to connect the final node. Each of the edges of weight 12 is a valid addition to the tree: they will connect the node and the result will be a tree, an acyclic graph. So the graph has two valid minimum spanning trees.

There is an easy rule to remember to know whether a graph will have a unique minimum/maximum spanning tree or not. If each edge has a distinct weight then there will be only one, unique minimum spanning tree. As soon as you have two edges with the same weight, you open the door to the possibility of having more than one minimum spanning tree. In fact, in an unweighted graph where we assume that all edges have the same weight equal to one, then every spanning tree of that graph is minimum.

Spanning trees have some closely related cousins that are worthwhile mentioning. The first one is the planar maximally filtered graph²⁷. As the name suggests, this is a technique to reduce any arbitrary graph into a planar version of itself, such that the edge weight sum is maximal (or minimal, depending on the meaning of your edge weights). Since a spanning tree is a tree, it means that it must have $|V| - 1$ edges. On the other hand, a planar maximally filtered graph must have $3(|V| - 2)$ or fewer edges.

Just like in the case of the tree, also in this case some motifs cannot appear. In a tree you cannot have cycles. In a planar graph you cannot have a motif that is impossible to draw as planar – i.e. on a 2D surface without edge crossings –, for instance a 5-clique or a 3,3-

²⁷ Michele Tumminello, Tomaso Aste, Tiziana Di Matteo, and Rosario N Mantegna. A tool for filtering information in complex systems. *Proceedings of the National Academy of Sciences*, 102(30):10421–10426, 2005

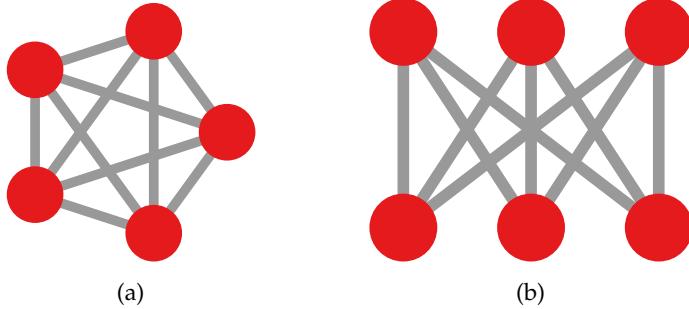


Figure 13.14: Two examples of non planar graphs that cannot be included in any planar maximally filtered graph. (a) A 5-clique; (b) a 3,3-biclique.

biclique. Look at Figure 13.14 and try to draw those graphs in two dimensions without having any edge crossing another one. You'll find out that is not possible.

The second cousin of spanning trees is the triangulated maximally filtered graph²⁸. This was originally proposed as a more efficient algorithm to extract planar maximally filtered graphs from larger graphs. However, it also allows to specify different topological constraints, which are not necessarily making the graph planar.

²⁸ Guido Previte Massara, Tiziana Di Matteo, and Tomaso Aste. Network filtering for big data: Triangulated maximally filtered graph. *Journal of complex Networks*, 5(2):161–178, 2016

13.5 Classic Combinatorial Problems

Graph exploration in general, and shortest paths in particular, are linked with some of the most famous problems discussed in computer science. We already saw one in Section 12.4 – graph coloring: how many colors do I need to make sure that I don't give the same one to two connected nodes? Here I mention another, related to the classic Traveling Salesman Problem. In the Traveling Salesman Problem, we have a set of cities and we want to find the path that allows us to visit all cities by covering the minimum possible distance.

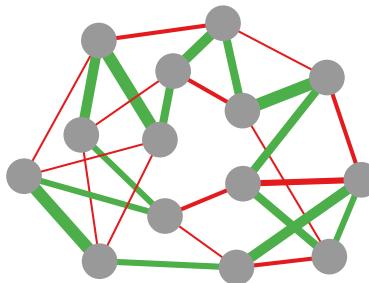


Figure 13.15: A graph with two Hamiltonian cycles highlighted using the edge color. Red = minimum Hamiltonian; green = maximum Hamiltonian. The edge's thickness is proportional to its weight.

In this scenario, we are assuming that cities live in a two dimensional space and there is a path between any two cities. However, we could impose the existence of a road graph that makes some city-city connections impossible. In this case, we want to find the path of minimum cost in a graph that visits each node exactly once (i.e. the minimum Hamiltonian path – see Chapter 10). Figure 13.15 shows an

example, with two Hamiltonian paths of different costs highlighted in red and green.

Such problems have a huge importance in computer science because they are classical examples of NP-hard problems. These problems have no known polynomial-time solution, meaning that we can usually only find approximate solutions in a reasonable time. Finding the best solution would require brute force algorithms whose time complexity make them unsuitable for problems of large size – i.e. if your graph has more than a handful nodes.

Combinatorics and graphs have a much deeper relationship than this one, though. A vast number of problems in combinatorics can be represented as a graph problem, and often graphs are the best tool to solve them. Two other examples are the classic SAT problem, where we want to know if there is a true/false assignment so that a set of logical propositions is not contradictory; and vehicle routing, where we want to find the optimal set of routes for several vehicles to reach their destinations from their origins.

13.6 Summary

1. There are many ways to explore a graph structure. Breadth-First Search means to explore all neighbors of a node before exploring their neighbors; Depth-First Search means to explore a neighbor's neighbors before moving on to the next direct neighbor; random node and edge access means to explore one node or edge at a time ignoring the graph's topology.
2. Shortest paths are the paths connecting two arbitrary nodes in the network using the minimum possible number of edges. In directed networks you have to respect the edge's direction, in weighted networks you have to minimize (or maximize, depending on the problem definition) the sum of the edge weights.
3. The most common algorithms to solve shortest path finding are Dijkstra (if you have a fixed origin and destination) or Floyd-Warshall (if you are calculating the shortest paths between all pairs of nodes in the network).
4. Two important network connectivity measures are the diameter and the average path length. The diameter is the length of the longest shortest path. The average path length is the average length of all shortest paths in the network.
5. A minimum spanning tree is a tree connecting all nodes in the network which minimizes the sum of edge weights.

13.7 Exercises

1. Label the nodes of the graph in Figure 13.12(a) in the order of exploration of a BFS. Start from the node in the bottom right corner.
2. Label the nodes of the graph in Figure 13.12(a) in the order of exploration of a DFS. Start from the node in the bottom right corner.
3. Calculate all shortest paths for the graph in Figure 13.12(a).
4. What's the diameter of the graph in Figure 13.12(a)? What's its average path length?

14

Node Ranking

The most direct way to find the most important nodes in the network is to look at the degree. The more friends a person has, the more important she is. This way of measuring importance works well in many cases, but can miss important information. What if there is a person with only few friends, but placed in different communities – just like in Figure 14.1? The removal of such person will create isolated groups, which are now unable to talk to each other. Shouldn't this person be considered a key element in the social network, even with her puny degree?

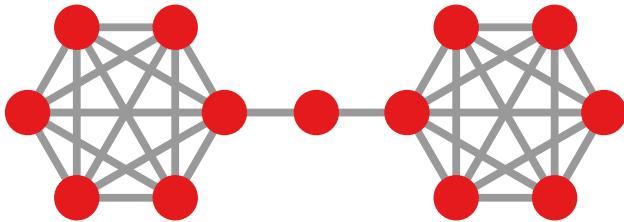


Figure 14.1: An example of a social network in which the degree does not necessarily convey all the information about node importance.

Many networks scientists agree that she should, and developed different centrality measures accordingly. Here we focus on a few examples.

14.1 Closeness

If we want to know the closeness centrality¹ of a node v , first we calculate all shortest paths starting from that node to every possible destination in the network: P . Each of these paths P_{vu} has a length, which is the number of edges you need to cross to get to your destination. Let's call it $|P_{vu}|$ – the length to go from v to u . We sum these distances in a total distance measure: $\sum_u |P_{vu}|$. We take the average of this value by dividing it by the number of all possible destinations u , which is the number of nodes in the network minus one (the origin): $\sum_u |P_{vu}| / (|V| - 1)$. Then, since the measure is called *closeness*,

¹ Alex Bavelas. A mathematical model for group structures. *Human organization*, 7(3):16, 1948

we don't want to look at it directly. Closeness is the opposite of distance. So we actually want the opposite of what we just calculated, or $(|V| - 1) / \sum_u |P_{vu}|$. If this looks familiar, that's because it is. The closeness centrality of v is nothing more than its inverse average path length (see Section 13.3), or $1/APL_v$.

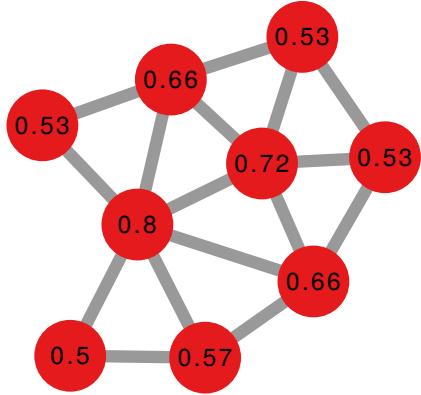
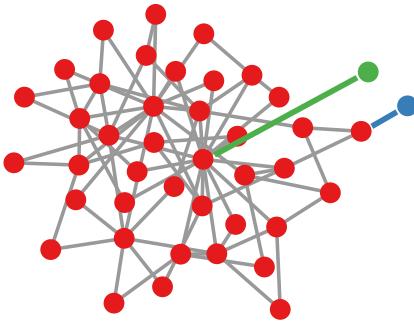


Figure 14.2: A sample network. Node labels represent the closeness centrality value for the node.

Let's look at an example – in Figure 14.2. Let's consider the node in the bottom left, labeled with 0.5. That is its closeness centrality. How do we get to that value? First, we start with the nodes directly connected to it. The shortest paths to get to them is to follow the direct connections, thus only one edge is crossed. Both neighbors contribute $|P_{vu}| = 1$. Moving on, the two neighbors allow our v node to access to four more nodes. These four nodes require to cross an additional edge, thus they contribute $|P_{vu}| = 2$. We are left with two more nodes that require a third edge to be reached: $|P_{vu}| = 3$. So to recap, the total distance of this node is $1 + 1$ (the two direct neighbors) $+ 2 + 2 + 2 + 2$ (the four nodes at distance two) $+ 3 + 3$ (the final two nodes at distance three) $= 16$. We then take the average $(16/(9 - 1))$ and convert this into a closeness: $(9 - 1)/16 = 0.5$.

The advantage of closeness centrality is that it has a spatial intuition: the closer you are on average to anybody, the more central you are. Exactly like standing in the middle of a room makes you closer on average to each member of the crowd in a party than standing in a corner. Empirically, in the vast majority of networks I analyzed, closeness centrality is distributed on a classical bell shape, i.e. normally. If you use closeness centrality, most of your nodes will have an average importance. This is not realistic for many networks: we know that degree distributions are very skewed – the vast majority of nodes are unimportant, while only a few selected superstars take all the glory.

Why does closeness centrality behave so differently from the degree? How can two nodes with very low degree – for instance equal to one – have different closeness centrality values so that they end up



distributing normally instead of on a skewed arrangement? One possible explanation is that edge creation is a lottery. The many nodes with degree equal to one that you have in broad degree distributions can get lucky with their choice of neighbor. Sometimes, like in the case of the green node in Figure 14.3, the neighbor is a hub. The green node's closeness centrality will then be high, because it is just one extra hop away from the hub itself – which is very central. Sometimes the new node will attach itself to the periphery – like the blue node in Figure 14.3 –, and thus have a very low closeness centrality.

14.2 Betweenness

Network scientists developed betweenness centrality^{2,3} to fix some of the issues of closeness centrality. Differently from closeness, with betweenness we are not counting distances, but paths. We still calculate all shortest paths between all possible pairs of origins and destinations. Then, if we want to know the betweenness of node v , we count the number of paths passing through v – but of which v is neither an origin nor a destination. In other words, the number of times v is *in between* an origin and a destination. If there is an alternative way of equal distance to get from the origin to the destination that does not use v , we discount the contribution of the path passing through v to v 's betweenness centrality. I provide an example in Figure 14.4.

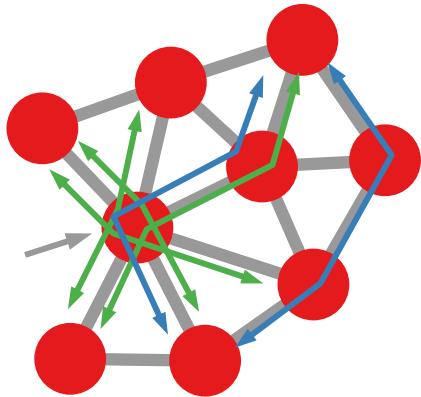
The total number of paths that can pass through a node – excluding the ones for which it is the origin or the destination – are $(|V| - 1)(|V| - 2)$ in a directed network, and $(|V| - 1)(|V| - 2)/2$ in an undirected one.

One intuitive way to think about betweenness centrality is asking yourself: how many paths would become longer if node v would disappear from the network? How much is the network structure dependent on v 's presence? Since real world networks have hubs which are closer to most nodes, the shortest paths will use them often. As a result, betweenness centrality distributes over many

Figure 14.3: The closeness centrality lottery. The blue and green nodes are new to the network and only have one edge to attach. The green node is lucky, and connects to a central hub. The blue node is unlucky and connects to a peripheral node. Thus nodes with the very same low degree end up with radically different closeness centrality values.

² Jac M Anthonisse. The rush in a directed graph. *Stichting Mathematisch Centrum. Mathematische Besliskunde*, (BN 9/71), 1971

³ Linton C Freeman, Douglas Roeder, and Robert R Mulholland. Centrality in social networks: II. experimental results. *Social networks*, 2(2):119–141, 1979



orders of magnitude, just like the degree. Unlike the degree, it takes into account more complex information than simply the number of connections.

The concept underlying betweenness centrality can be extended to go beyond nodes. You can use it to gauge the structural importance of edges. The definition is the same: the betweenness of an edge is the (normalized) count of shortest paths using the edge. If applied to connections, we call this measure “edge betweenness”. This is a key concept especially for the field of community discovery, as we will find out in Part X.

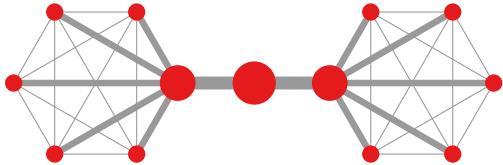


Figure 14.5 shows an example of edge betweenness centrality. The intuition here is the same: the edge betweenness is the number of paths that would get longer if the edge were to disappear. Note, though, that if we remove the edge from the network, almost all of the edge betweenness centralities will have to be recalculated. It is very hard to figure out the second order effects of the edge disappearance on the shortest paths of the network. The edge betweenness of some edges might increase by one, but it is not easy to understand which ones.

In some cases – and actually this might be very likely – the network will be broken up into multiple components, meaning that no edge will increase its betweenness and, instead, many will lose part of their centrality. That is because now there will be many node pairs that cannot reach each other any more. Consider again Figure 14.5: once we remove one of the two most central edges, no node

Figure 14.4: An example on how to calculate the betweenness centrality of the node marked with the gray arrow. The shortest paths passing through it – in green – contribute to its betweenness centrality. If there are n alternative paths not passing through the node, then the path contributes only $1/n$ to the node’s centrality – I show an example in blue.

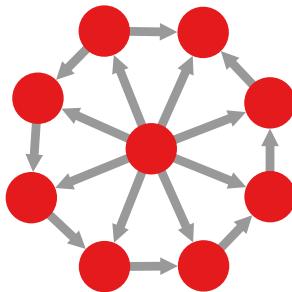
Figure 14.5: In this network the edge thickness is proportional to its edge betweenness value. The node size is proportional to the node betweenness.

in one clique can reach the nodes in the other one. All those paths passing through the removed edge are lost forever. The surviving edge between the two most central ones will have a much reduced edge betweenness centrality: it cannot be used to move between cliques any more. This consideration holds true not only for the edge betweenness, but also for the node betweenness.

A relaxed version of betweenness centrality does not use shortest paths, but random walks (Chapter 11). This simulates the spreading of information into the network. The definition is similar: this “flow” centrality is the number of random walker passing through the node during the information spread event^{4,5}. Just like with the regular betweenness centrality, also in this case you can take an edge-centric approach, and count the number of random walks going through a specific edge. This has been used, for instance, to solve the problem of community discovery^{6,7}.

14.3 Reach

Reach centrality is only defined for directed networks. The local reach centrality of a node v is the fraction of nodes in a network that you can reach starting from v ⁸. From this definition, one can see why it doesn’t make much sense for undirected networks. If your network has a single connected component, then all nodes have the same reach centrality, which is equal to one. That is also the case if your directed network has only one strongly connected component. In a strongly connected component there are no “sinks” where paths get trapped, thus every node can reach any other node.



However, when you have multiple (or no) strongly connected components, reach centrality tells you how much you can command in your network if you’re node v . Consider Figure 14.6. There is a clear boss in this figure: the central node. Following its edges, you can reach the entirety of the network. Thus its reach centrality is equal to one. On the other hand, the node on the top right has an out-degree of zero. You cannot reach anything from it, thus its reach

⁴ Mark EJ Newman. A measure of betweenness centrality based on random walks. *Social networks*, 27(1): 39–54, 2005a

⁵ Ulrik Brandes and Daniel Fleischer. Centrality measures based on current flow. In *Annual symposium on theoretical aspects of computer science*, pages 533–544. Springer, 2005

⁶ Santo Fortunato, Vito Latora, and Massimo Marchiori. Method to find community structures based on information centrality. *Physical review E*, 70(5):056104, 2004

⁷ Vito Latora and Massimo Marchiori. Vulnerability and protection of infrastructure networks. *Physical Review E*, 71(1):015103, 2005

⁸ Enys Mones, Lilla Vicsek, and Tamás Vicsek. Hierarchy measure for complex networks. *PloS one*, 7(3):e33799, 2012

Figure 14.6: A wheel graph with a flipped edge. The central node has the maximum reach centrality.

centrality is zero. Reach centralities progressively increase if you follow the wheel clockwise, as more and more of the network gets reachable from the nodes' perspective.

Calculating the reach centrality is trivial. You start from node v and you explore the graph with a BFS strategy. Once you cannot explore any more, you stop. The number of nodes you touched divided by the number of nodes in the network is your reach centrality. This is linear in the number of edges in the graph.

Reach centrality is a key concept we use to detect the hierarchical structure of networks, as we will see in Chapter 33.

14.4 Eigenvector

Betweenness centrality shares with closeness a drawback: computational complexity. Both measures require to calculate all shortest paths in the network. For large structures, this becomes unfeasible. The reason fully lies in the shortest paths calculation, which is very computationally expensive. One could approximate the node's importance for connectivity by looking not at shortest paths, but at random walks.

This is different from the flow centrality I explained at the end of Section 14.2 because, in this case, we're not simulating a spreading event. Instead, we're looking at infinite length random walks and we're not bounded by origin-destination pairs of spreading events. Here, we are interested in knowing the expected probability of ending up in a node when we perform a random walk in the network. That is, we start from a random node and we keep choosing to traverse random edges. What's the likelihood of ending up in node v ?

Calculating all shortest paths takes $|V|^3$ operations. By using clever linear algebra, running infinite length random walks could take only $|V|^2$. In fact, we already saw how to calculate the probability of ending in a node after an infinite length random walk: it is the stationary distribution, as I discussed in Section 11.1. By replacing the expensive step at the basis of betweenness centrality with simple random walks, you can obtain phenomenal speedups.

We call methods based on this technique "Eigenvector Centralities", as the stationary distribution is the leading left eigenvector of the stochastic adjacency matrix. If you take the straight up stationary distribution, you obtain what we call the eigenvector centrality^{9,10}.

PageRank

By far, the most famous approach in this category is PageRank¹¹: the

⁹ John R Seeley. The net of reciprocal influence. a problem in treating sociometric data. *Canadian Journal of Experimental Psychology*, 3:234, 1949

¹⁰ Mark EJ Newman. Mathematics of networks. *The new Palgrave dictionary of economics*, pages 1–8, 2016b

¹¹ Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999

algorithm that Google invented in 1998 to rank webpages in their nascent search engine. PageRank is nothing more than calculating a stationary distribution over a directed adjacency matrix. PageRank differs from eigenvector centrality in one tiny – but rather salient – aspect.

If you remember Section 11.1 you'll recall a small issue with the stationary distribution. If your network is not connected, meaning that it has more than one connected component (Section 10.4), you will obtain multiple incomparable stationary distributions. This is bad news for the use case of PageRank: you want to use it to sort out webpages, and the users want to see a single ranking, not one per connected component!

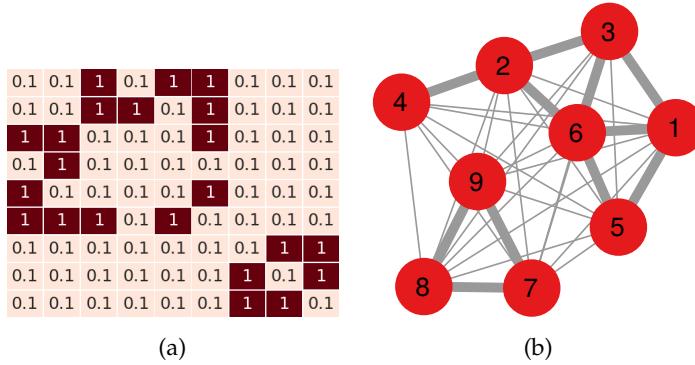


Figure 14.7: The practical implementation of the PageRank's teleportation trick: adjacency matrix (a) and resulting graph (b).

Google's solution for the PageRank algorithm was to give the walker a teleportation device. At each step, the walker has a minuscule chance to request a teleportation, which then might land it on a different component. This mathematical trick is embarrassingly easy to implement. It is equivalent to the creation of ghost edges with very little weight connecting the entire graph. On matrix notation, this is the same as adding a tiny constant to the (not yet normalized) adjacency matrix: $A^* = A + \epsilon$. Figure 14.7 shows this teleportation trick in practice.

However, PageRank is not immune from downsides. PageRank is very close to the degree. How closely the degree approximates the PageRank depends on the value of our teleportation parameter ϵ : in the literature, we call $1 - \epsilon$ the “damping factor”. The magic value of ϵ is 0.15, that is what Brin and Page used originally. If we set $\epsilon = 0$, PageRank is equivalent to π , and therefore to the degree^{12,13}.

Of course, nowadays Google uses a much more complex algorithm to sort the results. Most of the tricks are either secret or too specialized to include here. However, there are a few tweaks of note. For instance, a very popular variant of PageRank is the personalized PageRank¹⁴. In practice, one can split the network to a multilayer one depending on the topic of the hyperlink (e.g. its keywords) and

¹² Santo Fortunato, Marián Boguñá, Alessandro Flammini, and Filippo Menczer. Approximating pagerank from in-degree. In *International Workshop on Algorithms and Models for the Web-Graph*, pages 59–71. Springer, 2006

¹³ Gourab Ghoshal and Albert-László Barabási. Ranking stability and super-stable nodes in complex networks. *Nature communications*, 2:394, 2011

¹⁴ Taber H Haveliwala. Topic-sensi-

Ranit Hogenwald. Topic-sensitive pagerank. In *Proceedings of the 11th international conference on World Wide Web*, pages 517–526. ACM, 2002.

calculate a set of PageRanks, one per topic. There are other possible ways to define a multilayer PageRank¹⁵.

Katz

Another popular variant of eigenvector centrality is Katz centrality¹⁶. At a philosophical level, the difference between the two is that Katz says that nodes that are farther away from v should count less when estimating v 's importance. So it matters whether v is reached at the first step of the random walk, rather than at the second, or at the hundredth. For eigenvector centrality when you meet v in a random walk makes no difference, for Katz it does.

If we were to write the eigenvector centrality not as an eigenvector, but as a sum, we would end up with something that looks a bit like this:

$$EC_v = \sum_{k=1}^{\infty} \sum_{u \in V} (A^k)_{uv},$$

which means that v 's importance is the sum of the probabilities of getting from any u to v in k steps, with k going to infinity. Katz simply adds a term, α , which is lower than one. He plugs it in the formula as follows:

$$KC_v = \sum_{k=1}^{\infty} \sum_{u \in V} \alpha^k (A^k)_{uv}.$$

Since $0 < \alpha < 1$, as k grows the contribution of $(A^k)_{uv}$ becomes more and more insignificant. Which is what Katz wants: longer walks contribute less to v 's centrality.

UBIK

A paper of mine presents UBIK, which is the lovechild between Katz centrality and the personalized PageRank I presented before¹⁷. The weird acronym is short for “you (U) know Because I Know” and we developed it with networks of professional in mind (like LinkedIn). Each professional has skills, which allow her to perform her job. However, sometimes she is required to do something using a skill she doesn't have. In many cases, she might be able to perform the task anyway because she can ask for help in her social network. Think about any time you asked a friend to fix something in your script, or scrape some data, or patch a leaking water pipe.

Of course, if the task you need to perform requires only knowledge you have, you can do it quickly. Every level of social interaction you add will slow you down. If you're a computer scientist you can

¹⁵ Arda Halu, Raúl J Mondragón, Pietro Panzarasa, and Ginestra Bianconi. Multiplex pagerank. *PLoS one*, 8(10):e78293, 2013

¹⁶ Leo Katz. A new status index derived from sociometric analysis. *Psychometrika*, 18(1):39–43, 1953

¹⁷ Michele Coscia, Giulio Rossetti, Diego Pennacchioli, Damiano Cecarelli, and Fosca Giannotti. You know because i know: a multidimensional network approach to human resources problem. In *Proceedings of the 2013 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, pages 434–441. ACM, 2013b

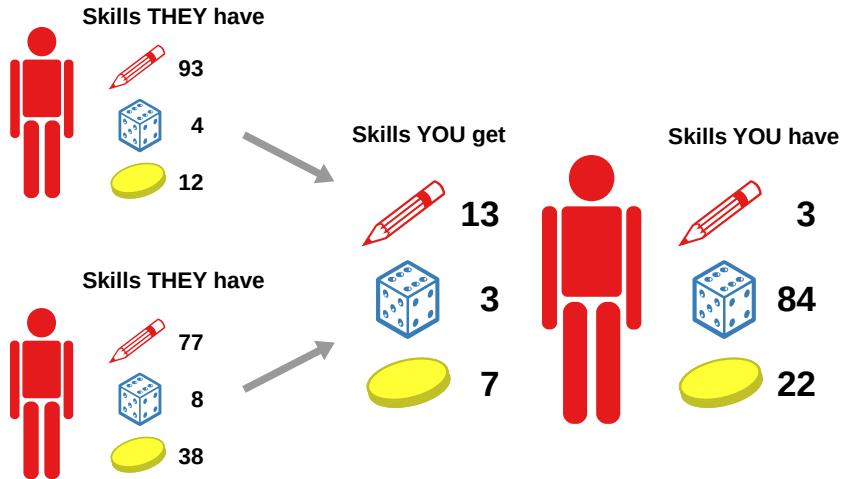


Figure 14.8: How UBIK works. Each node (you) has different proficiency for different skills, here we have three: writing (red), statistical analysis (blue), and finance (yellow). But what each node really can do is the sum of what it knows plus a combination of what its neighbors know. You get from your social network some skills, in this case the sum of their skills raised to the power of $-1/l$, where l is the degrees of separation plus one. So, in this case, the neighbors provide 13 writing skill points, because $(93 + 77)^{1/2} \sim 13$.

think about this as memory layers. What you know is in your brain, your cache: it is ready to run on your CPU. What your friends know is the main memory, the RAM. The main memory is slower than the cache, because the data need to travel from the memory to the cache before it can be used. Your friends' friends are like a hard disk, and the friends of the friends of your friends are the Internet: a limitless amount of distributed information that is hard to search and collect.

Figure 14.8 shows a vignette of this process.

So in UBIK we take the stance that a person's knowledge is the sum of her own knowledge plus some combination of her social networks and, ultimately, of mankind. This lofty philosophical picture boils down to just adding a few bells and whistles to Katz centrality. First, we don't use a simple graph, but a multilayer network. Different types of friends might have different levels of willingness or reactivity when asked to help. A colleague is just down the corridor, a close friend might want to do anything for you, that person you dated once during college maybe will pick up the phone if you call. So we have different adjacency matrices A with a different topology and a different coefficient favoring or hampering the centrality contribution.

Second, rather than giving a single centrality score to each node, we have multiple. Each node gets a different score for each skill. You might be a dragon when it comes to do multivariate regression analysis, but unable to make yourself understood in an email. As a consequence, the initial condition is also different. The skills aren't distributed equally in the network. The nodes don't all start from the same level in all skills. Each node has its own personal story, and might start with higher scores in some skills and lower in others.

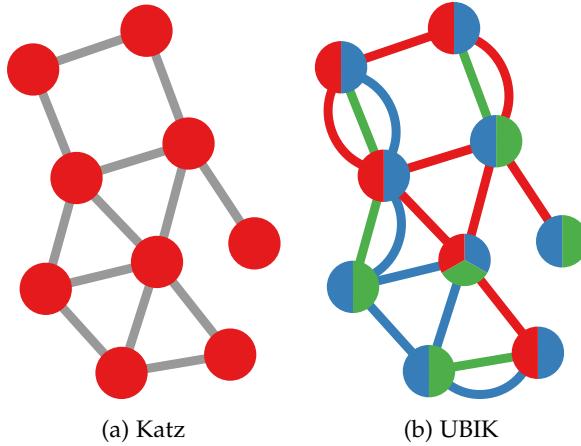


Figure 14.9: The difference in input between Katz and UBIK centralities. The node's color determines its initial condition: how much of a skill/centrality it possesses (different colors represent different skills/centralities). The edge's color determines its layer. Note how all nodes and all edges in Katz have the same color.

So the difference between UBIK and Katz is basically in the input data. Where Katz works with a single layer network, a single centrality measure, and a uniform initial condition, UBIK uses a multilayer network, with multiple centrality scores, initialized differently for different nodes. Figure 14.9 depicts this difference. Then the process is practically the same: direct neighbors have a big effect on your centrality scores and, as you go to more and more degrees of separation, the contributions fade away to zero. Sure, UBIK has to do this multiple times for each skill and needs the extra parameters to distinguish between different layers but, at the end of the day, UBIK is a glorified Katz centrality.

Where UBIK shines is in the analysis of so-called “expertise networks”: web-based communities of experts helping each other with problems related to their professions^{18,19}. One could also use it to investigate the question whether team formation is a process that happens better if it is organized from the top – like in organizations – or spontaneously from the bottom, like it happens for instance in large open source software projects²⁰.

Alpha

Another variant of eigenvector centrality is Bonacich’s Alpha centrality. If Katz wanted to penalize long walks, Bonacich wants to add an external source of importance to the node’s centrality²¹. Practically, we are saying that, to know how important a node is in a network, we don’t have to look exclusively at the topology of the network. The node might get its importance from somewhere else. A Web without Google would be poorer even if google.com would not be the most central node in the hyperlink network.

If, as we saw before, we can express the vanilla eigenvector centrality as an infinite sum:

¹⁸ Jun Zhang, Mark S Ackerman, and Lada Adamic. Expertise networks in online communities: structure and algorithms. In *Proceedings of the 16th international conference on World Wide Web*, pages 221–230, 2007a

¹⁹ Lada A Adamic, Jun Zhang, Eytan Bakshy, and Mark S Ackerman. Knowledge sharing and yahoo answers: everyone knows something. In *Proceedings of the 17th international conference on World Wide Web*, pages 665–674, 2008

²⁰ Christian Bird, David Pattison, Raissa D’Souza, Vladimir Filkov, and Premkumar Devanbu. Latent social structure in open source projects. In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, pages 24–35, 2008

²¹ Phillip Bonacich and Paulette Lloyd. Eigenvector-like measures of centrality for asymmetric relations. *Social networks*, 23(3):191–201, 2001

$$EC_v = \sum_{k=1}^{\infty} \sum_{u \in V} (A^k)_{uv},$$

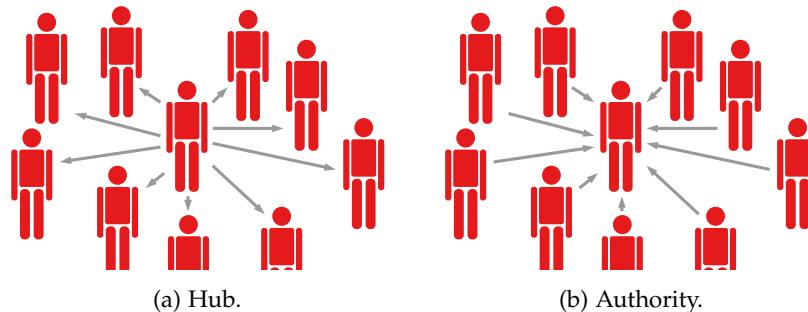
then we can express Alpha centrality as the same sum, plus an external source of non-network importance:

$$EC_v = (1 - \alpha)e_v + \sum_{k=1}^{\infty} \sum_{u \in V} \alpha(A^k)_{uv}.$$

Differently from Katz, α doesn't change as the length k increases: it just regulates how much weight we give to the traditional part of the eigenvector centrality. If $\alpha = 0$, then 100% of the node's importance comes from the vector e . Each entry e_v of e is the external importance of node v . In the Web network, Google's e_v would be through the roof. On the other hand, if $\alpha = 1$, this reduces to the classical eigenvector centrality.

14.5 HITS

HITS^{22,23} is an algorithm designed by Jon Kleinberg and collaborators to estimate a node's centrality in a directed network. It is part of the class of eigenvector centrality algorithms from Section 14.4, but it deserves its own section due to its interesting characteristics. Differently from other centrality measures, HITS assigns *two* values to each node. In fact, one can say that HITS assigns nodes to one of two roles – we will see more node roles in Chapter 15. The two roles are “hubs” and “authorities”.



In a sense, both hubs and authorities are central nodes in the network. However, when you're dealing with directed networks, there are two ways in which a core member of a community can play its role. A core member might be a person who maybe does not know many things, but knows the people who know them. You will go to this member with a question and she will *point to* someone who knows the answer. We call such linking resource a “hub”. Figure 14.10(a) provides an illustration.

²² Jon M Kleinberg, Ravi Kumar, Prabhakar Raghavan, Sridhar Rajagopalan, and Andrew S Tomkins. The web as a graph: measurements, models, and methods. In *International Computing and Combinatorics Conference*, pages 1–17. Springer, 1999.

²³ Jon M Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM (JACM)*, 46(5):604–632, 1999.

Figure 14.10: Hubs and authorities in directed networks.

The converse role of a hub is an authority. This is in principle the exact opposite of a hub – although it's possible for a node to be partly a hub and partly an authority at the same time –: this person might not know many people in the social circle, but she has mastered her own topic of specialization. Everybody knows that, and so she is *pointed by* everyone when someone asks about that particular topic. This happens because she is an “authority” on the subject. Figure 14.10(b) provides an illustration.

Hubs and authorities are an instance in which the quantitative approach of the centrality measures and the qualitative approach of the node roles meet. There is a way to estimate the degree of “hubbiness” and “authoritativeness” in a network. This is what the HITS algorithm does. The underlying principle is very simple. A good hub is a hub that points to good authorities. A good authority is an authority which is pointed by good hubs. These recursive definitions can be solved iteratively – or, more efficiently, with clever linear algebra – and they eventually converge.

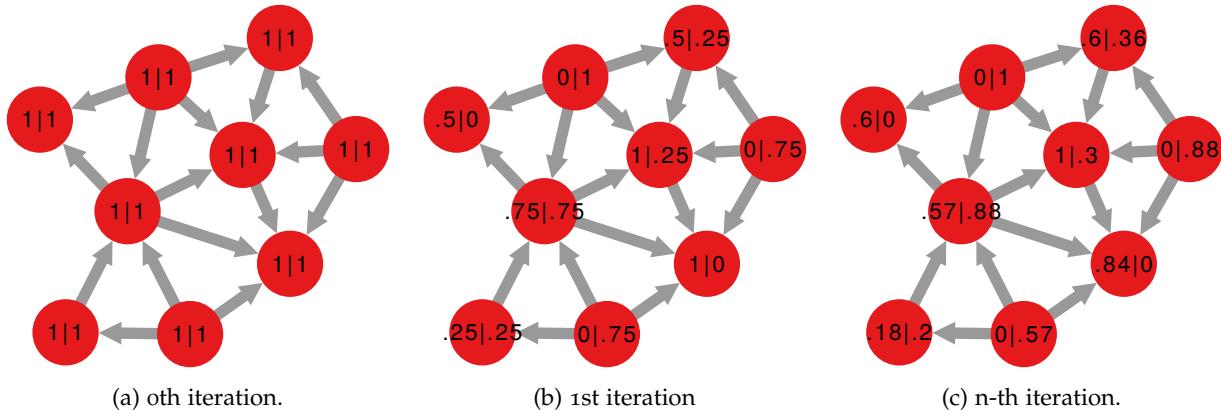


Figure 14.11 shows the progress when calculating the measure. Before the first iteration we assume that each node is equal. They thus have all the same hub score and authority score, equal to one – Figure 14.11(a). At each iteration, we sum all the incoming hub scores of a node to determine its authority score. At the same time, we sum all the outgoing authority scores of a node to obtain its new hub score. We then normalize so that the maximum hub and authority score is one. At the first iteration – Figure 14.11(b) – hub and authority scores are equivalent to a normalized out- and in-degree, respectively.

After a sufficient number of iterations – Figure 14.11(c) – the scores stabilize. We can see that nodes with the same in-degree can have different authority scores – the same holds for hub scores. Consider the two nodes with in-degree four: one has the maximum score of one, while the other has a score of 0.84. This is because the more

Figure 14.11: A sample progression of the HITS algorithm to estimate hub and authority scores. Node labels are their authority (left) and hub (right) scores, separated by a pipe.

authoritative node has, on average, incoming connections from more reputable hubs.

HITS is an important algorithm in the computer science portion of the network analysis community. It was modified and extended in a number of ways, notably to work on multilayer networks²⁴, enabling topic-dependent hub-authority scores. SALSA²⁵ is also a related method.

14.6 Harmonic

PageRank solves the problem of networks with multiple connected components. This is a common problem to have: all centrality measures based on shortest paths or random walks are ill defined when your network has pairs of unreachable nodes. This includes closeness, betweenness, reach, ... practically everything. But PageRank and its teleportation trick is not the only way to deal with multiple components.

The crux of the issue is that you cannot compare the closeness centrality of two nodes from different connected components, because one might have a higher closeness simply because its connected component is smaller. One approach to fix this issue is to consider the component size in the measure. We want the desirable property of saying that a central node in a large component is more important than a central node in a smaller component. Lin's centrality²⁶ achieves this by multiplying the closeness centrality of a node by the size of its connected component, which – incidentally – just means to square the numerator:

$$LC_v = (|V_v| - 1)^2 / \sum_{u \in V_v} |P_{vu}|,$$

with $V_v \subseteq V$ here being the set of nodes part of the component in which v resides.

Harmonic centrality represents another alternative which has been discussed in many slight different variations and scenarios^{27,28,29,30,31}. In practice, you calculate the harmonic mean of all distances – even those between unreachable nodes. Thus:

$$HC_v = \sum_u \frac{1}{|P_{vu}|}.$$

The harmonic centrality handles unreachable nodes properly, based on the assumption that $1/\infty = 0$.

²⁴ Tamara Kolda and Brett Bader. The tophits model for higher-order web link analysis. In *Workshop on link analysis, counterterrorism and security*, volume 7, pages 26–29, 2006

²⁵ Ronny Lempel and Shlomo Moran. Salsa: the stochastic approach for link-structure analysis. *ACM Transactions on Information Systems (TOIS)*, 19(2):131–160, 2001

²⁶ Nan Lin. *Foundations of social research*. McGraw-Hill Companies, 1976

²⁷ Massimo Marchiori and Vito Latora. Harmony in the small-world. *Physica A: Statistical Mechanics and its Applications*, 285(3-4):539–546, 2000

²⁸ Yannick Rochat. Closeness centrality extended to unconnected graphs: The harmonic centrality index. Technical report, 2009

²⁹ Paolo Boldi and Sebastiano Vigna. Axioms for centrality. *Internet Mathematics*, 10(3-4):222–262, 2014

³⁰ Edith Cohen and Haim Kaplan. Spatially-decaying aggregation over a network. *Journal of Computer and System Sciences*, 73(3):265–288, 2007

³¹ Raj Kumar Pan and Jari Saramäki. Path lengths, correlations, and centrality in temporal networks. *Physical Review E*, 84(1):016105, 2011

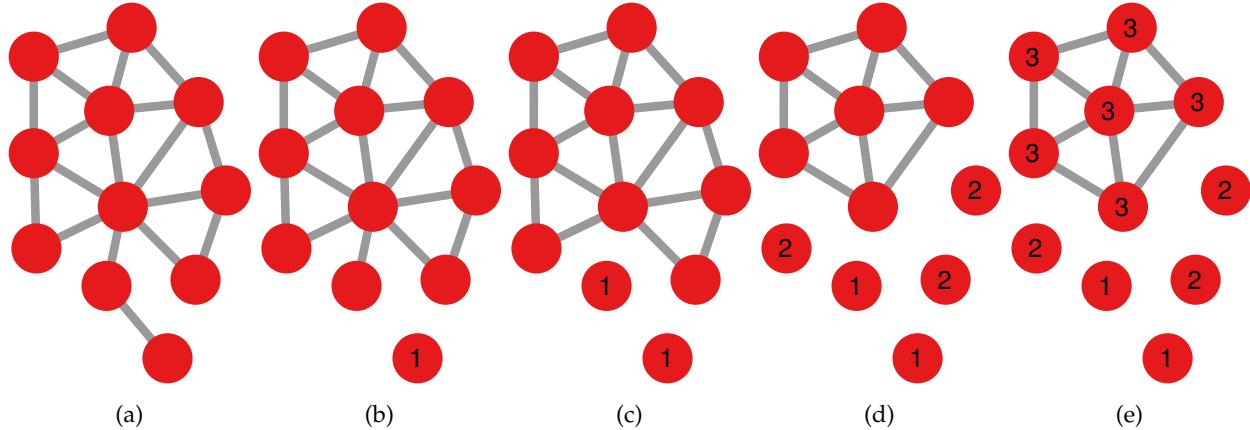
14.7 k -Core

When it comes to node centrality, one common term you'll hear thrown around is one of "core" node. This is usually a qualitative distinction – see Chapter 32, but sometimes we need a quantitative one. With k -core centrality we look for a way to say that a node is "more core" than another. A k -core in a network is a subset of its nodes in which all nodes have at least k connections to each other³². A connected component of a network is always a 1-core: each node in the component has at least one connection to the rest of the component. In a 2-core, each node must have at least two connections to the other nodes in the 2-core.

One can easily identify the k -core of a network via the k -core decomposition algorithm³³. In Figure 14.12 we represent a stylized version of it. Figure 14.12(a) shows the original network.

³² Stephen B Seidman. Network structure and minimum degree. *Social networks*, 5(3):269–287, 1983

³³ Vladimir Batagelj and Matjaz Zaveršnik. An $O(m)$ algorithm for cores decomposition of networks. *arXiv preprint cs/0310049*, 2003



The first step is identifying the nodes with degree one. They are labeled as part of the 1-core of the network, and removed from the structure. – Figure 14.12(b). We need to apply this step recursively: there could be nodes that originally had degree two, but now have lower degree because we removed one of their neighbors (or both!). Also these nodes are part of the 1-core of the network – Figure 14.12(c).

Once the minimum degree in the network is higher than one we can proceed to the next phase. In this phase we identify the nodes that are part of the 2-core of the network. These are, unsurprisingly, the nodes with degree two – and all nodes whose degree lowers to two or less once we remove their neighbors during this step (Figure 14.12(d)). We continue the procedure to detect 3-, 4-, ..., k -cores until there are no remaining nodes in the network – Figure 14.12(e).

Note that the k -core decomposition approach is only the most famous among many similar which define a structure of interest and

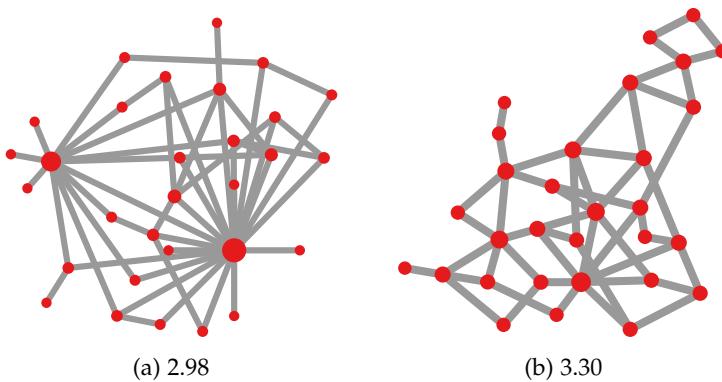
Figure 14.12: The steps to determine the k -core value for each node in the network. (a) The starting network. (b-e) The steps of the algorithm.

use it to define a centrality measure. Among popular examples we find D-cores³⁴ (for directed networks), k-shells³⁵, k-coronas³⁶, and more.

14.8 Centralization

This chapter is all about knowing which nodes are central in a network. So it is a node-centric chapter. To wrap it up, let's change the perspective a little. Let's see how node centrality can say something about your network as a whole. This would be the *centralization* of the network. A network is centralized when there is one node in it that is so much more central than everything else. Consider a star, where one node is in the middle, it is connected to every other node in the network and there are no other connections between its neighbors. A network cannot get more centralized than that³⁷.

There are two main ways to detect centralization. The first approach is information-theoretic. You can look at the degree distribution and transform it into a probability distribution. This distribution answers the question “What is the probability that an edge will be attached to this node?”. For instance, if you have the degree sequence [4, 2, 2, 1, 1], you divide every entry by the sum: [0.4, 0.2, 0.2, 0.1, 0.1]. Now you can calculate Shannon’s information entropy. As Figure 14.13 shows, a more centralized network will generate lower entropies³⁸ given that there are few nodes connected to everything.



Matthias Dehmer and Abbe Mow
showitz. A history of graph entropy
measures. *Information Sciences*, 181(1):
57–78, 2011

Figure 14.13: Two networks with different levels of centralization. The node size is proportional to its degree. Entropy values in the captions.

One issue with this approach is that it's not immediately obvious how centralized a network is by simply looking at the entropy value. Larger networks will generate higher entropies without necessarily being more centralized, because you need more bits to encode more entries – check Section 3.5 for a refresher. You could normalize the entropy with the maximum entropy you'd get with $|V|$ entries – the number of nodes in your network – which is $\log |V|$. However, it's not that simple. The maximum entropy for a graph of 30 nodes like the ones in Figure 14.13 is 3.40. This means that the apparently

centralized graph in Figure 14.13(a) still has 87% of the maximum entropy, which would lead you to conclude that it's decentralized. So you also need to figure out what is the *minimum* possible entropy given a graph, which can be tricky^{39,40}.

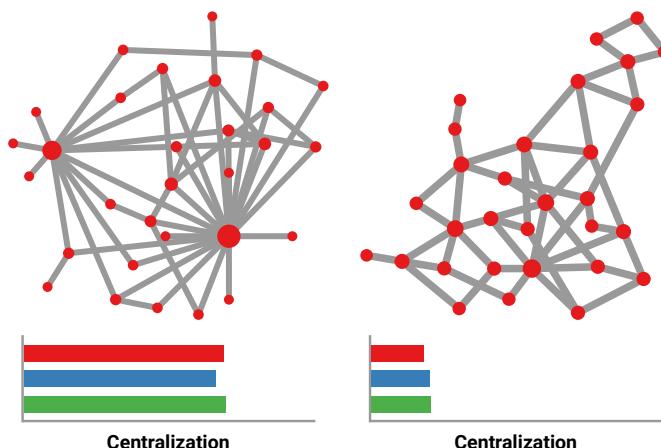
Note that Shannon's entropy is not the only option if you want to go this way: there's also Von Neumann's entropy^{41,42}. However, it's not the easiest thing in the world to calculate or interpret^{43,44}. And it tends to return similar values to Shannon's entropy anyway, so I personally stick with the version of entropy I explained.

One can also characterize the entropy of network ensembles^{45,46} – i.e. what is the general entropy of all graphs generated with a given set of rules (which we'll see in Part V). This leads to interesting discoveries, such as that most naturally occurring graphs are part of families with lower entropy, which is not what you'd expect given the first expectation about entropy in the physical world – entropy tends to be maximized.

If you don't want to go the information-theory route, you can calculate any centrality measure and see how skewed the maximum is with everything else. The procedure follows two steps. First, you sum the centrality differences between the most central node in the network and all other nodes. Then you calculate what would be the largest theoretical sum of differences in networks of comparable size. Usually the maximum is obtained by a star graph with the same number of nodes of your original network. The ratio between the two is the degree of centralization.

Note that, depending on the centrality measure you picked, you're going to obtain different results. Figure 14.14 shows a case using degree, betweenness, and closeness centrality.

While the centralization measures agree that the network on the left is more centralized than the network on the right, the levels of



³⁹ Stijn Cambie and Matteo Mazzamurro. Resolution of yan's conjecture on entropy of graphs. *arXiv preprint arXiv:2205.03357*, 2022

⁴⁰ Stijn Cambie, Yanni Dong, and Matteo Mazzamurro. Extremal values of degree-based entropies of bipartite graphs. *Information Sciences*, 676:120737, 2024

⁴¹ Filippo Passerini and Simone Severini. The von neumann entropy of networks. *arXiv preprint arXiv:0812.2597*, 2008

⁴² David Simmons, Justin Coon, and Animesh Datta. The quantum theil index: characterizing graph centralization using von neumann entropy. *arXiv preprint arXiv:1707.07906*, 2017

⁴³ Lin Han, Francisco Escolano, Edwin R Hancock, and Richard C Wilson. Graph characterizations from von neumann entropy. *Pattern Recognition Letters*, 33(15):1958–1967, 2012

⁴⁴ Giorgia Minello, Luca Rossi, and Andrea Torsello. On the von neumann entropy of graphs. *Journal of Complex Networks*, 7(4):491–514, 2019

⁴⁵ Ginestra Bianconi. Statistical mechanics of multiplex networks: Entropy and overlap. *Physical Review E*, 87(6):062806, 2013

⁴⁶ Kartik Anand and Ginestra Bianconi. Entropy measures for networks: Toward an information theory of complex topologies. *Physical Review E*, 80(4):045102, 2009

Figure 14.14: The bar chart below the network indicates its three centralization values according to degree (red), betweenness (blue) and closeness (green) centrality.

centralization differ. For the network on the left, degree centrality overestimates centralization when compared to betweenness centrality. For the network on the right, the opposite happens.

There are variants of centralization measures. For instance, you can allow the graph to have multiple central points, not just the one with the maximum centrality. Once you add this degree of freedom, you can also ask yourself: given that the graph has multiple centers, are these centers close together, or are they scattered far apart? In the former case the graph is more centralized than in the latter.

14.9 Summary

1. We've seen many alternatives to the degree to estimate a node's importance. Many are based on shortest paths. The first measure is closeness centrality, answering the questions: how far is on average a node from every other node in the network? Betweenness centrality, instead, asks: how many shortest paths would become longer if this node were to disappear?
2. Alternatively, you can look at random walks, since they're less computationally expensive to calculate. You can calculate a family of eigenvector centralities, of which PageRank is one of the most famous examples.
3. HITS is another famous eigenvector centrality measure for directed networks, which divides nodes in two classes: hubs, who dominate out-degree centrality; and authorities, who dominate in-degree centrality.
4. Harmonic centrality is a version of closeness centrality which solves the issue of networks with multiple connected components. In such networks, there are pairs of nodes that cannot reach each other, thus other approaches based on shortest paths and random walks wouldn't work.
5. k-Core decomposition also works with networks with multiple components. It recursively removes nodes from the network with increasing degree thresholds. At iteration k , we say that surviving nodes are part of the k th core.
6. Regardless of your centrality measure, you can estimate how centralized your network is by comparing the highest observed centrality with the theoretical maximum centrality of a network with the same number of nodes: a star graph.

14.10 Exercises

1. Based on the paths you calculated for your answer in the previous chapter, calculate the closeness centrality of the nodes in Figure 13.12(a).
2. Calculate the betweenness centrality of the nodes in Figure 13.12(a). Use to your advantage the fact that there is a bottleneck node which makes the calculation of the shortest paths easier. Don't forget to discount paths with alternative routes.
3. Calculate the reach centrality for the network in <http://www.networkatlas.eu/exercises/14/3/data.txt>. Keep in mind that the network is directed and should be loaded as such. What's the most central node? How does its reach centrality compare with the average reach centrality of all nodes in the network?
4. What's the most central node in the network used for the previous exercise according to PageRank? How does PageRank compares with the in-degree? (for instance, you could calculate the Spearman and/or Pearson correlation between the two)
5. Which is the most authoritative node in the network used for the previous question? Which one is the best hub? Use the HITS algorithm to motivate your answer (if using networkx, use the scipy version of the algorithm).
6. Based on the paths you calculated for your answer in the previous chapter, calculate the harmonic centrality of the nodes in Figure 13.12(a).
7. Calculate the k-core decomposition of the network in <http://www.networkatlas.eu/exercises/14/7/data.txt>. What's the highest core number in the network? How many nodes are part of the maximum core?
8. What's the degree of centralization of the network used in the previous question? Compare the answer you'd get by using, as your centrality measure, the degree, closeness, and betweenness centrality.

15

Node Roles

Sometimes, the differences between nodes can be estimated quantitatively. A person is measurably more or less connected in a social network. That is what we described in the previous chapter: if you can estimate the importance of the node (number of connections, centrality, etc), you do so by calculating the corresponding quantitative measure (degree, betweenness, etc).

Sometimes you cannot put a number to what you're trying to describe. What the person is doing in the social network does not have a quantity, but a quality: she is playing a specific role, which does not have a countable result. This could be explicitly represented in your data as a node attribute (Section 7.5): for instance, in a corporate network, nodes might be explicitly labeled as managers, executive, technicians, etc. But one thing is clear: in most cases, nodes perform different roles in the network.

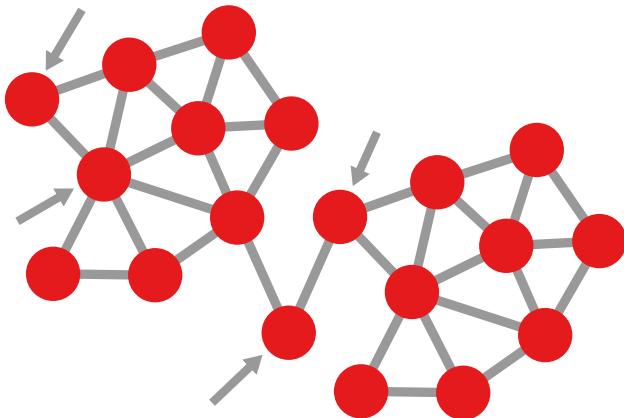
If you don't have explicit qualitative data, you might want to put a label on the nodes based on the structural network data. Rather than being a characteristic of the person by itself, the node role is determined by her position in the network.

There are many ways to define node roles, dependent on the aspect of the network you want to describe. The main split in the literature is on the type of procedure you're following: unsupervised or supervised. In unsupervised role learning, no node in your data has a role and you're making up your own definition of roles depending on what's meaningful to you. This is the classical approach, which I dissect in Sections 15.1 and 15.2. In supervised node learning, you already have partially labeled data and you want to figure out what are the underlying rules determining the node roles. This is the theme of Section 15.3.

15.1 Classic Node Classification

In this section we introduce the concept of node roles by picking network communities as our focus, just to give an example. If your focus is different, you will probably define different roles – for instance, you could look at paths in a directed network¹. In fact there are countless centrality measures developed to identify specific node roles in complex networks².

Let's consider the case of social circles. A social circle is a group of people interacting with each other because of shared interests and/or characteristics. We can say that Figure 15.1 shows two connected social circles. The communities are not completely homogeneous: they have structure. They have a boundary, members that are more or less central, and outsiders connecting to them. We could define four roles in this network: brokers, gatekeepers, core, and periphery (the latter two not to be confused with the core-periphery mesoscale structure that we will see in Chapter 32).



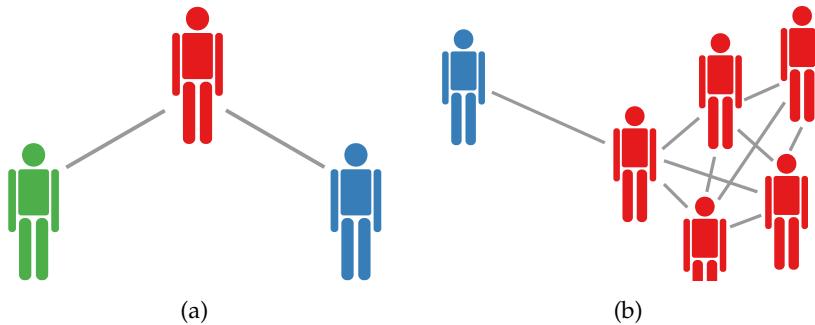
¹ Kathryn Cooper and Mauricio Barahona. Role-based similarity in directed networks. *arXiv preprint arXiv:1012.2726*, 2010

² Stephen P Borgatti and Martin G Everett. A graph-theoretic perspective on centrality. *Social networks*, 28(4): 466–484, 2006

Figure 15.1: Two hypothetical social circles: groups of nodes connected to each other on the left and on the right, with an intermediary in the middle. Are the nodes highlighted by the gray arrows all performing the same “role” in the network?

Broker. Suppose we have two social circles. If the two communities do not share any member it means that they cannot communicate. However, sometimes you have people who are not part of either community – because they only have few connections to each of them – but they still have friends in both. These nodes can enable communication to happen between the communities, and so they are performing the role of information brokers. Figure 15.2(a) provides an illustration.

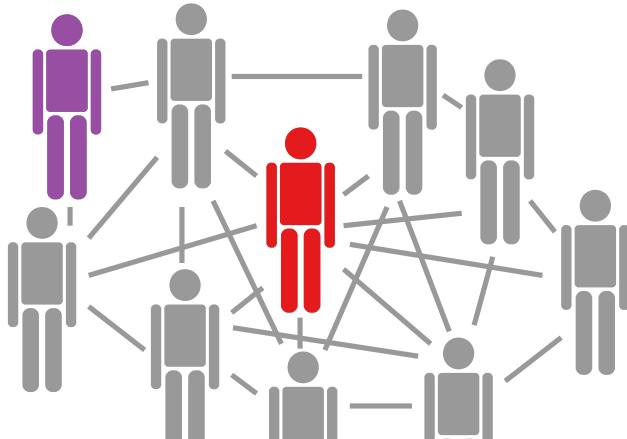
Gatekeeper. It is rare for a social circle to be completely isolated from the rest of society. Some of its members still have connections with people outside the community. If they do, they are managing how the community relates to society: both in the flow of information getting inside the community from the outside, and in what the community sends outside. These nodes are the gatekeepers. Figure



[15.2\(b\)](#) provides an illustration.

Core. Some members of the community have a more central role than others. They connect exclusively with other members of the community, without establishing relations with outsiders. They also have many connections. They are the heart of the social circle, and thus composing its core. Figure [15.3](#) provides an illustration.

Periphery. The other side of the coin of core members. A peripheral member does not have many connections in the community. Differently from brokers and gatekeepers, this is not because they also have connections to the external world. They just do not have many relations, and all they have are in their own community. They are thus peripheral to it. Figure [15.3](#) provides an illustration.



Rolx³ is one of the best known machine learning approaches for the extraction of node roles in complex networks – a topic we'll greatly expand on in Part XI of the book. The way it works is by representing nodes as vectors of attributes. Attributes can be, for instance, the degree, the local clustering, betweenness centrality, and so on. In practice, you decide which node features are relevant to determine the roles your nodes should be playing in the network. This means that, selecting the right set of features, you can recover all

Figure 15.2: (a) An example of a broker. Color indicates the membership to a social circle. The red node isn't part of the two social circles it connects, so it brokers information between them. (b) An example of a gatekeeper. The blue person is not part of the red community. The member of the community to which it connects is managing the information access to the community. Thus, it is gatekeeping it.

Figure 15.3: Core and periphery nodes in a community. The red element of the social circle is very embedded in it: she is a core member. On the other hand, the purple person only has two friends in the social circle and no other relation. She is in the periphery.

³ Keith Henderson, Brian Gallagher, Tina Eliassi-Rad, Hanghang Tong, Sugato Basu, Leman Akoglu, Danai Koutra, Christos Faloutsos, and Lei Li. Rolx: structural role extraction & mining in large graphs. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1231–1239. ACM, 2012

the roles I discussed so far – core, periphery, broker, gatekeeper.

Rolx works in the way you would expect from a standard machine learning framework. The features are represented in a space where redundancies are eliminated, for instance by running principal component analysis (Section 5.6). This space is then fed to a classifier, which tries to find the salient differences between different vector prototypes. These classes are the different roles a node can play.

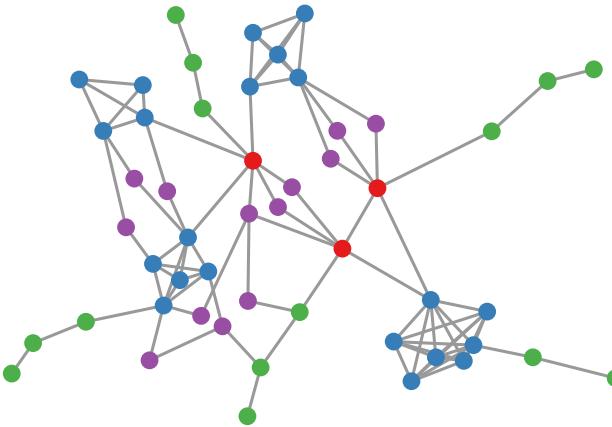


Figure 15.4: An example of Rolx output. Node color encodes the node's role.

Figure 15.4 shows an example of Rolx's output. The network represents co-authorship in a scientific network. Nodes are scientists connected if they collaborated on a paper. Rolx is able to find the different roles played by different authors. Specifically, authors found four roles of interest:

- Bridges (red): these are the hubs keeping the network together;
- Tightly knit (blue): these are the authors who have a reliable group of co-authors, and are usually embedded in cliques;
- Pathy (green): authors who are part of long stretches;
- Mainstreram (purple): everything else.

Note that, with Rolx, you can also estimate how much each role tends to connect with nodes in a similar role. For instance, by their very nature, bridges tend to connect to nodes with different roles, while tightly knit nodes band together. This is related to the concepts of homophily and disassortativity, which we'll explore in Chapter 30.

15.2 Node Similarity

Structural Equivalence

When two nodes have the same role in a network they are, in a sense, similar to each other. Researchers have explored this observation

and derived measures of “node similarity”. We can also call this “Structural Equivalence”, as two nodes with similar roles in similar areas of the network are keeping the network together in the same way. In fact, *structural equivalence* is the stricter test of node similarity, which we can relax to obtain alternative measures.

In this part of the book we have assumed a structural view of nodes, unless otherwise specified. What this means is that, for betweenness centrality or the k-core algorithm, nodes don’t have metadata, or internal statuses, or attributes. The only way to tell the difference between one node and another is by looking at their degrees and the nodes they connect to.

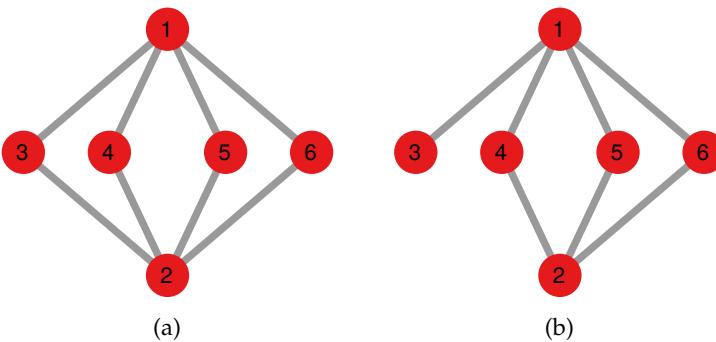


Figure 15.5: (a) An example of two structurally equivalent nodes (nodes 1 and 2). (b) Here, nodes 1 and 2 are not structurally equivalent, because node 1 has a neighbor that node 2 does not.

This is important to point out in this section, because it helps understanding the definition of structural equivalence. For two nodes to be structurally equivalent they have to be connected to the same neighbors⁴. If they do, they are indistinguishable from one another, therefore they cannot be any more similar. Consider Figure 15.5(a): nodes 1 and 2 have the same neighbors and no other additional one. If I were to flip their IDs, you would not be able to tell. There is no extra information for you to do so, because all you have is their neighbor set.

On the other hand, we can tell the difference between nodes 1 and 2 in Figure 15.5(b). That is because we know that node 1 also connects to node 3, which node 2 does not. So the two nodes are not structurally equivalent. You can use any vector similarity measure to estimate structural equivalence. For instance, you can calculate the number of common neighbors or the Jaccard similarity of the neighbor sets between u and v . In Figure 15.5(b), nodes 1 and 2 have three common neighbors out of four possible, thus their structural equivalence is 0.75.

Alternatively, one could use cosine similarity⁵, Pearson correlation coefficients, or inverse Euclidean distance. In all these cases, you have to transform the neighbor set into a numerical vector. If you sort the nodes consistently, each node can be represented as a vector of

⁴ Robert A Hanneman and Mark Riddle. Introduction to social network methods. 2005

⁵ Gerard Salton. Automatic text processing: The transformation, analysis, and retrieval of. Reading: Addison-Wesley, 169, 1989

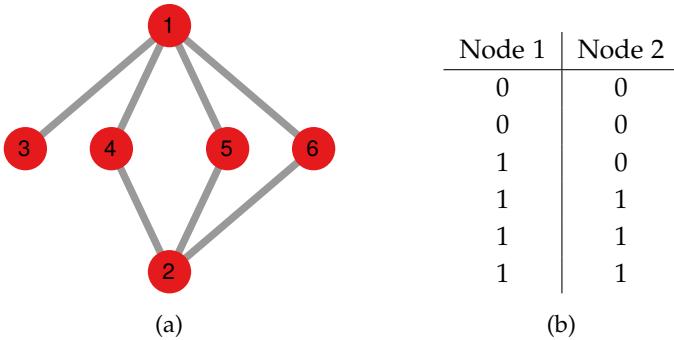


Figure 15.6: (a) A simple graph.
(b) The adjacency vector representations of node 1 and node 2.

zeros and ones. Zeros correspond to nodes not connected to u , while ones are u 's neighbors. These are the rows in the adjacency matrix corresponding to the nodes, as Figure 15.6 shows. You can input the vectors corresponding to u and v to any of the mentioned measures and obtain their structural equivalence. For instance, the Pearson correlation coefficient of nodes 1 and 2 in Figure 15.6 is around 0.7.

Automorphic Equivalence

Automorphic equivalence is a more relaxed version of structural equivalence. To understand it, we need to introduce the concepts of *isomorphism* and *automorphism*. We call two graphs “isomorphic” if they have the same topology: the graph in Figures 15.5(a) and 15.5(b) would be isomorphic if you were to add an edge in Figure 15.5(b) between nodes 2 and 3. As a mnemonic trick: “iso” = same, and “morph” = “shape” – two isomorphic graphs have the same shape. We’ll see how to determine whether two graphs are isomorphic in Section 41.3.

A graph is “automorphic” if it is isomorphic with itself. This means that you can shuffle all node IDs of your graph such that you preserve the neighborhoods of all nodes – of course we care about non-trivial automorphisms, you can always find an automorphism by not swapping anything. If node 1 was connected only to nodes 2 and 3 in G , you can only swap its ID with another node that only has two connections and those connections lead to nodes that have swapped their IDs with nodes 2 and 3. The graph in Figure 15.7(a) is automorphic because we can swap around labels respecting this rule – as I do in Figure 15.7(b).

So, for automorphic equivalence, two nodes are equivalent if you can perform this re-labeling. To understand what this means consider nodes 1 and 2 in Figure 15.6(a). They are NOT automorphically equivalent because, if we swap their labels, there is no further relabeling we can do to render the graph isomorphic. We’re not allowed to

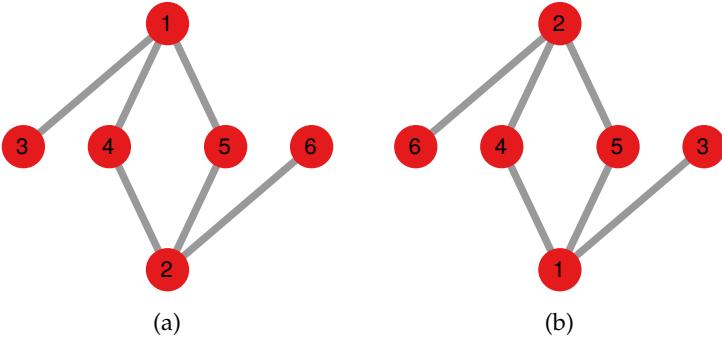


Figure 15.7: An example of relabeling of the graph to highlight the automorphic equivalence between nodes 1 and 2.

swap nodes 3 and 6, because they have a different number of neighbors. In Figure 15.7(a), instead, nodes 1 and 2 are automorphically equivalent. We can swap their labels, which forces us to swap node 3 and 6's labels too. The resulting graph, in Figure 15.7(b), is identical to the original.

In this case, nodes 1 and 2 are not structurally equivalent, because they both have neighbors that the other node doesn't have, but they are automorphically equivalent, because you can perform the re-labeling. Every structurally equivalent pair of nodes is also automorphically equivalent, but two automorphically equivalent nodes might not be structurally equivalent.

In an automorphic graph, there must be pairs of automorphic equivalent nodes by definition. In non-automorphic graphs, some nodes can still be automorphic equivalent if you can perform a local relabeling. You could imagine Figure 15.7(a) to be embedded in a larger non-automorphic graph, but that would not affect the automorphism between nodes 1 and 2.

While structural equivalence focused on nodes that had literally the same neighbors in the network, automorphic equivalence focuses on nodes that belong to the same local structure type, even if they don't have the exact same set of neighbors.

Regular Equivalence

The most relaxed variant of node similarity is *regular equivalence*. In regular equivalence, nodes can be equivalent to each other if they have connections to equivalent nodes. This is most easily understood in hierarchies. Consider Figure 15.8. In the figure, we can find three equivalence classes containing, respectively: {1}, {2,3}, and {4,5,6}. The third class is defined by those nodes connected to nodes of class two but without connections to class one. Class two connects to both class one and class three, even if the number of connections to the members of class three can vary. Class one is the mirror of class three:

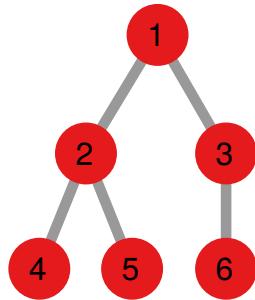


Figure 15.8: A graph with three regularly equivalent classes of nodes.

it connects to nodes in class two, but to no node in class three.

How can you quantify regular equivalence? One way is to realize this is a recursive problem: similar nodes connect to similar nodes^{6,7}. So, if we say our similarity score is σ , a matrix with a value per node pair, then we can say that two nodes u and v are similar if their neighbors are similar. In mathematical form:

$$\sigma_{uv} = \alpha \sum_{ij} A_{ui} A_{vj} \sigma_{ij},$$

so you are iterating over all pairs of u 's neighbors (i) and v 's neighbors (j) and using their similarity (σ_{ij}) to estimate u and v 's similarity. If you write this in matrix form, you get that $\sigma = \alpha A \sigma A$, which looks like an eigenvector problem – and, in fact, it is. To get the real similarity, you need to apply this formula many times, say l times. That is why you need an α in front: by applying this formula l times you are using paths of length l to estimate the similarity, but you care more about nodes that are closer to u and v and not those that are l hops away. If $\alpha < 1$, the $\alpha^l < \alpha^{l-1}$ and you get smaller contributions from nodes that are farther away. This should be familiar to you, because that's the same logic as Katz centrality we just saw in Section 14.4.

If you want to boost the similarity of a node with itself, you can always use the identity matrix: $\sigma = \alpha A \sigma A + I$. However, you should be careful because this measure as written here reduces to something similar to structural equivalence. If you run this formula on the network in Figure 15.8 you get that nodes 1, 4, 5 and 6 are all similar – in intuitive terms, you'd get that CEOs are similar to interns because they both connect to middle managers. Figure 15.9 shows you the similarity matrix you'd get, and you can see node 1 is as similar to node 4 as nodes 2 and 3 are to each other. So you need to ensure that your initial classes are respected somehow.

To sum up the difference between structural, automorphic, and regular equivalence, consider familial bonds. Two women with the same husband and the same children are structurally equivalent: they

⁶ Glen Jeh and Jennifer Widom. Simrank: a measure of structural-context similarity. KDD '02, New York, NY, USA, 2002. ACM. ISBN 158113567X. doi: 10.1145/775047.775126. URL <https://doi.org/10.1145/775047.775126>

⁷ Vincent D Blondel, Anahí Gajardo, Maureen Heymans, Pierre Senellart, and Paul Van Dooren. A measure of similarity between graph vertices: Applications to synonym extraction and web searching. *SIAM review*, 46(4): 647–666, 2004

| | | | | | |
|------|------|------|------|------|------|
| 0.24 | 0.00 | 0.00 | 0.06 | 0.06 | 0.08 |
| 0.00 | 0.29 | 0.06 | 0.00 | 0.00 | 0.00 |
| 0.00 | 0.04 | 0.27 | 0.00 | 0.00 | 0.00 |
| 0.03 | 0.00 | 0.00 | 0.22 | 0.05 | 0.01 |
| 0.03 | 0.00 | 0.00 | 0.05 | 0.22 | 0.01 |
| 0.04 | 0.00 | 0.00 | 0.01 | 0.01 | 0.24 |

have the same relationships with the same people (in fact, they would be the same person – although this is not a necessary requirement for structural equivalence). Two women can be automorphically equivalent if they have the same number of husbands and the same number of children. Finally, to be regularly equivalent to a married woman with children you have to be a married woman with children, even if you have a different number of relations – the woman with fewer husbands will definitely lead a less frustrating life.

There are many other measures of node similarity. Researchers usually define new ones to better solve a problem called “link prediction”, under the assumption that two similar nodes are more likely to connect to each other. We will see more node similarity measures than you want to know in Chapter 23. Node similarity can be used to estimate network similarity, which is the topic of Chapter 48.

15.3 Node Embeddings

So far we’ve been pretty rigid in the way we wanted to classify nodes into roles. Either we explicitly defined the roles with strict rules, or we adopted the similarity approach, finding which node plays a similar structural role to which other node. This is a sort of “zero-dimensional” approach, where everything collapses in a single label. One could use instead node embeddings, which determine the role of a node with a vector of numbers and then classifies the node with it. Recently, the most common way to discover such roles has become the use of graph neural networks. I will explain more in detail how they work much later in the book – if you think what follows is a bunch of gobbledegook, you should check out Part XI. Here I will just show the general shape of the problem they are solving and how it can be used to infer node roles.

When it comes to detect node roles, graph neural networks use machine learning techniques (Chapter 4) to learn a function classifying nodes^{8,9}. Figure 15.10 shows a general schema for graph neural networks. We start from some training data. This could be the graph itself with some nodes labeled and some not, or a graph or a collection of graphs with labeled nodes. We pass this graph as the input to the hidden layers. The role of the hidden layers is to learn a function

Figure 15.9: The regular equivalence of the nodes from the network in Figure 15.8. The rows and columns of the matrix are sorted according to the numerical id of the node.

⁸ Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, and Maosong Sun. Graph neural networks: A review of methods and applications. *arXiv preprint arXiv:1812.08434*, 2018

⁹ Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S Yu. A comprehensive survey on graph neural networks. *arXiv preprint arXiv:1901.00596*, 2019

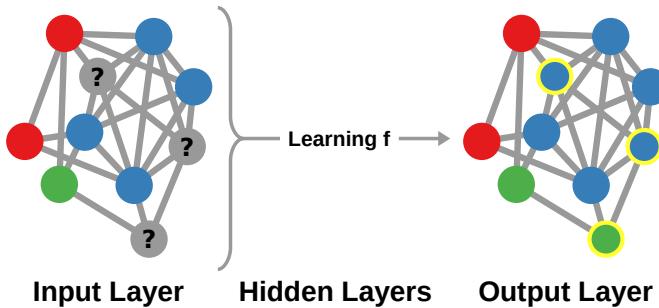


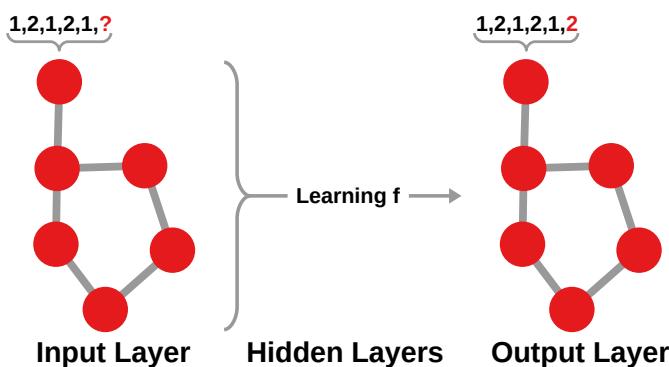
Figure 15.10: A general schema for graph convolutional learning. The gray nodes in the input network are unclassified. By learning the function f behind the classification of nodes, we can classify the rest of the network (yellow outline in the output layer).

f that can explain why each node has a specific label/value. Once f is learned, you will obtain the labels of the non-classified nodes, or you'll be able to classify a new, previously unseen, graph.

Typically, f assumes that each node can be represented by a vector, called state. We're going to see in Chapter 44 more than you want to know on how to make smart f s.

But modifying the way you learn f is not the only possible variant. You could also modify the input and output layers, to change the task itself. For instance, you could try to predict spatiotemporal networks^{10,11,12,13}: by having as input a dynamic network with changing node states, you could predict a future node state. Figure 15.11 shows a simplified schema for the task. Think, for instance, about traffic load: given the way traffic evolves, you want to be able to predict how many cars will hit a specific road straight (edge) or intersection (node).

Other possible applications are the generation of a realistic network topology (Chapter 18), the prediction of a link (Chapter 23), or summarizing the graph (Chapter 46). Given that these are not related to node roles, I'll deal with such applications in the proper chapters.



¹⁰ Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu. Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. *arXiv preprint arXiv:1707.01926*, 2017b

¹¹ Bing Yu, Haoteng Yin, and Zhanxing Zhu. Spatio-temporal graph convolutional networks: a deep learning framework for traffic forecasting. In *IJCAI*, pages 3634–3640. AAAI Press, 2018

¹² Sijie Yan, Yuanjun Xiong, and Dahua Lin. Spatial temporal graph convolutional networks for skeleton-based action recognition. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018

¹³ Ashesh Jain, Amir R Zamir, Silvio Savarese, and Ashutosh Saxena. Structural-rnn: Deep learning on spatio-temporal graphs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5308–5317, 2016

Figure 15.11: A schema for spatial-temporal neural networks. We have an activation timeline for each node (here showing only one). The task is predicting the activation state in the next timestep.

15.4 Summary

1. Going beyond node centrality, we can attach to nodes qualitative roles, rather than quantitative estimations of their importance. These qualitative roles are dependent on the node's position in the network topology. Traditionally, this is an "unsupervised" learning task, in which you don't know any node role and you're substantially inventing your own definition.
2. There are many ways to define roles, some well known are brokers – in between communities –, gatekeepers – on the border of a community –, and more. A popular algorithm to detect node roles is Rolx.
3. We can detect nodes playing the same role in a topology by estimating their structural similarity: their tendency of connecting to the same set of neighbors.
4. Node role could also be a supervised learning problem, where you have some node roles in your data and you want to discover the latent rules that determine them. Graph neural networks are a popular way to do so. In this technique, we don't have (necessarily) a definition of what the role is, but some already labeled data on which we can train the algorithm.

15.5 Exercises

1. For the network at <http://www.networkatlas.eu/exercises/15/1/data.txt>, I precomputed communities (<http://www.networkatlas.eu/exercises/15/1/comms.txt>). Use betweenness centrality to distinguish between brokers (high centrality nodes equally connecting to different communities) and gatekeepers (high centrality nodes connecting with different communities but preferring their own).
2. Use the network from the previous question to distinguish between core community nodes (high degree nodes with all their connections going to members of their own community) and peripheral community nodes (low degree nodes with all their connections going to members of their own community).
3. Calculate the structural equivalence of all pairs of nodes from the network used in the previous question. Which two nodes are the most similar? (Note: there could be ties)

Part V

Synthetic Graph Models

16

Random Graphs

To explore the properties of a network – the degree distribution, the clustering, the centrality of its nodes – there is one fundamental requirement. You have to have a network. If you don't have a network, you're going to look pretty silly when you try to analyze it¹. At the very beginning of network analysis, there was a widespread lack of data. Thus, some of the most brilliant mathematical minds in the field determined different ways to create synthetic network data by defining network models. These models are the subject of this part of the book.

There are fundamentally three reasons to generate synthetic data today. The first is explanatory in nature. After you analyze a bunch of real world networks, you may realize they all seem to have a common property. Maybe they have a broad degree distribution (Section 9.3), incredibly high clustering (Section 12.2), or they contain communities (Part X). And you ask yourself: how did these properties arise? You might want to apply very simple rules, to see if they can reproduce the property of interest. If you succeed, you might be closer to explain their origins. This is what I explore in Chapter 17.

The second reason is to have a way to test your algorithms and analyses. If that's your aim, you want to generate fake networks that are as similar as possible to real world networks. They must have the same properties, especially the ones that are important for testing your method. This is what I explore in Chapter 18.

The third and final class focuses on description. As I just said, you might find a network with a peculiar property. You might ask yourself not how this property arose, but if it is something you'd expect any network to have, given other characteristics on its topology. In other words, you want to estimate the statistical significance of that observation. It's pretty hard to talk about statistical significance when you have a single observation – a single network. So you might want to generate random networks with the same properties of your observed one and see if they all have that characteristic of interest.

¹ Just like searching in a dark room for a black cat that isn't there (https://en.wikipedia.org/wiki/Black_cat_analogy).

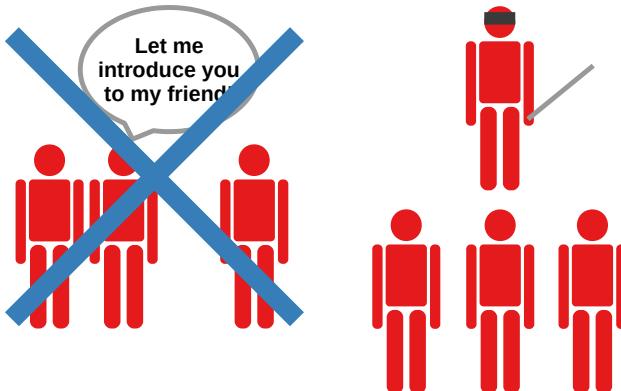
Chapter 19 is all about this task.

16.1 Building Random Graphs

Before diving deep into these more complex models, I need to spend some time with the grandfather of all graph models. It is the family of network generating processes created by Paul Erdős and Alfréd Rényi in their seminal set of papers^{2,3,4} (some credit goes also to Gilbert⁵ for a few variants of the model).

These are simply known colloquially as “Random graphs”. I can divide them fundamentally in two categories: $G_{n,p}$ and $G_{n,m}$ models. The way they work is slightly different, but their results are mathematically equivalent. The difference is there simply for convenience in what you want to fix: $G_{n,p}$ allows you to define the probability p that two random nodes will connect, while $G_{n,m}$ allows to fix the number of edges in your final network, m .

Ok, but... what is a random graph? We’re all familiar to the concept of random number. You toss a die, the result is random. But what does “random” mean in the context of a graph? What’s randomized here? For this chapter, I will answer these questions assuming uncorrelated random graphs – meaning that you can mentally replace “random” with “statistical independence”. This is not strictly speaking necessary: in the same way that you can study the statistics of correlated coins, you can study correlated random graphs. However, that would make for a nasty math, and it isn’t super useful for the aim of this book.



² P Erdős and A Rényi. On random graphs. *Publicationes Mathematicae Debrecen*, 6:290–297, 1959

³ Paul Erdos and Alfréd Rényi. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci.*, 5(1):17–60, 1960

⁴ Paul Erdos and Alfred Renyi. On random matrices. *Magyar Tud. Akad. Mat. Kutató Int. Közl.*, 8(455–461):1964, 1964

⁵ Edgar N Gilbert. Random graphs. *The Annals of Mathematical Statistics*, 30(4):1141–1144, 1959

Figure 16.1: In a social setting, friends introduce each other, thus edges are correlated: having a common friend increases the chance of being connected (see example on the left). In random graphs, edges are independent: blindfolded people establish the connections, as in the example on the right.

For network scientists, “random” applies to the edges. In a social setting, connections are not independent: it is more likely for you to know people your friends know, because they can introduce you. In random graphs, this is strictly forbidden, as I show in the vignette in Figure 16.1. If you are getting into a random graph and deciding where to put your new connection, you’re going in completely blind.

This means that you don't know anything about the connections that are already there.

If you're tired to read a book, this is the perfect occasion for a physical exercise. Here's a process you can follow to make your own random network. First, take a box of buttons and pour it on the floor. Yup, you heard me: just make a mess. Then take a yarn, cut some strings, and drop them on the buttons. The buttons are now your nodes, and the strings the edges. Congratulations! You have a random network! Time to calculate!

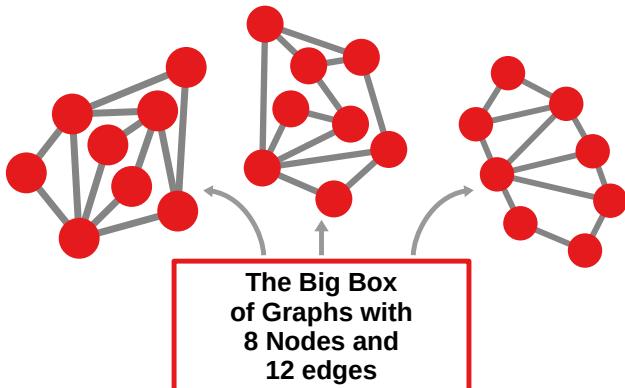


Figure 16.2: How a $G_{n,m}$ model works: the n and m parameters determine the box from which you will extract your graph. The box contains all possible graphs with n nodes and m edges.

Less facetiously, this process can illustrate clearly the distinction between $G_{n,p}$ and $G_{n,m}$ models. In $G_{n,m}$ you first fix the characteristics of the graph you want. You decide first how many nodes the graph should have – which is the n parameter –, say 8. This is the number of buttons. Then you fix the number of edges it should have – which is the m parameter –, say 12. This is the number of yarn strings you cut out. With those in mind, you go and take all possible graphs with n nodes and m edges, and you pick one at random – as Figure 16.2 shows. All the graphs, by the power of having the right number of nodes and edges, are equally likely to be the result of our operation. Alternatively, you could simply extract m random node pairs and connect them. The result will be the same.

In the $G_{n,p}$ variation we still say how many nodes we want: n . However, rather than saying how many edges we want, we just decide what's the probability that two nodes are connected to each other: p . It can be fifty-fifty, a coin toss. Then we consider all possible pairs of nodes, and for each one we toss the coin. If it lands heads we connect the nodes, if it lands tails we do not. $G_{n,p}$ is the perfect example of what we mean by “random graph”. Given a pair of nodes we toss the coin and if it lands on heads we connect them. Another pair, same procedure. The two tosses are independent: the fact that one landed on heads does not influence the other. The coin is also the same, so each edge has an equal probability to appear.

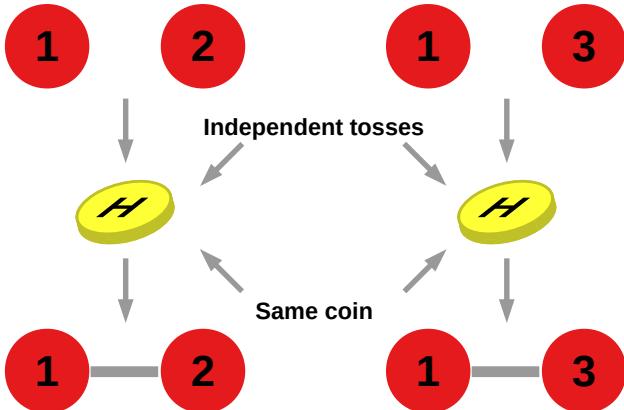


Figure 16.3: How a $G_{n,p}$ model works: the p parameter determines whether a node pair connects. The coin is not loaded, so it always has the same probability of landing on heads. The tosses are independent of each other, so the result of one doesn't affect the result of the other.

Figure 16.3 depicts the process. Note that throwing coins for each node pair isn't exactly the most efficient way to go about generating a $G_{n,p}$ – although I invite you to try. A few smart folks determined an algorithm to generate $G_{n,p}$ efficiently⁶.

Since $G_{n,m}$ and $G_{n,p}$ generate graphs with the same properties, it means that p and m must be related. Since the graphs have n nodes, we can derive the number of edges (m) from p . p is applied to each pair of nodes independently. We know how many pairs of nodes the graph has, which is $n(n - 1)/2$. Thus we have an easy equation to derive m from p : the number of edges is the probability of connecting any random node pair times the number of possible node pairs, or

$$p \frac{n(n - 1)}{2} = m.$$

This is useful if you use $G_{n,p}$ but you want to have, more or less, control on how many edges you're going to end up with.

By the way this gives you an idea of what's the typical density of a random $G_{n,p}$ graph. The density (see Section 12.1) is the number of links over the total possible number of links. We just saw that the number of links in a random graph is $p \frac{n(n - 1)}{2}$ and the total possible number of links is $\frac{n(n - 1)}{2}$. One divided by the other gives you p . So, if you want to reproduce the sparseness of real world networks, you can do that at will. Just tune the p parameter to be exactly the density you want.

In the following sections we explore each property of interest of random graphs, to see when they model the properties of real world networks well, and when they don't. The latter is the starting point of practically any subsequent graph model developed after Erdős and Rényi.

⁶ Vladimir Batagelj and Ulrik Brandes. Efficient generation of large random networks. *Physical Review E*, 71(3):036113, 2005

16.2 Degree Distribution

What's the expected degree of a node in a $G_{n,p}$ network with 9 nodes? Well, we start by looking at the first potential connection and toss a coin. We do that for every possible node in the network. Since this is independent, if the probability of connection is 50% and we make 8 tosses – one per potential neighbor –, on average we expect 8×0.5 head flips: 4, plus minus a small random fluctuation.

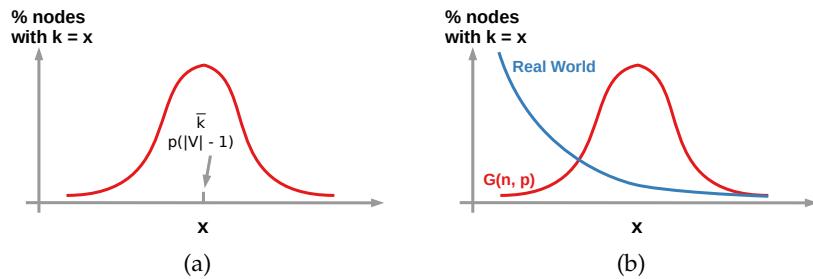


Figure 16.4: (a) The typical degree distribution of $G_{n,p}$ networks. (b) Comparing a $G_{n,p}$ degree distribution with one that you would typically get from a real world network.

A process like that generates a binomial distribution, where most nodes have a specific degree: p times the number of nodes minus one – because we avoid creating self loops. This is the average degree of the network. Figure 16.4(a) shows an example of a random degree distribution. Very few nodes have a much lower or much higher degree, because of how rarely you will get many heads or tails in a row from a fair coin.

Although the result is a binomial, many papers studying the degree distributions of random graphs use a Poisson distribution instead. They are practically identical, so this choice doesn't really matter – specifically a binomial becomes equivalent to a Poisson when the number of trials is very large and the expected number of successes remains fixed (see Section 3.2). We use a Poisson because the parameters regulating it make it easier to calculate the things that interest us – they all depend on a single parameter: \bar{k} , the average degree.

The random graph's degree distribution is at odds with most real world networks which, as we saw in Section 9.3, have many nodes with low degree. Moreover, the outliers with many connections – the hubs – in real networks have a much higher degree than the highest degrees you'll find in a random network. See Figure 16.4(b) for an example. So this is the first pain point of random networks: their degree distributions don't match the skewed ones we observe in the real world. Where is the real world broad degree distribution coming from? That's a question for Section 17.3.

16.3 Connected Components

Besides broad degree distributions, we're interested in giant components, because all real world networks seem to have them. Remember that giant components are components that include the vast majority of – or even all – the nodes of the network, see Section 10.4 for a refresher.

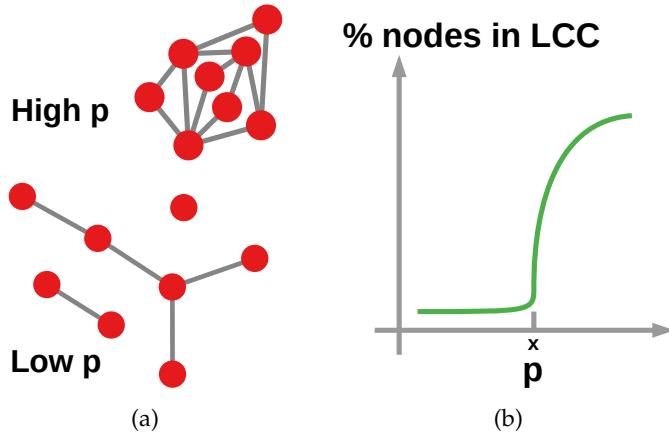


Figure 16.5: (a) The effect of p on $G_{n,p}$'s connectivity. (b) The evolution of the number of nodes in the largest connected component of $G_{n,p}$ as p changes.

In a $G_{n,p}$ graph, the presence of a giant component is dependent on p . As it is easy to see from Figure 16.5(a), if p is high there are a lot of edges, if it is low there are few and the graph could be disconnected. One could make a plot, showing for which values of p we have how many nodes in the largest connected component. One could expect a linear relationship, but that is not what we see: there is a special value of p for which we observe a phase transition – which I show in Figure 16.5(b) –: if p is lower than that value there is no giant component, if p is higher then most nodes are in the GCC^{7,8,9,10}.

Can we determine this magical value of p ? Logically, a node cannot be part of any component if it doesn't have an edge. If the average degree is less than one, many nodes won't have edges. Thus they cannot be part of the largest connected component. When the average degree is higher than one, the giant component appears. Thus the magical value of p is $1/|V|$.

So this is the value beyond which we start to see the largest connected component gobbling up the majority of the network's nodes. But there is another question. Is there a value of p such that *all* nodes are part of the giant component? This is equivalent of asking: when do we have fewer than one node without connections to the giant component? We start with the probability of connecting a node u with a node v . This is p . It follows that the probability of u and v **not** to connect is $1 - p$. Generalizing this, having no connection to a

⁷ Paul Erdős and Alfréd Rényi. On the strength of connectedness of a random graph. *Acta Mathematica Hungarica*, 12(1-2):261–267, 1961

⁸ Paul Erdos and Alfréd Rényi. On the existence of a factor of degree one of a connected random graph. *Acta Math. Acad. Sci. Hungar.*, 17(3-4):359–368, 1966

⁹ Dimitris Achlioptas, Raissa M D'souza, and Joel Spencer. Explosive percolation in random networks. *Science*, 323(5920):1453–1455, 2009

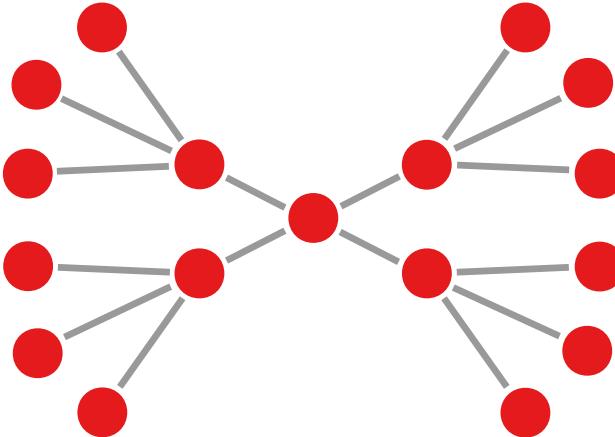
¹⁰ Raissa M D'Souza and Michael Mitzenmacher. Local cluster aggregation models of explosive percolation. *Physical review letters*, 104(19):195702, 2010

component with $|V|$ nodes is the same of landing tails for $|V|$ times in a row: $(1 - p)^{|V|}$. The number of nodes without a connection to a component with $|V|$ nodes is equal to make $|V|$ attempts – since we have $|V|$ nodes in our network –, all with that probability of success. The result is $|V|(1 - p)^{|V|}$.

Our original question was knowing when there are no nodes outside the GCC. This is equivalent to say that we want to have fewer than one node outside GCC. We just said that the number of nodes outside the GCC is $|V|(1 - p)^{|V|}$. Thus we want to know for which p we have $|V|(1 - p)^{|V|} < 1$. Which is $p = \ln |V| / |V|$.¹¹

Note that $\ln |V| / |V|$ tends to be a rather small number as $|V|$ grows, given that it pits a logarithmic growth in the numerator against a linear one in the denominator. Thus we discover that random graphs tend to have giant components as they grow larger, which is exactly what we see happening in real world networks. One point to team Erdős-Rényi!

16.4 Average Path Length



In a $G_{n,p}$ network, the number of nodes directly connected to a node is the average degree. This is the usual expectation as the connection probability is fixed to p , as we saw in Figure 16.4(a). In $G_{n,p}$ we can easily calculate the number of nodes at two hops from v . This is expected to be the average degree squared because, on average, each neighbor gives you access to its average degree number of neighbors – see Figure 16.6 for an example. This goes on for any number of hops l . The number of nodes at l hops away is \bar{k}^l – remember that \bar{k} indicates the average degree of the network.

If we know when $\bar{k}^l = |V|$, then we know the average path length of $G_{n,p}$: $\ln |V| / \ln(|V|p)$ ^{12,13}. Note that this is relatively short, unless

¹¹ The full mathematical derivation rests on the assumption that $(1 - x/n)^n \sim e^{-x}$ if n is large. At that point the derivation follows: $|V|(1 - p)^{|V|} = |V| \left(1 - \frac{|V|p}{|V|}\right)^{|V|} = |V|e^{-|V|p}$. If $|V|e^{-|V|p} = 1$, as we said we want in the text, then:
 $|V|e^{-|V|p} = 1$
 $e^{-|V|p} = |V|^{-1}$
 $e^{|V|p} = |V|$
 $|V|p = \ln |V|$
 $p = \ln |V| / |V|$.

Figure 16.6: A simplification of what happens to the average path length in $G_{n,p}$ random graphs. If the average degree ends up being four, a random node – at the center – will have four neighbors. Each of them, will contribute on average three more neighbors when considering paths of length two.

¹² Ithiel de Sola Pool and Manfred Kochen. Contacts and influence. *Social networks*, 1(1):5–51, 1978

¹³ Time for another mathematical derivation!

$$\begin{aligned} \bar{k}^l &= |V| \\ l \ln \bar{k} &= \ln |V| \\ l &= \ln |V| / \ln \bar{k}. \end{aligned}$$

Since in a $G_{n,p}$ network $\bar{k} = |V|p$, you get the final derivation in the text.

p is really low. Which is another thing that $G_{n,p}$ graphs have in common with real world networks. It seems that these random graphs are not too shabby when it comes to reproducing real world properties!

16.5 Clustering

What's the clustering of a $G_{n,p}$ network? Remember Section 12.2: the local clustering CC_v of a node v is number of triangles over the number of triads centered in v . The number of triads, as we saw then, is the number of possible edges among neighbors: $\bar{k}(\bar{k}-1)/2$. We can use \bar{k} instead of k_v , because any v in a $G_{n,p}$ graph is expected to have an average degree. That's the denominator of CC_v .

The numerator of CC_v is the number of triangles centered of v . This is the probability an edge exists (p), times the number of possible edges among neighbors. Again, on the basis of Section 12.2, we know that the number of possible edges among neighbors is $\bar{k}(\bar{k}-1)/2$.

$$CC_v = \frac{\# \text{ Triangles}_v}{\# \text{ of Triplets}_v}$$

$$\# \text{ Triplets} = \frac{n * (n - 1)}{2}$$

$$\# \text{ neighbors} = \bar{k}$$

$$\left(p * \frac{\bar{k} * (\bar{k} - 1)}{2} \right) / \left(\frac{\bar{k} * (\bar{k} - 1)}{2} \right)$$

Much lower than the one of real world networks! **p** Every node has the same (So it's also the average cc)

Figure 16.7: The derivation of the clustering coefficient of a random $G_{n,p}$ network.

If we say, for simplicity, that $\bar{k}(\bar{k}-1)/2 = x$, then our CC_v formula looks like: $CC_v = px/x = p$. This means that the clustering coefficient doesn't depend on any node characteristic, and it's expected to be the same – equal to p – for all nodes. Figure 16.7 provides a graphical version of this derivation.

Compared to real world networks, p is usually a very low value for the clustering coefficient. In real world networks, it's more likely to close a triangle than to establish a link with a node without common neighbors. Thus the clustering coefficient tends to be higher than simply the probability of connection. This is a second pain point of Erdős-Rényi graphs when it comes to explain real world properties, after the lack of a realistic degree distribution, as we saw in Section 16.2. Such a low clustering usually implies also the absence of

a rockstar feature of many real world networks: communities.

To recap, $G_{n,m}$ and $G_{n,p}$ models correctly estimate the emergence of giant components and the relatively small diameters of real world systems. However they fail three of the most crucial tests: degree distribution, clustering, and communities. This is cause enough for researchers to push forward in the quest for better graph models.

You can find a deeper treatment of random graph models in Bollobás's seminal work¹⁴.

¹⁴ Béla Bollobás. Random graphs. In *Modern graph theory*, pages 215–252. Springer, 1998

16.6 Summary

1. Random graph models are useful for testing your algorithms, explain how specific properties might arise in real world networks, and test whether an observed network is really as special as you think it could be, or if its properties are due to random chance.
2. The oldest and most venerable random graph model is the $G_{n,p}$ (or $G_{n,m}$) model, where we fix the number of nodes n and the probability p of connecting a random node pair, and we extract edges uniformly at random.
3. These random graphs have a binomial degree distribution, which is very different from the broad degree distributions of real world networks.
4. There is a phase transition when it comes to the largest connected component: if your random graph has an average degree higher than one, you'll have a connected component including most of the nodes; if the average degree is lower, you won't have such component.
5. Random graphs have a short average path length just like real world networks typically have. However, they have a much lower clustering coefficient than what you find in the wild.

16.7 Exercises

1. Consider the network in <http://www.networkatlas.eu/exercises/16/1/data.txt>. Generate an Erdős-Rényi graph with the same number of nodes and edges. Plot both networks' degree CCDFs, in log-log scale. Discuss the salient differences between these distributions.
2. Generate a series of Erdős-Rényi graphs with 1,000 nodes and an increasing p value, from .00025 to .0025, with increments of .000025. Make a plot with the p value on the x axis and the size of

the largest connected component on the y axis. Can you find the phase transition?

3. Generate a series of Erdős-Rényi graphs with $p = .02$ and increasing number of nodes, from 200 to 1,400 with increments of 200. Make a plot with the $|V|$ value on the x axis and the average path length on the y axis. Since the graph might not be connected, only consider the largest connected component. How does the APL scale with the number of nodes?
4. Generate an Erdős-Rényi graph with the same number of nodes and edges as the network used for question 1. Calculate and compare the networks' clustering coefficients. Compare this with the connection probability p of the random graph (which you should derive from the number of edges and number of nodes using the formula I show in this chapter).

17

Understanding Network Properties

We now move on to the class of network models developed primarily to explain network properties. The two most famous examples in this class are the small world model proposed by Watts and Strogatz and the preferential attachment model, independently discovered in many variants multiple times across many decades but usually attributed to Albert and Barabási. The first aims at explaining the high level of clustering and small diameter of real world networks. The second focuses on power law degree distributions.

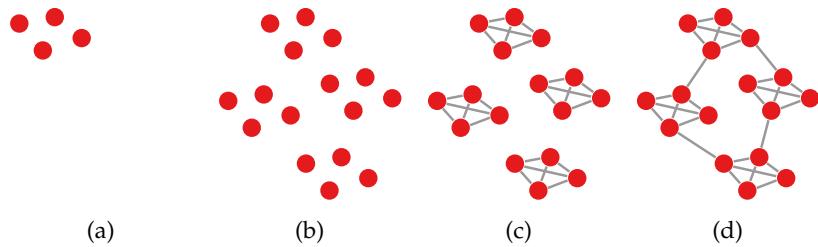
The interest in these models is twofold. First, it is a historic interest: these models were the first developed in the new wave of network science in the late 90s. Their impact in the development of the field was huge – the original papers both accumulated more than 46 thousand citations. Second, they give an idea of what are some of the original guiding principles of a certain flavor of network science: the hunt for universal patterns that apply to any network representation of a complex system. In practice, nowadays you'd seldom use these vanilla models, as they have been superseded by more sophisticated – albeit often less mathematically tractable – ones.

17.1 Clustering

Before the Stone Age, a caveman society was very simple. You had tribes living in their own caves. The tribes were very small, they were families. Everybody knew everyone else in their cave, but between caves there was almost no communication. Maybe there could have been one weak link if the two caves were close enough.

This metaphor was the starting point for Watts in developing his “cavemen” model¹. The cavemen model is part of the family of simple networks (see Section 6.4). It takes two parameters: the cave size (Figure 17.1(a)) and the number of caves (Figure 17.1(b)). The cave size is the number of people living in each cave. A cave is a clique: as said, everyone in the cave knows every caveman (Figure

¹ Duncan J Watts. Networks, dynamics, and the small-world phenomenon. *American Journal of sociology*, 105(2): 493–527, 1999



17.1(c)). Each cave “elects” a random member which will connect to a random member of the nearest cave on the left, and another member to connect to the nearest cave to the right (Figure 17.1(d)). That’s it: the cavemen model.

By construction, the cavemen model has only one component, because the caves are always connected with their nearest neighbors. However, the cavemen model is worse than $G_{n,p}$ in approximating realistic diameters. To go from one cave to the farthest one in the network it takes a really long path. The degree distribution is also weird: in the example of Figure 17.1, all nodes inside a cave have the same degree (equal to three) and the nodes in between caves all have degree equal to four. Needless to say, this system with only two distinct degree values isn’t found anywhere in natural networks.

So why do we want this type of graph? Well, differently from $G_{n,p}$, cavemen gives us clustering and communities. Having such well separated groups – the caves – makes it an ideal dataset to test whether your community discovery algorithm is working or if it is returning random results. You can’t get anything clearer than a clique with just two edges pointing outwards.

17.2 Path Lengths

The small-world model is a more famous model developed by the same author². It also models high clustering, but its primary target was to explain small diameters, which were discovered in real world social networks by Milgram, as I showed in Section 13.3. In a small world we start from you. You are standing in a certain point in space. Then there are other people, also in their spots. You can only communicate with people that are nearby you, because they are the ones who can listen to you. Their sets of listeners overlap with yours, because you’re all nearby each other. But, since they are not exactly occupying your position, there are some folks whom you can reach and they cannot, and vice versa. They can talk to an extra neighbor. And so can their neighbors, to infinity.

This creates a regular network, a lattice, where each node is the same as each other node. This is a simple network. Figure 17.2

Figure 17.1: (a) First step of cavemen: decide the size of the cave. (b) Second step: decide the number of caves. (c) Third: make each cave in a clique. (d) Finally connect the nearest caves via random cave members.

² Duncan J Watts and Steven H Strogatz. Collective dynamics of ‘small-world’ networks. *nature*, 393(6684):440, 1998

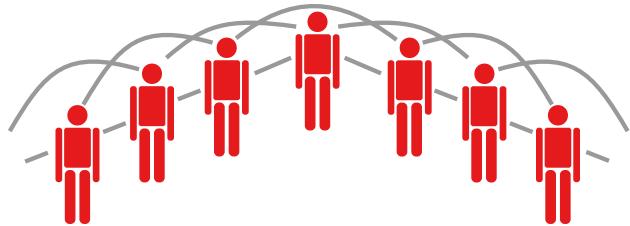


Figure 17.2: The first step of a small world model: each individual can talk to their four most immediate neighbors.

provides a simple visualization. To use a more formal language, in the first step of a small world model you put nodes into a single dimensional space and connect them with their k nearest neighbors – with k being the first parameter of the model that you can specify.

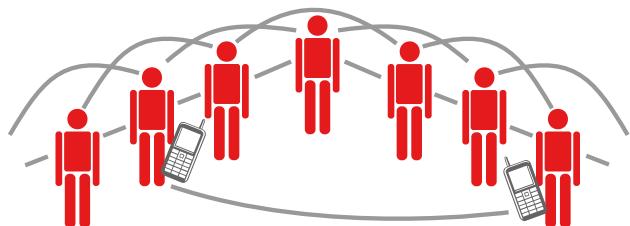
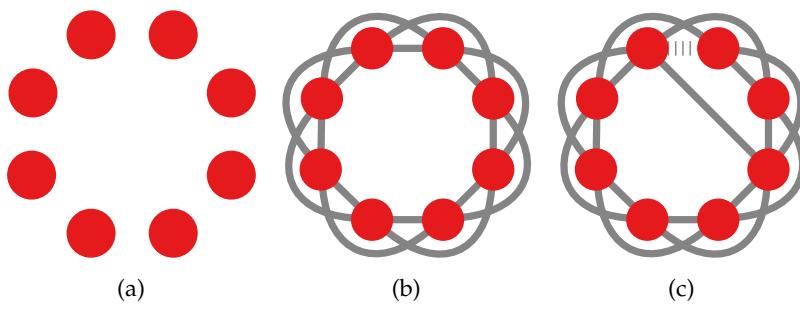


Figure 17.3: The second step of a small world model: random individuals can talk to someone in the network, no matter their physical locations.

However, sometimes, two folks can talk at distance, maybe because they have each other phone number. And so a shortcut is made, as I show in Figure 17.3. Formally, you establish a rewiring probability p – the second parameter of the model. For each edge you toss a coin: if it lands on heads you delete the edge and you rewire it by picking a random destination. In variants of the model³ you do not delete the original edge: for each existing edge you have a certain probability to pick an additional random destination for one of the two connected nodes.



³ Mark EJ Newman and Duncan J Watts. Renormalization group analysis of the small-world network model. *Physics Letters A*, 263(4-6):341–346, 1999

Figure 17.4: The steps of the small world mode: (a) Place nodes in a one-dimensional ring space. (b) Connect each node with its k nearest neighbors. (c) Randomly rewire edges according to the parameter p .

Figure 17.4 shows the full process. Again, by construction we don't need fancy math to prove that this model generates a single component. Since each node connects to a few nearest neighbors we have one component by default. In the rewiring model you could divide the network in separated components by unlucky rewiring

draws, but this is pretty unlikely, especially for high k values. Anyhow, given how unlikely this is, we can say that the small world model properly recovers the giant connected component property of real world networks.

Differently from cavemen, this time we have short paths. They are regulated by the rewiring probability p . Rewiring creates bridges that span across the network. Even a tiny bridge probability can connect parts of the network that are very far away. Thousands of shortest paths will use it and will be significantly shorter.

Still, the degree distribution of a small world model is very weird. If p is low, it looks like a cavemen graph, because almost all nodes will have the same number of neighbors. If p is high it means that we are practically rewiring every edge in the graph. At that point, randomness overcomes every other feature of the model, and the result would be almost indistinguishable from a $G_{n,p}$ model. We have high clustering because each connection that we don't rewire will create triangles with some of the neighbors of the connected nodes⁴.

⁴ Unless you set $k = 2$, then the clustering would be zero. But why would you set $k = 2$ in a small world model? People are weird.

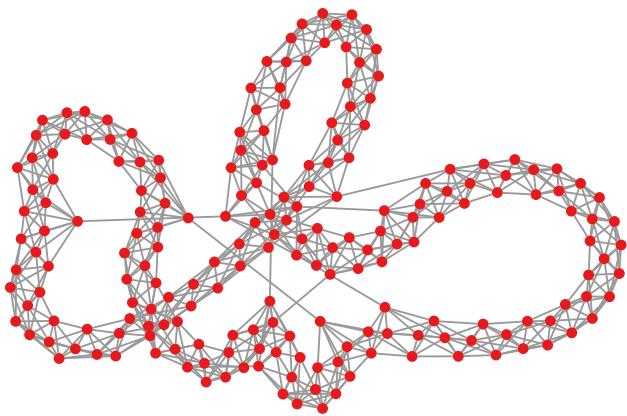


Figure 17.5: A small world graph with 200 nodes, average degree \bar{k} equal to 8, and rewiring probability $p = 0.01$.

However, high clustering does not necessarily mean that you are going to have communities. In fact, a small world model typically doesn't have them. The triangles are distributed everywhere uniformly in the network. There are no discontinuities in the density, no differences between denser and sparser areas. This is especially evident for high k and low p : as I show in Figure 17.5, small world networks with such parameter combinations just look like odd snakes without clear groups of densely connected nodes. This is a precondition to have communities, and so you cannot find them in a small world model.

17.3 Degree Distribution

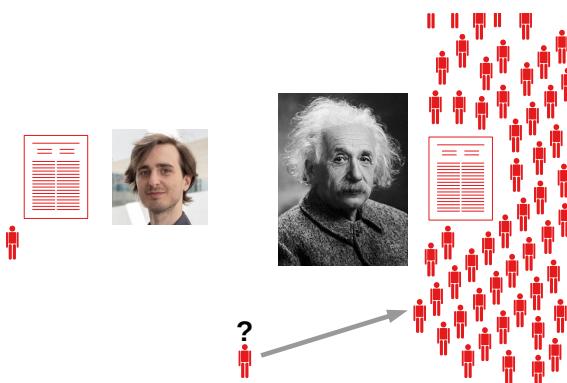
If we want to reproduce the degree distribution of a scale free network we need a process that can generate a power law degree distribution. One of the most popular approaches is cumulative advantage⁵. This is a fundamentally dynamic model. You have an initial condition and then you keep adding one element at a time. Each element you add does not contribute to the preexisting ones uniformly at random, but *prefers* to contribute to specific older elements, according to a rule you determine.

The preferential attachment model starts from the assumption that the rich get richer. For instance, suppose you have one coin and you invest in the stock market. If you're lucky, after a while, you will have another coin. Consider instead somebody who has a lot of coins. Not only she can match your returns, she can probably do better, because she can have a diversified portfolio which is resilient to market shocks and black swans – highly improbable but also massively impactful events, like the one at the basis of the mortgage crisis⁶. Moreover, she can probably pay better advisers, and capital – according to Piketty⁷ – just has better returns at scale. In the time it takes for you to make a coin, she makes hundreds. Being already rich makes her proportionally richer than you.

⁵ Jonathan R Cole and Stephen Cole. Social stratification in science. 1974

⁶ Nassim Nicholas Taleb. *The black swan: The impact of the highly improbable*, volume 2. Random house, 2007

⁷ Thomas Piketty. Capital in the 21st century. 2014



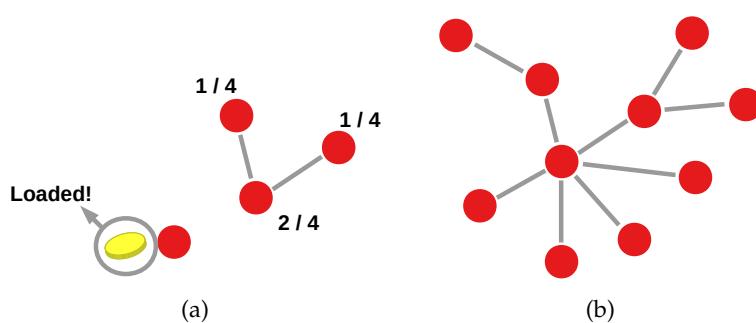
The textbook case of cumulative advantage is in scientific publishing⁸. In terms of networks, consider a citation network. I can write a paper, and maybe at some point somebody will read it and cite it. On the other hand, we might have an actual researcher with a paper that has been cited hundreds of thousands of times. If we have a newcomer to the citation network, what is she more likely to see, and therefore to cite? The paper everybody knows. And so she will add to the pool, further increasing the odds that the paper will be seen by another newcomer. Figure 17.6 shows a vignette depicting this process.

⁸ Robert K Merton. The matthew effect in science: The reward and communication systems of science are considered. *Science*, 159(3810):56–63, 1968

As I said, this reasoning is at the basis of a dynamical **preferential attachment** model^{9,10,11}. In preferential attachment you start from an initial (set of) node(s) and you keep adding more. Each time you add a new node, it will connect to m of the old ones, where m is a parameter of the model. It will connect at random, but *preferentially* to nodes with higher degree: the higher a node's degree, the more likely it is it will gather more connections. Let's follow the process step by step.

The initial condition, meaning the topology of the network from which you start, is not specified by the model. You can have practically what you want: a clique, a chain, a star. The idea is that, after enough steps, what you started from doesn't matter. Of course, this seed has to have some characteristics that make it compatible with your model. For instance, since you are going to add a new node with m connections, it means that the initial condition has to have at least m nodes. An accepted convention is to have them be connected in a clique, so that they all have the same degree. One downside of this flexible initial condition is that it makes the model a bit less tractable mathematically, as you don't really have a formula describing an arbitrary graph. There are some variants of the model that fix this issue, for instance the linearized chord diagram¹².

When you add your first new node, since you only have m initial nodes in the seed and you have to place m connections, there isn't much choice in deciding to whom you connect it. The new node will connect to all nodes in the seed. When you add more nodes, you flip a coin m times to decide who gets the edges from the new node. By the time you're adding the third node, you have more than m nodes to choose from, and they do not have the same degree: some have degree m , others have degree $m + 1$. You still flip a coin to decide where the edges go, but now it's a loaded one – see Figure 17.7(a), where I fix $m = 1$. The new edges are more likely to go to the nodes with more connections. If you keep repeating the process, you end up with something looking like Figure 17.7(b).



⁹ Herbert A Simon. Models of man; social and rational. 1957

¹⁰ Derek de Solla Price. A general theory of bibliometric and other cumulative advantage processes. *Journal of the American society for Information science*, 27(5):292–306, 1976

¹¹ Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *science*, 286(5439):509–512, 1999

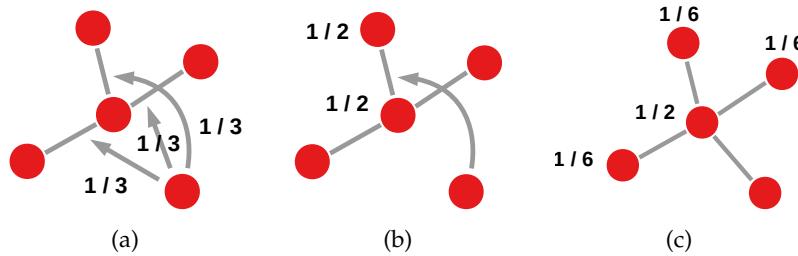
¹² Béla Bollobás, Oliver Riordan, Joel Spencer, and Gábor Tusnády. The degree sequence of a scale-free random graph process. *Random Structures & Algorithms*, 18(3):279–290, 2001

Figure 17.7: (a) Adding a fourth node in a preferential attachment model creates uneven probabilities of attachment (floating next to the nodes that are already part of the network). (b) A possible end result of the preferential attachment after adding ten nodes.

As I said earlier, the law determining the connection probabilities floating next to the nodes in Figure 17.7(a) is the number of connections the node already has over twice the number of edges. In practice, newcomers prefer to attach to high degree nodes. This advantage accumulates over time: if Einstein is the highest degree node, he is the most likely to get a new edge, which makes it even more likely for him to get the next edge, and so on. You see that newcomers have an ever decreasing chance to get the new connections.

This is not the only way to create a cumulative advantage. In fact, the model has some defects. Preferential attachment requires the newcomer nodes to have global information about all the existing nodes' degree. This might be unrealistic in some cases – you may not know the number of citations of every paper when you are making a citation. It is also not a necessary feature to generate a cumulative advantage.

An alternative to preferential attachment is **link selection**¹³. In link selection, the newcomer node selects a link at random from the ones that exist in the network (Figure 17.8(a)). Then, it connects with one of the two nodes connected by that edge – choosing uniformly at random between the two (Figure 17.8(b)). Cumulative advantage arises because nodes with more links are more likely to be selected, thus getting more links on average (Figure 17.8(c)). No matter which link you select, the central hub is connected to it.

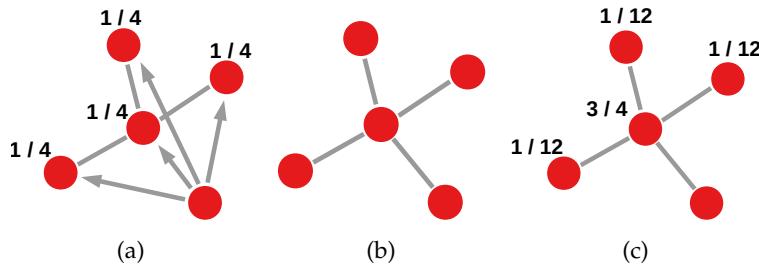


A third alternative from preferential attachment and link selection is the **copying model**. Just like in link selection, the newcomer node has no information about the network, it just picks something uniformly at random. Differently from the link selection model, here it picks another existing node, rather than a link (Figure 17.9(a)). It then copies one of its connections (Figure 17.9(b)). You can see again how it's more likely to connect to the hub: the hub has more neighbors, thus it is more likely to select one of its neighbors. Moreover, the neighbor of a hub is likely to be low degree, increasing the chances of selecting the hub in the copying step (Figure 17.9(c)). The copying model is based on an analogy on how webmasters create new hyperlinks to pre-existing content on the web¹⁴.

¹³ Sergey N Dorogovtsev and Jose FF Mendes. Evolution of networks. *Advances in physics*, 51(4):1079–1187, 2002

Figure 17.8: Adding a new node with the link selection model. (a) Pick an existing link at random. (b) Pick one of the two nodes connected by that link. (c) Connect to it. The probabilities of connecting to each node in this process are the ones floating next to it.

¹⁴ Jon M Kleinberg, Ravi Kumar, Prabhakar Raghavan, Sridhar Rajagopalan, and Andrew S Tomkins. The web as a graph: measurements, models, and methods. In *International Computing and Combinatorics Conference*, pages 1–17. Springer, 1999



It is easy to see why a network generated with either of these three models has a single connected component. Since you always connect a new node with one that was already there, there is no step in which you have two distinct connected components. Thus, any cumulative advantage network following any of these models will have all its nodes in the same giant connected component, as it should.

These networks also have short diameters and average path lengths. Mechanically it is easy to see why. Hubs with thousands of connections can be used as shortcuts to traverse the network.

Mathematically, the diameter of a preferential attachment network grows as $\frac{\log |V|}{\log \log |V|}$, thus very slowly, slower than a random graph.

Figure 17.10 shows some simulations, comparing the average path length of a $G_{n,m}$ and a preferential attachment network with the same number of nodes and the same number of edges, as their size grows.

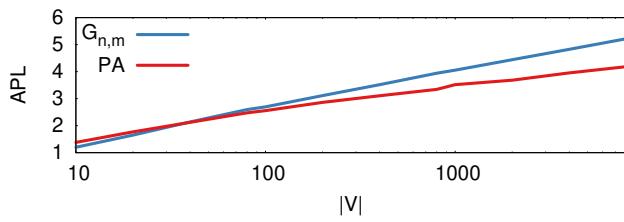


Figure 17.9: Adding a new node with the copying model.
 (a) Pick an existing node at random. (b) Copy one of its connections. (c) The probabilities of connecting to each node in this process are the ones floating next to it.

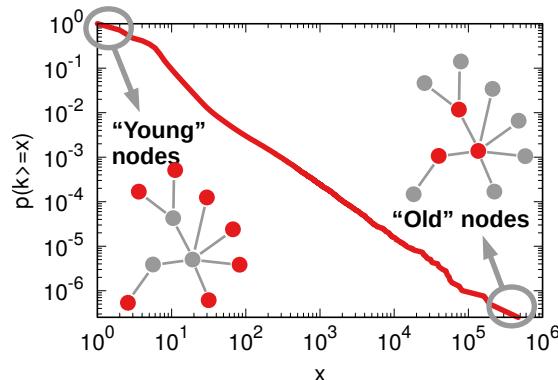
These models also reproduce power law degree distributions – that's what they were developed for. In fact, you can calculate the exact degree distribution exponent for the standard preferential attachment model, which is $\alpha = 3$. This is independent from the m parameter, meaning that you cannot tune it to obtain different exponents (obviously, you might get different exponents because of the randomness of the process, but as $|V| \rightarrow \infty$, then $\alpha \rightarrow 3$). If you want to reproduce a real world network with $\alpha = 2$, you cannot use the basic preferential attachment model.

However, there is a peculiar aspect about their degree distributions that is worth considering. As you saw from all examples, the cumulative advantage applies especially to “old” nodes. The earlier the node entered in your structure, the more likely it is to become a hub.

Figure 17.10: The average shortest path length (y axis) for increasing number of nodes (x axis) for $G_{n,m}$ (blue) and preferential attachment (red) models, with the same average degree.

This is especially easy to see in preferential attachment: the first node getting the second edge in Figure 17.7 already has an advantage that newcomers are unlikely to match.

If you look at a degree distribution of a preferential attachment model, you'll find the old nodes in the tail – the hubs – and the head is going to be mostly composed by “young” nodes. I show an example in Figure 17.11. Put it another way, there is a positive correlation between a node's age and its degree. This is particularly interesting because that is not something we observe in real world systems¹⁵. For instance, you could argue that, on the web, the number one website at the moment is google.com. However, google.com isn't the oldest website in the world. The web was already four years old when google.com was created. Moreover, readers a hundred years from now¹⁶ might not even know what google.com is, because something replaced it.



There are ways to tweak the classical preferential attachment model to fix some of its issues. For instance, one way is to balance popularity and similarity¹⁷. To determine where the connections of a new node attach to, the classical preferential attachment uses exclusively the popularity of the already present nodes: the more connections a node has, the more it'll gather. However, nodes will want to connect to other nodes that are similar to them – we'll see this real world tendency when we'll discuss about homophily in Chapter 30. Thus, if you have metadata about how similar two nodes are, you can create a model where this similarity score is as important as a node's popularity to determine which nodes will connect to it when they first arrive in the network.

From the examples made so far, you probably figured out that there's another thing missing: clustering. In particular, so far I made simple examples where a newcomer node will add only one edge to the network (the parameter m is equal to one). If we add one node and one edge at a time it is impossible to create triangles.

¹⁵ Lada A Adamic and Bernardo A Huberman. Power-law distribution of the world wide web. *science*, 287(5461):2115–2115, 2000

¹⁶ I'm very optimistic about the success of this book.

Figure 17.11: The age effect in the degree distribution of the preferential attachment model.

¹⁷ Fragkiskos Papadopoulos, Maksim Kitsak, M Ángeles Serrano, Marián Bojuná, and Dmitri Krioukov. Popularity versus similarity in growing networks. *Nature*, 489(7417):537–540, 2012

You could set the parameter $m > 1$ to add two or more edges per new node, but that helps only to a certain point: it's not so likely to strike two already connected nodes thus creating a triangle. The preferential attachment model has a higher clustering than a random $G_{n,p}$ one, but not by much. It is still a far cry from the clustering levels you see in real world networks. Moreover, this is only true for $m > 1$. In that case, you add more than one edge per newcomer node, which means you end up losing the head of your distribution. The network will not contain a single node with degree equal to one.

Thus the clustering in these cumulative advantage models is much lower than real world networks, and there are no communities – because everything connects to hubs which make up a single core. There are some extensions of the model which try to include clustering¹⁸. At every step of this model you have a choice. You either add a node with its links, or you just add links between existing nodes without adding a new one. The probability of taking that step regulates the clustering coefficient of the network.

However, triangles close randomly, thus we have no communities just like in the small world model. If we want to look at models which generate more realistic network data, we have to look at the ones I discuss in the next chapter.

17.4 Summary

1. To explain the high clustering and small diameter in real world networks we could use the small world model by Watts and Strogatz. In it, we place nodes on a regular distance in a low dimensional space and connect them to k of their neighbors, ensuring high clustering. We then create few shortcuts connecting pairs of nodes at random with probability p , ensuring a small diameter.
2. The small world model has no communities, which you could generate with a caveman graph: a ring of cliques. However, the caveman graph has a long diameter.
3. To explain power law degree distributions you could use a preferential attachment model. In it, you grow the network one node at a time. Each node brings m random connections. The probability of connecting to a node u already present in the graph is proportional to u 's degree. This model has low diameter, but also low clustering.
4. Alternative models recreating power law degree distributions are the link selection and the copying models.

¹⁸ Petter Holme and Beom Jun Kim. Growing scale-free networks with tunable clustering. *Physical review E*, 65(2):026107, 2002

5. Another unrealistic effect of the preferential attachment model is the correlation between a node's age (how long ago we added it to the network) and its degree. Such correlation might not exist in real world networks.

17.5 Exercises

1. Generate a connected caveman graph with 10 cliques, each with 10 nodes. Generate a small world graph with 100 nodes, each connected to 8 of their neighbors. Add shortcuts for each edge with probability of .05. The two graphs have approximately the same number of edges. Compare their clustering coefficients and their average path lengths.
2. Generate a preferential attachment network with 2,000 nodes and average degree of 2. Estimate its degree distribution exponent (you can use either the `powerlaw` package, or do a simple log-log regression of the CCDF).
3. Implement the link selection model to grow the graph in <http://www.networkatlas.eu/exercises/17/3/data.txt> to 2,000 nodes (for each incoming node, copy 2 edges already present in the network). Compare the number of edges and the degree distribution exponent with a preferential attachment network with 2,000 nodes and average degree of 2.
4. Implement the copying model to grow the graph in <http://www.networkatlas.eu/exercises/17/4/data.txt> to 2,000 nodes (for each incoming node, copy one edge from 2 nodes already present in the network). Compare the number of edges and the degree distribution exponent with networks generated with the strategies from the previous two questions.

18

Generating Realistic Data

Both the small world and the preferential attachment models are useful because they give us ideas on how some real world network properties arise. The small world model tells us that small diameters happen because a clustered network might have some random shortcuts. The preferential attachment model tells us that broad degree distributions arise because of cumulative advantage: having many links is the best way to attract more links.

Yet, neither of them is able to reproduce all the features of a real world network. If we want to do so, we have to sacrifice the explanatory power of a model. We have to fine tune the model so that we force it to have the properties we want, regardless of what realistic process made them emerge in the first place. This is the topic of this chapter.

18.1 Configuration Model

The easiest way to ensure that your network will have a broad degree distribution is to force it to have it. No fancy mechanics, no emerging properties. You first establish a degree sequence and then you force each node to pick a value from the sequence as its degree. This simple idea is at the basis of the configuration model¹. In fact, the configuration model is more general than this. You can use it to match the degree sequence of *any* real world graph, regardless of the simplicity or complexity of its actual degree distribution.

The configuration model starts from the assumption that, if we want to preserve the degree distribution, we can take it as an input of our network generating process. We know exactly how many nodes have how many edges. So we forget about the actual connections, and we have a set of nodes with “stubs” that we have to fill in. Figure 18.1 shows an example.

There’s a relatively simple algorithm to generate a configuration model network, the Molloy-Reed approach^{2,3}. First, as we saw, you

¹ Mark EJ Newman. The structure and function of complex networks. *SIAM review*, 45(2):167–256, 2003b

² Michael Molloy and Bruce Reed. A critical point for random graphs with a given degree sequence. *Random structures & algorithms*, 6(2-3):161–180, 1995

³ Mark EJ Newman, Steven H Strogatz, and Duncan J Watts. Random graphs with arbitrary degree distributions and their applications. *Physical review E*, 64(2):026118, 2001

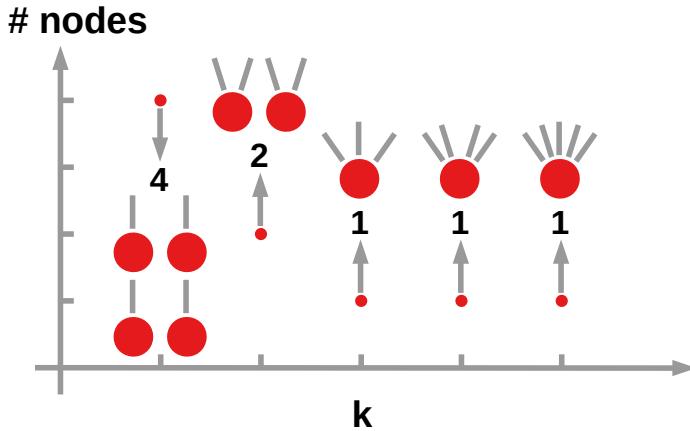


Figure 18.1: In a configuration model, you start from the degree histogram to determine how many nodes have how many open “edge stubs”.

create a degree sequence, in which each value represents a node’s degree. This sequence has a few constraints: the most important is that it has to sum to an even number. If it were to sum to an odd number, you’d have a node which cannot assign its last stub to any other neighbor – i.e. the sequence is not “graphic”.

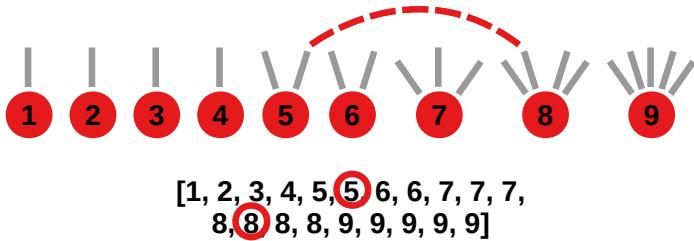
Second, each node gets a unique identifier. As third step, you generate a list of these identifiers. You repeat each identifier in this list as many times as its assigned degree. For instance, if you know that node 1 has degree four, the list will contain four 1s. If node 2 has degree twenty, you add twenty 2s.

Finally, you create the actual connections. You pick two elements at random from this list, which are two node identifiers. If the identifiers are different and the two nodes are not already connected to each other, you connect them. The two conditions are necessary to avoid the creation of self loops and parallel edges – if you’re ok with either, you can skip these checks. Note that you might end up with a few unassigned edges, but usually these are an insignificant number which will not affect the degree distribution too much.

Note that, each time we pick two ids from the list, we remove them from it. This ensures that each node will be picked only as many times as its degree – or fewer times if we cannot find any legal connections at the end of the process. Figure 18.2 shows a depiction of a connection step in the configuration model.

The Molloy-Reed approach is not the only way to generate a random graph with a given degree distribution. For starters, there are closely related alternatives like the Chung-Lu model⁴. An alternative is the double swap edge algorithm which I describe in Section 19.1. As you’ll see, in that case one doesn’t need to worry about self-loops or parallel edges. The two algorithms are in different chapters because of their different aims. If you simply need a realistic graph with a given degree distribution for testing an algorithm or

⁴ Fan Chung and Linyuan Lu. Connected components in random graphs with given expected degree sequences. *Annals of combinatorics*, 6(2):125–145, 2002b



do asymptotic mathematics, the Molloy-Reed configuration model is good enough. But, if you want to compare a random graph to data, the differences are crucial⁵ and that's why the edge swap algorithm is in the chapter dedicated to evaluating statistical significance.

You can add a few features to the configuration models that were not trivial to add to either small world or preferential attachment. For instance, you can make a directed version of it. The only thing you need is to generate two degree distributions: one for the in-degree and one for the out-degree. This makes the connection step a bit more complex, as you have to pick the source from one list and the destination from another, but it is not a big deal.

You can see that this model will be spot on in replicating the vast majority degree distributions you pass to it. The way to estimate the difference between two distributions is by performing a Kolmogorov-Smirnov test^{6,7}. The test identifies the point of maximum separation between two distributions, along with a p -value telling you how likely it is that the two distributions are indistinguishable from each other.

The configuration model tends to generate giant connected components just like a random $G_{n,p}$ graph would, although this heavily depends on the α parameter of your power sequence⁸ (see Section 9.4 for a refresher about the meaning of α). Deriving the expected average shortest path length is a bit trickier, but it can be done⁹ and it is realistically short in most cases.

The clustering coefficient of a configuration model also depends on α . For the majority of realistic values of α – between 2 and 3 –, the clustering coefficient of a configuration model tends to zero, which is very unrealistic. There are a few valid values of α generating a properly high clustering, but these are rare enough that researchers needed to modify the configuration model to explicitly include the generation of triangles^{10,11}.

This is usually achieved by generating a joint degree sequence. Rather than simply specifying the degree of each node, we now have to fix two values. The first is the number of triangles to which the node belongs. The second is the number of remaining edges the node has that are not part of a triangle. One can see that we're still

Figure 18.2: A depiction of the algorithm to generate a network following the configuration model: the edge stubs on the nodes with their identifiers (top) and the list with node ids from which we pick nodes (bottom). The red circled ids are the ones picked, so we connect node 5 to node 8.

⁵ Bailey K Fosdick, Daniel B Larremore, Joel Nishimura, and Johan Ugander. Configuring random graph models with fixed degree sequences. *SIAM Review*, 60(2):315–355, 2018

⁶ Andrey Kolmogorov. Sulla determinazione empirica di una legge di distribuzione. *Inst. Ital. Attuari, Giorn.*, 4: 83–91, 1933

⁷ Nickolay Smirnov. Table for estimating the goodness of fit of empirical distributions. *The annals of mathematical statistics*, 19(2):279–281, 1948

⁸ William Aiello, Fan Chung, and Linyuan Lu. A random graph model for massive graphs. In *Proceedings of the thirty-second annual ACM symposium on Theory of computing*, pages 171–180. Acm, 2000

⁹ Fan Chung and Linyuan Lu. The average distances in random graphs with given expected degrees. *Proceedings of the National Academy of Sciences*, 99(25):15879–15882, 2002a

¹⁰ Mark EJ Newman. Random graphs with clustering. *Physical review letters*, 103(5):058701, 2009

¹¹ Joel C Miller. Percolation and epidemics in random clustered networks. *Physical Review E*, 80(2):020901, 2009

specifying the degree, because the number of triangles is simply half the number of edges we're adding to that node: each node in a triangle connects to other two nodes. If you know the number of triangles and the total degree of each node you know the clustering coefficient of the network (Section 12.2), thus you can generate a sequence that will have the desired clustering coefficient.

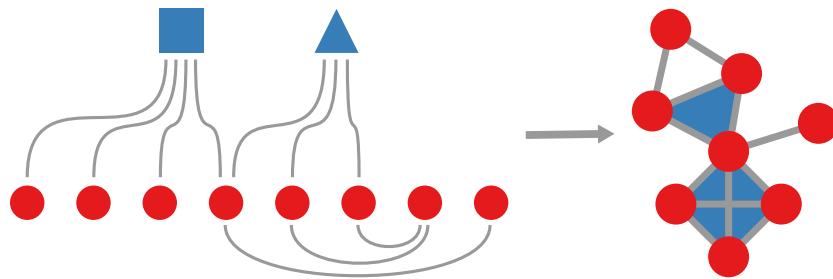


Figure 18.3: A simplicial configuration model with 8 nodes (in red) and two simplices (in blue). The nodes have the following open stubs (from left to right): $\{1, 1, 1, 3, 2, 2, 2, 1\}$.

An alternative – and more flexible – way to build higher order structures in your configuration model is to allow it to generate simplicial complexes. I introduced simplicial complexes in Section 7.3: these are graphs including relations between multiple nodes, rather than just normal edges, which are binary relationships. In a simplicial configuration model, besides nodes with open stubs, you also have simplices, each with the number of stubs determined by their dimension (from 2 up)^{12,13}. Figure 18.3 shows an example.

You can connect each node stub with either another node stub, or with a simplex, until there are no more open stubs. Note that, in this model, the number of open stubs is not the degree any more. Every time you connect a node to a simplex, the node's degree increases by the dimensionality of the stub minus one (thus by 2 if you connect it to a triangular complex, by 3 if you connect it to a square, and so on). Thus controlling the exact degree of a node is trickier.

Another complication is that you have more topological constraints. Forget about simply generating a graphical degree sequence: you now have a set of forbidden moves. A node can be part of multiple simplices, but you cannot make two distinct simplices using the very same nodes (Figure 18.4(a)). Nor you can use two stubs from the same node in the same simplex (Figure 18.4(b)). The first case is forbidden because you'd be creating two indistinguishable simplices – and therefore only one instead of two. In the second case, you wouldn't be constructing a simplex at all!

The result is a network that has exactly the desired number of simplices. It might have, however, more triangles than you expect, because you can connect three nodes independently. There's an example of this mishap in Figure 18.3: one of the two triangles is

¹² Owen T Courtney and Ginestra Bianconi. Generalized network structures: The configuration model and the canonical ensemble of simplicial complexes. *Physical Review E*, 93(6):062311, 2016

¹³ Jean-Gabriel Young, Giovanni Petri, Francesco Vaccarino, and Alice Patania. Construction of and efficient sampling from the simplicial configuration model. *Physical Review E*, 96(3):032312, 2017

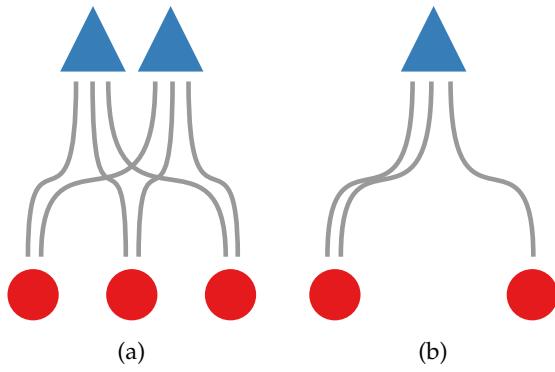


Figure 18.4: Two forbidden moves in the simplicial configuration model.

not filled, because it's made by three independent edges, rather than being a three-way relationship, like the blue-filled simplex.

You don't have to necessarily end up with a simplicial network or a hypergraph: once you have placed the shapes you can "forget" that they are higher order structures, and consider your network as a simple graph.

18.2 Communities

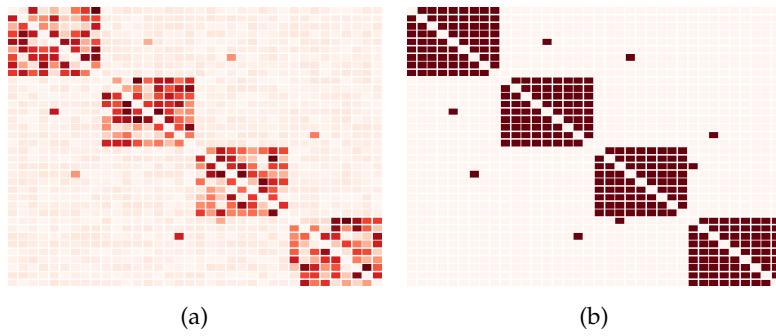
The configuration model can be tuned to include a high clustering, but it will still close triangles randomly in the network. This means that the number of triangles is correct, but their distribution in the network is random. As I mentioned multiple times, this is not the case for real world networks. The triangles tend to correlate and form denser areas we call communities – specifically, assortative communities, the concept of community is a bit more complex than that and requires to read Part X, for now let's just roll with this simplification. Thus, the configuration model is still inadequate to fully reproduce a quasi-real network. Doing so is the task of a few models: stochastic block models, GN and LFR benchmark, and Kronecker graphs.

Stochastic Block Models

The easiest way to create communities in your graph model is to make a simple observation. Since communities are dense areas – with nodes connecting to each other – separated by sparse areas, it just means that nodes have two different probabilities to connect to each other. One probability, say p_{in} , determines the probability of a node u to connect to another node inside the same community. Another probability, p_{out} , determines the likelihood of connecting to a node from a different community. Obviously, if we want dense communities, $p_{out} < p_{in}$.

p_{in} and p_{out} are the first two parameters of a Stochastic Block Model¹⁴ (SBM). To fully specify the model you need a few additional ingredients. First, you have to specify $|V|$, the number of nodes in the graph. Then you have to plant a community partition. In other words, for each node – from one to $|V|$ – you have to specify to which community it belongs. Otherwise we don't know how to use the p_{in} and p_{out} parameters.

It's easy to see that, if $p_{in} = p_{out}$, then the SBM is fully equivalent to a $G_{n,p}$ model: each node has the same probability to connect to any other node in the network. However, if we respect the constraint that $p_{out} < p_{in}$, the resulting adjacency matrix will be block diagonal. Most of the non zero entries will be close to the diagonal, whose blocks are exactly the communities we planted in the first place!



¹⁴ Paul W Holland, Kathryn Blackmond Laskey, and Samuel Leinhardt. Stochastic blockmodels: First steps. *Social networks*, 5(2):109–137, 1983

One could go even deeper and determine that each pair of nodes u and v can have its own connection probability. This would generate an input matrix for the SBM that looks like the one in Figure 18.5(a). Figure 18.5(b) shows a likely result of the SBM using Figure 18.5(a) as an input. It's easy to see why we call this matrix "block diagonal". The blocks are.... on the diagonal, man.

In one swoop we obtained what we were looking for: both communities and high clustering. The very dense blocks contribute a lot to the clustering calculation, more than the sparse areas around the communities can bring the clustering down. One observation we will come back to is that, in real world networks, the community sizes distribute broadly, just like the degree: there are few giant communities and many small ones. This can be embedded in SBM, since we're free to determine the input partition as we please.

If you set p_{out} to a relatively high value, you might make your communities harder to find, but you gain something else: smaller diameters. You're also free to set $p_{in} < p_{out}$ in which case you'd find a *disassortative* community structure, where nodes tend to dislike nodes in the same community. See Section 30.2 to know more about what disassortativity means.

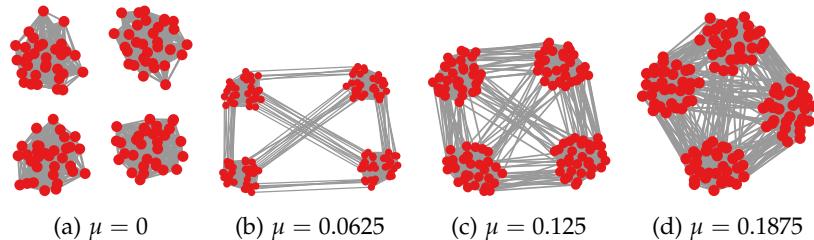
Many real world networks will have overlapping communities sharing nodes. The standard SBM cannot handle this: in the input phase we can only put a node in a single community. However, smart folks have created Mixed-Membership Stochastic Block Models¹⁵, in which nodes are allowed to span across communities. Another important variation of SBM is the degree-correlated SBM¹⁶, which allows you to fix the degree distribution just like the configuration model does.

¹⁵ Edoardo M Airoldi, David M Blei, Stephen E Fienberg, and Eric P Xing. Mixed membership stochastic blockmodels. *Journal of Machine Learning Research*, 9(Sep):1981–2014, 2008

¹⁶ Brian Karrer and Mark EJ Newman. Stochastic blockmodels and community structure in networks. *Physical review E*, 83(1):016107, 2011

GN Benchmark

The GN benchmark is a modification of the cavemen graph and one of the first network models designed to test community discovery algorithms¹⁷. The first defining characteristic of this model is setting some of the parameters of the cavemen graph as fixed. In the benchmark, we have only four caves and each cave contains 32 nodes. Differently from the caveman graph, the caves are not cliques: each node has an expected degree of 16, thus it can connect at most to half of its own cave.



¹⁷ Michelle Girvan and Mark EJ Newman. Community structure in social and biological networks. *Proceedings of the national academy of sciences*, 99(12):7821–7826, 2002

Figure 18.6: Results from the GN benchmark for increasing values of μ .

The GN benchmark then introduces a parameter, usually called “mixing” parameter, or μ . This is the share of edges that a node has to nodes that are not part of its own cave. You can use μ to introduce the amount of noise you want in the community structure. If $\mu = 0$, then all nodes will exclusively connect with fellow members of the same cave. This results in four isolated connected components with no paths among them. If $\mu = 0.5$, half of the edges of a node point outside its community, to a random other cave. You can see the effect of μ in the sequence from Figure 18.6.

The GN benchmark isn’t particularly realistic. It has a fixed number of nodes, rather low when compared with real world networks. Its degree distribution is binomial, which rarely happens in the real world. It is also rare to have equally-sized communities. The LFR benchmark fixes all these issues.

LFR Benchmark

The LFR Benchmark was developed to serve as a test case for community discovery algorithms¹⁸. The objective is to generate a large number of benchmarks to test a new algorithm such that we know the “true” allegiance of each node. Once an algorithm returns us a possible node partition, we can compare its solution with the true communities.

Since you want to have networks with lots of realistic properties, some of which are difficult to reproduce organically, the LFR benchmark takes lots of input parameters. If you want an LFR network, you have to specify:

- The α exponent of the power law degree distribution of the graph;
- The β exponent of the power law size distribution of the communities in the graph;
- The $|V|$ number of nodes in your graph;
- The \bar{k} average degree of the nodes – you can also set k_{min} and k_{max} as the minimum and maximum degree, respectively;
- s_{min} and s_{max} , as the minimum and maximum community size;
- The μ mixing parameter, regulating the fraction of edges going outside their planted communities;

Optional o_n : the fraction of nodes overlapping between multiple communities, if using the overlapping variant of LFR;

Optional o_m : the number of communities to which an overlapping node belongs.

As you can see, the LFR assumes that both the degree distribution and the size of your communities distribute like a power law. The latter is regulated by the β parameter, with one gigantic community including the majority of nodes and many trivial communities of size s_{min} . In Figure 18.7 I show a real world network and its three largest communities, showing how their sizes rapidly decline. This is a rather realistic assumption, although there are obvious exceptions. You can take care of such exceptions by forcing the maximum community size to a known – and lower – s_{max} size. You can also set the minimum community size if you don’t want to have too many trivial communities in your network, using the s_{min} parameter.

The mixing parameter μ regulates how hard it is to find communities in the network. If $\mu = 0$ all edges run between nodes part of the same community, i.e. each community becomes a distinct connected

¹⁸ Andrea Lancichinetti, Santo Fortunato, and Filippo Radicchi. Benchmark graphs for testing community detection algorithms. *Physical review E*, 78(4):046110, 2008

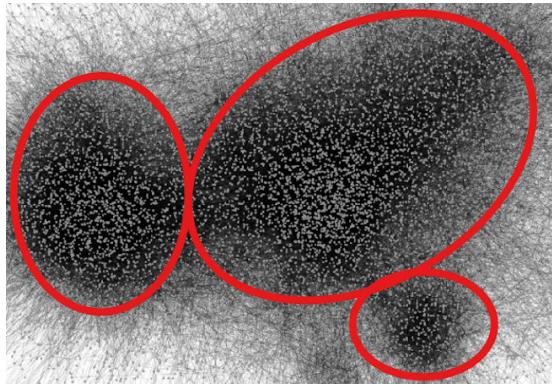


Figure 18.7: A messy real world network in which I highlighted with red outlines the three largest communities.

component of the network (Figure 18.8(a)). In this scenario, recovering community information is trivial. If $\mu = 1$ then nodes in the same community do not connect at all (Figure 18.8(d)). In this scenario, recovering the community wiring is impossible. Usually, you want to set μ to a reasonably low non-zero value. From Figures 18.8(b-c) you can see why this is the case: even the seemingly low value of $\mu = 0.2$ generates hard to distinguish communities (Figures 18.8(c)).

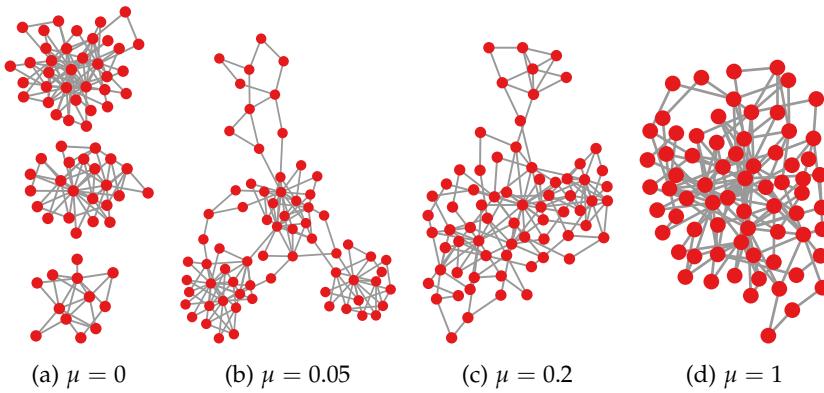


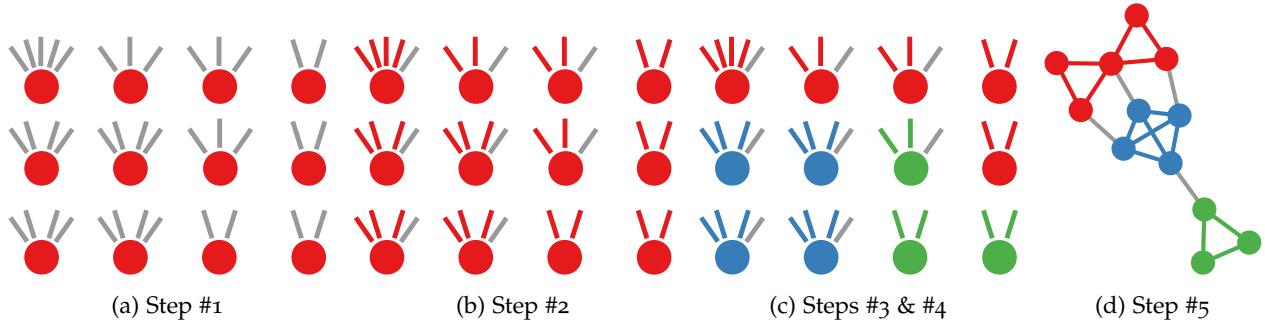
Figure 18.8: Examples of LFR benchmarks at different μ levels.

The basic LFR algorithm, which generates disjoint communities, works using the following strategy:

1. Generate the degree distribution of the network, with exponent α and whose minimum and maximum degree are k_{min} and k_{max} (Figure 18.9(a));
2. Mark a fraction μ of its edges as connecting outside the community and the rest as connecting inside the community (Figure 18.9(b));
3. Generate the community size distribution, with exponent β and whose minimum and maximum size are s_{min} and s_{max} ;

4. Assign each node v to a community c at random, ensuring that $k_v(1 - \mu) < s_c$, i.e. that the community will have enough nodes for v to connect to them¹⁹ (Figure 18.9(c));
5. Apply a modified configuration model to establish the edges. Each node v connects to $k_v(1 - \mu)$ random nodes in its community c , and to $k_v\mu$ random nodes outside c (Figure 18.9(d)).

¹⁹ If $k_v(1 - \mu) > s_c$, once you gave to v s_c internal edges, you still have internal edges to assign to v that cannot be attached to nodes inside c , because v is already connected to all its community mates, thus breaking the model. Remember that s_c is the size of community c , in number of nodes.



Note that, in light of step #4, you have some constraints in your choice of parameter. Specifically $k_{min} < s_{min}$ and $k_{max} < s_{max}$, otherwise the nodes with minimum and maximum degree will never belong to any community. Even with such constraints, sometimes the combination of parameters will require the generation of an impossible graph, so the LFR benchmark will always be some sort of approximation of your desires. In practice, differences are going to be relatively tiny and insignificant.

Since we're plugging in a power law degree distribution and communities, it is obvious that LFR benchmarks will reproduce these characteristics of real world networks well – although now you're actually *forced* to have a power degree distribution, which in some cases you might not want. They also respect clustering and small diameters, making them the most realistic model we have.

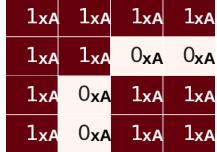
Kronecker Graphs

The idea of a Kronecker graph originates from the Kronecker product operation. The Kronecker product is the matrix equivalent of the outer product of two vectors. The outer product of two vectors u and v is a $|u| \times |v|$ matrix, whose i, j entry is the multiplication of u_i to v_j . The Kronecker product is the same thing, applied to matrices. Figure 18.10 shows an example. To calculate $A \otimes B$, we're basically multiplying each entry of A with B ²⁰.

When it comes to generating graphs, the matrix we're multiplying is the adjacency matrix. We usually multiply it with itself. So we're calculating $A \otimes A$, as I show in Figure 18.10(b). This generates a new

Figure 18.9: A run through a simple LFR model.

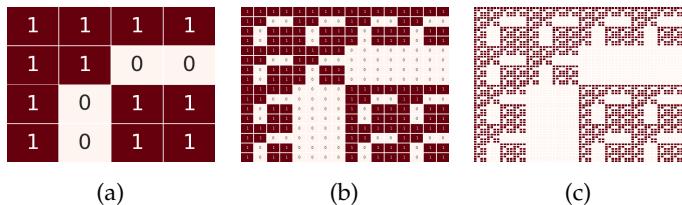
²⁰ G Zehfuss. Über eine gewisse Determinante. *Zeitschrift für Mathematik und Physik*, 3(1858):298–301, 1858

| | |
|---|---|
|  |  |
| (a) | (b) |

squared matrix, whose size is the square of the previous size. We can multiply this new adjacency matrix with our original one once more, for as many times as we want. We stop when we reach the desired number of nodes^{21,22}.

Figure 18.11 shows the progression of the Kronecker graph. Figure 18.11(a) is our seed graph which we multiply to itself (Figure 18.11(b)) twice (Figure 18.11(c)).

One small adjustment that is customary to do when generating a Kronecker graph is to fill the diagonal with ones instead of zeros. If you remember my linear algebra primer, this means we consider every node to have a self-loop to itself. This is because we want the Kronecker graph to be a block-diagonal matrix, with lots of connections around the diagonal. This is required if we want them to show a sort of community partition.



By how the Kronecker product is defined you can see that, if the seed matrix had an empty diagonal, we would not get a block diagonal matrix after applying the Kronecker product. Figure 18.12 shows an example, in which you can see the devastating effect on the graph's density of leaving the main diagonal empty. We can always reset the main diagonal to zero once we're done with the Kronecker products.

The question underlying generating graphs with an iterative Kro-

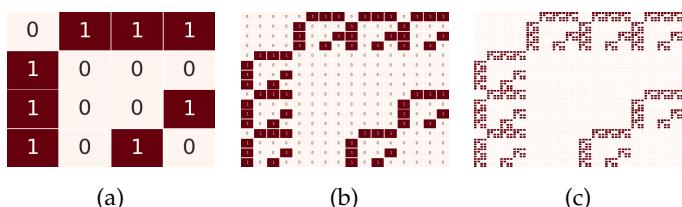


Figure 18.10: An example of Kronecker product. (a) A matrix A . (b) The operation we perform to obtain $A \otimes A$.

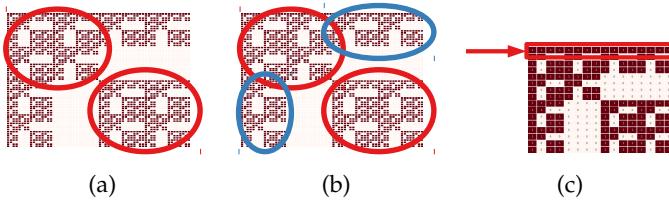
²¹ Juri Leskovec, Deepayan Chakrabarti, Jon Kleinberg, and Christos Faloutsos. Realistic, mathematically tractable graph generation and evolution, using kronecker multiplication. In *European Conference on Principles of Data Mining and Knowledge Discovery*, pages 133–145. Springer, 2005b

²² Jure Leskovec and Christos Faloutsos. Scalable modeling of real graphs using kronecker multiplication. In *Proceedings of the 24th international conference on Machine learning*, pages 497–504. ACM, 2007

Figure 18.11: An example of Kronecker graph. (a) The seed adjacency matrix. (b) Kronecker product of (a) with itself. (c) Kronecker product of (b) with (a).

Figure 18.12: An example of Kronecker graph, similar to the one in Figure 18.11, but without setting the main diagonal to one.

necker product is: why? Well, for starters, Kronecker graphs are fractals. Personally, I don't need any other reason than that. Look at Figure 18.11(c): if you tell me it doesn't speak to your heart then I question whether you're really human. If you're not an incurable fractal romantic like me, the deceptively simple process that generates Kronecker graphs solves all the issues we want from a graph generating process. In some cases, it is even better than LFR.



Kronecker graphs have high clustering and communities (Figure 18.13(a)), even hierarchical and overlapping (Figure 18.13(b)). They even have a hint of core-periphery structures, which I'll present fully in Chapter 32. They are small world (Figure 18.13(c)), and with a power-lawish degree distribution (usually shifted because they have few low degree nodes). LFR is preferred because it leaves space for the randomness of real world noise, but Kronecker graphs have the advantage of being more simple to understand and implement.

18.3 Random Geometric Graph

Another property you might want to preserve in your graph is the spatial structure. Many networks live on a physical space, and this physical space constraints the edge generating process. If two nodes are too far way from each other, they cannot connect. For instance, if two cities are at the antipodes of the globe, there might not be a plane able to fly directly from once city to the other. We can model these constraints using random geometric graphs^{23,24}.

The concept is simple. You first decide the dimensionality of your space: is it a 2D plane, three dimensional, or n-dimensional? Then you generate $|V|$ points in this space, by extracting them uniformly at random. Finally, you connect two points if they are at r distance – or less – from each other. Every point will be at distance zero from itself but, for the sake of simplicity, we ignore self-loops. Note that you are free to decide how to calculate the distance between points: you're not forced to use the Euclidean.

This is a rather simple way of generating random graphs. A random geometric graph is fully described by a handful of parameters: the number of nodes $|V|$ – which is the number of points you extracted –; the maximum distance r ; the number of dimensions; and

Figure 18.13: Some properties of Kronecker graphs. (a) Communities – circled in red –; (b) Communities (red) with their overlap (blue); (c) Small diameter – as the highlighted node in red is connected to every other node in the network making the diameter equal to two.

²³ Mathew Penrose et al. *Random geometric graphs*, volume 5. Oxford university press, 2003

²⁴ Jesper Dall and Michael Christensen. Random geometric graphs. *Physical review E*, 66(1):016121, 2002

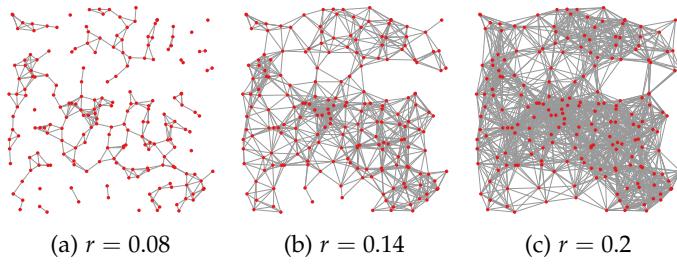


Figure 18.14: Random geometric graphs with 200 nodes. X and Y positioning of nodes are the same for all figures, but the r parameter increases from left to right, generating a higher number of longer edges.

the measure you used to calculate point-point distances. Figure 18.14 shows a few results. In the figure, I used a 2D plane and the Euclidean distance measure. The networks have the same number of nodes – in fact their coordinates are the same –, and I simply play with the r parameter.

There is a simple naive algorithm to generate random geometric graphs. First, you extract $|V|$ uniform random tuples – depending on your chosen dimensionality, for a 2D plane they'd be pairs. Then you calculate the pairwise distance between all of them and connect the nodes if the distance is lower than r . There are smarter algorithms²⁵, especially designed to avoid computing all pairwise distances – and exploiting parallel processing.

Assuming that you use the Euclidean distance, you can derive the probability of having a given number of isolated vertices or a value of clustering coefficient by looking at the parameters you used to generate the network. In general, giant connected components appear easily in these types of graphs, provided that $|V|e^{-\pi r^2 |V|} < 1$. This magic value derives from the fact that the expected degree of a node is $\pi r^2 |V|$, given that it will connect to all nodes in a circle around it – which has an area of πr^2 . For instance, in Figure 18.14(a), $r = 0.08$ and thus $|V|e^{-\pi r^2 |V|} \sim 3.6$, which allows for a few isolated nodes; while in Figure 18.14(c) this value is $\sim 2 \times 10^{-9}$, which makes the presence of a single connected component almost certain.

You can also have probabilistic random geometric graphs²⁶. The difference is that you do not always connect nodes at distance lower than r with probability 1, but rather with some probability $p < 1$.

18.4 Graph Generative Networks

One thing that all models discussed so far have in common is that they are engineered to have specific properties. These are the properties we think are salient in real world networks: broad degree distributions, community structures, etc. But what if we are wrong? Maybe some of these properties are not the most relevant things about a network we want to model. Moreover: what if there are other

²⁵ Daniel Funke, Sebastian Lamm, Ulrich Meyer, Manuel Penschuck, Peter Sanders, Christian Schulz, Darren Strash, and Moritz von Looz. Communication-free massively distributed graph generation. *Journal of Parallel and Distributed Computing*, 131: 200–217, 2019.

²⁶ Bernard M Waxman. Routing of multipoint connections. *IEEE journal on selected areas in communications*, 6(9): 1617–1622, 1988.

properties that we aren't seeing? Edges are dependent on each other, but these dependencies can be complex and it's difficult to put them in simple measures we can then optimize. Here I will describe a couple of approaches in isolation. However, to make more sense of the neural network lingo, you should look up Section 45.3.

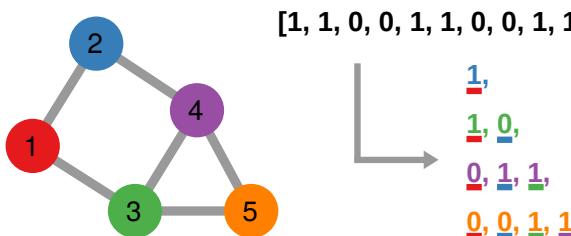
The field of graph generative networks^{27,28,29} aims at tackling this problem. Here we want to generate networks that look like specific real world networks, without us knowing what "looking like" actually means. In other words, we want the generative process to "learn" how a real world network looks like, so that it can generate synthetic versions at will.

The most trivial way you can do this is by feeding the adjacency matrix of your graph – or a suitably modified version of it – to an algorithm that can learn the dependencies between edges. You can think of this approach as an SBM process without inferring the communities beforehand. SBM wants to preserve the community structure and, on this basis, learns edge probabilities that only depend on the community affiliation. Here, we want to preserve the general network properties, thus each edge probability is dependent on the entirety of the adjacency matrix.

However, this approach has two problems. First, it will only generate a graph with the same number of nodes as the input, while you might want to vary the size of your synthetic networks. Second, it can only learn from a single graph at a time. Sometimes, you might want to model a class of graphs.

These limitations are solved in a variety of ways. Just to give an example, GraphRNN³⁰ allows for two moves: a graph-level update and an edge-level update. In the first step, GraphRNN adds a new node into the network. Every time a new node is added, the edge-level update is triggered, determining to which nodes the new node connects. This is achieved by representing the graph as a sequence. For each node, in order, we list to which of the previous nodes it connects. For instance, the sequence [1, 1, 0, 0, 1, 1, 0, 0, 1, 1] corresponds to the graph in Figure 18.15.

To see why, it is useful to break down the sequence in sections, each one referring to a node, as the figure does. The first node has



²⁷ Yujia Li, Oriol Vinyals, Chris Dyer, Razvan Pascanu, and Peter Battaglia. Learning deep generative models of graphs. *arXiv preprint arXiv:1803.03324*, 2018

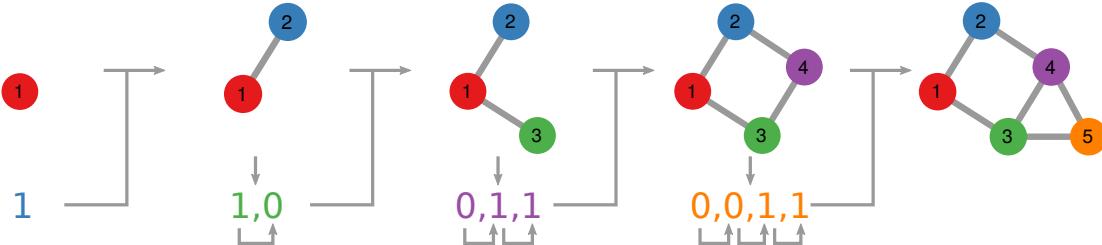
²⁸ Nicola De Cao and Thomas Kipf. Molgan: An implicit generative model for small molecular graphs. *arXiv preprint arXiv:1805.11973*, 2018

²⁹ Aleksandar Bojchevski, Oleksandr Shchur, Daniel Zügner, and Stephan Günnemann. Netgan: Generating graphs via random walks. In *International Conference on Machine Learning*, pages 609–618, 2018

³⁰ Jiaxuan You, Rex Ying, Xiang Ren, William Hamilton, and Jure Leskovec. Graphrnn: Generating realistic graphs with deep auto-regressive models. In *International Conference on Machine Learning*, pages 5694–5703, 2018

Figure 18.15: A graph and its sequence representation in GraphRNN. Each element in the sequence belongs to a node (character color) and records whether that node connects to a specific node preceding it in the sequence (underline color).

no element in the sequence, because it has no preceding nodes. The second node contributes only one element to the sequence: 0 if it doesn't connect to the first node, 1 otherwise. The third node contributes two values, one for its edge with the first node and one for the second node, and so on.



In practice, GraphRNN expands the sequence, by adding the n th node (graph-level update) as a new subsequence of length $n - 1$ (edge-level update). Figure 18.16 shows a simple iteration. These updates are implemented via autoregressive models (see Section 45.3).

18.5 Summary

1. The configuration model is a way to have a synthetic network with an arbitrary degree distribution. However, if you don't allow for parallel edges or self loops, the degree distribution is likely only going to be approximated.
2. Stochastic block models can recreate a community structure by taking as input a node partition and the probabilities of connecting to nodes inside the same community and between communities.
3. The GN and LFR benchmark were created to test community discovery algorithms. The GN benchmark creates equal size communities and normal degree distributions, while LFR is able to return power-law degree distributions and communities of varying size.
4. Kronecker graphs are generated from a simple seed matrix to which you recursively apply the Kronecker product, creating high clustering networks with shifted power law degree distributions and communities.
5. Alternatively, you can make a random geometric graph, by placing nodes uniformly on an n-dimensional space and connecting nodes to all their closest neighbors, at a maximum distance that you can set as parameter.

Figure 18.16: The GraphRNN workflow. From left to right, we progressively add new nodes, in the form of an extension of the sequence representing the connections.

6. Finally, you can learn a neural network representation from your original (set of) network(s), which will be able to generate more synthetic networks with comparable properties to your original one(s).

18.6 Exercises

1. Generate a configuration model with the same degree distribution as the network in <http://www.networkatlas.eu/exercises/18/1/data.txt>. Perform the Kolmogorov-Smirnov test between the two degree distributions.
2. Remove the self-loops and parallel edges from the synthetic network you generated in the previous question. Note the % of edges you lost. Re-perform the Kolmogorov-Smirnov test with the original network's degree distribution.
3. Generate an LFR benchmark with 100,000 nodes, a degree exponent $\alpha = 3.13$, a community exponent of 1.1, a mixing parameter $\mu = 0.1$, average degree of 10, and minimum community size of 10,000. (Note: there's a networkx function to do this). Can you recover the α value by fitting the degree distribution?
4. Use `kron` function from `numpy` to implement a Kronecker graph generator. Plot the CCDF degree distribution of a Kronecker graph with the following seed matrix multiplied 4 times (setting the main diagonal to zero once you're done):

$$A = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix}$$

19

Evaluating Statistical Significance

One of the big issues when it comes to analyzing complex networks is that, usually, you only have one network to base your observations on. Therefore, whenever you observe a given property – power law degree distribution, reciprocity, clustering – you don't have the statistical power to claim that what you're observing is interesting. You need to have multiple versions of your network, a null model, to test your observation. If keeping everything fixed about a network minus the property of interest gives you something indistinguishable from your observation, then you know that the particular feature arose at random. There is no fundamental non-random force behind it.

To do so, you need to generate a (set of) synthetic graph(s), which is why I put this chapter in this part of the book. In other words, you consider your observed network as part of a family of networks, which all have the same fixed properties. Then you ask if the one you did not fix is also a typical characteristic of this family of networks. If it is, then it's not an interesting discovery. If it isn't, the deviation of the network from its family is interesting.

As far as I know, there are two ways to generate this network family: the easy way – network shuffling, Section 19.1 –, and the right way – the Exponential Random Graph approach, Section 19.2.

19.1 Network Shuffling

Network shuffling is a way to generate synthetic networks that is based on directly manipulating your observed network. At a fundamental level, it is a process of rewiring edges for a given number of times, until we think that we are sufficiently far from the starting point. I call this the “easy” way because, as we'll see in a moment, the process is usually straightforward. On the other hand, this method is significantly less rigorous than the Exponential Random Graph model (ERGM), and thus should be used only when you don't

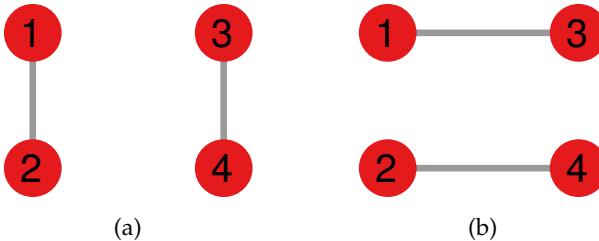


Figure 19.1: The edge swap procedure.

need the statistical power ERGM can give you.

The fundamental basis of the network shuffling model is the edge swap operation. Figure 19.1 depicts it in all its simplicity. You pick two pairs of connected nodes, in this examples node 1 is connected to node 2, and node 3 is connected to node 4. Then, you flip the edge around, deleting (1,2) and replacing it with (1,3), and deleting (3,4) replacing it with (2,4). If you do this enough times, the resulting network will be quite different from your original one. However, it will still have the same number of nodes, the same number of edges, and the same degree distribution – also in case of a directed network, provided that you always swap edges in the correct direction.

This procedure is usually performed in network games, where edges are rewired with some objective in mind¹. Note that the number of swaps to perform before stopping is a non trivial quantity to evaluate².

An attentive reader will surely notice that this result is practically the same as the one you would obtain from a configuration model. However the two approaches have completely different objectives. The configuration model wants to simply generate a network with more or less the same degree distribution. The networks generated by shuffling are significantly more similar to the original network than the ones obtained from a configuration model, because they need to be compared to it.

This becomes more obvious once you explore the differences with the configuration model more in depth. First, edge swapping is always possible, while at some point in the configuration model process you might have to create self-loops or parallel edges. You cannot create self-loops in network shuffling unless there were already self loops in the original network. You can easily avoid parallel edges by checking that your node pairs are not connected – for instance, you can reject the operation in the example in Figure 19.1 if either the (1,3) or the (2,4) edges are already in the network.

Second, what I explained is only the simplest way to perform network shuffling. You can add a few more features that are hard to embed in a configuration model. For instance, you can keep fixed the number of connected components, by rejecting an edge swap if it

¹ Noga Alon, Erik D Demaine, Mohammad T Hajiaghayi, and Tom Leighton. Basic network creation games. *SIAM Journal on Discrete Mathematics*, 27(2): 656–668, 2013

² Giulio Bottazzi and Davide Pirino. Measuring industry relatedness and corporate coherence. 2010

would disconnect more nodes, or join two different components. You can keep the clustering fixed, by making sure to keep the number of triads and triangles constant. You can preserve the communities, by only allowing edge swaps inside the clusters. And so on.

If all you want to do is to have a randomized version of your original network, then you're done. But, since I mentioned the problem of determining the statistical significance of your observations, let's push on. How would you use the networks generated via edge swap for such a task? I usually apply the following procedure:

1. Fix all reasonable properties of the network (at least number of nodes, edges, and degree distribution) except the one of interest;
2. Generate a large set of shuffled networks, with independent shuffles – the number of shuffles depends on the number of edges;
3. Calculate the property of interest in the observed and in the generated networks;
4. Estimate the distribution of the property in the shuffled networks and calculate how far from the expectation the observed value is.

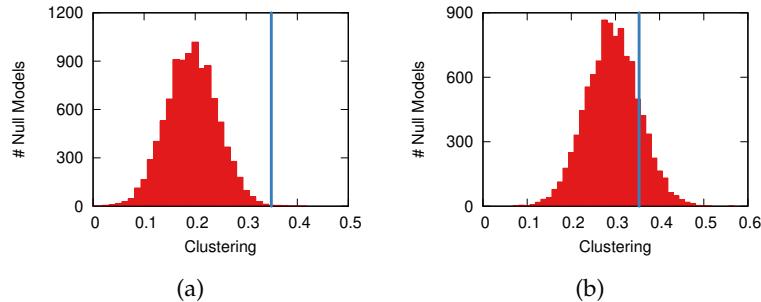


Figure 19.2: The edge swap statistical test. The plot shows how many null models (y-axis) scored a given value of the property of interest (in this case clustering, x-axis). The blue vertical bar shows the observation.

Usually, this ends up with a plot looking like Figure 19.2(a) or 19.2(b). The histogram shows how many null models scored a value of the property of interest in a given interval. The blue vertical bar shows the value for the observed network.

In the easiest scenario, the null model will show a nice normal or pseudo-normal distribution, making the estimation of statistical significance easier. That's the case for the figures I show, which means I can simply calculate how many standard deviations from the average my observation is. In the case of Figure 19.2(a) the observation is significantly higher than expectation, given it's three standard deviations away from the average. That is not true for Figure 19.2(b): in that case, we cannot reject the null explanation. The observation is less than a standard deviation away from null expectation.

By the way, the thing that I call “number of standard deviations above (or below) average” is known as z-score. You can automatically convert from the z-score into a p-value (Section 3.3), provided that you know whether you’re interested in a one-sided or a two-sided test. The one-sided test means that your success is exclusively on one side of the distribution – e.g. you want to score more than average, you’re not interested whether your score is significantly below average³ (or vice versa).

Of course, if the null model distribution is not pseudo-normal, estimating the statistical significance is a bit trickier. We don’t need to go into that, because we’re about to learn how to perform this task in the “right” way, using ERGMs.

19.2 Exponential Random Graphs

As introduced in this chapter, ERGM is a technique to generate a set of graphs that have the same properties of an observed network. ERGM is also known in the literature as p^* model^{4/5}. The observed network is seen as the result of a stochastic (random) process with a set of parameters. ERGM creates other networks using the same process and the same parameters. The problem we need to solve to generate an exponential random graph ensemble of networks is figuring out which parameter values to use. This is usually achieved through maximizing the likelihood function I introduced in Section 4.3.

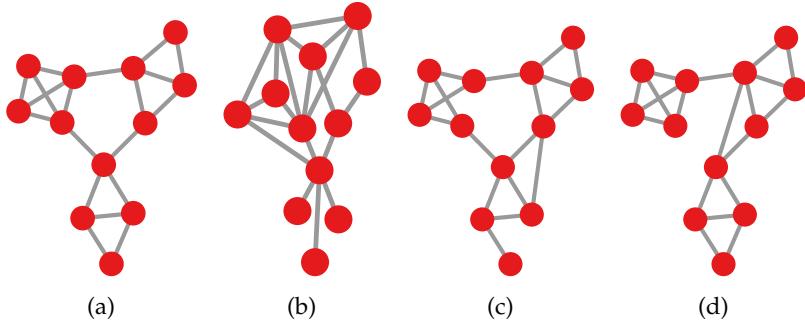
The sketch of the solution is the following. Suppose you’re observing a graph G . There is an immensely large set of other random G graphs we could have observed: they are those we could generate with a random process with a given set of parameters (same number of nodes, edges, ...). However, some of these random graphs are more likely than others to be observed – namely, the ones most similar to G . Knowing this, we can identify the parameters values these likely G s have in common. Once we find these values, we can generate an arbitrary number of graphs with them. We can use them as new synthetic graphs for our purposes, or we can test them against the observed G to verify whether they also share with it some other property we did not fix. Figure 19.3 provides a vignette of this process.

If you talk statistics, the following process might give you an inkling about how ERGMs work. In this scenario, we consider an edge as a random variable. In the simplest case of unweighted network, this will be a binary variable, equal to one if the edge is present, and to zero if it isn’t. You hypothesize what sort of process might be the one determining the edge presence in your network. This is sort of similar to estimating a logistic regression. You have

³ When discussing discoveries in physics, you’ll hear often the term “five sigma” (5σ) thrown around. This means a z-score equal to 5. In turn, this can be converted to a (one-sided) p-value lower than 10^{-6} , way lower than the $p < 0.01$ you’ll see in other fields. For $p < 0.01$, you’re looking at a z-score a bit higher than 2.3. I’m simplifying a lot here, since this is not – and never will be – a statistics book.

⁴ Carolyn J Anderson, Stanley Wasserman, and Bradley Crouch. A p^* primer: Logit models for social networks. *Social networks*, 21(1):37–66, 1999

⁵ Garry Robins, Pip Pattison, Yuval Kalish, and Dean Lusher. An introduction to exponential random graph (p^*) models for social networks. *Social networks*, 29(2):173–191, 2007



a binary outcome (edge present/absent) and a set of variables that might be able to predict its value.

Let's make an example. Figure 19.4(a) represents our observed graph. I generated it with a configuration model with the degree sequence $(9, 4, 3, 3, 2, 2, 2, 1, 1, 1, 1, 1)$, but the ERGM doesn't know that. The edge presence, the outcome, is our adjacency matrix A . So the edge between u and v is $A_{u,v}$. I now make an hypothesis: the degree of a node influences its likelihood of getting a connection. Or, in mathematical terms, $A_{u,v} = \beta_1 k_u + \beta_2 k_v$. The degrees of u and v (k_u and k_v) can be used to predict the probability of existence of an edge. This is equivalent of running a logit regression on an edge table like the one in Figure 19.4(b): we're trying to predict the binary $A_{u,v}$ variable using the degrees of u and v .

Once the logit model is done, for each (u, v) pair we have a probability of its existence: $\overline{A_{u,v}}$ (Figure 19.4(c)). We can now flip a loaded coin for each node pair and add the edge in case of success. $\overline{A_{u,v}}$ is determined by the β_1 and β_2 parameters. Since they are the result of a logit model estimation, they are the ones most likely to describe the family of random graphs from which we extracted the observed

Figure 19.3: An illustrative example of the ERGM process. (a) Observed network. (b) Unlikely random network. (c) Random network more likely to be in the same family of (a) than (b). (d) Most likely random network. The parameters used to generate it are more likely to be the ones of the family of random graphs to which (a) belongs.

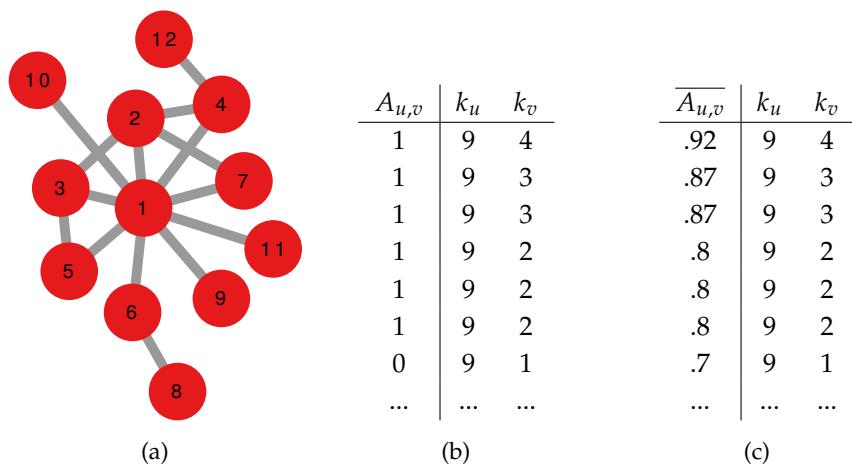
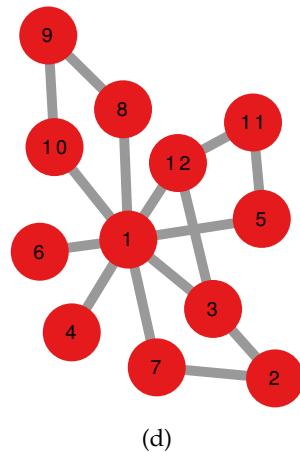


Figure 19.4: Step-by-step example of a simple ERGM process. (a) Observed network. (b) Observed edge table (only first seven rows). $A_{u,v}$ is one if the edge is present, zero otherwise; k_u and k_v are the degrees of the two nodes. (c) Result of the logit regression (only first seven rows). $\overline{A_{u,v}}$ is the estimated probability of the edge existing. (d) An extracted ERGM from the edge probabilities in (c).



G. By using Figure 19.4(c) to generate a new graph (e.g. the one in Figure 19.4(d)), we're sure to extract a graph from the same family that generated the original one – at least when it comes to its degree distribution.

So far, I simplified the process for the sake of intuition. For instance, I assumed that the likelihood of an edge only depends on a node's characteristic – in this case the degree. This is not necessarily the case in the general ERGM. You can plug in all complex structures you can express mathematically. For instance, you can ensure triadic closure to preserve the clustering coefficient or other, more complex, motifs. I also assumed the functional form of the model, namely a linear one. The degrees of the two nodes interact linearly to give us the result. That might not be the case.

We can describe the full model making no such assumptions as:

$$Pr(A = A') = \frac{1}{B} \exp \left(\sum_g \beta_g g_{A'} \right).$$

There's a bit to unpack here:

- $Pr(A = A')$ is the probability that the adjacency matrix A we extract is equal to a given adjacency matrix A' , dependent on A' 's characteristics.
- \exp is the exponential function. We use it to define the probability because exponentials come from maximum entropy distributions. We want to use a function that can have the highest possible entropy while still having a positive and definite mean – which is necessary to define a probability (see Section 2.3). "Highest possible entropy" simply means that whatever statistic we haven't incorporated in the model will be "as random as possible".
- g is a graph configuration. It can be any pattern, for instance a triangle, or a clique of four nodes, or even just an edge.
- β_g is the parameter corresponding to this particular configuration. In our previous example, it is the thing telling you how much the degree of a node influences the connection probability. This is the knob you have to use to maximize your quality function. The "right" β_g value is the one best describing your data.
- $g_{A'}$ is a function applying pattern g to A' . It tells you whether the pattern is in the network. Mathematically: $g_{A'} = \prod_{A'_{uv} \in A'} A'_{uv}$, which means that $g_{A'} = 1$ if and only if all parts of g are in A' .
- B is simply a normalization parameter needed to ensure that the rest of the equation is a proper probability distribution – i.e. that it sums to one.

To use human language: the probability of observing an adjacency matrix A is the probability of extracting a random A' from all possible adjacency matrices, weighted by how well A' 's topological properties fit the β parameters we observed in our original graph, over the patterns g that interest us – any other pattern not in g is assumed to appear entirely at random.

Such a model can have lots of β parameters. That is why usually there is an additional step of parameter reduction, through what we call “homogeneity constraints”. For instance you could have a parameter for each node, telling us how likely that node is to reciprocate connections. However, the homogeneity assumption says that – most likely – all nodes in the same network have more or less the same tendency of reciprocating connections. Thus, rather than having $|V|$ reciprocity parameters, you have a single, network-wide, one.

This functional form is a general version of more specific ones that were studied in the literature in the eighties: p_1 models⁶ and Markov graphs⁷.

To understand a bit more the magic behind the formula I just presented, let's consider a few special cases. Given their general form, ERGMs include many of the network models we saw so far. For instance, we can represent a $G_{n,p}$ model, by noting that, in this case, edges are all independent to each other. Without the homogeneity assumption, we would have a parameter for each pair of nodes, giving us the equation:

$$Pr(A = A') = \frac{1}{B} \exp \left(\sum_{u,v} \beta_{u,v} A'_{u,v} \right).$$

In this case, the graph pattern g is a single u,v edge. Since $g_{A'}$ is equal to one if A' contains pattern g , it reduces to $A'_{u,v}$, which is one if A' contains the u,v edge. Further, we have a different $\beta_{u,v}$ (β_g) per edge. Edges present in our observed network will have corresponding high $\beta_{u,v}$ parameters, while absent edges will have $\beta_{u,v}$ values close to zero. This in turn implies that an A' is likely to be extracted if it has edges attached to high $\beta_{u,v}$ values.

However, as we said, this is too many parameters. If we apply the homogeneity assumption, we will just say that any pair of nodes has the same probability p of connecting – which is the same assumption of the $G_{n,p}$ model. This means that we get rid of a lot of parameters, which are substituted by the single parameter p :

$$Pr(A = A') = \frac{1}{B} \exp \left(\sum_{u,v} p A'_{u,v} \right).$$

⁶ Paul W Holland and Samuel Leinhardt. An exponential family of probability distributions for directed graphs. *Journal of the american Statistical association*, 76(373):33–50, 1981

⁷ Ove Frank and David Strauss. Markov graphs. *Journal of the american Statistical association*, 81(395):832–842, 1986

Since $\sum_{u,v} A'_{u,v}$ is simply the number of edges $|E'|$, this simplifies to:

$$\Pr(A = A') = \frac{1}{B} \exp(p|E'|),$$

which is exactly a $G_{n,p}$ model: a graph whose edges are all equally likely to be observed. We can also simulate a stochastic blockmodel by not reducing all connection probabilities to p but by having multiple p_s for each block (and for inter-block connections). If you have a directed graph you can represent reciprocity with the probability p_1 of a node to reciprocate the connection, adding a term to the $G_{n,p}$ model:

$$\Pr(A = A') = \frac{1}{B} \exp(p|E'| + p_1 R(A')).$$

Here $R(A')$ is the number of reciprocated ties. You can make edges dependent on node attributes as researchers do in p_2 models^{8,9}. Finally, you can also plug higher-order structures in the model, for instance:

$$\Pr(A = A') = \frac{1}{B} \exp(p|E'| + \tau T(A')),$$

where – under the homogeneity assumption – $T(A')$ is the number of triangles in A' . This way, you can also control the transitivity of the graph.

These models can be very difficult to solve analytically for all but the simplest networks. Modern techniques rely on Monte Carlo maximum likelihood estimation^{10,11}. We don't need to go too much into details, but these work similarly to any Markov chain Monte Carlo method¹². However, if your network is dense, your estimation might need to take an exponentially large number of samples to estimate your β s¹³. There are ways to get around this problem by expanding the ERG model¹⁴, but by now we're already way over my head and I don't think I can characterize this fairly.

How does all of this look like in practice? The result of your model might look like something from Figure 19.5. Here we decide to have four parameters: a single edge (this is always going to be present in any ERGM), a chain of three nodes, a star of four nodes, and a triangle. Each motif has a likelihood parameter: the higher the parameter the more likely the pattern. Negative values mean that the pattern is less likely than chance to appear.

The negative value for simple edges means that the network is sparse: two nodes are unlikely to be connected. The positive value for the triangle means that triangles tend to close: when you have a triad, it is more likely than chance to have the third edge. The other two configurations are not significantly different from zero (you can't

⁸ Emmanuel Lazega and Marijtje Van Duijn. Position in formal structure, personal characteristics and choices of advisors in a law firm: A logistic regression model for dyadic network data. *Social networks*, 19(4):375–397, 1997

⁹ Marijtje AJ Van Duijn, Tom AB Snijders, and Bonne JH Zijlstra. p2: a random effects model with covariates for directed graphs. *Statistica Neerlandica*, 58(2):234–254, 2004

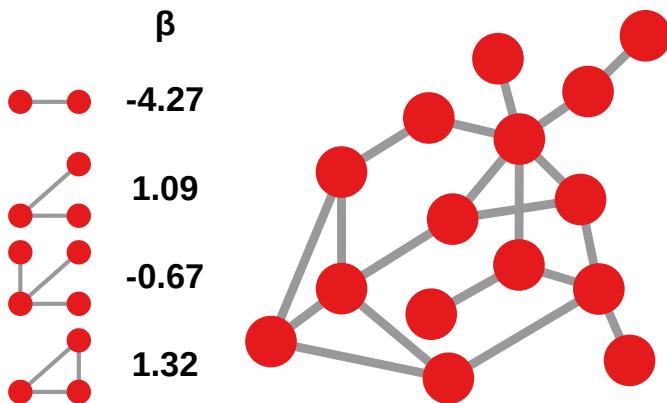
¹⁰ Tom AB Snijders. Markov chain monte carlo estimation of exponential random graph models. *Journal of Social Structure*, 3(2):1–40, 2002

¹¹ Tom AB Snijders, Philippa E Pattison, Garry L Robins, and Mark S Handcock. New specifications for exponential random graph models. *Sociological methodology*, 36(1):99–153, 2006

¹² Walter R Gilks, Sylvia Richardson, and David Spiegelhalter. *Markov chain Monte Carlo in practice*. Chapman and Hall/CRC, 1995

¹³ Shankar Bhamidi, Guy Bresler, and Allan Sly. Mixing time of exponential random graphs. In *2008 49th Annual IEEE Symposium on Foundations of Computer Science*, pages 803–812. IEEE, 2008

¹⁴ Arun G Chandrasekhar and Matthew O Jackson. Tractable and consistent random graph models. Technical report, National Bureau of Economic Research, 2014



tell because I omitted the standard errors, but trust me on that). Thus we should not emphasize their interpretation too much.

On the right side of the figure you can see a potential network that is very likely to be extracted by this ERGM. In fact, I cheated a bit, because that is the network on which I fitted the model. It is the famous graph mapping the business relationship between Florentine families in the Renaissance¹⁵.

In this chapter I presented only the simplest of the ERGM forms. Recent research has shifted to more sophisticated models. A few of those are:

- Longitudinal ERGMs¹⁶, which are specialized to deal with networks that are evolving over time, for instance co-sponsorship of bills in the US Congress – two representatives might co-sponsor a bill in one year, but not in another;
- Similarly, TERGMs¹⁷ introduce the temporal aspect in ERGMs. This contains the “separable” TERGMs¹⁸ which works on discrete models, rather than modeling the evolution as happening on a continuous time flow;
- ERGMs that can take into account edge weights, initially only continuous weights¹⁹, but subsequently also discrete ones²⁰;
- ERGMs for multilayer networks²¹.

ERGMs have been successfully applied in many fields. For instance, they help in cases in which longitudinal network data collection is unfeasible – e.g. informal face to face contacts in certain business clusters²², or inside firms²³. In economics they are particularly useful because of their ability to estimate structural network parameters, extending conventional analyses that use, for instance, gravity models²⁴. To make an example, in migration a gravity model would say that the number of migrants from country u to v is directly

Figure 19.5: On the left we have the estimated parameters from the observation for four patterns, with positive values indicating a “more than chance” occurrence of the pattern, and negative values a “less than chance”. On the right we have a likely network extracted from the set of ERGM with the given parameters.

¹⁵ John F Padgett and Christopher K Ansell. Robust action and the rise of the medici, 1400–1434. *American journal of sociology*, 98(6):1259–1319, 1993

¹⁶ Skyler J Cranmer and Bruce A Desmarais. Inferential network analysis with exponential random graph models. *Political analysis*, 19(1):66–86, 2011

¹⁷ Steve Hanneke, Wenjie Fu, Eric P Xing, et al. Discrete temporal models of social networks. *Electronic Journal of Statistics*, 4:585–605, 2010

¹⁸ Pavel N Krivitsky and Mark S Handcock. A separable model for dynamic networks. *Journal of the Royal Statistical Society. Series B, Statistical Methodology*, 76(1):29, 2014

¹⁹ Bruce A Desmarais and Skyler J Cranmer. Statistical inference for valued-edge networks: The generalized exponential random graph model. *PloS one*, 7(1):e30136, 2012

²⁰ Pavel N Krivitsky. Exponential-family random graph models for valued networks. *Electronic journal of statistics*, 6: 1100, 2012

²¹ Alberto Caimo and Isabella Gollini. A multilayer exponential random graph modelling approach for weighted networks. *Computational Statistics & Data Analysis*, 142:106825, 2020

²² Pierre-Alexandre Balland, José Antonio Belso-Martínez, and Andrea Morrison. The dynamics of technical and business knowledge networks in industrial clusters: Embeddedness, status, or proximity? *Economic Geography*, 92(1):35–60, 2016

²³ Tom Broekel and Matté Hartog. Explaining the structure of inter-organizational networks using exponential random graph models. *Industry and Innovation*, 20(3):277–295, 2013

proportional to the size – in number of inhabitants – of the two countries, and inversely proportional to their distance. ERGMs allow you to model more complex interdependencies²⁵.

19.3 Summary

1. Network shuffling is a way to create a null version of your network, by performing edge swapping. In edge swapping, you pick two pairs of connected nodes and you rewire the edges to connect nodes from the other pair.
2. Once you generate thousands of null versions of your network, you can test a property of interest and obtain an indication of how statistically significant your observation is, by counting the number of standard deviations between the observation and the null average.
3. In Exponential Random Graphs you use a series of characteristics of the network of interest as a predictor of the presence of an edge between two nodes.
4. Once you know the relationship between these parameters and the presence of an edge, you can randomly extract graphs that are likely results of such predictors.
5. There is a trade off between the number of parameters you can use and the complexity of the extraction process. Many different heuristics have been proposed to sample the space of all possible ERGMs.

19.4 Exercises

1. Perform 1,000 edge swaps, creating a null version of the network in <http://www.networkatlas.eu/exercises/19/1/data.txt>. Make sure you don't create parallel edges. Calculate the Kolmogorov-Smirnov distance between the two degree distributions. Can you tell the difference?
2. Do you get larger KS distances if you perform 2,000 swaps? Do you get smaller KS distances if you perform 500?
3. Generate 50 $G_{n,m}$ null versions of the network in <http://www.networkatlas.eu/exercises/19/3/data.txt>, respecting the number of nodes and edges. Derive the number of standard deviations between the observed values and the null average of clustering and average path length. (Consider only the largest connected component) Which of these two is statistically significant?

²⁴ Tom Broekel, Pierre-Alexandre Balland, Martijn Burger, and Frank van Oort. Modeling knowledge networks in economic geography: a discussion of four methods. *The annals of regional science*, 53(2):423–452, 2014

²⁵ Michael Windzio. The network of global migration 1990–2013: Using ergms to test theories of migration between countries. *Social Networks*, 53: 20–29, 2018

4. Repeat the experiment in the previous question, but now generate 50 Watts-Strogatz small world models, with the same number of nodes as the original network and setting $k = 16$ and $p = 0.1$.

Part VI

Spreading Processes

20

Epidemics

So far, we've seen some dynamics you can embed in your network. In Section 7.4 I showed you how to model graphs whose edges might appear and disappear, while in the previous book part we've seen models of network growth: nodes arrive steadily into the network and we determine rules to connect them such as in the preferential attachment model. This part deals with another type of dynamics on networks. Here, edges don't change, but nodes can transition into different states.

The easiest metaphor to understand these processes is disease. Normally, people are healthy: their bodies are in a homeostatic state and they go about their merry day. However, they also constantly enter into contact with pathogens. Most of the times, their immune systems are competent enough to fend off the invasion. Sometimes this does not happen. The person transitions into a different state: they now are sick. Sickness might be permanent, but also temporary. People can recover from most diseases. In some cases, recovery is permanent, in others it isn't.

These are all different states in which any individual might find themselves at any given time. Like individuals, nodes too can change state as time goes on. This book part will teach you the most popular models we have to study these state transitions. In this chapter we look at three models we defined to study the progression of diseases through social networks¹. Note that such models can easily represent other forms of contagion, for instance the spread of viruses in computer and mobile networks².

We're going to complicate these models in Chapter 21, to see how different criteria for passing the diseases between friends affect the final results. Then, in Chapter 22, we'll see how the same model can be adapted to describe other network events, such as infrastructure failure and word-of-mouth systems to aid a viral marketing campaign.

Another complication is the one introduced by simplicial spread-

¹ Romualdo Pastor-Satorras, Claudio Castellano, Piet Van Mieghem, and Alessandro Vespignani. Epidemic processes in complex networks. *Reviews of modern physics*, 87(3):925, 2015

² Pu Wang, Marta C González, César A Hidalgo, and Albert-László Barabási. Understanding the spreading patterns of mobile phone viruses. *Science*, 324(5930):1071–1076, 2009

ing. If you remember (Section 7.3 is there if you don't), simplicial complexes contain these simplices, linking together multiple nodes in higher-order structures (triangles, 4-cliques, etc). In some cases, to be infected you might need to be part of such a complex structure. For instance, peer-pressure might not work well if you're only connected by an edge. However, if you're part of a simplicial complex of three nodes, that might be enough to trigger you. The combined pressure from your two friends overcomes your resistance. I'll expand on this in Section 34.1, when we'll do a deep dive into this kind of high order interactions.

20.1 SI

Sickness can be fatal for the individual and extremely debilitating for entire societies. If tomorrow an epidemic sends to bed 90% of the population at the same time – even if it doesn't kill anyone – it can grind the planet to a halt³. For this reason, humans have a strong incentive to study contagion dynamics at large, to predict whether such a situation might occur in the future. Researchers have developed simple models to describe the dynamics of diseases^{4,5,6}. These are usually known as "Compartmental models" – although I've been calling "state" what is traditionally known as "compartment".

The model divides individuals into two states. The first state is called Susceptible. A person in the Susceptible state is... well... susceptible to contract a disease. This marks healthy people that show no symptoms and are functioning properly. The second state is called Infected. People in the Infected state – you'll never believe it – have contracted the disease. We use S to indicate the set of individuals in the Susceptible state, and I to indicate the set of individuals in the Infected state.

The model allows for only one possible transition between states. The only thing that can happen in this model is the transition from the Susceptible to the Infected state: $S \rightarrow I$. In this world, the only possible action is for a healthy person to contract the disease. Nothing else is allowed.

Given that there are only two states (S and I) and only one transition ($S \rightarrow I$), we call this the SI Model. Figure 20.1 shows the schema fully defining the model. In practice, SI models diseases with no recovery. An example would be some variants of the herpes virus. Love goes by, herpes is forever.

There is one assumption underlying the traditional SI Model: **homogenous mixing** – keep this in mind because it's important. In homogenous mixing, we assume that each susceptible individual has the same probability to come into contact with an infected person.

³ This chapter was drafted before COVID-19 happened, and it shows.

⁴ William Ogilvy Kermack and Anderson G McKendrick. A contribution to the mathematical theory of epidemics. *Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character*, 115(772):700–721, 1927

⁵ David Clayton, Michael Hills, and A Pickles. *Statistical models in epidemiology*, volume 161. Oxford university press Oxford, 1993

⁶ Abdel R Omran. The epidemiologic transition: a theory of the epidemiology of population change. *The Milbank Quarterly*, 83(4):731–757, 2005

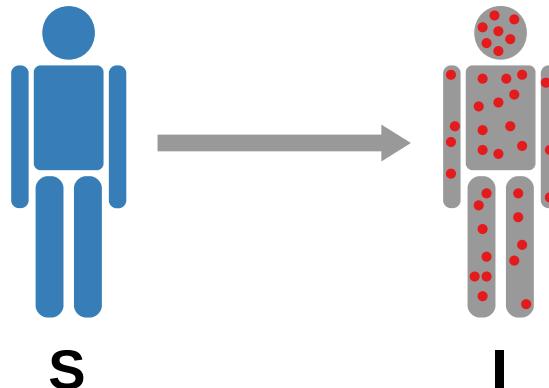


Figure 20.1: The schema underlying the SI Model: two possible states and one possible transition.

This is simply determined by the current fraction of the population in the infected state. Once the susceptible individual meets an infected, there is a probability that they will transition into the I state too. This probability is a parameter of the model, traditionally indicated by β . If $\beta = 1$, any contact with an infected will transmit the disease, while if $\beta = 0.2$, you have an 20% chance to contract the disease.

Once you have β you can solve the SI Model. Usually, the way it's done is assuming that at the first time step you have a set of one or more patient zeros scattered randomly into society. Then, you track the ratio of people in the I status as time goes on, which is $|I|/(|I| + |S|)$. This usually generates a plot like the one in Figure 20.2.

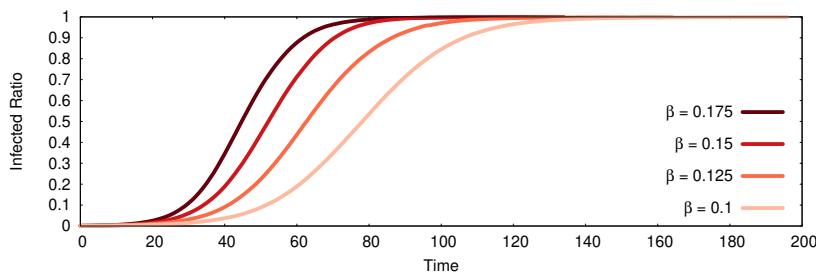


Figure 20.2: The solution of the SI Model for different β values. The plot reports on the y axis the share of infected individuals ($i = |I|/(|I| + |S|)$) at a given time step (x axis).

SI models have the same signature. At first, the ratio of infected individuals grows slowly, because there are few people in the I state. Then, as soon as I expands a little, we see an exponential growth, as more and more people have a chance to meet an infected individual. After a critical point, the growth of I slows down, because there aren't many people left in S to infect.

Eventually, all SI models stop when every single individual is in the set I and so no one else can transition. All SI Models, no matter the value of β will end up with a complete infection, where S is empty and I contains the entirety of society. The only thing β affects

– as you can see from Figure 20.2 – is the speed of the system: when the exponential growth of I starts to kick in and when S gets emptied out.

We can re-tell the story I've just exposed in mathematical form. In our SI model, the probability that an infected individual meets a susceptible one is simply the number of susceptible individuals over the total population, because of the homogenous mixing hypothesis: $|S|/|V|$ (remember $|V|$ is our number of nodes). There are $|I|$ infected individuals, each with \bar{k} meetings (the average degree). Thus the total number of meetings is $\bar{k} \frac{|I||S|}{|V|}$. Since each meeting has a probability β of passing the disease, at each time step there are $\beta \bar{k} \frac{|I||S|}{|V|}$ new infected people in I .

We can simplify the equation a bit, because $|I|/|V|$ and $|S|/|V|$ are related. They sum to one, since S and I are the only possible states in which you can have a node. So, if we say $i = |I|/|V|$, that is, the fraction of nodes in I , then $|S|/|V| = 1 - i$. So our formula becomes: $i_{t+1} = \beta \bar{k} i_t (1 - i_t)$ ⁷, where t is the current time step. If we integrate over time, we can derive the fraction of infected nodes depending solely on the time step⁸:

$$i = \frac{i_0 e^{\beta \bar{k} t}}{1 - i_0 + i_0 e^{\beta \bar{k} t}}.$$

This is the mathematical solution to the SI model with homogeneous mixing, generating the plot in Figure 20.2. You can see why you have an initial exponential growth at the beginning and a flat growth at the end. If $i_0 \sim 0$, then the denominator is 1 and the numerator is dominated by the $e^{\beta \bar{k} t}$ factor: exponential growth (very slow at the beginning because multiplied with the small i_0). When $i_0 \sim 1$, both the denominator and the numerator reduce to $e^{\beta \bar{k} t}$, which means that, in the end, $i \sim i_0 \sim 1$, so there's no growth.

Why did we go to the trouble of all this math? Because, at this point, we have to tear down the homogenous mixing hypothesis. The formulas will allow to see the difference better.

Homogenous mixing is based on the assumption that the more people are infected, the more likely you're going to be infected. In practice, it assumes everybody is the same. In homogenous mixing, the global social network is a lattice: a regular grid where each node is connected only to its immediate neighbors. Figure 20.3 shows an example of square lattice (Section 6.4): each node connects regularly to four spatial neighbors. On a lattice, the infection spreads like water filling a surface⁹.

We know that real networks are not neat regular lattices. The degree is distributed unevenly, with hubs having thousands of con-

⁷ Note that this is a differential equation, so you need to integrate it to actually find the share of infected nodes at $t + 1$. Also, I'm only including the addition to i_{t+1} , not its full composition. So, pedantically, the correct formula should be $i_{t+1} = i_t + \beta \bar{k} i_t (1 - i_t)$, but that would make the discussion harder to follow – and it would not change the results we are interested in here. This warning applies to all formulas with the time subscript.

⁸ Albert-László Barabási et al. *Network science*. Cambridge university press, 2016

⁹ This is a useful mental image: https://upload.wikimedia.org/wikipedia/commons/a/a6/SIR_model_simulated_using_python.gif.

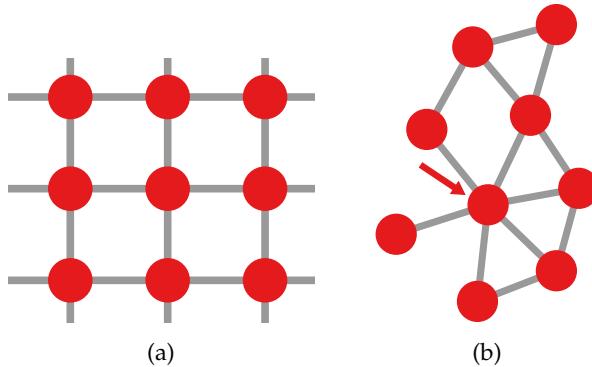


Figure 20.3: The underlying assumption of an SI model: that social networks look like a uniform lattice (a). Instead, the degree is distributed differently (b), with hubs – pointed by the red arrow – having many more connections and, thus, infection chances.

nections – see Section 9.3. When the infection hits such a hub, it will accelerate faster through the network. In fact, it is extremely easy to infect an hub early on. Hubs have more connections, thus they are more likely to be connected to one of your patient zeros. Those same connections make them super-spreaders: once infected, the hub will allow the disease to reach the rest of the network quickly. In fact, when searching information in a peer-to-peer network, your best guess is always to ask your neighbor with highest degree¹⁰.

To treat the SI model mathematically you have to first group nodes by their degree. Rather than solving for i – the fraction of infected nodes –, you solve for i_k : the fraction of infected nodes of degree k . The formula for a network-aware SI model is similar as the one we saw for the vanilla SI model:

$$i_{k,t+1} = \beta k f_k (1 - i_{k,t}).$$

The two differences are that: (i) we replace the average degree \bar{k} with the actual node's degree k , and (ii) rather than using $i_{k,t}$ we use f_k – a function of the degree k . This is because real world networks typically have degree correlations: if you have a degree k the degree of your neighbors is usually not random (see Section 31.1 for more). If it were random, then we could simply use $i_{k,t}$, because the number of infected individuals around you should be proportional to the current infection rate. But it isn't: in presence of degree correlations, if you have k neighbors then there exists a function f_k able to predict how many neighbors they have. Thus the likelihood of a node of degree k of having infected neighbors is specific to its degree, and not (only) dependent on $i_{k,t}$.

If you do the proper derivations¹¹, you'll discover that in a $G_{n,p}$ network the dynamics have the same functional form to the ones of the homogeneous mixing, as Figure 20.4 shows. In $G_{n,p}$ the exponential rises faster at the beginning – due to the few outliers with high degree – and tails off slower at the end – due to the outliers with low degree – but the rising and falling of the infection rates is still an

¹⁰ Lada A Adamic, Rajan M Lukose, Amit R Puniyani, and Bernardo A Huberman. Search in power-law networks. *Physical review E*, 64(4):046135, 2001

¹¹ Romualdo Pastor-Satorras and Alessandro Vespignani. Epidemic dynamics and endemic states in complex networks. *Physical Review E*, 63(6):066117, 2001a

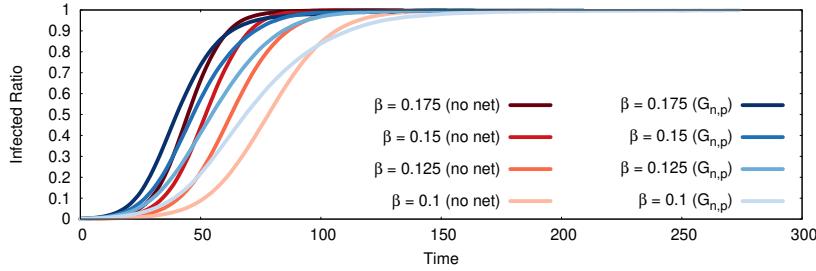


Figure 20.4: The solution of the SI Model for different β values in homogeneous mixing (reds) and $G_{n,p}$ graphs (blues). The plot reports on the y axis the share of infected individuals ($i = |I|/(|I| + |S|)$) at a given time step (x axis).

exponential. It also depends, obviously, on the average degree you give to the lattice and to the $G_{n,p}$ graph.

Is that it? Did I really throw Greek letters at you for such an underwhelming discovery? Of course not. Remember that $G_{n,p}$ is a poor approximation of social networks, *especially* when it comes to degree distributions. Let's look at what happens when you have a network with a power law degree distribution.

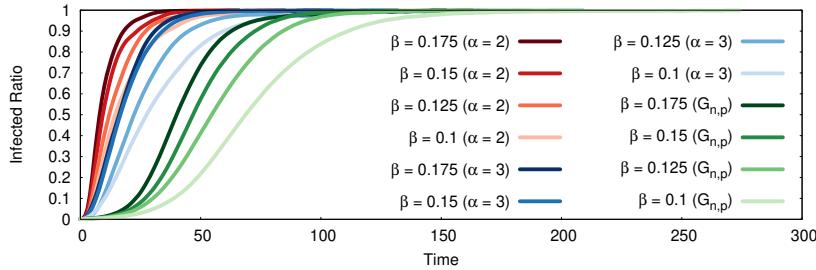


Figure 20.5 shows you the results of a bunch of simulations on networks with different degree distributions. The slowest infections (in green) happen for $G_{n,p}$ graphs. When looking at a power law random network, the thing that matters the most is the exponent of the degree distribution, α (for a refresher on its meaning, see Section 9.3).

If $\alpha = 3$ we have not-so-large hubs. These hubs contribute enormously to the speed of infection: it is easy to catch them and, once you do, the disease spreads faster. You can see how the exponential growth regimes, for the blue data series, are much steeper than in the green $G_{n,p}$ cases. Even for $\beta = 0.1$, a mildly contagious disease, a power law degree distribution with $\alpha = 3$ gets infected faster than a $G_{n,p}$ network with a much more aggressive disease (with $\beta = 0.175$, almost twice as infectious!).

The same comparison applies when pitting the $\alpha = 3$ case with $\alpha = 2$ networks. In the latter case, there's not even a recognizable

Figure 20.5: The solution of the SI Model for different β values for networks with power law degree distribution with exponent $\alpha = 2$ (reds), $\alpha = 3$ (blues) and $G_{n,p}$ graphs (greens). The plot reports on the y axis the share of infected individuals ($i = |I|/(|I| + |S|)$) at a given time step (x axis).

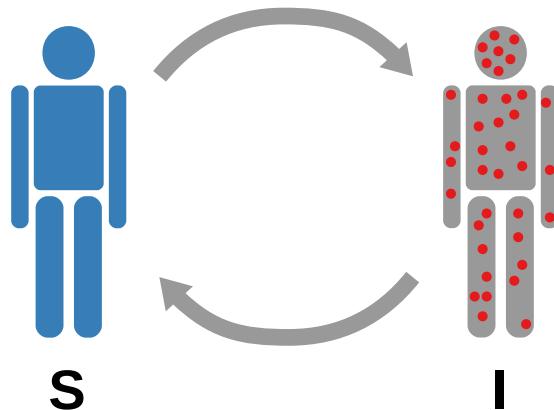
exponential warm up any more (in red in Figure 20.5). You know that, no matter where you started, you’re going to hit the largest hub of the network at the second time step $t = 2$, because it is connected to practically every node. And, since it is connected to practically every node, at $t = 3$ you’ll have almost the entire network infected.

In fact, I ran the simulations from Figure 20.5 on imperfect and finite power law models. Theoretically, if you had a perfect infinite power law network, infection would be *instantaneous for any non-zero value* of β . Meaning that, no matter how infectious a disease is, with $\alpha = 2$ it will infect the entire network almost immediately. And things get even more complicated when you add to the mix the fact that networks evolve over time¹². Scary thought, isn’t it?

20.2 SIS

Just like in the SI model, also in the SIS model nodes can only either be Susceptible or Infected¹³. However, the SIS model adds a transition. Where in SI you could only get infected without possibility of recovery ($S \rightarrow I$), in SIS you can heal ($I \rightarrow S$).

Thus the SIS model requires a new parameter. The first one, shared with SI, is β : the probability that you will contract the disease after meeting an infected individual. Once you’re infected, you also have a recovery rate: μ . μ is the probability that you will transition from I to S at each time step. High values of μ mean that recovering from the disease is fast and easy. Note that recovery puts you back to the Susceptible state, thus you can catch the disease again in the future.



¹² Eugenio Valdano, Luca Ferreri, Chiara Poletto, and Vittoria Colizza. Analytical computation of the epidemic threshold on temporal networks. *Physical Review X*, 5(2):021005, 2015

¹³ Herbert W Hethcote. Three basic epidemiological models. In *Applied mathematical ecology*, pages 119–144. Springer, 1989

Figure 20.6: The schema underlying the SIS Model: two possible states and two possible transitions.

Figure 20.6 shows the schema fully defining the model. In practice, SIS models disease with recovery and relapse. An example would be the general umbrella of the flu family. Once you heal from a particular strain of the flu you’re unlikely to fall ill again under the

same strain. However, you can easily catch a similar strain, thus cycling each year between the S and I states.

The presence of μ changes the outcome of the model. SI models always reach full saturation: eventually, every node will end up in status I . For SIS models that is not true, because a certain fraction of nodes – μ – heal at each time step. The interplay between the recovery rate μ , the infection rate β , and the average degree \bar{k} determines the asymptotic size of I : the share of infected nodes as time approaches infinity ($t \rightarrow \infty$). To see how, let's look at the math again.

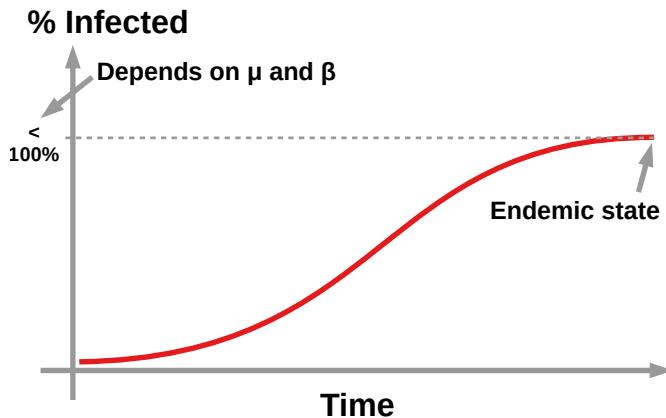


Figure 20.7: The typical evolution of an endemic SIS model: the equilibrium state is the one in which a constant fraction $i < 1$ contracted the disease. The rate at which infected people recover and the infection rate are perfectly balanced, as all things should be.

The SI model could be described by the formula $i_{t+1} = \beta k i_t (1 - i_t)$. If, at each time step, a fraction μ of the infected nodes i recovers, we just have to remove it from i . Thus, the SIS model is simply $i_{t+1} = \beta k i_t (1 - i_t) - \mu i_t$. This should raise your eyebrow. As i_t grows, so does μi_t , obviously. Eventually, $\beta k i_t (1 - i_t) = \mu i_t$: that is when the share of infected nodes i doesn't grow any more. We reached the endemic state where the number of people recovering is perfectly balanced by the new infected. Figure 20.7 depicts this situation.

Is it possible that $\beta k i_t (1 - i_t) < \mu i_t$? Meaning: is there a situation when people are recovering faster than new infected pop up? Yes! We can get rid of a disease in the SIS model. If you do the proper derivations¹⁴, you discover that the magic value of μ for that to happen is $\beta \bar{k}$. If $\mu < \beta \bar{k}$, recovery isn't fast enough to escape the endemic state, and you're in the situation we saw in Figure 20.7. But, if $\mu \geq \beta \bar{k}$, eventually the endemic state is the one for which $i = 0$! Congratulations! No more infected people. You defeated the disease. The evolution of the outbreak looks like what I sketch in Figure 20.8.

In my simulations for Figure 20.8 I set $\bar{k} = 1$, to make the comparisons easier. You can see that, when $\mu \geq \beta$, eventually the disease dies out. The case for which $\mu < \beta$ reaches the endemic state, showing that the disease will persist in the population. For $\mu = \beta$, the disease dies out, although not as quickly as for $\mu > \beta$.

¹⁴ Romualdo Pastor-Satorras and Alessandro Vespignani. Epidemic spreading in scale-free networks. *Physical review letters*, 86(14):3200, 2001b

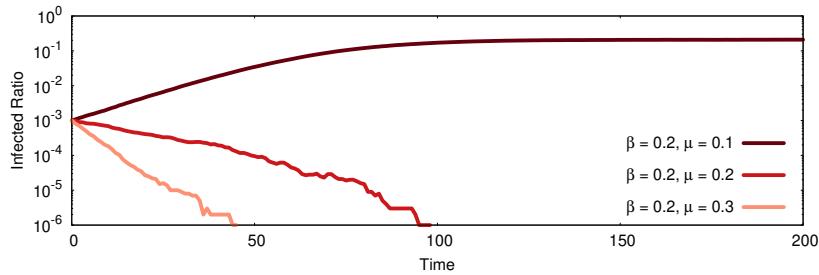


Figure 20.8: The solution of the SIS Model, keeping β fixed but varying μ . The plot reports on the y axis the share of infected individuals ($i = |I|/(|I| + |S|)$) at a given time step (x axis).

The relationship between μ and β is so important that we can study the evolution of infections in a network according to their ratio $\lambda = \beta/\mu$. Since we ignore the degree when calculating λ , we know that λ depends exclusively by the pathogen's characteristics. We just saw in Figure 20.8 that, in some cases, the SIS model predicts a non-zero endemic state – there are always at least $i > 0$ infected individuals – and, in other cases, the disease dies out – thus $i = 0$. So there must be a critical value of λ that make us transition between the endemic and non-endemic state.

In $G_{n,p}$ networks with homogeneous mixing this critical λ value depends on the average degree of the network \bar{k} . Specifically, if $\lambda > 1/(\bar{k} + 1)$ then the disease will be endemic. If λ is below that threshold, then the pathogen will eventually disappear. Note that, in a $G_{n,p}$ graph, \bar{k} is always positive and equal to $p|V|$ (see Section 16.2 for a refresher). This mean that you can find a value of λ below the critical endemic threshold: the only way for $1/(\bar{k} + 1)$ to be equal to zero would be if $\bar{k} = \infty$, which is clearly nonsense. The average degree in a $G_{n,p}$ graph cannot be infinite. Thus any $G_{n,p}$ graph will be resistant to a disease with a λ lower than $1/(\bar{k} + 1)$.

Surprising absolutely no one, when we drop the homogeneous mixing assumption and we look at a preferential attachment network the situation changes radically. Here, the critical value is \bar{k}/\bar{k}^2 : the average degree over the average squared degree – note that we square the degrees and *then* we take the average, we don't simply raise the average degree to the power of 2. Here's the problem: the average degree in a preferential attachment network is low. But the network contains large hubs with a ridiculously high degree: squaring it eclipses the small contributions from the peripheral nodes that kept \bar{k} . In other words, as you add more and more nodes to the network, \bar{k} remains constant and low – because you're adding peripheral nodes with low degree – but \bar{k}^2 grows fast. Each of those new nodes tend to add to the degree of the largest hubs, because of preferential attachment – shooting \bar{k}^2 in the stratosphere.

The consequence? Well, if \bar{k} stays constant – or even decreases –

and \bar{k}^2 grows relatively to it, the critical threshold \bar{k}/\bar{k}^2 tends to zero. If we say that you have an endemic value if $\lambda > \bar{k}/\bar{k}^2$ and $\bar{k}/\bar{k}^2 = 0$, then any disease, no matter β and μ , will be endemic in a network with a power law degree distribution. Oops.

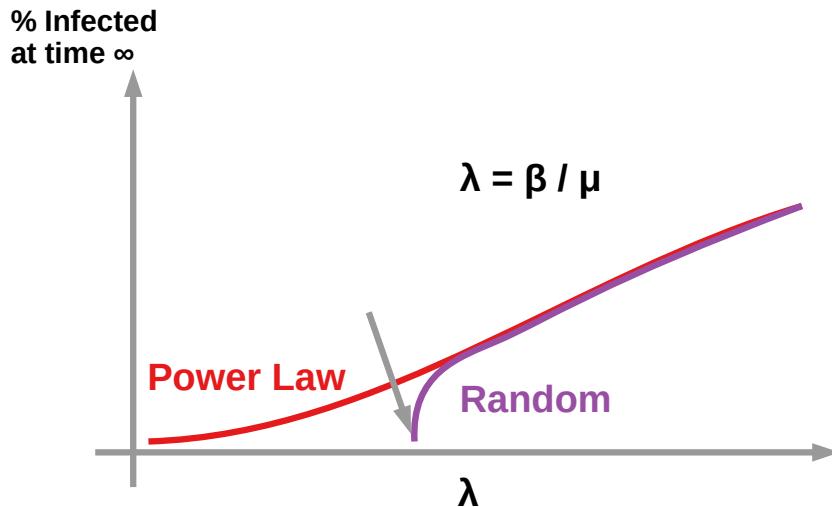


Figure 20.9: The solution of the SIS Model for λ . As λ grows (x axis), I show the share of infected individuals i at the endemic state ($t \rightarrow \infty$).

I sum up the situation in Figure 20.9: in a power law random network, no matter λ , the pathogen can always be endemic, even if it's not very infectious. In a $G_{n,p}$ network you see that for some values of λ greater than zero you do not have endemic infections, thus the network's topology has a big effect on the dynamics of the epidemic. Heavy tailed degree distributions, which are ubiquitous in reality, are closer to the power law line than to the $G_{n,p}$ line, meaning that we should expect to see a similar behavior in real networks.

20.3 SIR

The next step in modeling epidemics on networks is by considering those diseases you can catch only once in your lifetime. Think about the bubonic plague. If you have the bad luck of encountering the *Yersinia pestis*, there are only two possible outcomes. Either you die, or you survive. If you survive, your immune system is now trained to recognize the bacterium and will not allow you to be infected again. In either case, you are Removed from the outbreak.

Removed is exactly the state we add to the SI model in the SIR model. Now the only two possible state transitions are $S \rightarrow I$ – when you contract the disease – and $I \rightarrow R$ – when you heal or die. Figure 20.10 shows the schema fully defining the model.

The defining characteristic of a SIR model is its lack of endemicity. Either the disease kills everybody, or every individual still alive has

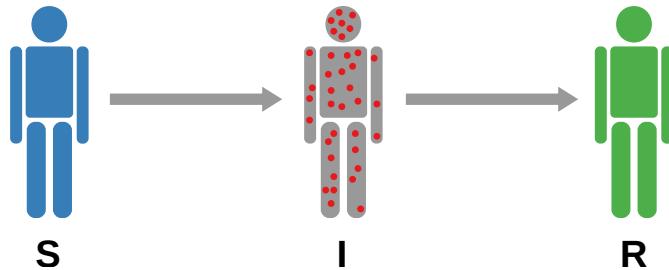


Figure 20.10: The schema underlying the SIR Model: three possible states and two possible transitions.

had the disease and healed. Figure 20.11 shows such a typical evolution. At the beginning, everybody is susceptible. Then, people start getting infected, so I grows. R cannot start growing immediately, as I is still too small for the recovery parameter μ to significantly contribute to R size. As I grows, though, there are enough infected individuals that start being removed. Eventually every I individual transitions to R .

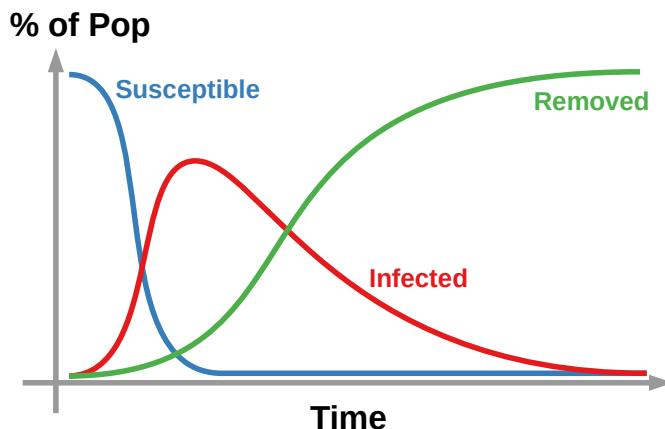


Figure 20.11: The typical evolution of an SIR model: after an initial exponential growth of the infected, the removed ratio takes over until it occupies the entire network.

Note that the evolution of Figure 20.11, where eventually there are only people in the R state, isn't necessarily the only possible. If you're lucky, I empties out before S , meaning that the disease dies out in R before every susceptible individual has had the privilege of sneezing.

Mathematically speaking, the evolution of the recovery ratio $r = |R|/|V|$ is the simplest possible. At each time step, a fraction μ of I transitions into R . In SIR, just like in SIS, μ is the recovery parameter. So $r_{t+1} = \mu r_t$. The evolution of i is a bit trickier, but it boils down to making sure of removing the nodes in $|R|$ from the potential pool of infected.

Of course, in the quest of making models more and more accurate to fit the actual dynamics of infections, you don't have to stop with the SIR model. In the literature you can find: SEIR, adding an "Exposed" status before the infection triggers in an individual^{15,16}; you

¹⁵ Michael Y Li and James S Muldowney. Global stability for the seir model in epidemiology. *Mathematical biosciences*, 125(2):155–164, 1995

¹⁶ Michael Y Li, John R Graef, Liancheng Wang, and János Karsai. Global dynamics of a seir model with varying total population size. *Mathematical biosciences*, 160(2):191–213, 1999

can have an immune status M; relapsing to susceptibility in a SIRS model after being removed¹⁷; and, of course, combining everything together in a warm and fuzzy pile of states, in the MSEIRS model (I wish I was kidding). As you might expect, the math becomes fiendishly complicated and it's just not worth delving into that for an introductory chapter to network epidemics such as this one.

This chapter is also by necessity just a superficial sketch of network epidemics. There's plenty more research on endemic and epidemic states and their relationship with network topology^{18,19,20} that you can check if you find the topic fascinating.

20.4 Summary

1. In simple contagion epidemics models, nodes are in specific states given their exposure to the disease and can transition in different states according to simple contact rules.
2. In SI models there are two states: Susceptible and Infected. Nodes transition from S to I with a certain probability β if they have at least an I neighbor.
3. All SI models end up with the entire network in the I state. β determines how quickly this happens. Networks with a degree distribution characterized by a low α exponent are infected more quickly.
4. SIS models are like SI models, but nodes can transition back to S state with a stochastic probability μ at each time step.
5. The $\lambda = \beta/\mu$ ratio determines whether the disease will be endemic or it will die out. In power law random networks, no matter λ , the disease will always be endemic.
6. In SIR models, nodes in state I recover at a μ rate rather than moving back to S . Eventually, all nodes will move to the R state, and no disease can be endemic.

20.5 Exercises

1. Implement an SI model on the network at <http://www.networkatlas.eu/exercises/20/1/data.txt>. Run it 10 times with different β values: 0.05, 0.1, and 0.2. For each run (in this and all following questions) pick a random node and place it in the Infected state. What's the average time step in which each of those β infects 80% of the network?

¹⁷ Chun-Hsien Li, Chiung-Chiou Tsai, and Suh-Yuh Yang. Analysis of epidemic spreading of an sirs model in complex heterogeneous networks. *Communications in Nonlinear Science and Numerical Simulation*, 19(4):1042–1054, 2014

¹⁸ Yang Wang, Deepayan Chakrabarti, Chenxi Wang, and Christos Faloutsos. Epidemic spreading in real networks: An eigenvalue viewpoint. In *22nd International Symposium on Reliable Distributed Systems, 2003. Proceedings.*, pages 25–34. IEEE, 2003

¹⁹ Rick Durrett. Some features of the spread of epidemics and information on a random graph. *Proceedings of the National Academy of Sciences*, 2010

²⁰ Claudio Castellano and Romualdo Pastor-Satorras. Thresholds for epidemic spreading in networks. *Physical review letters*, 105(21):218701, 2010

2. Run the same SI model on the network at <http://www.networkatlas.eu/exercises/20/2/data.txt> as well. One of the two networks is a $G_{n,p}$ graph while the other has a power law degree distribution. Can you tell which is which by how much the disease takes to infect 80% of the network for the same starting conditions used in the previous question?
3. Extend your SI model to an SIS. With $\beta = 0.2$, run the model with μ values of 0.05, 0.1, and 0.2 on both networks used in the previous questions. Run the SIS model, with a random node as a starting Infected set, for 100 steps and plot the share of nodes in the Infected state. For which of these values and networks do you have an endemic state? How big is the set of nodes in state I compared to the number of nodes in the network? (Note, randomness might affect your results. Run the experiment multiple times)
4. Extend your SI model to an SIR. With $\beta = 0.2$, run the model for 400 steps with μ values of 0.01, 0.02, and 0.04 and plot the share of nodes in the Removed state for both the networks used in Q1 and Q2. How quickly does it converge to a full R state network?

21

Complex Contagion

You may or may not have noticed that, in the previous chapter, all our models of epidemic contagion shared an assumption. Every time a susceptible individual comes in contact with an infected individual, they have a chance to become infected as well. If that doesn't happen, the healthy person is still in the susceptible pool. The next time step represents a new occasion for them to contract the disease. And so on, *ad infinitum*.

Without that assumption, the models wouldn't be mathematically tractable. For instance, if each node gets only one chance to be infected, you can easily see how it is not given that a SI model would eventually infect the entire network. In fact, it takes any $\beta < 1$ to make that impossible. The first time you fail to infect somebody you won't get the chance to try again.

SI, SIS, and SIR models are useful and generated tons of great insights. But this limitation allows them to model only rather specific types of outbreak. We usually consider them models of simple contagion. There are fundamentally two ways to make such models more complex and realistic. They involve changing two things: (i) the triggering mechanism, which is the condition regulating the $S \rightarrow I$ transition, and (ii) the assumption that each individual gets infinite chances to infect their neighbors.

We deal with the triggering mechanisms in Section 21.1 and infection chances in Section 21.2. We also explore the possibilities of interfering with the outbreak in Section 21.3, dedicated to epidemic interventions.

21.1 Triggers

As I mentioned before, in the simple contagion we explored in Chapter 20, one contact with an infected node is enough for you to have a chance to be infected. The main difference between simple and complex contagion is that, in the latter, you require reinforcement. You

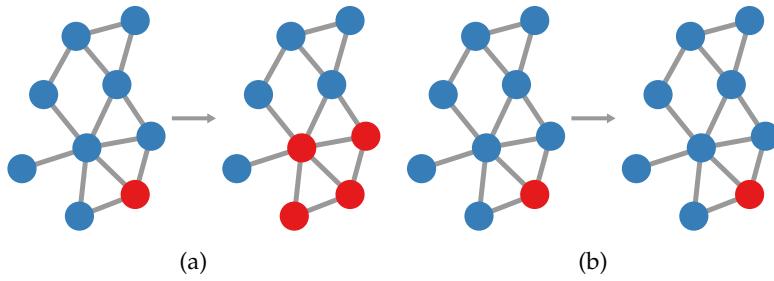


Figure 21.1: A simple introduction to complex contagion. (a) A simple contagion where any contact can and will transmit the disease. (b) A complex contagion where you need two contacts to contract the disease.

can consider Figure 21.1 as the simplest possible introduction to this concept. If we require a node to enter into contact with two infected individuals rather than one, the figure shows that the outbreak from a single seed is impossible.

In reality, complex contagion is more nuanced than this. A single contact may or may not infect you, but if you have multiple contacts your likelihood to transition into the I state grows. There are fundamentally two types of reinforcement we can consider, which are subtypes of complex contagion: Cascade and Threshold. Before looking at them, though, let's consider an easy extension of simple contagion since, in a sense, it can be turned into the simplest possible reinforcement mechanism.

Classical

In classical reinforcement you have an independent probability of being infected for each of your neighbors that are infected. Note that this is different from the simple contagion of Chapter 20: in there, you get β chance to transition regardless whether you have one or more infected neighbors. Here, more infected neighbors mean more chances of infection.

If you have n sick friends, and you visit them one by one, at each visit you toss a coin. To calculate the probability you are going to be infected, it is easier to calculate the probability of not being infected by any contact, and then invert it. If our parameter β tells us the probability of being infected by a single contact, then $(1 - \beta)$ is the probability of not being infected. Since the coin tosses are all independent, the probability of never being infected by any of the n contacts is $(1 - \beta)^n$. So the probability that at least one contact will infect us is $1 - ((1 - \beta)^n)$.

Figure 21.2 shows a vignette of this process. The healthy individual has four neighbors, all of which are infected. Thus she has to make four independent coin tosses, each of which has β chance to succeed. Thus, the more infected neighbors the more likely she will contract the disease. The difference with a simple SI model without

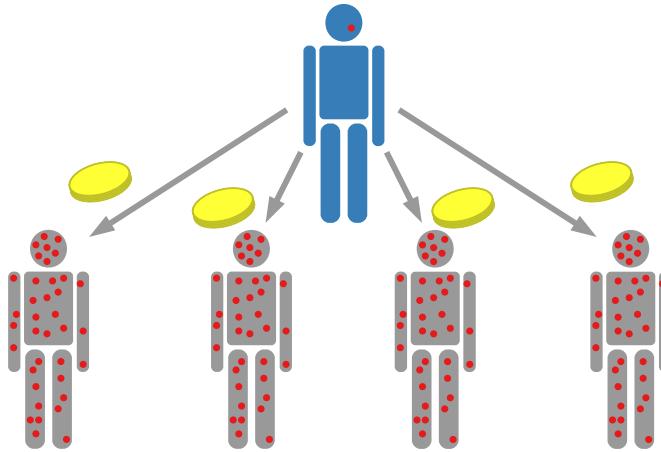


Figure 21.2: An example of classical contagion. The healthy individual performs an independent check for contagion – a coin toss – with each of their neighbors.

reinforcement is that in the simple SI model you always toss a single coin at each time step, no matter how many infected neighbors you have – as long as you have at least one. So, at each time step, in simple SI the infection probability is β if you have 1 or n infected neighbors. In classical complex SI, you have $1 - ((1 - \beta)^n)$ probability of being infected. The whole difference between the two models is that the latter depends on n , the number of your friends that are infected.

The vignette makes clear why, in the classical model, it's easy to infect hubs: they have more neighbors. More neighbors mean that they toss their coins much more often. This is what generates the super-exponential – theoretically instantaneous – outbreak growth in power law models with large hubs.

Threshold

The threshold model is a sophisticated version of the introductory example I made with Figure 21.1. To be infected, you need multiple infected neighbors. The threshold model adds a parameter, let's call it κ . If more than κ of your neighbors are infected, then they pass the infection to you¹.

For instance, if $\kappa = 4$, you need four infected friends to have a chance to be infected. Note that you can still inject in this model the β chance of infection, by saying that, once you clear the κ threshold, you have a chance $\beta < 1$ to contract the disease. In this latter case, if $\kappa = 1$, this model is the same as the simple SI without reinforcement: as long as you have at least one infected neighbor, you toss your β coin.

Figure 21.3 shows a vignette of this process. If we were in classical contagion, the hub at the top would toss three coins with β chance of

¹ Mark Granovetter. Threshold models of collective behavior. *American journal of sociology*, 83(6):1420–1443, 1978

getting infected at each check. In threshold contagion, since $\kappa = 4$, the probability of her being infected is zero. Threshold models usually find an easy time to infect hubs, because we usually set κ to be low. Any hub will have more infected friends than that. Any $\kappa > 1$ renders peripheral nodes safe, since most of them have only one connection.

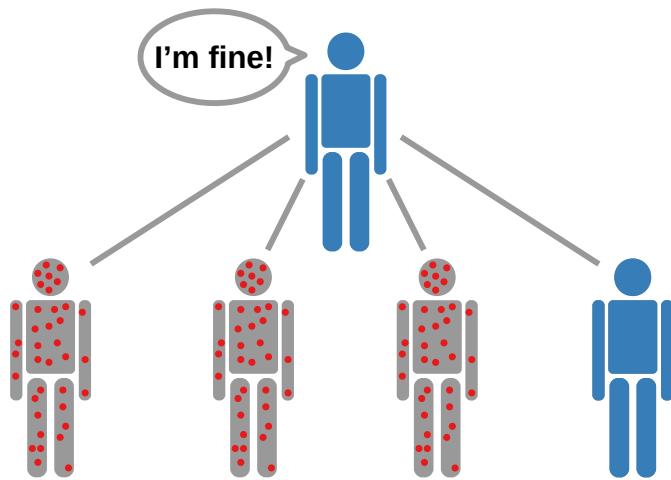


Figure 21.3: An example of threshold contagion. The healthy individual checks how many infected neighbors she has. If they're less than κ , she's fine. In this example, $\kappa = 4$.

The threshold model is where epidemiology starts to blend in with sociology. Rather than modeling the spread of a virus, the threshold assumption works best when explaining the spread of a behavior. The assumption is that individuals' behavior depends on the number of other individuals already engaging in that behavior². This can be used to explain racial segregation³ and customer demand⁴. We're going to dive in deep on the racial segregation angle when we'll deal with homophily in social networks in Chapter 30. The customer demand angle explains why variations of the threshold models are one of the favorite instruments of researchers involved in studying viral marketing. We'll see more of that later on in this chapter.

You can spice up the threshold model by allowing κ to be a node-dependent parameter, rather than a global one. This means that each node v has a different κ_v activation threshold. Some might be convinced to change their behavior by a single individual contact. Or, to keep our epidemic metaphor, they might have a weak immune system, prone to concede defeat to the disease after the first exposure. Others require a high κ_v : their defining characteristic is being stubborn, whether in their head or in their antibodies. When it comes to social behavior, individuals' thresholds may be influenced by many factors: social economic status, education, age, personality, etc.

² Mark Granovetter and Roland Soong. Threshold models of diffusion and collective behavior. *Journal of Mathematical sociology*, 9(3):165–179, 1983

³ Mark Granovetter and Roland Soong. Threshold models of diversity: Chinese restaurants, residential segregation, and the spiral of silence. *Sociological methodology*, pages 69–104, 1988

⁴ Mark Granovetter and Roland Soong. Threshold models of interpersonal effects in consumer demand. *Journal of Economic Behavior & Organization*, 7(1):83–99, 1986

Cascade

The cascade model is a straightforward variant of the threshold model, in fact they were developed together by the same researchers. However, their differences are important enough to mention them separately. In the cascade model you also need reinforcement from more than one neighbor to transition to the I state. However, while in the threshold model this was governed by an *absolute* parameter κ , here we use a *relative* one.

In other words, in the cascade model you need a fraction of your neighbors to be infected in order for you to be infected⁵. In the cascade model, the size of your neighborhood influences your likelihood to transition. In the threshold model it didn't: whether you have one or one hundred neighbors, you always need the same κ number of them in the I state to consider transitioning.

So if you have four friends, but you need a fraction $\beta > .75$ to transition, if only one, two, or three of them are infected you're fine. In such a condition, you need all of your neighbors to be infected in order to get sick. Only when the fourth neighbor is infected you'll be triggered. Here, I use the same notation β I had in the classical contagion, but note that its meaning is slightly different. β is not the probability of infection given a contact, but the share of neighbors in set I required for you to transition to I . If in classical contagion a low β means low infection chance, in cascade a low β means that it's more likely to get infected, as you need fewer neighbors to make you transition.

Why do I separate the cascade and the threshold model? Because of hubs. We saw that, in the threshold model, infecting hubs was easy. Since they have lots of connections, the likelihood of them having at least κ infected friends is high. In the cascade model the opposite holds. It's harder to infect hubs in a cascade than in a threshold model, because – for a hub – the β fraction of neighbors required to be infected usually includes hundreds or thousands of nodes. So, in a threshold model, hubs are the primary spreaders of the disease. In a cascade model, they're the last bastion of defense. Once the hubs fall, there's no more chance for salvation.

Again, you're allowed to vary β to account for the heterogeneity of gullibility. With proper, rather complicated, fine tuning of κ_v in the threshold model and β_v in the cascade model, you can render them equivalent. However, it's useful to know that there is a simple way to model contagion in the two cases where you want to simulate a high or low resistance of hubs to the new spreading behavior.

Separating the cascade and threshold models in different compartments would make you think they obey completely different

⁵ Duncan J Watts. A simple model of global cascades on random networks. *Proceedings of the National Academy of Sciences*, 99(9):5766–5771, 2002

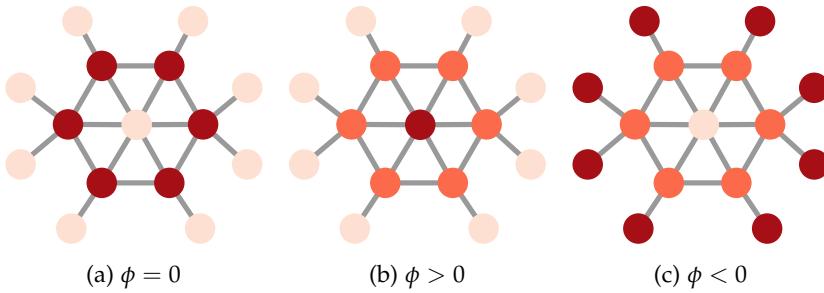


Figure 21.4: The three classes of complex contagion as regulated by the ϕ parameter. The node color represent the activation time of the node from dark (early) to bright (late).

rules and they are just different phenomena. This needs not to be the case. The separation is mostly done out of a pedagogical need. In fact, there is a universal model of spreading dynamics concentrating on hubs⁶. We don't need to delve deep into the details on how this model works, but it mostly hinges on a parameter: ϕ . ϕ determines the interplay between the degree of a node and its propensity of being part of the epidemics. If $\phi = 0$, the likelihood of contagion of the node is independent with its degree. If $\phi > 0$ we are in the threshold scenario: hubs have a stronger impact on the network. With $\phi < 0$, as you might expect, the opposite holds. Figure 21.4 shows a vignette of the model.

Sprinkling a bit of economics into the mix, you can relate the threshold or the cascade parameter with the utility an actor v gets from playing along or not. Each individual calculates their cost and benefit from undertaking or not undertaking an action. There is a cost in adopting a behavior before it gets popular, and in not doing so after it did⁷. Being aware of these effects makes for very effective strategies to make your own decisions while you're in doubt. You can establish a Schelling point which determines whether or not you're going to undertake an action⁸, which effectively means you consciously set your own κ_v . However, this is getting dangerously close to a weird blend of economics, philosophy, and game theory. If you're interested in learn more, you'd be best served by closing this book and looking elsewhere⁹.

21.2 Limited Infection Chances

So far we have mainly looked at diseases spreading through a network of contacts as a bad thing that we want to minimize. If you want to look at the opposite problem – how to spread things faster and faster through a social network – without looking like a sociopath, you need to slightly change the perspective. You can concoct a scenario in which, for instance, you want to sell a product, thus

⁶ Baruch Barzel and Albert-László Barabási. Universality in network dynamics. *Nature physics*, 9(10):673, 2013b

⁷ Thomas C Schelling. Hockey helmets, concealed weapons, and daylight saving: A study of binary choices with externalities. *Journal of Conflict resolution*, 17(3):381–428, 1973

⁸ <https://www.lesswrong.com/posts/Kbm6QnJv9dgWsPHQP/schelling-fences-on-slippery-slopes>

⁹ Herbert Gintis. *The bounds of reason: Game theory and the unification of the behavioral sciences*. Princeton University Press, 2014

you want people to talk about it and convince each other. In practice, you want them to *infect* themselves with the *idea* that the product is good^{10,11}.

The obvious strategy would be to target hubs, since they have more connections. However, this heavily depends on your triggering model, and hubs come with a disadvantage. First, by being prominent, hubs are targeted by many things, thus they have a very high barrier to attention. Second, they have many connections: if the triggering mechanism requires reinforcement, most of their connections might not get it, thinning out the intervention. A third and final problem might be that you have only one shot at convincing a person. If you fail, it's game over forever. If a hub fails, you might not have a second shot to get to all their peripheral nodes.

¹⁰ Pedro Domingos and Matt Richardson. Mining the network value of customers. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 57–66. ACM, 2001

¹¹ Dashun Wang, Zhen Wen, Hanghang Tong, Ching-Yung Lin, Chaoming Song, and Albert-László Barabási. Information spreading in context. In *Proceedings of the 20th international conference on World wide web*, pages 735–744. ACM, 2011b

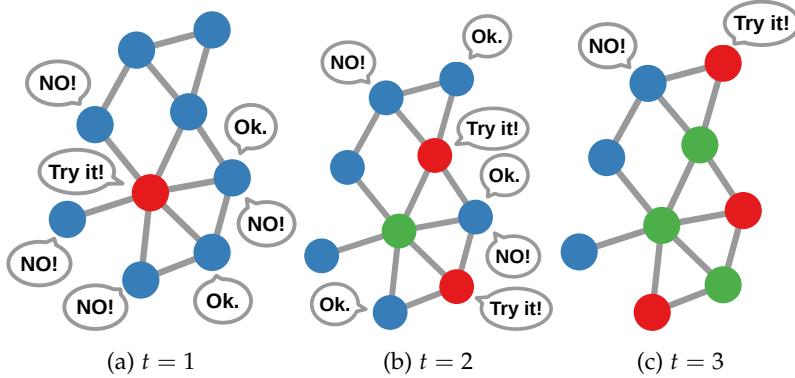


Figure 21.5: An example of independent cascade. The node's state is encoded by the color: blue = susceptible, red = infected and contagious, green = infected but not contagious.

You can think of this model as a modified SI or SIR, as I show in Figure 21.5. Suppose that an infected neighbor makes you transition from S to I at time t . At the next time step $t + 1$ you will attempt to do the same to your S neighbors. However, at $t + 2$ you transition to a non-contagious stage, where you won't attempt to convert your neighbors any more. You will be in I forever, this is the reason why this is an SI model, but you won't propagate the disease. Or, you could see yourself as transitioning to R , with $\mu = 1$: every i individual will always transition to R immediately. The difference is that R individuals are still infected, they just cannot pass the disease.

Note how the node at the bottom in Figure 21.5(a) resisted the hub at time $t = 1$, but in Figure 21.5(b) gave up on the second attempt by another node at time $t = 2$. At the same time, the rightmost node has to give two answers because of two independent attempts from its two neighbors. At time $t = 3$ (Figure 21.5(c)) we see only one persuasion attempt, given that the other infectious nodes are connected to already infected ones. The cascade ends with unconvincing nodes, due to the lack of any possible further move in $t = 4$.

It should be clear by now that, once you tried the first time to

convince me to buy a product, any further attempts won't work if you didn't convince me immediately. Maybe another person will persuade me, just not you. This is a crucial difference with regard to SI models. We know that any SI model will eventually fill the entire network. The independent cascade model¹² won't: the nodes we choose to start the infection with are very important to maximize the reach of our message.

In the simplest model, node u has a probability $p_{u,v}$ of convincing node v . However, the past history of attempts to convince v might influence this probability, that's why you should get to hubs when you're the most sure you're going to convince them. So we can modify that probability as $p_{u,v}(S)$, with S being the set of nodes who already tried to influence v ^{13,14,15}. The process ends when all infected nodes exhausted all their chances of convincing people so no more moves can happen.

So you get the problem: find the set of cascade initiators I_0 such that, when the infection process ends at time t , the share of infected nodes in the network i_t is maximized. Kempe et al. solve the problem with a greedy algorithm. We start from an empty I_0 . Then we calculate for each node its marginal utility to the cascade. We add the node with the largest utility, meaning the number of potential infected nodes, to I_0 and we repeat until we reach the size we can afford to infect. Of course, each node we add to I_0 changes the expected utility of each other node, because they might have common friends, thus we cannot simply choose the $|I_0|$ nodes with the largest initial utility.

There are many improvements for this algorithm, focused on improving time efficiency, lowering the expected error, and integrating different utility functions. However, things get more interesting when you start adding metadata to your network. For instance, Gurumine¹⁶ is a system that lets you create influence graphs, as I show in Figure 21.6. You start from a social network (Figure 21.6(a)) and a table of actions (Figure 21.6(b)). You know when a node did what.

You can use the data to infer that node v does action a_1 regularly after node u performed the same action. In the example, for two actions a_1 and a_2 you see node 2 repeating immediately after node 1. Since these two nodes are connected, maybe node 1 is influencing node 2. You can use that to infer $p_{u,v} = 0.66$ (Figure 21.6(c)) – or, if you're really gallant, to infer $p_{u,v}(S)$ by looking at all neighbors of v performing a_1 before it.

Note that node 6 performed the same action at the same time as node 3. Node 6 could only be influenced by node 2. For node 3 we prefer inferring that node 1 did it, because we know that it influenced node 2 too, so that's the most parsimonious hypothesis. The size in

¹² Jacob Goldenberg, Barak Libai, and Eitan Muller. Using complex systems analysis to advance marketing theory development: Modeling heterogeneity effects on new product growth through stochastic cellular automata. *Academy of Marketing Science Review*, 9(3):1–18, 2001

¹³ David Kempe, Jon Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 137–146. ACM, 2003

¹⁴ David Kempe, Jon Kleinberg, and Éva Tardos. Influential nodes in a diffusion model for social networks. In *International Colloquium on Automata, Languages, and Programming*, pages 1127–1138. Springer, 2005

¹⁵ Note that, if we only use $p_{u,v}$ we call this *independent cascade model*, because the previous attempts do not influence future attempts. When we introduce $p_{u,v}(S)$ the cascades are not independent any more. Specifically, for the paper I'm citing, we have *decreasing* cascades because, the more people try, the hardest it is to convince v , i.e. $p_{u,v}(S) < p_{u,v}(S \cup z)$. If we did the opposite, $p_{u,v}(S) > p_{u,v}(S \cup z)$, then this model would be practically equivalent to the threshold model: the more infected neighbors you have, the more likely you're going to turn.

¹⁶ Amit Goyal, Francesco Bonchi, and Laks VS Lakshmanan. Learning influence probabilities in social networks. In *Proceedings of the third ACM international conference on Web search and data mining*, pages 241–250. ACM, 2010

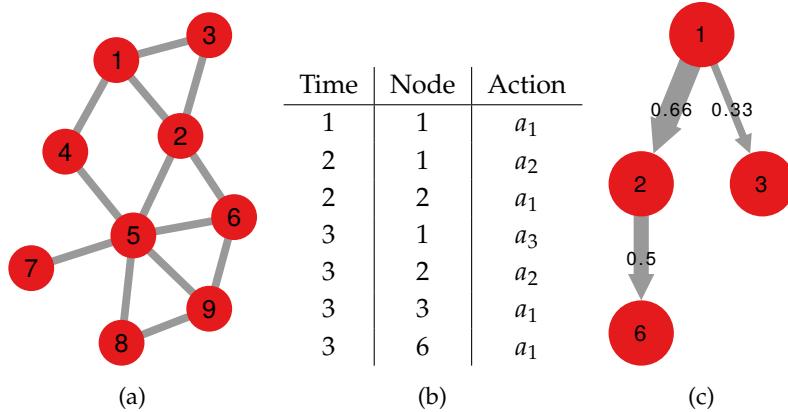
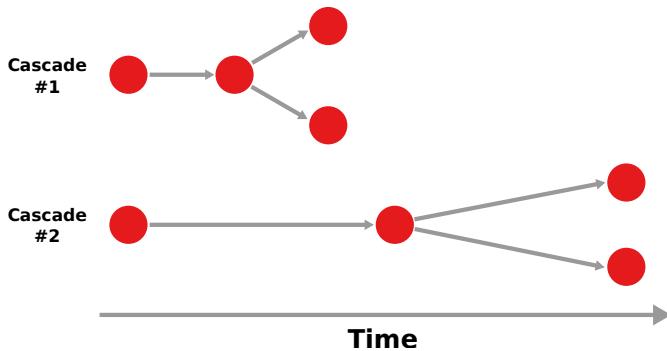


Figure 21.6: (a) The underlying social network. (b) The actions nodes made. (c) A possible inferred influence graph.

number of nodes of these cascades can be approximated by – you guessed it – a power law¹⁷.

When running such models on real data you can find funny things. For instance, I ran it with some co-authors on LastFM data, a social network recording which user listened to which musical artist at which time¹⁸ – the artist is considered the “action”. In doing so, we discovered that we could build these influence graphs and describe their trade offs. For instance, the more intensely a user was influenced by a prominent friend – meaning that they listened the new artist a lot – the fewer friends the influencer hit. In other words: the stronger you want to influence people, the fewer people you can influence.



Similar studies on Facebook tried to find which early signs we can use to predict the size of a cascade. A cascade is when I share something on my profile, then other people share it too and so on until it hits the news. Counterintuitively, the answer seem to have very little to do with the actual content of the idea per se, but with the speed with which it triggers other people^{19,20}. For instance, in Figure 21.7 we have two hypothetical cascades with the same number of shares and the same topological pattern: one re-sharer then two.

¹⁷ Eytan Bakshy, Jake M Hofman, Winter A Mason, and Duncan J Watts. Everyone’s an influencer: quantifying influence on twitter. In *Proceedings of the fourth ACM international conference on Web search and data mining*, pages 65–74. ACM, 2011.

¹⁸ Diego Pennacchioli, Giulio Rossetti, Luca Pappalardo, Dino Pedreschi, Fosca Giannotti, and Michele Coscia. The three dimensions of social prominence. In *International Conference on Social Informatics*, pages 319–332. Springer, 2013.

Figure 21.7: Anatomies of two different cascades. Time flows from left to right. A node at a given position on the x axis denotes when they share the content on their profile. An arrow indicates from where they re-shared an item, i.e. who influenced them.

¹⁹ Justin Cheng, Lada Adamic, P Alex Dow, Jon Michael Kleinberg, and Jure Leskovec. Can cascades be predicted? In *WWW*, pages 925–936. ACM, 2014.

²⁰ Justin Cheng, Lada A Adamic, Jon M Kleinberg, and Jure Leskovec. Do cascades recur? In *WWW*, pages 671–681. ACM, 2016.

However, the fact that the first cascade happened faster is enough for us to infer that it's much more likely to end up being much larger than the second, slower, one. I'm going to explore more in depth this idea about memes spreading when talking about classical results in network analysis in Chapter 52.

You can further complicate models by having competing ideas spreading into the network. There are some people who are complete enthusiasts about iPhones, while others really hate them. The love/hate opinions are both competing to spread through the network. You can see them, for instance, as a physical heating/cooling process which will eventually make nodes converge to a given temperature²¹. The classic survey of viral marketing applications of network analysis²² is a good starting point for diving deeper into the topics only skimmed in this section.

21.3 Interventions

Once we have a disease spreading through a social network, we might be interested in using our knowledge to prevent people to become sick. In practice, if this were a SIR model, we want to flip some people directly from the *S* to the *R* state, without passing by *I*. This is equivalent to vaccinate them and, if done properly, would stop the epidemics in its tracks. You can try an online game with this premise and see how much of a network you can save from an evil disease²³.

The first question is: who should we vaccinate? The answer is rather obvious once you run your simulation numbers: the hubs. If the disease attacks the hubs, it will spread to the entirety of the network almost instantly. This assumes that its degree exponent is $\alpha < 3$ and we know that, unfortunately, this is true for the majority of social systems we know.

However, now we have a second question: how do we find hubs? We might not have a complete – or even a partial! – picture of our social network. Luckily, this book has prepared you to figure out a way to find hubs even if you know nothing about the network's topology. You can exploit the fact that hubs have lots of connections. The simplest and unreasonably effective vaccination strategy is to pick a node at random in the network and vaccinate one of its friends²⁴. Statistically speaking, the friend of our random sampled individual is more likely to have a higher degree than our first choice.

To see why, consider Figure 21.8. Here we apply the “vaccinate-a-friend” strategy and report the probability of choosing each node. Note that this is done completely blindly, we don't know anything about the topology of this network. If we were to vaccinate the

²¹ Hao Ma, Haixuan Yang, Michael R Lyu, and Irwin King. Mining social networks using heat diffusion processes for marketing candidates selection. In *Proceedings of the 17th ACM conference on Information and knowledge management*, pages 233–242. ACM, 2008

²² Jure Leskovec, Lada A Adamic, and Bernardo A Huberman. The dynamics of viral marketing. *ACM Transactions on the Web (TWEB)*, 1(1):5, 2007a

²³ <https://github.com/digitalepidemiologylab/VaxGame>

²⁴ Reuven Cohen, Shlomo Havlin, and Daniel Ben-Avraham. Efficient immunization strategies for computer networks and populations. *Physical review letters*, 91(24):247901, 2003

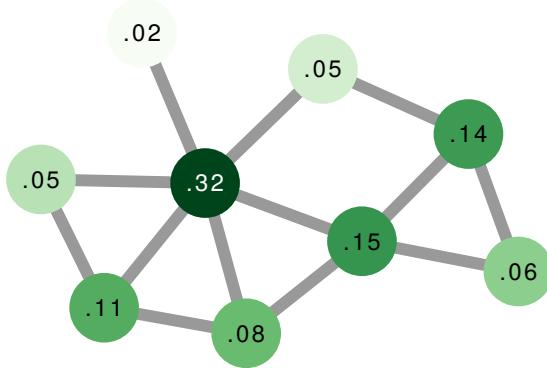


Figure 21.8: The probability of vaccinating a node with the “vaccinate-a-friend” strategy.

randomly sampled node, we would have only one chance out of nine to find the hub, given that the example has nine nodes. However, the probability of vaccinating the hub with our strategy is almost three times as high. This is related to a curious network effect on hubs, known as the “Friendship Paradox”, which we’ll investigate further in Section 31.2.

Of course, this strategy makes a number of assumptions that might not hold in practice. For instance, we only consider a simple SIR model, without looking at the possibility of complex contagion. Luckily, there is a wealth of research relaxing this assumption and proposing ad hoc immunization strategies that can work in realistic scenarios²⁵. One of the most historically important approaches in this category is Netshield²⁶.

How do we know if we did a good job? How can we evaluate the impact of an intervention? There are two things we want to look at. First, we look at the size of the final infected set and simply subtract the predicted infected share without immunization with the one with immunization. The higher the difference the better. Figure 21.9 gives you a sense of this. An SI model without immunization reaches saturation when all nodes are infected. A smart immunization strategy can make sure that the outbreak stops at a share lower than 100%.

A second criterion might be just delaying the inevitable. Once immunized, the nodes can revert to the S state, and therefore to I , after a certain amount of time. This time can be used to develop a real vaccine or might be a feature in itself, preventing having too many people transitioning to I at the same time. Figure 21.10 provides an example. In this case, we might want to either calculate the time t at which the system reaches saturation, or compute the area between the two curves as a more precise sense of the delay we imposed.

We can combine the two criteria at will. By immunizing nodes, we make the disease unable to reach saturation at 100% infection AND

²⁵ Chen Chen, Hanghang Tong, B Aditya Prakash, Charalampos E Tsourakakis, Tina Eliassi-Rad, Christos Faloutsos, and Duen Horng Chau. Node immunization on large graphs: Theory and algorithms. *IEEE Transactions on Knowledge and Data Engineering*, 28(1): 113–126, 2015

²⁶ Hanghang Tong, B Aditya Prakash, Charalampos Tsourakakis, Tina Eliassi-Rad, Christos Faloutsos, and Duen Horng Chau. On the vulnerability of large graphs. In *2010 IEEE International Conference on Data Mining*, pages 1091–1096. IEEE, 2010

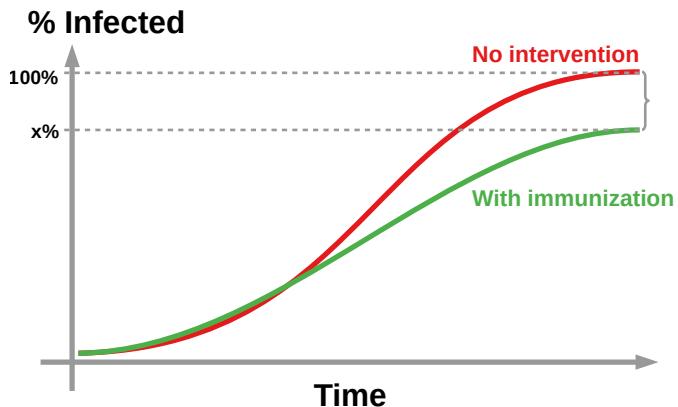


Figure 21.9: The first criterion of immunization success: the share of infected nodes at the end of the outbreak is lower than 100% in a SI model.

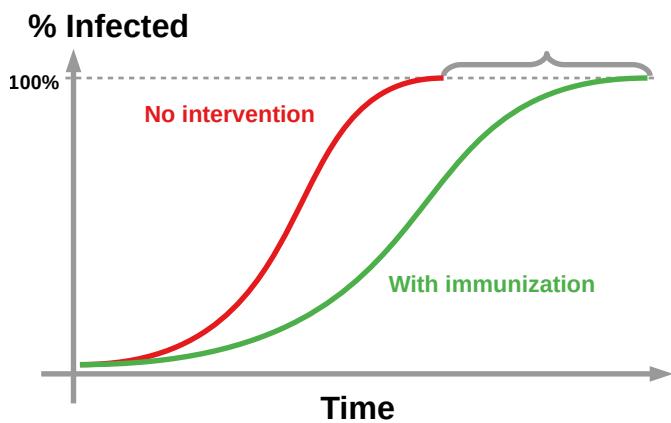


Figure 21.10: The second criterion of immunization success: temporary immunity can delay propagation.

we delay its spread in the network. Thus the two scenarios are not mutually exclusive.

Obviously, here I only assumed the perspective of limiting the outbreak of a disease. If you're in the viral marketing case you can invert the perspective: your interventions want to favor the spread of the idea in the social network. In this case, the second scenario makes more sense: even if the idea was bound to reach everyone eventually, if it does so faster it can have great repercussion. Think about the scenario of condom use to prevent HIV infections. You want to convince as many people as fast as you can, even if eventually your message was going to reach everybody anyway.

21.4 Controllability

A related problem is the classical scenario of the controllability of complex networks^{27,28,29}. Here the task is slightly different: nodes can change their state freely and there can be an arbitrary number of states in the network. What we want to ensure is that all – or most – nodes in the network end up in the state we desire. To do so, we

²⁷ Yang-Yu Liu, Jean-Jacques Slotine, and Albert-László Barabási. Controllability of complex networks. *nature*, 473(7346):167, 2011

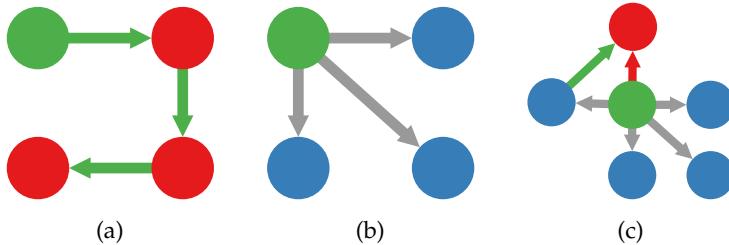
²⁸ Jianxi Gao, Yang-Yu Liu, Raissa M D'souza, and Albert-László Barabási. Target control of complex networks. *Nature communications*, 5(1):1–8, 2014

²⁹ Gang Yan, Georgios Tsekenis, Baruch Barzel, Jean-Jacques Slotine, Yang-Yu Liu, and Albert-László Barabási. Spectrum of controlling and observing complex networks. *Nature Physics*, 11(9):779–786, 2015

need to identify driver nodes: the smallest possible subset of nodes we have to manipulate so that they will influence the other nodes to switch to the state we want them to assume.

There already is a branch of mathematics dedicated to figure out how to control simple engineered or natural systems, unoriginally named control theory^{30,31}. However, we define complex systems exactly on their nature of being difficult to predict, as their parts interact with each other and thus let non-obvious properties and behaviors emerge.

In complex systems, controllability is a bit more complicated. Figure 21.11 shows a few simple examples. In Figure 21.11(a), a chain, we only need to control the origin of the chain and the rest of the system will fall into place. In Figure 21.11(b), somewhat surprisingly, one needs to control at least two among the blue nodes besides the hub in green to ensure control the system. One also needs to control three of the four blue nodes in Figure 21.11(c). Unfortunately, the mathematical details to reach this conclusion are beyond the scope of this book, and I invite you to read the papers cited at the beginning of this section if you're interested in them.



As you might expect from a complex network paper, the final conclusion is that sparse networks with a power law degree distribution characterized by a low α exponents are extremely difficult to control: they require a large number of driver nodes. In another twist going against our intuition, driver nodes tend not to be hubs. You would expect nodes connecting to the majority of the network to be natural choices to control the system, yet it seems that peripheral nodes have their role.

There are numerous applications of controllability in complex systems, for instance in networks modeling the brain³².

21.5 Summary

1. In simple contagion at each timestep you have the same chance of getting infected if you have one or more infected neighbors. In complex contagion more infected neighbors reinforce the in-

³⁰ Ernest Bruce Lee and Lawrence Markus. Foundations of optimal control theory. Technical report, Minnesota Univ Minneapolis Center For Control Sciences, 1967

³¹ B Francis. *A course in H 1 control theory. Lectures notes in control and information sciences*, volume 88. Springer Verlag Berlin, 1987

Figure 21.11: Different controllability scenarios. The nodes we're forced to select as drivers are in green, the ones we could or could not choose are in blue, the ones we don't need as drivers are in red. Similarly for links, the links we need to have to ensure controllability are in green, the ones we could or could not have are in gray, and the ones we could remove from the network without hampering controllability are in red.

³² Jonathan D Power, Alexander L Cohen, Steven M Nelson, Gagan S Wig, Kelly Anne Barnes, Jessica A Church, Alecia C Vogel, Timothy O Laumann, Fran M Miezin, Bradley L Schlaggar, et al. Functional network organization of the human brain. *Neuron*, 72(4):665–678, 2011

fection chances. Different models work with different triggering mechanisms.

2. Classically, you can have an independent β probability to transition for each infected neighbor. In the threshold model you need at least κ neighbors, independently of your degree. In the cascade model you need at least a fraction of neighbors.
3. This changes the behavior of hubs: in the threshold model it is easy to have at least κ infected contacts because hubs have so many neighbors, but for the very same reason it is difficult to reach the relative limit in the cascade model.
4. You can estimate which type of infection model a real world outbreak follows by estimating the universality class of the spreading, via its parameter ϕ : $\phi > 0$ is similar to a threshold model (positive correlation between degree and chance of infection), $\phi < 0$ is similar to a cascade model (negative correlation between degree and chance of infection).
5. In viral marketing models of word-of-mouth you don't have infinite chances to infect a node. The problem becomes identifying the set of initial infected seeds so that you maximize the number of infected nodes in the network.
6. One effective strategy to prevent a global outbreak is to immunize the friends of randomly chosen nodes. This strategy works because the randomly picked neighbors of randomly picked nodes are more likely to be hubs.

21.6 Exercises

1. Modify the SI model developed in the exercises of the previous chapter so that it works with a threshold trigger. Set $\kappa = 2$ and run the threshold trigger on the network at <http://www.networkatlas.eu/exercises/21/1/data.txt>. Show the curves of the size of the I state for it (average over 10 runs, each run of 50 steps) and compare it with a simple (no reinforcement) SI model with $\beta = 0.2$.
2. Modify the SI model developed in the previous exercise so that it works with a cascade trigger. Set $\beta = 0.1$ and compare the I infection curves for the three triggers on the network used in the previous exercise (average over 10 runs, each run of 50 steps).
3. Modify the simple SI model so that nodes become resistant after the second failed infection attempt. Compare the I infection curves

of the SI model before and after this operation on the network used in the previous exercise, with $\beta = 0.3$ (average over 10 runs, each run of 50 steps).

4. Run a classical SIR model on the network used in the previous exercise, but set the recovery probability $\mu = 0$. At each timestep, before the infection phase pick a random node. Pick one random neighbor in status S , if it has one, and transition it to the R state. Compare the I infection curves with and without immunization, with $\beta = 0.1$ (average over 10 runs, each run of 50 steps).

22

Catastrophic Failures

In Section 21.2 we saw an interesting thing: when limiting the infection chances nodes get, the disease might be unable to reach some parts of the network. This is great when fighting a disease, but networks model much more than social systems hosting pathogens. The roads you use every day for your commute are part of a network. The power grid is a network. The beloved cat pictures you look at every day flow through edges of the interwebz. The fact that something might prevent them to reach you is alarming and deserves to be studied.

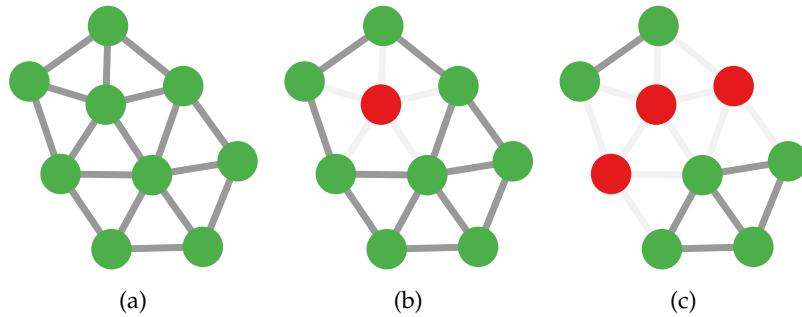


Figure 22.1: The effect of node failures on the connectivity of a network. Node color: green = active; red = failing. (a) Starting condition, all nodes active. (b) First failure. (c) Propagating failure disconnects the two nodes on top from the main component.

In this chapter we do exactly that. We again slightly change the perspective of our epidemic model to study the conditions under which networks break down. Rather than propagating a disease, we propagate failures. In their standard status, nodes are active and fulfill their duties. See Figure 22.1(a) for an example. However, for random or deliberate reasons they might transition into an *R* status: they might fail. The fundamental question is: how does the network react to such failures? Can information still flow through the Internet if some routers go down? How many blocked roads does it take for cars not to be able to drive around town?

Networks are usually resilient to small failures: a single node going down does not affect communication in the structure (see Figure 22.1(b)). Our criterion to say whether a network is still fulfilling its

purpose is the share of nodes part of its largest component. If there still is a path between all or most nodes in the network, even if it becomes longer, the network still works. However, when nodes start breaking down in multiple components and getting isolated – as in Figure 22.1(c) – then the network is failing.

We start by looking at random failures in Section 22.1 to move then to deliberate attacks in Section 22.2. We then put some dynamics on failures by considering correlated cascade failures in Section 22.3, giving a special attention to a specific case of multilayer structures: interdependent networks (Section 22.4).

This chapter is related to the mathematical problem of percolation theory¹, which has then been adapted to the network scenario^{2,3,4}. Note that I'm using the power grid example mostly as a way to give color to the math – and for traditional reasons. However, power grids failures don't necessarily follow such percolation approach. The model here propagates failures linearly and locally, but real failures in power lines are neither, due to the underlying laws governing electrical flows.

You can use these methods in other scenarios, but only if you're sure that the assumptions made here are respected in the phenomenon you're studying.

22.1 Random Failures

In this section we look at random failures. In this case, nodes can spontaneously break for uncorrelated and not deliberate reasons. Think about normal wear and tear. Any power generator can only take so much. Moreover, slight differences in the manufactory process, or in the model, can give different failure rates. Thus it is difficult to predict when one component will fail. The error rate will appear to be more or less random.

How does a network respond to these random failures? We're assuming that all its nodes are in the same Giant Connected Component (GCC), so that power can flow freely through the grid. When will the network lose its giant component? In other words: when will we need to rely on local generators rather than on the entire grid?

The answer depends on the original topology of the network. Let's start by considering the case of a random $G_{n,p}$ network. What we want to see is how much the probability of a node being part of the GCC changes as we put more and more nodes in the failure state R . I ran a few simulations and they generate a plot similar to what Figure 22.2 shows.

Now, if this seems the very same plot as one you've already seen, calm down: you're not taking crazy pills. You have indeed seen

¹ Dietrich Stauffer and Amnon Aharony. *Introduction to percolation theory*. Taylor & Francis, 2014

² Réka Albert, Hawoong Jeong, and Albert-László Barabási. Error and attack tolerance of complex networks. *nature*, 406(6794):378, 2000

³ Paolo Crucitti, Vito Latora, Massimo Marchiori, and Andrea Rapisarda. Error and attack tolerance of complex networks. *Physica A: Statistical mechanics and its applications*, 340(1-3):388–394, 2004

⁴ Jianxi Gao, Baruch Barzel, and Albert-László Barabási. Universal resilience patterns in complex networks. *Nature*, 530(7590):307–312, 2016a

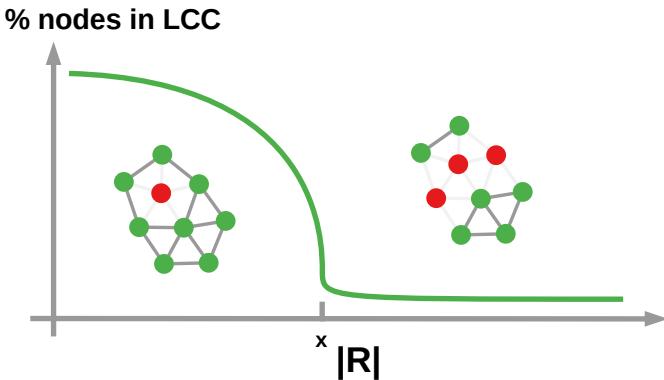


Figure 22.2: The probability of being part of the largest connected component as a function of the number of failing nodes in a random $G_{n,p}$ graph.

something like this. It was back to Figure 16.5(b), when we were talking about the probability of a node being part of the GCC in a $G_{n,p}$ model. In that case, the function on the x-axis was the probability p of establishing an edge between two nodes. In fact, the two are practically equivalent: if you have a $G_{n,p}$ graph with failures it is as if you're manipulating n and p .

What Figure 22.2 says is that a $G_{n,p}$ network will withstand small failures: a few nodes in R will not break the network apart. However, the failure will start to become serious very quickly, until we reach a critical value of $|R|$ beyond which the GCC disappears and the network effectively breaks down. Just like the appearance of a GCC for increasing p in a $G_{n,p}$ model is not a gradual process, so are random failures. At some point there is a *phase transition*, from having to not having a GCC.

Ah – you say – but we're not amateurs at this. Who would engineer a random power grid network? For sure it won't be a $G_{n,p}$ graph. Good point. In fact that's true: the power grid's degree distribution is skewed. For the sake of the argument – and the simplicity of the math – let's check the resilience to random failures of a network with a power law degree distribution.

Good news everybody! Power law random networks are more resilient than $G_{n,p}$ networks to random failures. The typical signature of a power law network under random node disappearances looks something like Figure 22.3. In the figure you see no trace of the phase transition. The critical value under which the GCC disappears is much higher than in the $G_{n,p}$ case. Of course the size of the largest connected component goes down, because you're removing nodes from the network. However, the nodes that remain in the network still tend to be able to communicate to each other, even for very high $|R|$.

Why would that be the case? The reason is always the power law degree distribution. If you remember Section 9.3, having a heavy

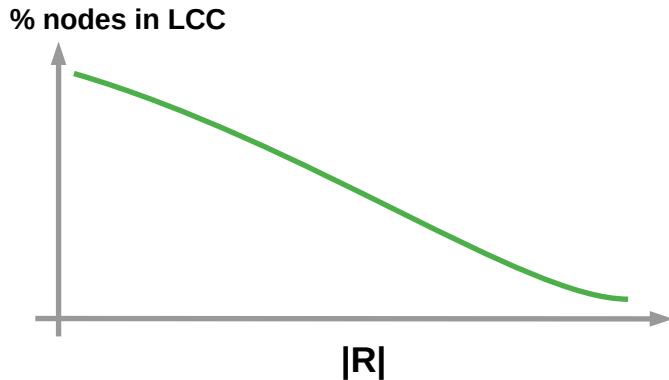


Figure 22.3: The probability of being part of the largest connected component as a function of the number of failing nodes in a network with a skewed degree distribution.

tailed degree distribution means to have very few gigantic hubs and a vast majority of nodes of low degree. When you pick a node at random and you make it fail, you're overwhelmingly more likely to pick one of the peripheral low degree ones. Thus its impact on the network connectivity is low. It is extremely unlikely to pick the hub, which would be catastrophic for the network's connectivity.

Since, by now, you must be a ninja when it comes to predict the effect of different degree exponents on the properties of a network with a power law degree distribution, you might have figured out what's next. The exponent α is related to the robustness of the network to random failures. An $\alpha = 2$, remember, means that there are fewer hubs and their degree is higher. If $\alpha > 3$, the hubs are more common and less extreme.

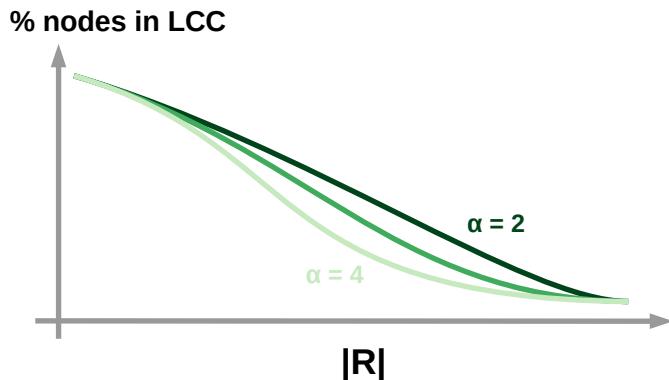


Figure 22.4: The probability of being part of the largest connected component as a function of the number of failing nodes in a network for different α exponents of its power law degree distribution.

More common hubs equals higher likelihood of picking them up in a random extraction. Thus the failure functions for different α values follow the pattern I show in Figure 22.4. The lower your α the fewer hubs, the more resilient the network. By now you probably start to get an inkling on why network scientists are so obsessed about finding that their networks are scale free. If they are, then there are tons of properties you can infer by just knowing its degree

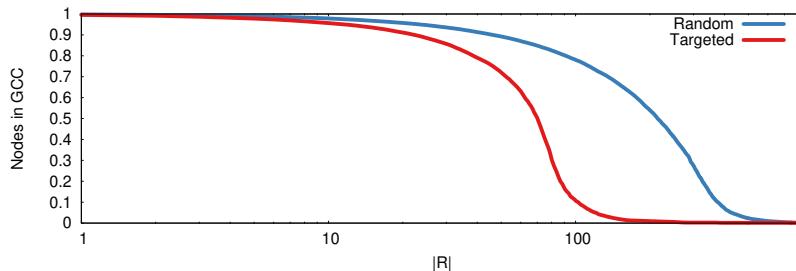
exponent and relatively simple math. In this part we already saw two: robustness to random failures (here) and outbreak size and speed in SI and SIS models (in Chapter 20).

By the way, so far we've been looking at random *node* failures, i.e. a generator blowing up in the power grid. *Edge* failures can be equally common: think about road blocks. However, the underlying math is rather similar and the functions describing the failures are not so different than the ones I've been showing you so far^{5,6}. For this reason we keep looking at node failures.

22.2 Targeted Attacks

So far we've assumed the world is a nice place and, when things break down, they do so randomly. We suspect no foul play. But what if there was foul play? What if we're not observing random failures, but a deliberate attack from a hostile force? In such a scenario, an attacker would not target nodes at random. They would go after the nodes allowing them to maximize the amount of damage while minimizing the effort required.

This translates into prioritizing attacks to the nodes with the highest degree. Taking down the node with most connections is guaranteed to cause the maximum possible amount of damage. What would happen to our network structure?



⁵ Duncan S Callaway, Mark EJ Newman, Steven H Strogatz, and Duncan J Watts. Network robustness and fragility: Percolation on random graphs. *Physical review letters*, 85(25):5468, 2000

⁶ Reuven Cohen, Keren Erez, Daniel Ben-Avraham, and Shlomo Havlin. Resilience of the internet to random breakdowns. *Physical review letters*, 85(21):4626, 2000

Figure 22.5: The probability of being part of the largest connected component as a function of the number of failing nodes in a $G_{n,p}$ network, for random (blue) and targeted (red) failures.

Let's start again by considering a $G_{n,p}$ network. I ran a few simulations and Figure 22.5 shows the result. We knew that $G_{n,p}$ networks aren't particularly good under random failures. It turns out that targeted attacks don't change the scenario much. Sure, the critical threshold is a bit lower, but the failure function is fundamentally the same.

Why? Remember that a $G_{n,p}$ model generates a normal degree distribution. This means that hubs are less common and their degree isn't much different from the average degree of all other nodes. If you pick up nodes randomly, you are likely to pick a node with

higher-than-average degree and, even if you don't, whatever you pick isn't much different.

The case is oh-so-much different when we turn our attention to networks with power law degree distributions. Since they have large hubs, prioritizing them for your attack will have devastating effects, as Figure 22.6 shows. Removing even a single node brings down the GCC size by almost 20% in this case. To make a similar damage to a $G_{n,p}$ network, you have to remove around 40 nodes.

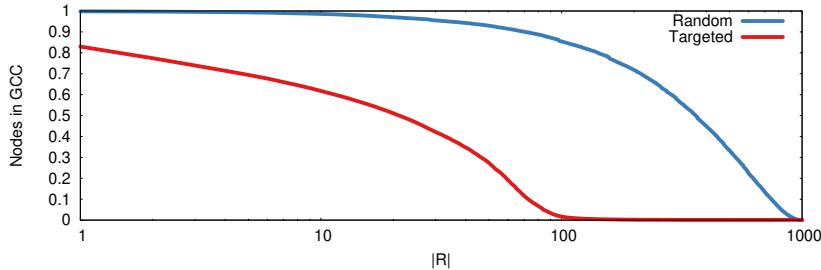
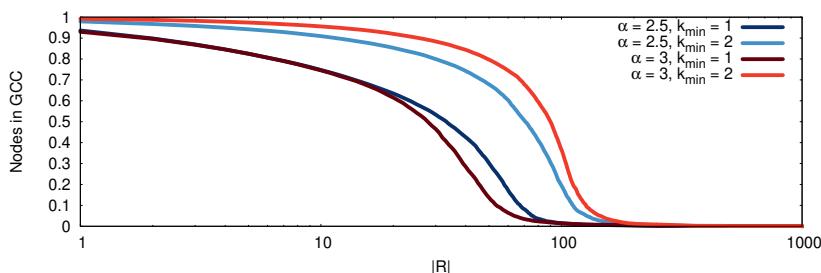


Figure 22.6: The probability of being part of the largest connected component as a function of the number of failing nodes in a degree skewed network, for random (blue) and targeted (red) failures.

In fact, networks with power law degree distributions break down *more* easily than $G_{n,p}$ equivalents, when under a targeted attacks. As a consequence, different topologies should be used for different failure scenarios. If we're talking about random failures, your should plan your network to be scale free. If you want to defend from hostile takeovers, you probably want something similar to a random $G_{n,p}$ graph or, even better, a mesh-like network⁷.



⁷ Paul Baran. Introduction to distributed communications networks. Technical report, Memorandum RM-3420-PR, Rand Corporation, 1964

Figure 22.7: The probability of being part of the largest connected component as a function of the number of failing nodes in a degree skewed network, for different α and k_{min} combinations.

As you might expect, the α exponent of the power law degree distribution has something to do with the fragility of a network to deliberate attacks. However, it is a non-linear relationship, which also depends on the minimum degree of the network k_{min} ^{8,9}. Figure 22.7 shows the results of a few simulations. If k_{min} is low, higher α exponents tend to make your network rather fragile, so it's better to have $\alpha = 2.5$ rather than $\alpha = 3$. However, if we increase k_{min} , then the opposite holds true: higher α actually make your network stronger.

⁸ Reuven Cohen, Keren Erez, Daniel Ben-Avraham, and Shlomo Havlin. Breakdown of the internet under intentional attack. *Physical review letters*, 86(16):3682, 2001

⁹ Béla Bollobás and Oliver Riordan. Robustness and vulnerability of scale-free random graphs. *Internet Mathematics*, 1(1):1–35, 2004

22.3 Chain Effects

So far in this chapter we've relying on an unreasonable assumption: failures don't propagate. We said that a power generator goes boom randomly and studied what this means for the structure of the power grid. However, it is important to note that energy demand does not go down just because there was a failure. People will still turn their light bulbs on. Therefore, whatever power that generator was providing has to come from somewhere else. However, if that generator was there, there was a reason. Maybe the other nodes in the network cannot satisfy the additional demand. Thus there is a high chance that they will themselves go out of business.

In this scenario, the failure of one node propagates in a cascade and causes more correlated failures. This sort of snowball effect can turn into an avalanche and shut the entire network down. And it has happened, many times¹⁰, also in structures that have nothing to do with power grids such as airline schedules¹¹.

The models we use to simulate such propagating failures are yet another family of variations of the threshold model from Granovetter (Section 21.1), for instance the Failure Propagation model¹². You can define failure propagation as being literally equivalent to the threshold model, by having a node V failing if a fraction f_v of its neighbors are failing. However, things become more interesting when you take into account more information.

All nodes start in the state S . They are characterized by a current load and by a total capacity. Think of this as road intersections: the load is how many cars pass on average through the intersection and the capacity is the maximum amount of cars that can pass before congestion happens.

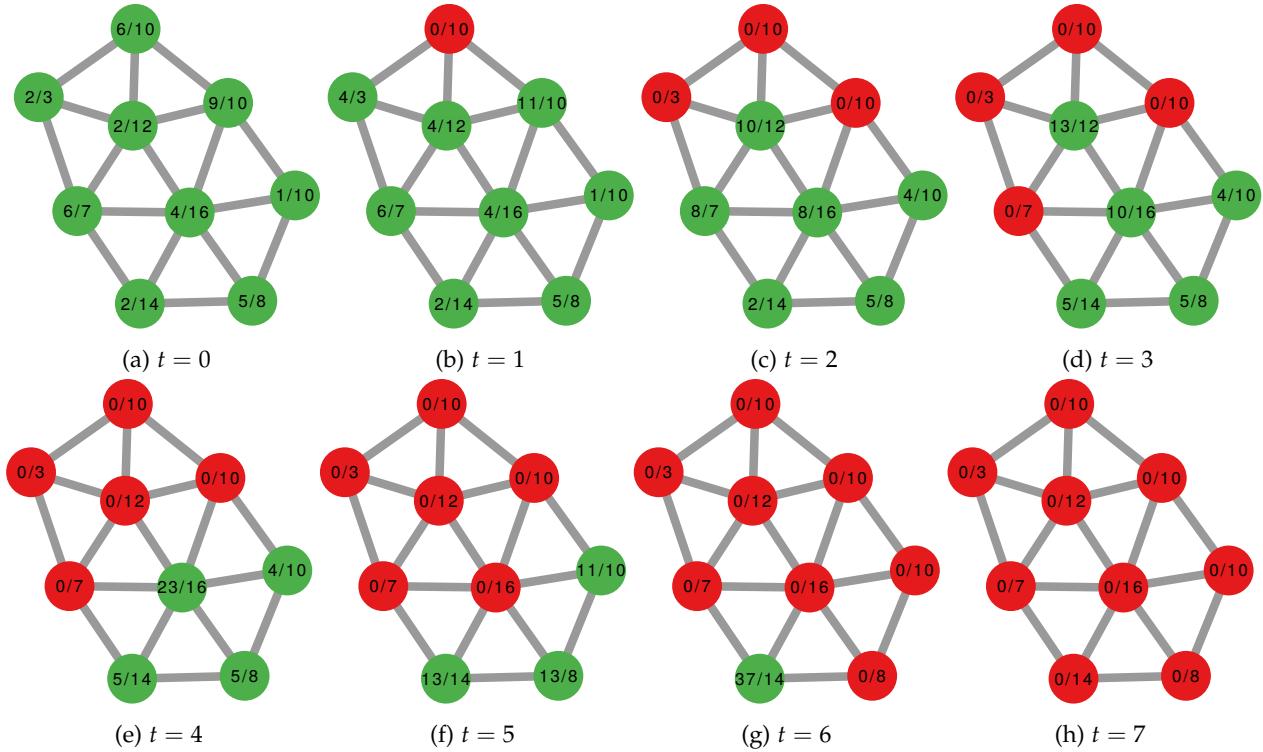
At time $t = 1$ we shut down a node in the network. Maybe the traffic light failed and so no one can pass through until we repair it. This means that the node transitions to state I . People still need to do their errands, so we have to redistribute the load of cars that wanted to pass through that intersection through alternative routes: the neighbors of that node. However, that means that their load will increase. If the new load exceeds the capacity of the node, also this node shuts down due to congestion. So its load has also to be redistributed to its neighbors and so on and so forth.

Figure 22.8 shows an example of failure propagation with this load-capacity feature. You can see that the network was built with some slack in mind: its normal total load is 37 – the sum of all loads of all nodes – for a maximum capacity of 90 – the sum of all nodes' capacities. Yet, shutting down the top node whose load was only 6 and redistributing the loads causes a cascade that, eventually, brings

¹⁰ Ian Dobson, Benjamin A Carreras, Vickie E Lynch, and David E Newman. Complex systems analysis of series of blackouts: Cascading failure, critical points, and self-organization. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 17(2):026103, 2007

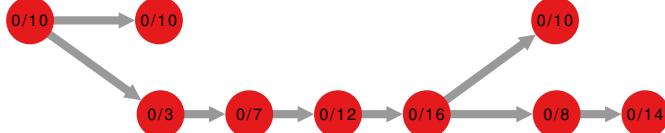
¹¹ Pablo Fleurquin, José J Ramasco, and Victor M Eguiluz. Systemic delay propagation in the us airport network. *Scientific reports*, 3:1159, 2013

¹² Ian Dobson, Benjamin A Carreras, and David E Newman. A loading-dependent model of probabilistic cascading failure. *Probability in the Engineering and Informational Sciences*, 19(1):15–32, 2005



the whole network down.

One could represent the failure cascade from Figure 22.8 as the branches of a tree. The first node failing is the root. We then connect each node to the nodes it causes to fail. The final structure would be something like Figure 22.9.



Using this perspective has its own advantages. It makes the failure propagation model more amenable to analysis. The final size of the failure cascade depends on the average degree of nodes in this tree \bar{k} . The critical value here is $\bar{k} = 1$. If, on average, the failure of a node generates another node failure – or more – the cascade will propagate indefinitely, until all nodes in the network will fail. If, instead, $\bar{k} < 1$, the failure will die out, often rather quickly.

It's easy to see why if you have the mental picture of a domino snake: each domino falling will cause the fall of another domino, until there's nothing standing. If, however, there is as much as a single gap in this chain, the rest of the system will be unaffected.

Figure 22.8: A simulated failure propagation. Each node is labeled with “load / capacity”. Green nodes are active, red nodes have failed. If load > capacity, the node will fail at the next time step.

Figure 22.9: The branch model representing the same cascade failure of Figure 22.8.

Quick show of hands: how many of you expect the size of a failure cascade to be a power law? Good, good: by now you learned that every goddamn thing in this book distributes broadly. The exponent of the cascade size is also related to the α exponent of your degree distribution. With a power degree exponent $\alpha > 3$, networks behave like $G_{n,p}$ graphs, but for $\alpha < 3$ then the cascade size will grow with exponent $\alpha / (\alpha - 1)$.

22.4 Interdependent Networks

There is an additional thing you have to consider when describing cascading failures in real world structures. So far, we have considered our networks as living in isolation. A failure in the power grid propagates only through the power grid. In our interconnected world that is not the case. In fact, once you realize how fragile networked systems can be, why would you rely on such systems without additional fail-safes? For instance, you might want to control what's happening on a power grid with an automatic computerized controller, so that it can try to isolate the failure and prevent it from propagation.

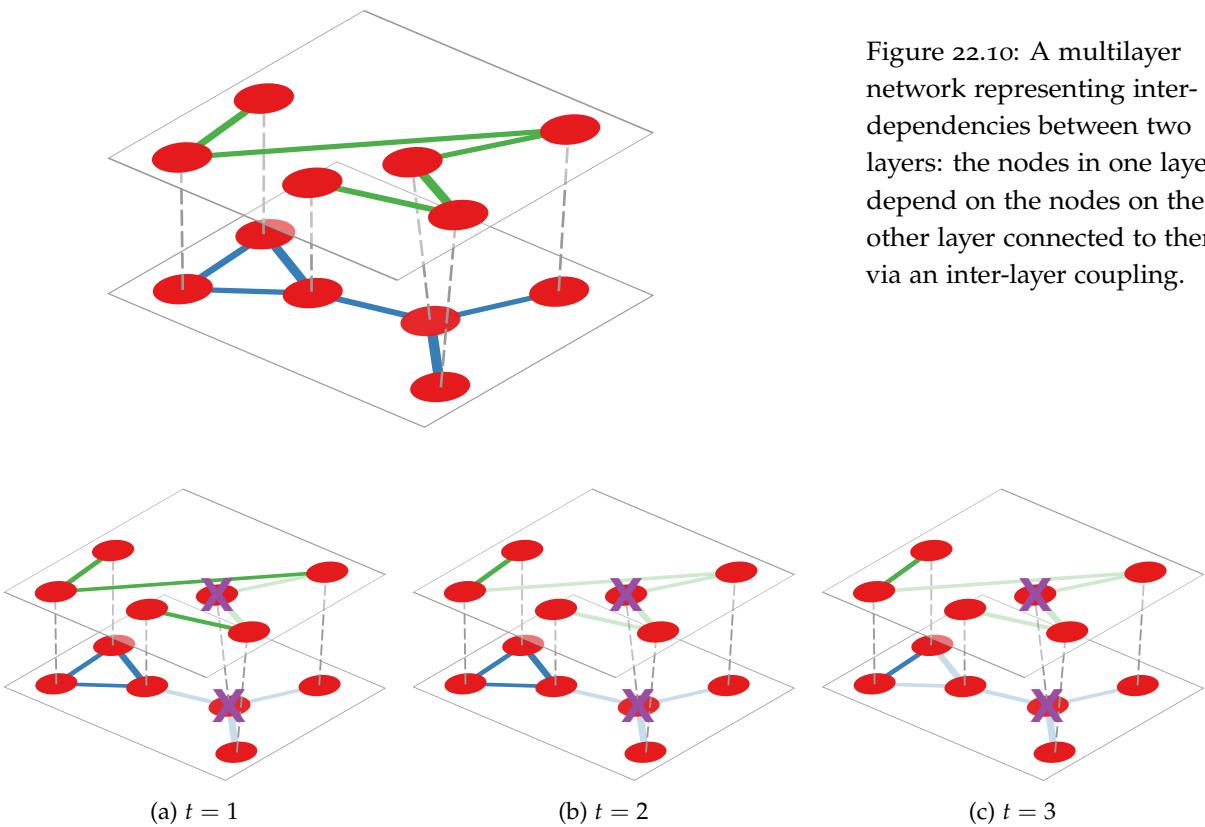
How are you going to do that with a single computer? You need a network of terminals close to the action. How are you going to provide the power they need? Through the power grid itself. So now you have two interdependent networks: the power grid needs computers to work and the computers need power to work. Interdependent networks don't behave like isolated networks¹³. Researchers have studied propagating failures in such interdependent systems and found out that *even if the two networks are resilient to random failures in isolation, the inter-dependencies cause them to be fragile to failures propagating back and forth between them*¹⁴. Ouch.

You model this problem using multilayer networks (Section 7.2). Figure 22.10 shows an example. The blue layer represents the power stations and the green layer represents the computers. A power station needs the coupled computer to work and vice versa. The network can work via connected components in both layers: if nodes get isolated in one layer, the nodes on the other layer coupled to different components get disconnected. A power station needs to know the statuses of its neighboring stations: if they are on a different computer component it cannot know it, so their links deactivate.

If a node in one layer fails (Figure 22.11(a)) it breaks its connections and causes its coupled node to lose its connections too. Now we have multiple connected components in one layer, so the links in the other layer going across components start to fail as well (Figure 22.11(b)). These failures propagate in a chain reaction until we end up in a situation where practically every node in both layers is

¹³ Jianxi Gao, Sergey V Buldyrev, H Eugene Stanley, and Shlomo Havlin. Networks formed from interdependent networks. *Nature physics*, 8(1):40, 2012

¹⁴ Sergey V Buldyrev, Roni Parshani, Gerald Paul, H Eugene Stanley, and Shlomo Havlin. Catastrophic cascade of failures in interdependent networks. *Nature*, 464(7291):1025, 2010



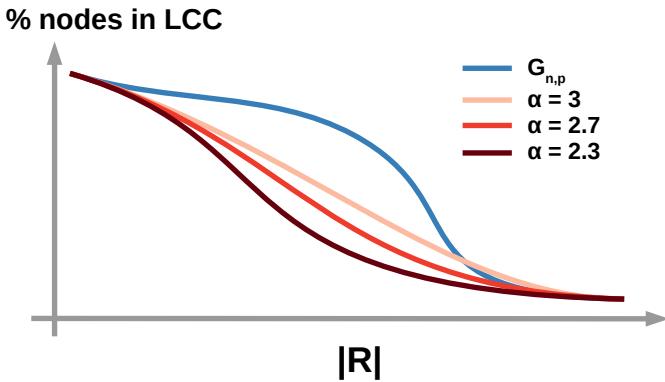
isolated, and the network almost completely failed (Figure 22.11(c)).

When describing failures in single layer networks, we asked ourselves what's the value of $|R|$ such that the network breaks down. In other words: what's the fraction of initially failing nodes that will make the GCC disappear. We can ask the same question here, realizing that, in interdependent networks, this $|R|$ value is much lower than the corresponding one for single layer networks. In fact, *if you were to calculate the critical $|R|$ value for each layer separately you would obtain a result much higher than the one for the interdependent network as a whole.* Meaning that, if you were to analyze the layers independently, you'd grossly underestimate the risk of a catastrophic failure propagating through the entire network.

Remember when, in Section 22.1, I said that networks with a heavy tail in their degree distribution are particularly robust to random failures? The reason was that large hubs keep the network together and there are very few of them, so it's unlikely to pick them up at random. Well... In two interdependent networks it is likely that hubs in one layer will couple to nodes with a lower degree in the other. Guess what: that makes coupled power law networks fragile to random failures. In fact, they're more fragile than random $G_{n,p}$

Figure 22.10: A multilayer network representing interdependencies between two layers: the nodes in one layer depend on the nodes on the other layer connected to them via an inter-layer coupling.

Figure 22.11: Propagating failures in interdependent networks. The purple cross shows the original failing nodes. At each time step, nodes depending on nodes in different components lose their connections (indicated by a faded edge color).



graphs, contrarily to what was the case before.

Figure 22.12 shows a schema of fragility for different coupled network topologies. Moving from random failures to targeted attacks still makes interdependency bad: failures will spread more easily than in isolated networks¹⁵.

Fixing this issue is not easy. First, one needs to estimate the propensity of the network to run the risk of failure. This has been done in two-layer multiplex networks¹⁶. One would think that the best thing to do is to create more connections between nodes, to prevent breaking down in multiple components. However that's not a trivial operation, as these networks are embedded in a real geographical space, where creating new power lines might not be possible. However, there are also theoretical concerns that show how more connections could render the network more fragile, as it would give the cascade more possible pathways to generate a critical failure¹⁷. A better strategy involves so-called "damage diversification": mitigating the impact of the failure of a high degree node¹⁸.

Note that we assumed that power law networks are randomly coupled: hubs in one layer will pick a random node to couple to in the other layer. As a consequence, they'll likely to pick a low degree node. Other papers study the effect of degree correlations in inter-layer coupling¹⁹: what if hubs in one layer tend to connect to hubs in the other layer? If such correlations were perfect, we'd obtain again the robustness of power law networks to random failures. These correlations are luckily observed in real world systems^{20,21}, showing how they're not as fragile as one might fear. Phew.

22.5 Summary

1. $G_{n,p}$ networks are fragile to random failures: beyond a critical number of removed nodes the giant connected component will disappear. Networks with skewed degree distributions are instead

Figure 22.12: The relationship between the degree exponent α of coupled power law networks and the fraction of nodes in R state (x axis) needed to destroy the GCC (shades of red). In blue the equivalent plot for coupled random $G_{n,p}$ graphs.

¹⁵ Xuqing Huang, Jianxi Gao, Sergey V Buldyrev, Shlomo Havlin, and H Eugene Stanley. Robustness of interdependent networks under targeted attack. *Physical Review E*, 83(6):065101, 2011b

¹⁶ Rebekka Burkholz, Matt V Leduc, Antonios Garas, and Frank Schweitzer. Systemic risk in multiplex networks with asymmetric coupling and threshold feedback. *Physica D: Nonlinear Phenomena*, 323:64–72, 2016b

¹⁷ Charles D Brummitt, Raissa M D'Souza, and Elizabeth A Leicht. Suppressing cascades of load in interdependent networks. *Proceedings of the National Academy of Sciences*, 109(12):E680–E689, 2012

¹⁸ Rebekka Burkholz, Antonios Garas, and Frank Schweitzer. How damage diversification can reduce systemic risk. *Physical Review E*, 93(4):042313, 2016a

¹⁹ Byungjoon Min, Su Do Yi, Kyu-Min Lee, and K-I Goh. Network robustness of multiplex networks with interlayer degree correlations. *Physical Review E*, 89(4):042811, 2014

²⁰ Roni Parshani, Celine Rozenblat, Daniele Ietri, Cesar Ducruet, and Shlomo Havlin. Inter-similarity between coupled networks. *EPL (Europhysics Letters)*, 92(6):68002, 2011

²¹ Saulo DS Reis, Yanqing Hu, Andrés Babino, José S Andrade Jr, Santiago Canals, Mariano Sigman, and Hernán A Makse. Avoiding catastrophic failure in correlated networks of networks. *Nature Physics*, 10(10):762, 2014

robust because they rely on few hubs which are unlikely to be picked by random failures.

2. In targeted attacks we take down nodes from the most to least connected. Power law random networks are very fragile and break down quickly under this scenario.
3. When one node fails, all its load needs to be redistributed to non-failing nodes. This can and will make the failure propagate on the network in a cascade event which might end up bringing the entire network down.
4. In interdependent networks we have a multilayer network whose nodes in one layer are required for the functioning of nodes in the others. Depending on the degree correlations among layers, failures can propagate across layer and bring down power law networks even under random accidents.

22.6 Exercises

1. Plot the number of nodes in the largest connected component as you remove 2,000 random nodes, one at a time, from the network at <http://www.networkatlas.eu/exercises/22/1/data.txt>. (Repeat 10 times and plot the average result)
2. Perform the same operation as the one from the previous exercise, but for the network at <http://www.networkatlas.eu/exercises/22/2/data.txt>. Can you tell which is the network with a power law degree distribution and which is the $G_{n,p}$ network?
3. Plot the number of nodes in the largest connected component as you remove 2,000 nodes, one at a time, in descending degree order, from the networks used for the previous exercises. Does the result confirm your answer to the previous question about which network is of which type?
4. The network at <http://www.networkatlas.eu/exercises/22/4/data.txt> has nodes metadata at http://www.networkatlas.eu/exercises/22/4/node_metadata.txt, telling you the current load and the maximum load. If the current load exceeds the maximum load, the node will shut down and equally distribute all of its current load to its neighbors. Some nodes have a current load higher than their maximum load. Run the cascade failure and report how many nodes are left standing once the cascade finishes.

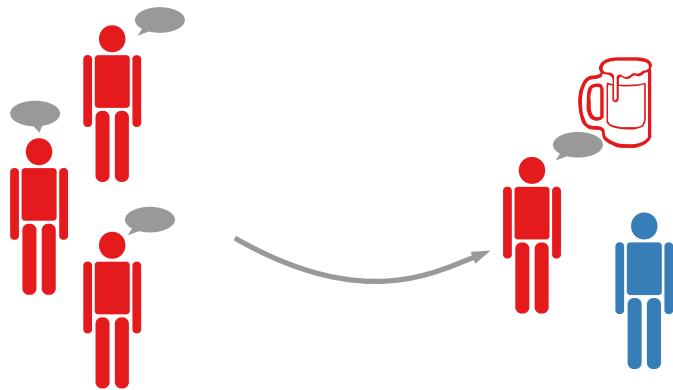
Part VII

Link prediction

23

For Simple Graphs

Link prediction is the branch of network analysis that deals with the prediction of new links in a network. In link prediction, you see the network as fundamentally dynamic, it can change its connections. Suppose you're at a party. You came there with your friends, and you're talking to each other, using the old connections. At some point, you want to go and get a drink so you detach from the group. On the way, you could meet a new person, and start talking to them. This creates a new link in the social network. Link prediction wants to find a theory to predict these events – in this case, that alcohol is the main cause of new friendships at parties, or so I'm told –, as I show the vignette in Figure 23.1.



In other words, given a network with nodes and edges, we want to know which link is the most likely to appear in the future. Or, if we think we're seeing an incomplete version of the network, we ask ourselves which edges are currently missing from the structure.

Link prediction happens in three steps. The starting point is your desire to place a new link in the network: which edge will appear next? The first thing you do is to observe the current links. On the basis of this observation you formulate a hypothesis on how nodes decide to link in the network. Finally, you operationalize this

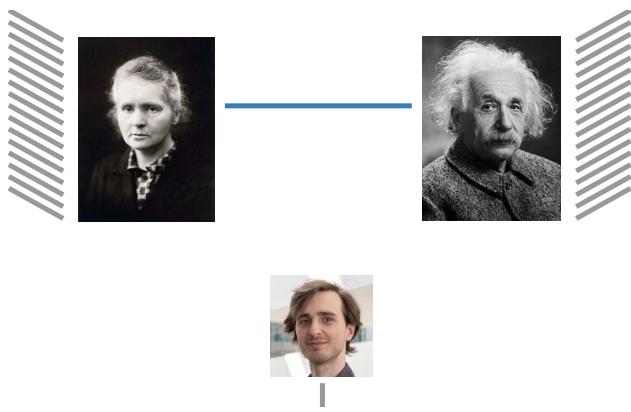
Figure 23.1: A vignette explaining the link prediction aim. at a party, red individuals know each other and express their relationships (or links) by talking. One member might detach from the group for any reason and, in doing so, she exposes herself to the possibility of establishing a new link with the blue individual. Link prediction is all about finding the true reason that might make this happen.

hypothesis: if nodes are created via process x , you apply x to the current status of the network and that will tell you which link is most likely to appear next.

In this chapter, we are going to focus on the simplest possible case for link prediction: predicting new links in a simple graph. We delve deep into the classical approaches to link prediction, which are the simplest and most used in the literature^{1,2,3,4,5}: Preferential Attachment, Common Neighbor, Adamic-Adar, Hierarchical Random Graph models, Resource Allocation, and Graph Evolution Rules. We also briefly mention other approaches, to give justice to a gigantic subfield of network analysis in computer science.

We will deal with multilayer link predictions later, in Chapter 24. Once we understand how to assign a score to every possible future link, it's time to estimate whether we did a good job. This will be covered in Chapter 25.

23.1 Preferential Attachment



Let's start with Preferential Attachment (PA). Consider scientific publishing. We have three authors: two of them – Einstein and Curie – have a lot of collaborators, while the third – me – has only few – see Figure 23.2. If we have to make a guess of what collaboration is more likely to happen next, which one would we expect? It's more likely to see the two high degree hubs to connect, because they're more prominent, and thus visible to each other.

If we want to predict links, we have to formulate a hypothesis and then translate it into a score of u connecting to v for any pair of u, v nodes: $score(u, v)$. In PA, the hypothesis is that “rich get richer”, nodes with lots of edges will attract more edges^{6,7}. So we look for pairs of nodes that have attracted so far the most edges. Our PA model would consider it strange if they are not connected to each

¹ Lise Getoor and Christopher P Diehl. Link mining: a survey. *Acm Sigkdd Explorations Newsletter*, 7(2):3–12, 2005

² David Liben-Nowell and Jon Kleinberg. The link-prediction problem for social networks. *Journal of the American society for information science and technology*, 58(7):1019–1031, 2007

³ Linyuan Lü and Tao Zhou. Link prediction in complex networks: A survey. *Physica A: statistical mechanics and its applications*, 390(6):1150–1170, 2011

⁴ Peng Wang, BaoWen Xu, YuRong Wu, and XiaoYu Zhou. Link prediction in social networks: the state-of-the-art. *Science China Information Sciences*, 58(1):1–38, 2015a

⁵ Víctor Martínez, Fernando Berzal, and Juan-Carlos Cubero. A survey of link prediction in complex networks. *ACM Computing Surveys (CSUR)*, 49(4):69, 2017

Figure 23.2: An example of a Preferential Attachment link prediction. Two hubs (Einstein and Curie) have a lot of connections (in gray), while a third author only has few. For PA, the most logical link to predict is in blue between the hubs, because rich get richer, and thus they will attract the new connections.

⁶ Mark EJ Newman. Clustering and preferential attachment in growing networks. *Physical review E*, 64(2):025102, 2001a

⁷ Albert-Laszlo Barabási, Hawoong Jeong, Zoltan Néda, Erzsebet Ravasz, Andras Schubert, and Tamas Vicsek. Evolution of the social network of scientific collaborations. *Physica A: Statistical mechanics and its applications*, 311(3-4):590–614, 2002

other. Our best guess is that they will connect soon. In practice, the probability of connecting two nodes is directly proportional to their current degree: $score(u, v) = k_u k_v$, where k_u and k_v are u 's and v 's degrees, respectively.

23.2 Common Neighbor

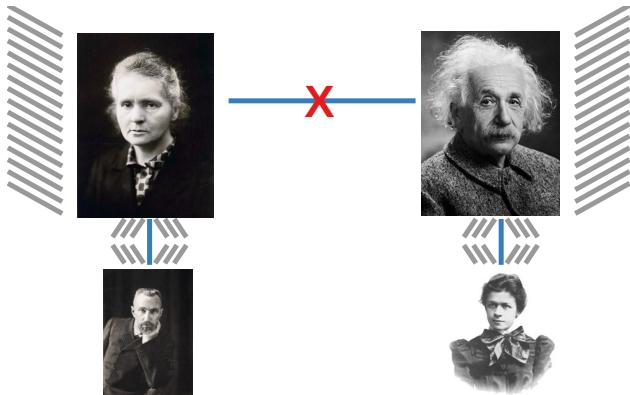


Figure 23.3: An example of a Common Neighbor link prediction. The hubs (Einstein and Curie) do not share any connection, thus it's less likely they will connect to each other. But they share a lot of connections with other lower degree nodes. CN will give the possibility of linking to them a boost.

The preferential attachment example in Figure 23.2 has one defect: its prediction is wrong, Curie and Einstein never collaborated. This is because PA fails to consider the social element: it is more likely to collaborate not only if one is good at collaborating, but also if the two people are likely to meet. Given that Curie and Einstein are from slightly different fields, it is difficult for a meeting between the two to stick into a collaboration. On the other hand, they might have a lot of common collaborators with other people in the same field: Curie shared a Nobel prize with her husband Pierre Curie, and Einstein owes a great debt to Marić – see Figure 23.3. Neither Pierre nor Marić had as many collaborations as Curie or Einstein, but for the Common Neighbor (CN) model the thing that matters most is the number of neighbors they share with them.

Common Neighbor's basic theory is that triangles close: the more common neighbors u and v have, the more triangles we can close with a single edge connecting u to v ⁸. So the likelihood of connecting two nodes is proportional to the number of shared elements in their neighbor sets: $score(u, v) = |N_u \cap N_v|$, where N_u and N_v are the set of neighbors of u and v , respectively. A variant controls for how many neighbors the two nodes have: the same number of common neighbors weighs more if it's the total set of connections the two neighbors have. This is the Jaccard variant: $score(u, v) = |N_u \cap N_v| / |N_u \cup N_v|$.

⁸ Mark EJ Newman. Clustering and preferential attachment in growing networks. *Physical review E*, 64(2):025102, 2001a

23.3 Adamic-Adar

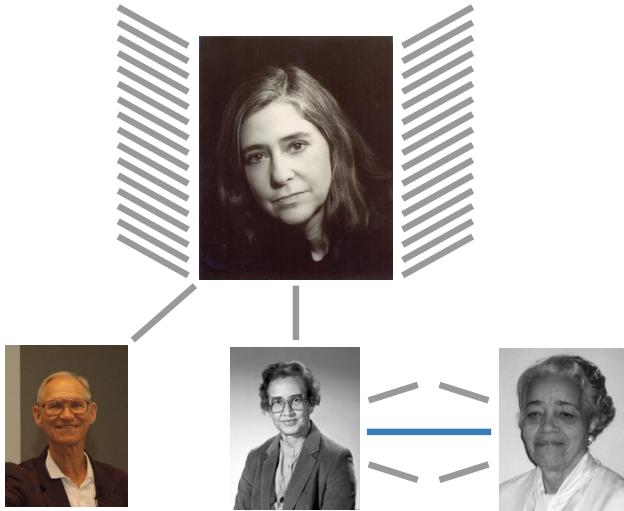


Figure 23.4: An example of a Adamic-Adar link prediction. As a hub, Hamilton (top) does not have enough time to introduce all possible pairs of her many collaborators. Thus it's less likely Johnson (middle bottom) can connect to Boehm (left bottom) than she can connect to Vaughan (right bottom), with whom she shares common connections with fewer collaborators than Hamilton.

Common Neighbor has a problem which is bandwidth. Even if you have Hamilton as a collaborator, she has collaborated with so many other people that the likelihood of Johnson to connect to Boehm – both Hamilton's collaborators – is low, because Hamilton does not have enough bandwidth to make the introduction. On the other hand, few common collaborators, if they have few connections, can represent a stronger attraction between two people. In this case, they are more likely to make the introduction, because they have the time to do so. Thus they make the new collaboration happen, as is the case for Johnson and Vaughan – see Figure 23.4.

In Adamic-Adar (AA)⁹ we say that common neighbors are important, but the hubs contribute less to the link prediction than two common neighbors with no other links, because the hubs do not have enough bandwidth to make the introduction. In AA, our score function discounts the contribution of each node with the logarithm of its degree: $score(u, v) = \sum_{z \in N_u \cap N_v} \frac{1}{\log k_z}$. The formula says that, for each common neighbor, instead of counting one – as we do in Common Neighbor when we look at the intersection –, we count one over the common neighbor's degree (log-transformed).

23.4 Resource Allocation

The Resource Allocation index¹⁰ is almost identical to Adamic-Adar. It stems from the very same principle: nodes have bandwidth. The likelihood of u connecting to v is proportional to the amount of resources u can send to v and vice versa. Thus we have $score(u, v) =$

⁹ Lada A Adamic and Eytan Adar. Friends and neighbors on the web. *Social networks*, 25(3):211–230, 2003

¹⁰ Tao Zhou, Linyuan Lü, and Yi-Cheng Zhang. Predicting missing links via local information. *The European Physical Journal B*, 71(4):623–630, 2009

$\sum_{z \in N_u \cap N_v} \frac{1}{k_z}$. The only difference with Adamic-Adar is that the scaling is assumed to be linear rather than logarithmic. Thus, Resource Allocation punishes the high-degree common neighbors more heavily than Adamic-Adar. You can see that the difference between k_z and $\log k_z$ is practically nil for low values of k_z , but balloons when k_z is high.

One could make a more complex version of the Resource Allocation index by assuming that the bandwidth of each node and of each link is not fixed. Thus the amount of resources u sends can change, and the amount of resources that can pass through the (u, v) link can also be different from the one passing through other edges.

23.5 Hierarchical Random Graphs

With the Hierarchical Random Graph (HRG) model¹¹ we start to look at different approaches to link prediction. Its main difference with what we saw so far is that in HRG we're not just looking at pairs of nodes and their neighbors, but at the entire network. First we look at all connections and we create a hierarchical representation of it that fits the data. In practice, we want to group nodes in the same part of the hierarchy if they have a high chance of connecting. In our recurring example, the field of study of the scientist is a good way to group nodes. Then we say that it is more likely for nodes in the same part of the hierarchy to connect in the future if they haven't done so yet. Making a long path through this hierarchy to establish a new connection is less likely – see Figure 23.5.

¹¹ Aaron Clauset, Cristopher Moore, and Mark EJ Newman. Hierarchical structure and the prediction of missing links in networks. *Nature*, 453(7191):98, 2008

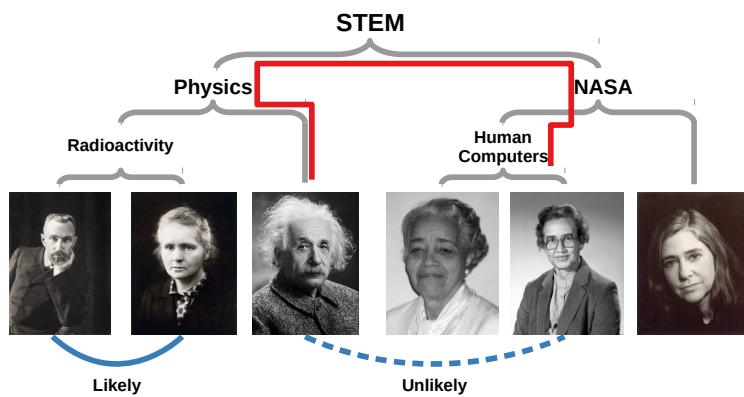


Figure 23.5: An example of a Hierarchical Random Graph link prediction. The hierarchy fits the observed connections, showing that researchers in the same field are more likely to connect. Then HRG looks at pairs of nodes in the same part of the hierarchy that are not yet connected, and gives them a higher score.

In HRG we're basically saying that communities matter: it is more likely for nodes in the same community to connect. Thus we fit the hierarchy and then we say that the likelihood of nodes to connect is proportional to the edge density of the group in which they both are. If the nodes are very related, the group containing both nodes might

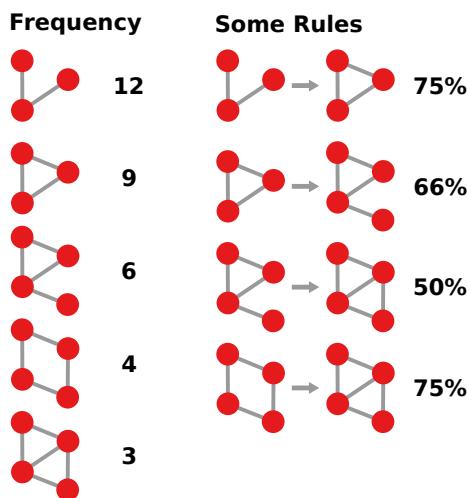
be a semi-clique with almost maximum density; if the nodes are far apart, the group containing both nodes might be just the entire network. In a schematic way: $score(u, v) = |E_c|/e(|E_c|)$, where c is the community to which u and v belong, $|E_c|$ is the number of edges it contains, and $e(|E_c|)$ is the number of edges we expect it to contain, under a null random configuration model assumption (see Section 18.1).

23.6 Association Rules

Just like HRG, GERM^{12,13} is a peculiar approach to link prediction that has almost nothing in common with the standard approach of simply evaluating node-node similarity. GERM is short for Graph Evolution Rule Mining and it is rarely considered in link prediction surveys because it's a bit harder to implement and its only known implementation is proprietary software. But the approach deserves to be mentioned, given its cleverness.

GERM looks at any possible network motif (see Section 41.2) and counts how many times each appears in the network. This is a spectacularly hard problem, and I'll tell you how to perform it in the part of this book dedicated to graph mining (Chapter 41). For now, let's just assume that an oracle told us how many times each pattern occurs in our network.

The idea is to identify all graph patterns that are a simple extensions of other, simpler patterns. By "simple extension" I mean that the consequent should have at most one additional edge – and, possibly, one additional node – added to its antecedent. This is the hard part. Once you detect all possible antecedent-consequent pairs, you can generate the rules, as Figure 23.6 shows.



¹² Michele Berlingario, Francesco Bonchi, Björn Bringmann, and Aristides Gionis. Mining graph evolution rules. In *Joint European conference on machine learning and knowledge discovery in databases*, pages 115–130. Springer, 2009.

¹³ Björn Bringmann, Michele Berlingario, Francesco Bonchi, and Arisitdes Gionis. Learning and predicting the evolution of social networks. *IEEE Intelligent Systems*, 25(4):26–35, 2010.

Figure 23.6: An example of GERM. The frequency of each pattern is on the left. The graph evolution rules with their relative frequency is on the right. Note from the last two rules how the same consequent can be predicted with different levels of confidence by two different antecedents.

Suppose you have two patterns G' and G'' , which differ only by one edge – with G'' including G' , for instance the top two patterns in Figure 23.6's left column. Given their frequencies, you know that, 75% of the times you see G' , you'll also see G'' . So you can infer, with 75% confidence, that G' evolves into G'' .

A crucial difference between GERM and whatever we saw so far is that it doesn't directly assign a similarity score to any two particular nodes. Each of the methods listed so far has a $score(u, v)$ for each u, v pair. In GERM, rather than iterating over each pair of unconnected nodes to estimate their score, we iterate over rules and identify which pairs should be connected.

To understand the process, consider Figure 23.7. Imagine our starting network is on the left. Suppose that we found two rules whose antecedents match the data. The first (on top) is a classic triad closure pattern that we see in many real world networks (see Section 12.2 for a refresher on clustering). The second (on bottom) says that these “square” patterns attract a fifth node.

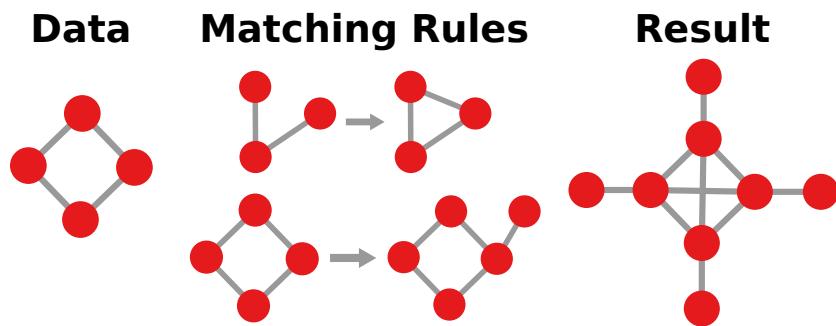


Figure 23.7: An example of link prediction in GERM. The pattern matches two rules – in the middle –, which we apply to all combinations of it to obtain, at the next time step, the extended pattern on the right.

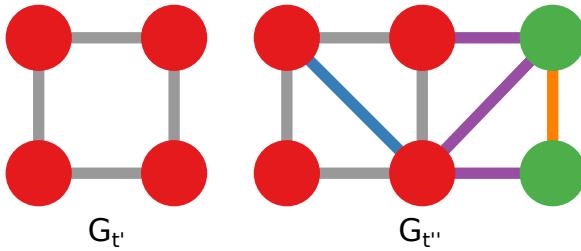
With the power of these two rules we know we can close the two open triads we have and add a new neighbor to each node in the original data. The end result is on the right. We now have more open triads and could apply the rules again, which would in turn create more square patterns and so on and so forth. In fact, one could use GERM not only as a link predictor but also as a graph generator (and put it in Chapter 17).

Note that I made two simplifications to GERM that the original papers don't make. First, in Figure 23.7, I assumed that each rule applies with the same priority. I ignored its frequency and its confidence. Of course, that would be sub-optimal, so the papers describe a way to rank each candidate new edge according to the frequency and confidence of each rule that would predict it.

Second, in all my examples I always assumed that the rules add a new edge in the next time step and that all edges in the antecedent are present at the same time. In reality, GERM allows to have more

complex rules spanning multiple time steps. You could have a rule saying something like: you have a single edge at time $t = 0$, you add a second edge at time $t = 1$ creating an open triad, and *then* you close the triangle at time $t = 2$. This triad closure rule spans three time steps, rather than only two.

GERM has a final ace up its sleeve. We can classify new links coming into a network into three groups: old-old, old-new, and new-new. We base these groups according to the type of node they attach to. I show an example in Figure 23.8. An “old-old” link appearing at time $t + 1$ connected two nodes that were already present in the network at time t . These are two “old” nodes. You can expect what an “old-new” link is: a link connecting an old node with a node that was not present at time t – a “new” node. New nodes can also connect to each other in a “new-new” link. If the network represents paper co-authorships, this would be a new paper published by two or more individuals who have never published before.



Every method we saw so far – and the ones we’ll see – predict exclusively “old-old” links. They work by creating a score between nodes u and v and, if either of those nodes were not present at time t , then their score is undefined. GERM is the only method I know that is able to predict also old-new and new-new links. Look again at Figure 23.7: the result of GERM’s prediction has more nodes than the original graph. That is because GERM predicted a few old-new links.

There’s nothing stopping GERM to, in principle, predict also new-new links. However, estimating its precision in doing so is tricky. Thus the papers presenting the algorithm did not explore that dimension.

Finally, note how all the rules I mentioned so far have no information on the nodes. This is a simplification I made which can be thrown away with ease. We can have labels on the nodes, so that we will obtain different link prediction scores for the same edges depending on the characteristics of the nodes. Figure 23.9 provides an example. Only one qualitative information can be represented at a time in this scenario. Also, implementing quantitative node attributes – such as the degree – is non-trivial, as one would ideally want to

Figure 23.8: Two observations of the graph G at time t' (left) and t'' (right). The node color encodes its type (red = “old”, green = “new”). The edge color encodes its type: gray = original link; blue = a new “old-old” link between two old nodes; purple = a new “old-new” link between an old node and a new node; orange = a new “new-new” link, between two nodes which were not in the graph at time t' .

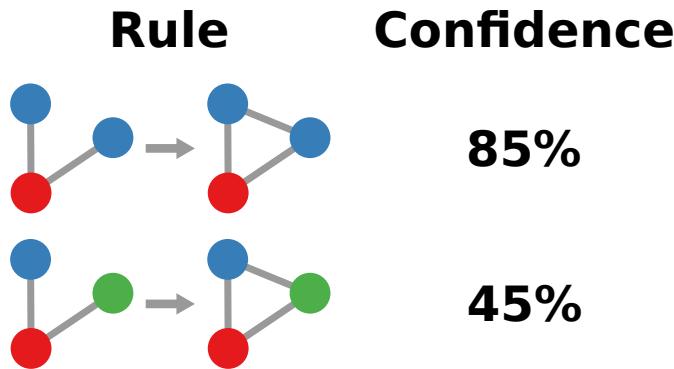


Figure 23.9: An example of link prediction in GERM considering also node types. The node color here represent the node's label.

encode the fact that the attribute values 1 and 2 should be considered “more similar” to each other than 1 and 1,000. Extensions taking care of these limitations might be possible, but I’m not aware of one.

This is not a function unique to GERM, though. Many link prediction methods can be extended to take into consideration node attributes as well. In fact, this is also a key ingredient in some network generating processes. Node attributes are used, for instance, when modeling exponential random graphs, as we saw in Section 19.2. In this case, differently than GERM, quantitative attributes represent no issue.

23.7 Other Approaches

Link prediction is a vast subfield of network analysis. It is not quite as vast as community discovery (Part X) is, but we’re not that far off. I cannot give justice to all methods out there. I chose the ones for the previous sections because they are the simplest and most didactic examples that everyone knows. In this section, I group together the most prominent examples of “all the rest”.

I don’t have the mental firepower to give you refined intuitions of the following methods as I did so far. Thus this is going to be just a big lump of concepts and formulas, necessarily superficial. If I did it any other way, this would not be a network analysis book, but a link prediction book. If you want to specialize in the field and really understand all of these methods – and more! – please go and read the review papers I cited at the beginning of the chapter, and a few newer ones¹⁴. Understood? Good.

And now: God helps us all, we’re going in.

Graph Embeddings. This is a very big family of methods which has blasted into scene recently – not only in link prediction, but in network analysis as a whole. We’re going to see in details what a “graph embedding” is in Chapter 42. For now, suffice to say that it

¹⁴ Amir Ghasemian, Homa Hosseini-mardi, Aram Galstyan, Edoardo M Airola, and Aaron Clauset. Stacking models for nearly optimal link prediction in complex networks. *Proceedings of the National Academy of Sciences*, 117(38): 23393–23400, 2020

is a way to represent a node as a vector of values. Depending on how you construct this vector of values, you could argue that similar nodes tend to connect to each other. On this basis, you can use any vector similarity measure to predict links in your network.

This is a dumbed-down version of the general approach shared by most of graph embedding link predictors. As you might expect, this general template has been applied in multiple ways to tackle different challenges. For instance, embeddings can feed their node representation to a deep neural network¹⁵; they can easily go beyond purely structural methods, because node attributes are just another entry in the vector that can be fed to a machine learning framework¹⁶; finally, they have been used to predict relations in semantic graphs^{17,18,19}, which encode relationships between different entities such as “king marries queen”.

The list goes on and on and it would deserve a specialized book on the subject, but you get the picture so let’s move on.

*Katz*²⁰. We already saw Katz’s name when we talked about centrality (in Section 14.4). His idea of centrality was one where you get a centrality contribution from your neighbors, their neighbors, the neighbors of their neighbors, and so on. With each additional degree of separation, Katz established a penalty: the farther away a node is, the less it contributes to your centrality.

We can apply a similar strategy to derive our $score(u, v)$. Two nodes are strongly related if there are many short paths between them. So one would calculate all paths between u and v and sum their count. Of course, short paths contribute more because they represent a closer relationship. Thus the formula would be something like: $score(u, v) = \sum_{l=1}^{\infty} \alpha^l |P_{u,v}^l|$.

The assumption is that, the more short paths are between u and v , the more the two nodes are related. This is regulated by the $0 < \alpha < 1$ parameter: a lower α penalizes long paths more – because they have a high l . Here, α plays the exact same role it did in Katz centrality. If we choose a very small α , the Katz score is practically equivalent to counting common neighbors, as $\alpha^2 \sim 0$.

Figure 23.10 shows an example of this correction, with longer paths fading to almost no contribution.

Hitting Time. We all tried to forget the shenanigans of Section 11.3, when I defined the hitting time: the expected number of steps it takes for a random walker to reach v from u . Alas, we’re reminded of it now, as there is a way to use $H_{u,v}$ as a basis for $score(u, v)$. After all, if u and v have very low hitting times, doesn’t it mean that they are very related, and thus likely to connect?

¹⁵ Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks. In *Advances in Neural Information Processing Systems*, pages 5165–5175, 2018

¹⁶ Lizi Liao, Xiangnan He, Hanwang Zhang, and Tat-Seng Chua. Attributed social network embedding. *IEEE Transactions on Knowledge and Data Engineering*, 30(12):2257–2270, 2018

¹⁷ Guoliang Ji, Shizhu He, Liheng Xu, Kang Liu, and Jun Zhao. Knowledge graph embedding via dynamic mapping matrix. In *IJCNLP*, pages 687–696, 2015

¹⁸ Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. Complex embeddings for simple link prediction. International Conference on Machine Learning (ICML), 2016

¹⁹ T Dettmers, P Minervini, P Stenetorp, and S Riedel. Convolutional 2d knowledge graph embeddings. In *32nd AAAI Conference on Artificial Intelligence, AAAI 2018*, volume 32, pages 1811–1818. AAAI Publications, 2018

²⁰ Leo Katz. A new status index derived from sociometric analysis. *Psychometrika*, 18(1):39–43, 1953

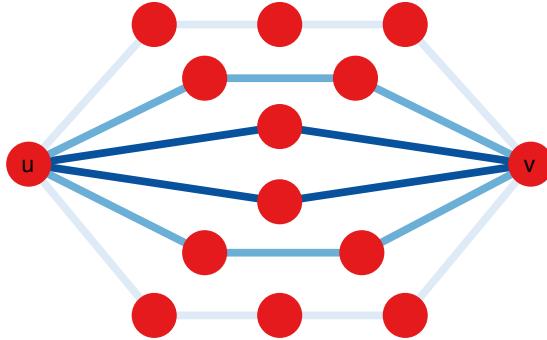


Figure 23.10: An example of the Katz correction for using paths as a score for link prediction. The shade of blue of a path is proportional to its contribution to $\text{score}(u, v)$, with darker paths contributing more.

The easiest way to use hitting time as a score is to simply negate it ($\text{score}(u, v) = -H_{u,v}$) or negate the commute time ($\text{score}(u, v) = -(H_{u,v} + H_{v,u})$). For instance, in Figure 23.11, nodes 2 and 4 are more likely to connect ($\text{score}(2, 4) = -(H_{2,4} + H_{4,2}) = -16$) than nodes 1 and 5 ($\text{score}(1, 5) = -(H_{1,5} + H_{5,1}) = -32$) because they are closer. However, this has the problem of greatly favoring connections to hubs, since their hitting times as destinations are quite low. Thus some authors normalize them by multiplying $H_{u,v}$ with the stationary distribution of the target ($\text{score}(u, v) = -H_{u,v}\pi_v$). Hubs have higher stationary distribution values, thus they are penalized more.

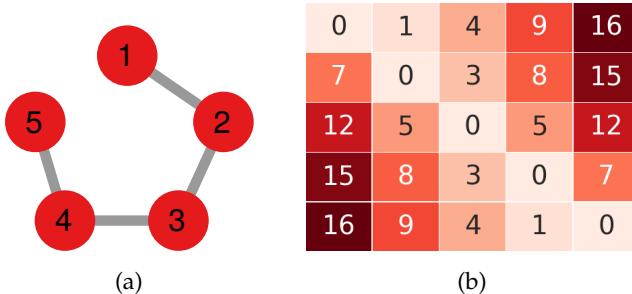


Figure 23.11: (a) A chain of five nodes. (b) The corresponding hitting time matrix H .

Another problem of using $H_{u,v}$ is that the hitting time might increase even if u and v are nearby in the graph, simply because the graph has many vertices and edges that can lead the random walker astray. To counteract this problem, a solution could be allowing the random walker to restart from u . This is practically equivalent to calculate the PageRank, with the difference that you fix the origin of the random walker. For this reason, since your random walker has a root (u), it is usually called “rooted PageRank”.

*SimRank*²¹. As the name suggests, SimRank is based on an idea of node similarity. The more similar two nodes are, the more likely they are to connect. Similarity here is defined recursively: two nodes are similar if they are connected to similar neighbors. This is a definition in line with the philosophy of the regular equivalence we saw in

²¹ Glen Jeh and Jennifer Widom. Scaling personalized web search. In *Proceedings of the 12th international conference on World Wide Web*, pages 271–279. ACM, 2003

Section 15.2. Thus, if we say that $score(u, u) = 1$, we can define all other scores as:

$$score(u, v) = \gamma \frac{\sum_{a \in N_u} \sum_{b \in N_v} score(a, b)}{k_u k_v}.$$

γ is a parameter you can tune. This is surprisingly similar to the hitting time approach. The expected value of a SimRank score is γ^l , where l is the length of an average random walk from u to v .

*Vertex similarity*²². The name of this approach should tip you off regarding its relationship with SimRank. However, it's actually much closer to the Jaccard variant of common neighbor. In fact, the only difference with Jaccard is the denominator. While Jaccard normalizes the number of common neighbors by the total possible number of common neighbors – which is the union of the two neighbor sets – this approach builds an expectation using a random configuration graph as a null model. This is a definition in line with the philosophy of the structural equivalence we saw in Section 15.2.

In practice, $score(u, v) = |N_u \cap N_v| / (k_u k_v)$. This is because two nodes u and v with k_u and k_v neighbors are expected to have $k_u k_v$ common neighbors (multiplied by a constant derived from the average degree which would not make any difference as it is the same for all node pairs in the network).

The same authors in the same paper also make a global variant of this measure. Their inspiration is the Katz link prediction, where they again provide a correction for a random expectation in a random graph with the same degree distribution as G . I won't provide the full derivation, which you can find in the paper, but their score is:

$$score(u, v) = 2|E|\lambda_1 D^{-1} \left(I - \frac{\phi A}{\lambda_1} \right)^{-1} D^{-1}.$$

The elements in this formula are the usual suspects: $|E|$ is the number of edges, λ_1 is the leading eigenvalue of the adjacency matrix A (not the stochastic, as that would be equal to one), D is the degree matrix, and I is the identity matrix. The only odd thing is $0 < \phi < 1$, which is a parameter you can set at will. This is similar to the parameter of Katz: smaller ϕ give more weight to shorter paths.

*Local and superposed random walks*²³. These two methods are a close sibling to the hitting time approach. To determine the similarity between u and v , we place a random walker on u and we calculate the probability it will hit node v . Note that, if we were to do infinite length random walks, this would be the stationary distribution π . This would be bad, as you know that this only depends on the degree

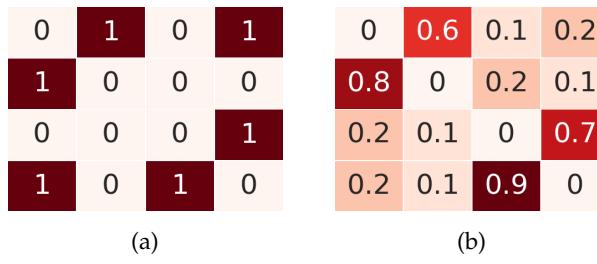
²² Elizabeth A Leicht, Petter Holme, and Mark EJ Newman. Vertex similarity in networks. *Physical Review E*, 73(2):026120, 2006

²³ Weiping Liu and Linyuan Lü. Link prediction based on local random walk. *EPL (Europhysics Letters)*, 89(5):58007, 2010

of v , not on your starting point u . For this reason, the authors limit the length of the random walk, and also add a vector q determining different starting configurations – namely, giving different sources different weights.

To sum up, the local random walk method determines $\text{score}(u, v) = q_u \pi_{u,v} + q_v \pi_{v,u}$. The superposed variant works in the same way, with the difference that the random walker is constantly brought back to its starting point u . This tends to give higher scores to nodes closer in the network.

*Stochastic block models*²⁴. We saw the stochastic block models (SBM) as a way to generate graphs with community partitions (Section 18.2) – and we will see them again as a method to detect communities (Section 35.1). In fact, any link prediction approach, in a sense, is a graph generating model. Given the close relationship of SBMs with community discovery, this class of solutions is particularly related to the Hierarchical Random Graph approach.



²⁴ Roger Guimerà and Marta Sales-Pardo. Missing and spurious interactions and the reconstruction of complex networks. *Proceedings of the National Academy of Sciences*, 106(52):22073–22078, 2009

Figure 23.12: (a) The adjacency matrix of a simple graph. (b) One of the possible connection probability matrices that could generate the graph in (a). Each cell reports the probability of observing the edge.

Suppose you're observing a graph, in the form of its adjacency matrix (Figure 23.12(a)). Given an adjacency matrix, we can infer a matrix of connection probabilities (Figure 23.12(b)), telling us the likelihood of observing each edge. We don't need to know how this works – we'll study this process in details when it comes to use SBMs for community detection – but for now suffice to say we use the Expectation Maximization algorithm²⁵.

This matrix will give you a probability of observing a connection that isn't there (yet). You can use that probability as your $\text{score}(u, v)$ for your link prediction. The cool thing about this approach is that, differently from the ones we saw so far, it can also tell you when an observed link is likely to be spurious, because it is associated with a low probability.

This is a family of solutions, because you can bake in different assumptions on how your stochastic blockmodel works. For instance, you can have mixed-membership ones where nodes are allowed to be part of multiple blocks. Or you can have versions working with multilayer networks.

²⁵ Todd K Moon. The expectation-maximization algorithm. *IEEE Signal processing magazine*, 13(6):47–60, 1996

Probabilistic models^{26,27,28}. In this subsection I group not a single method, but an entire family of approaches to link prediction. They have their differences, but they share a common core. In probabilistic models, you see the graph as a collection of edges and attributes attached to both nodes and edges. The hypothesis is that the presence of an edge is related to the values of the attributes.

In practice, the hypothesis is that there exist a function taking as input the attribute values of each node and edge. This function models the observed graph. Then, depending on the values of the attributes for nodes u and v , the function will output the probability that the u, v edge should appear – and with which attributes.

*Mutual information*²⁹. In Section 3.5 I introduced the concept of mutual information: the amount of information one random variable gives you over another. This can be exploited for link prediction. If you remember how it works, you'll remember that MI allows you to calculate the relationship between two non-numerical vectors, which is not really possible using other correlation measures – not without doing some non-trivial bending of the input. In link prediction, this advantage is crucial: you can define your function as $score(u, v) = MI_{uv}$, where the “events” that allow you to calculate MI_{uv} are the common neighbors between nodes u and v .

*CAR Index*³⁰. In this index you favor pairs of nodes that are part of a local community, i.e. they are embedded in many mutual connections. This is a variant of the idea of common neighbor: each shared connection counts not equally, but proportionally more if it also shares neighbors with u and v . This basic idea can be implemented in multiple ways, depending on which of the traditional link prediction methods we want to extend. For instance, if we extend vanilla common neighbors, you'd say that:

$$score(u, v) = \sum_{z \in N_u \cap N_v} 1 + \frac{|N_u \cap N_v \cap N_z|}{2}.$$

Note how here we simply added a second intersection to the basic common neighbors, the one with the common neighbor z . Figure 23.13 shows an example. Node a in this case contributes much more than node b , because it shares four common neighbors with u and v . Node b , on the other hand, lies outside this local community.

One could use this CAR approach to create a new family of measures. For instance, we can have a CAR version of the resource allocation approach:

$$score(u, v) = \sum_{z \in N_u \cap N_v} \frac{|N_u \cap N_v \cap N_z|}{|N_z|}.$$

²⁶ Nir Friedman, Lise Getoor, Daphne Koller, and Avi Pfeffer. Learning probabilistic relational models. In *IJCAI*, volume 99, pages 1300–1309, 1999

²⁷ David Heckerman, Chris Meek, and Daphne Koller. Probabilistic entity-relationship models, prms, and plate models. *Introduction to statistical relational learning*, pages 201–238, 2007

²⁸ Kai Yu, Wei Chu, Shipeng Yu, Volker Tresp, and Zhao Xu. Stochastic relational models for discriminative link prediction. In *Advances in neural information processing systems*, pages 1553–1560, 2007

²⁹ Fei Tan, Yongxiang Xia, and Boyao Zhu. Link prediction in complex networks: a mutual information perspective. *PloS one*, 9(9):e107056, 2014

³⁰ Carlo Vittorio Cannistraci, Gregorio Alanis-Lobato, and Timothy Ravasi. From link-prediction in brain connectomes and protein interactomes to the local-community-paradigm in complex networks. *Scientific reports*, 3:1613, 2013

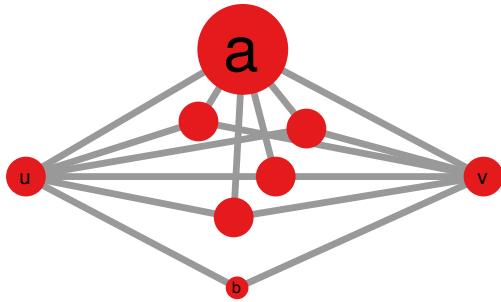


Figure 23.13: Comparing the CAR index contribution to $score(u, v)$ for nodes a and b . Node size is proportional to the contribution.

Here, we normalize by z 's bandwidth.

*Katz Tensor Factorization*³¹. This approach doesn't really add a new idea to link prediction, but it is worthwhile mentioning for a few reasons. It is an application of the Katz criterion we saw earlier. However, it implements it via tensor factorization. If you recall Section 8.1, we could view particular tensors as "3D" matrices. In practice, they are a collection of adjacency matrices. In that section I introduced the idea to represent multilayer networks, where each adjacency is a layer of the network. Here, instead, each layer is a temporal snapshot of the network.

The advantage of this approach is that, like GERM (Section 23.6), one could predict links not necessarily at time $t + 1$, but also at $t + 2$, $t + 3$, etc... This is possible because the adjacency matrix at any time t is simply a slice of the tensor. Thus, there's nothing fundamentally different between predicting a link in the slice $t + 1$ or in the slice $t + 3$ – your precision might be a bit lower because of the strongly sequential nature of link appearance, but it's still possible to provide a good guess.

*Rank aggregation techniques*³². When it comes to link prediction, you can go full meta. And I never drop an opportunity to go full meta. You can take all – yes, I mean all – the methods listed so far. Each method will rank unobserved edges differently. Meaning that they have an ordered list of preferences as to which new link should be the next one observed. You can use a rank aggregation method to create a final list that uses all the information from all the methods. Rank aggregation is the general process of having two ordered lists and producing a single ordered list that is the one agreeing the most with both your inputs.

There is a classical way to solve the issue, which is the Borda's method³³. This is what happens in those electoral systems where citizens will vote not for one candidate, but will rank their top n candidates. Each candidate receives a number of points proportional to its rank, and the candidate with most points win.

³¹ Daniel M Dunlavy, Tamara G Kolda, and Evrim Acar. Temporal link prediction using matrix and tensor factorizations. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 5(2):10, 2011

³² Cynthia Dwork, Ravi Kumar, Moni Naor, and Dandapani Sivakumar. Rank aggregation methods for the web. In *Proceedings of the 10th international conference on World Wide Web*, pages 613–622. ACM, 2001

³³ https://en.wikipedia.org/wiki/Borda_count

A more sophisticated aggregation methods uses the Kendall τ . The Kendall τ counts the number of pairwise disagreements: pairs of edges that have the opposite rankings in the two lists. If in one list u_1, v_1 is ranked higher than u_2, v_2 while the opposite holds in the other list, then you have a disagreement – this is sort of similar to the Spearman rank correlation³⁴.

23.8 Summary

1. In link prediction we want to take an observed network and infer the most likely connections to appear in the future, or the ones that might already be there but for some reason we aren't seeing yet.
2. The most common approach is to compute a score for each pair of unconnected nodes by using some theory about the topology of the network. For instance, we can say that usually triangles close and thus count the number of common neighbors between two nodes.
3. Other approaches model mesoscale structures of the network such as communities, or find overexpressed graph patterns in the network and rank node pairs on whether they are likely to make more of these patterns appear in the network.
4. Many approaches from other branches of network science can be used to predict links, for instance ranking algorithms (Katz), random walk hitting time, stochastic blockmodels, mutual information, etc.
5. Nothing stops you from using all the link prediction methods at once and then aggregate their results. Really, it's a free country.

23.9 Exercises

1. What are the ten most likely edges to appear in the network at <http://www.networkatlas.eu/exercises/23/1/data.txt> according to the preferential attachment index?
2. Compare the top ten edges predicted for the previous question with the ones predicted by the jaccard, Adamic-Adar, and resource allocation indexes.
3. Use the mutual information function from `scikit-learn` to implement a mutual information link predictor. Compare it with the results from the previous questions.

³⁴ Douglas G Bonett and Thomas A Wright. Sample size requirements for estimating pearson, kendall and spearman correlations. *Psychometrika*, 65(1):23–28, 2000

4. Use your code to calculate the hitting time (from exercise 3 of Chapter 11) to implement a hit time link predictor – use the commute time since the network is undirected. Compare it with the results from the previous questions.

24

For Multilayer Graphs

Link prediction takes a distinctive new flavor when your input is a multilayer network. In single layer networks, all you have to do is asking the question: “who will be the next two people to become friends with each other?” The question becomes harder and more interesting in multilayer networks. Here you don’t want to only know *which* two people will connect next, but also *how*. Are they going to be best buds? Work colleagues? Lovers? Enemies? You see that this new dimension adds a lot of spice to the problem. You don’t want to make a friend suggestion on Facebook for two people with a strong connection prediction if you also knew that the connection type between the two would be an enmity link – and researchers in social media studies have looked at this problem¹.

Multilayer link prediction is the topic of this chapter. We start from the simplest case where we have only two layers in the network with a very precise semantic: link prediction in signed networks (Section 24.1). We then move on to the generalized case of an arbitrary number of layers with no clear semantic relationship with each other (Section 24.2).

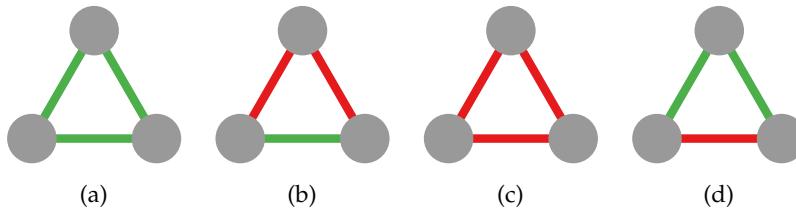
24.1 Signed Networks

Social Balance Theory

There are two reasons why signed networks represent the simplest case of link prediction when your network has multiple different types of connections. First, signed networks are a subtype of multilayer networks with strong constraints on the edges. You can only have two edge types: positive and negative. Moreover, these edge types are exclusive: if you have a positive edge between u and v , you cannot have also a negative one – unless the network is directed and the edge direction flows in the opposite way. This reduces the search space for a link predictor.

¹ Jiliang Tang, Shiyu Chang, Charu Aggarwal, and Huan Liu. Negative link prediction in social media. In *Proceedings of the eighth ACM international conference on web search and data mining*, pages 87–96. ACM, 2015b

The second reason why signed networks are easier to predict is because the positive/negative sign of an edge gives it a precise meaning. We have strong priors as to which structures we can see in a signed network. These are discussed in what we call “Social Balance Theory”^{2,3}. According to the theory, positive and negative relationships are balanced. For instance, if I have two friends, they are more likely to like each other. On the other hand, if I have an enemy, I expect my friends to be enemies of them as well.



Social balance theory looks predominantly at triangles – although there are ways to look at longer cycles⁴. It divides them in two classes: balanced and unbalanced, see Figure 24.1. Balanced triangles can be understood with common sense: the friend of my friend is my friend, the enemy of my friend is my enemy. Unbalanced triangles are relationships that we expect to change in the future. In fact, the prediction is that balanced triangles are overexpressed in real networks over our expectation – and unbalanced triangles are underexpressed –, and that is generally observed.

One note about the all-negative triangle (Figure 24.1(c)): in some views it is not considered unbalanced, and it is in fact more commonly found in real networks than the other unbalanced triangle (Figure 24.1(d)). A typical case of balanced all-negative triangle is campanilism in Tuscany: the worst enemy of a person from Pisa is a person from Livorno. The second worst enemy for both of them is a person from Lucca. And people from Lucca hate indiscriminately both Pisa and Livorno. And this has gone on for centuries. Pretty balanced.

You can calculate a summary statistics telling how much, on average, your whole network is balanced. There are many ways to do this, but I think the most popular one is called *frustration*⁵. In frustration, you count the number of edges whose removal – or negation – would result in a perfectly balanced network. You can normalize this over the total number of edges in the network. Figure 24.2 shows an example network with two unbalanced triangles: (3, 4, 5) and (4, 5, 6). Both triangles would turn balanced if we were to flip the sign of edge (4, 5) – or, alternatively, frustration would dissipate if we were to remove the edge altogether. Thus, the frustration of this graph is 1/9, since it contains 9 edges.

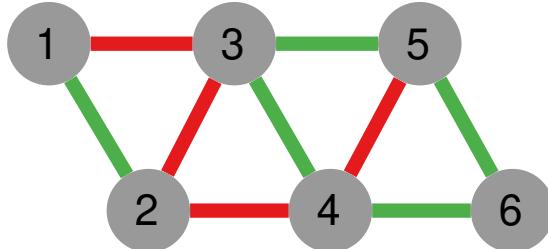
² Fritz Heider. *The psychology of interpersonal relations*. Psychology Press, 2013

³ Tibor Antal, Pavel L Krapivsky, and Sidney Redner. Dynamics of social balance on networks. *Physical Review E*, 72(3):036121, 2005

Figure 24.1: The four possible types of triangles when considering a mutually exclusive pair of positive (in green) and negative (in red) relationships. (a) and (b) are balanced triangles because they have an odd number of positive relationships. (c) and (d) are classically considered unbalanced, although, under certain circumstances, (c) can be considered a balanced or neutral configuration.

⁴ Kai-Yang Chiang, Nagarajan Natarajan, Ambuj Tewari, and Inderjit S Dhillon. Exploiting longer cycles for link prediction in signed networks. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, pages 1157–1162. ACM, 2011

⁵ Frank Harary. On the measurement of structural balance. *Behavioral Science*, 4(4):316–323, 1959



Frustration is a bit computational complex to calculate, but there are heuristics you can use to speed up your calculations⁶. One relatively simple check you can do to figure out whether your signed network is balanced is by looking at the smallest eigenvector of its signed Laplacian – I told you how to calculate the signed Laplacian in Section 8.4. The smallest eigenvector of a balanced graph will be zero. If your signed graph is not balanced, the smallest eigenvector of its signed Laplacian will be higher than zero. In fact, how much higher than zero it is puts an upper bound to the frustration value⁷ – that is to say, frustration can only be lower than or equal to the smallest eigenvector of the signed Laplacian. Figure 24.3 shows you an example of balanced and an example of unbalanced graph with their eigenvectors.

Figure 24.2: A graph with two unbalanced triangles, the ones including edge $(4, 5)$.

⁶ Samin Aref and Mark C Wilson. Balance and frustration in signed networks. *Journal of Complex Networks*, 7(2):163–189, 2019

⁷ Francesco Belardo. Balancedness and the least eigenvalue of laplacian of signed graphs. *Linear Algebra and its Applications*, 446:133–147, 2014.

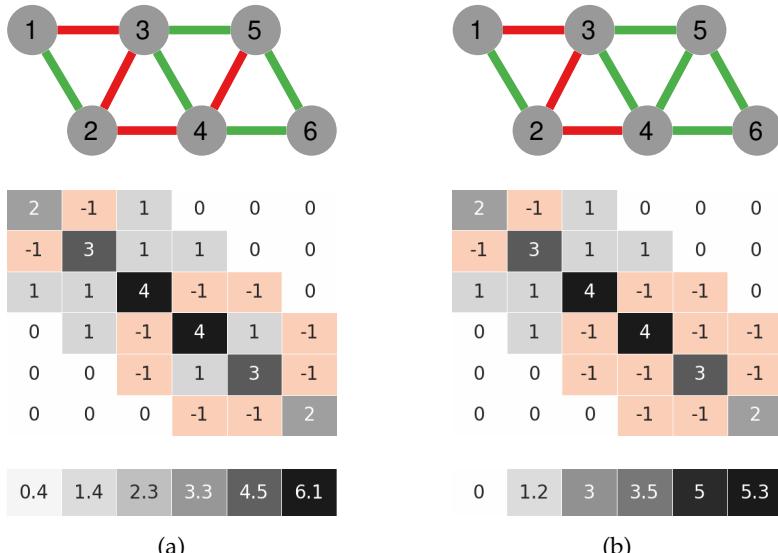


Figure 24.3: From top to bottom: signed graph, its signed Laplacian, the sorted eigenvalues of the signed Laplacian. (a) Unbalanced graph. (b) Balanced graph.

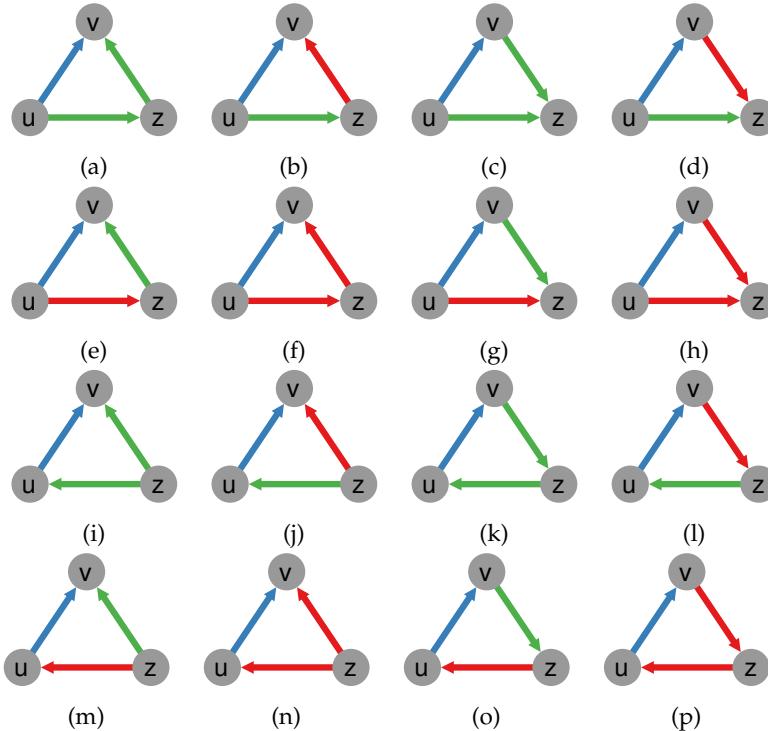
This relates to link prediction when we consider evolving signed networks. If we find a configuration with three nodes connected by two positive edges, it is overwhelmingly more likely that, in the future, the triangle will close with a positive relationship (Figure 24.1(a)) rather than with a negative one (Figure 24.1(d)). On the other hand, if we find a positive and a negative relationship, we expect the

triangle to close with a negative edge (Figure 24.1(b)). The case with an initial condition of two negative edges is more difficult to close, but we prefer to close it with a positive edge (Figure 24.1(b)) than with a negative one (Figure 24.1(c)).

So you see that you can perform signed link prediction by first predicting the pair of nodes that will connect, calculating a $score(u, v)$ with any of the methods presented in the previous chapter. Then you will decide the sign of the link, by using social balance theory.

Social Status Theory

There is a competing theory to social balance, which is the status theory⁸. This arises from a different interpretation of the sign. A positive sign in a social setting might mean that the user originating the link feels to be lower status than – and thus giving social credit to – whomever receives the link. Conversely, a negative link is a way for a higher status node to shoot down a lower status one. Note that here we started talking about the direction of an edge, meaning that we have more than four types of triangles. In fact, we have 32.



⁸ Jure Leskovec, Daniel Huttenlocher, and Jon Kleinberg. Signed networks in social media. In *Proceedings of the SIGCHI conference on human factors in computing systems*, pages 1361–1370. ACM, 2010a

Figure 24.4: The 16 templates of directed signed triangles in social status networks. The color of the edge determines its status: green = positive, red = negative, blue = the edge we are trying to predict – can be either positive or negative.

Figure 24.4 shows the 16 main configurations of these triangles. The closing edge connecting u to v can be either positive or negative, generating 32 final possible configurations. Status theory generates predictions that are more sophisticated and – sometimes – less immediately obvious. Some cases are easy to parse. For instance, consider

Figure 24.4(a). The objective is to predict the sign of the (u, v) edge. In the example, u endorses z as higher status. z endorses v . If v is on a higher level than z , and z is on a higher level than u , then it's easy to see how v is also on a higher level than u . Thus the edge will be of a positive sign. In fact, this specific configuration is grossly over represented in real world data.

The situation is not as obvious for other triangles. For instance, the one in Figure 24.4(i). Here we have the z node endorsing both u and v . We don't really know anything about their relative level, only that they are both on a higher standing with respect to z . The paper presenting the theory makes a subtle case. The edge connecting u to v is more likely to be positive than the generative baseline on u , but less likely to be positive than the receiving baseline of v . So, suppose that 50% of edges originating from u are positive, while 80% of the links v receives are positive. The presence of a triangle like the one in Figure 24.4(i) would tell us that the probability of connecting u to v with a positive link is higher than 50%, but lower than 80%.

Given this sophistication, and the fact that social status works with more information than social balance – namely the edge's direction $-$, it is no wonder that there are cases in which social status vastly outperforms social balance. For instance, the original authors apply social status to a network of votes in Wikipedia. Here the nodes are users, who are connected during voting sessions to elect a new admin. The admin receives the links, positive if the originating user voted in favor, negative if they voted against. Triangles in this network connect with the patterns predicted by social status theory.

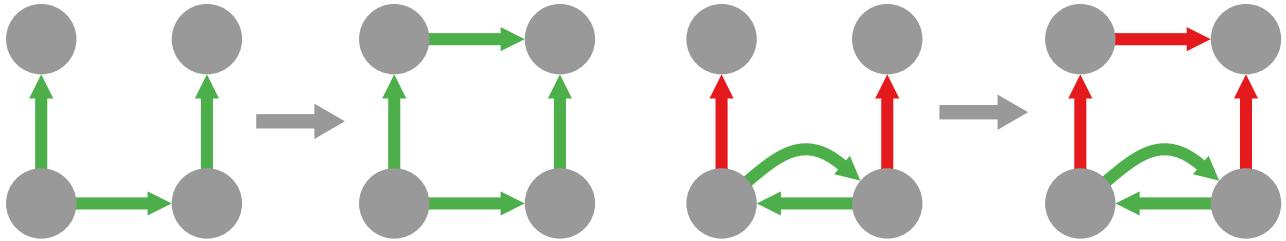
Atheoretical Sign Prediction

Of course, calling onto us the powers we unlocked in Section 23.6, we can apply a strategy similar to GERM to extract graph association rules also in this scenario⁹. Figure 24.5 shows two possible association rules extracted from two different datasets. Looking at the figure, two advantages for this strategy emerge.

First, using a variation of GERM we free ourselves from the tyranny of the triangles. We can look at an arbitrary set of rules, not necessarily involving three nodes and triadic closure, which may not apply for all networks.

Second, what social balance and status have in common is that they will make the same prediction no matter the network you're going to have as input. They establish universal laws that might apply in general, but overlook specific laws that might apply for the phenomenon we're observing right now. For instance, voting in Wikipedia might be very different from trusting someone's opinion in

⁹ Giacomo Bachi, Michele Coscia, Anna Monreale, and Fosca Giannotti. Classifying trust/distrust relationships in online social networks. In *2012 International Conference on Privacy, Security, Risk and Trust and 2012 International Conference on Social Computing*, pages 552–557. IEEE, 2012



a product review database. Consider the rule on the right in Figure 24.5. That is a voting pattern in Wikipedia. It makes sense from a social status point of view: the node receiving the last negative link should expect to receive it, because it already received one, so it received a signal of being of low status.

But that rule makes little sense when moving to the scenario of trusting reviews. What the negative link means here is that the originator of the edge doesn't trust the recipient of the edge. However, we already know that the node originating the last link disagrees with the bottom two nodes, who trust each other. So we would expect it to trust – to send a positive rather than a negative link – to the node in the top right. In fact, while the pattern is widely popular in the Wikipedia network, it doesn't appear in the social review dataset.

24.2 Generalized Multilayer Link Prediction

So far we have only considered the case of two possible edge types. Moreover, these two types have a clear semantic: one type is positive, the other is negative. Both assumptions make the link prediction problem easier: there are few degrees of freedom and we move in a space constrained by strong priors. It is now time to drop these assumptions and face the full problem of multilayer link prediction as the big boys we are.

Generalized multilayer link prediction is the task of estimating the likelihood of observing a new link in the network, given the two nodes we want to connect and the layer we want to connect them through. Nodes u and v might be very likely to connect in the immediate future, but they might do so in any, some, or even just a single layer. Thus, we extend our score function to take the layer as an input: from $score(u, v)$ to $score(u, v, l)$.

Layer Independence

As you might expect, there are tons of ways to face this problem. The most trivial way to go about it is to apply any of the single layer

Figure 24.5: Two possible graph association rules extracted from a directed signed network.

link prediction methods from Chapter 23 to each layer separately. Then, you can create a single ranking table by merging all these predictions¹⁰.

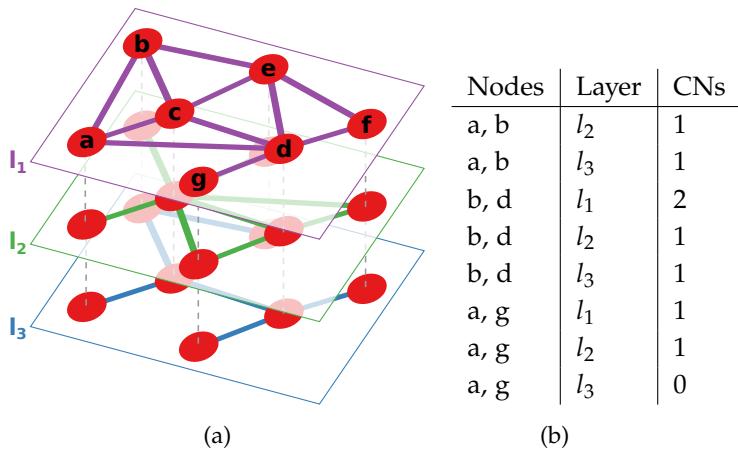


Figure 24.6 depicts an example for this process. Note that here I use a rather trivial approach to aggregate, by comparing directly the various scores. One could also apply to this problem the rank aggregation measures presented in the previous chapter. In this way, you could also aggregate different scores using different criteria: common neighbors, preferential attachment, and so on.

This is practically a baseline: it will work as long as we have an assumption of independence between the layers. As soon as having a link in a layer changes the likelihood of connecting into another layer, we expect to grossly underperform.

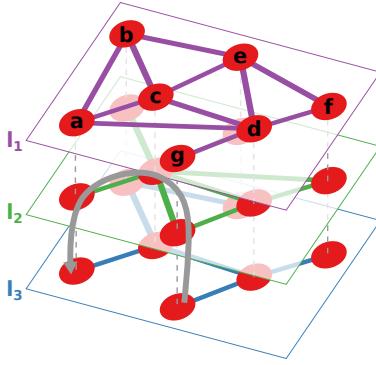
Blending Layers

A slightly more sophisticated alternative is to consider the multilayer network as a single structure and perform the estimations on it. For instance, consider the hitting time method. This is based on the estimation of the number of steps required for a random walker starting on u to visit v . We can allow the random walker to, at any time, use the inter layer coupling links exactly as if they were normal edges in the network. At that point, a random walker starting from u in layer l_1 can and will visit node v in layer l_2 . The creation of our connection likelihood score is thus well defined for multilayer networks. Figure 24.7 depicts an example for this process.

These paths crossing layers are often called meta-paths. The information from these meta-paths can be used directly as we just saw, informing a multilayer hitting time. Or we can feed them to a classifier, which is trying to put potential edges in one of two cate-

¹⁰ Manisha Pujari and Rushed Kanawati. Link prediction in multiplex networks. *NHM*, 10(1):17–35, 2015

Figure 24.6: The easiest way to perform multilayer link prediction. Given the input network, perform single layer link prediction on each of the layer separately. In this case, we count the number of common neighbors between pairs of nodes. We then predict the one with the overall highest score.



gories: future existing and future non-existing links. Any classifier can perform this job once you collect the multilayer information from the meta-path: naive Bayes, support vector machines (SVM), and others¹¹.

Other extensions to handle multilayer networks have been proposed¹². These studies show that multilayer link prediction is indeed an interesting task, as there is a correlation between the neighborhood of the same nodes in different layer. The classical case involves the prediction of links in a social media platform using information about the two users coming from a different platforms¹³. Such layer-layer correlations are not limited to social media, but can also be found in infrastructure networks¹⁴.

Multilayer Scores

The last mentioned strategy is better, but it still doesn't consider all the wealth of information a multilayer network can give you. To see why, let's dust off the concept of layer relevance we introduced in Section 9.1. That is a way to tell you that a node u has a strong tendency of connecting through a specific layer. If a layer exclusively hosts many neighbors of u , that might mean that it is its preferred channel of connection.

This suggests that other naive ways to estimate node-node similarity for our score should be re-weighted using layer relevance¹⁵. Consider Figure 24.8. We see that the two nodes have many common neighbors in the blue layer. They only have one common neighbor in the red layer. However the blue layer, for both nodes, has a very low exclusive layer relevance. There is no neighbor that we can reach using exclusively blue edges. In fact, in this case, the exclusive layer relevance is zero.

The opposite holds for the layer represented by red edges. There are many neighbors for which red links are the only possible choice. In this particular case, we might rank the red layer as more likely

Figure 24.7: A slightly more sophisticated way to perform multilayer link prediction. Given the input network, perform the link prediction procedure on the full structure. In this case, the gray arrow simulates a random walker going from node g in layer l_3 to node a in the same layer, passing through node c in layer l_2 . The multilayer random walker contributes to the $\text{score}(g, a, l_3)$.

¹¹ Mahdi Jalili, Yasin Orouskhani, Milad Asgari, Nazanin Alipourfard, and Matjaž Perc. Link prediction in multiplex online social networks. *Royal Society open science*, 4(2):160863, 2017

¹² Darcy Davis, Ryan Lichtenwalter, and Nitesh V Chawla. Multi-relational link prediction in heterogeneous information networks. In *ASONAM*, pages 281–288. IEEE, 2011

¹³ Desislava Hristova, Anastasios Noulas, Chloë Brown, Mirco Musolesi, and Cecilia Mascolo. A multilayer approach to multiplexity and link prediction in online geo-social networks. *EPJ Data Science*, 5(1):24, 2016

¹⁴ Kaj-Kolja Kleineberg, Marián Boguná, M Ángeles Serrano, and Fragkiskos Papadopoulos. Hidden geometric correlations in real multiplex networks. *Nature Physics*, 12(11):1076, 2016

¹⁵ Giulio Rossetti, Michele Berlingario, and Fosca Giannotti. Scalable link prediction on multidimensional networks. In *2011 IEEE 11th International Conference on Data Mining Workshops*, pages 979–986. IEEE, 2011

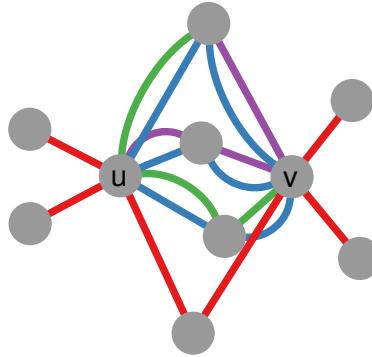


Figure 24.8: The multilayer neighborhood of nodes u and v . Edge color indicates the layer to which the connection belongs.

to host a connection for nodes u and v . In practice, this boils down to multiplying the layer relevance to the common neighbor score. Such weighting schema can be applied to most of the link prediction strategies we saw so far.

A related approach tries to estimate not whether a link will exist in the future, but its strength. If we have an unweighted multilayer network, we might still be able to estimate how strong a connection is. By looking at the various layers connecting two nodes, one could estimate such tie strength¹⁶.

Another approach to multilayer link prediction is the usage of tensor factorization. We briefly mentioned tensor factorization at the end of the previous chapter, for single layer link prediction. In that case, the third dimension of our tensor was representing time. In this case, we can apply the same technique by changing the meaning of this third dimension. Rather than using it to represent time, we can use it to represent the layer in which the edge appear. The same technique can now be applied, to discover in which layer new edges are likely to pop up.

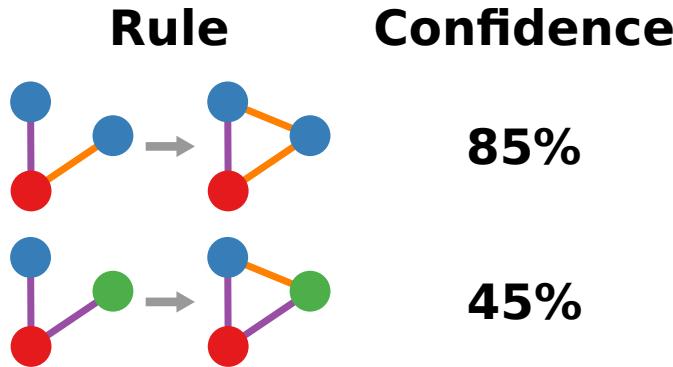
And, since we're mentioning flexible methods that can be applied in multiple scenarios, why don't we dust off GERM again? We already saw how graph association rules can be extracted in signed networks without batting an eye. There is, in principle, no issue in extending the algorithm to deal with multilayer networks¹⁷ – as long as you can properly and efficiently solve the graph isomorphism problem (Section 41.3) for labeled multigraphs.

Figure 24.9 shows multilayer association rules in all their glory. In this case I encode also node attributes – because why not? – rendering the rules extracted by multilayer GERM extremely multifaceted. By collecting all the rules I showed so far in this book part, you realize that there are really a lot of ways to close a triangle in complex networks!

By using the edge labels to represent the layer in which an edge appears, we lose one of the powers of GERM. Namely, we are not

¹⁶ Luca Pappalardo, Giulio Rossetti, and Dino Pedreschi. "how well do we know each other?" detecting tie strength in multidimensional social networks. In *2012 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, pages 1040–1045. IEEE, 2012

¹⁷ Michele Coscia and Michael Szell. Multiplex graph association rules for link prediction. *arXiv preprint arXiv:2008.08351*, 2020



able to make predictions at time steps farther than one. Remember that, with GERM, we could predict that a link will appear at time $t + 2$. This is not the case any more here, because the way we were able to do that was by encoding the edge arrival time in its label. But here we're using the edge label to indicate the layer in which it appears. This is an acceptable price to pay if we're able to perform multilayer link prediction.

Finally, as in the single layer case, there are some promising approaches using multilayer network embedding to predict links in multilayer networks, both in the signed case¹⁸ and in the multilayer proper case^{19,20,21}.

Heterogeneous Networks

In computer science, specifically in data mining and machine learning, multilayer networks are often called “heterogeneous” networks, because they have edges of different, heterogeneous, types. Heterogeneous link prediction is one of the main tasks tackled in this subfield. This is actually where metapaths were firstly developed²². Figure 24.10 shows examples of possible metapaths in a co-authorship network. These metapaths form the input of a classifier, which will then spit out the most likely new metapaths involving specific nodes.

Other common approaches use a ranking factor graph model²³, which searches for common general patterns shared by the various layers of the network; or consider link prediction as a matching problem²⁴.

By the way, the converse of what I said about GERM and tensor factorization applies also to heterogeneous link predictions. There is research showing how you can use this class of approaches to predict *when* an edge will appear, rather than its type²⁵.

I should also mention that link prediction, community discovery, and generating synthetic networks are sides of the same weirdly triangular coin. This holds also for multilayer networks. There are

Figure 24.9: Two possible graph association rules extracted with the multilayer version of GERM. Node color represents the node's label, while edge color represents the edge's layer.

¹⁸ Suhang Wang, Jiliang Tang, Charu Aggarwal, Yi Chang, and Huan Liu. Signed network embedding in social media. In *Proceedings of the 2017 SIAM international conference on data mining*, pages 327–335. SIAM, 2017b

¹⁹ Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In *Advances in neural information processing systems*, pages 2787–2795, 2013

²⁰ Hongming Zhang, Liwei Qiu, Lingling Yi, and Yangqiu Song. Scalable multiplex network embedding. In *IJCAI*, volume 18, pages 3082–3088, 2018a

²¹ Ryuta Matsuno and Tsuyoshi Murata. Mell: effective embedding method for multiplex networks. In *Companion Proceedings of the The Web Conference 2018*, pages 1261–1268, 2018

²² Yizhou Sun, Rick Barber, Manish Gupta, Charu C Aggarwal, and Jiawei Han. Co-author relationship prediction in heterogeneous bibliographic networks. In *2011 International Conference on Advances in Social Networks Analysis and Mining*, pages 121–128. IEEE, 2011

²³ Yuxiao Dong, Jie Tang, Sen Wu, Jilei Tian, Nitesh V Chawla, Jinghai Rao, and Huanhuan Cao. Link prediction and recommendation across heterogeneous social networks. In *2012 IEEE 12th International conference on data mining*, pages 181–190. IEEE, 2012

²⁴ Xiangnan Kong, Jiawei Zhang, and Philip S Yu. Inferring anchor links across multiple heterogeneous social networks. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*, pages 179–188. ACM, 2013

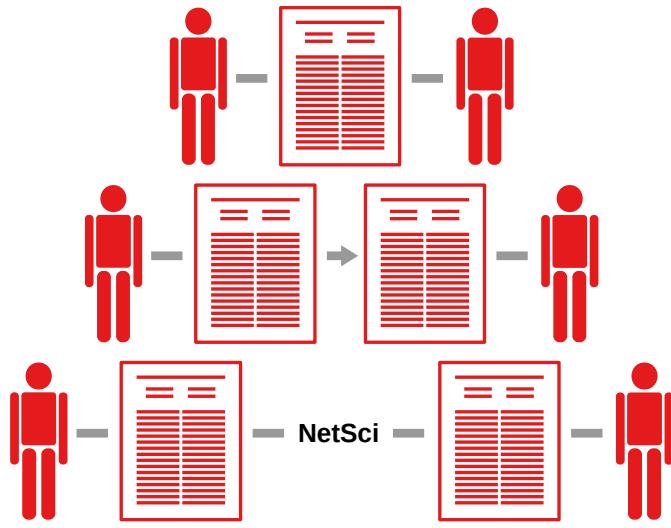


Figure 24.10: Some examples of metapaths in a heterogeneous network with multiple node types, in a scientific publication scenario. From top to bottom, connecting authors because: they co-author a paper (both nodes of type author are connected to the same node of type paper); they cite each other (a node of type author connects to a node of type paper citing another paper-type node); or they publish in the same venue.

efforts to create models generating multilayer networks than can then be applied to predict new links on already existing real-world multilayer networks^{26,27}.

In this chapter, as in all chapters of this book, I presented only the most prominent methods to tackle the issue at hand, and the ones I'm most familiar with. The study of a deeper review work²⁸ is necessary if you want to make a living off solving multilayer link prediction.

24.3 Summary

1. In multilayer link prediction, besides predicting the appearance of a new edge, you also need to guess in which layer the new edge will appear. It's not only about *whether* two nodes will connect, it's also about *how* they will connect.
2. A simplified version of multilayer link prediction involve signed networks. In this case, real world networks have a preference for balanced structures, which you can use to predict the sign of the relationship.
3. An alternative approach is by using status theory. Whether you should use social balance or social status depends on what your network represents, for instance trust would follow balance, while voting would follow status.
4. For generalized multilayer link prediction the common approach is to create multilayer generalizations of single layer predictors, with some strategy to aggregate multilayer information.

²⁵ Yizhou Sun, Jiawei Han, Charu C Aggarwal, and Nitesh V Chawla. When will it happen?: relationship prediction in heterogeneous information networks. In *Proceedings of the fifth ACM international conference on Web search and data mining*, pages 663–672. ACM, 2012

²⁶ Caterina De Bacco, Eleanor A Power, Daniel B Larremore, and Christopher Moore. Community detection, link prediction, and layer interdependence in multilayer networks. *Physical Review E*, 95(4):042317, 2017

²⁷ A. Roxana Pamfil, Sam D. Howison, and Mason A. Porter. Edge correlations in multilayer networks. *arXiv preprint arXiv:1908.03875*, 2019

²⁸ Yizhou Sun and Jiawei Han. Mining heterogeneous information networks: a structural analysis approach. *Acm Sigkdd Explorations Newsletter*, 14(2):20–28, 2013

5. Other approaches rely on multilayer extensions of graph mining and on the use of metapaths: paths connecting nodes across layers.

24.4 Exercises

1. You're given the undirected signed network at <http://www.networkatlas.eu/exercises/24/1/data.txt>. Count the number of triangles of the four possible types.
2. You're given the directed signed network at <http://www.networkatlas.eu/exercises/24/2/data.txt>. Does this network follow social balance or social status? (Consider only reciprocal edges. For social balance, the reciprocal edges should have the same sign. For social status they should have opposite signs)
3. Consider the multilayer network at <http://www.networkatlas.eu/exercises/24/3/data.txt>. Calculate the Pearson correlation between layers (each layer is a vector with an entry per edge. The entry is 1 if the edge is present in the layer, 0 otherwise). What does this tell you about multilayer link prediction? Should you assume layers are independent and therefore apply a single layer link prediction to each layer?

25

Designing an Experiment

As the name of the problem suggests, link prediction is fundamentally a task that involves making claims about the future. Evaluating the performance of an oracle in getting things right is harder than it might seem. There are surprising ways to get it wrong. Luckily, making predictions is the bread and butter of machine learning. Thus we have a large set of best practices we can follow. In Chapter 4 we saw the general architecture of a machine learning framework. This chapter is a crash course on how to adapt that general pipeline to the specific problems of link prediction. This is mostly taken from the literature¹, which you should check out to get a deeper view on the problem.

The chapter is divided in two parts. First, we have to figure out how to perform the test (Section 25.1). Since link prediction is about the future, one option would be to just wait until new data comes in. This is often not ideal, because you don't really know when you'll get new information, and you want to publish your paper *right now*. So you have to work with the data you have. Once you make your prediction, you have to then evaluate how well you perform (Section 25.2).

25.1 Train/Test Sets

The Basics

When it comes to evaluate your prediction algorithm, you have to distinguish between the training and the test datasets. The training dataset is what your model uses to learn the patterns it is supposed to predict. For instance, if you're doing a common neighbor link predictor, the training dataset is what you use to count the number of shared connections between two nodes. Once you're done examining the input data, you have generated the results of the $score(u, v)$ function for all possible pairs of u, v inputs.

¹ Yang Yang, Ryan N Lichtenwalter, and Nitesh V Chawla. Evaluating link prediction methods. *Knowledge and Information Systems*, 45(3):751–782, 2015

The test dataset is a set of examples used to assess performance. When your model is done learning on the training dataset, it is unleashed on the test dataset and will start making predictions. Every time it gets it right you increase its performance, every time it gets it wrong you decrease it.

Figure 25.1 shows an example of the difference between the two sets. Figure 25.1(a) is the training dataset: it contains all the information we can use to infer our scores. Figure 25.1(b) are the scores we calculate based on the data from Figure 25.1(a). Specifically, for each pair of nodes I calculate the number of common neighbors shared by the two nodes. Figure 25.1(c) shows the test set in blue. These are the actual new edges that appeared in the network. I include the training set in gray because it makes it easier to put the new edges into context – besides, when performing a prediction you need to make sure you remember what was in the training set and discard any prediction you might have done about those edges. For instance, in this case, we need to throw away the (1,2) edge prediction.

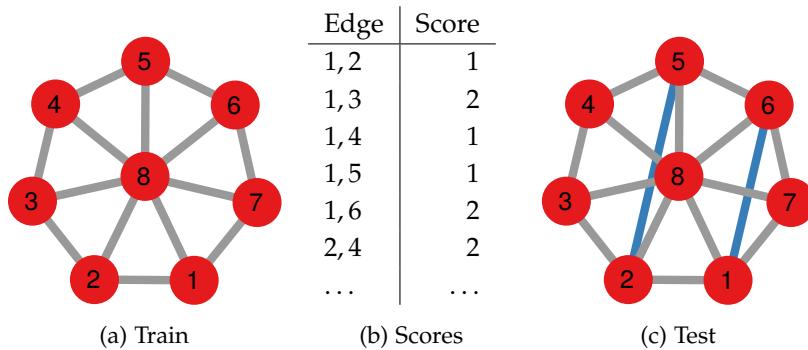


Figure 25.1: An example of train and test sets for a network. The information (a) we use to build the score table (b), using the common neighbor approach. I highlight the test edges (c) in blue.

In machine learning there is also what you'd call a "validation" set (Section 4.1), but there is nothing special about the validation set for link prediction. Just like in any other machine learning scenario, if you worry about overfitting in your link prediction, you should have a validation set. Since there is nothing special about link prediction and validation sets, I simply refer you back to Chapter 4.

The fundamental tenet of machine learning also holds for link prediction: you can never ever ever use the data that trained your model to test it. In other words, training and test sets have to be *disjoint*. If you test your method on the same data that trained it, you're going to grossly overestimate your actual performance in the real world. You have only learned about your training set and nothing about the general forces that shaped it the way it is.

What this means in link prediction is that you cannot claim to have predicted a link that was already in your data. You have to focus only on those pairs of nodes that were not connected in the training

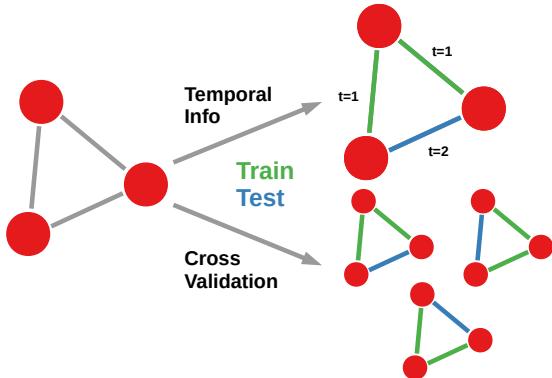


Figure 25.2: Two approaches to build your train and test sets for link prediction. On the left you have the input data. On the right, the partition of links into the two sets: train (green) and test (blue).

set. That is why in Figure 25.1(c) the edges that were already in the training set are gray rather than blue: we won't make predictions on those, because we already know they exist.

So now the problem is: how do you do that? If you have a network, how do you divide it into training and test sets?

You have two options, as Figure 25.2 shows. If you have temporal information on your edges you can use earlier edges to predict the later ones. Meaning that your train set only contains links up to time t , and the test set only contains links from time $t + 1$ on. If you don't have the luxury of time data, you have to do k-fold cross validation: divide your dataset in a train and test set (say 90% of edges in train and 10% in test) and then perform multiple runs of train-test by rotating the test set so that each edge appears in it at least once².

Specific Issues

Link prediction comes with a few peculiarities that might not be a problem in other machine learning tasks. I focus on two: size of the search space and sparsity of positives.

The first refers to the fact that, as we saw in Section 12.1, real networks are sparse. I made the example of the Internet backbone: with its 192,244 nodes, the number of possible edges is $|V|(|V| - 1)/2 = 18,478,781,646$. However, it only contains 609,066 actual edges. This means that, if you were to use its current state as a train set, the score function would have to compute a result for $18,478,781,646 - 609,066 = 18,478,172,580$ potential edges. That is an unreasonable burden, both for computation time and memory storage.

For this reason, when you perform link prediction you will often sample your outputs. You will not calculate $score(u, v)$ for every possible u, v pair, but you will sample the pairs according to some expectation criterion. Such criterion can be as hard to pin down as

² Ron Kohavi et al. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Ijcai*, volume 14, pages 1137–1145. Montreal, Canada, 1995

the link prediction problem itself.

The second problem is intertwined with the first. Suppose that the Internet backbone adds edges at a 5% rate per time step. That means that, if at time t you had 609,066 edges, at time $t + 1$ you will observe $609,066 \times .05 \sim 30,453$ new edges. As we just saw, the number of potential edges is just above 18B. Putting these two facts together lets us reach an absurd conclusion: we can build a link prediction method that will tell us that no new link will ever appear. If we do so, we would be right 99.999% of the times. We would make 18B correct predictions – no edge – and we would get it wrong only 30k times. The accuracy of the “always negative” predictor in Figure 25.3 is $\sim 85\%$: not bad!

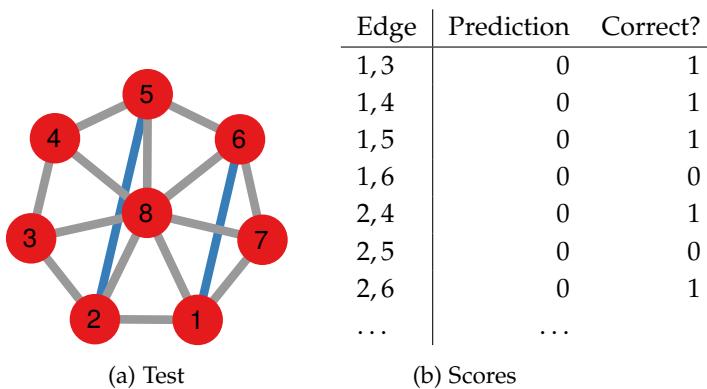


Figure 25.3: Estimating the performance of the “always negative” predictor on our test set.

However that’s... kind of not the point? We’re in this business because we want to predict new links. Returning a negative prediction for all possible cases is not helpful. The usual fix for this problem is building your test set in a balanced way^{3/4}. Rather than asking about all possible new edges, you create a smaller test set. Half of the edges in the test set is an actual new edge, and then you sample an equal number of non-edges. This would make our Internet test set containing 60k edges, not 18B. We called this sampling technique “negative sampling” in Section 4.4.

25.2 Evaluating

Let’s assume that we have competently built our training and test set. We made our model learn on the former. We now have two things: prediction – the result of the model – and reality – the test set. We want to know how much these two sets overlap. Since you know what edges are in the test set, this is a supervised learning task (Section 4.1) and we can focus on the loss/quality functions (Section 4.3) specific for this scenario. Even more narrowly, link prediction is a binary task: you give a yes/no answer that is either completely

³ Ryan N Lichtenwalter, Jake T Lussier, and Nitesh V Chawla. New perspectives and methods in link prediction. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 243–252, 2010

⁴ Ryan Lichtenwalter and Nitesh V Chawla. Link prediction: fair and effective evaluation. In *2012 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, pages 376–383. IEEE, 2012

correct or wrong. So there are four possible cases:

- True Positives (TP): you predict a link that really appeared;
- False Positives (FP): you predict a link that didn't appear;
- True Negatives (TN): you correctly didn't predict a link that, in fact, didn't appear;
- False Negatives (FN): you didn't predict a link that appeared.

These are simple counts on which we can build several quality measures. Two basic combinations of these counts are the True Positive Rate (TPR) and False Positive Rate (FPR). TPR – also known as sensitivity or recall – is the ratio between true positives and all positives: $TPR = TP / (TP + FN)$. It tells you how many times you got it right over the maximum possible number of times you could. Or, what's the share of correct results you found.

FPR is defined similarly: $FPR = FP / (FP + TN)$. This is the share of your wrong answers over all the possible instances of a negative prediction.

Confusion Matrix

Humans like single numbers, because seeing a number going up tingles our pleasure centers (wait, what? You don't feel inexplicable arousal while maximizing scores? I question whether you're in the right line of work...). However, we should beware of what we call "fixed threshold metrics", i.e. everything that boils down a complex phenomenon to a single number. Usually, to reduce everything to a single measure you have to make a number of assumptions and simplifications that may warp your perception of performance.

That is why one of the first thing you should look at is a confusion matrix. A confusion matrix is simply a grid of four cells, putting the four counts I just introduced in a nice pattern⁵. You can see an example in Figure 25.4. Confusion matrices are nice because they don't attempt to reduce complexity, but at the same time you see information in an easy-to-parse pattern.

By looking at two confusion matrices you can say surprisingly sophisticated things about two different methods. The one in Figure 25.5(a) does a better job in making sure a positive prediction really corresponds to a new link: there are very few false positives (one) compared to the true positives (15). The one in Figure 25.5(b) minimizes the number of false negatives, with the downside of having a lot of false positives.

⁵ Stephen V Stehman. Selecting and interpreting measures of thematic classification accuracy. *Remote sensing of Environment*, 62(1):77–89, 1997

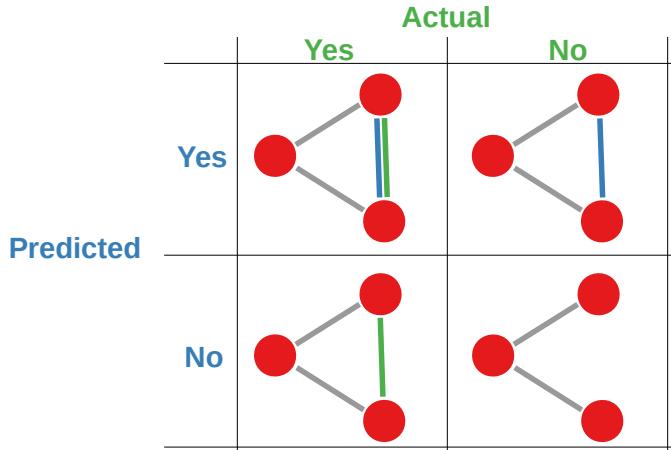


Figure 25.4: The schema of a confusion matrix for link prediction. From the top-left corner, clockwise: true positives, false positives, true negatives, false negatives.

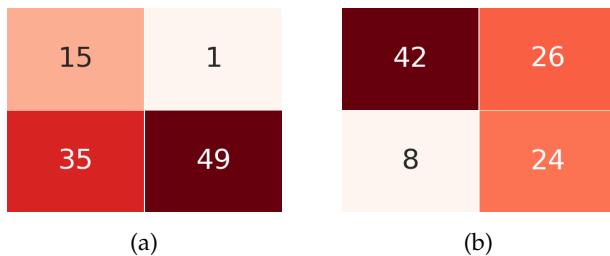


Figure 25.5: Two distinct confusion matrices for different predictors.

By combining the cells of a confusion matrix, you can easily derive measures like TPR or FPR, or many others. They are simple operations on the rows and columns.

If you didn't balance your test set, the confusion matrix can end up being irrelevant, as the vast majority of your observations will end up in the true negative cell, obliterating all the rest.

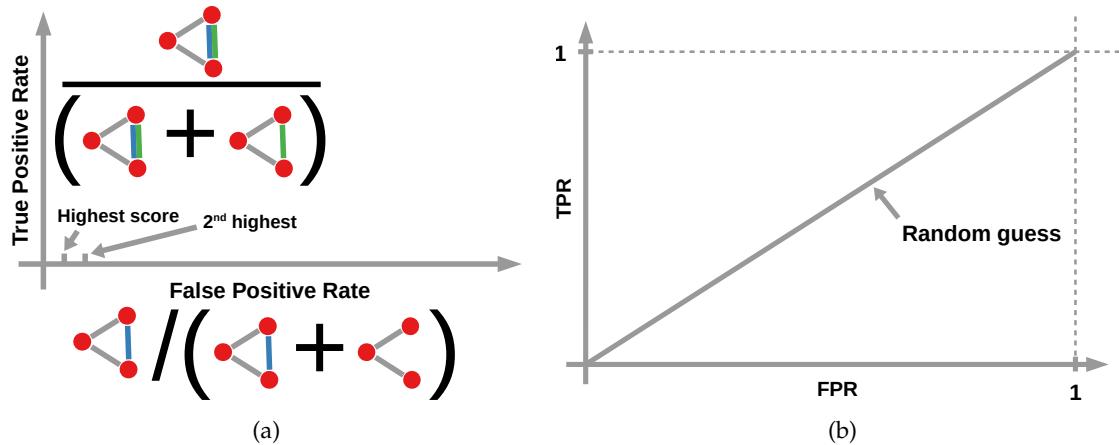
Another disadvantage of the confusion matrix is that you have to pick a threshold in your score. In other words, you predict the appearance of a link if it obtains a score higher than the specific threshold, otherwise you don't. This is in itself a problematic choice, thus it is common to show the evolution of your accuracy as you change that threshold. For high values of the threshold you only report high confidence predictions, which become less and less confident as you decrease the threshold. This is the topic explored in the rest of the chapter.

ROC Curves & AUC

The classic evaluation instrument for classification tasks is the Receiver Operating Characteristic (ROC) curve. This is a plot, with the false positive rate on the x-axis and the true positive rate on the y-axis (see Figure 25.6(a))^{6/7}. We sort all our predictions by their score

⁶James A Hanley and Barbara J McNeil. The meaning and use of the area under a receiver operating characteristic (roc) curve. *Radiology*, 143(1):29-36, 1982

⁷ Tom Fawcett. An introduction to roc analysis. *Pattern recognition letters*, 27(8): 861–874, 2006



such that we look at the highest scores first. Then we keep track of the evolution of TPR and FPR.

In a ROC curve, the 45 degree line corresponds to the random guess (Figure 25.6(b)). Suppose 80% of possible links did not appear and 20% did, and there are a total of 20 new links. If we make ten random guesses, we'll get eight false positives and two true positives. The two true positives represent 10% of all the positives, so $\text{TPR} = 2/20 = 0.1$. On the other hand, we know that there are 80 negatives. Since we got eight false positives, $\text{FPR} = 8/80 = 0.1$. This shows that FPR and TPR grow at the same rate for a random predictor.

What we want to see is that our best guesses are more likely to be true positives, and thus contribute to the y-axis more than they do to the x-axis. Just like in the confusion matrix, there are multiple ways for this to happen. We can be very precise at high scores, or at all scores on average. The two classifiers in Figure 25.7 will be used in different scenarios with different requirements.

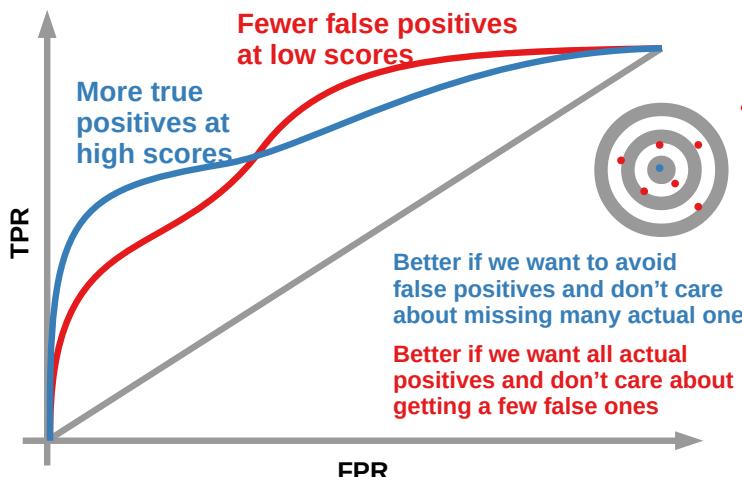


Figure 25.6: Schema of ROC curves.

Figure 25.7: An example of ROC curves. The gray line corresponds to random guesses. The blue and red lines correspond to two different predictors, with different behaviors at different score levels.

ROC curves are great – you might even say that they ROC – but, at the end of the day, you might want to know which of the two classifiers is better on average. ROC curves can be reduced to a single number, a fixed threshold metric. Since we just said that the higher the line on the ROC plot the better, one could calculate the Area Under the Curve (AUC). The more area under that curve, the better your classifier is, because for each corresponding FPR value, your TPR is higher – thus encompassing more area.

You don't need to know calculus to estimate the area under the curve, because it's such a standard metric that any machine learning package will output it for you. The AUC is 0.5 for the random guess: that's the area under the 45 degree line. An AUC of 1 – which you'll never see and, if you do, it means you did something wrong – means a perfect classifier.

Note that ROC curves and AUCs are unaffected if you sample your test set randomly, namely if you only test potential edges at random from the set of all potential edges – I discussed before how this is a common thing to do because of the unmanageable size of the real test set. However, that is not true if you perform a non-random sampling. This means choosing potential edges according to a specific criterion. If your criterion is "good", meaning that your sampling method is correlated with the actual edge appearance likelihood, you're going to see a different – lower – AUC value. That is because, if you don't sample, the vast number of easy-to-predict false negatives increases your classifier's accuracy.

Precision & Recall

Another way of putting a number to evaluate the quality of the prediction is to look at Precision and Recall⁸. Precision means that, when we predict that a link exists, it exists (even if we fail to predict actual links). Recall means that there are very few existing links we do not predict, even if we might have predicted many that didn't actually exist. So Precision is true positives over all predicted positives (including false positives): $TP/(TP + FP)$. Recall is another name for the True Positive Rate: $TP/(TP + FN)$. Figure 25.8 shows a visual example.

You can do a few things with precision and recall. First, you can transform them into fixed threshold metrics. This is done by calculating what we call "*Precision@n*", defining n as the number of predictions we want to make. For instance, in *Precision@100* we only consider as an actual prediction the 100 pairs of nodes that have the highest scores. Everything else is classified as "no link".

You can also combine precision and recall to generate a derived

⁸ David Martin Powers. Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation. 2011

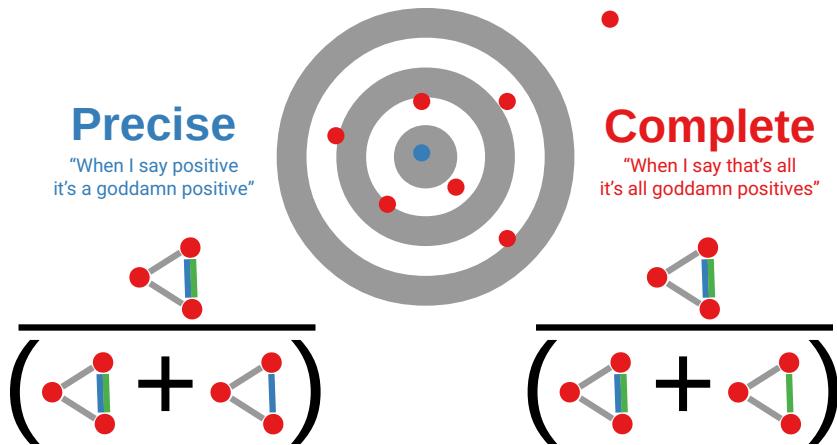


Figure 25.8: A representation of precision and recall.

score, balancing them out. This is known as the *F1*-score, which is their harmonic mean: $F1 = 2(Precision \times Recall) / (Precision + Recall)$. This is a single number, like AUC, capturing both types of errors: failed predictions and failed non-predictions.

A powerful way to use precision and recall is by using them as an alternative to ROC curves. The so-called Precision-Recall curves have the recall on the x-axis and the precision on the y-axis (see Figure 25.9). They tell you how much your precision suffers as you want to recover more and more of the actual new edges in the network. Recall basically measures how much of the positive set your recover. But, as you include more and more links in that set, you're likely to start finding lots of false positives. That will make your recall go up, but precision go down.

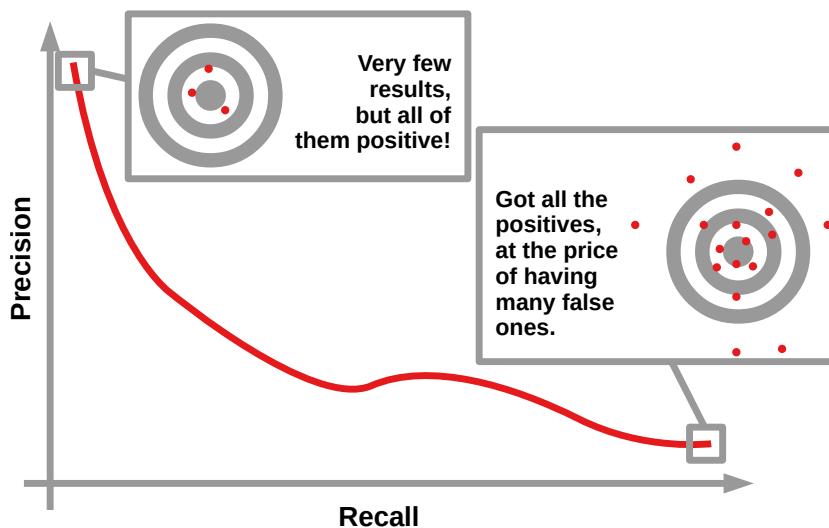


Figure 25.9: A representation of precision-recall curves.

In a Precision-Recall plot, the random classifier is a horizontal line. If you have 10 positive samples in a dataset of 100 entries, then the horizontal line is at $10/100 = 0.1$. That is because, if you're making random predictions, you always have a 10% chance of getting it right. It is a horizontal line because you can achieve any recall value, as long as you keep trying. You can make 100 predictions, which by definition will get you all positives – and a recall of 1 – while the precision will still be 0.1.

A final way to use precision for evaluating link prediction methods is to use the *prediction power*⁹. This is a measure that compares the precision of your classifier with the one you would obtain from a random classifier returning random links without looking at the network topology. If we say that your precision is P and the random precision is P_r , then the prediction power PP is

$$PP = 10 \log_{10} \frac{P}{P_r}.$$

This is a decibel-like logscale: a $PP = 1$ implies your predictor is ten times better than random, while $PP = 2$ means you are one hundred times better than random. You can also create PP -curves by having on the x-axis the share of links you remove from your training set. By definition, the random predictor is an horizontal line at 0. The more area your PP curve can make over the horizontal zero, the more precise your predictor is.

In closing, I should also mention another popular measure: accuracy. This is simply $(TP + TN) / (TP + TN + FP + FN)$: the number of times you got it right over all the attempts. The lure of accuracy is its straightforward intuition. However, it hides the difference between type I and type II errors – false positives and false negatives – and thus it should be handled with care.

25.3 Summary

1. To evaluate the quality of a link prediction you need to train your algorithm and then test it. To do so, you need to divide the data in mutually exclusive train and test sets.
2. If your data has temporal information you can decide a cutoff date to divide the two sets. Otherwise you have to perform cross validation: divide the data in ten blocks and rotate one block as test set using the other nine as training, until you tested on all data.
3. Since real networks are sparse, there are more non-edges than edges. Thus a link prediction always predicting non-edge would

⁹ Carlo Vittorio Cannistraci, Gregorio Alanis-Lobato, and Timothy Ravasi. From link-prediction in brain connectomes and protein interactomes to the local-community-paradigm in complex networks. *Scientific reports*, 3:1613, 2013

have high performance. That is why you should balance your test sets, having an equal number of edges and non-edges.

4. A classical evaluation strategy is the ROC curve, recording your true positive rate against your false positive rate. The more area under this curve (AUC) you have the better your prediction performance.
5. Precision is the ability of returning only true positive results at the price of missing some. Recall is the ability of returning all positive results, at the price of returning also lots of false positives. You can draw precision-recall curves, again with the objective of maximizing their AUC.

25.4 Exercises

1. Divide the network at <http://www.networkatlas.eu/exercises/25/1/data.txt> into train and test sets using a ten-fold cross validation scheme. Draw its confusion matrix after applying a jaccard link prediction to it. Use 0.5 as you cutoff score: scores equal to or higher than 0.5 are predicted to be an edge, anything lower is predicted to be a non-edge. (Hint: make heavy use of `scikit-learn` capabilities of performing KFold divisions and building confusion matrices)
2. Draw the ROC curves on the cross validation of the network used at the previous question, comparing the following link predictors: preferential attachment, jaccard, Adamic-Adar, and resource allocation. Which of those has the highest AUC? (Again, `scikit-learn` has helper functions for you)
3. Calculate precision, recall, and F1-score for the four link predictors as used in the previous question. Set up as cutoff point the ninetieth percentile, meaning that you predict a link only for the highest ten percent of the scores in each classifier. Which method performs best according to these measures? (Note: when scoring with the `scikit-learn` function, remember that this is a binary prediction task)
4. Draw the precision-recall curves of the four link predictors as used in the previous questions. Which of those has the highest AUC?

Part VIII

The Hairball

26

Bipartite Projections

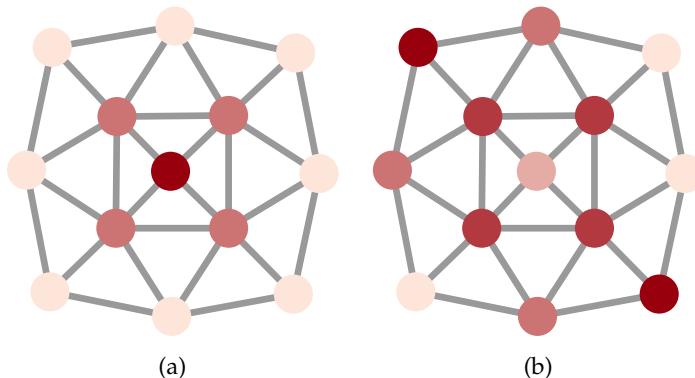
Reality does not usually match expectations. Let's consider three examples:

1. Degree distributions;
2. Epidemics spread;
3. Communities.

Many papers have been written on how power law degree distributions are ubiquitous^{1,2,3,4}. Chances are that any and all the networks you'll find on your way as a network analyst do not have even a hint of a power law degree distribution. In the best case scenario you are going to have shifted power laws, or exponential cutoffs – if you're lucky – (for a refresher on these terms, see Section 9.4).

My second example is epidemics spread – Figure 26.1. As we saw in Part VI, SIS/SIR models tell us exactly when the next node is going to be activated. In practice, data about real activation times has (a) high levels of noise, (b) many exogenous factors that have as much power in influencing how the infection spreads as the network connections have.

Third, and more famously, communities. We are not going to dive deeply into the topic only until Part X. But, very superficially, when



¹ Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *science*, 286(5439):509–512, 1999

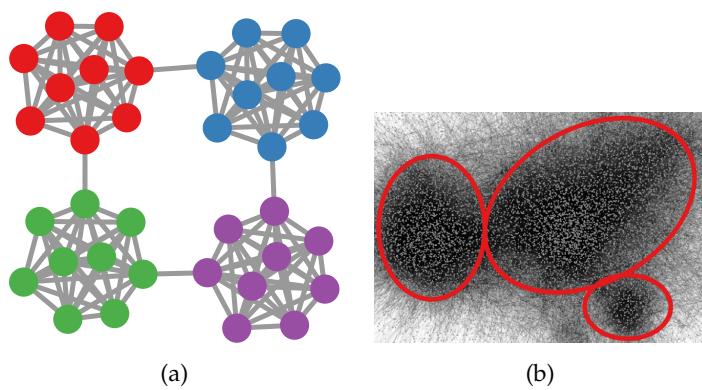
² Albert-László Barabási and Eric Bonabeau. Scale-free networks. *Scientific american*, 288(5):60–69, 2003

³ Reka Albert. Scale-free networks in cell biology. *Journal of cell science*, 118(21):4947–4957, 2005

⁴ Albert-László Barabási. Scale-free networks: a decade and beyond. *science*, 325(5939):412–413, 2009

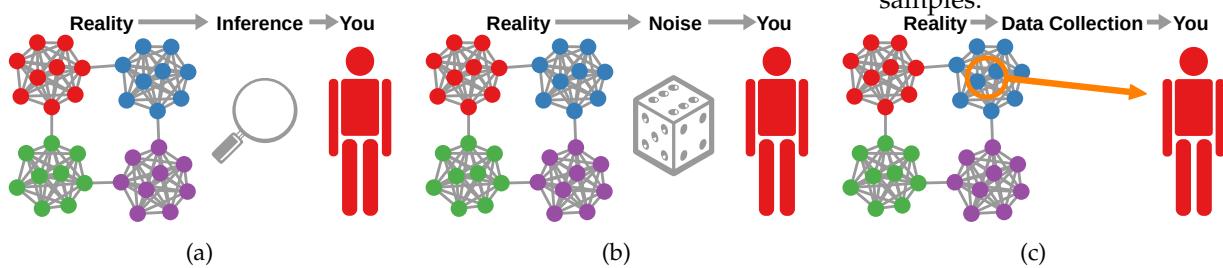
Figure 26.1: (a) Theory-driven mechanically explained activation times, represented by the node color (from dark to bright). (b) Real data swamped with noise, which only mildly conforms to the network topology.

it comes to community discovery, the vast majority of papers propose a very naive standard definition of what constitute communities in a network: “Groups of nodes that have a very large number of connections among them and very few to nodes outside the group”. Many papers claim that most networks have this kind of organization – references provided in Part X. 99% of networks will instead look like a blobbed mess. We have not one but three names for this useless visualization of an (apparently) useless network structure: ridiculogram – a term which you can find sneaking around in some papers⁵ and attributed to Marc Vidal –; spaghettigraph – a term I’m fond of due to my Italian origins; and hairball – the term I’ll use from now on in the book. See Figure 26.2 for an example.



There are a few ways in which hairballs arise, which are the focus of this book part. First, many networks are not observed directly: they are inferred (Figure 26.3(a)). If the edge inference process you’re applying does not fit your data, it will generate edges it shouldn’t. Second, even if you observe the network directly, your observation is subject to noise (Figure 26.3(b)), connections that do not reflect real interactions but appear due to some random fluctuations. Finally, you might have the opposite problem: you’re looking at an incomplete sample (Figure 26.3(c)), and thus missing crucial information.

In the chapters of this book part, we tackle each one of these problems to see some examples in which you can avoid giving birth to yet another hairball. Chapter 27 deals with network backboning: how to clear out noise from your edge observations. Chapter 29



⁵ Petter Holme, Mikael Huss, and Sang Hoon Lee. Atmospheric reaction systems as null-models to identify structural traces of evolution in metabolism. *PLoS One*, 6(5):e19759, 2011

Figure 26.2: (a) Well-separated groups internally densely connected. (b) The ubiquitous and mighty hairball.

Figure 26.3: The typical breeding grounds for hairballs: (a) Indirect observation, (b) Noise in the data, (c) Incomplete samples.

focuses on the problem of network sampling: if you have a huge network in front of you, how do you extract a part of it so that your sample is representative?

Here, we start by tackling the first problem: how to deal with indirectly observed networks. Most of the times, you want to connect things because they are somehow similar, or they do similar things, or they relate to similar things. For instance, you want to connect users because they watch the same movies on Netflix. The most natural way to represent these cases is with bipartite networks (see Section 7.1): in my example, a network connecting each user to the movie they watched. However, you don't want a bipartite network, you want a normal, down-to-earth, honest-to-god unipartite network. What can you do in this case?

Project! Bipartite projection means that you have a bipartite network with nodes of type V_1 and V_2 , and you want to create a unipartite network with only nodes of type V_1 (or V_2). In my Netflix example, all you observe is people watching movies. As I said before, this is a bipartite network: nodes of type V_1 are people, nodes of type V_2 are movies, and edges go from a person to a movie if the person watched the movie. However, the holy grail is to know which movies are similar, to make recommendations to similar users.

In the following sections we explore the different ways in which one can project a bipartite network. They all boil down to the same strategy: we use a different criterion to give the projected edges a weight, we establish a threshold, and drop the edges below this minimum acceptable weight.

26.1 Simple Weights

Let's stick with our Netflix example⁶. Naively, you might think that you can connect movies because the same people watched them⁷ – as in Figure 26.4. The problem is that – as we saw – degree distributions

⁶ Note that, hereafter, I ignore the fact that in Netflix you could also rate the movie, i.e. that the bipartite network is weighted. In my example, I treat the bipartite network as unweighted.

⁷ Mark EJ Newman. Scientific collaboration networks. i. network construction and fundamental results. *Physical review E*, 64(1):016131, 2001b

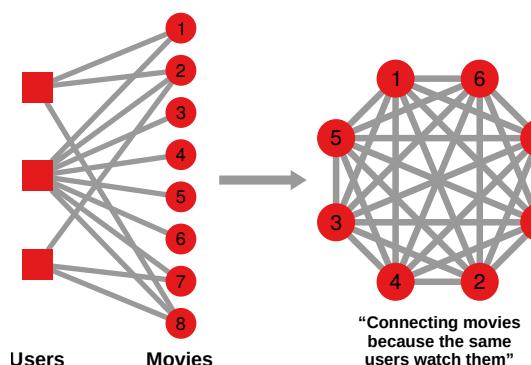


Figure 26.4: An example of naive bipartite projection, where we connect nodes of one type if they have a common neighbor.

are broad. This means that there are going to be some users in your bipartite user-movie network with a very high degree. These are power users, people who watched everything. They are a problem: under the rule we just gave to project the bipartite networks, you'll end up with all movies connected to each other. A hairball. The key lies in recognizing that not all edges have the same importance. Two movies that are watched by three common users are more related to each other than two movies that only have one common spectator.

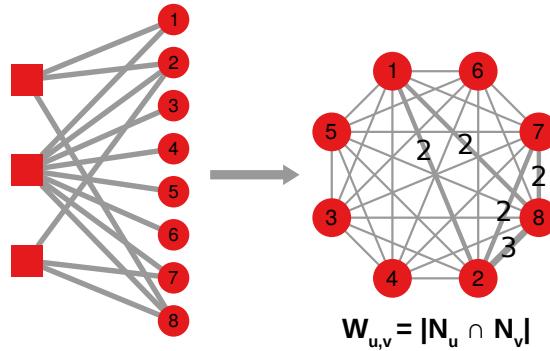


Figure 26.5: An example of Simple Weight bipartite projection, where we connect nodes of one type with the number of their common neighbors.

The easiest way to take this information into account is to perform **simple weighting**. For each pair of nodes you identify the number of common neighbors they have, and that's the weight of the edge – see Figure 26.5. In practice, you don't simply require that movies are connected if there is at least one person who has watched both of them. You connect movies with a weighted link, and the weight is the number of people who watched them both: $w_{u,v} = |N_u \cap N_v|$. This weighting scheme is similar to Common Neighbors in link prediction (Section 23.2), and of course you can do a Jaccard correction by normalizing it with the size of the union of the neighbor sets: $w_{u,v} = |N_u \cap N_v| / |N_u \cup N_v|$.

If you like to think in terms of matrices (Chapter 8), this is equivalent to multiplying the bipartite adjacency matrix with its transpose. Of course, you need to pay attention to the dimension onto which you're projecting. If A is a $|V_1| \times |V_2|$ matrix, then AA^T is a $|V_1| \times |V_1|$ matrix, while A^TA is a $|V_2| \times |V_2|$ one. When multiplying binary matrices, the result in cell A_{uv} is the number of common entries set to one between the u th and the v th rows, which is exactly the number of common neighbors between nodes u and v . The diagonal will tell you the degree of the node, which you can simply set to zero.

This approach can be integrated with a second step⁸. In this second step, one wants to evaluate the statistical significance of the edge weights you obtained by counting the number of common neighbors. This is sort of the same thing as first projecting and then performing network backboning – a task we'll see in Chapter 27.

⁸ Fabio Saracco, Mika J Straka, Riccardo Di Clemente, Andrea Gabrielli, Guido Caldarelli, and Tiziano Squartini. Inferring monopartite projections of bipartite networks: an entropy-based approach. *New Journal of Physics*, 19(5):053022, 2017

The main difference is that this backboning is specially defined to clean up the result of bipartite projections. One can define a series of null bipartite network models, either via exponential random graphs (Section 19.2) or configuration model (Section 18.1). These null models will give birth to a bunch of null projections, which will give an expected weight for all possible edges in the unipartite network. Then, you can keep in your projection only those links significantly exceeding random expectation.

26.2 Vectorized Projection

There are many criticisms of the simple counts as a weighting approach. Here we see the one called saturation problem⁹. Another issue is the bandwidth problem, which I explain in Section 26.3, along with the projection methods designed to fix it.

Some authors noticed that the simple count scheme has what they call a “saturation” problem. As an illustration, consider the following example: suppose you are an author and you collaborated with another scientist on a new paper. The contribution of that new paper to your similarity is not linear. If in your previous history you only had a single other paper with this person, then the new paper is your second collaboration. This is a strong contributor: it represents 50% of your entire scientific output. If, instead, this was your hundredth collaboration, this new paper only adds little to your connection strength. Giving the same weight in these two different scenarios is not a good proxy to estimate the similarity in the original network.

We can exploit edge weights to solve the saturation problem. Edge weights are something that simple counting cannot handle easily, and if you try to handle them by doing a weighted simple counting, you probably end up doing something similar to what I present in this section anyway. In this scenario, you don’t want to count each common V_2 neighbor equally. You need your adjacency matrix to contain non-zero values different than one. For simplicity, I’m going to make the following examples with a binary matrix anyway, also to show that these techniques can handle this simpler scenario as well.

Our sophisticated needs imply that we need to change the way we look at the problem. As the title of this section suggests, we are considering **nodes as vectors**. Specifically, consider the binary adjacency matrix of our bipartite network. Each row is a node of type V_1 . Each entry tells us whether it is connected to a node of type V_2 . So we can see a node as a vector of zeroes and ones.

Once we do – as Figure 26.6 shows –, we discover that we can apply a large number of distance metrics between two numerical vectors. If these numerical vectors represent two V_1 nodes, then the

⁹ Menghui Li, Ying Fan, Jiawei Chen, Liang Gao, Zengru Di, and Jinshan Wu. Weighted networks of scientific communication: the measurement and topological role of weight. *Physica A: Statistical Mechanics and its Applications*, 350(2-4):643–656, 2005

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 |

Simple Weight = 2

CosineSim = 0.66

Pearson + 1 = 1.52

1 / (Euclidean + 1) = 0.41

Figure 26.6: An example of vectorized bipartite projection, where we connect nodes of one type with the inverse of some vector distance measure of their rows in the bipartite adjacency matrix.

distance between them must be – inversely – related to how similar they are. Popular choices to establish the strength of the connection between these two nodes are the Euclidean distance, cosine similarity and Pearson correlation – but the list could be much longer and you can get inspiration from the set of vector distance measures implemented in any statistical library¹⁰.

One nice thing about many of these measures, besides properly handling edge weights, is that they handle also common zeroes. In simple weighting, you only count common neighbors. However, two nodes might be similar also based on the neighbors they *don't* connect to. This is elegantly handled in the Pearson correlation, for instance. Such indirect effects are not always good: for instance they are a problem when performing link prediction¹¹.

You need to be aware of a few problems with this approach. First, it's not always immediately obvious how to translate a distance into a similarity while preserving its properties. You cannot always take the inverse, or multiply by minus one, or doing one minus the distance. Each of these solutions might work with some measures, but catastrophically fail with others.

The second issue is more subtle. None of these measures were really developed with network data in mind. So they might not work because they don't take into account what the edge creation process of the bipartite network looks like. They are not going to necessarily solve the issues simple weighting has, because they're still prone to fall into the trap of large hubs and very skewed degree distributions.

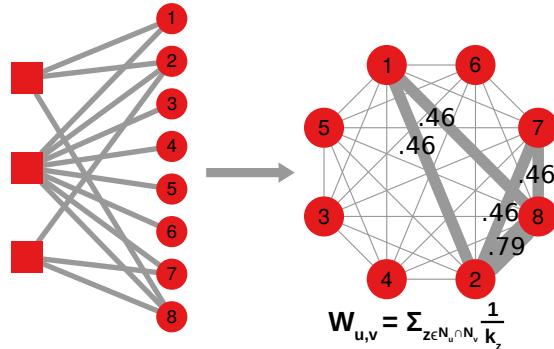
26.3 Hyperbolic Weights

The second problem is similar to the saturation one, but cannot be solved by looking at edge weights. If you're in a CERN paper, you coauthor with hundreds of people, but you don't really know all of them. In practice, we're acknowledging that "bandwidth" is finite: having too many coauthors implies having only a superficial relationship with all of them. This bandwidth argument is not new,

¹⁰ <https://docs.scipy.org/doc/scipy/reference/spatial.distance.html>

¹¹ Baruch Barzel and Albert-László Barabási. Network link prediction by global silencing of indirect correlations. *Nature biotechnology*, 31(8):720–725, 2013a

we saw a similar one when we introduced link prediction methods like Adamic-Adar in Section 23.3 and Resource Allocation in Section 23.4.



In **hyperbolic weight** we recognize that hubs contribute less to the connection weight than non-hubs¹². Such a weight scheme is similar to the link prediction strategies I just mentioned: each common neighbor z contributes k_z^{-1} rather than 1 to the weight of the edge connecting the two nodes: $w_{u,v} = \sum_{z \in N_u \cap N_v} \frac{1}{k_z - 1}$. The final result in this example is similar to simple weight – see Figure 26.7 –, but it exaggerates the differences, so that thresholding becomes easier.

Note that the minus one in the denominator – which we do because u never checks its similarity with itself – means that we’re effectively ignoring all papers with only one author. And, if an author only wrote with herself, she won’t appear in the network. This makes sense at some level – how can you connect with anybody else if you never collaborate? – but it also implies that there is going to be no information in the diagonal of the resulting adjacency matrix.

Again, this projection is simple to implement as a matrix operation. Rather than multiplying the bipartite adjacency matrix with its transpose, you multiply it with the transpose of its degree normalized stochastic version. If you do so, rather than counting the common ones, you sum up all the $1/k_z$ entries. Again, pay attention to the dimension over which you project, because normalizing by row sum or by column sum will change the result.

26.4 Resource Allocation

In **resource allocation** we do the same thing as hyperbolic weight, but considering two steps instead of one. Rather than only looking at the degree of the common neighbor, we also look at the degree of the originating node¹³. In the paper-writing example, not only it is unlikely to be strongly associated with a co-author in a paper

Figure 26.7: An example of Hyperbolic Weight bipartite projection, where each common neighbor z contributes k_z^{-1} to the sum of the edge weight.

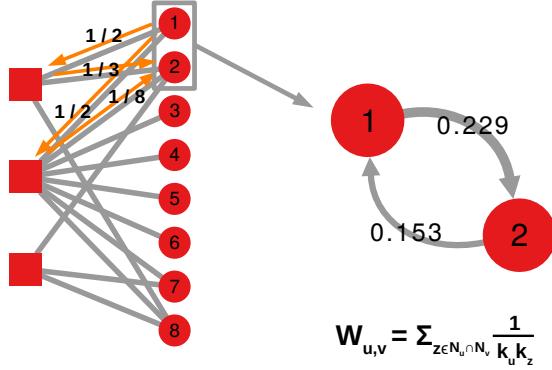
¹² Mark EJ Newman. Scientific collaboration networks. ii. shortest paths, weighted networks, and centrality. *Physical review E*, 64(1):016132, 2001c

¹³ Tao Zhou, Jie Ren, Matúš Medo, and Yi-Cheng Zhang. Bipartite network projection and personal recommendation. *Physical Review E*, 76(4):046115, 2007

with hundreds of authors, it is also difficult to give attention to a particular co-author if you have many papers with many other people. So each common neighbor z that node u has with node v contributes not k_z^{-1} as in hyperbolic weights, but $(k_u k_z)^{-1}$ – see Figure 26.8. The weight is then:

$$w_{u,v} = \sum_{z \in N_u \cap N_v} \frac{1}{k_u k_z}.$$

This generates the unipartite weight matrix W .



This strategy also works for weighted bipartite networks. If B is your weighted bipartite adjacency matrix, the entries of W are:

$$w_{u,v} = \sum_{z \in N_u \cap N_v} \frac{B_{uz}}{k_u k_z}.$$

In practice, you replace the 1 in the numerator with the edge weights connecting z to v and u . Moreover, we can also have node weights, noticing that some nodes might have more resources than others. Suppose that you have a function f giving each node in the network a resource weight. After you perform the resource allocation projection, each node will have a new amount of resources $f' = Wf$.

Note that, in this case, W is not symmetric: in the scenario with a single common neighbor z , u 's score for v would be $(k_u k_z)^{-1}$, while v 's score would be $(k_v k_z)^{-1}$. If $k_u \neq k_v$, then the scores are different. In many cases, this provides a better representation of the network than one ignoring asymmetries. You might be the most similar author to me because I always collaborated with you, but if you also contributed to many other papers with other people, then I might not be the author most similar to you.

W has a well-defined diagonal: $w_{u,u} = \sum_{z \in N_u} \frac{1}{k_u k_z} = \frac{1}{|N_u|} \sum_{z \in N_u} \frac{1}{k_z}$.

In fact, this diagonal is the maximum possible similarity value of the row: only a node v with the very same neighbors and nothing else can have a weight $w_{u,v} = w_{u,u}$.

Figure 26.8: An example of Resource Allocation bipartite projection, where each common neighbor z contributes $(k_u k_z)^{-1}$ to the sum of the edge weight. When connecting node 1 to node 2, from node 1's perspective the edge weight is $(1/2 * 1/3) + (1/2 * 1/8)$, because the two common neighbors have degree of 3 and 8, respectively, and node 1 has degree of two. However, from node 2's perspective, the edge weight is $(1/3 * 1/3) + (1/3 * 1/8)$, because node 2 has three neighbors.

In some other cases you might consider having a directed projection an inconvenience, because you really want an undirected network as a result. You can make the result of resource allocation symmetric by always choosing the minimum or maximum between $w_{u,v}$ and $w_{v,u}$, or simply their average: $(w_{u,v} + w_{v,u})/2$. Also self-loops can be annoying sometimes. If you have no use for them, you can manually set W 's diagonal to zero.

The resource allocation as presented so far is only one of the many possible variants following the same idea. The one I explained so far is known as ProbS and uses k_u , the degree of the origin of the two-step random walk, as the normalizing factor. A variant known as HeatS¹⁴ uses instead k_v , the destination of the random walk. Thus, the weight of the u,v connection is now $w_{u,v} = \sum_{z \in N_u \cap N_v} \frac{1}{k_v k_z}$.

Surprising absolutely no one, some authors decided to combine ProbS and HeatS in a single Hybrid framework¹⁵. The combination is exactly what you would expect: $w_{u,v} = \sum_{z \in N_u \cap N_v} \frac{1}{k_u^\lambda k_v^{(1-\lambda)} k_z}$. This introduces a parameter in the equation: λ . This should be a number between zero and one, determining how much importance the degree of the origin has compared to the degree of the destination. For $\lambda = 0$ you have HeatS, for $\lambda = 1$ you have ProbS, and for $\lambda = 1/2$ you have the middle point between HeatS and ProbS.

In matrix terms, ProbS is the same as the hyperbolic projection (Section 26.3), but now you normalize differently. In hyperbolic, you multiply the adjacency matrix A with its degree-normalized transpose. In ProbS you multiply the stochastic with a stochastic version of the transpose. Meaning, in hyperbolic first you normalize then you transpose, in ProbS first you transpose and then you normalize. Finally, HeatS is the transpose of ProbS.

26.5 Random Walks

In **Random Walks**, we take the resource allocation to the extreme. Rather than looking at 2-step walks, we look at infinite length random walks. Which means that the strength between u and v is the probability of visiting v starting from u . If we have infinite random walks, this means that we can use the stationary distribution to estimate the edge weight: $w_{u,v} = \pi_v A_{u,v}$, where A is a transition probability matrix (recording the probability of the path $u \rightarrow z \rightarrow v$, for any z)¹⁶. Note that A here is different than a simple binary adjacency matrix, as it encodes the probabilities of all random walks of length two. This means that its interpretation is slightly different than what I originally presented for π in Section 14.4.

¹⁴ Tao Zhou, Zoltán Kuscsik, Jian-Guo Liu, Matúš Medo, Joseph Rushton Wakeling, and Yi-Cheng Zhang. Solving the apparent diversity-accuracy dilemma of recommender systems. *Proceedings of the National Academy of Sciences*, 107(10):4511–4515, 2010

¹⁵ Linyuan Lü and Weiping Liu. Information filtering via preferential diffusion. *Physical Review E*, 83(6):066119, 2011

¹⁶ Muhammed A Yildirim and Michele Coscia. Using random walks to generate associations between objects. *PloS one*, 9(8):e104813, 2014

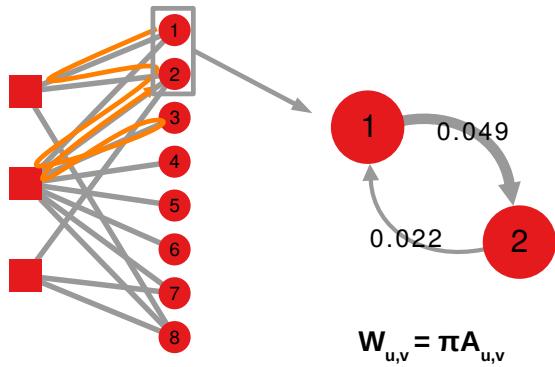


Figure 26.9: An example of Random Walks bipartite projection, where the connection strength between u and v is dependent on the stationary distribution π , telling us the probability of ending in v after a random walk.

In matrix terms, you take the result ProbS' multiplication, which is a square $|V_1| \times |V_1|$ matrix, and you multiply it with its stationary distribution.

As in the resource allocation case, this means that the measure is not symmetric, and the differences between nodes now are more extreme than before: the $1 \rightarrow 2$ edge weight is now more than twice as $2 \rightarrow 1$, while in resource allocation it was just about 50% higher. See Figure 26.9 for an example. Another parallelism between these two approaches is the presence of a well-defined diagonal, which you can use in case you're not afraid of self-loops (I am).

26.6 Comparison in a Practical Scenario

I showed you how these different methods approach the projection process and the different results they obtain in a toy example. Do these differences in simple scenarios translate to big practical differences in real-world cases? To answer this question, let's just take a superficial look at the projections I get using a bipartite network extracted from Twitter. In the network, the nodes of type V_1 are websites, and nodes of type V_2 are Twitter users. I connect a Twitter user to a website if the user included the URL of the website in one of her tweets.

I project the network so that I have a unipartite version with only V_1 nodes: websites are connected if the same users tweet about them. This is a sort of website similarity index. Now let's see how different the space of edge weights looks like if we use different approaches. This is what Figure 26.10 is all about.

The figure shows that the space of the edge weights looks pretty different according to different projection methods. For instance, the simple projection (Figure 26.10(a)) shows a power law distribution of edge weights, with more than four million edges with weight equal to one and one edge with weight equal to 2,252, while the average

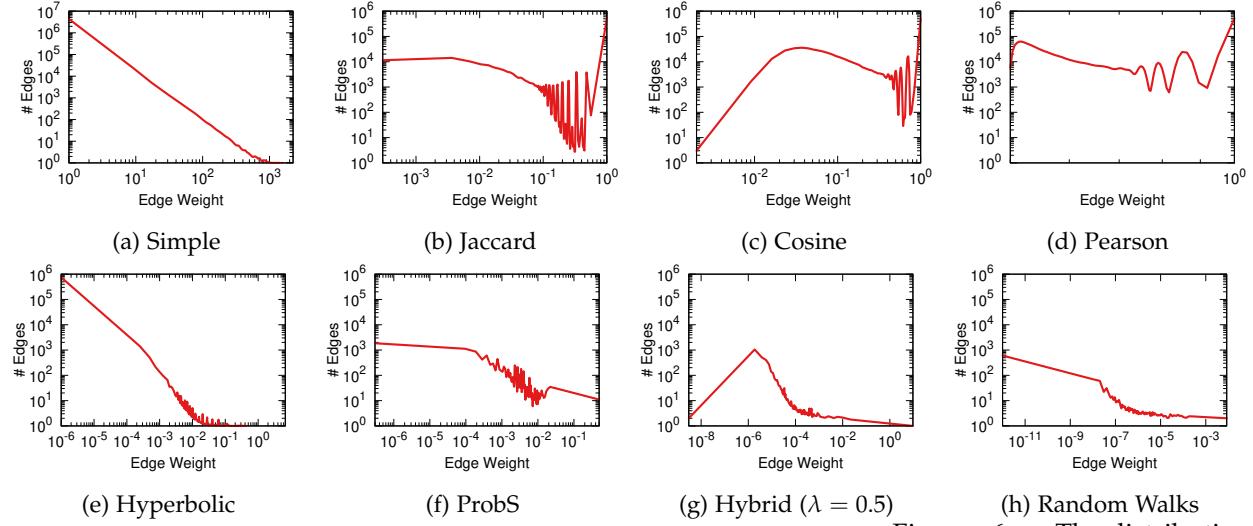


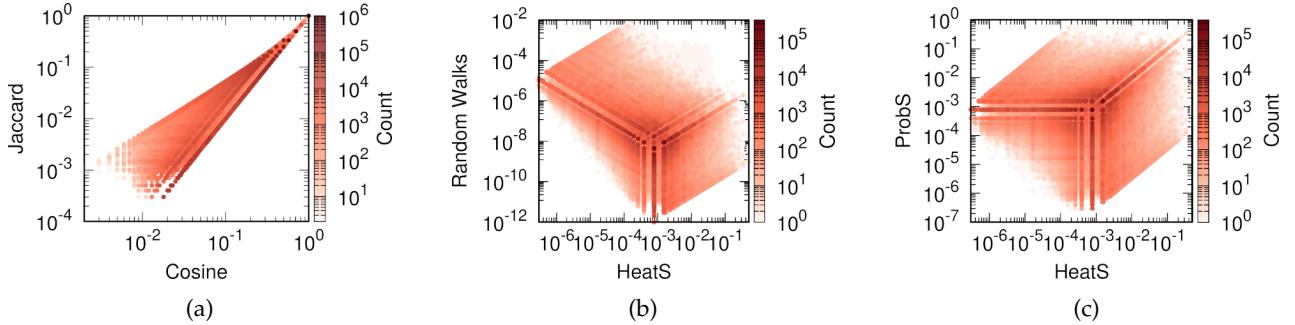
Figure 26.10: The distributions of edges weights in the projected Twitter network for eight different projection methods. The plot report the number of edges (y axis) with a given weight (x axis).

weight is 2.14. This is very much not the case for other projection strategies such as Jaccard (Figure 26.10(b)), where there is no trace of a power law. And, in many cases such as cosine and Pearson (Figures 26.10(c-d)), the highest edge weight is actually the most common value, rather than being an outlier such as in the hyperbolic projection (Figure 26.10(e)).

Is the difference exclusively in the shape of the distribution, or do these approaches disagree on the weights of specific edges? To answer this question we have to look at a scattergram comparing the edge weights for two different projection strategies. This is what I do in Figure 26.11.

I picked three cases to show the full width of possibilities. In Figure 26.11(a), I compare the cosine projection against the Jaccard one. This is the pair of projections that, in this dataset, agree the most. Their correlation is > 0.94 . Looking at the figure, it is easy to see that there isn't much difference. You can pick either method and you're going to have comparable weights. The opposite case compares two method that are anti-correlated the most. This would be HeatS and the random walks approach, in Figure 26.11(b). They correlation in a log-log space is a staggering -0.7 . From the figure you can probably spot a few patterns, but the lesson learned is that the two methods build fundamentally different projections.

Ok, but these are extreme cases. How does the average case looks like? To get an idea, I chose a particular pair of measures: HeatS and ProbS (Figure 26.11(c)). You might expect the two to be more similar than the average method: after all, one is the transpose of the other. You'd be very wrong. In this dataset, HeatS and ProbS are actually anti correlated, at -0.34 in the log-log space. HeatS and ProbS would be positively correlated if the nodes of type V_1 with similar degrees



connect to the same nodes of type V_2 . But that is not the case in this specific Twitter dataset. Here, it is not true that the people sharing lots of URLs share the same URLs.

At this point, you might be asking yourself how do you choose the projection method that is most suitable for your application. The general guideline is to study what each method does and see if it aligns with your expected edge generation process. However, I feel it's a bit too early to ask this question. That is because network projection is rarely the only thing you're going to do. Almost all these methods return the same set of non-zero weighted edges. They also return extremely dense projections, as a single common node is enough to create an edge in the projection.

In fact, the Twitter data I just used has $\sim 15k$ users and $\sim 14k$ domains, with $\sim 175k$ edges connecting them. The undirected projections return $\sim 5.3M$ edges, meaning a density of $2 \times 5.3M / 14k^2 \sim 5\%$, or an average degree of ~ 713 . This is usually way too much for an intelligible network. That is why, if we want to avoid hairballs and related problems, these techniques – while necessary – are not usually sufficient. The process to get rid of hairballs has two steps: first one performs the bipartite projection, and then she applies a threshold to throw away low-weighted edges. The next chapter expands on how to perform this second step properly.

26.7 Summary

1. Most network analysis algorithms work with unipartite networks, but many phenomena have a natural bipartite representation. To transform a bipartite network into a unipartite network you need to perform the task of network projection.
2. In network projection you pick one of the two node types and you connect the nodes of that type if they have common neighbors of the other type. Normally you'd count the number of common neighbors they have (simple weighted) and then evaluate their

Figure 26.11: The comparison between the edge weights according to different network projections. Each point is an edge. The x-y coordinates encode its weight in the two different projections. The color encodes how many edges share the same x-y score. (a) Cosine vs Jaccard; (b) HeatS vs Random Walks; (c) HeatS vs ProbS.

statistical significance.

3. Real world bipartite networks have broadly distributed degrees which might make your projection close to a fully connected clique. Then you need a smart weighting scheme to aid you in removing weak connections.
4. You could use standard vector distances (cosine, euclidean, correlation) but we have specialized network-aware techniques. For instance, considering nodes as allocating resources to their neighbors, inversely proportional to the number of neighbors they have (hyperbolic).
5. In resource allocation, you also have nodes sending resources, but you take two steps instead of one: you're not discounting only for the degree of nodes of type one, but also for the degree of nodes of type two.
6. Finally, you can also do resource allocation with infinite length random walks by looking at the stationary distribution. The resulting edge weights from all these techniques can create very different network topologies.

26.8 Exercises

1. Perform a network projection of the bipartite network at <http://www.networkatlas.eu/exercises/26/1/data.txt> using simple weights. The unipartite projection should only contain nodes of type 1 ($|V_1| = 248$). How dense is the projection?
2. Perform a network projection of the previously used bipartite network using cosine and Pearson weights. What is the Pearson correlation of these weights compared with the ones from the previous question?
3. Perform a network projection of the previously used bipartite network using hyperbolic weights. Draw a scatter plot comparing hyperbolic and simple weights.

27

Network Backboning

Network backboning is the problem of taking a network that is too dense and removing the connections that are likely to be not significant – or “strong enough”. If you ever found yourself in a situation thinking “there are too many edges in this network, I’m going to filter some out”, then you performed network backboning. Even if it is rarely explicitly labeled like that, network backboning is one of the most common tasks performed in network analysis.

There are many reasons why you would want to backbone your network. First, this is a book part about hairballs. If your network is a hairball, meaning that the tangle of connections is too dense to reach any meaningful conclusion, you might want to sparsify your network. Graph sparsification^{1,2} – sometimes called “pruning” in combinatorics³ and neural networks⁴ – could be an alternative name for backboning, but it is often used in a more narrow context, namely the second application field of backboning: your network simply has too many connections to be computationally tractable and so you need to filter out the ones that are unlikely to affect your computation. Finally, a third scenario might be the presence of noise: you don’t know whether the edges you’re observing are real connections and you need a statistical test to determine that. This overlaps with the notion of measurement error – which is the topic of Chapter 28 –, but it is slightly different. The techniques presented in this chapter aim to remove connections that are likely to be noisy, while in Chapter 28 we will see how to deal with such connections without removing them, mainly via the use of probabilistic networks.

When wearing its “graph sparsification” hat, network backboning could be confused with graph summarization: the task of taking a large complex network and reducing its size so that we can describe it better. However, there is a crucial difference between the two tasks: one of the central objectives of graph summarization is to reduce the number of nodes of the network as much as possible, often even merging them into “meta nodes”. This is exactly the opposite of what

¹ Daniel A Spielman and Shang-Hua Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 81–90, 2004

² Venu Satuluri, Srinivasan Parthasarathy, and Yiye Ruan. Local graph sparsification for scalable clustering. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pages 721–732, 2011

³ Daniel Damir Harabor, Alban Grastien, et al. Online graph pruning for pathfinding on grid maps. In *AAAI*, pages 1114–1119, 2011

⁴ Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. *arXiv preprint arXiv:1810.05270*, 2018c

backboning wants to do: in network backboning you do not merge nodes and you want to keep as many as possible in your network. The reason is that you want to let the strong connections emerge, but you want to preserve all the entities in your data. If you remove nodes from your network, you cannot describe them directly any more in your analysis. In a nutshell, network backboning wants to allow you to perform node- and global-level analyses, while graph summarization only focuses on empowering meso-level analysis (Part IX) where you lose sight of the single individual nodes. For this reason, graph summarization has its own chapter (Chapter 46) in a totally different part of this book.

There are several network backboning methods which aim to tackle this problem. I'll look at a few techniques divided in two macro categories: structural approaches (naive thresholding, Doubly Stochastic, High Salience Skeleton, convex network reduction), and statistical ones (Disparity Filter, Noise Corrected). These are, to the best of my knowledge, the most used and are the ones that I'm the most familiar with. There are other backboning methods, many of which are based on the same "urn extraction" procedure we'll see in depth when we talk about the noise-corrected backbone: for bipartite networks⁵, using Polya urns⁶, and more. Another common approach is to create a null version of the observed network and testing the edge weights against such expectation⁷, in line with the disparity filter we'll see.

Note that finding the maximum spanning tree, planar maximally filtered graphs, and the triangulated maximally filtered graphs could also be considered a way to perform structural network backboning, and they were covered in Section 13.4.

The vast majority of methods in this area of research work on weighted networks. You could, in principle, apply some of them to unweighted networks as well, but you might fall off the use cases that were taken in consideration when developing such algorithms.

27.1 Naive

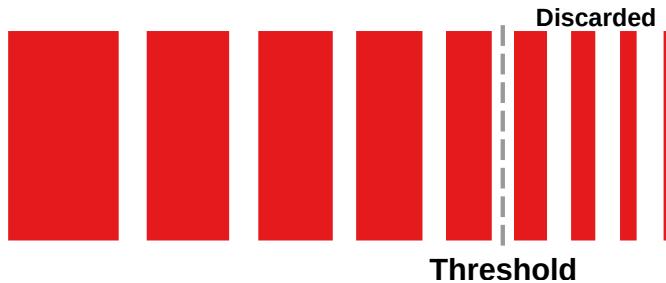
The reason not many network researchers mention the backboning problem is because they usually apply a limited set of naive strategies and do not recognize it as a problem in itself. In fact, there is an easy naive solution that most researchers apply without a second thought. If we have a weighted network and we want to keep the "strongest connections", we sort them in decreasing order of intensity. We decide a threshold, a minimum strength we accept in the network. Everything not meeting the threshold is discarded.

Figure 27.1 provides a vignette of this procedure. There are two

⁵ Michele Tumminello, Salvatore Micciche, Fabrizio Lillo, Jyrki Piilo, and Rosario N Mantegna. Statistically validated networks in bipartite complex systems. *PloS one*, 6(3):e17994, 2011

⁶ Riccardo Marcaccioli and Giacomo Livan. A pólya urn approach to information filtering in complex networks. *Nature Communications*, 10(1):745, 2019

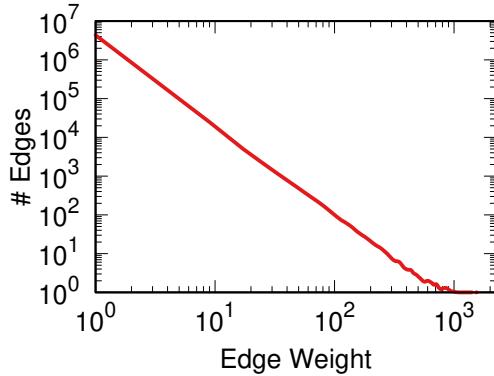
⁷ Filippo Radicchi, José J Ramasco, and Santo Fortunato. Information filtering in complex weighted networks. *Physical Review E*, 83(4):046101, 2011



problems with the naive strategy.

Broad Weight Distributions

The first problem is that, in real world networks, edge weights distribute broadly in a fat-tail highly skewed fashion, much like the degree (Section 9.3). Let's take a quick look again at the edge weight distribution we got using the simple projection in the previous chapter for our Twitter network. I show the distribution again in Figure 27.2.



In this network, 82% of the edges have weight equal to one. The smallest possible hard threshold would remove 82% of the network, without allowing for any nuance. Moreover, since we have a fat tailed edge weight distribution, it is hard to motivate the choice of a threshold. Such a highly skewed distribution lacks of a well-defined average value and has undefined variance (Section 3.4). You cannot motivate your threshold choice by saying that it is " x standard deviations from the average" or anything resembling this formulation.

Figure 27.1: A vignette of the naive thresholding procedure. Each red bar is an edge in the network. The bar's width is proportional to the edge's weight. Here, I sort all edges in decreasing weight order. I then establish a threshold and discard everything to its right.

Figure 27.2: The distribution of edge weights in the projected Twitter network using the simple projection strategy. The plot reports the number of edges (y axis) with a given weight (x axis).

Local Edge Weight Correlations

The second problem is that edge weights are usually correlated. Nodes that connect strongly tend to connect strongly with everybody. In our sample Twitter network, let's consider a user u who only had shared a single URL. If u connects to any v , there can be only one possible edge weight in the simple projection: one. All edges around u will have weight equal to one. On the other hand, if u had shared thousands of URLs, it will likely connect to another user with similar sharing patterns, because statistically speaking they have high odds of sharing at least few of the same URLs, even if it happens by chance. Thus many edges around u will have high weights.

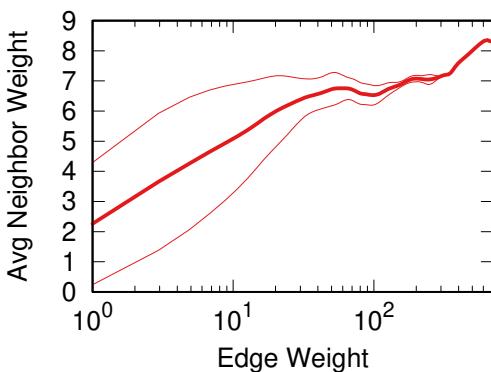
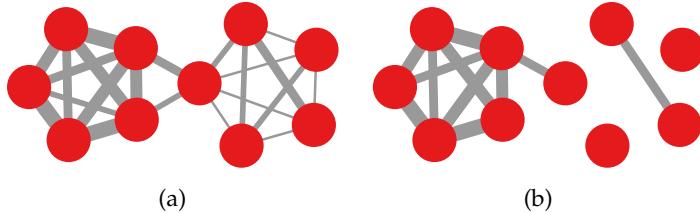


Figure 27.3: The average weight of edges sharing a node with a focus edge (y axis) against the weight of the focus edge (x axis). Thin lines show the standard deviation. One percent sample of the Twitter network.

This is what I mean when I say that the weight of an edge is correlated with the weights of the edges of the nodes it connects. Figure 27.3 shows how this correlation looks like in the Twitter network. The higher an edge weight, the higher on average the weights of edges sharing a node with it. Here, the correlation is ~ 0.69 . The figure has the edge weights in log scale, since they are broadly distributed. The correlation of the average neighbor weight against the logarithm of the edge weight is ~ 0.84 .

This means that there are areas of the network with high edge weights and areas with low weights. If we impose the same threshold everywhere, some nodes will retain all their connections and others will lose all of theirs, without making the structure any clearer. Figure 27.4 provides a vignette of this issue. In the figure, we completely destroy the topological information in the rightmost clique, while at the same time being unable to sparsify the leftmost clique.

An alternative “naive” strategy you could apply is to simply pick the top n strongest connections for each node. This would not be affected by the issues I mentioned. However, by applying it you're effectively determining the minimum degree of the network to be n . This is a heinous crime against the God of power law degree



distributions and, if you commit it, you will be tormented by scale free demons in network hell for all eternity.

27.2 Doubly Stochastic

The next approach we look at is the **doubly stochastic** strategy. Remember what a stochastic matrix is: it is the adjacency matrix normalized such that the sum of the columns is 1 (Section 8.2). A *doubly stochastic* matrix is a matrix in which the sums of both rows and columns are equal to one. You can transform an adjacency matrix into its corresponding doubly stochastic by alternatively normalizing rows and columns until they both sum to 1.

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|-----|-----|-----|-----|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|--|---|---|---|---|---|-----|---|-----|-----|-----|-----|-----|---|-----|-----|-----|-----|-----|---|-----|-----|-----|-----|-----|---|---|---|---|---|---|---|---|---|---|-----|---|---|---|-----|-----|---|---|---|-----|-----|---|---|---|-----|-----|---|---|---|-----|-----|---|---|---|---|---|
| <table border="1"><tr><td>0</td><td>3</td><td>4</td><td>4</td><td>5</td></tr><tr><td>3</td><td>0</td><td>5</td><td>5</td><td>4</td></tr><tr><td>4</td><td>5</td><td>0</td><td>2</td><td>3</td></tr><tr><td>4</td><td>5</td><td>2</td><td>0</td><td>4</td></tr><tr><td>5</td><td>4</td><td>3</td><td>4</td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td><td>2</td><td>3</td></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>2</td></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr></table> | 0 | 3 | 4 | 4 | 5 | 3 | 0 | 5 | 5 | 4 | 4 | 5 | 0 | 2 | 3 | 4 | 5 | 2 | 0 | 4 | 5 | 4 | 3 | 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 2 | 3 | 0 | 0 | 0 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | <table border="1"><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr><tr><td>0.2</td><td>0</td><td>0.4</td><td>0.3</td><td>0.2</td></tr><tr><td>0.3</td><td>0.4</td><td>0</td><td>0.1</td><td>0.2</td></tr><tr><td>0.2</td><td>0.3</td><td>0.1</td><td>0</td><td>0.2</td></tr><tr><td>0.2</td><td>0.2</td><td>0.2</td><td>0.2</td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0.2</td></tr><tr><td>0</td><td>0</td><td>0</td><td>0.2</td><td>0.1</td></tr><tr><td>0</td><td>0</td><td>0</td><td>0.1</td><td>0.2</td></tr><tr><td>0</td><td>0</td><td>0</td><td>0.4</td><td>0.2</td></tr><tr><td>0</td><td>0</td><td>0</td><td>0.2</td><td>0.2</td></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr></table> | 0 | 0 | 0 | 0 | 0 | 0.2 | 0 | 0.4 | 0.3 | 0.2 | 0.3 | 0.4 | 0 | 0.1 | 0.2 | 0.2 | 0.3 | 0.1 | 0 | 0.2 | 0.2 | 0.2 | 0.2 | 0.2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0.2 | 0 | 0 | 0 | 0.2 | 0.1 | 0 | 0 | 0 | 0.1 | 0.2 | 0 | 0 | 0 | 0.4 | 0.2 | 0 | 0 | 0 | 0.2 | 0.2 | 0 | 0 | 0 | 0 | 0 |
| 0 | 3 | 4 | 4 | 5 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 3 | 0 | 5 | 5 | 4 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 4 | 5 | 0 | 2 | 3 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 4 | 5 | 2 | 0 | 4 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 5 | 4 | 3 | 4 | 0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 0 | 0 | 0 | 0 | 0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 0 | 0 | 0 | 0 | 1 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 0 | 0 | 0 | 1 | 0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 0 | 0 | 0 | 2 | 3 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 0 | 0 | 0 | 0 | 2 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 0 | 0 | 0 | 0 | 0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 0 | 0 | 0 | 0 | 0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 0.2 | 0 | 0.4 | 0.3 | 0.2 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 0.3 | 0.4 | 0 | 0.1 | 0.2 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 0.2 | 0.3 | 0.1 | 0 | 0.2 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 0.2 | 0.2 | 0.2 | 0.2 | 0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 0 | 0 | 0 | 0 | 0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 0 | 0 | 0 | 0 | 0.2 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 0 | 0 | 0 | 0.2 | 0.1 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 0 | 0 | 0 | 0.1 | 0.2 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 0 | 0 | 0 | 0.4 | 0.2 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 0 | 0 | 0 | 0.2 | 0.2 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 0 | 0 | 0 | 0 | 0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| (a) | (b) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

After you perform such normalization, the scale of all edges is the same, and you break local correlations – as I show in Figure 27.5. You can now threshold the edges without fearing for the issues we mentioned before⁸. The original paper proposing this technique has specific guidelines on how to perform this thresholding. You should pick the threshold that allows your graph to be a single connected component. However, in many cases you might have different analytic needs. Thus you can specify your own threshold.

The downside of this approach is that not all matrices can be transformed into a doubly stochastic. Only strictly positive matrices can^{9,10}, meaning that the matrix cannot contain zero elements. Since real world networks are sparse, they actually contain lots of zeros.

So this solution cannot be always applied, although, in practice, my experience is that failure to converge is the exception rather than the rule. The easy solution of adding a small ϵ to the matrix to get rid of zero entries does not always make sense. As $\epsilon \rightarrow 0$,

Figure 27.4: Establishing a hard threshold in a network with correlated edge weights. (a) I represent the edge weight with the width of the line. (b) I eliminate all the edges with a weight lower than a given threshold, equal for all edges.

Figure 27.5: An example of Doubly Stochastic network backboning. (a) The adjacency matrix has areas of the network with different edge weight scales. (b) Its doubly stochastic counterpart has no such correlations.

⁸ Paul B Slater. A two-stage algorithm for extracting the multiscale backbone of complex weighted networks. *Proceedings of the National Academy of Sciences*, 106(26):E66–E66, 2009

⁹ Richard Sinkhorn. A relationship between arbitrary positive matrices and doubly stochastic matrices. *The annals of mathematical statistics*, 35(2):876–879, 1964

¹⁰ Richard Sinkhorn and Paul Knopp. Concerning nonnegative matrices and doubly stochastic matrices. *Pacific Journal of Mathematics*, 21(2):343–348, 1967

meaning that $A + \epsilon \rightarrow A$, the normalization of $A + \epsilon$ does not converge.

Note also that a doubly stochastic matrix must be square. This is easy to see: if all rows sum to one, then the sum of all entries in the matrix must be the number of rows. On the other hand, if all columns sum to one, the sum of all the entries of the matrix must be the number of columns. Thus, the number of rows and the number of columns are the same number. This cheeky proof means that you cannot apply the doubly stochastic backboning to bipartite networks, unless $|V_1| = |V_2|$.

Doubly stochastic matrices have other fun properties. If you remember Section 11.1, the leading left eigenvector of a stochastic adjacency matrix is the stationary distribution, while the leading right eigenvector is a constant – assuming the graph is connected. In a doubly stochastic matrix, both the left and the right eigenvectors are equal to a constant or, in other words, the stationary distribution of a doubly stochastic matrix is constant. This isn't really a necessary thing to know while doing network backboning, but I thought it was cool, so do with this information what you will.

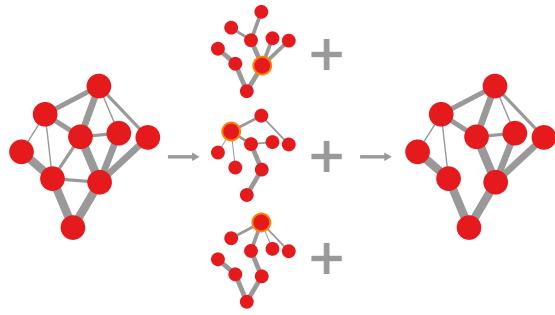
27.3 High-Salience Skeleton

The intuition behind the **high salience skeleton** (HSS) is that a network is a structure facilitating the exchange of information or goods. Thus, some connections are more important than others because they keep the network together in a single component. The main imperative is to allow all nodes to reach all other nodes in the most efficient and high-throughput way possible. Thus you need to interrogate each node and ask them what are the most efficient paths from their perspective. This cannot be done repurposing measures such as edge betweenness – whose objective is also telling us how structurally important an edge is (Section 14.2) – because these measures adopt a “global” point of view: they are the salient connections for the network *as a whole*, but they might leave some nodes poorly served.

To build an HSS we loop over the nodes and we build their shortest path tree: a tree originating from a node, touching all other nodes in the minimum number of hops possible and maximum amount of edge weight possible. In practice we start exploring the graph with a BFS and note down the total edge weight of each path. When we reach a node that we already visited we consider the edge weights of the two paths and the one with the highest one wins.

Note that we have the constraint of the structure originating from a node to be a tree. Thus it cannot contain a triangle. Consider Figure

[27.6](#) as an example. In the bottom example, we might want to save two edges at the same time. Our origin node, the one at the top of the network connects strongly with one node which also connects strongly to the node on the left. However, we cannot have both edges in the shortest path tree, as that would create a cycle. The final salience skeleton is allowed to have triangles and cycles, because it is the sum of all the shortest path trees.



We perform this operation for all nodes in the network and we obtain a set of shortest path trees. We sum them so that each edge now has a new weight: the number of shortest path trees in which it appears¹¹. The network can now be thresholded with these new weights.

Forbidding the creation of cycles in shortest path trees causes the main difference with the edge betweenness measure (Section [14.2](#)). One could think that the edges are simply sorted according to their contributions to all shortest paths in the network, but that is not the case. By forcing the substructures to be trees, we are counting the edges that are salient from each node's local perspective, rather than the network's global perspective. The authors in the paper show the subtle difference between shortest path tree counts and edge betweenness, also showing how a hypothetical skeleton extracted using edge betweenness performs more poorly.

The HSS makes a lot of sense for networks in which paths are meaningful, like infrastructure networks. However, it requires a lot of shortest path calculations – which makes it computationally expensive. Moreover, the edges are either part of (almost) all trees or of (almost) none of them. Figure [27.7](#) shows an example of this edge weight distribution, showcasing the typical “horns” shape of the HSS score attached to the original edges. You can see clearly that there are two peaks: one at zero – the edge is in no shortest path tree –; the other at one – the edge is part of all shortest path trees.

This can be nice, because it means HSS can be almost parameter free: the thresholding operation does not have many degrees of freedom. On the other hand, when there are few edges with weights

Figure 27.6: An example of High Salience Skeleton network backboning. The original graph is used to create a shortest path tree for each node in the network. In each tree, I highlight the focus node with the orange outline. The trees are then summed, and the result is new edge weights for the original graph that can be thresholded.

¹¹ Daniel Grady, Christian Thiemann, and Dirk Brockmann. Robust classification of salient links in complex networks. *Nature communications*, 3:864, 2012

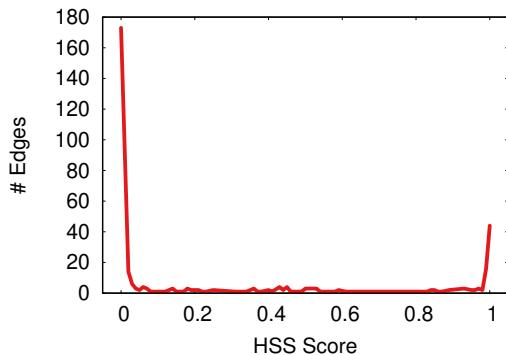
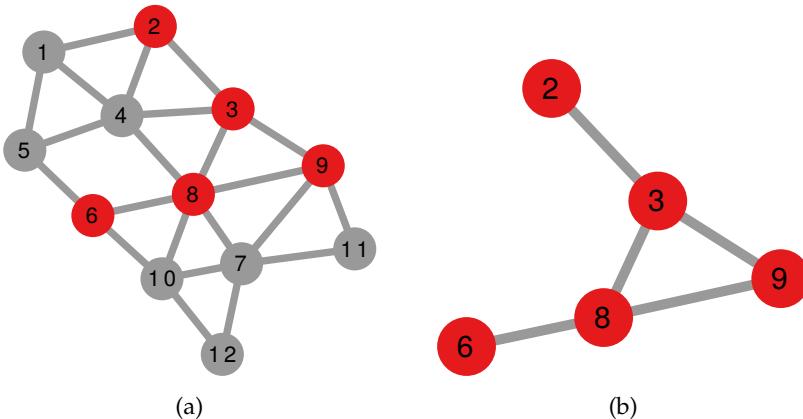


Figure 27.7: A typical “horns” plot for the edge weight distribution in HSS. The plot reports how many edges (y axis) are part of a given share of shortest path trees (x axis).

close to one your skeleton might end up being too sparse and it is difficult to add more edges without lowering the threshold close to zero.

27.4 Convex Network Reduction

A subgraph of a network G is convex if it contains all shortest paths existing in the main network G between its $V' \subseteq V$ nodes¹². We can expand this concept of convexity to apply to a full network G . To do so, we need to introduce the concept of “induced” subgraph: an induced subgraph is a graph formed from a subset of the vertices of the graph and all of the edges connecting pairs of vertices in that subset. Figure 27.8 shows an example of the inducing procedure.



¹² Frank Harary, Juhani Nieminen, et al. Convexity in graphs. *Journal of Differential Geometry*, 16(2):185–190, 1981

Figure 27.8: An example of induced graph. (a) The original graph. I highlight in red the nodes I pick for my induced graph. (b) The induced graph of (a), including only nodes in red and all connections between them.

A network is convex if all its connected induced subgraphs are convex. No matter which set of nodes you pick: as long as they are part of a single connected component, they are all going to be convex. This might look like a weird and difficult to understand concept, but you can grasp it with the help of elementary building blocks you already saw in this book.

Figure 27.9 shows the two basic alternatives for a convex network.

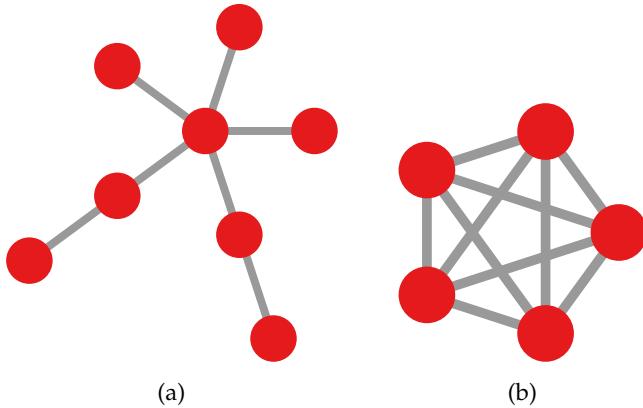


Figure 27.9: Two examples of convex networks. (a) A tree. (b) A clique.

In a tree – Figure 27.9(a) – any set of connected nodes is a convex subgraph. There are no other edges in G you can use to make shortcuts, because they'd create a cycle and trees cannot contain a cycle. A clique – Figure 27.9(b) – is a convex network as well: all possible connections are part of G , so picking any subset V' of nodes will also result in a clique. Since all nodes are connected to each other in a clique, you have all the shortest paths between them, making it convex.

You can build an arbitrary convex network by stitching together trees and cliques. In practice, it's just stitching together cliques, because the "tree-like" parts are nothing more than 2-cliques.

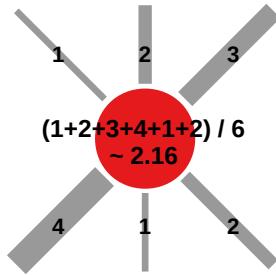
One could use the concept of convex networks to create a skeleton of a real world network¹³. Convex networks are almost impossible in nature, because adding a single edge to a tree or removing a single edge from a clique completely destroys convexity. However, one could make a real world network into a convex network by finding the minimal set of edges to remove to reduce the network into a tree of cliques. This is a possible way of backboning your network.

¹³ Lovro Šubelj. Convex skeletons of complex networks. *Journal of The Royal Society Interface*, 15(145):20180422, 2018

27.5 Disparity Filter

In this and the following section, we're slightly turning the perspective on network backboning. You could consider these as a different subclass of the problem. They all apply a general template to solve the problem of filtering out connections, which relate to the "noise reduction" application scenario of network backboning. Up until now, we adopted a purely structural approach which re-weights edges according to some topological properties of the graph. Here, instead, given a weighted graph, we adopt a template composed by three main steps: (1) define a null model based on node distribution properties; (2) compute a p-value (Section 3.3) for every edge to determine the statistical significance of properties assigned to edges

from a given distribution; (3) filter out all edges having p-value above a chosen significance level, i.e. keep all edges that are least likely to have occurred due to random chance.



The **disparity filter** (DF) is the first example in this class of solutions. It takes a node-centric approach. Each node has a different threshold to accept or reject its own edges. This is done by modeling an expected typical “node strength”, for instance the average of its edge weights. Then we keep only those edges which are higher than the expected edge weight for this node, making sure that this difference is statistically significant¹⁴. Figure 27.10 depicts a simplification of the method.

More precisely, the disparity filter defines u 's p-value for an edge u, v of weight $w_{u,v}$ as:

$$p((u,v), u) = \left(1 - \frac{w_{u,v}}{\sum_{v' \in N_u} w_{u,v'}}\right)^{(|N_u|-1)},$$

where N_u is the set of u 's neighbors (thus, $|N_u|$ is a fancy way to represent u 's degree). The original paper also shows how to calculate the expected edge weight and its variance, from which you can derive this p-value, but the procedure is a bit too convoluted to be included here. All you need, really, is the p-value. You can easily see that, if $|N_v| \neq |N_u|$ and/or $\sum_{v' \in N_u} w_{u,v'} \neq \sum_{v' \in N_v} w_{v,v'}$, the p-values for the same edge u, v will be different depending whether we focus on u or on v .

The disparity filter doesn't take into account that some nodes have inherently stronger connections. For instance, consider a mobility network, tracking commuters between cities in the United States. Figure 27.11 provides an example. New York has a lot of people and thus will have strong mobility links with any place in the US. In the disparity filter, edges are checked twice from both nodes' perspectives: few of New York's links are stronger than its average, but almost all of them are the strongest in the perspective of the smaller towns to which New York connects.

Figure 27.10: A schematic simplification of Disparity Filter network backboning. The node determines its customized threshold by building an expectation of its average connection strength. Every edge weight higher than this expectation in a statistically significant way is kept.

¹⁴ M Ángeles Serrano, Marián Boguná, and Alessandro Vespignani. Extracting the multiscale backbone of complex weighted networks. *Proceedings of the national academy of sciences*, 106(16): 6483–6488, 2009

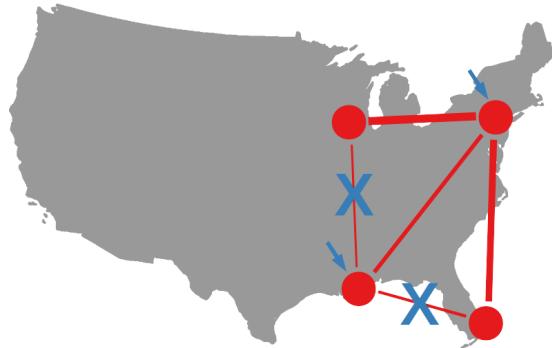


Figure 27.11: An example of hub dominance in the DF filtering schema. Edge thickness is proportional to the weight. We check the same edge from both perspectives (blue arrows).

We check New York against a small town in the south, for instance Franklington in Louisiana. Let's say that New York's connections, on average, involve 10k travelers. The traveler traffic with Franklington involves only 1k. This is way less than New York's average so, when we check this edge from New York's perspective, we mark it for deletion. However, on average, Franklington's connections involve only 500 travelers. Thus, when we check the edge from Franklington's perspective, we will find it significant and so we will keep it.

Since you need one success out of the two attempts to keep the edge, you end up with strong hubs connected to the entire network, and few peripheral connections (hub-spoke structure, or core-periphery, with no communities). In other words, the disparity filter tends to create networks with high centralization (Section 14.8), broad degree distributions, and weak communities. In many cases, that is fine. For some other scenarios, we might want to consider an alternative.

In summary, this means that the disparity filter ignores the weights of the neighbors of a node when deciding whether to keep an edge or not. There is a collection of alternatives^{15,16} that take this additional piece of information into account and are thus less biased.

In this section I explained the disparity filter only in the case of undirected networks. You can apply the same technique also for directed networks. In this case, you need to make sure that you're properly accounting for direction in your p-value calculation: the edge must be significant either when compared to the out-connections of the node sending the edge, or when compared to the in-connection weights of the node receiving it.

27.6 Noise-Corrected

The **noise-corrected** (NC) approach attempts to fix the issues of the disparity filter¹⁷. In spirit, it is very similar to it. However, the focus is shifted towards an edge-centric approach: each edge has a different

¹⁵ Navid Dianati. Unwinding the hairball graph: pruning algorithms for weighted complex networks. *Physical Review E*, 93(1):012304, 2016

¹⁶ Valerio Gemmetto, Alessio Cardillo, and Diego Garlaschelli. Irreducible network backbones: unbiased graph filtering via maximum entropy. *arXiv preprint arXiv:1706.00230*, 2017

¹⁷ Michele Coscia and Frank MH Neffke. Network backboning with noisy data. In *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*, pages 425–436. IEEE, 2017

threshold it has to clear if it wants to be included in the network. The assumption is that an edge is a collaboration between the nodes. It has to surpass the weight we expect given both nodes' typical connection strength. Again, we have to make sure that this difference is statistically significant. Figure 27.12 depicts a simplification of the method.

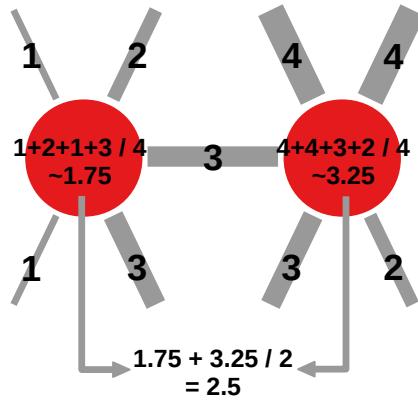


Figure 27.12: A schematic simplification of Noise Corrected network backboning. The edge determines its customized threshold by building an expectation of the average connection strength of its two nodes. If its weight is higher than the expectation in a statistically significant way then the edge is kept.

Formally speaking, the p-value of NC is calculated by looking at the CDF of a binomial distribution. The observed value (number of successes) is the weight of the edge $w_{u,v}$, the number of trials is the total sum of edge weights in the network $\sum_{u,v} w_{u,v}$, and the probability of success is given by:

$$p_{u,v} = \frac{\sum_{v' \in N_u} w_{u,v'} \times \sum_{u' \in N_v} w_{u',v'}}{\left(\sum_{u',v'} w_{u',v'} \right)^2}.$$

So, in practice, we're looking at the probability of having a weight higher than $w_{u,v}$ in a binomial distribution with $\sum_{u,v} w_{u,v}$ trials and a probability of success $p_{u,v}$. Given that we use a binomial as a null model, you can see that NC works only for discrete counts as edge weights, because the binomial is a discrete distribution (Section 3.2). Moreover, all the elements here ($w_{u,v}$, $\sum_{u,v} w_{u,v}$, and $p_{u,v}$) are the same in the perspective of u and v , thus this measure is u, v specific, differently from the disparity filter. Of course, if your network is directed, $w_{u,v} \neq w_{v,u}$ and you'll get a different null expectation for either direction of the edge, because the u, v edge is different from the v, u edge.

In the mobility network example I used before, a way to understand the difference between DF and NC is that, in NC, we require both nodes to agree to keep the edge. So, in this case, the edge between Franklington and New York will not be kept, because New

York voted for deletion. If you attempt to create backbones with the same number of edges, Franklington will end up connecting to its local neighborhood, because those edges are more likely to be agreed upon by all the smaller towns nearby our focus.

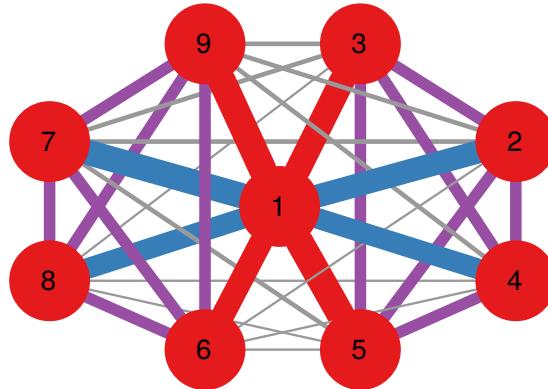


Figure 27.13: Different choices between DF and NC backboning. Edge width is proportional to its weight. Edge color: red = selected by both DF and NC; blue = DF only; purple = NC only; gray = neither.

Figure 27.13 abstracts from our geographical example to show the crux of the difference between DF and NC. DF favors the centralization around a hub. NC favors the horizontal peripheral connections which are the basis of the community structure of the network.

As you might expect, these methods exist because they give very different results. It is up to you to decide which of their assumptions best fits the network you are analyzing and the type of things you want to say about the network. A naive threshold fixes the same obstacle for all nodes no matter how strong, favoring the connections of the hub; HSS can include weaker links if they're the only path to a node; DF is similar to naive, but can recognize important weak edges; and NC overweights peripheries and communities: it is the most punishing method for the central hubs.

27.7 Summary

1. Backboning is the process of removing edges in a network. Reasons to do so span from a simple need of getting a sparser network, to facilitate computation on large networks, to the removal of connections that are not statistically significant. Most methods are developed assuming weighted networks, although you could apply some of them to unweighted networks as well.
2. One cannot simply establish a fixed threshold and remove all edges with a weight lower than the threshold. Edge weights usually distribute broadly and are correlated in different parts of the network, both factors that make the naive threshold approach not reasonable.

3. In doubly stochastic backboning, you transform the adjacency matrix in a doubly stochastic matrix (whose rows and columns sum to one) to break the local edge correlations. Such transformation is not always possible.
4. In high-salience skeleton, you calculate the short path tree for each node and you re-weight the edges counting the number of trees using them. Then you keep the most used edges. This is usually computationally expensive.
5. In disparity filter and noise corrected, you create a null expectation of the edge weight and keep only the ones whose weight is significantly higher than the expectation. This expectation is node-centric in the disparity filter and edge-centric in noise-corrected.

27.8 Exercises

1. Plot the CCDF edge weight distribution of the network at <http://www.networkatlas.eu/exercises/27/1/data.txt>. Calculate its average and standard deviation. NOTE: this is a directed graph!
2. What is the minimum statistically significant edge weight – the one two standard deviations away from the average – of the previous network? How many edges would you keep if you were to set that as the threshold?
3. Can you calculate the doubly stochastic adjacency matrix of the network used in the previous exercise? Does the calculation eventually converge? (Limit the normalization attempts to 1,000. If by 1,000 normalizations you don't have a doubly stochastic matrix, the calculation didn't converge)
4. How many edges would you keep if you were to return the doubly stochastic backbone including all nodes in the network in a single (weakly) connected component with the minimum number of edges?

28

Uncertainty & Measurement Error

To any person who has ever worked with real world data, it should come as no surprise that datasets are often disappointing. They contain glaring errors, incomprehensible omissions, and a number of other issues that make them borderline useless if you don't pour hours of effort into fixing them. The infamous 80-20 rule¹ that holds for literally everything (including degree distributions, as we saw in Section 9.4) will tell you that 80% of data science is just about cleaning data, and only 20% about shiny and fun analysis techniques. This obviously applies to network data as well. You'll find edges in your networks with incorrect weights and perhaps they shouldn't even be there, and you'll have plenty of missing or unobserved connections. And don't take my word for it: there are some works outside network science proper that also cite measurement error as one of the many things you should think about when working with networked data².

Admittedly, techniques to clean network data would deserve an entire book part but, frankly, there are surprisingly few techniques to deal with this problem. In fact, a survey paper about measurement error in network data³ points out that measurement error is routinely considered only a problem about missing data, rather than the more general framing as uncertainty. Network data cleaning is thus the lovechild of network backboning and link prediction, but that's a rather barren marriage – as far as I know. In fact, one of the few papers I know⁴ delivers the truth in a brutal and deadpan way: “[in network analysis] the practice of ignoring measurement error is still mainstream”. I hope this chapter will contribute to make things change.

To give you an idea of the significance of the measurement error blind spot in network science, consider the Zachary Karate Club network. As I'll explain in details in Section 53.4, everyone in our field is madly in love with this toy example. The paper presenting the network⁵ has been cited more than six thousand times – and not

¹ Vilfredo Pareto. *Manuale di economia politica con una introduzione alla scienza sociale*, volume 13. Società editrice libaria, 1919

² Arun Advani and Bansi Malde. Empirical methods for networks data: Social effects, network formation and measurement error. Technical report, IFS Working Papers, 2014

³ Dan J Wang, Xiaolin Shi, Daniel A McFarland, and Jure Leskovec. Measurement error in network data: A re-classification. *Social Networks*, 34(4):396–409, 2012a

⁴ Tiago P Peixoto. Reconstructing networks with unknown and heterogeneous errors. *Physical Review X*, 8(4):041011, 2018

⁵ Wayne W Zachary. An information flow model for conflict and fission in small groups. *Journal of anthropological research*, 33(4):452–473, 1977

everybody using this network cites it. The fun thing about this graph is that we actually don't know whether it has 77 or 78 edges. The bewildering thing about this graph is that *almost no one even mentions this problem!*

I start the chapter with a few methods that model measurement error to return a classical adjacency matrix, on which we can operate with classical algorithms. The rest of the chapter, instead, deals with probabilistic networks, structures whose edges have a probability of existing. Then, we want to define techniques to extract various things we care about – such as the degree distribution or the shortest paths – from a network containing edges that we're not certain they are there.

To wrap up this introduction, let's mention one obvious thing someone might want to do when they know their networks are affected by measurement error: to correct these errors! We'll see some techniques in this chapter, but in general one can use generic algorithm to find missing edges (link prediction, in Chapter 23), to throw away spurious edges (network backboning, in Chapter 27), or to find node duplicates. In the latter case, we can use network techniques to find such nodes, e.g. with node similarity (Section 15.2). However, you could also approach this task with non-network techniques such as entity resolution⁶ – e.g. figuring out that two nodes with different names "Sofia Coppola" and "Sophia Coppola" are actually referring to the same entity. This is a natural language processing task and therefore not of interest here.

28.1 Error Estimation & Correction

To be more concrete about errors in network, we can talk about some specific cases in which we have certified experience of such imperfections. For instance, there is a great deal of uncertainty when it comes to detect interactions between proteins in living organisms⁷. Some researchers have worked trying to reconstruct the physical backbone of the Internet⁸, but as technology evolves this task gets harder and harder. Finally, social networks are not immune to this problem⁹: people might answer to surveys incorrectly – because their memory is faulty –, and researchers can do it as well – as in the case of Zachary.

Let's organize all the ways in which measuring a network can go wrong. I have a graphical representation in Figure 28.1 of what could be happening.

- Missing nodes/edges: failing to report entities or connections between them that should have been there. This can be considered

⁶ Lise Getoor and Ashwin Machanavajjhala. Entity resolution: theory, practice & open challenges. *Proceedings of the VLDB Endowment*, 5(12):2018–2019, 2012

⁷ Shoshana J Wodak, Shuye Pu, James Vlasblom, and Bertrand Seéraphin. Challenges and rewards of interaction proteomics. *Molecular & Cellular Proteomics*, 8(1):3–18, 2009

⁸ Aaron Clauset and Cristopher Moore. Accuracy and scaling phenomena in internet mapping. *Physical Review Letters*, 94(1):018701, 2005

⁹ Peter V Marsden. Network data and measurement. *Annual review of sociology*, 16(1):435–463, 1990

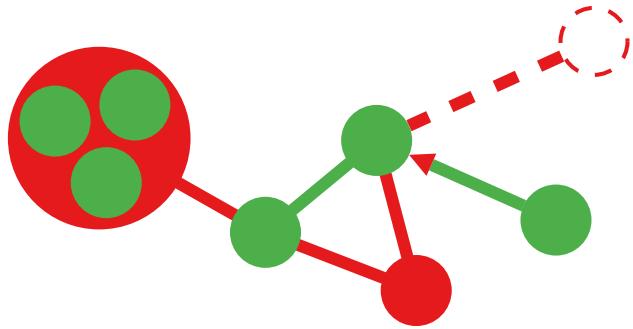


Figure 28.1: A network with all sorts of errors. Correct features are in green, errors are in red.

akin to having false negatives in your data. In Figure 28.1 you can see a node with a dashed red outline that we missed, and its corresponding red dashed edge – we missed that too;

- Additional nodes/edges: reporting entities or connections that don't actually exist. This is instead more similar to have false positives. In Figure 28.1, those would be the solid red nodes and edges;
- Duplicated nodes: an entity might exist, but it has been mistakenly split in two nodes. For instance, if you are making a network of actors, you might not know that the same actor might act under different pseudonyms. In Figure 28.1, the solid red node could be a duplicate of one of the green nodes;
- Aggregated nodes: it is possible to mistakenly think of a group of distinct entities as a single one. For instance, a music group might be reported as a single entity, even if we are interested in individual artists. In Figure 28.1, we have a big node with three green nodes in them: we didn't see the green nodes and mistakenly thought they were the red one (who appears to be very shocked by this);
- Misestimated node/edge features: the edge might exist, but in a weighted network we might be unsure about the actual strength of the relationship; in a directed network we may have gotten the direction wrong; or nodes and/or edge attributes might be incorrect. In Figure 28.1, there is a directed edge that does exist, but its arrowhead is red, because we got the direction wrong.

So what do you do when these things happen in your network? We're mostly focusing here on presence/absence of edges, because that's the case in which network techniques are most useful. As mentioned before, some of these problems can be tackled with non-network techniques, such as when you need to disambiguate nodes.

The basic way to go about estimating (and correcting) measurement error is by measuring network data multiple (random and independent) times¹⁰. This is a way to reconstruct a primary error estimate, i.e. to diagnose how good or bad our data collection is. Before going to networks, let's consider what you do with normal data, for instance estimating a quantity.

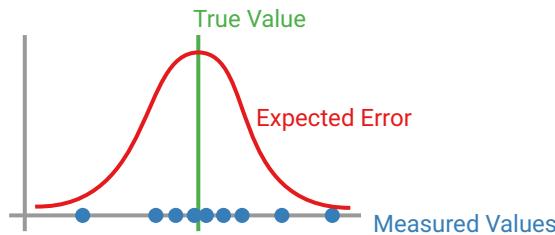


Figure 28.2 shows the usual scenario: you make a lot of measurements, none of them is the exact value you look for, but you can assume a Gaussian error. Once you know what sort of errors you might expect, you can do what one calls “maximum likelihood”¹¹ estimation: given the measurements I have and the expected distribution of the errors, what is the most likely true value? One powerful algorithm to deal with this problem is Expectation Maximization¹² (EM) – we already mentioned it for link prediction (Section 23.7) and we will encounter it again when talking about community discovery.

The same framework – maximum likelihood estimation via Expectation Maximization – can be done also for networks. There is nothing magical differentiating estimating an adjacency matrix A from a single number. It's just a little bit more complicated. One needs to introduce additional parameters – dealing with different types of errors I before dubbed false positives (extra edges) and false negatives (missing edges). The additional complexity is not a problem for the EM algorithm, and has the nice advantage that this framework is basically performing link prediction and network backboning at once.

Inferring directly the most likely adjacency matrix A is not necessarily the best thing you can do. If you focus on a specific measure of the network – say, the clustering coefficient – you can get its expected value with this method. The reason why this is better is because, this way, we can use alternative A s that might be slightly less likely than the most likely A , but still pretty likely. As a consequence, rather than having a single number that is the clustering coefficient of the most likely A , you get a distribution of potential values, each weighted by how likely their corresponding A is. The distribution would represent a more accurate and useful information than the simple estimate – since the most likely A could still be pretty unlikely

¹⁰ Mark EJ Newman. Network structure from rich but noisy data. *Nature Physics*, 14(6):542–545, 2018b

Figure 28.2: Estimating a true quantity (in green) with different measurements (in blue). In red we have the distribution of the expected errors. The x-axis represents the measurement value, the y-axis the likelihood of observing a given measurement value.

¹¹ The “likelihood” I mention here is the log-likelihood loss function I introduced in Section 4.3.

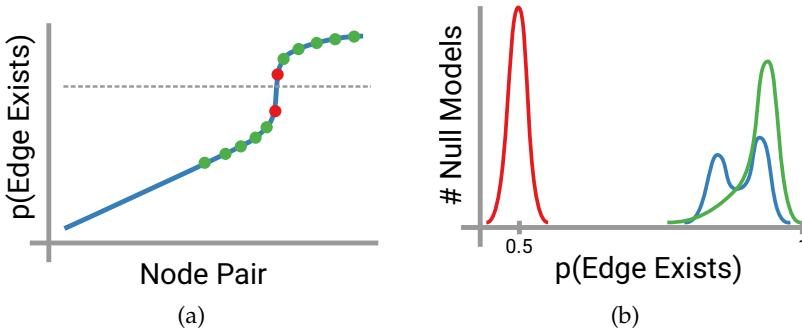
¹² Todd K Moon. The expectation-maximization algorithm. *IEEE Signal processing magazine*, 13(6):47–60, 1996

in absolute terms, and thus its clustering coefficient might be very wrong.

This all sounds awesome, but there are a couple of problems with this approach. First, we need to make an assumption about the errors on our measurement. Different networks will call for different error models for networks. Those models are available^{13,14}, but you're still going to be uncertain whether you have picked the right error model for your network.

Second, it is very very unlikely you're going to measure your network multiple times. The most likely scenario is that you will either measure it once, or that you're going to work with a network you find in the literature, also measured once and without any possibility of making a second measurement. Finally, it is extremely likely that, even if you can measure the same network twice, your measurements are not going to be independent. In a social setting, if people forget one of their social relationships, they're likely to forget it again if they are asked a second time. And that's a problem¹⁵.

Luckily there are ways out. For instance, in the paper I cited in the introduction of this chapter¹⁶, the author can build a clever Bayesian framework to find the most likely false positive and false negative edges. One still has to have a model, but instead of modeling the *errors* as we did above, we actually model the *signal*. We assume that the observed network fits a specific model and then we can use the model to figure out which edges are more or less likely to be unobserved or spurious.



You can use any of the network generative methods I showed in Part V. Any of those will allow to get edge probabilities. If you do things properly, you should see something like Figure 28.3(a). For each node pair you get a probability of them being connected. At some point, you'll find a discontinuity, transitioning from very low to very high existence probabilities. That's where you can put your threshold (above that threshold you expect the edge to exist, below you expect it not to exist. Any unconnected node pair above this

¹³ Carter T Butts. Network inference, error, and informant (in) accuracy: a bayesian approach. *social networks*, 25(2):103–140, 2003

¹⁴ Roger Guimerà and Marta Sales-Pardo. Missing and spurious interactions and the reconstruction of complex networks. *Proceedings of the National Academy of Sciences*, 106(52):22073–22078, 2009

¹⁵ Ye Yuan, Guoren Wang, Lei Chen, and Haixun Wang. Graph similarity search on large uncertain graph databases. *The VLDB Journal*, 24(2):271–296, 2015

¹⁶ Tiago P Peixoto. Reconstructing networks with unknown and heterogeneous errors. *Physical Review X*, 8(4):041011, 2018

Figure 28.3: (a) The probability (y axis) of a node pair (x axis) being connected given a network model. Node pairs are sorted from least to most likely. The gray horizontal dashed line represents the probability threshold for accepting/rejecting an edge hypothesis. Red points represent errors in the data according to the model. (b) Number of null models (y axis) producing a probability for an edge existing (x axis) for different network models: $G_{n,p}$ (red), configuration model (blue), stochastic blockmodel (green).

threshold (in red in the figure) is likely to be a missing edge. Any connected node pair below this threshold is likely to be a spurious edge (also in red in the figure). In green in the figure you see cases in which theory and data agree: expected edges actually there (above the dashed line) and unexpected edges that are indeed absent from the data (below the line).

However, if the model you choose is not a good representation of the data you have, your estimates are going to be way off. Figure 28.3(b) shows an example of this. Each model run will give you a distribution of probabilities for an edge existing. If the model is pretty bad – say a $G_{n,p}$ model (Chapter 16) – you’ll probably end up with a 50-50 chance of having or not having the edge, which is pretty useless. Better models fitting the data better, like the configuration model (Section 18.1) or stochastic blockmodels (SBMs, Section 18.2), will give more useful estimates. In the figure we’re testing for an edge we know it’s there, and thus these more accurate models have most of their guesses close to $p = 1$.

Of course, it’s not a given that these models will work better for your data. Different data will call for different models. So if you don’t find a good model fitting your data, you’re back at square one. The original paper uses a sophisticated Hierarchical Degree Corrected SBM¹⁷ (HDCSBM). HDCSBM is great, because it can model both (hierarchical) communities and the degree distribution. Still, it won’t be able to fit every single network and it may not be the best choice in any case, so you’re still left with the task of finding the best model for your network.

28.2 Probabilistic Networks

In the previous section we saw how to estimate errors and assign probabilities to edges. The papers I cited so far give you an idea on how to work with edge probabilities, assuming a model of the network or of the errors. In some cases, you might have neither. One example could be that the edges in your network come already with their own probabilities, and you don’t know the model that generated them. In this case, you have to use a probabilistic network.

A probabilistic network is a graph defined as $G = (V, E, \Pi)$. In this case, Π is a vector that assigns a probability of existence $p_{u,v}$ to each observed u, v edge in E . Normally, the probabilities in Π are all independent from each other – we’re going to work with this assumption here, but it’s not the only one: in some cases the edge probabilities should be dependent, as for instance in some cases edge failures might affect the local structure of the network¹⁸. Figure 28.4 provides an example.

¹⁷ Tiago P Peixoto. Nonparametric bayesian inference of the microcanonical stochastic block model. *Physical Review E*, 95(1):012317, 2017

¹⁸ Ye Yuan, Guoren Wang, Lei Chen, and Haixun Wang. Efficient subgraph similarity search on large probabilistic graph databases. *arXiv preprint arXiv:1205.6692*, 2012

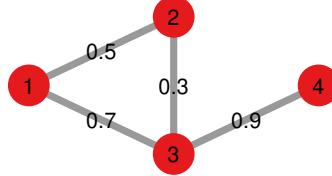


Figure 28.4: A probabilistic network. The edge label indicates the edge's probability of existing.

Of course the models can be more complex than simply attaching a probability to edges (what about nodes¹⁹? What about edge weights²⁰?), but to preserve our mental sanity we'll go with this simpler example and then it's up to you if you want to go down this specific rabbit hole. It's worth reiterating a difference between probabilistic network analysis and network backboning: in the latter we want to remove spurious edges, here we want to use all the information we have and integrate it in the analysis, no matter how unlikely an edge is to exist.

Probabilistic networks have been used in many different fields from mobility sensors²¹, to protein-protein interactions in biology²². They have been used to extend the k nearest neighbor algorithm to work with probabilities²³ and, as we'll see, to estimate centrality measures for road networks²⁴, and communities^{25,26} (of course).

The naive approach to deal with probabilistic networks would be to realize that each edge (u, v) can either exist or not. So you could create two “possible worlds”, which is to say two alternative networks, one with the edge and one without. The first has likelihood $p_{u,v}$ (because the edge exists) and the other has likelihood $1 - p_{u,v}$ (because the edge doesn't exist). These likelihoods act as a sort of weight. For instance, if you're absolute certain about the existence of an edge ($p_{u,v} = 1$) the network in which that edge doesn't exist isn't considered in the degree distribution calculation, because that possible world happens with probability zero.

The problem with this naive approach is that you need to do this for each edge. In a network with $|E|$ edges, you have $2^{|E|}$ possible worlds, with each possible combination of edges existing or not existing. This can (and will) get out of hand pretty quickly. Even the simple network in Figure 28.4(a) will generate the monstrous amount of possible worlds I show in Figure 28.5 – and here I omit all possible worlds with $p = 0$, for instance the one connecting only node 1 to node 4.

Looking at Figure 28.5, it is immediately obvious that one cannot compute all possible worlds for networks beyond a trivial size. Even if one could do it, it would still be a bad idea. What do you do when your analysis is by itself already expensive to calculate on a single network – as is the case, for instance, for the calculation of the

¹⁹ Yongxin Tong, Xiaofei Zhang, Caleb Chen Cao, and Lei Chen. Efficient probabilistic supergraph search over large uncertain graphs. In *CIKM*, pages 809–818, 2014

²⁰ Francesco Bonchi, Francesco Gullo, Andreas Kaltenbrunner, and Yana Volkovich. Core decomposition of uncertain graphs. In *SIGKDD*, pages 1316–1325, 2014

²¹ Xiaofeng Gao, Zhiyin Chen, Fan Wu, and Guihai Chen. Energy efficient algorithms for k -sink minimum movement target coverage problem in mobile sensor network. *IEEE/ACM Transactions on Networking*, 25(6):3616–3627, 2017

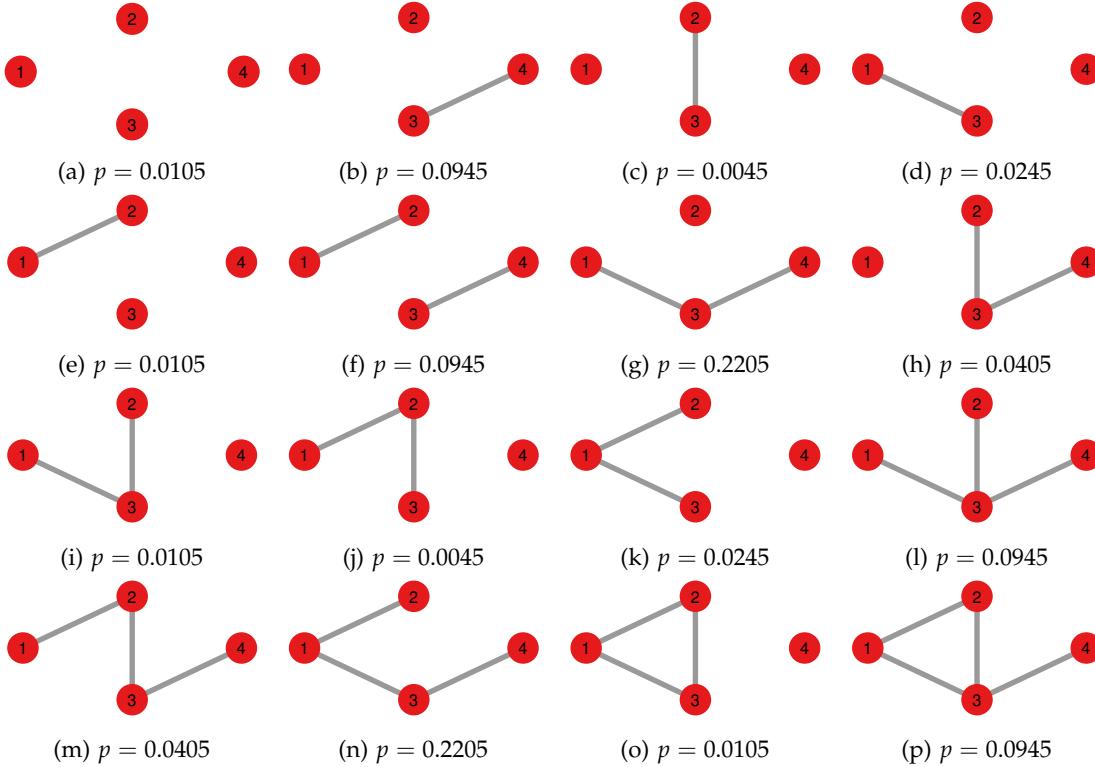
²² Sriganesh Srihari and Hon Wai Leong. A survey of computational methods for protein complex prediction from protein interaction networks. *Journal of bioinformatics and computational biology*, 11(02):1230002, 2013

²³ Michalis Potamias, Francesco Bonchi, Aristides Gionis, and George Kollios. K-nearest neighbors in uncertain graphs. *Proceedings of the VLDB Endowment*, 3(1-2):997–1008, 2010

²⁴ Takayasu Fushimi, Kazumi Saito, Tetsuo Ikeda, and Kazuhiro Kazama. A new group centrality measure for maximizing the connectedness of network under uncertain connectivity. In *Complex Networks and Their Applications VII: Volume 1 Proceedings The 7th International Conference on Complex Networks and Their Applications COMPLEX NETWORKS 2018 7*, pages 3–14. Springer, 2019

²⁵ George Kollios, Michalis Potamias, and Evmaria Terzi. Clustering large probabilistic graphs. *IEEE Transactions on Knowledge and Data Engineering*, 25(2):325–336, 2011

²⁶ Lin Liu, Ruoming Jin, Charu Aggarwal, and Yelong Shen. Reliable clustering on uncertain graphs. In *2012 IEEE 12th international conference on data mining*, pages 459–468. IEEE, 2012



average shortest path length? One solution would be to sparsify²⁷: if you have fewer probabilistic edges to begin with, you have much fewer possible worlds.

There is one way to sparsify that picks the edges to keep by minimizing entropy, which will give you the most representative possible worlds for the measure you want to calculate. The reason why you want to minimize entropy is that because you want to sample possible worlds using a Monte Carlo approach (we briefly mentioned it in Section 19.2): in this case, the lower the entropy the more accurate results you will get by sampling fewer possible worlds. However, this sampling approach is not general purpose: the edges you want to keep are going to be different depending on your objective – for instance clustering coefficient rather than average path length.

This leads to focus on specific problems we want to solve and see how specialized approaches can lead us to deal with uncertain graphs.

28.3 Specific Probabilistic Network Solutions

I will make a small deep dive into the following classical problems and one of their possible solutions in probabilistic networks:

Figure 28.5: Each possible world from the probabilistic network in Figure 28.4. The sub-captions report the probability of the possible world.

²⁷ Panos Parchas, Nikolaos Papailiou, Dimitris Papadias, and Francesco Bonchi. Uncertain graph sparsification. *IEEE Transactions on Knowledge and Data Engineering*, 30(12):2435–2449, 2018

- Node degree (for the classical version, see Chapter 9);
- Ego networks (which I formally introduce only in Section 30.1);
- Betweenness centrality (see Section 14.2 for a refresher);
- Connected components (which we deal with in Section 10.4);
- Finding the densest subgraph (similar to k-core decomposition – Section 14.7).

Some of these things we'll only see later in the book, so I'll give a brief context for the time being.

Node Degree

If you wanted to know the degree distribution of the network in Figure 28.4, what would you do? The naive approach would be to determine the expected degree of each node: go over all the possible worlds and take the average of the degrees of the node, weighted by how likely the world is to exist.

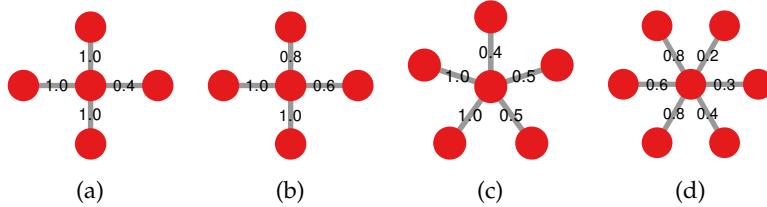


Figure 28.6: Probabilistic networks. The edge label reports an edge's probability of existing.

However, that would be a bit silly: the degree is a count, it shouldn't be a continuous number. The interpretability of what you'd do by averaging would go out of the window. To see why, consider Figure 28.6. The nodes in the figure have the same expected degree (3.4 – what the hell does it mean for a node to have degree equal to 3.4?), but their connection topologies are radically different.

It is much better to give each node a probability distribution for each possible degree value which allows you to differentiate between the nodes in Figure 28.6. In Figure 28.7(a), I focus on node 3 of Figure 28.4. Once you do all your weighted averages for all the possible worlds with their likelihoods, you'll end up with the correct degree distribution in Figure 28.7(b). To get to Figure 28.7(b) the method needs to take into account the whole probability distribution for the node, rather than a simple point estimate²⁸.

One way to do it is by counting how many nodes had a specific degree in each of the worlds in Figure 28.5 and then weight the probability of those discrete values with the likelihood of the world. Similar approaches can work, e.g., to perform core decomposition in

²⁸ Amin Kaveh, Matteo Magnani, and Christian Rohner. Comparing node degrees in probabilistic networks. *Journal of Complex Networks*, 7(5):749–763, 2019

²⁹ Francesco Bonchi, Francesco Gullo, Andreas Kaltenbrunner, and Yana Volkovich. Core decomposition of uncertain graphs. In *SIGKDD*, pages 1316–1325, 2014

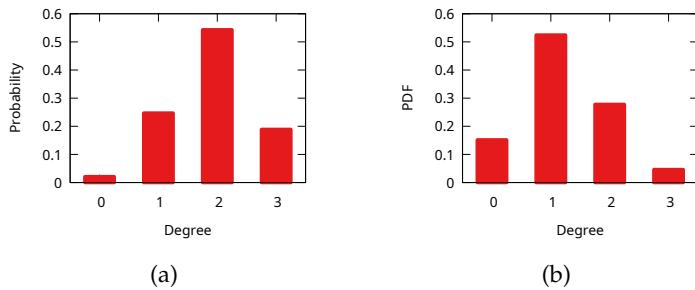


Figure 28.7: (a) The probability (y axis) for node 3 in Figure 28.4 to have a given degree (x axis). (b) The probabilistic degree PDF for the entire network in Figure 28.4.

probabilistic networks²⁹, since to perform core decomposition you need to know the degrees of the nodes (see Section 14.7).

Ego Networks

As we will see in Section 30.1, an ego network is a subset of a graph. The subset contains one node – the “ego” –, all of its direct neighbors and all of the connections between these nodes (ego + neighbors). When dealing with probabilistic networks, there are fundamentally two ways of extracting an ego network³⁰. You can generate first the possible worlds and then get the ego networks from each world separately (Figure 28.8) or you can extract the ego network directly from the probabilistic network and then apply specialized algorithms to calculate your measure of interest on the probabilistic ego network (Figure 28.9).

You might think this is a “Well, duh” moment, but I invite you to consider a few fun things that could happen. In an ego network, by definition, the ego is connected to all of its neighbors. Logically, ego networks must have a single connected component. If you go for the first option “possible worlds first, ego networks later,” both of these properties are guaranteed in your result. However, if you extract an ego network as a probabilistic network, when you then realize the various possible worlds you are likely to get a few neighbors that are not connected to the ego and you might even get disconnected components!

More in general, the results you’d get from these two approaches are highly correlated, but they’re not the same. So this choice matters. And having disconnected ego networks or an ego not connected to all of its neighbors can be a valid way to analyze your ego network, depending on the requirements you need to satisfy to answer your research question. So there isn’t necessarily a “best option” between the two.

²⁹ Amin Kaveh, Matteo Magnani, and Christian Rohner. Defining and measuring probabilistic ego networks. *Social Network Analysis and Mining*, 11:1–12, 2021a

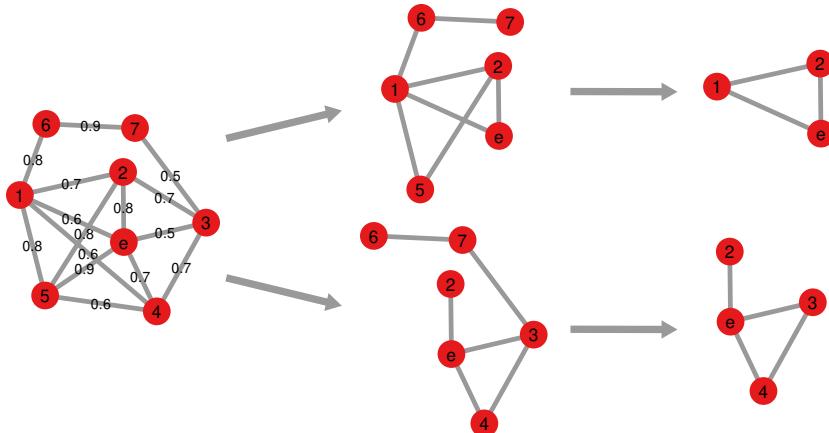


Figure 28.8: The process of extracting an ego network from a probabilistic network by first realizing the possible worlds and then extracting their ego networks. Ego marked with the “e” label.

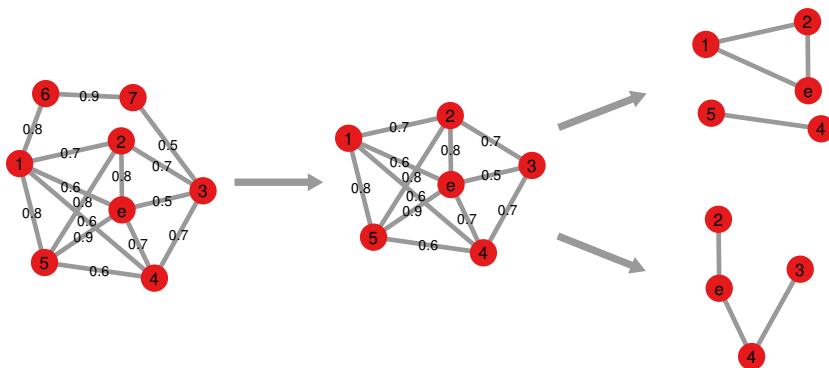


Figure 28.9: The process of extracting an ego network from a probabilistic network by first extracting a probabilistic ego network and then realizing its possible worlds. Ego marked with the “e” label.

Betweenness Centrality

Not everything is more difficult when working with probabilistic networks. Sometimes, you’re lucky. Betweenness centrality is one of those cases. You remember from Section 14.2 that betweenness centrality is the number of shortest paths passing through a node. In probabilistic networks, you have to weight each path by its probability of existing.

Since a path is a sequence of edges, and since each edge has a probability of existing, it follows that a long path is – under reasonable assumptions – less likely to exist than a short path. A path can only exist if all of the edges composing it exist. Therefore, its probability of existing is the product of all its edge probabilities. Here we assume that each edge’s existence probability is independent from all other edges, which is what people normally assume when working with probabilistic networks. And the probability of two independent events happening is the product of their probabilities – the probability of getting heads twice is the probability of getting heads once,

squared – see Section 2.3 for a refresher.

Imagine a probabilistic network where every single edge has 50/50 odds of existing. Any path of length 3 in this network exists with probability $p = (1/2)^3 = 0.125$. But a path of length 4 has $p = (1/2)^4 = 0.0625$ and one of length 5 has $p = (1/2)^5 = 0.03125$. As you can see, these probabilities keep halving. At some point, we can decide that the contributions from paths of length l is negligible, and simply ignore them. The resulting betweenness will be practically the same³¹. This speeds up computation quite a bit. On the other hand, the number of possible paths will go up, which might counteract the vanishing probabilities. Finding the right balance is something that probably depends on the probability distributions on your edges and there is no silver bullet.

³¹ Amin Kaveh, Matteo Magnani, and Christian Rohner. Probabilistic network sparsification with ego betweenness. *Applied Network Science*, 6:1–21, 2021b

Connected Components

In a deterministic network, when two nodes are part of the same connected component they are reachable: there exists a path following the edges of the network leading you from one node to another (see Section 10.4). This is not necessarily true in a probabilistic network. Once you think about this in terms of probabilistic networks, it is not hard to see why.

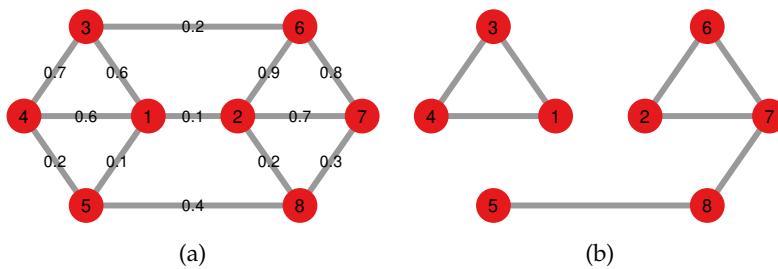


Figure 28.10: (a) A probabilistic network with edge probabilities as the edge's labels. (b) One of the possible worlds of (a).

Consider Figure 28.10(a). This is a probabilistic network in which nodes 1 and 2 are not only in the same component, they are directly connected to each other. However, many of the possible worlds of Figure 28.10(a) have nodes 1 and 2 in different components – Figure 28.10(b) is but one example.

In probabilistic networks, we can estimate a probability that two nodes are reachable – i.e. they are part of the same connected component. We call this probability ‘reliability’. The reliability of node pairs (1, 2) is simply the sum of the probabilities of all possible worlds in which they are in the same connected component^{32,33}.

³² Ruoming Jin, Lin Liu, and Charu C Aggarwal. Discovering highly reliable subgraphs in uncertain graphs. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 992–1000, 2011

³³ Arijit Khan, Francesco Bonchi, Aristides Gionis, and Francesco Gullo. Fast reliability search in uncertain graphs. In *EDBT*, pages 535–546, 2014

Densest Subgraph

In many applications, we might want to extract the densest subgraph of a network³⁴. As the name implies, the densest subgraph of a network is a subset of its nodes and the edges between them such that its density is the highest than the one of any other subgraphs in the network. You cannot find a subgraph denser than the one you extracted. Of course there are trivial solutions – for instance any connected node pair is a densest subgraph because it has density one – but one can specify a minimum number of nodes they are interested in, or a different measure to optimize such as the average degree.

Of course, there are ways to solve this problem in many different scenarios, for instance in streaming graphs as the data comes in little by little³⁵, but here we're interested in probabilistic networks specifically. The idea is, as usual, to try and define a measure of expected density for a subgraph, which is its density across all possible worlds weighted by their probability of existing. There are many methods to do so, and a few notable ones can also avoid making the assumption that the edge existence probabilities are independent³⁶.

One related problem is to find not the densest, but any specific subgraph instead of a large graph. This is something we will see in depth when we talk about subgraph mining (Chapter 41). For now, let's just say that you might have a specific pattern in mind and you want to know how many times it appears in the data. When you have a probabilistic network, you could instead ask for all patterns that are more likely to exist than a certain probability^{37,38}.

The potentially huge search space – a large graph has a lot of potential subgraphs – can be efficiently reduced. Let's take a look at Figure 28.11(a).

This probabilistic network is extremely simple – it's only a chain with four nodes and three edges. Even this ridiculously simple net-

³⁴ Andrew V Goldberg. Finding a maximum density subgraph. 1984

³⁵ Bahman Bahmani, Ravi Kumar, and Sergei Vassilvitskii. Densest subgraph in streaming and mapreduce. *Proceedings of the VLDB Endowment*, 5(5), 2012

³⁶ Zhaonian Zou. Polynomial-time algorithm for finding densest subgraphs in uncertain graphs. In *Proceedings of MLG Workshop*, 2013

³⁷ Zhaonian Zou, Hong Gao, and Jianzhong Li. Discovering frequent subgraphs over uncertain graph databases under probabilistic semantics. In *SIGKDD*, pages 633–642, 2010

³⁸ Odysseas Papapetrou, Ekaterini Ioannou, and Dimitrios Skoutas. Efficient discovery of frequent subgraph patterns in uncertain graph databases. In *EDBT*, pages 355–366, 2011

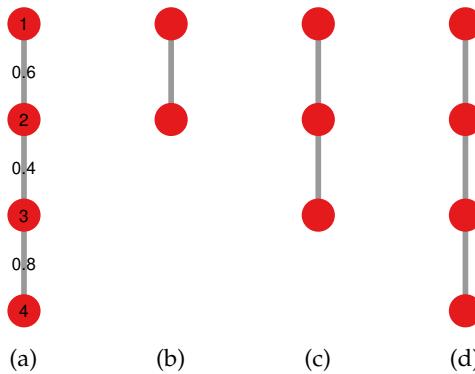


Figure 28.11: (a) A probabilistic network with edge probabilities as the edge's labels. (b-d) Some subgraphs that can be extracted from (a). They exist with probabilities: (b) $p = 0.8$, (c) $p = 0.32$, (d) $p = 0.192$ – the probability of a pattern is the maximum combined probability of the set of edges that can form the pattern.

work already has a surprisingly high number of possible subgraphs. However, Figures 28.11(b-d) shows a subset of them, and it also shows that the probabilities of these patterns existing goes down quite dramatically with their size (also discussed in Section 28.3). A larger pattern has a strictly lower probability of existing than a smaller pattern – basically the product of all its edges' probabilities, if these probabilities are all independent. If you say that you only want patterns with 25% probability of existing or more, the number of potential subgraphs you need to check in Figure 28.11(a) becomes more sane.

28.4 Alternatives to Probability Theory

As I discuss in Chapter 2, probability theory is not the only game in town when confronted with uncertainty. One could use Dempster-Shafer's theory of evidence (DST) or fuzzy logic (Section 2.8 – go back and refresh your understanding of these two approaches, or the rest of this section won't make much sense). DST has been used to classify³⁹ and predict⁴⁰ edges. Besides shortest path distance⁴¹ – as we will see in a moment – fuzzy logic has also been applied to the problem of link prediction⁴² and to estimate the centrality of nodes⁴³. There is also uncertainty theory⁴⁴, which has been used for transportation planning⁴⁵ and shortest paths⁴⁶, but I won't treat it here.

For the sake of simplicity, I'll focus on how DST and fuzzy logic will represent a problem differently than probability theory and lead to different results. I'll limit the discussion to the estimation of the path distance between two nodes. Figure 28.12 shows three representations of the same network with uncertain edge weights. So what is the distance between nodes 1 and 3 – counting the weights of the edges?

Let's start with probability theory – Figure 28.12(a). There are four possible worlds out there: both weights could be 1 (with $p = 0.2 \times 0.5 = 0.1$), then they could be 1-2 (with $p = 0.2 \times 0.5 = 0.1$), 2-1 (with $p = 0.8 \times 0.5 = 0.4$), or both 2 (with $p = 0.2 \times 0.5 = 0.4$). So probability theory will tell you that the distance between node 1 and node 3 has a probability distribution: it could be 2 with $p = 0.1$, it could be 3 with $p = 0.5$, or it could be 4 with $p = 0.4$. Splendid.

Onto DST, then. Looking at Figure 28.12(b) we see that we now have also to deal with the fact that we might not be able to prove that the edge weight is either 1 or 2. We know those are the only two possibilities, though, so we need to estimate the belief and plausibility of the weight being 2 – for brevity's sake I'll do the math only for the first edge. The Belief of the 1,2 edge having weight 2 is

³⁹ Salma Ben Dhaou, Mouloud Kharoune, Arnaud Martin, and Boutheina Ben Yaghane. A belief approach for detecting spammed links in social networks. In *International Conference on Agents and Artificial Intelligence*, 2019

⁴⁰ Sabrine Mallek, Imen Boukhris, Zied Elouedi, and Eric Lefèvre. Evidential link prediction in social networks based on structural and social information. *Journal of computational science*, 30: 98–107, 2019

⁴¹ Tiago Simas and Luis M Rocha. Distance closures on complex networks. *Network Science*, 3(2):227–268, 2015

⁴² Behnaz Moradabadi and Mohammad Reza Meybodi. Link prediction in fuzzy social networks using distributed learning automata. *Applied Intelligence*, 47:837–849, 2017

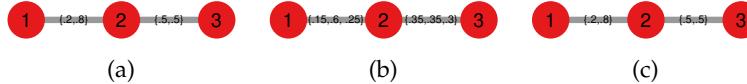
⁴³ Ren-Jie Hu, Qing Li, Guang-Yu Zhang, and Wen-Cong Ma. Centrality measures in directed fuzzy social networks. *Fuzzy Information and Engineering*, 7(1):115–128, 2015

⁴⁴ Baoding Liu. *Uncertainty theory*. Springer, 2010

⁴⁵ Yuan Gao, Lixing Yang, and Shukai Li. Uncertain models on railway transportation planning problem. *Applied Mathematical Modelling*, 40(7-8):4921–4934, 2016b

⁴⁶ Yuhong Sheng and Xuehui Mei. Uncertain random shortest path problem. *Soft Computing*, 24:2431–2440, 2020

0.6 – equal to its Mass since this set has no subsets. The Plausibility is $1 - \text{Belief}(1) = 0.85$. Following the literature⁴⁷, we can define the edge bounds as follows: $1 + [1 \times 0.6, 1 \times 0.85] = [1.6, 1.85]$ – i.e. the minimum weight we think the edge has is 1.6 and the maximum is 1.85 .



Why did we aggregate this way? As I mentioned, we take for granted that the edge weight must be at least 1 , so that's why we start with " $1+$ ". Then, $\text{Belief}(2)$ – which is 0.6 – tells us how much we can prove there is an additional cost of 1 . So the total cost cannot be anything less than 1.6 . $\text{Belief}(1)$, instead, tells us how much we can prove the edge weight is not 2 (0.15) which means we can prove that the weight cannot be any higher than 1.85 .

Applying the same reasoning to the $2, 3$ edge leads us with an interval of $[1.35, 1.65]$. The final cost of the full path must then be within the interval $[1.6, 1.85] + [1.35, 1.65] = [2.95, 3.5]$. Our available evidence tells us that the path costs at least 2.95 to traverse, but not more than 3.5 .

Just like with DST and probability theory, there are multiple ways to use fuzzy logic⁴⁸, so here I'll just pick one as an example of the different results different framework can give. Since the edge weights are fuzzy, the length of the shortest path $1 \rightarrow 2 \rightarrow 3$ is fuzzy as well. It can belong to three classes: length 2 (if both edges have weight 1), length 3 (if one edge has weight 2), and length 4 (when both edges have length 2). To find out to which degrees the path belongs to these length classes we need to work backwards. First we ask the distance of 3 with itself, which can only be 0 , by definition.

Then we need the distance between node 2 and node 3 , which is 1 with degree 0.5 and 2 with degree 0.5 – because those are the classes of the $2, 3$ edge and node 3 contributes zero. We write this as $\{1/0.5, 2/0.5\}$.

We apply the same reasoning to get the length of $1 \rightarrow 2 \rightarrow 3$. The $1, 2$ edge contributes $\{1/0.2, 2/0.8\}$ to what we already had from the edge $2, 3$. We need to add up all the possible combinations and taking the minimum belongings. The path can be of length 2 only if both edges weights are 1 , so its belonging in the minimum of the two (see Section 2.8) which is 0.2 . The path can be of length 3 if the weights are either $2 - 1$ (which is $\min(0.8, 0.5) = 0.5$) or $1 - 2$ (which is $\min(0.2, 0.5) = 0.2$). In fuzzy logic, we need to take the maximum between the two, which is 0.5 . Finally, to be of length 4 both edges need to belong to the weight 2 class, and that's $\min(0.8, 0.5) = 0.5$. So

⁴⁷ Gabor Szucs and Gyula Sallai. Route planning with uncertain information using dempster-shafer theory. In *2009 International Conference on Management and Service Science*, pages 1–4. IEEE, 2009

Figure 28.12: The edge labels represent the uncertain edge weights, using different models. (a) Probability theory: the probability of the edge being of weight 1 and 2 . (b) DST: Mass values of edge weights $\{1\}$, $\{2\}$, and $\{1, 2\}$. (c) Fuzzy logic: degree of belonging to the classes of weight 1 and 2 .

⁴⁸ Cerry M Klein. Fuzzy shortest paths. *Fuzzy sets and systems*, 39(1):27–41, 1991

the final result is $\{2/0.2, 3/0.5, 4/0.5\}$.

As you can see the three theories give three very different results. The differences are not only quantitative, but also qualitative: you get results that are truly incompatible with each other – a distribution, an interval, a set of classes.

28.5 Summary

1. Network data is not immune to measurement error. There could be missing or extra edges/nodes, and incorrectly reported features such as edge direction, weight, or any sort of edge/node attribute.
2. To correct measurement errors you need to have a model of your errors. If you know the type of errors you are getting, then you can use statistical methods to infer the most likely true value, based on the measurements you observe. This, however, requires to measure your network multiple times – which is unusual in network data.
3. One can sidestep the requirement of measuring the network multiple times by having an accurate model of the generating process of the network. Then one can generate many synthetic versions of the observed data, as if they were multiple measurements. The downside being that one now has to have an accurate model of the network, which can be tricky.
4. If one has data about the probability of existence of an edge, then they can work directly with probabilistic networks – network whose edges have probability values. To analyze them, one has to create all the possible networks – which are $2^{|E|}$ because each edge can exist or not exist. Then they can calculate a measure of interest in all possible worlds and average it with the probabilities of each possible world.
5. Since there are so many possible worlds, there are specialized techniques to reduce the computational complexity and estimate the probability distribution of given network measures such as the degree, betweenness centrality, connected components, and more.
6. One can work with alternative approaches to deal with uncertainty and create different types of networks with uncertain data, which will give different interpretations (and results) with the same data.

28.6 Exercises

1. Consider the network at <http://www.networkatlas.eu/exercises/28/1/data.txt>. This is an undirected probabilistic network with

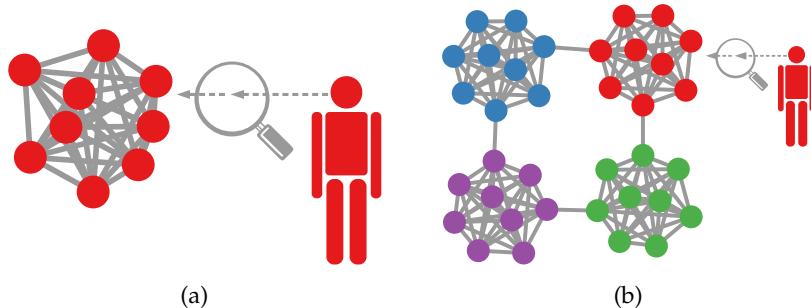
four columns: the two connected nodes, the probability of the edge existing and the probability of the edge non existing. Generate all of its possible worlds, together with their probability of existing. (Note, you can ignore the fourth column for this exercise)

2. Calculate the probabilistic degree distribution of the network used in exercise 1. (Note, you can ignore the fourth column for this exercise)
3. Estimate the reliabilities of node 4 from the previous network with each of the other nodes in the network. (Note, you can ignore the fourth column for this exercise)
4. Calculate the length of the shortest path between node 2 and node 4 in the previous network using fuzzy logic, assuming that the third column reports the probability of the edge to have weight 1 and the fourth column reports the probability of the edge to have weight 2.

29

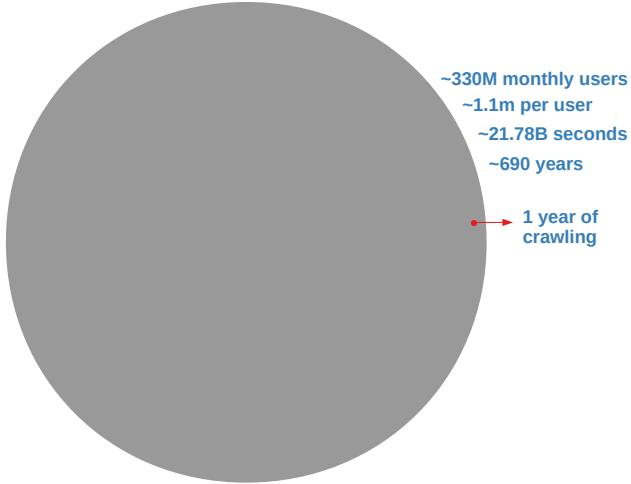
Network Sampling

Sometimes, having a good edge induction or network backboning technique still doesn't help you. Sometimes you're observing a network directly and it's just a hairball. In these cases, it's useful to make a step back and consider that the act of observation in itself is not neutral. We decide what to focus on, whether we do it because of our interests, or simply because of data availability. If we could zoom out, we would see the structure, as Figure 29.1 shows. When you're unable or unwilling to look at the entire network you have to perform network sampling.



"Network sampling" means to extract from your network a smaller version of it. This smaller version, the sample, should be a representative subset of the data. By "representative" we mean that the property you're interested in studying should be more or less the same in the sample as in the network at large. For instance, if you're interested in estimating the clustering coefficient, extracting the only triangle from a large network which otherwise has none wouldn't be a good sampling. The sample's clustering is one, while the network at large has a clustering approaching to zero. Put it in other words, a proper network sampling will ensure that the tiny sliver you observe is carrying the properties of the whole structure you're interested in. Sampling can be extremely important also in new emerging scenarios, as it is fundamental for some types of graph neural networks to be able to deal with gigantic graphs (Section 45.5).

Figure 29.1: A representation of the sampling conundrum. (a) The sample you're able to observe looks like a hairball, with everything connected with everything else. (b) Zooming out to the whole structure shows a different story, with a clear community structure we could not observe due to the improper sample.



To put in perspective how bad the problem is, consider Twitter. As of writing this paragraph, Twitter has more than 300 million active users. According to its API, it takes a bit more than a minute on average to fully know the connections of a user. This means that it takes more than 20 billion seconds to crawl the entirety of Twitter, or just a bit less than 700 years. If you were to crawl constantly for one year, you'd get a bit more than 0.1% of Twitter. If you're a visual thinker, Figure 29.2 shows a depiction of the fact I just narrated. You can understand that what ends up in your 0.1% has to be the best possible representation of the whole, and thus it has to be chosen carefully.

We already saw some ways to explore a graph: BFS and DFS (Section 13.1). They are reasonable ways to explore a graph, but their underlying assumption is that, eventually, they will cover the entire network. Here we focus on a slightly different perspective. We don't want the entire network: we want to prevent biases to creep into our sample.

We can classify network sampling strategies – in the broadest terms possible – as induced and topological techniques. These are the focus of the next sections. What I'm writing is based on review works on network sampling^{1,2,3,4}. I'm going to mostly focus on the case in which the sampled network is stable, or it is evolving too slowly to make any significant difference during the sampling procedure. There are specialized methods to sample graphs when this assumption is not true. For instance streaming or evolving graphs, whose properties might significantly change as you explore them^{5,6,7}.

Nowadays, you rarely want to sample a large graph that you fully own. We have enough computing and storing capabilities to

Figure 29.2: The gray circle represents the set of users in Twitter. Given the platform's API constraints, a non-stop one-year crawl of the Twitter network would yield the set of nodes encompassed by the red circle.

¹ Minas Gjoka, Maciej Kurant, Carter T Butts, and Athina Markopoulou. Walking in facebook: A case study of unbiased sampling of osns. In *2010 Proceedings IEEE Infocom*, pages 1–9. Ieee, 2010

² Minas Gjoka, Maciej Kurant, Carter T Butts, and Athina Markopoulou. Practical recommendations on crawling online social networks. *IEEE Journal on Selected Areas in Communications*, 29(9):1872–1892, 2011

³ Anirban Dasgupta, Ravi Kumar, and D Sivakumar. Social sampling. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 235–243. ACM, 2012

⁴ Neli Blagus, Lovro Šubelj, and Marko Bajec. Empirical comparison of network sampling techniques. *arXiv preprint arXiv:1506.02449*, 2015

⁵ Daniel Stutzbach, Reza Rejaie, Nick Duffield, Subhabrata Sen, and Walter Willinger. Sampling techniques for large, dynamic graphs. In *Proceedings IEEE INFOCOM 2006. 25TH IEEE International Conference on Computer Communications*, pages 1–6. IEEE, 2006

⁶ Amir H Rasti, Mojtaba Torkjazi, Reza Rejaie, D Stutzbach, N Duffield, and W Willinger. Evaluating sampling techniques for large dynamic graphs. *Univ. Oregon, Tech. Rep. CIS-TR-08*, 1, 2008

⁷ Nesreen K Ahmed, Jennifer Neville, and Ramana Komella. Network sampling: From static to streaming graphs. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 8(2):7, 2014

process humongous structures. The case is different when you rely on an external data source. Most of the times, such data source will be a large social media platform. In this scenario, one has to apply double carefulness. API-based sampling is affected by fundamental issues. Works in the past have shown that one has to be careful when working with data sources that potentially yield non-representative samples of the phenomenon at large^{8,9}.

Note that you are not the only person in the world performing network sampling. In most cases, you're going to work with data that has already been sampled by somebody else and you have no control over how they extracted that sample from reality. This is true also if you're convinced that you are at the data source itself, for instance the API of the social media platform. But then you should ask yourself a few questions. Who has decided to use the platform? Who is active and who is present but inactive? What data does the provider make available? In such cases, you might need to carefully consider what you do with the data and/or decide to perform a network completion process (Section 29.5) – if it is possible at all.

29.1 Induced

Induced sampling works with a guiding principle. You specify a set of elements that must be in your sample. Then, you collect all information that is connected to the elements you selected¹⁰.

This is related, but not the same thing as, the concept of induced subgraph, a graph formed from a subset of the vertices of the graph and all of the edges connecting pairs of vertices in that subset – see Section 27.4. When performing an induced subgraph, you only focus on nodes, and you won't obtain new nodes from your induction procedure. When performing induced samples, instead, you usually want to add nodes to your sample as well, besides edges.

Differently from simply making an induced graph, you can do induced sampling in two ways: by focusing on nodes or by focusing on edges.

Node Induced

If you focus on nodes, it means that you are specifying the IDs of a set of nodes that must be in your sample. Then, usually, what you do is collecting all their immediate neighbors. The issue here is clearly deciding the best set of node IDs from which to start your sampling. There are a few alternatives you could consider.

The first, obvious, one is to choose your node IDs completely at random. Random sampling is a standard procedure in many

⁸ Fred Morstatter, Jürgen Pfeffer, Huan Liu, and Kathleen M Carley. Is the sample good enough? comparing data from twitter's streaming api with twitter's firehose. In *Seventh international AAAI conference on weblogs and social media*, 2013

⁹ Fred Morstatter, Jürgen Pfeffer, and Huan Liu. When is it biased?: assessing the representativeness of twitter's streaming api. In *Proceedings of the 23rd international conference on world wide web*, pages 555–556. ACM, 2014

¹⁰ Jure Leskovec and Christos Faloutsos. Sampling from large graphs. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 631–636. ACM, 2006

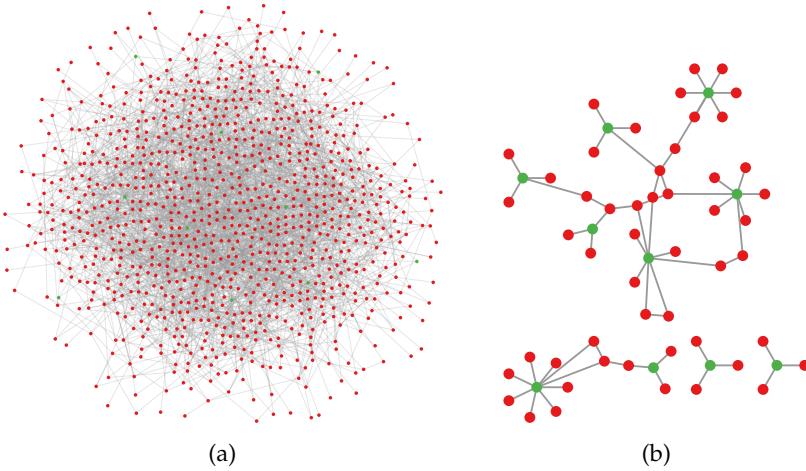


Figure 29.3: (a) A graph with a thousand nodes. I select uniformly at random 1% of the nodes, in green. I then induce a graph with the selected nodes, all their neighbors, and all connections between them. (b) The resulting node-induced graph.

other scenarios, and has its advantages. If the properties you’re interested in studying are normally distributed in your population, a large enough random sample will be representative. However, when it comes to real world networks, such expectation might not be accurate. For two reasons.

First, if your network is large – and if your network isn’t large why the heck are you sampling it? – choosing node IDs at random might end up reconstructing a disconnected sample. The likelihood of two random nodes – or their neighbors – being connected is stupidly low. Figure 29.3 shows an example of this issue. Even with a very generous 1% random node sampling – which, in the Twitter example I made earlier, would mean three million nodes! – the resulting node-induced graph breaks down in multiple components. This might not be a problem but, usually, large social networks are connected. Thus ending up with a disconnected network, by definition, will mean that you don’t have a representative sample.

Second, one of the properties most network scientists are interested in is the degree distribution. The degree distribution is emphatically not distributed normally in your population (Section 9.3). Thus, a random node-induced sample is unlikely to fairly represent the hubs in your network.

Standard solutions for these two issues are simple. One can weight their samples. Nodes are more likely to be extracted and be part of the sample if they have a higher degree or PageRank. However, this requires knowing this information in advance, which is not feasible if you’re crawling your network from an API system.

Edge Induced

Another way to generate induced samples is to focus on edges rather than nodes. This means selecting edges in a network and then crawl

their immediate neighbors. There are a few techniques to do so. One is the obvious extension of random node induced sampling: random edge induced sampling. You select edges at random and you collect all their direct neighbors. Two more sophisticated approaches are Totally Induced Edges Samples (TIES)¹¹ and Partially Induced ones (PIES)¹².

The idea behind edge sampling is that it counteracts the downward bias when it comes to the degree. In a network with a heavy-tailed degree distribution, most nodes have a low degree. Thus, if you pick one at random, it's overwhelmingly likely that it will be a low degree node. On the other hand, most edges are attached to large hubs. Thus, if you pick an edge at random, it is likely that a hub will be attached to it. This is a similar consideration of the vaccination strategy we saw in Section 21.3.

There is an obvious downside to the edge sampling technique. You cannot easily use it when interfacing yourself with a social media API system. Very rarely such systems will allow you to start your exploration from a randomly selected edge. Thus, in one way or another, you're always going to perform some form of node-induced sampling.

29.2 Topological Breadth First Search Variants

The alternative to induced sampling is topological sampling. In topological sampling you also start from a random seed, but then you start exploring the graph. You're not limited to the immediate neighborhood of your seed as in the induced sampling, but you can get arbitrarily far from your starting point. That is why one of the key differences between induced sampling and topological sampling is the seed set size. In induced sampling you have to have the largest possible seed set, while in topological sampling you can start from a single seed and explore from there.

One of the key advantages of topological sampling is that it works well with API systems. There is also research showing that topological sampling is, in general, less biased than induced sampling¹³. If used for sampling purposes, DFS and BFS graph exploration fall into this category.

There are fundamentally two families of topological sampling. The first is a modification of the BFS approach. The idea is to perform a BFS, but then adding a few rules to prevent some of the issues affecting that strategy. This is what we focus on in this section. The second big family is based on random walks and it will be the topic of the next section. Note that this division is largely arbitrary, as there is cross-pollination between these two categories, but it is a

¹¹ Nesreen Ahmed, Jennifer Neville, and Ramana Rao Kompella. Network sampling via edge-based node selection with graph induction. 2011

¹² Nesreen K Ahmed, Jennifer Neville, and Ramana Kompella. Space-efficient sampling from social activity streams. In *Proceedings of the 1st international workshop on big data, streams and heterogeneous source mining: algorithms, systems, programming models and applications*, pages 53–60. ACM, 2012

¹³ Sang Hoon Lee, Pan-Jun Kim, and Hawoong Jeong. Statistical properties of sampled networks. *Physical Review E*, 73(1):016102, 2006

useful way to organize this chapter.

Snowball

In **Snowball** sampling we start by taking an individual and asking her to reveal k of her connections^{14,15}. She might have more than k friends, but we only take k . Then, we use these new individuals and we ask them the same question: to name k friends. We do so with a BFS strategy. In practice, Snowball is BFS, but imposing a cap in the number of connections we collect at a time: k . Figure 29.4 depicts the process. Note that k is not the maximum degree of the network, because a node might be mentioned by more than k neighbors, if they have them.

¹⁴ Leo A Goodman. Snowball sampling. *The annals of mathematical statistics*, pages 148–170, 1961

¹⁵ Patrick Biernacki and Dan Waldorf. Snowball sampling: Problems and techniques of chain referral sampling. *Sociological methods & research*, 10(2): 141–163, 1981

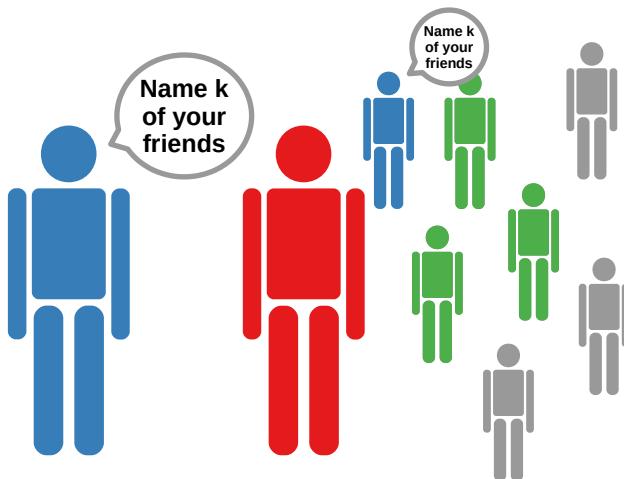


Figure 29.4: Snowball sampling. Your sampler (blue) starts from a seed (red) and asks for $k = 3$ connections. Red names their green friends, but not the gray ones. The interviewer then recursively asks the same question to each of the newly sampled green individuals. If no one ever mentions the gray ones, those are not sampled and won't be part of the network.

Snowball has some advantages. It is cheap to perform in the real world, where the cost of identifying nodes is high, because the nodes identify themselves as a part of the survey process. This is less relevant for social media, where node discovery is relatively easy. Snowball has a smaller degree bias: with the “nominate-a-friend” strategy we’re likely to encounter hubs. However, their degree is somewhat capped, since they can only name k of their friends, rather than the full list. This generates weird degree distributions with a sharp cutoff, which aren’t very realistic.

When it comes to sampling from social media, Snowball has a surprising advantage. It works well with pagination: in API systems, when you ask the connections of a node, you rarely get all of them. Social media *paginate* results, so you only get k connections at a time. With Snowball you can easily decide the maximum number of pages you want.

Forest Fire

In **Forest Fire**, like in Snowball, the base exploration is a BFS. However, once we get all neighbors of a node, we do not explore them all. Instead, for each of them, we flip a coin and we explore the node only with probability p . The advantage of Forest Fire is usually linked with a proper estimation of the clustering coefficient of the network, since with a BFS we would overestimate it – because we fully explore the neighborhood of nodes¹⁶.

¹⁶Jure Leskovec and Christos Faloutsos. Sampling from large graphs. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 631–636. ACM, 2006.

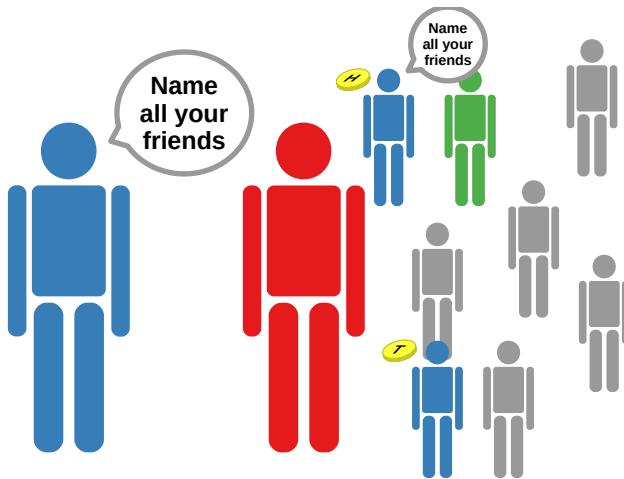


Figure 29.5: Forest fire sampling. Your sampler (blue) starts from a seed (red) and asks for all the connections a node. If the probability test succeeds, the neighbor turns green and is also explored. If it fails, the neighbor remains gray and is not explored further.

Figure 29.5 provides an example. After sampling a node and getting all its neighbors, we continue the BFS exploration. But, before sampling the neighbors of a neighbor, we flip a coin. If the test fails, we skip the neighbor and we go to the next one. Usually, one won't try to visit again the neighbors that have been skipped.

Forest Fire has an interesting relationship with your sampling budget. Usually, you're in a scenario in which you have a limited amount of resources to gather your network – normally, the time it takes to perform the crawl. Assuming your network is sufficiently large, all sampling methods seen so far will eventually use up all your budget. However, if you set p sufficiently low, you might end up in a situation where your Forest Fire crawl ends before you used up your budget. In this case you have to decide whether you want to stop your crawl and forgo the rest of your budget, or you're allowed to re-visit skipped nodes. The decision should be made depending on what's most important to keep: if the sample's topological properties are paramount, you cannot re-visit skipped nodes and you will have to make peace with having wasted part of your budget.

29.3 Random Walk

The random walk sampling family does exactly what you would expect it to do given its name: it performs a random walk on the graph, sampling the nodes it encounters. After all, if random walks are so powerful and we can use them for ranking nodes (Section 14.4) or projecting bipartite networks (Section 26.5), why can't we use them for sampling too? I'll start by explaining the simplest approach and its problems, moving into sophisticated variants that address its downsides.

Vanilla

In **Random Walk** (RW) sampling, we take an individual and we ask them to name one of their friends at random. Then we do the same with her and so on. Figure 29.6 shows the usual vignette applied to this strategy.

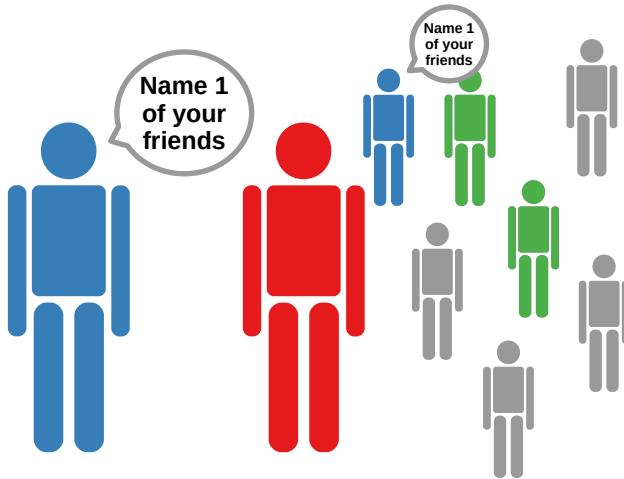


Figure 29.6: Random walk sampling. Your sampler (blue) starts from a seed (red) and asks for all the connections a node (green + gray). One of the neighbors is picked at random and becomes the new seed (green) and, when asked, will name another green node to become the new seed.

This is an easy approach which can be very effective, but it has problems. First, you might end up trapped in an area of the network where you already explored all nodes, thus unable to find new ones. This can be easily solved by allowing a random teleportation probability, just like PageRank does to avoid being stuck in a connected component of the network.

More importantly, RW sampling has a degree bias. Remember the stationary distribution (Section 11.1): the probability of ending in a node with a random walk is known and constant no matter where we started. And the stationary distribution has a 1-to-1 correspondence to the degree. This means that high degree nodes are very likely to be sampled, while low degree nodes not so much. Thus, with RW, your sample is not representative – at least when it comes to

representing nodes with all degrees fairly.

Note that not all biases are entirely bad, some are useful¹⁷. Specifically, we could compare this upward degree bias with the downward degree bias of node induced sampling. Arguably, if we have to be biased, at least let's oversample the important nodes in the network, rather than the unimportant ones. This philosophy is implemented by the Sample Edge Counts (SEC) method¹⁸. SEC ranks the neighbors of all the sampled nodes according to their degree and then explores the neighbor with the highest edge count towards already explored nodes.

In this vein, one could avoid the limit of performing a single random walk at a time. A simple extension of RW sampling is m-dependent Random Walk (MRW)¹⁹. This involves performing m random walks at once. The random walkers are not independent: we choose which of the m random walker will take the next step by looking at the degree of the nodes they are currently visiting. Thus, if there are three random walkers and they are currently on nodes with degrees 3, 2, and 1, we will continue from the first random walker with $3/(3+2+1) = 0.5$ probability.

Metropolis-Hastings

One way in which we could fix the issues of random walk sampling is by performing a “random” walk. Meaning that we still pick a neighbor at random to grow our sample, but we become picky about whether we really want to sample this new node or not.

In the **Metropolis-Hastings Random Walk** (MHRW), when we select a neighbor of the currently visited node, we do not accept it with probability 1. Instead, we look at its degree. If its degree is higher than the one of the node we are visiting, we have a chance of rejecting this neighbor and trying a different one. This probability is the old node's degree over the new node's degree. The exact formula for this decision is $p = k_v/k_u$, assuming that we visited v and we're considering u as a potential next step^{20,21}.

If the current node v has degree of 3, and its u neighbor has degree of 100, the probability of transitioning to u is only 3% – note that this is *after* we selected u as the next step of the random walk, thus the visit probability is actually lower than 3%: first you have a $1/k_v$ probability of being selected and *then* a k_v/k_u probability of being accepted. If we were, instead, to transition from u to v , we would always accept the move, because $100/3 > 1$, thus the test always succeeds. In practice, we might refuse to visit a neighbor if its degree is higher than the currently visited node. The higher this difference, the less likely we're going to visit it. A random walk with this rule

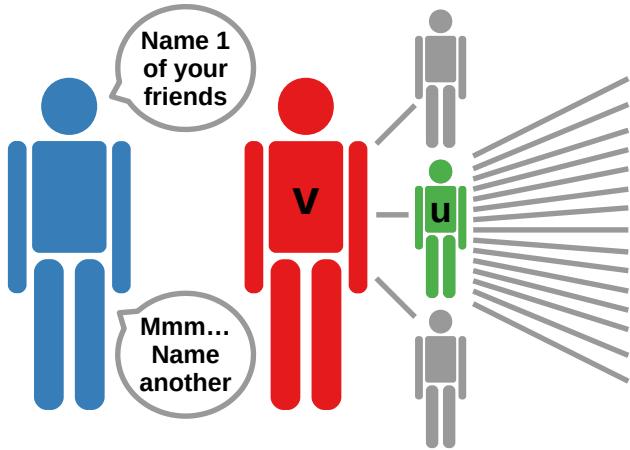
¹⁷ Sho Tsugawa and Hiroyuki Ohsaki. Benefits of bias in crawl-based network sampling for identifying key node set. *IEEE Access*, 8:75370–75380, 2020

¹⁸ Arun S Maiya and Tanya Y Berger-Wolf. Benefits of bias: Towards better characterization of network sampling. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 105–113. ACM, 2011

¹⁹ Bruno Ribeiro and Don Towsley. Estimating and sampling graphs with multidimensional random walks. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, pages 390–403, 2010

²⁰ Daniel Stutzbach, Reza Rejaie, Nick Duffield, Subhabrata Sen, and Walter Willinger. On unbiased sampling for unstructured peer-to-peer networks. *IEEE/ACM Transactions on Networking (TON)*, 17(2):377–390, 2009

²¹ Balachander Krishnamurthy, Phillipa Gill, and Martin Arlitt. A few chirps about twitter. In *Proceedings of the first workshop on Online social networks*, pages 19–24. ACM, 2008



will generate a uniform stationary distribution. Figure 29.7 shows the mental process of our Metropolis-Hastings sampler.

Re-Weighted

In Re-Weighted Random Walk (RWRW) we take a different approach. We don't modify the way the random walk is performed. We extract the sample using a vanilla random walk. What we modify is the way we look at it. Once we're done exploring the network, we correct the result for the property of interest^{22,23}. Say we are interested in the degree. We want to know the probability of a node to have degree equal to i . We correct the observation with the following formula:

$$p_i = \frac{\sum_{v \in V_i} i^{-1}}{\sum_{v' \in V} x_{v'}^{-1}}.$$

The formula tells us the probability of a node to have degree equal to i (p_i). This is the sum of i^{-1} – the inverse of the value – for all nodes in the sample with degree i (V_i), over 1/ degree ($x_{v'}^{-1}$) of all nodes in the sample (V). This is also known as Respondent-Driven Survey²⁴, because it is used in sociology to correct for biases in the sample when the properties of interest are rare and non-randomly distributed throughout the population. Figure 29.8 attempts to break down all parts of the formula.

Let's make an example. Suppose you want to estimate the probability of a node to have degree $i = 2$. First, you perform your vanilla random walk sample. Say you extracted 100 nodes. Twenty of those nodes have degree equal to two. So your numerator in the formula will be the sum of $i^{-1} = 1/2$ for $|V_i| = 20$ times: $20 * 1/2 = 10$. If we assume that there were 50 nodes of degree 1, 10 of degree 3, 8 of

Figure 29.7: Metropolis-Hastings Random Walk sampling. Your sampler (blue) starts from a seed (red) and asks for all the connections a node (green + gray). One of the neighbors is picked at random and we attempt to make it the new seed (green). However, since u has so many connections, it is likely that the sampler will ask for a different neighbor.

²² Matthew J Salganik and Douglas D Heckathorn. Sampling and estimation in hidden populations using respondent-driven sampling. *Sociological methodology*, 34(1):193–240, 2004

²³ Amir Hassan Rasti, Mojtaba Torkjazi, Reza Rejaie, Nick Duffield, Walter Willinger, and Daniel Stutzbach. Respondent-driven sampling for characterizing unstructured overlays. In *IEEE INFOCOM 2009*, pages 2701–2705. IEEE, 2009

²⁴ H Russell Bernard and Harvey Russell Bernard. *Social research methods: Qualitative and quantitative approaches*. Sage, 2013

$$p_i = \frac{\sum_{v \in V_i} i - 1}{\sum_{v' \in V} x_{v'} - 1}$$

Figure 29.8: The Re-Weighted Random Walk formula, estimating the probability p_i of observing the i value in a property of interest, using the set of sampled nodes V_i with that particular value in the total set of v sampled nodes.

degree 4, 7 of degree 5, and 5 of degree 6, our denominator would be:

$$(50/1) + (20/2) + (10/3) + (8/4) + (7/5) + (5/6),$$

which is $67.5\bar{6}$. Hopefully, you can spot what I did there. To bring the formula together, we discover that $p_2 = 10/67.5\bar{6} \sim 0.148$. So RWRW is telling us that the overall probability of a node having degree equal to 2 is not 20% as we would have inferred from the – biased – random walk sample. It is actually lower, it is 14.8%.

Note that the formula reported here only works when the variable of interest is discrete, i.e. i is an integer, like in the case of the degree. You can still apply RWRW sampling even if the variable you want to study is continuous, for instance the local clustering coefficient. However, you'll have to perform a kernel density estimate^{25,26,27}, and I'm not particularly keen of going into that nest of vipers. You're on your own, have a blast.

RWRW works particularly well when there are hidden populations who might actively try not to be sampled²⁸. For instance, it has been successfully applied to the sampling of drug users²⁹.

RWRW has a crucial downside. While it is excellent to estimate the distribution of a property in a network, it will still return a biased vanilla random walk sample. So, if what you need was the sample rather than the estimation of a simple measure, you're out of luck. You cannot use this method to have a representative sample.

Neighbor Reservoir Sampling

Neighbor Reservoir Sampling³⁰ (NRS) is one of those methods blending between the two families of sampling I talked about. It happens in two phases. In the first phase, NRS builds its set of core

²⁵ Murray Rosenblatt. Remarks on some nonparametric estimates of a density function. *The Annals of Mathematical Statistics*, pages 832–837, 1956

²⁶ Emanuel Parzen. On estimation of a probability density function and mode. *The annals of mathematical statistics*, 33(3):1065–1076, 1962

²⁷ https://en.wikipedia.org/wiki/Kernel_density_estimation

²⁸ Douglas D Heckathorn and Christopher J Cameron. Network sampling: From snowball and multiplicity to respondent-driven sampling. *Annual review of sociology*, 43:101–119, 2017

²⁹ David C Bell, Elizabeth B Erbaugh, Tabitha Serrano, Cheryl A Dayton-Shotts, and Isaac D Montoya. A comparison of network sampling designs for a hidden population of drug users: Random walk vs. respondent-driven sampling. *Social science research*, 62:350–361, 2017

³⁰ Xuesong Lu and Stéphane Bressan. Sampling connected induced subgraphs uniformly at random. In *International Conference on Scientific and Statistical Database Management*, pages 195–212. Springer, 2012

nodes and connections. Starting from the seed we provide as an input, NRS performs a normal random walk, including in the sample all nodes and edges it finds during this exploration.

However, the majority of NRS's budget is spent in the second phase. Once we have a core, we start modifying it. Suppose that, after the first phase, you sampled nodes in a set V' . In this second phase, you make a loop. At each iteration i of the loop, you pick two nodes at random: u and v . Node v is a member of V' , the set of explored nodes. Node u is not a member of V' – meaning that you haven't explored it yet, but it is a neighbor of a node in V' .

Our objective is to add u to V' and to remove v from V' at the same time. We can do it only if two conditions are met. First condition: we want our sample to be a single connected component. We cannot remove v if that would break the graph into multiple components – adding u isn't going to add new components, because we only consider us that are connected to a node in V' ensuring connectivity. Note that u and v usually are not connected to each other.

The second condition is a random test. We extract a uniform random number $0 < \alpha < 1$. We perform the switch if and only if $\alpha < |V'|/i$, where i is set to be equal to $|V'|$ at the beginning and it is increased by one at each attempt. In practice, this has a few consequences. By swapping u and v , we ensure that the size of our sample stays constant, i.e. $|V'|$ doesn't change. Moreover, at the beginning, since $i \sim |V'|$ all initial attempts to modify V' succeed. As we progress, the chances of accepting a new node in the set vanish.

This isn't really a random walk nor a BFS, because the random neighbor selected can be from any node in V' . So you can see how hard it is to fit it into a neat category.

NRS ensures a realistic clustering coefficient distribution. How can that happen? The trick lies in the connectivity test. We only perform the $u-v$ swap if it doesn't break the network into distinct connected components. Which means that not all vs have the same probability of being removed from the sample. The v with higher clustering have higher probability to be replaced, because by removing them it is more likely that the graph will stay connected. High clustering coefficient means that their neighbors are connected to each other (see Section 12.2), thus making it more likely there are alternative paths to keep the network together.

To see why it's the case, consider Figure 29.9. NRS will pick a node in green and a node in red at random. It will then remove the red node and add the green node. However, it will always refuse to perform the operation if the red node you pick is node 5. Removing that node will create two connected components, which is unaccept-

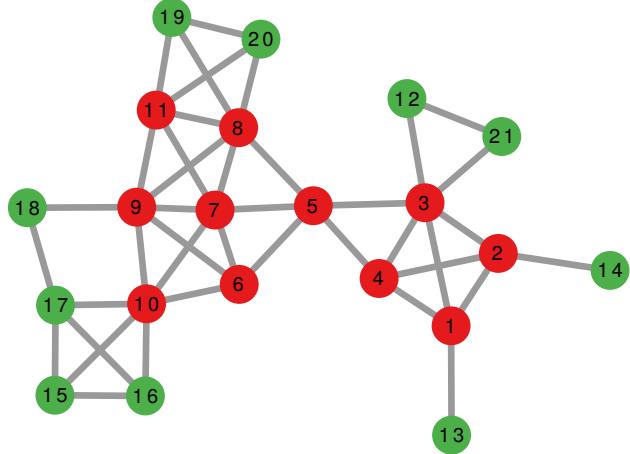


Figure 29.9: Neighbor Reservoir sampling. Nodes in the explored set V' are in red. Neighbors of V' – the reservoir – are in green.

able. Other unlucky $u-v$ draws are forbidden too. For instance, you cannot perform the swap if you pick nodes 3 and 12.

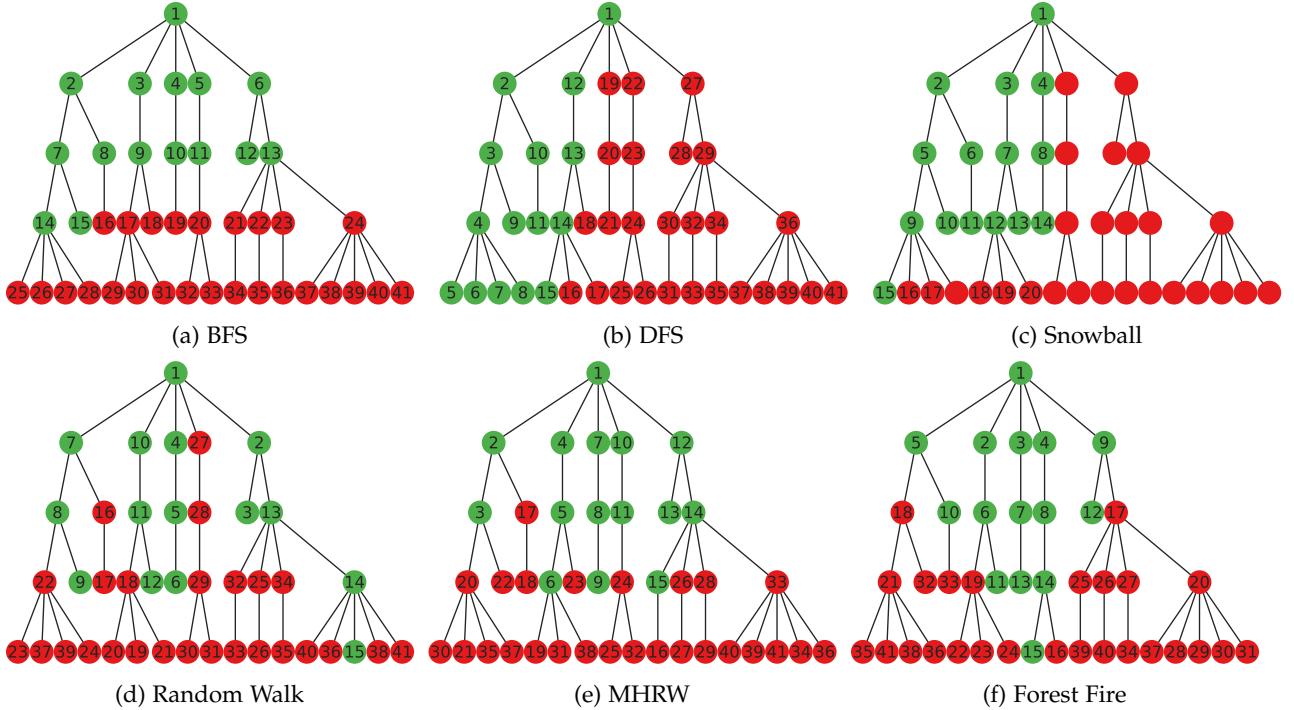
Figure 29.10 shows an example of how some of these different strategies would explore a simple tree. I don't show RWRW, because the samples it extracts are indistinguishable from the vanilla random walk ones. I also don't include NRS, because it's too subtle to really be appreciated in a figure like this one.

29.4 Sampling Issues

When talking about Snowball sampling I mentioned the issue of pagination. To recap: social media APIs will rarely give you all connections of a user when you ask for them. Rather, they will send you a list of k , chosen with some criterion that is opaque to you (likely in the order they are stored in their internal database). If you want the remaining ones, you have to ask again. You're always getting k connections at a time. Each request is a "page", with k being the page size.

It can be tricky to know how pagination will affect crawl time. Imagine two different API policies. The first returns big pages – say 100 edges per page – but requires a long waiting time between queries – say two seconds. The second policy returns small pages – ten edges per page – but more often – you only need to wait one second between queries. We can calculate the edge throughput of these two policies. In this case, the one with big pages returns more edges per unit of time, on average: 50 edges per second versus 10 edges per second. However, how will these policies behave on a real world network?

As we saw in Section 9.3, real world networks have broad degree distributions, like the one we show in Figure 29.11. For some of these



nodes, the second policy is better: if they have 10 or less edges, we can fully explore them with a single query, thus we're going faster because we have lower waiting times between nodes. In Figure 29.11, we color in blue the part of the degree distribution for which this holds true. If the node has a degree higher than 10, then the first policy is better, because it requires to perform fewer queries, even if they are spaced out more in time. In Figure 29.11, we color in purple the part of the degree distribution for which this holds true.

The second policy is in theory 5 times slower than the first (10 edges/sec versus 50 edges/sec, on paper), however it will allow you to crawl this network in half of the time³¹. This is because, in a broad degree distribution, we have way more nodes with low degree – for which the second policy is faster. In Figure 29.11, out of 500k nodes, 492k have degree of 10 or less.

You could conclude that the best API policy possible is the one that gives you only one node at a time, imposing no waiting time between requests. This is true only in theory. In practice there are a few things you need to consider, which you can lump into the issue of network latency. First, it still takes time for the information to travel from the server to your computer. This is not exactly the speed of light, so the requests will never be truly instantaneous. Second, a server which gets hit too frequently with too many requests will also naturally slow down, often in unpredictable ways. Thus some

Figure 29.10: Examples of how different network sampling strategies explore a given network. Each node is labeled with the order in which it is explored. The node color shows whether the node was sampled (green) or not (red), assuming a budget of 15 units and a constant cost of 1 unit per node. Snowball assumes $n = 3$ (unlabeled nodes are not explored due to this parametric restriction), while Forest Fire has a burn probability of .5.

³¹ Michele Coscia and Luca Rossi. Benchmarking api costs of network sampling strategies. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 663–672. IEEE, 2018

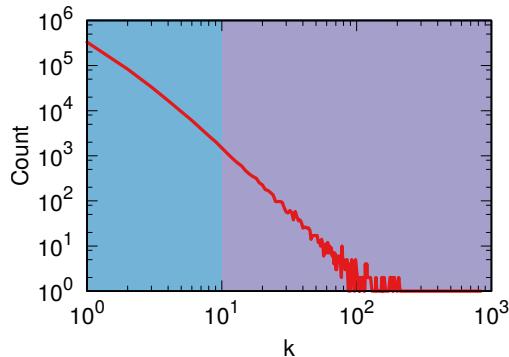


Figure 29.11: A power law degree distribution, showing the count of nodes (y-axis) with a given degree (x-axis). The colors in the plot represent in which cases the first API policy described in the text is faster than the second (purple) and when the second is faster than the first (blue).

level of pagination and waiting time will always be part of an API system. Which means that there are going to be trade-offs when reconstructing the underlying network.

Pagination is often not the only thing you need to worry about. Other challenges might be sampling a network in presence of hostile behavior³². For instance, some hostile nodes will try to lie about their connections and it's your duty to reconstruct the true underlying structure. Or not: there are reasonable and legit reasons to lie about one's connection, for instance to protect one own privacy.

In another scenario, you might not be interested in the topological properties of the full network. What's interesting for you is just estimating the local properties of one – or more – nodes. In that case, specialized node-centric strategies can be used³³.

29.5 Network Completion

Network completion is a related – but not identical – problem. Like sampling, it wants to establish a topological strategy for the exploration of a network. Different from sampling, its aim is not to extract a smaller version of the full dataset. Rather, we want to complete the sample. The idea is that you downloaded from somewhere a sample of a network, but you are able to process a larger dataset. Rather than starting collecting data from scratch, you can use what you have as a seed and try to complete it.

The question now is: what's the most efficient way to do so? What strategy would give you the most information about the full structure with the least amount of effort? Specifically, you want to obtain the highest possible number of new nodes by asking the lowest amount possible of new queries. You could simply apply any of the network sampling strategies I explained so far. However, there are dedicated techniques developed to solve this problem.

Note that here you don't know the strategy originally used to collect the sample you're given. If you knew that it was a Metropolis-

³² Edward Bortnikov, Maxim Gurevich, Idit Keidar, Gabriel Kliot, and Alexander Shraer. Brahms: Byzantine resilient random membership sampling. *Computer Networks*, 53(13):2340–2359, 2009

³³ Manos Papagelis, Gautam Das, and Nick Koudas. Sampling online social networks. *IEEE Transactions on knowledge and data engineering*, 25(3):662–676, 2013

Hastings random walk you'd probably use a different strategy than if it was a standard BFS. But, since you don't know this piece of information, you need a general strategy working regardless of the shape of the initial sample.

Naively, you might think to just go and probe the nodes with the highest degree. However, there are a few considerations to make. First, since – by definition – your sample is incomplete, you don't really know the true degree of a node. You only know how many neighbors it has in your sample. Second, since the node has a high degree in your sample, there's some chance you already explored all its neighbors, thus probing it won't help you.

The first technique, MaxReach³⁴, estimates the true degree of a node and its clustering coefficient using the information gathered so far. It does so with a technique similar to Re-Weighted Random Walk. The difference is that, in RWRW, we only want to know how many nodes have a given degree i . In MaxReach, we want to also know which nodes have that given degree value. At this point, the score of a node is the difference between its estimated degree and its degree in the sample. Nodes with higher scores are probed earlier. After each probe, since we gathered more information in the sample, MaxReach will recalculate the degree estimates.

ϵ -wgx³⁵ is a more recent alternative.

29.6 Summary

1. Network sampling is a necessary operation when the network you need to analyze is too large and/or you need to gather data one node/edge at a time from a high latency source (e.g. the API of a social media platform). Sometimes the decision is not up to you and all you can access is a sample made by somebody else.
2. The main objective is to extract a sample that is representative of the network at large for the property you're interested in studying. For instance, it has to have a comparable degree distribution if you want to infer its shape (e.g. whether it is a power law).
3. Sampling methods can be induced or topological. In induced sampling you extract a random sample of nodes/edges and you collect all that it is attached to it. In topological sampling you explore the structure one node at a time.
4. Variants of BFS explorations are: Snowball, in which we impose a maximum number k of explored neighbors of a node; and Forest Fire, in which we have a probability of rejecting some edges.

³⁴ Sucheta Soundarajan, Tina Eliassi-Rad, Brian Gallagher, and Ali Pinar. Maxreach: Reducing network incompleteness through node probes. In *2016 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, pages 152–157. IEEE, 2016

³⁵ Sucheta Soundarajan, Tina Eliassi-Rad, Brian Gallagher, and Ali Pinar. ϵ -wgx: Adaptive edge probing for enhancing incomplete networks. In *Proceedings of the 2017 ACM on Web Science Conference*, pages 161–170. ACM, 2017

5. Variants of random walk exploration are: Metropolis-Hastings, where we have a probability of refusing visiting high degree nodes; and Re-Weighted, where we correct the statistical properties of the sample after we collected it.
6. When sampling from real API systems one has to be careful that the throughput in edges per second is not necessarily a good indicator of how quickly you can gather a representative sample. Due to pagination, high-throughput sources might return smaller samples.
7. A related problem is network completion: given an incomplete sample of a network, find the best strategy to complete the sample in the least number of queries possible.

29.7 Exercises

1. Perform a random walk sampling of the network at <http://www.networkatlas.eu/exercises/29/1/data.txt>. Sample 2,000 nodes (1% of the network) and all their connections (note: the sample will end up having more than 2,000 nodes).
2. Compare the CCDF of your sample with the one of the original network by fitting a log-log regression and comparing the exponents. You can take multiple samples from different seeds to ensure the robustness of your result.
3. Modify the degree distribution of your sample using the Re-Weighted Random Walk technique. Is the estimation of the exponent of the CCDF more precise?
4. Modify your random walk sampler so that it applied the Metropolis-Hastings correction. Is the estimation of the exponent of the CCDF more precise? Is MHRW more or less precise than RWRW?

Part IX

Mesoscale

30

Homophily

“Mesoscale” is the term we use to indicate network analyses that operate at the level that lies in between the global and the local one. At the global level, we have analyses that sum up the topological characteristics of a network with a single number. For instance, the exponent of the degree distribution, the global clustering coefficient, or the diameter. At the local level, we sum up individual node characteristics with a single number. They can be its degree, its local clustering coefficient, or closeness centrality.

At the mesoscale we want to describe groups of nodes. How do they relate to each other? How does their local neighborhood look like? There are many different meso-level analyses you can perform. This part of the book groups almost all of them together, leaving one out: community discovery. Community discovery is, by far, the most popular meso-level analysis of complex networks. Given its size, it deserves a part on its own, which will be the next one. Here, we’re talking about all the meso-rest.

We start with homophily: the love (*philia*) of the similar (*homo*). “Birds of a feather flock together” is a popular way of saying. It originates from the fact that many species of birds flock with individuals of their own kind and coordinate when moving around. This phrase has been adopted in sociology to exemplify the concept of homophily: people will tend to associate with people with similar characteristics as their own. In a social network, homophily implies that nodes establish edges among them if they have similar attributes¹. If you have a particular taste in movies, and there are two potential friends, you are more likely to choose the one with similar tastes as yours, because you have more things to talk about and have less potential for conflict.

Many factors influence and favor homophily, and they are not necessarily exclusively explained by individual preference: sometimes homophily is a property emerging from access. It’s not only the fact that you don’t *like* the different, but rather that you cannot

¹ Miller McPherson, Lynn Smith-Lovin, and James M Cook. Birds of a feather: Homophily in social networks. *Annual review of sociology*, 27(1):415–444, 2001

access the different. In other words, homophily is not only the result of our preferences, but also of social constructs. That is why the term “homophily” is problematic, and we use it only because of historic reasons.

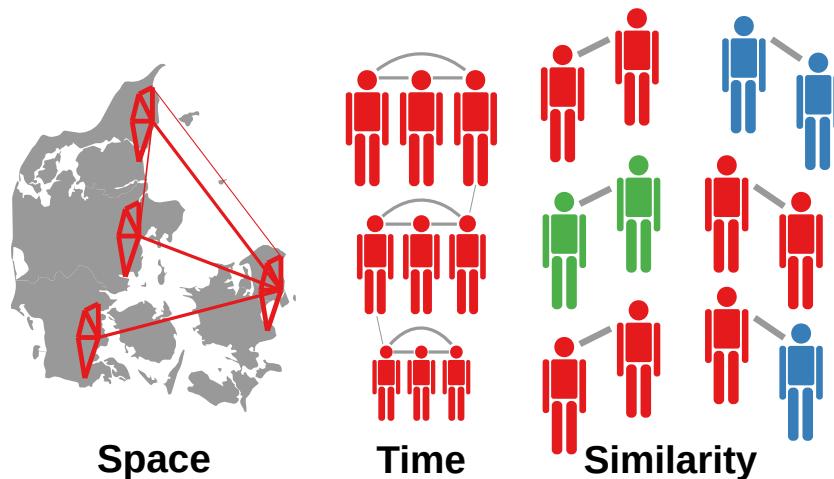


Figure 30.1: Some examples of homophily driven by spatial, temporal, and attribute similarity.

With that said, let me go through a carousel of examples of homophily, some of which I represent in Figure 30.1. There are so many observed examples in real world social networks that I have to push their references down to the next page otherwise my Latex won’t compile. So have a picture of my cat Ferris. He is, incidentally, a great example of homophily, in that he hates everything that is not himself.



First, gender² and race³ are glaring examples. School children are more likely to make friends with people sharing their gender or race⁴. We observe this in adults too: in marriage ties it is so overwhelmingly likely to date⁵ or marry⁶ someone of the same race that sociologists

² Juliette Stehlé, François Charbonnier, Tristan Picard, Ciro Cattuto, and Alain Barrat. Gender homophily from spatial behavior in a primary school: a sociometric study. *Social Networks*, 35(4): 604–613, 2013

Figure 30.2: My cat Ferris. In the picture, I color in orange the parts of the cat that are orange.

³ Marta C González, Hans J Herrmann, J Kertész, and Tamás Vicsek. Community structure and ethnic preferences in school friendship networks. *Physica A*, 379(1):307–316, 2007

⁴ Kara Joyner and Grace Kao. School racial composition and adolescent racial homophily. *Social science quarterly*, pages 810–825, 2000

⁵ Elizabeth Aura McClintock. When does race matter? race, sex, and dating at an elite university. *Journal of Marriage and Family*, 72(1):45–72, 2010

⁶ Kelly Raley, Megan Sweeney, and Danielle Wondra. The growing racial and ethnic divide in us marriage patterns. *Future of children*, 25(2):89, 2015

don't study this fact any more because it's so boringly obvious. In this, we're truly similar to other animals we often look down to⁷. Rather than asking whether romantic ties show homophily, it's more interesting to use the degree of homophily of romantic ties to compare societies.

In Figure 30.3 you see an example of mixed marriage in the United States. To that diagonally dominated matrix, you have to add the consideration that the United States is probably one of the most diverse countries in the world. Imagine how this would look like elsewhere!

| | | Wife | | | | |
|--|--|-------|-------|-------|-------|-------|
| | | White | Black | Asian | Other | |
| | | White | 0.977 | 0.003 | 0.01 | 0.009 |
| | | Black | 0.086 | 0.892 | 0.009 | 0.013 |
| | | Asian | 0.07 | 0.003 | 0.918 | 0.009 |
| | | Other | 0.44 | 0.016 | 0.034 | 0.51 |

Another example is spatial homophily: living in the same place makes it easier to have stronger connections, a factor that overcomes other correlates such as race^{8,9,10}. A sub-type of spatial homophily is mobility homophily: going to the same places influences the likelihood of connecting socially^{11,12}. The reverse is also true – as it might seem obvious –: being friends increases the likelihood to go to the same places¹³. The connection between geographical space and social space is very strong, showing how, even in presence of (almost) limitless communication ranges, social ties still decay with distance^{14,15,16}.

Another factor of homophily is time, meaning that being in the same age range favors connections. Think about school friends: 38% of a person's friends are within a 2-year age gap – this figure comes from McPherson's paper.

In the rest of the chapter we are going to explore some techniques to study the mesoscale, such as the usage of ego networks. We're going to quantify homophily and see some of the consequences in network dynamics.

⁷ Yuxin Jiang, Daniel I Bolnick, and Mark Kirkpatrick. Assortative mating in animals. *The American Naturalist*, 181(6):E125–E138, 2013

⁸ Salvatore Scellato, Anastasios Noulas, Renaud Lambiotte, and Cecilia Mascolo. Socio-spatial properties of online location-based social networks. In ICWSM, 2011

⁹ Kerstin Sailer and Ian McCulloh. Social networks and spatial configuration—how office layouts drive social interaction. *Social networks*, 34(1):47–58, 2012

Figure 30.3: The mixing matrix of interracial marriage in the US: share of husbands per race with a wife of a given race (Census Bureau).

¹⁰ Ling Heng Wong, Philippa Pattison, and Garry Robins. A spatial model for social networks. *Physica A*, 360(1):99–120, 2006

¹¹ Dashun Wang, Dino Pedreschi, Chaoming Song, Fosca Giannotti, and Albert-László Barabási. Human mobility, social ties, and link prediction. In SIGKDD, pages 1100–1108. ACM, 2011a

¹² Jameson L Toole, Carlos Herrera-Yaqüe, Christian M Schneider, and Marta C González. Coupling human mobility and social ties. *Royal Society Interface*, 12(105):20141128, 2015

¹³ Junjoon Cho, Seth A Myers, and Jure Leskovec. Friendship and mobility: user movement in location-based social networks. In SIGKDD, pages 1082–1090. ACM, 2011

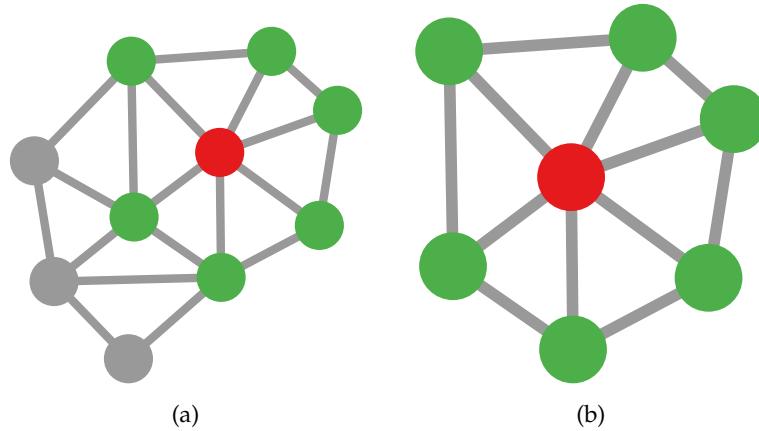
¹⁴ Jukka-Pekka Onnela, Samuel Arbesman, Marta C González, Albert-László Barabási, and Nicholas A Christakis. Geographic constraints on social network groups. *PLoS one*, 6(4):e16939, 2011

¹⁵ Michele Coscia and Ricardo Hausmann. Evidence that calls-based and mobility networks are isomorphic. *PLoS one*, 10(12):e0145091, 2015

¹⁶ Pierre Deville, Chaoming Song, Nathan Eagle, Vincent D Blondel, Albert-László Barabási, and Dashun Wang. Scaling identity connects human mobility and social interactions. *PNAS*, 113(26):7047–7052, 2016

30.1 Ego Networks

Ego networks are a common technique to explore the meso-level around a node¹⁷. “Ego” in Latin means “I”, the self. An ego network is a subset of a larger network. You first have to identify your “ego”: the node on which the ego network is centered. Then, you select all of its neighbors and the connections among them. The resulting network formed by all the nodes and edges you selected is an ego network. Figure 30.4 provides an example of this procedure.



¹⁷ Nick Crossley, Elisa Bellotti, Gemma Edwards, Martin G Everett, Johan Koskinen, and Mark Tranmer. *Social network analysis for ego-nets: Social network analysis for actor-centred networks*. Sage, 2015

Figure 30.4: The procedure to extract an ego network from a larger network. (a) We select the ego (in red) and all its neighbors (in green). (b) We create a view only using red and green nodes, and all connections among them.

Once you have an ego network, you can start investigating its “global” properties such as the degree distribution or its homophily, and these are not properties of the global network as a whole, but of the local neighborhood of the ego, the ego network, which lives in the mesoscale. Ego networks are frequently used in social network analysis^{18,19}, for instance to estimate a person’s social capital²⁰.

A consequence of this procedure is that we know that an ego node is connected to all nodes in its ego network. This is unfortunate in some cases, depending on our analytic needs. For instance, all ego networks have a single connected component and will have a diameter of two. If those forced properties are undesirable, one can extract an ego network and then remove the ego and all its connections.

30.2 Quantifying Homophily

When it comes to homophily, we want to have an objective way to quantify how much it is driving a network’s connections. This means that the nodes connect to other nodes depending on the value of one of their attributes. There are two possible scenarios. The attribute driving the connections could be quantitative (e.g. age) or qualitative (e.g. gender). When the attribute is quantitative, you can use the

¹⁸ Stephen P Borgatti, Ajay Mehra, Daniel J Brass, and Giuseppe Labianca. Network analysis in the social sciences. *science*, 323(5916):892–895, 2009

¹⁹ Jure Leskovec and Julian J Mcauley. Learning to discover social circles in ego networks. In *Advances in neural information processing systems*, pages 539–547, 2012

²⁰ Stephen P Borgatti, Candace Jones, and Martin G Everett. Network measures of social capital. *Connections*, 21(2):27–36, 1998

same technique to estimate the degree assortativity, which we cover in the next chapter.

Here we focus on the case of a qualitative attribute. Let's start by making a simple scenario: biological sex. In humans, this is – barring rare and exceptional cases – a categorical binary attribute.

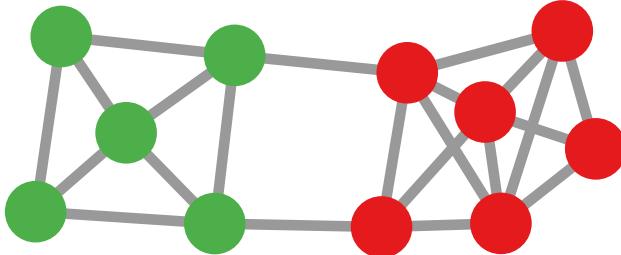


Figure 30.5: A toy example to test our measures of homophily. We represent the categorical binary attribute with node color, with two possible values: red and green.

In this scenario, you can estimate the probability of an edge to connect alike nodes, and compare it to the probability of connection in the network. Consider Figure 30.5. We have 20 edges connecting nodes with the same color over 22 total edges in the graph. Therefore, the observed probability of edges between alike nodes is 20/22. In the graph we have 11 nodes, thus the number of possible edges is $|V|(|V| - 1)/2 = 55$ (with $|V| = 11$). So the probability of having a connection between any node pair is 22/55. Thus we see that the probability of an edge being between alike nodes is more than twice what we would have expected: $(20/22)/(22/55) \sim 2.27$. Values higher than one imply homophily, while values lower than one mean that nodes tend to connect with similar nodes less than we expect – i.e. the network is disassortative, nodes don't like to connect to similar nodes, another totally valid thing that can happen often in social networks (in Section 30.4 I call this concept "heterophily").

This approach breaks down if you have more than two possible values for your attribute, and also if some values are more popular than others. In these cases, you might conclude that there is assortativity in a non-assortative network, simply because you're assuming the incorrect null model of equal attribute value popularity.

In this case, you should use a different approach^{21,22}. You want to look at the probability of edges connecting alike nodes per attribute value i , and then compare it to the probability of an edge to have at least one node with attribute value i . The formula is:

$$r = \frac{\sum_i e_{ii} - \sum_i a_i b_i}{1 - \sum_i a_i b_i},$$

where e_{ii} is the probability of an edge to connect two nodes which both have value i , a_i is the probability that an edge has as origin a

²¹ Mark EJ Newman. Assortative mixing in networks. *Physical review letters*, 89(20):208701, 2002

²² Mark EJ Newman. Mixing patterns in networks. *Physical Review E*, 67(2):026126, 2003a

node with value i , and b_i is the probability that an edge has as destination a node with value i . In an undirected network, the latter two are equal: $a_i = b_i$. This formula takes values between -1 (perfect disassortativity) and 1 (perfect assortativity: each attribute is a separate component of the network).

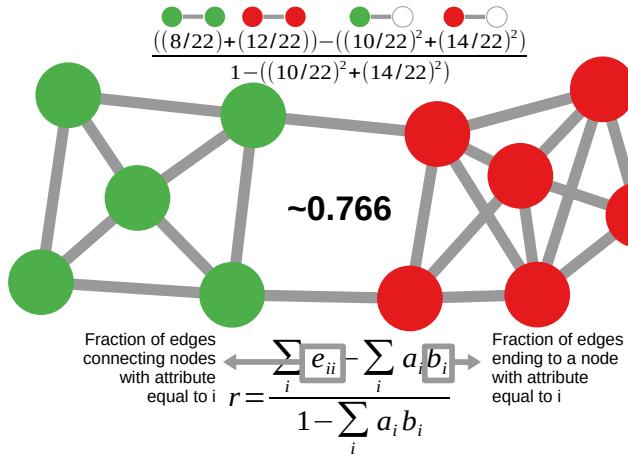


Figure 30.6: How to calculate homophily using the formula in the text.

In Figure 30.6 we have two values i : red and green. There are 22 edges in the graph: eight green-green edges – thus the probability is $8/22$ – and 12 red-red edges – thus the corresponding e_{ii} value is $12/22$. Ten edges originate (or end) in a green node: $a_i = b_i = 10/22$; and 14 originate (or end) in a red node: $a_i = b_i = 14/22$. The final value of homophily is ~ 0.766 . This value is interpretable as a sort of Pearson correlation coefficient, which means that 0.766 is pretty high.

30.3 Strength of Weak Ties

Is homophily a good thing? In some aspects yes. A person who is surrounded by people with similar tastes and behaviors is happy. But suppose this person is looking for a job. It is very difficult, in presence of high homophily, for a message to arrive to the job seeker, because she only has close ties who cannot broker to her new information from the outside – assuming that the network has a strong assortative community structure.

The ties that bind different communities with different people are the so-called “weak ties” and they have been shown to be fundamental in the job market by Granovetter^{23,24}. To put it simply: it’s rarely your closest friends who make you find a job, but that far acquaintance with whom you rarely speak, because your close friends usually access the same information as you do and so cannot tell you anything new. Figure 30.7 shows an example of the weak ties effect.

Note that Granovetter divides ties in three types: weak, strong,

²³ Mark S Granovetter. The strength of weak ties. In *Social networks*, pages 347–367. Elsevier, 1977

²⁴ Mark Granovetter. The strength of weak ties: A network theory revisited. 1983

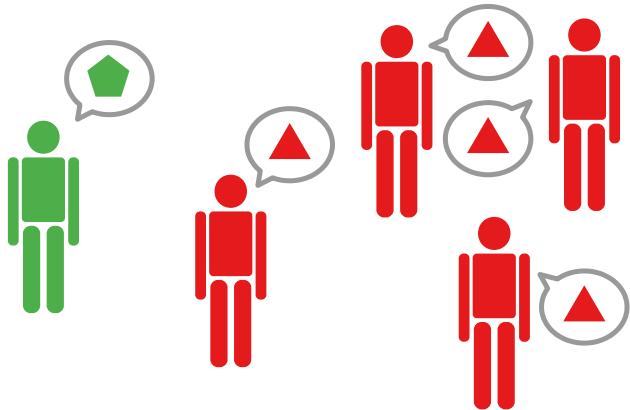


Figure 30.7: An example of strength of weak ties. The green individual is part of a different community and thus only weakly linked with the red community. However, by being exposed to different information, she can bring it to the community she is not part of, but connected to it via a weak tie.

and absent. The terminology should not fool you. In this case, we are not referring to the edge's weight (Section 6.3). This is rather a categorical difference, more akin to multilayer networks (Section 7.2). A weak tie is established between individuals whose social circles do not overlap much. A strong tie is the opposite: an edge between nodes well embedded in the same community. The absent tie is more of a construct in sociology, which lacks a well-defined counterpart in network science. It can be considered as a potential connection lurking in the background. For instance, there is an absent tie between you and that neighbor you always say "hello" to but never interact beyond that. You could consider an absent tie as one of the most likely edges to appear next, if you were to perform a classical link prediction (Part VII).

You can see now that you can have strong, weak, and absent ties in an unweighted network. We can, of course, expect a correlation between being a weak tie and having a low weight. However, we can construct equally valid scenarios in which there is an anti-correlation instead. For instance, we could weight the edges by their edge betweenness centrality (Section 14.2). A weak tie must have a high edge betweenness, because by definition it spans across communities and thus all the shortest paths going from one community to the other must pass through it.

Note that, notwithstanding their usefulness in favoring information spread, weak ties are not the only game in town in a society. The competing concept of the "strength of strong ties" shows that strong ties are important as well. They are specifically useful in times of uncertainty: "Strong ties constitute a base of trust that can reduce resistance and provide comfort²⁵".

²⁵ David Krackhardt, N Nohria, and B Eccles. The strength of strong ties. *Networks in the knowledge economy*, 82, 2003

30.4 Social Contagion

Homophily can lead to a surprising number of counter-intuitive social dynamics. This section is intimately linked with Part VI, where we looked at spreading events in networks. Here, we explore some more social explanations behind behavioral change in networks, mostly fueled by the right combination of homophily and heterophily (the love of the different).

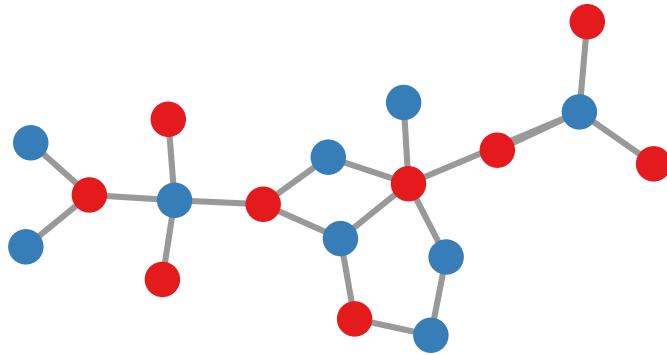


Figure 30.8: A dating network. The node color encodes the gender (red = female, blue = male).

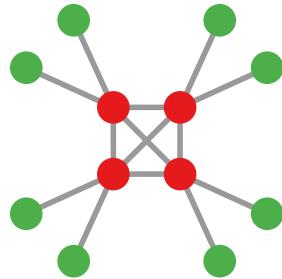
What's heterophily? Some things in social networks are very disassortative. For instance, consider sexual partners. When looking at some attributes, it is a network driven by homophily: people try to find mates with similar characteristics. They like the same music, movies, food. On the other hand, other attributes are very disassortative, for instance gender. Notwithstanding notable exceptions, the majority of edges in this network are between unlike genders – as Figure 30.8 shows.

If we live in a network governed by homophily, we know that connections are driven by the characteristics of the nodes. In some cases that is the only possible explanation. For instance, race is given: one cannot change their race and race homophily means that one's race influences which social connections are more or less likely to be established.

But consider the other side of the coin: if we observe a strong homophily, it could be because our social connections are influencing us into adopting behaviors we would not otherwise. For instance, drug use. One can decide whether to use drugs, and will be more likely to do so if the majority of their friends are drug users. It turns out that the right network topology can create an illusion of majority. Even if the majority of people do not use drugs, we can draw a network in which everybody thinks that the opposite is true²⁶!

You can look at Figure 30.9 to see an example of this counter intuitive result. Or, you can play a simple game showing how to build networks fooling people into thinking everybody is binge-

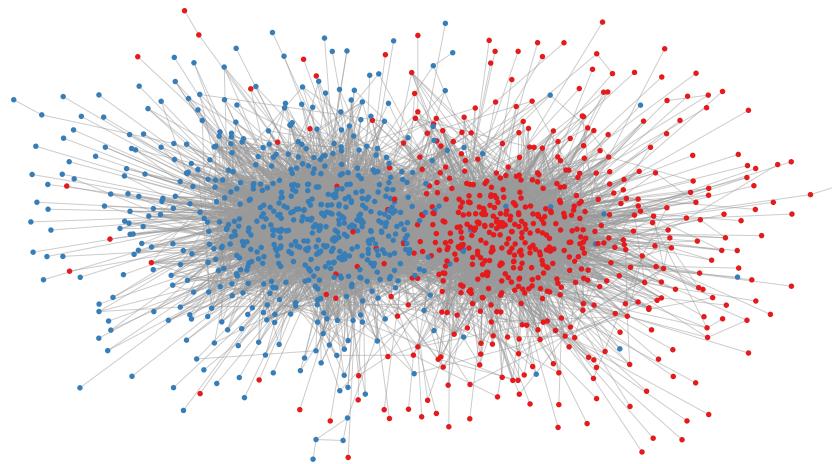
²⁶ Kristina Lerman, Xiaoran Yan, and Xin-Zeng Wu. The "majority illusion" in social networks. *PLoS One*, 11(2):e0147617, 2016



drinking²⁷.

Since humans are social animals and tend to succumb to peer pressure, homophily can be a channel for behavioral changes. In a health study, researchers looked at health indicators from thousands of people in a community over 32 years. They saw that behavior and health risks that should not be contagious actually are. For instance obesity: if you have an obese friend, the likelihood of you becoming obese increases by 57% in the short term²⁸. This is like the Susceptible-Infected epidemic models we saw, even if obesity is not a biological virus. It is rather a social type of virus.

Same with smoking, although in this case it worked the opposite: people were quitting in droves²⁹. This is due to social pressure and homophily: a behavior you might not adopt by yourself is brokered by your social circle, which you trust because it is made by people like you – it speaks to your identity.



Another paper shows strong homophily in political blogs³⁰. In Figure 30.10 we see a visualization of how people writing online about politics connect to each other. A common political vision is the clear driving force behind the creation of an hyperlink from one blog to another.

Figure 30.9: The majority illusion in a toy network. Nodes in red are drug users, nodes in green are not. For every node, its neighbors include a majority of drug users.

²⁷ <http://ncase.me/crowds-prototype/>

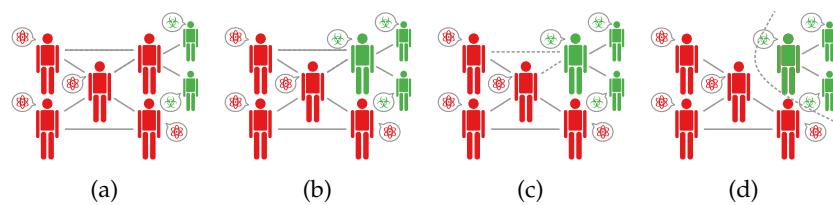
²⁸ Nicholas A Christakis and James H Fowler. The spread of obesity in a large social network over 32 years. *New England journal of medicine*, 357(4): 370–379, 2007

²⁹ Nicholas A Christakis and James H Fowler. The collective dynamics of smoking in a large social network. *New England journal of medicine*, 358(21): 2249–2258, 2008

Figure 30.10: The network of political blogs. Each node is a blog. Node's color encodes its political leaning (blue = democrat, red = republican). Two nodes are connected if either blog links to the other.

³⁰ Lada A Adamic and Natalie Glance. The political blogosphere and the 2004 us election: divided they blog. In *Proceedings of the 3rd international workshop on Link discovery*, pages 36–43. ACM, 2005

Homophily arises very strongly even with mild preferences. One classical example is segregation. The famous “Parable of Polygons³¹” starts from a simple assumption: people want to live with at least some similar people next to them. Even if they do not seek a majority of alike neighbors the end result is very clustered. Try to make an experiment and set the threshold to 40%, which means that people are happy being in the *minority*. You’ll still end up with segregation. There’s no network in this interactive example, but one could easily introduce one by allowing nodes to rewire their friendship preferences according to the same rules. Experiments building relation graphs via RFID tags show that these dynamics may shape the topology of networks of face-to-face interactions³².



We are venturing now in new territory. So far we have seen homophily as a constructive force, meaning that people with similar characteristics link to each other. But with segregation we’re doing something different. We’re seeing homophily as a *destructive* force: polygons are moving away if their expectation of uniformity isn’t met. In network terms, people who are connected and discover differences in their characteristics might decide to rescind their connection.

Recently, researchers have started investigating this effect: rather than preferring to connect to similar strangers, we preferably rescind connections from dissimilar friends. Suppose you’re on Facebook and you share a lot of content on scientific topics. One of the members of your community has outside connections, which could be convincing them of something like anti-vaccination. This person starts sharing anti-vax content, and the rest of the community is likely to rescind its connections. Which ends up creating groups that cannot connect any more people with different ideas, and thus reinforce each other convictions without any debate³³. Figure 30.11 shows a vignette of this process.

This is particularly problematic, as there is a large body of research showing how easy it is for misinformation to spread through social media³⁴ and how strong online echo chambers can be^{35,36}. In fact, a sufficiently determined single actor can magnify their impact online, as the challenge in creating and operating difficult-to-detect bot nets is easy to overcome^{37,38}. There is suggestive research show-

³¹ <https://ncase.me/polygons/>

³² Ciro Cattuto, Wouter Van den Broeck, Alain Barrat, Vittoria Colizza, Jean-François Pinton, and Alessandro Vespignani. Dynamics of person-to-person interactions from distributed rfid sensor networks. *PLoS one*, 5(7), 2010

³³ Michela Del Vicario, Antonio Scala, Guido Caldarelli, H Eugene Stanley, and Walter Quattrociocchi. Modeling confirmation bias and polarization. *Scientific reports*, 7:40391, 2017

Figure 30.11: Homophily driving echo chambers. Science-oriented people (in red) rescind connections from conspiracy theorists (green) creating communities which have no possibility of communicating.

³⁴ Michela Del Vicario, Alessandro Bessi, Fabiana Zollo, Fabio Petroni, Antonio Scala, Guido Caldarelli, H Eugene Stanley, and Walter Quattrociocchi. The spreading of misinformation online. *PNAS*, 113(3):554–559, 2016a

³⁵ Michela Del Vicario, Gianna Vivaldo, Alessandro Bessi, Fabiana Zollo, Antonio Scala, Guido Caldarelli, and Walter Quattrociocchi. Echo chambers: Emotional contagion and group polarization on facebook. *Scientific reports*, 6:37825, 2016b

³⁶ Eytan Bakshy, Solomon Messing, and Lada A Adamic. Exposure to ideologically diverse news and opinion on facebook. *Science*, 348(6239):1130–1132, 2015

³⁷ Chengcheng Shao, Giovanni Luca Ciampaglia, Onur Varol, Alessandro Flammini, and Filippo Menczer. The spread of fake news by social bots. *arXiv*, pages 96–104, 2017

³⁸ Emilio Ferrara, Onur Varol, Clayton Davis, Filippo Menczer, and Alessandro Flammini. The rise of social bots. *Communications of the ACM*, 59(7):96–104, 2016

ing how this might already have happened³⁹.

30.5 Summary

1. Homophily or assortativity is the tendency of nodes in a network to connect with nodes that are similar to them in some attribute. For instance, people tend to be friends in a social network with other people of similar age or same race.
2. A way to study this meso scale property is by creating ego networks: you pick one node as ego and then you create a network view including only its neighbors and the connections among them.
3. There are measures to estimate attribute assortativity, usually interpreted as a correlation coefficient taking values from +1 (perfect assortativity) to -1 (perfect disassortativity).
4. Disassortativity is the opposite of assortativity: nodes tend to connect to other nodes with different attributes from their own. For instance, the dating network tends to be disassortative by gender.
5. Homophily interacts with network process. Links lowering homophily connect nodes with different attributes, which can favor information spread (the “strength of weak ties”).
6. In other cases, nodes can be fooled into seeing a minority attribute as always the majority option in their friends: the majority illusion.

30.6 Exercises

1. Load the network at <http://www.networkatlas.eu/exercises/30/1/data.txt> and its corresponding node attributes at <http://www.networkatlas.eu/exercises/30/1/nodes.txt>. Iterate over all ego networks for all nodes in the network, removing the ego node. For each ego network, calculate the share of right-leaning nodes. Then, calculate the average of such shares per node.
2. What is the assortativity of the leaning attribute?
3. What is the relative popularity of attribute values “right-leaning” and “left-leaning”? Based on what you discovered in the first exercise, would you say that there is a majority illusion in the network?

³⁹ Alessandro Bessi and Emilio Ferrara. Social bots distort the 2016 us presidential election online discussion. 2016

31

Quantitative Assortativity

In the previous chapter we talked about homophily, the love of the similar. It is our tendency of liking the people who are similar to us: similar race, similar places we hang around, similar movies we watch. These are all *qualitative* attributes. In this chapter, we make the jump towards *quantitative* homophily.

Many node attributes are quantitative: age, number of friends, etc. We can still estimate the level of homophily in a network based on these attributes. In this case, we perform a small change in terminology. We use the term “assortativity” instead. This change is largely arbitrary, but can help you in differentiating between the concepts. Just like “(qualitative) homophily = (quantitative) assortativity”, we have “(qualitative) heterophily = (quantitative) disassortativity”.

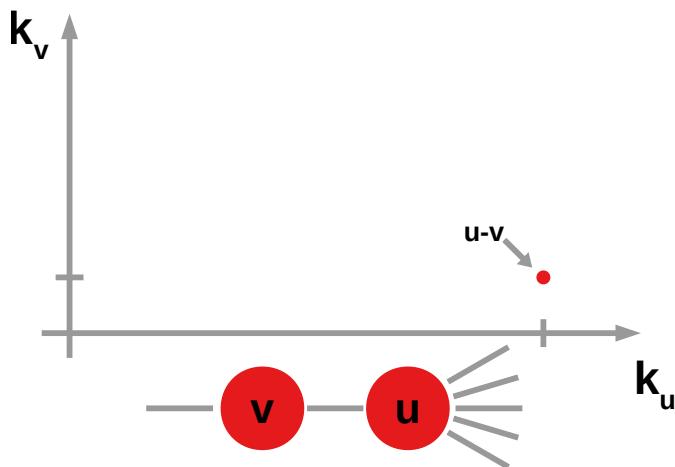
Shifting our attention to quantitative attributes means we can use slightly different tools to estimate homophily, since there is a clear sorting in the attribute values and an intuition of similarity. If two nodes of values 1 and 2 connect, it is true that they have a different attribute value, but it still counts more towards assortativity than, say, connecting a node of value 1 to a node of value 100.

The set of possible quantitative attributes can be vast. For this chapter, I’m going to focus mainly on one example, which is the most studied case: the degree. However, don’t be fooled: any technique for the estimation of degree assortativity can be employed to estimate any other quantitative attribute’s assortativity. However, by focusing on the degree, I can introduce other fun assortativity-related network effects, such as the friendship paradox.

31.1 Degree Correlations

The degree is the most studied example of assortativity because it is directly related to the edge creating process. In a degree-assortative network we see that hubs connect preferentially to hubs, while peripheral nodes connect preferentially to other peripheral nodes.

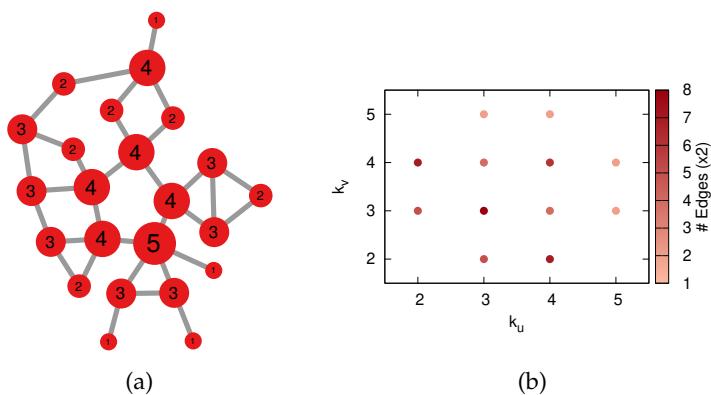
This is like a dating network, where celebrities hook up with each other much more than you would expect if the dating network would be fair¹.



In a disassortative network, hubs connects to periphery, as it happens for instance in protein networks². This is more similar to the classical preferential attachment, where newcomer low-degree nodes will connect more often to older and high-degree hubs.

A way to visualize degree assortativity is to consider each edge as an observation. We create a scattergram, recording the degree of one node on the x-axis and of the other node on the y-axis. So each point in this scatter plot is an edge of the network. Remember that the degree in real world networks follows a skewed distribution spanning many orders of magnitude. So, usually, these plots will have a log-log scale. Figure 31.1 shows the skeleton of such a scatter plot. For an example network, such scatter would look like the one in Figure 31.2.

Such a visualization suggests us a way to compute a possible



¹ Which is *clearly* the only reason why I've been unsuccessful in getting a date with Jennifer Lawrence. No other possible explanation.

Figure 31.1: A scatter plot we can use to visualize degree assortativity. For each edge, we have the degree of one node on the x axis and of the other node on the y axis.

² Peter Uetz, Loic Giot, Gerard Cagney, Traci A Mansfield, Richard S Judson, James R Knight, Daniel Lockshon, Vaibhav Narayan, Maithreyan Srinivasan, Pascale Pochart, et al. A comprehensive analysis of protein–protein interactions in *saccharomyces cerevisiae*. *Nature*, 403(6770):623, 2000

Figure 31.2: (a) A network with nodes labeled with their degree. (b) A scatter plot we can use to visualize (a)'s degree assortativity. Each point is a possible degree combination of an edge. The data point color tells you how many edges have that particular degree combination. Usually, this count should also be log-transformed.

index of degree assortativity. This is the first of two options you have if you want to quantify the network's assortativity. You iterate over all the edges in the network and put into two vectors the degrees of the nodes at the two endpoints. Note that each edge contributes two entries to this vector – unless your network is directed. So, if your network only contains a single edge connecting nodes 1 and 2, your two vectors are $x = [k_1, k_2]$ and $y = [k_2, k_1]$, with k_v being the degree of node v . Then, assortativity is simply the Pearson correlation coefficient of these two vectors.

There is only one way to achieve perfect degree assortativity. In such a scenario, each node is connected only to nodes with the exact same degree. This is true only in a clique. Thus, a perfectly degree assortative network is one in which each connected component is a clique.

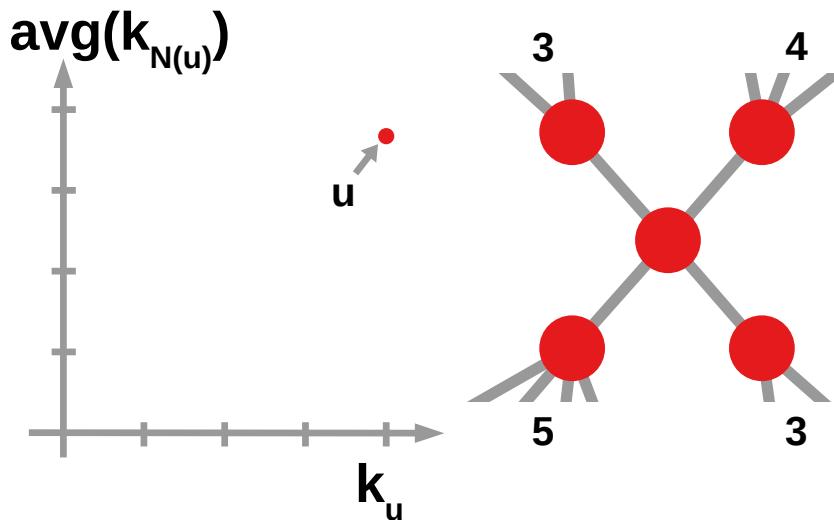


Figure 31.3: A second strategy to visualize degree assortativity. The scatter plot has a point for each node in the network, reporting its degree (x axis) against the average degree of its neighbors (y axis).

Figure 31.3 shows the second strategy to estimate degree assortativity in a network. Rather than plotting each edge, we plot each node. We compare a node's degree with the average degree of its neighbors. In a degree assortative network, we expect to see a positive correlation: the more connections the node has, the more connections, on average, its neighbors have³.

Again, you shouldn't forget to log-transform the degree values, given the broad degree distributions of most real world networks. This also means that you should perform a power fit. The exponent of such a fit tells you whether the network is degree assortative (if it's positive), disassortative (if it's negative), or non assortative (if it's statistically indistinguishable from zero).

Figure 31.4 shows few examples of assortativity in real world net-

³ Romualdo Pastor-Satorras, Alexei Vázquez, and Alessandro Vespignani. Dynamical and correlation properties of the internet. *Physical review letters*, 87(25):258701, 2001

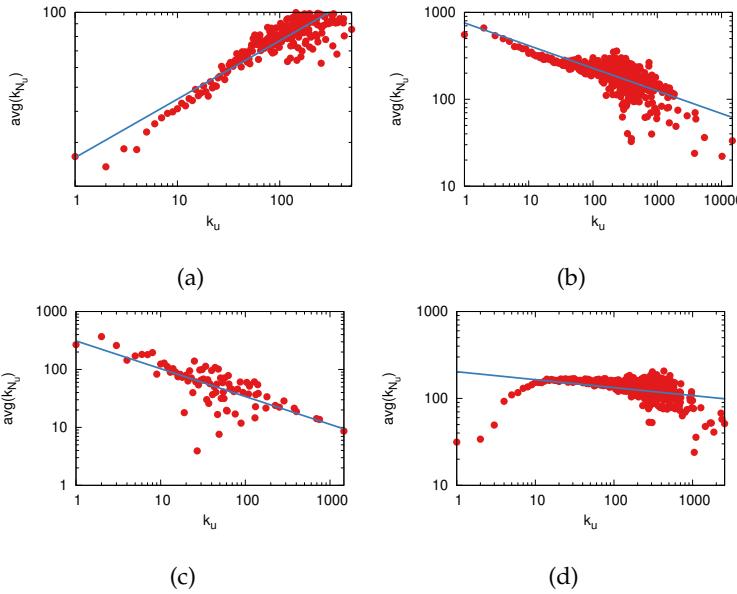


Figure 31.4: A collection of assortativity plots for four real world networks: (a) co-authorship in scientific publishing, (b) P2P network, (c) Internet routers, (d) Slashdot social network.

works. Coauthorship is assortative: if I have a lot of coauthors, on average, they have a lot of coauthors too. Gowalla is disassortative: users with few friends likely attach to hubs. Slashdot is still disassortative, but it is the closest we could find to show what a neutral network looks like: one where my degree doesn't tell me anything about my neighbors' degree.

Note that, for real world networks, we usually aggregate in the same data point all nodes with the same degree. Thus what we're plotting is actually the average degree of all neighbors of all nodes of degree k . Otherwise, we would have an unreadable cloud of points at low degree values, since most nodes in real world networks have a degree equal to one or two.

Degree assortativity is a super important property for your network. Degree correlations radically change many network dynamics^{4,5,6,7}. This is especially true when you have multilayer networks and we're looking at assortativity inter-layer besides intra-layer⁸. Meaning: if I am a hub in one layer, am I also a hub in the other layers? We touched the topic in Section 22.4. If the answer to this question is yes, failure-resistant layers become failure-prone^{9,10}.

There is a curious tension between degree assortativity and other common statistical properties of real world networks. For instance, we just saw that scientific collaboration is an assortative network. However, we also know it has a broad degree distribution.

The two properties clash against each other: assortativity means that hubs connect to hubs, but in a network with a heavy-tailed degree distribution there are few huge hubs and many one-degree

⁴ Marián Boguñá, Romualdo Pastor-Satorras, and Alessandro Vespignani. Absence of epidemic threshold in scale-free networks with degree correlations. *Phys. Rev. Lett.*, 90:028701, Jan 2003. DOI: [10.1103/PhysRevLett.90.028701](https://doi.org/10.1103/PhysRevLett.90.028701)

⁵ Alexei Vázquez and Yamir Moreno. Resilience to damage of graphs with degree correlations. *Phys. Rev. E*, 67:015101, Jan 2003. DOI: [10.1103/PhysRevE.67.015101](https://doi.org/10.1103/PhysRevE.67.015101)

⁶ F Sorrentino, M Di Bernardo, G Huerta Cuellar, and S Boccaletti. Synchronization in weighted scale-free networks with degree-degree correlation. *Physica D: nonlinear phenomena*, 224(1-2):123–129, 2006

⁷ Márton Pósfai, Yang-Yu Liu, Jean-Jacques Slotine, and Albert-László Barabási. Effect of correlations on network controllability. *Scientific reports*, 3:1067, 2013

⁸ Zhen Wang, Lin Wang, Attila Szolnoki, and Matjaž Perc. Evolutionary games on multilayer networks: a colloquium. *The European physical journal B*, 88(5):124, 2015b

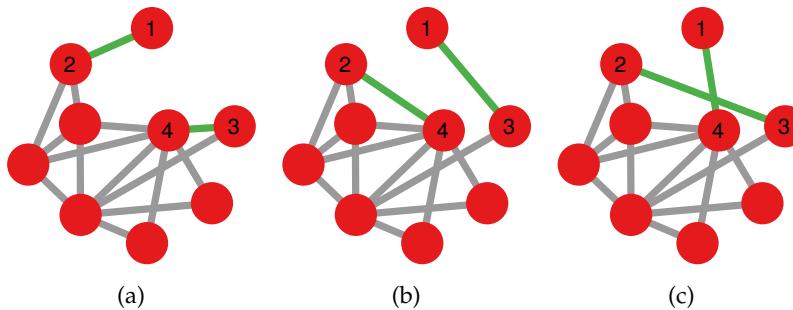
⁹ Jianxi Gao, Sergey V Buldyrev, Shlomo Havlin, and H Eugene Stanley. Robustness of a network of networks. *Phys. Rev. Lett.*, 107:195701, Nov 2011. DOI: [10.1103/PhysRevLett.107.195701](https://doi.org/10.1103/PhysRevLett.107.195701)

nodes. The likelihood of connecting a hub to many small nodes seems too high. In fact, if we were to generate a random version of the co-authorship network respecting its degree distribution – for instance via a configuration model (Section 18.1) –, we would obtain a degree disassortative network¹¹. This makes quite interesting networks that both have a skewed degree distribution and are degree assortative! They have some non-trivial machinery driving their nodes' connections that cannot be captured by simple models.

The network generators I discussed in Part V weren't really designed with assortativity in mind. As we just saw, the configuration model is naturally disassortative, as is preferential attachment and anything which imposes a power law degree distribution. By definition, random graphs such as $G_{n,p}$ are non-assortative.

However, if you start with any synthetic network, there are algorithms to rewire the edges such that the degree distribution will be preserved, but you will obtain an assortative (or disassortative) network^{12,13,14}.

This happens through edge swap, as Figure 31.5 shows. First, you select two connected node pairs. Then you sort them according to their degree, in the figure the order is nodes 4, 2, 3, 1 ($k_4 = 6$, $k_2 = 3$, $k_3 = 2$, $k_1 = 1$). The next move depends on whether you want to induce assortativity or disassortativity. In the first case, you connect the two nodes with the highest degree to each other, and the two with lowest degree to each other (Figure 31.5(b)). In the second case, you do the opposite: connect the highest degree node with the lowest, and the two middle ones (Figure 31.5(c)).



Note that this swap doesn't always change the topology nor alter the characteristics of the network. For instance, if all nodes have the same degree, the move would not affect assortativity. But, after enough trials in a large enough network, you'll see that these operations will have the desired effect.

Degree assortativity, as I discussed it so far, is defined for undirected networks. There are straightforward extensions for directed networks¹⁵. The standard strategy is to look at four correlation coeffi-

¹⁰ Jia Shao, Sergey V Buldyrev, Shlomo Havlin, and H Eugene Stanley. Cascade of failures in coupled network systems with multiple support-dependence relations. *Phys. Rev. E*, 83:036116, Mar 2011. DOI: 10.1103/PhysRevE.83.036116

¹¹ Marián Boguñá, Romualdo Pastor-Satorras, and Alessandro Vespignani. Cut-offs and finite size effects in scale-free networks. *The European Physical Journal B*, 38(2):205–209, 2004

¹² Marián Boguñá and Romualdo Pastor-Satorras. Class of correlated random networks with hidden variables. *Phys. Rev. E*, 68:036112, Sep 2003. DOI: 10.1103/PhysRevE.68.036112

¹³ Ramon Xulvi-Brunet and Igor M Sokolov. Reshuffling scale-free networks: From random to assortative. *Physical Review E*, 70(6):066102, 2004

¹⁴ R. Xulvi-Brunet and I. M. Sokolov. Changing Correlations in Networks: Assortativity and Dissortativity. *Acta Physica Polonica B*, 36:1431, 5 2005

Figure 31.5: The (dis)assortativity inducing model. (a) Select two pairs of connected nodes (in green the edges we select). (b) Assortativity inducing move. (c) Disassortativity inducing move.

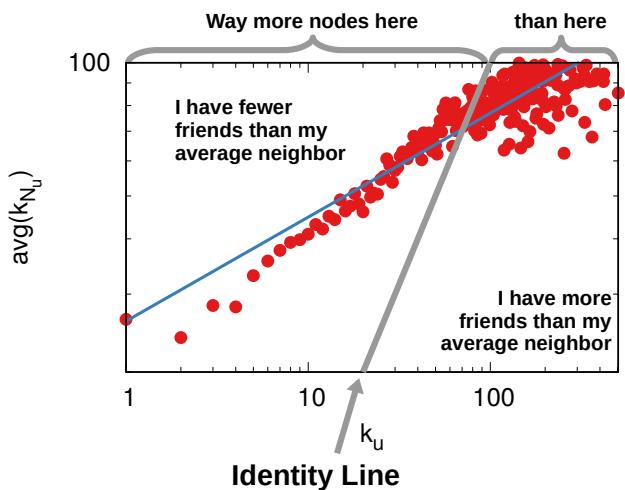
¹⁵ Jacob G Foster, David V Foster, Peter Grassberger, and Maya Paczuski. Edge direction and the structure of networks. *Proceedings of the National Academy of Sciences*, 107(24):10815–10820, 2010

clients: in-degree with in-degree, in-degree with out-degree (and vice versa), and out-degree with out-degree.

Obviously, everything I wrote so far on degree assortativity also applies to any other quantitative attribute you might have on your nodes. For instance, if you have a social network, it applies to a person's age, height, weight, and so on. You can build the scattergrams and calculate the best fit to figure out if your social circle sort themselves according to their height or income.

31.2 Friendship Paradox

You might want to make a double take on Figure 31.4, because it contains a not-so-obvious but intriguing – and enraging – message. In Figure 31.6 I focus on one of those assortativity plots, the one about scientific collaborations. I add an additional line to the plot: the identity line. This line runs through the part of the plane where the x axis and the y axis have the same value.



In other words, the identity line divides the space in two. Above the identity line we have all the nodes for which, on average, the neighbor degree is higher than the node's degree. Below the identity line it's the opposite: the node's degree is higher than the neighbors degree.

At first glance, the situation seems balanced. There are as many points above the identity line as there are below. However, remember that we're aggregating all nodes with the same degree value in a single point. We know that the degree has a broad distribution, because we visualized it for the co-authorship network before. Therefore, there are way more nodes above the identity line than below.

That's the friendship paradox: your friends are, on average, more

Figure 31.6: The friendship paradox. The plot shows a node's degree (x axis) against the average degree of its neighbors (y axis). The blue line is the best fit, while the gray line is the identity line. Nodes above the identity line have fewer friends than their friends' average.

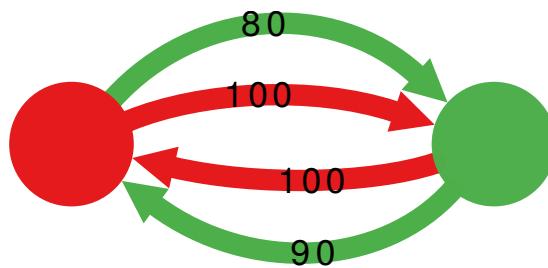
popular than you^{16,17}! This means that, for the average node, its degree is lower than the average degree of their neighbors. This is actually pretty obvious once you think about it: a node with degree k appears in k other nodes' averages, and hence is "over-counted" by an amount equal to how much larger it is than the network's mean degree. A high degree node appears in many more node neighborhoods than does a low degree node, and hence it skews many local averages. The only way to escape such paradox is by having a network whose degree distributes mostly regularly: for instance small-world networks (Section 17.2) are usually immune to the friendship paradox because most nodes have the same degree (the probability of rewiring is low).

The friendship paradox sounds pretty depressing, but we actually already made use of it in a rather uplifting scenario. The effective "vaccinate-a-friend" scheme I discussed in Section 21.3 is nothing else than a practical application of this network property.

31.3 Distribution of Quantitative Attributes

Quantitative assortativity does not stop at the degree. There are examples of papers analyzing quantitative attributes discovering all sort of interesting phenomena. I'm going to give you one example I know well, as it came from a paper I wrote¹⁸.

In the paper, I analyze a business to business network, connecting businesses if they are customers or suppliers of each other. Each edge is a B2B transaction and both businesses have to report it, as Figure 31.7 shows. Thus, if they report a different amount, we know someone is lying. I create a measure of trustworthiness, which is the average value of mismatch a business has, weighted by how trustworthy its neighbors are.



This trustworthiness score is a quantitative attribute. It is strongly correlated with the likelihood that the business was in fact cheating on their taxes, as I have information whether the audited businesses were fined and, if they were, how much they had to pay.

Simulations show that, with this correction, in a randomly wired

¹⁶ Scott L Feld. Why your friends have more friends than you do. *American Journal of Sociology*, 96(6):1464–1477, 1991

¹⁷ Ezra W Zuckerman and John T Jost. What makes you think you're so popular? self-evaluation maintenance and the subjective side of the "friendship paradox". *Social Psychology Quarterly*, pages 207–223, 2001

¹⁸ Mauro Barone and Michele Coscia. Birds of a feather scam together: Trustworthiness homophily in a business network. *Social Networks*, 54:228 – 237, 2018. ISSN 0378-8733

Figure 31.7: The data model of the business to business network. The edge color tells corresponds to the node making the claim about the transaction. For instance, the green node reports selling 90 to the red node and buying 80 back.

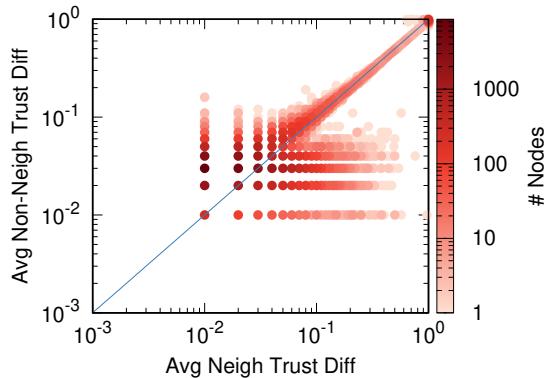


Figure 31.8: The average trust-worthiness score difference between neighbors (x axis) and non-neighbors (y axis). The blue line shows the identity, and the point color the number of nodes with the given value combination.

network the score should be disassortative. Instead, in the real observed network, the score is assortative. Figure 31.8 shows the relation: there are more nodes above the identity line than below – it might appear that the opposite is true, but you need to take into account the color of the dots. Being above the identity line means that the trust score difference between neighbors is lower than between non-neighbors: a sign of assortativity.

Given the connection with the actual tax fines, the assortativity analysis can make us conclude something tangible about business connections. In this case, that fraudulent and untrustworthy businesses band together. If I know your customer/suppliers are scammers, I should update my priors on whether you're a scammer as well.

A more famous example of quantitative attribute assortativity is related to the friendship paradox that I just described in the previous section, and it is ten times more enraging. We humans are social animals. Notwithstanding introverted cavemen like myself, in general our level of happiness is correlated with the number of friends we have. In fact, just like the degree, happiness is assortative in social networks: happy people tend to befriend each other¹⁹.

What I just stated is that more friends imply more happiness. And the friendship paradox tells us that our friends have more friends than us. Do I mean to tell that, like with friendship, there is also a happiness paradox? Why, yes there is²⁰. If you ask people about their level of happiness in a social network, you will find out that the average happiness level of one's friends tends to be higher than their own happiness level. That is probably why you think everyone is having such a great time on social media. Everyone but you. It's not you, it's the system. Luckily, the researchers behind this discovery have a few guidelines on how to unplug from social media toxicity and live a more fulfilling life²¹.

In general, everything that correlates with degree – be it happiness,

¹⁹ Johan Bollen, Bruno Gonçalves, Guangchen Ruan, and Huina Mao. Happiness is assortative in online social networks. *Artificial life*, 17(3):237–251, 2011

²⁰ Johan Bollen, Bruno Gonçalves, Ingrid van de Leemput, and Guangchen Ruan. The happiness paradox: your friends are happier than you. *EPJ Data Science*, 6(1):4, 2017

²¹ Johan Bollen and Bruno Gonçalves. Network happiness: How online social interactions relate to our well being. In *Complex Spreading Phenomena in Social Systems*, pages 257–268. Springer, 2018

income, or tax fraud – will get its own paradox for free.

31.4 Summary

1. Assortativity works not only on qualitative, but also on quantitative attributes. The most studied case is degree assortativity: the tendency of high degree nodes to connect to other high degree nodes (degree assortative) or to low degree nodes (degree disassortative).
2. You can calculate degree assortativity – or any quantitative assortativity – by correlating the attribute values at the endpoints of each edge. Alternatively, you can correlate a node's attribute value with the average of their neighbors.
3. Graph generators usually are unable to provide degree assortative networks, but there are postprocessing techniques that can induce either degree assortativity or degree disassortativity.
4. By a cruel mathematical property of degree distributions, social systems are affected by the friendship paradox: the average person has fewer friends than their friends on average.
5. Even crueler, since in social system happiness is usually correlated with the number of friends a person has, the friendship paradox also implies than the average person is less happy than their friends on average. So it's not just you.

31.5 Exercises

1. Draw the degree assortativity plots of the network at <http://www.networkatlas.eu/exercises/31/1/data.txt> using the first (edge-centric) and the second (node-centric) strategies explained in Section 31.1. For best results, use logarithmic axes and color the points proportionally to the logarithmic count of the observations with the same values.
2. Calculate the degree assortativity of the network from the previous question using the first (edge-centric Pearson correlation) and the second (node-centric power fit) strategies explained in Section 31.1.
3. Prove whether the network from the previous questions is affected or not by the friendship paradox.

32

Core-Periphery

When you obtain a new network dataset and you plot it for the first time, in the vast majority of cases you will see a blobbed mess. This is usually due to the fact that raw network data is usually a hairball, and you need to backbone it, or perform other data cleaning tasks, as I detailed in Part VIII. However, in some cases, there is an unobjectionable truth. It might be that, deep down, your network really is a hairball.

Many large scale networks have a common topology: a very densely connected set of core nodes, and a bunch of casual nodes attaching only to few neighbors. This should not be surprising. If you create a network with a configuration model and you have a broad degree distribution, the high degree nodes have a high probability of connecting to each other – see Section 18.1. The surprising part is that the cores of some empirical networks are even denser than what you'd anticipate by looking at the degree distribution of the network¹!

Since everything that departs from null expectation is interesting, this phenomenon in real world networks has attracted the attention of network scientists. They gave a couple of names to this special meso-scale organization of networks: core-periphery^{2,3}, with the core sometimes dubbed as “rich club”⁴.

We already saw the concept of core and periphery when we discussed node roles and k-core centrality in Section 14.7. Such method is not included here because it finds a fundamentally different type of core-periphery structure, which is more similar to a hierarchical decomposition of the network estimating the centrality of the nodes⁵. In this chapter I discuss ways in which you can detect a core-periphery structure that is less hierarchical and more “hub-and-spoke”, in which we want to determine which nodes are in the core and which others are in the periphery. I also discuss the tension between the ubiquity of core-periphery structures and the equally common and (only apparently) contradictory presence of

¹ Shi Zhou and Raúl J Mondragón. The rich-club phenomenon in the internet topology. *IEEE Communications Letters*, 8(3):180–182, 2004

² Petter Holme. Core-periphery organization of complex networks. *Phys. Rev. E*, 72:046111, Oct 2005. DOI: 10.1103/PhysRevE.72.046111

³ Peter Csermely, András London, Ling-Yun Wu, and Brian Uzzi. Structure and dynamics of core/periphery networks. *Journal of Complex Networks*, 1(2):93–123, 2013b

⁴ Vittoria Colizza, Alessandro Flammini, M Angeles Serrano, and Alessandro Vespignani. Detecting rich-club ordering in complex networks. *Nature physics*, 2(2):110–115, 2006b

⁵ Ryan J Gallagher, Jean-Gabriel Young, and Brooke Foucault Welles. A clarified typology of core-periphery structure in networks. *arXiv preprint arXiv:2005.10191*, 2020

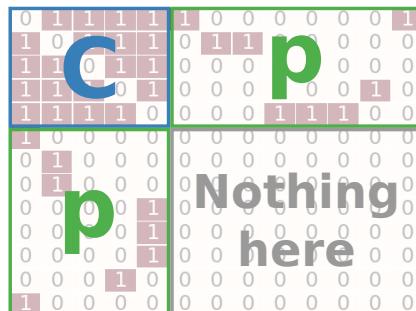
communities in complex networks. Finally, I'll connect core-periphery structures with a few real world dynamics that might be able to generate them.

32.1 Models

There are many ways to extract core-periphery structures from your networks. However, two methods dominate in the literature, especially in sociology. These are the discrete and the continuous model⁶.

Discrete

Core-periphery networks emerge when all nodes belong to a single group. Some nodes are well connected while others, while still being part of that group, are not. In a pure idealized core-periphery network the nodes can be classified strictly into two classes. The core nodes are the one with a high degree of interconnectedness. The periphery nodes are the rest, the ones that are only sparsely connected in the network.



There can be only two types of connections: between core nodes – which is the most common edge type, since the core is densely connected – and between a core-periphery pair. Peripheral nodes do not connect to each other. In the adjacency matrix, which I show in Figure 32.1, there's a big area with no connections. This is known as the "Discrete Model". It is a very strict one, and rarely real world networks comply with this standard. A perfect discrete model in which the core is composed by a single node is a star.

If you want to detect the core-periphery structure using the discrete model, you have a simple quality measure you want to maximize. This is $\sum_{uv} A_{ij}\Delta_{uv}$, with A being the adjacency matrix, and Δ a matrix with a value per node pair. An entry in Δ is equal to one if either of the two nodes is part of the core.

⁶ Stephen P Borgatti and Martin G Everett. Models of core/periphery structures. *Social Networks*, 21(4): 375 – 395, 2000. ISSN 0378-8733. DOI: [https://doi.org/10.1016/S0378-8733\(99\)00019-2](https://doi.org/10.1016/S0378-8733(99)00019-2)

Figure 32.1: A toy example of the Discrete Model for core-periphery structures. This adjacency matrix shows, highlighted in blue, a dense area of the network with many connections. In green, a sparser area: the periphery. Connections only go to (or from) a core member, meaning that in the main diagonal in the peripheral area there are no entries larger than zero.

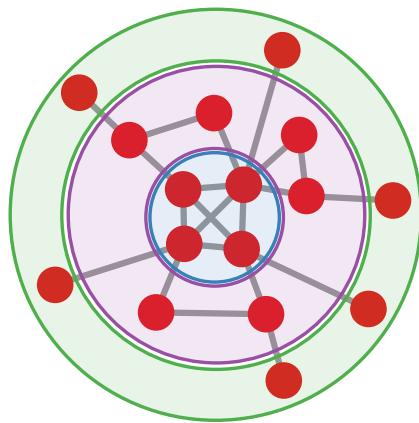
Since A is immutable, your quest is to find the best Δ such that the sum is maximized, i.e. you are capturing all edges established between a core node and all other nodes in the network. In a network following the perfect discrete model, this sum is equal to $|E|$ the number of edges in a network.

Of course, you cannot compare the “coreness” of two different networks unless they have the same number of nodes and edges, because this measure will take different expected values. That is why, sometimes, you can simply calculate the Pearson correlation coefficient between A and Δ .

Without going into details of specialized algorithms, one can find the best Δ using classical randomization algorithms. Options are genetic algorithms, simulated annealing, or basin hopping.

Continuous

Reality rarely conforms with strict expectations. Having only two classes in which to put nodes is exceedingly restrictive. What if nodes can be sorted in three classes? What if a semi-periphery exists? This is an enticing opportunity, until you realize that you could also ask: why three classes? Why not four? Why not five? Why not... you get the idea.



This is where the Continuous Model comes into play. Looser than the Discrete Model, in the Continuous Model we have an arbitrary number of classes for the nodes beyond the core. The intermediate classes can interconnect with each other and with the periphery proper, just in a looser way than the core proper. I show an example in Figure 32.2.

To be more precise, rather than creating an arbitrary number of classes, we assign to each node a “coreness” value. Mathematically speaking, the difference with the discrete model is tiny. The quality

Figure 32.2: A toy example of the Continuous Model for core-periphery structures. This network shows a densely connected core (blue), a pure periphery whose nodes do not connect to each other (green), and an intermediate stage which is not dense enough to be part of the core, but whose nodes still connect to each other (purple).

measure we want to optimize is still $\sum_{uv} A_{uv}\Delta_{uv}$. However, now the entries of Δ are not binary any more. Instead we have $\Delta_{uv} = c_u c_v$, with c_u being the coreness value of node u .

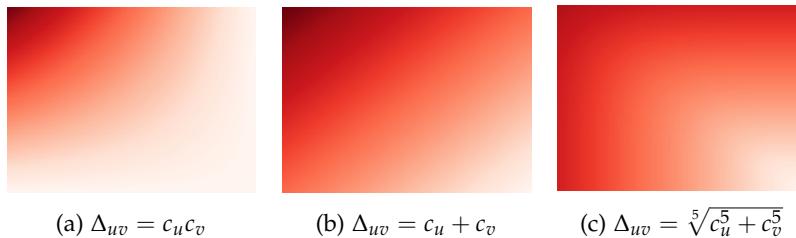
For instance, we could say that, in Figure 32.2, the nodes in the blue circle have coreness equal to 1, the ones in purple have coreness equal to 0.5, and the rest has coreness equal to 0. If you force nodes to have a coreness of either 1 or 0, you're back in the discrete model.

How could one establish the c_u values for the Continuous Model? I already mentioned the k-core centrality algorithm from Section 14.7 in this chapter. This could be a good choice, because we know that nodes with low k-core centrality have a low degree, while nodes with a high k-core value tend to connect to each other in a core.

Of course, that is an a priori approach: it fixes your Δ , which may or may not fit your data well. The alternative is to use a technique similar to the ones I mentioned before, to build your Δ via the c vector in such a way that your quality measure is maximized.

Other Approaches

The continuous model is powerful, but it doesn't really tell you much on how you should build your c_i values. Rombach et al.⁷ propose a way to build such vector, introducing two parameters, α and β . β determines the size of the core, from the entirety of the network to an empty core. α regulates the c score difference between the core classes. If a node u at a specific core level has a score c_u , the node v at the closest highest core level will have $c_v = \alpha + c_u$ – or, really, any function taking α as a parameter.



Another freedom you can take is to build Δ differently. In the standard continuous model, we build it via multiplication: $\Delta_{uv} = c_u c_v$. An alternative is to build it via a p-normalization: $\Delta_{uv} = \sqrt[p]{c_u^p + c_v^p}$. The higher p , the more weight you're putting into a classic discrete model core. Figure 32.3 shows the different effects of different criteria to build Δ .

Other approaches in the literature make use of the Expectation Maximization or the Belief Propagation algorithms⁸.

⁷ M Puck Rombach, Mason A Porter, James H Fowler, and Peter J Mucha. Core-periphery structure in networks. *SIAM Journal on Applied mathematics*, 74(1):167–190, 2014

Figure 32.3: Some matrix masks you can use to build Δ . The darkness of the color is directly proportional to how much the core value combination contributes to Δ . Matrices are sorted so that the top-left corner shows the value for $c_u = c_v = 1$ and the bottom-right corner shows the value for $c_u = c_v = 0$.

⁸ Xiao Zhang, Travis Martin, and Mark EJ Newman. Identification of core-periphery structure in networks. *Physical Review E*, 91(3):032803, 2015

All methods discussed so far detect the core via a statistical inference or the development of a null model. Thus there are issues of scalability when you need to infer the parameters of the model to make the proper inference. Alternative methods exploit the fact that the core is densely connected and nodes have a high degree. Thus, the expectation is that a random walker would be trapped for a long time in the core⁹. By analyzing the behavior of a random walker, one could detect the boundary of the core.

What about multilayer networks (Section 7.2)? Does it make sense to talk about a core in a network spanning multiple interconnected layers? The analysis of some naturally occurring multilayer networks, for instance the brain connectome^{10,11}, suggest that it is. However, I would say that at the moment of writing this paragraph, a systematic investigation of core-periphery in multilayer networks, along with a general method to detect them, is an open problem in network science.

32.2 Tension with Communities

There is a tension between core-periphery structures (CP) and the classical community discovery (CD) assumption: in CP there isn't space for communities, given that there's only one dense area and everything connects to it. In CD, there's little space for peripheries, and there are multiple cores.

You can see this mathematically. For simplicity, let's consider the discrete model: $\sum_{uv} A_{ij}\Delta_{uv}$. There is a strong correlation between being an high degree node and being in the core: after all, the nodes in the core are highly connected. We'll see that a community is a set of nodes densely connected to each other. Thus, nodes in a community have a relatively high degree and should be considered part of the core. So, for all nodes deeply embedded in a community, $\Delta_{uv} = 1$.

However, a traditional community is also sparsely connected to nodes outside the community. This means that, if nodes u and v are in different communities, likely $A_{uv} = 0$. But we just saw that their Δ_{uv} should be 1 because they have high degree! All of that score is wasted! Maximizing the quality function would imply to put all nodes in the same core, sacrificing the defining characteristic of a core: the fact that nodes in it should tend to connect to each other. Figure 32.4 shows the difference between the two archetypal meso-scale organizations.

This is problematic since we have evidence that core-periphery structures are ubiquitous, and so are communities. There are a couple of explanations we can use to restore our sanity.

The first explanation is realizing that every network lives on a

⁹ Athen Ma and Raúl J Mondragón. Rich-cores in networks. *Plos one*, 10(3):e0119678, 2015

¹⁰ Federico Battiston, Jeremy Guillon, Mario Chavez, Vito Latora, and Fabrizio De Vico Fallani. Multiplex core-periphery organization of the human connectome. *Journal of the Royal Society Interface*, 15(146):20180514, 2018

¹¹ Jeremy Guillon, Mario Chavez, Federico Battiston, Yohan Attal, Valentina La Corte, Michel Thiebaut de Schotten, Bruno Dubois, Denis Schwartz, Olivier Colliot, and Fabrizio De Vico Fallani. Disrupted core-periphery structure of multimodal brain networks in alzheimer's disease. *Network Neuroscience*, 3(2):635–652, 2019

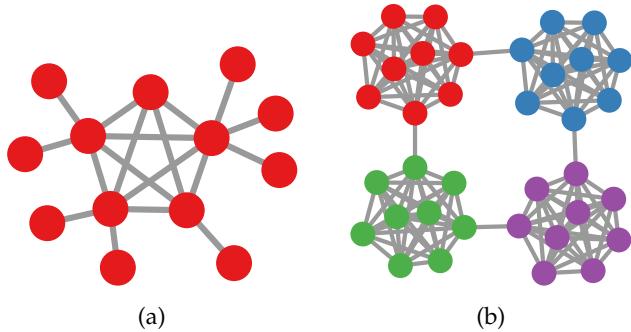
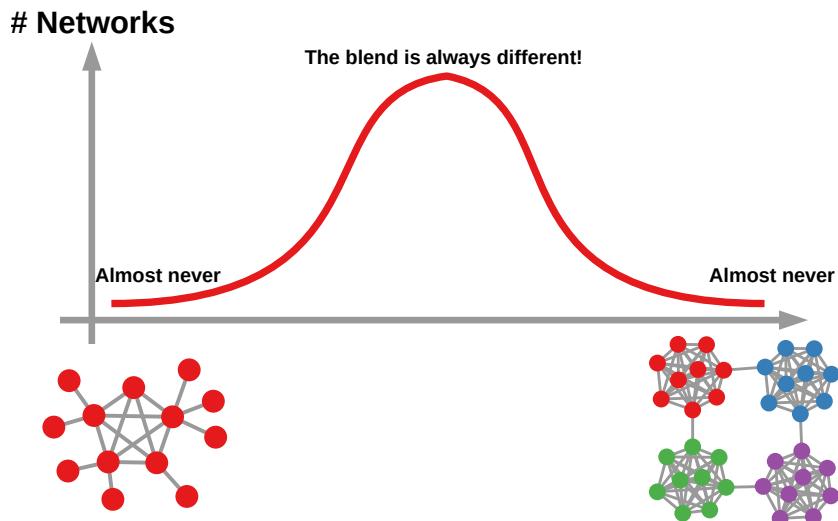


Figure 32.4: (a) A classical core-periphery structure. (b) A classical community structure.

core-periphery to community structure continuum. The real world networks we observe distribute through this continuum in such a way that perfect instances are extremely rare – as Figure 32.5 shows. You’ll very rarely find a natural discrete model, exactly as rarely as finding a real world network organizing like a caveman graph (Section 17.1) – the quintessential community structure. As a consequence, one way to solve this conundrum is to admit that a network might have multiple cores. Thus, one first performs community discovery to find the multiple cores and then applies the core-periphery detection algorithm independently on each community¹².



¹² Sadamori Kojaku and Naoki Masuda. Core-periphery structure requires something else in the network. *New Journal of Physics*, 20(4):043012, 2018

Figure 32.5: The number of networks with a pure core-periphery network and with a pure community structure is actually tiny.

In most cases, networks are in a middle way. Another important thing to keep in mind is that this continuum is not monodimensional – although on this page I had to squeeze it on a line. The way in which each network blends core-peripheries with communities is always different and difficult to fully characterize.

There are many mechanisms that makes this the case. One is

related to overlapping communities (Chapter 38). These communities allow nodes to belong to multiple groups at the same time. These nodes in between communities can be considered a special core of the network¹³, bringing together the community structure with the core-periphery organization.

Alternative explanations use the power of random walkers to explain core-peripheries¹⁴, an approach that is also commonly used in community discovery. Other models attempt to embed nodes into a spatial dimensions, showing how this can create core-periphery structures¹⁵. It is following this example that I'll try to draw a connection between core-periphery and some well studied aspects of economics in the next section.

32.3 Emergence from Social Behavior

The emergence of core-periphery structures can be observed in many systems. I'll make one example from economic geography.

Consider Hotelling's law¹⁶. Suppose that you are on a beach with two ice cream vendors of equal quality. People want ice cream and, since the two vendors offer the same quality, the customers will go to the closest vendor. Therefore, a rational vendor would move their stand so that it can capture the people in the middle. The other vendor would do the same. The solution is an equilibrium in which vendors concentrate in the middle, even though that means increasing the walk length for every customer.

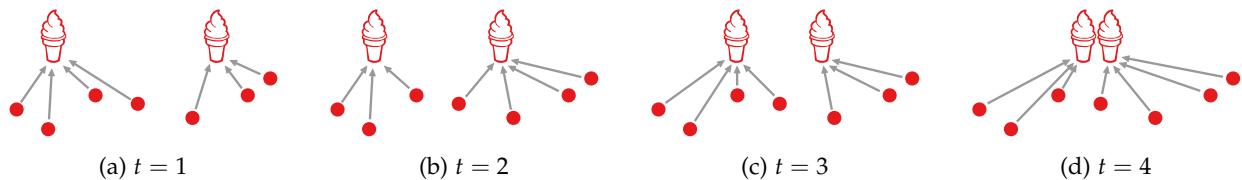


Figure 32.6 captures the law in all of its enraging, negative-sum, glory. The situation hardly changed for the businesses, while the customers are served less efficiently.

This is observed in reality, at multiple levels. Consider a federal government structure, where state governments coalesce into state capitals that need to be visited by their peripheries. In turn, federal capitals provide even higher level services and thus attract people from each state.

This is related to ekistics, the science of human settlements¹⁷. In ekistics, Doxiadis shows how improvements in human transportation have introduced a cumulative advantage for the best connected city centers. These are swallowing up their surroundings, sparsify-

¹³ Jaewon Yang and Jure Leskovec. Overlapping communities explain core-periphery organization of networks. *Proceedings of the IEEE*, 102(12):1892–1902, 2014

¹⁴ Fabio Della Rossa, Fabio Dercole, and Carlo Piccardi. Profiling core-periphery network structure by random walkers. *Scientific reports*, 3:1467, 2013

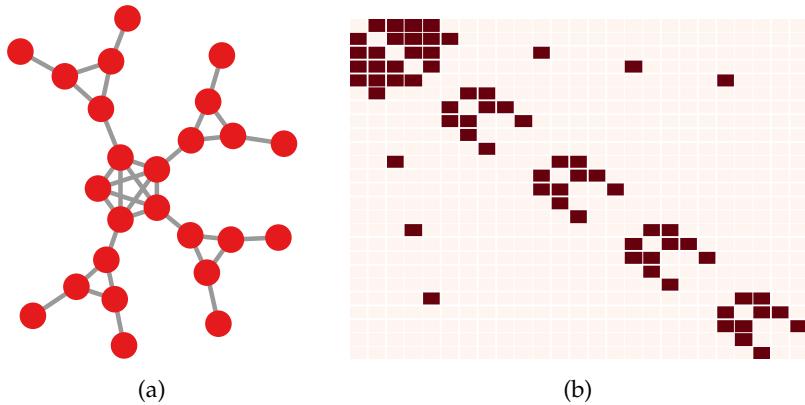
¹⁵ Daniel A Hojman and Adam Szeidl. Core and periphery in networks. *Journal of Economic Theory*, 139(1):295–309, 2008

¹⁶ Harold Hotelling. Stability in competition. *The Economic Journal*, 39(153):41–57, 1929

Figure 32.6: A depiction of Hotelling's Law. Each customer (circle) walks to the closest ice-cream stand. As times goes by, the stands get closer and closer, to capture the people in the middle.

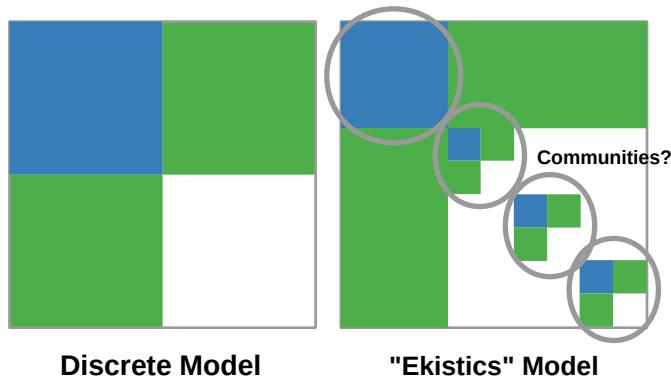
¹⁷ Constantinos Apostolos Doxiadis et al. Ekistics; an introduction to the science of human settlements. 1968

ing periphery-periphery connections and attracting all wealth and connections for themselves, creating a centralized rich club.



If we look at the types of networks generated with such theory in mind – for instance Figure 32.7 –, we realize they do not conform very much to the strict core-periphery structure imposed by the Discrete Model, and they are not quite the same as the Continuous Model, due to their multi-core nature.

However, their schematic structure brings us some sort of closure when it comes to the tension between core-periphery and community discovery. The networks do have a core, and also some sort of communities, coalescing around the secondary cores. Figure 32.8 shows a possible model representing a temporary core-periphery organization – which ekistics predicts to eventually collapse into a single core. These networks are very similar to the Kronecker graphs we saw in Section 18.2, in all their fractal glory.



The Central Place Theory in economic geography¹⁸ is an alternative and less pessimistic interpretation of geographical core-periphery structures. It says that settlements function as “central places” providing services to surrounding areas. The more sophisticated the

Figure 32.7: A toy example of a network built accordingly to Doxiadis' ekistics. (Left) Graph view: the most central 5-clique is the union of 5 settlements that make arise the best connected metropolis. The smaller 3-cliques represent other competing cities connecting to their peripheries, which are bound to be eventually absorbed by the core. (Right) Adjacency matrix view.

Figure 32.8: Two schematic representations of ideal core-periphery structures: the discrete model on the left and the ekistics model on the right. The blue area represents a densely connected set of nodes, while the green area is a sparser periphery.

¹⁸ Walter Christaller. *Central places in southern Germany*. Prentice Hall, 1966

service the harder it is to provide, and thus it requires more skill integration and thus more centralization. If we keep centralizing sophisticated services, we end up with a high-level central core, several secondary lower level cores and various peripheries, in a fractal way. However, the conceptual jump to core-periphery structures is larger here, because to explain central place we need to introduce this service-providing system – with sophistication and skills –, while ekistics uses the simpler cumulative advantage mechanics that we already know are a part of many networked systems.

32.4 Nestedness

Core-periphery structures are a generalization of a specific meso-scale organization of complex systems that is relevant in multiple fields: nestedness¹⁹. A nested system is one where the elements containing few items only contain a subset of the items of elements with more items.

In terms of networks, an ideal nested system has a hub which is connected to all nodes in the network. The node with the second largest degree is connected to a subset of the neighbors of a hub. The third largest degree node is connected, in turn, to a subset of the neighbors of the second node. It rarely connects to nodes not connected to its antecedent in this hierarchy²⁰.

Nestedness was originally developed in ecology studies^{21,22,23} – specifically biogeography. It originated from an observation of related ecosystems. Suppose you have a mainland with a set of species. Then you have an archipelago, with many islands at an increasing distance from the coast. The closest island contains all the species able to cross water. The second island contains species that can cross water and have a slightly higher range. The third island contains only species with a further increased range, and so on. Clearly, the species which did not make it to the second island are unable to get to the third. Thus the third island is a perfect subset of the second.

I should point out that this explanation of nestedness is not the only one in literature: other authors suggest that nestedness could arise simply from the degree distribution²⁴, and that there are fewer nested system than we originally thought.

A typical nested network is bipartite. One node type, in our ecology case, is the species, and the other is the ecosystem. If you were to plot the adjacency matrix of such network, you'd end up with a picture that looks like Figure 32.9. To highlight the nested pattern, you want to plot the matrix sorting rows and columns by their sum in descending order. If you do so, most of the connections end up in

¹⁹ Sang Hoon Lee et al. Network nestedness as generalized core-periphery structures. *Physical Review E*, 93(2):022306, 2016

²⁰ Jordi Bascompte, Pedro Jordano, Carlos J Melián, and Jens M Olesen. The nested assembly of plant–animal mutualistic networks. *Proceedings of the National Academy of Sciences*, 100(16):9383–9387, 2003

²¹ Robert H Mac Arthur and Edward Osborneouat Wilson. The theory of island biogeography. Technical report, 1967

²² Bruce D Patterson. The principle of nested subsets and its implications for biological conservation. *Conservation Biology*, 1(4):323–334, 1987

²³ Ugo Bastolla, Miguel A Fortuna, Alberto Pascual-García, Antonio Ferrera, Bartolo Luque, and Jordi Bascompte. The architecture of mutualistic networks minimizes competition and increases biodiversity. *Nature*, 458(7241):1018, 2009

²⁴ Claudia Payrató-Borras, Laura Hernández, and Yamir Moreno. Breaking the spell of nestedness: The entropic origin of nestedness in mutualistic systems. *Physical Review X*, 9(3):031024, 2019

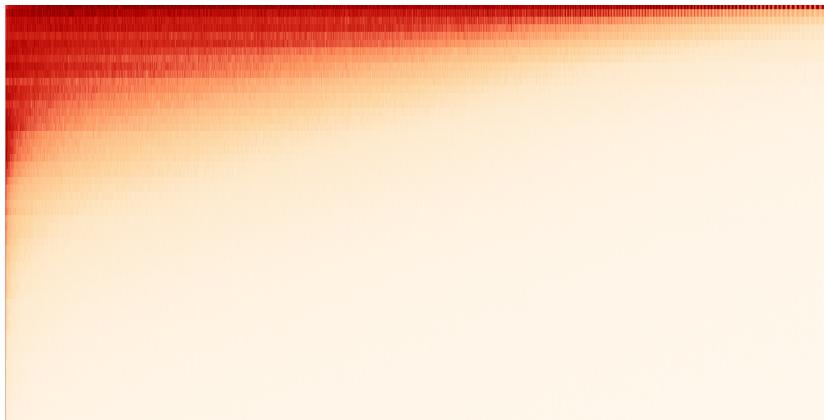


Figure 32.9: A matrix view of a nested network. The matrix has species on the columns and ecosystems on the rows.

the top-left of the matrix. That is why we often call nested matrices “upper-triangular”.

Just like in the discrete model, it’s also rare for a real world network to be perfectly nested. Thus, researchers developed methods to calculate the degree of nestedness of a matrix^{25,26,27}. I’m going to give you a highly simplified view of the field.

These measures are usually based on the concept of temperature, using an analogy from physics. A perfectly nested matrix has a temperature of zero, because all the “particles” (the ones in the matrix) are frozen where they should be: in the upper-triangular portion. Every particle shooting outside its designated spot increases the temperature of the matrix, until you reach a fully random matrix which has a temperature of 100 degrees.

A key concept you need to calculate a matrix’s temperature is the isocline. The isocline is the ideal line separating the ones from the zeroes in the matrix. You should try to draw a line such that most of the ones are on one side and most of the zeros are on the other. Usually, there are two ways to go about it.

The parametric way is to figure out which function best describes the curve in your upper triangular matrix: a straight line, a parabolic, a hyperbolic curve, the ubiquitous and mighty power-law. Then you fit the parameters so that the isocline snugs as close as possible to said border.

The non-parametric way is to simply create a jagged line following the row sum or the column sum. If your row (or column) sums to 50, then you expect a perfectly nested matrix to have 50 ones followed only by zeros. So your isocline should pass through that point.

Once you have your isocline, you can simply calculate how many mistakes you made: how many ones are on the side of the zeroes, and vice versa? Figure 32.10 shows an example of the different levels

²⁵ Wirt Atmar and Bruce D Patterson. The measure of order and disorder in the distribution of species in fragmented habitat. *Oecologia*, 96(3):373–382, 1993

²⁶ Paulo R Guimaraes Jr and Paulo Guimaraes. Improving the analyses of nestedness for large sets of matrices. *Environmental Modelling & Software*, 21(10):1512–1513, 2006

²⁷ Samuel Jonhson, Virginia Domínguez-García, and Miguel A Muñoz. Factors determining nestedness in complex networks. *PloS one*, 8(9):e74025, 2013

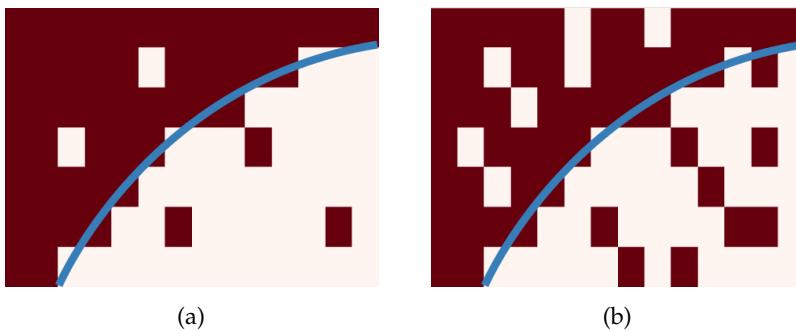


Figure 32.10: Two matrices at a different level of nestedness: (a) high nestedness (low temperature), (b) low nestedness (high temperature). The line in blue is the best isocline.

of nestedness two toy matrices can have.

I'm mentioning nestedness in this book because I have encountered it a surprisingly high amount of times in my research, in complex systems that have nothing to do with ecology – economics for instance.

Some colleagues built a bipartite network connecting countries to the products they can export successfully in the global market^{28,29}. The most competitive export-oriented economies (Japan, Germany, ...) are able to have an export edge in almost any product, no matter how complex. As you go down the ladder of competitiveness, the countries are able to export a subset of what their immediate higher ranked country can. When you get to the least diversified economies in the world, you end up with countries that are able to export only the products that every country in the world can make.

I personally found nested patterns in a supermarket matrix, with customers and products as the two node types³⁰. A connection goes from a customer to the product they bought in significant quantities. Nestedness emerges as there are different customer types, organized in a continuum: from those who buy everything they need in the shop we studied, to those who only buy immediate necessities.

Note that I constantly used adjacency matrix pictures to convey the ideas behind core-periphery and communities – from Figure 32.1 to Figure 32.10. This is because one could unify core-periphery and communities in a single model using a mixing matrix, which is at the basis of using stochastic blockmodels (Section 18.2) to find communities (Section 35.1) and/or cores in networks.

32.5 Summary

1. A core-periphery structure is a meso-level organization of a complex network in two parts: one set of nodes densely interconnected with each other (the core) and a sparsely connected set of nodes with just one or few connections per node to the core (the periph-

²⁸ Sebastián Bustos, Charles Gomez, Ricardo Hausmann, and César A Hidalgo. The dynamics of nestedness predicts the evolution of industrial ecosystems. *PLoS one*, 7(11):e49393, 2012

²⁹ Matthieu Cristelli, Andrea Tacchella, and Luciano Pietronero. The heterogeneous dynamics of economic complexity. *PLoS one*, 10(2):e0117174, 2015

³⁰ Diego Pennacchioli, Michele Coscia, Salvatore Rinzivillo, Fosca Giannotti, and Dino Pedreschi. The retail market as a complex system. *EPJ Data Science*, 3(1):33, 2014

ery).

2. The discrete model allows to detect such structures by penalizing periphery-periphery connections. Other approaches, such as the continuous model, are more flexible and allow for varying degrees of “coreness”.
3. Core-periphery structures are ubiquitous just as community structures are, yet a pure core-periphery structure is incompatible with the general notion of community. In reality, these two meso-scale organizations of complex networks co-exist on a spectrum.
4. Many real world dynamics may be at the basis of a core-periphery structure. For instance, geographical agglomeration – i.e. the creation of a localized core – makes sense when combining skills to provide complex services in an economy.
5. Nestedness in ecology and economics is another classical core-periphery structure for bipartite networks. In an archipelago, islands with the most species have all species, while islands with few species only have the species that are present in all islands.

32.6 Exercises

1. Install the `cpalgorithm` library (`sudo pip install cpalgorithm`) and use it to find the core of the network using the discrete model (`cpa.BE`) and Rombach’s model (`cpa.Rombach`) on the network at <http://www.networkatlas.eu/exercises/32/1/data.txt>. Use the default parameter values. (Warning, Rombach’s method will take a while) Assume that Rombach’s method puts in the core all nodes with a score higher than 0.75. What is the Jaccard coefficient between the cores extracted with the two methods?
2. The network at <http://www.networkatlas.eu/exercises/32/2/data.txt> has multiple cores/communities. Use the Divisive algorithm from `cpalgorithm` to find the multiple cores in the network.
3. <http://www.networkatlas.eu/exercises/32/3/data.txt> contains a nested bipartite network. Draw its adjacency matrix, sorting rows and columns by their degree.

33

Hierarchies

We usually represent organizations with networks. Each person in the company is a node. Directed edges connect nodes. They usually flow from the superior to the subordinate, from the coordinator to its team. These networks have a particular structure. In an ideal organization, there is a single head. If this were a corporation, that would be the CEO. The head commands a small group, its top level executives. They, in turn, have their own team of managers. The managers command their own groups, and so on and so forth, until we get to the foot soldiers.

Hierarchical networks arise not only in social systems¹, but also – and especially – in biological ones^{2,3,4,5,6,7}.

This is some sort of core-periphery structure (Chapter 32), with a few distinctions.

First, in core-periphery there's still some degree of horizontal connections. People at the same level are able to connect to each other. In a perfect hierarchy that is not the case: horizontal connections are banned. You can assign each worker to a level, and workers can only connect to lower level – and being connected by higher levels.

Second, usually hierarchical networks are directed, while core-periphery normally doesn't care all that much about the direction of the edges.

We can then consider hierarchies as some sort of special case of core-periphery structures. In this chapter we'll explore the concept of hierarchical networks, as it can be interpreted in multiple ways. We're then going to introduce key concepts – and the main methods using such concepts – to estimate the “hierarchicalness” of a directed network.

33.1 Types of Hierarchies

When talking about “hierarchies” in complex networks, researchers mainly refer to three related but distinct concepts. We can classify

¹ Aaron Clauset, Samuel Arbesman, and Daniel B Larremore. Systematic inequality and hierarchy in faculty hiring networks. *Science advances*, 1(1):e1400005, 2015

² Erzsébet Ravasz, Anna Lisa Somera, Dale A Mongru, Zoltán N Oltvai, and A-L Barabási. Hierarchical organization of modularity in metabolic networks. *science*, 297(5586):1551–1555, 2002

³ Erzsébet Ravasz and Albert-László Barabási. Hierarchical organization in complex networks. *Physical review E*, 67(2):026112, 2003

⁴ Haiyuan Yu and Mark Gerstein. Genomic analysis of the hierarchical structure of regulatory networks. *Proceedings of the National Academy of Sciences*, 103(40):14724–14731, 2006

⁵ A Vazquez, R Dobrin, D Sergi, J-P Eckmann, ZN Oltvai, and A-L Barabási. The topological relationship between the large-scale attributes and local interaction patterns of complex networks. *Proceedings of the National Academy of Sciences*, 101(52):17940–17945, 2004

⁶ Peter Csermely, Tamás Körkemáros, Huba JM Kiss, Gabor London, and Ruth Nussinov. Structure and dynamics of molecular networks: a novel paradigm of drug discovery: a comprehensive review. *Pharmacology & therapeutics*, 138(3):333–408, 2013a

⁷ Samuel Johnson and Nick S Jones. Looplessness in networks is linked to trophic coherence. *Proceedings of the National Academy of Sciences*, 114(22):5618–5623, 2017

them in three categories: order, nested, and flow hierarchy⁸. I'll present the three of them in this section, noting how this chapter will then only focus on flow hierarchy. Order and nested hierarchies are covered elsewhere in this book with different names.

Order

In an order hierarchy, the objective is to determine the order in which to sort nodes. We want to place each node to its corresponding level, according to the topology of its connections. Usually, this is achieved by calculating some sort of centrality score. The most central nodes are placed on top and the least central on the bottom.

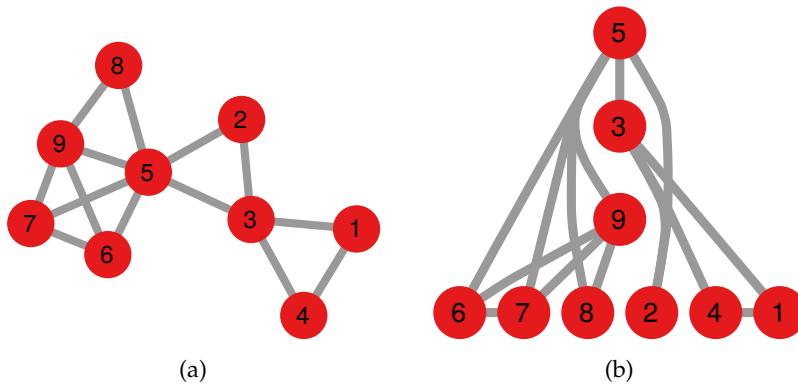


Figure 33.1 provides an example of detecting an order hierarchy in a toy network, using betweenness centrality as the guiding principle. Node 5 has the highest betweenness centrality, followed by node 3 and then node 9. All other nodes have the same betweenness centrality – equal to zero.

Perhaps, the most famous example in this class is SpringRank⁹. SpringRank only works for directed networks. It sees an $u \leftarrow v$ edge as a sign that node u is above v in the hierarchy. SpringRank tries to put all nodes in a specific height – h_u and h_v for nodes u and v respectively. As the name suggests, it sees each edge as a spring. Therefore, for any positioning h_u and h_v of the nodes, it can calculate the energy in the spring, which is $H_{uv} = \frac{1}{2}(h_u - h_v - 1)^2$ – this is the literal amount of energy in a literal spring of that literal size, it's a purely physical approach. Once you have the energy for a pair of nodes, you can calculate the energy for the entire network, which is the sum of the energies of all edges that exist or, mathematically:

$$H_G = \sum_{u,v \in V} A_{uv} H_{uv} = \frac{1}{2} \sum_{u,v \in V} A_{uv} (h_u - h_v - 1)^2,$$

the A_{uv} term is the adjacency matrix and will cancel to zero the

⁸ Enys Mones, Lilla Vicsek, and Tamás Vicsek. Hierarchy measure for complex networks. *PLoS one*, 7(3):e33799, 2012

Figure 33.1: (a) A toy network. (b) Its order hierarchy. I place nodes in descending order of betweenness centrality from top to bottom.

⁹ Caterina De Bacco, Daniel B Larremore, and Christopher Moore. A physical model for efficient ranking in networks. *Science advances*, 4(7):eaar8260, 2018

energies of edges that don't exist, because in that case $A_{uv} = 0$. Then SpringRank finds a few clever ways to minimize this quantity, which involve the pseudoinverse of the Laplacian we discussed in Section 11.4.

In general, solutions to the order hierarchy result in a continuous value of each node in the network that puts it on the y axis on a line. This is quite literally equivalent to finding a *ranking* of nodes in the network. One can easily see that we already covered this sense of hierarchical organization of complex networks. The order hierarchy is nothing more than a different point of view of node centrality. Thus, I refer to Chapter 14 for a deeper discussion on the topic.

Nested

Nested hierarchy is about finding higher-order structures that fully contain lower order structures, at different levels ultimately ending in nodes. In the corporation example, the largest group is the corporation itself, encompassing all workers. We can first subdivide the corporation into branches, if it is a multinational, they could be regional offices. Each office can be broken down into different departments, which have teams and, finally, the workers in each team.

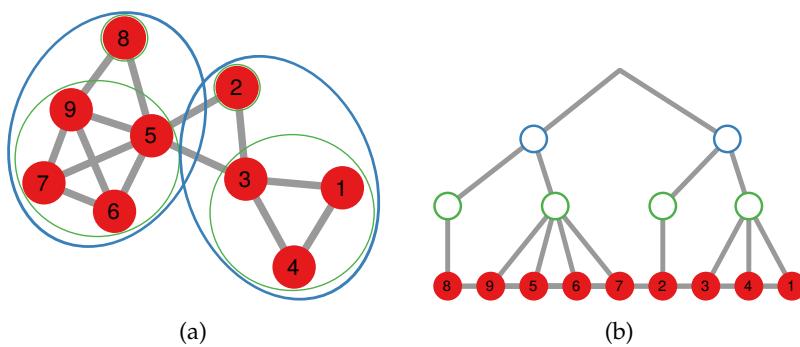


Figure 33.2: (a) A toy network. Colored circles delineate nested substructures. (b) Its nested hierarchy, according to the highlighted substructures. Each node and substructure is connected to the substructure it belongs to.

Figure 33.2 provides an example of detecting a nested hierarchy in a toy network. This is usually done by detecting smaller and smaller densely connected units in the network. Note how, in this case, the hierarchy does not place nodes on levels, but organizes the detected substructures.

This is equivalent to performing hierarchical community discovery on complex networks. Thus, I refer to Chapter 37 for further reading.

Flow

In a flow hierarchy, nodes in a higher level connect to nodes at the level directly beneath it, and can be seen as managers spreading infor-

mation or messages to the lower levels. We call it a “flow” hierarchy because you can see the highest level node as the origin of a flow, which hits first the nodes at the level directly beneath it, and so on until it reaches the leaves of the network: the nodes at the bottom layer.

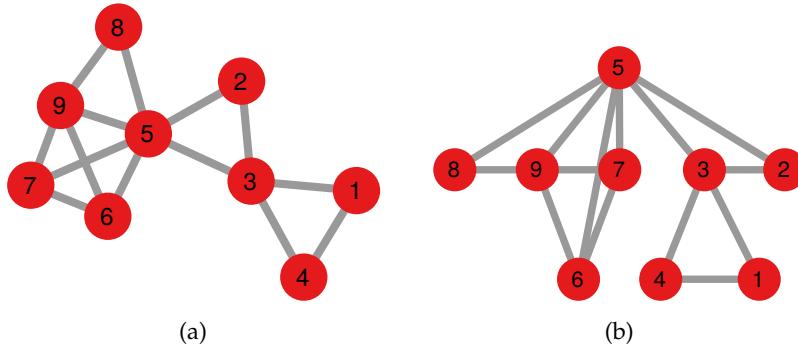


Figure 33.3: (a) A toy network.
(b) Its flow hierarchy.

Figure 33.3 provides an example of detecting a flow hierarchy in a toy network. Note how it tends to have high centrality nodes on top, like the order hierarchy, but it creates a substantially different organization. Nodes directly connected to a given level tend to belong to the level immediately beneath it, no matter their different centrality values. One could think that the order hierarchy is a special case of flow hierarchy, but that is incorrect: in a flow hierarchy all nodes belonging to a level need to connect to the level directly below and above, while that’s not the case for the order hierarchy. Moreover, the order hierarchy is just something we add on top of a node-level measure (centrality), while the flow hierarchy is a meso-level analysis: it describes how groups of nodes – the ones at different hierarchical levels – relate to each other.

Since this concept is not covered elsewhere in this book, I focus on flow hierarchies for the rest of this chapter. There are four main approaches to detect hierarchies and quantify the hierarchicalness of real world directed networks. They are: cycle-based Flow Centrality, Global Reach Centrality (GRC), Agony, and Arborescence.

From now on, we always assume that the network we’re analyzing is directed.

33.2 Cycles

I introduced the concept of cycles in Section 10.2: special paths that start and end in the same node. Cycles are natural enemies of hierarchies. There is no way to have a perfect hierarchy if you have a cycle in your network. In a perfect hierarchy, you can always tell who’s your boss. Your boss will never take orders from you, nor from

your peer, and even less so from any of your underlings.

However, if you have a cycle, that is not true. A cycle means that your boss gives you an order, you pass it down to one of your underlings and, somehow, they give it back to your boss. This is clear nonsense and should be avoided at all costs.

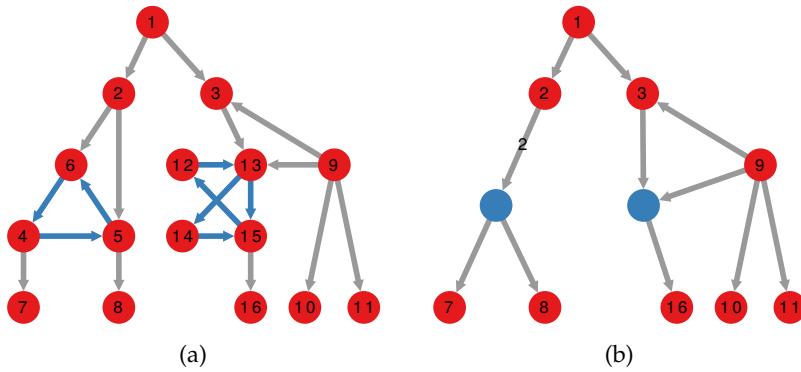


Figure 33.4: (a) A directed network. In the figure, I highlight in blue the edges partaking in cycles. (b) The condensed version of (a), where all nodes part of a strongly connected component are condensed in a node (colored in blue).

Thus, the simplest way to estimate the hierarchicalness of a directed network is to count the number of edges involved in a cycle. The fewer edges are part of a cycle in a network, the more hierarchical it is¹⁰. You can see in Figure 33.4(a) that the somewhat hierarchical network has only a handful of edges involved in cycles.

You can calculate the flow hierarchicalness of a network by simply condensing the graph. Graph condensation follows two steps. First, you reduce all the graph's strongly connected components each to a single node. Then you connect that node to all nodes that were connected to a node part of that component. When condensing the graph in Figure 33.4(a), you'll obtain the graph in Figure 33.4(b). Note that, if you're merging two edges, you need to keep track of this information, for instance in the edge weight. The (2,5) and (2,6) edges collapse in a single edge outgoing from node 2, which now must have a weight of two.

The ratio between the sum of the edge weights in the condensed graph and the number of edges in the original graph is the flow hierarchy of your network. The original graph in Figure 33.4(a) had 20 edges. Since the condensed graph in Figure 33.4(b) has 11 edges with a total weight sum of 12 (because of the edge of weight two coming out of node 2), we can conclude that the network's flow hierarchy is equal to $12/20 = 0.6$.

A consequence of this definition is that any directed network composed by a single strongly connected component has a hierarchicalness of zero by definition.

Flow hierarchy has a major flaw: it's too lenient. In fact, it says that any and all directed acyclic graphs are perfect hierarchies. It

¹⁰ Jianxi Luo and Christopher L Magee. Detecting evolving patterns of self-organizing networks by flow hierarchy measurement. *Complexity*, 16(6):53–61, 2011

is very easy to construct toy examples of less-than-ideal structures that this cycle-based flow hierarchy will consider perfect. Figure 33.5 shows two of such examples.

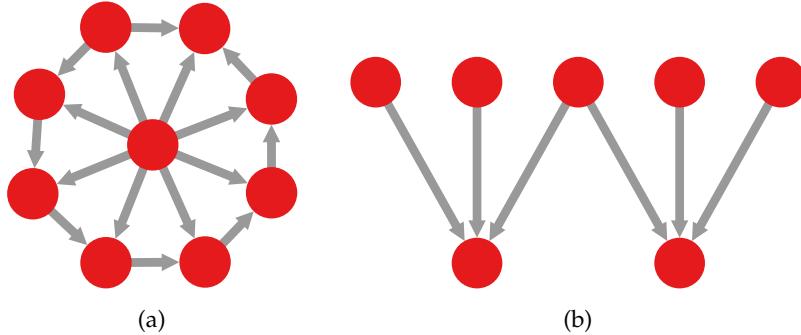


Figure 33.5: Two directed acyclic graphs not conforming to our intuition of a perfect hierarchy. (a) A wheel graph with one flipped edge. (b) A “hierarchy” with more bosses than workers.

In Figure 33.5(a) we have no cycles because I flipped one edge. However, arguably, one should not be able to go from a perfect hierarchy to less than 50% hierarchicalness by simply flipping one direction. Figure 33.5(b) shows another case where there are more bosses than workers: you don’t need to have a master in management to see that this is no way to run an organization! Yet, since they are both directed acyclic graphs, for this cycle-based definition of hierarchy, both toy examples are ideal hierarchies.

33.3 Global Reach Centrality

I introduced the concept of reach centrality back in Section 14.3: the reach centrality of node v in a directed network is the fraction of nodes it can reach using directed paths originating from itself. This measure is often dubbed “local” reach centrality, because the same authors defined a “global” reach centrality¹¹ (GRC). GRC is not a measure for nodes any more: it is a way to estimate the hierarchicalness of a network.

The intuition behind GRC is simple. A network has a strong hierarchy if there is a node which has an overwhelming reaching power compared to the average of all other nodes. Or, to put it in other words, if there is an overseer that sees all and knows all. To calculate GRC you first estimate the local reach centrality of all nodes in the network. You then find the maximum value among them, say LRC_{MAX} . Then:

$$GRC = \frac{1}{|V| - 1} \sum_{v \in V} LRC_{MAX} - LRC_v.$$

In practice, you average out its difference with all reach centrality values in the network. This is an effective way of counteracting the

¹¹ Enys Mones. Hierarchy in directed random networks. *Physical Review E*, 87(2):022817, 2013

degeneracy of cycle-based hierarchy measures. In both toy examples from Figure 33.5, GRC is well behaved, returning values of 0.555 and 0.22, respectively.

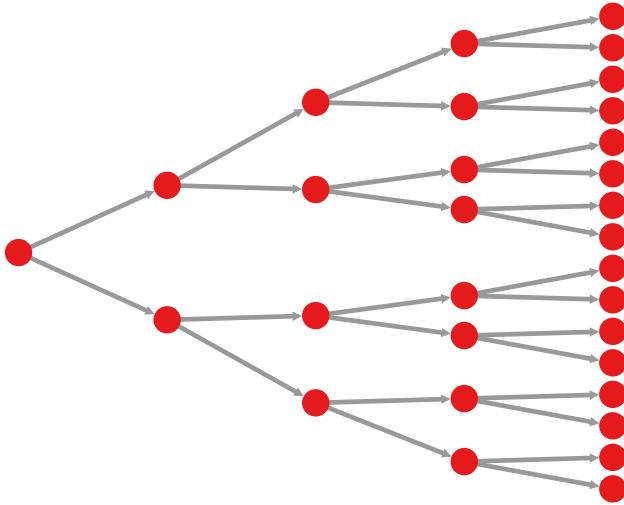


Figure 33.6: A network we could consider a perfect hierarchy, for which GRC fails to give a perfect score.

However, GRC has a blind spot of its own. Since we're averaging the differences between the most central node against all others, we know we will never get a perfect GRC score if there is more than one node with non-zero local reach centrality. Consider Figure 33.6. I don't know about you, but to me it looks like a pretty darn perfect hierarchy. Yet, we know that the two nodes connected by the root don't have a zero local reach centrality. In fact, the GRC for that network is 0.89.

So, if cycle-based flow hierarchy is too lenient – every directed acyclic graph is a perfect hierarchy –, GRC is too strict: even flawless hierarchies might fail to get a 100% score. If for cycle-based flow hierarchy the perfect hierarchy is a DAG, for GRC the only perfect hierarchy is a star: a central node connected to everything else, and no other connections in the network.

33.4 Arborescences

We introduced the concept of arborescence in Section 10.2, as a stricter definition of a directed acyclic graph. To sum up: an arborescence is a directed tree in which all nodes have in-degree of one, except the root, which has in-degree of zero. For instance, Figure 33.6 is an arborescence. Given their properties, arborescences seem particularly well suited to inform us about hierarchies. Every arborescence is a perfect hierarchy: all nodes have a single boss, there are no cycles, and there is one node with no bosses – the CEO.

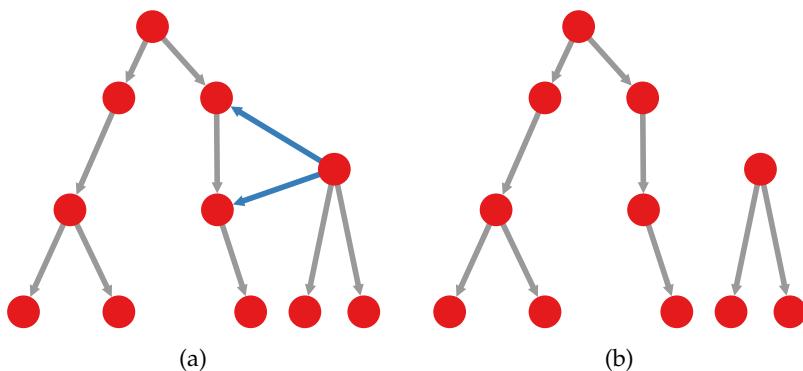
That is why I used them to create my own hierarchicalness score¹².

¹² Michele Coscia. Using arborescences to estimate hierarchicalness in directed complex networks. *PloS one*, 13(1): e0190825, 2018

I take an approach similar to the one developed from the cycle-based flow hierarchy. In fact, the first step is almost identical: take your directed graph and condense all its strongly connected components into a single node. The only difference is that here we ignore edge weights. This gives us a directed acyclic graph version of the original network. Note that there are alternative methods to reduce a generic directed network to a DAG^{13,14}, which can preserve more edges.

To transform it in an arborescence, we need to remove all edges going “against” the flow. We cannot allow any node to have an in-degree larger than one. So all edges contributing to a larger-than-one in-degree have to be removed. We cannot remove them at random: we need to remove the ones pointing towards the root and keep the ones pointing away from it. I use closeness centrality to determine which edges to keep: the ones coming from the node with the lowest closeness centrality. This is a bit counter-intuitive: the closer a node is to the root, the lower its closeness centrality is. This is due to the fact that these nodes have more possible paths originating from them, and they tend to be longer because they can reach a larger portion of the network.

Once all edges breaking the arborescence requirements are eliminated, we can count how many connections survived. This is also equivalent to the final step of the cycle-based flow hierarchy. The more edges we needed to remove to obtain an arborescence, the less the original network was resembling a perfect hierarchy. In the example from Figure 33.4, we would preserve nine edges out of 20, giving us an arborescence score of $9/20 = 0.45$. Differently from the cycle-based measure, the arborescence score is not fooled by the examples in Figure 33.5, returning a score of 0.5 and 0.33, respectively.



¹³ Can Lu, Jeffrey Xu Yu, Rong-Hua Li, and Hao Wei. Exploring hierarchies in online social networks. *IEEE Transactions on Knowledge and Data Engineering*, 28(8): 2086–2100, 2016

¹⁴ Jiankai Sun, Deepak Ajwani, Patrick K Nicholson, Alessandra Sala, and Srinivasan Parthasarathy. Breaking cycles in noisy hierarchies. In *Proceedings of the 2017 ACM on Web Science Conference*, pages 151–160. ACM, 2017

Figure 33.7: The additional step to go from cycle-based hierarchy to arborescence. (a) I highlight in blue the two edges going “against the flow”. (b) The final result, reduced from Figure 33.4(a): an arborescence forest.

Note from Figure 33.7 that, technically speaking, this technique reduces an arbitrary directed network into an arborescence forest, not an arborescence. This is another difference with the cycle-based method, as condensing a graph will never break it into multiple

weakly connected components. Arborescence is a very punitive measure, much more than cycle-based flow hierarchy, but less so than GRC.

33.5 Agony

In the agony measure we start from the assumption that we can partially order nodes into levels. The CEO lives at the top of the hierarchy (level 1), its immediate executive are at level 2, the managers beneath them are at level 3, and so on. Let's say that l_v tells us the level of node v . In this scenario, a perfect hierarchy only has edges going from a node in a lower (more important) level to a node in a higher level. If $l_u < l_v$ then a $u \rightarrow v$ edge is ordinary and expected. On the other hand, a $u \leftarrow v$ edge will cause "agony": something isn't as it is supposed to be.

How much agony does it cause? Well, this is proportional to the level difference between the nodes. You won't be shocked if the CEO accepts orders from another top executive, but you'll go to the madhouse if she does what the most recently hired intern says. In the original paper¹⁵, the authors define the agony of the $u \leftarrow v$ edge as: $l_v - l_u + 1$. The $+1$ is necessary because, if we were to exclude it, we could put all nodes in the same level and obtain zero agony, which would defeat the purpose of the measure.

Ultimately, this reduces to calculating the result of:

$$A(G, l) = \sum_{(u,v) \in E} \max(l_v - l_u + 1, 0).$$

Every time $l_u < l_v$, we contribute zero to the sum. Note that agony requires you to specify the l_u value for all nodes in the network. This is not usually something you know beforehand. So the problem is to find the l_u values that will minimize the agony measure. There are efficient algorithms to estimate the agony of a directed graph¹⁶.

Consider Figure 33.8. In both cases, we have only one edge going against the flow. Agony, however, ranks these two structures differently. In Figure 33.8(a), the difference in rank is only of one, thus the total agony is 2. In Figure 33.8(b), the difference in rank is 3, resulting in a higher agony. Also a cycle-based flow hierarchy measure takes different values, as Figure 33.8(b) involves more edges in a cycle (four edges, versus just two in Figure 33.8(a)).

Ultimately, the resting assumption of agony is the same of the cycle-based flow hierarchy. Agony considers any directed acyclic graph as a perfect hierarchy. Thus it will give perfect scores to the imperfect hierarchies from Figure 33.5.

¹⁵ Mangesh Gupte, Pravin Shankar, Jing Li, Shanmuganayagam Muthukrishnan, and Liviu Iftode. Finding hierarchy in directed online social networks. In *Proceedings of the 20th international conference on World wide web*, pages 557–566. ACM, 2011

¹⁶ Nikolaj Tatti. Hierarchies in directed networks. In *2015 IEEE international conference on data mining*, pages 991–996. IEEE, 2015

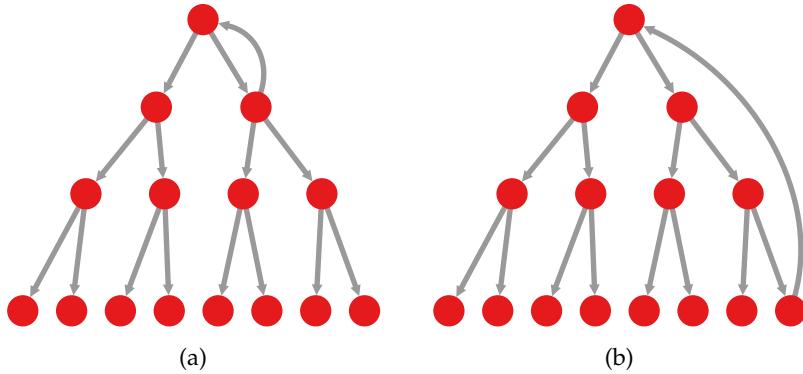


Figure 33.8: Two hierarchies with different values of agony. The vertical positioning of each node determines its level, from top ($l_u = 1$) to bottom ($l_u = 4$).

33.6 Drawing Hierarchies

To wrap up this chapter, note that all these methods have a various degree of graphical flavor to them. Meaning that you can use them to create a picture of your hierarchy, which might help you to navigate the structure. The most rudimentary method is the cycle-based flow hierarchy, because it just reduces the graph to a DAG, which doesn't help you much.

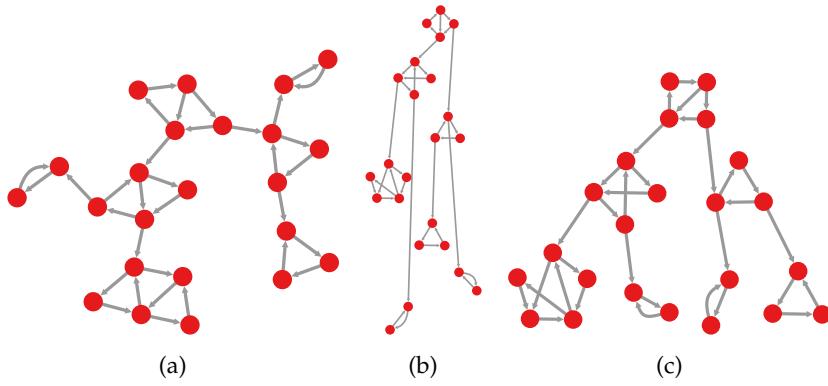


Figure 33.9: (a) A directed graph. (b) The graph from (a), layout according to local reach centrality – with the most central nodes on top and the least central on the bottom. (c) The graph from (a) layered according to its arborescence scheme or its agony levels – the two are equivalent.

Global reach centrality is better, as you can place nodes on a vertical level according to their local reach centrality value. In Figure 33.9(b) I apply a reach centrality informed layout to the directed graph from Figure 33.9(a). The reach centrality layout is quite rudimentary, as the resulting picture looks more akin to an order hierarchy than a flow hierarchy. In the figure, I had to do a bit of manual work to make it look more like a flow hierarchy, which you might not be able to do for larger graphs. Since the method allows you to find flow hierarchies, this mismatch could be confusing. However, at least it allows you to find out the root of the hierarchy, the node(s) with the highest reach centrality, which the previous method could not do.

The arborescence approach is a further step up. Since it reduces the network to an arborescence, one can draw the resulting condensed graph, identifying not only the root of the hierarchy, but at which level each node lies. The same can be said for agony: it assigns each node to a level, thus you can plot the network by layering nodes vertically according to their assigned rank. Figure 33.9(c) shows a possible layout informed by arborescence (or agony).

33.7 Summary

1. There are many different ways to intend the meaning of “hierarchy” in complex networks. Order hierarchy is like centrality: sorting nodes according to their importance. Nested hierarchy is like communities: grouping nodes in teams and teams of teams.
2. Here we look at flow hierarchy: a structural organization where we have nodes working at different levels and information always flows in one direction, from nodes in a higher level to nodes in the directly lower level. We assume networks are directed.
3. There are many ways to estimate the hierarchicalness of a network. A perfect hierarchy cannot have cycles, which are nodes at a lower level linking against the flow to higher levels. One can simply remove cycles, or calculate how much “agony” a connection brings to the structure.
4. We can identify the head of the hierarchy as the node with the highest reach. Alternatively, arborescences are perfect hierarchies – directed acyclic graphs with all nodes having in-degree of one, except the head of the hierarchy having in-degree of zero.

33.8 Exercises

1. Calculate the flow hierarchy of the network at <http://www.networkatlas.eu/exercises/33/1/data.txt>. Generate 25 versions of the network with the same degree distributions of the observed one (use the directed configuration model) and calculate how many standard deviations the observed value is above or below the average value you obtain from the null model.
2. Calculate the global reach centrality of the network at <http://www.networkatlas.eu/exercises/33/1/data.txt> (note: it’s much better to calculate all shortest paths beforehand and cache the result to calculate all local reaching centralities). Is there a single head of the hierarchy or multiple? How many?

3. The arborescence algorithm is simple: condense the graph to remove the strongly connected components and then remove random incoming edges from all nodes remaining with in-degree larger than one, until all nodes have in-degree of one or zero. Implement the algorithm and calculate the arborescence score.
4. Perform the null model test you did for exercise 1 also for global reach centrality and arborescence. Which method is farther from the average expected hierarchy value?

34

High-Order Dynamics

Networks are a great tool to represent a complex system in a simple and elegant way. They embed most of their subtleties into a coherent structure. However, if you only look at the structure you might miss part of the story. This is especially true if you're interested in knowing how different agents act in it.

For instance, consider air travel. The airlines give you the structure, by deciding from where their planes take off and to where they land. And, if you're interested only in studying how different airports connect together, that is all you need to know. But you might instead want to study the behavior of the passengers taking those flights. In this case, the structure itself might be misleading. If you ever made some air travel, you know that, in most cases and especially for long trips, you would take connecting flights. Meaning that you will hop through one or more airports before you reach your intended destination from your origin.

But, if all you look at is the structure of flights, you're not going to be able to recover such information. When a traveler boards in u and jumps off in v , if all you have is the structure, you only know that they are going to either stay in v or go to one of v 's neighbors, maybe even back to u . If you want to really model the traveler's behavior, you need some sort of *memory*, you need to know they arrived from u into v . The next step is not dependent exclusively on the fact we're in v .

Figure 34.1 shows an example of this. The network might have equal weights on the edges, because from the central airport there is an equal number of flights or passengers in each link. Thus, all we can say is that all steps from the central node are equally likely (left). However, from observed data, we might see that some second steps – from our given origin – are much more likely than others (right). In this case, we cannot trust the unweighted edges at face value: we need to take this additional information into account.

When you start talking about memory, you're talking about higher

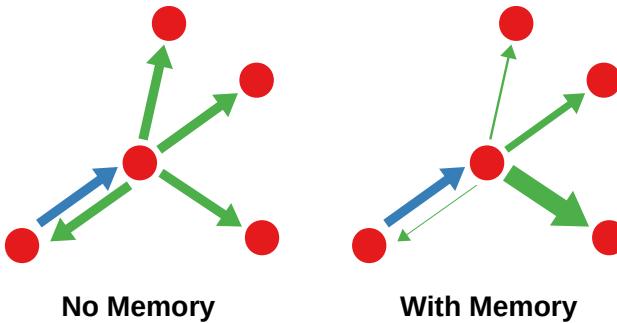


Figure 34.1: An example of possible high order dynamics. The blue arrow shows the first step, while the green arrows show the potential second steps.

order dynamics – for a refresher on the terminology, see Chapter 2. In this chapter, we'll explore three ways of embedding high-order interactions in your network. The first two are structural, the latter is algorithmic. The first is by using simplicial complexes. These don't have memory, but have a way to encode many-to-many relationships, which can handle a large variety of high order interactions. Then we're moving to explicitly encoding memory into your network analysis workflow. You can either modify your network data, embedding the higher order dynamics into the structure; or you can modify your algorithm.

34.1 High Order with Simplicial Complexes

I defined the terminology and some basic facts about simplicial complexes in Section 7.3, and showed how to generate one in Section 18.1. We're focusing on complexes here even though, technically, some of the things you can do with simplicial complexes you can also do with hypergraphs. But simplicial complexes are more flexible and completely contain hypergraphs.

Simple High Order Statistics

There are a bunch of simple analyses you can do with simplicial complexes. The first is generalizing the degree. In a network without simplices, the degree is only a property of a node. But in a simplicial complex, each face has a generalized degree¹. With $k_{d,m}$ we can indicate the number of d dimensional simplices incident on an m -face – with $m < d$. Consider Figure 34.2. The $k_{2,0}$ of node 4 is four. That's because we're looking at $d = 2$, i.e. 2-simplices – also known as triangles –, on an $m = 0$ face – which is a node. Node 4 gets three 2-simplices from its 3-simplex connecting it to nodes 6, 7, and 8, and another 2-simplex with nodes 2 and 3. On the other hand, the $k_{2,1}$ of edge (4,8) is two. The 1-simplex – i.e. edge – (4,8) participates in two 2-simplices – i.e. triangles – with nodes 6 and 7.

¹ Ginestra Bianconi and Christoph Rahmede. Network geometry with flavor: from complexity to quantum geometry. *Physical Review E*, 93(3):032315, 2016

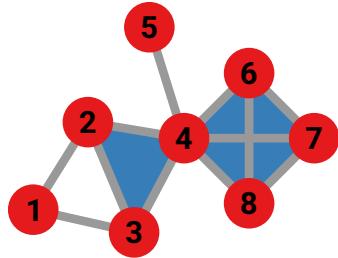


Figure 34.2: An example of simplicial complex. The blue shades represent the two simplices in the complex.

There is a special generalized degree that we call “incidence”. This is $k_{d,d-1} - 1$, i.e. the number of d dimensional simplices incident on a face that is one dimensional step below them, so a $d - 1$ face. Incidence is important to define manifolds, but before we do that we also need to expand the concept of connected component in a simplicial complex.

A d -connected component is the set of adjacent facets of at least dimension d . Two facets of dimension d are connected if there is a sequence of simplices of at least dimension d that allows you to go from one to the other. A 0-connected component is the same as the notion of connected component in a normal network, since 0-simplices are vertices and thus any edge connecting two 0-simplices will make them part of the same 0-connected component. The 1-connected component is a little more tricky. Consider Figure 34.3.

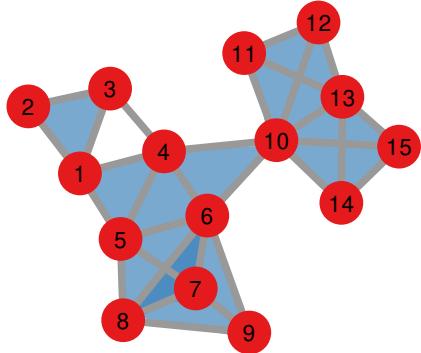


Figure 34.3: A simplicial complex. The blue shades represent the two simplices in the complex. A darker shade indicates simplices that overlap.

The simplicial complex has three 1-components: $\{1, 4, 5, 6, 7, 8, 9, 10\}$, $\{10, 11, 12, 13, 14, 15\}$, and $\{1, 2, 3\}$. That is because nodes 1, 3, and 4 make up a triangle but not a simplex, so there is no simplex of dimension at least 1 connecting those 1-simplices together – there are only 0-simplices, the nodes themselves. The complex only has a 2-component: $\{5, 6, 7, 8, 9\}$. There are other facets of dimension 2, but they are only connected by a 1-simplex, so they do not make up a 2-component.

Now we can define a manifold. A manifold is a simplicial complex that is d connected and all its d faces have incidence of either 0 – i.e.

they are not connected – or 1 – i.e. they only have a single incident face. Figure 34.4(a) shows a case that is not a manifold, because there is a face with incidence larger than one, while Figure 34.4(b) shows a manifold.

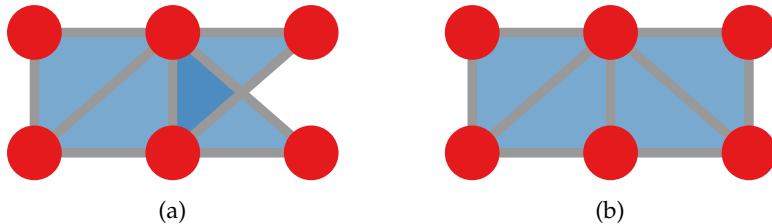


Figure 34.4: Simplicial complexes with simplices shaded in blue. (a) Not a manifold. (b) A manifold.

High Order Processes on Simplicial Complexes

The reason why you want to work with simplicial complexes is not because of slightly fancier network measures: it is because you want to study processes where these high order interactions can make a difference. Of course, there are too many to report on them all, so I picked three that can hopefully give you a taste of the possibilities in front of you. We’re going to see briefly the problems of **synchronization**, **percolation**, and **epidemics**.

Networks have been used to study **synchronization** for a long time. One classical example is thinking of nodes as neurons who are firing at random intervals. Each node fires with a frequency that is drawn uniformly at random. However, nodes that are connected will influence each other. At each pulse, their frequencies will get closer and closer together – a process regulated by a synchronization speed parameter σ . Figure 34.5 shows a simple example, where nodes get synchronized over time.

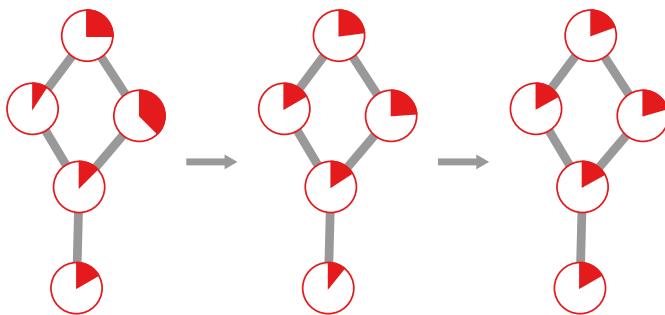


Figure 34.5: Synchronization on a network. The shaded portion of each node is proportional to its firing frequency. From left to right we progress over time, with nodes changing their frequencies according to the ones of their neighbors.

This process has a phase transition. If σ is too low, no or very few nodes will synchronize. Beyond a critical value of σ , the fraction of nodes that is firing at the same rate will start to increase abruptly, until it reaches almost the entirety of the network. Figure 34.6(a)

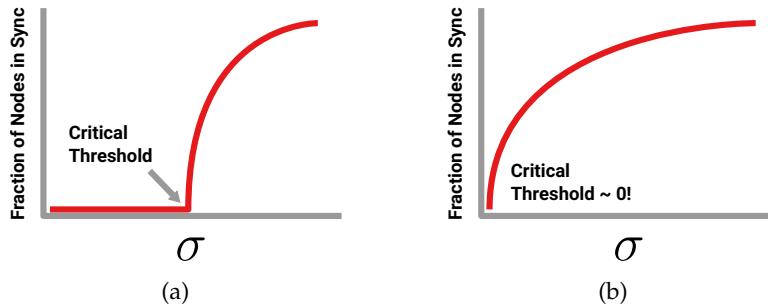


Figure 34.6: Different synchronization dynamics: fraction of nodes at the same frequency (y axis) at a given σ value (x axis). (a) No many-to-many interactions. (b) With many-to-many interactions.

shows how the fraction of synchronized nodes evolves for increasing values of σ .

When you run this model on a simplicial complex, you enable many-to-many interactions: now nodes in a simplex will all influence each other at the same time. There are many ways to define such a model^{2,3,4}, but one key result is that the critical σ threshold tends to go to zero – as Figure 34.6(b) shows. In practice, when you enable many-to-many interactions, you will always see at least some synchronization in the system, even for small synchronization strength σ , which you would not see it if many-to-many interactions were to be disabled.

With **percolation**, I intend the catastrophic network failures I discussed in Chapter 22. In that case, we saw that nodes or edges could fail, and their failures could propagate (= percolate) through the network, causing a chain reaction. In simplicial complexes, you can have simplices failing as well. The classical case of node or edge failure is a $d = 1$ type of failure. But you can have also simplices failing at $d = 2$, which means you have a certain probability triangles will fail⁵. And it goes without saying that you can experience failures at higher dimensions d .

Finally, in **epidemics** you can expand the SI and related models I explained to you in Chapters 20 and 21. In the basic model, you have a certain probability of transitioning β if you have a contact with an infected individual. When you have simplices, you can have a different transition probability if you are involved in a simplex with an infected individual. This models well the case when you enter a room and you don't particularly interact with anyone, but you're in the same space with someone infected. That's a many-to-many interaction, given that there could be multiple (potentially infected) people in the room.

So now you have, say, a β_1 for the normal face-to-face interactions – which are a $d = 1$ simplex –, a β_2 for a many-to-many interaction with two people – because that's a $d = 2$ simplex –, and so on^{6,7}... What we normally do when we ignore many-to-many interactions is

² Per Sebastian Skardal and Alex Arenas. Abrupt desynchronization and extensive multistability in globally coupled oscillator simplexes. *Physical review letters*, 122(24):248301, 2019

³ Ana P Millán, Joaquín J Torres, and Ginestra Bianconi. Explosive higher-order kuramoto dynamics on simplicial complexes. *Physical Review Letters*, 124(21):218301, 2020

⁴ Reza Ghorbanchian, Juan G Restrepo, Joaquín J Torres, and Ginestra Bianconi. Higher-order simplicial synchronization of coupled topological signals. *Communications Physics*, 4(1):120, 2021

⁵ Ginestra Bianconi and Robert M Ziff. Topological percolation on hyperbolic simplicial complexes. *Physical Review E*, 98(5):052308, 2018

⁶ Nicholas W Landry and Juan G Restrepo. The effect of heterogeneity on hypergraph contagion models. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 30(10), 2020

⁷ Iacopo Iacopini, Giovanni Petri, Alain Barrat, and Vito Latora. Simplicial models of social contagion. *Nature communications*, 10(1):2485, 2019

that we only study the fraction of infected individuals for different values of β_1 . This basically means we assume $\beta_2 = 0$, and we get a given function telling us how much we're screwed given how infective at the face-to-face level a disease is.

However, Figure 34.7 shows you that, if $\beta_2 > 0$, then the contagion will happen *faster* and will infect *more* nodes than we would expect for the *same value* of β_1 .

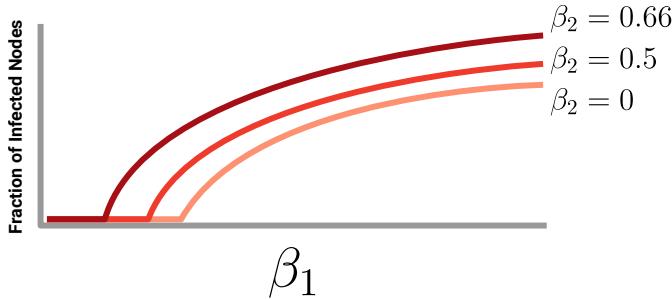


Figure 34.7: The fraction of infected nodes (y axis) for a given value of β_1 (x axis) and β_2 (line color).

Generating High Order Networks

Another reason to love manifolds – besides enabling the interesting high order dynamics we just saw – lies in their generative power. For the rest of the section we assume a manifold with $d = 2$. In any d -manifold, you can classify links as either saturated or unsaturated. A link is saturated if it is part of exactly d simplices. If not, it is unsaturated. Once you classify the links you have, you can grow the manifold in two ways. First, you can pick a single unsaturated link with probability $1/|E_u|$, with E_u being the set of saturated links, and saturate it by gluing a triangle to it – which implies that you need to also add a node with which to close the triangle. Second, with probability p – which is a parameter you can choose – you can find two unsaturated links and add the simplex to saturate them – this does not add a node. Figure 34.8 shows how these two moves look like.

If you do this, you can grow a manifold⁸. The reason why this is cool is because, with properly defined parameters d and p , you can reproduce a lot of real world properties in your manifold – which was the holy grail of Chapter 17. For some specific d and p values – for Figure 34.9 I set $p = 0.5$ and $d = 2$ – you obtain a small world network, with a broad degree distribution, high clustering, and high modularity as well. This hints at the fact that these sort of high order interactions might play a role in the formation of many real world networks.

The cool thing about generating manifolds this way is that, with a slightly more advanced model, you can have a single network

⁸ Zhihao Wu, Giulia Menichetti, Christoph Rahmede, and Ginestra Bianconi. Emergent complex network geometry. *Scientific reports*, 5(1):10073, 2015

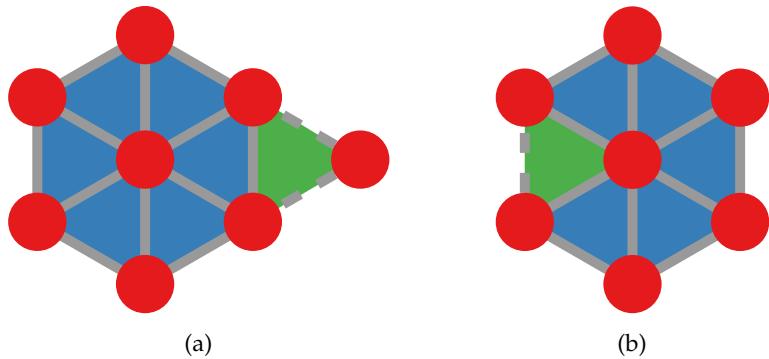


Figure 34.8: Two rules to grow a manifold. The blue shade is for the simplices already in the manifold, the added simplex is in green. (a) Adding a simplex to an unsaturated link. (b) Saturating two links.

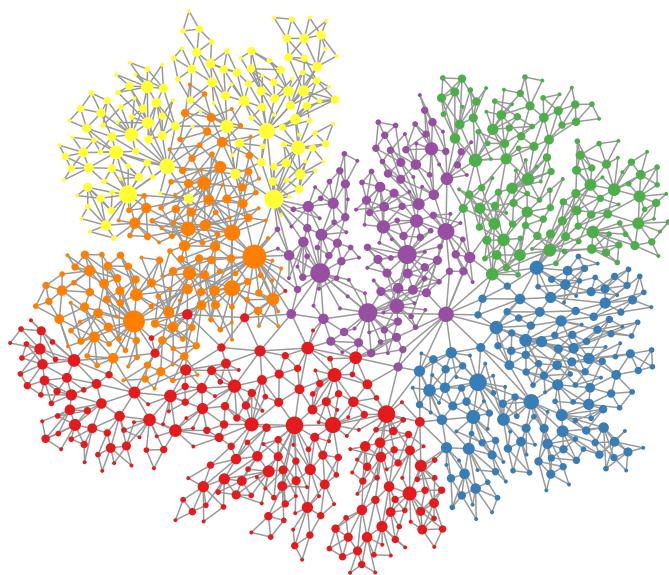


Figure 34.9: A manifold grown with the rules from Figure 34.8. The node color is its community. The node size is proportional to its degree.

generating process that generates a wide variety of other models. With suitable choices of d and p , you can have $G_{n,p}$ model, or a preferential attachment one, and you get clustering and communities for free.

Just like edges, simplices can also have weights, and these weights can be used meaningfully to enhance the network generating process⁹. Even more interestingly, we can assign an energy value to each node – an arbitrary non-negative number. The nodes pass their energy to the face they belong to. This energy acts like a weight but, since it comes from the node, will lead to different dynamics when growing the network – again regulated by a parameter of your choice. Depending on the values of this parameter, your manifold's properties will approach the ones of different quantum states: Fermi-Dirac, Boltzmann, or Bose-Einstein – which, admittedly, sounds unbelievably cool, but I will stop here because my quantum physics isn't exactly up to the standards required to discuss this. The excellent

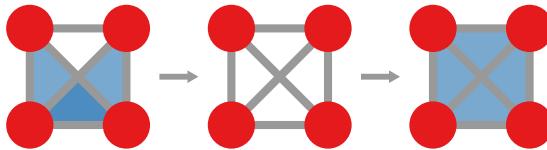
⁹ Owen T Courtney and Ginestra Bianconi. Weighted growing simplicial complexes. *Physical Review E*, 95(6):062301, 2017

book by Bianconi¹⁰ is what you should use to quench your knowledge thirst.

Additionally, there are models that can generate multilayer simplicial complexes¹¹.

Suppose that you find simplices cool, but you already have your network – so generating a manifold is not an option. However, your network is not a simplicial complex. What do you do? Technically, you can make any arbitrary network into a simplicial complex. You can consider every clique in the network as a simplex and perform the analyses I described on that structure.

You should be careful about one thing, though. As I told you in Section 7.3, you can do the opposite operation: every simplicial complex can be reduced to a normal complex network by ignoring the simplices and treating them like cliques. However, these two operations – simplices to cliques and cliques to simplices – are *not commutative*. If you apply them one after the other, you’re not going to go back to your original simplicial complex – bar weird coincidences. Figure 34.10 shows you a case in which a clique of four nodes was actually made up by two 2-simplices rather than being a 3-simplex, and therefore the reconstruction by cliques leads to a different simplicial complex.



¹⁰ Ginestra Bianconi. *Higher-order networks*. Cambridge University Press, 2021

¹¹ Hanlin Sun and Ginestra Bianconi. Higher-order percolation processes on multiplex hypergraphs. *Physical Review E*, 104(3):034306, 2021

34.2 Embedding Memory into the Structure

A natural way to have higher order memory in your network analysis is by embedding it into the structure itself. This is a powerful approach, because it allows you to use any non-high-order algorithm you want. You have the entirety of the network analysis toolbox at your disposal. The price you have to pay is that you need to keep track of your operation. You need to reconstruct the original structure if you want to properly interpret your results.

This practically boils down to performing a pre-processing on your data structure and a post-processing on your results. Here we focus mainly on the pre-processing as, hopefully, how to post-process the results should be straightforward. Unfortunately, the pre-process is not going to be as simple as I make it to be in Figure 34.1. You cannot simply re-weight your edges, because the re-weighting would be dependent on the current position of the agent in the network. Thus

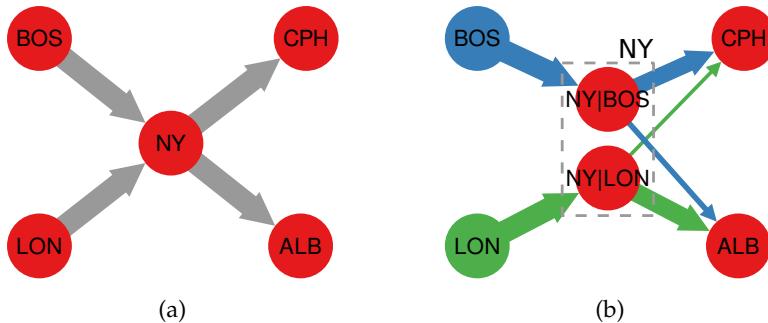
Figure 34.10: From left to right: a simplicial complex with two 2-simplices, its skeleton, the simplicial complex reconstructed by assuming each clique in a network is a simplex.

you'd have to have a re-weighting for every node in the network. Worse still, if you do that you're just implementing second-order dynamics: if you need to go to a higher order than that (say, you need to remember the last two nodes through which you passed) you're in no better position than before.

I'm going to specifically focus on a few papers in this line of research, but hopefully you could see how the general approach in this category of high-order analysis works.

High Order Network

What we want to build here is a High Order Network¹² (HON). Let's go back to our airline travel example. In the original network, nodes are airports and edges represent flows of passengers between them. Now, the crucial assumption we're making here is that, if I'm landing in New York in a plane coming from Boston, this is fundamentally a different travel than the one which makes me land in the same airport, but from London. Thus, in the HON representation, we split the New York node in two meta nodes. One of the two meta nodes captures the passenger flow from Boston, the other from London. Figure 34.11 shows an example of this procedure.



¹² Jian Xu, Thanuka L Wickramarathne, and Nitesh V Chawla. Representing higher-order dependencies in networks. *Science advances*, 2(5):e1600028, 2016

Figure 34.11: (a) A simple directed network of airplane travel. (b) Its corresponding High Order version, introducing conditional nodes depending on the origin of the flow.

What happens here is that now we have a way to represent different flows. A passenger from London might be much more likely to want to go to Alberta, while the Bostonian would instead go to Copenhagen. Thus we can re-part the outgoing edge weight of New York into those two meta nodes, to make them a more accurate representation of the data.

Of course, this represents only a single step in the creation of the HON structure. The origin nodes that brought us to New York were themselves the product of another high order transition. Thus they are also split into several meta nodes. The general HON structure looks like the one in Figure 34.12.

You might have noticed that we use the conditional probability notation introduced in Chapter 2. Each node represents the transition

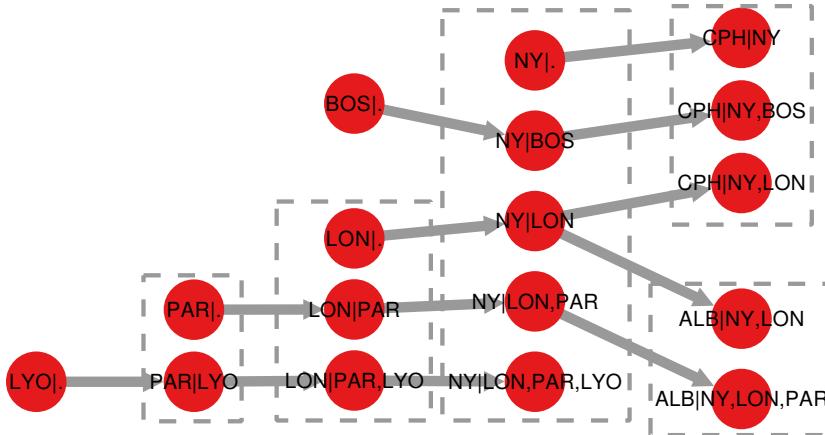


Figure 34.12: A small example of a full High Order Network with complex dependencies up to an order of four. The dashed outlines group all the meta nodes originating from the same original node.

probability of getting into node v given that we come from node u – and, possibly, given that we reached u from another node, and so on. While airline travel might have short dependency chains – rarely a trip involves more than two transfers – other networks show dependency chains up to length five: the global shipping network, for instance. When I label a node v as, for instance, $v|.$, it means that this specific node has no dependencies: it represents passengers who are starting their first step in v .

If you think HON structures look like a big and complex Bayesian network – see Section 6.4 – don’t worry: you’re not alone.

There are potential downsides of using a HON representations. First, as you can see from Figure 34.12, the structure can become really unwieldy. The original network only had 7 nodes and 6 edges, and ballooned into having 17 nodes and 16 edges. Therefore you need to find the right trade off between the complexity of your HON representation and the analytic gains it gives you. In the original paper, for instance, the authors limit themselves to an order of five, even if the shipping network they analyze might have even longer dependencies. The increase in complexity simply wasn’t worth it.

Finally, HON networks tend to transform into weakly connected graphs, or even not connected. The original network from Figure 34.12 had a single connected component, while its HON representation splits into 5 connected components. If your algorithm cannot handle multiple connected components, you might be in trouble.

Memory Network

Alternatives to HON exist. For instance, researchers built what they call a “memory network¹³”. To model second-order dynamics we can create a line graph. If you recall Section 6.1, a line graph is a

¹³ Martin Rosvall, Alcides V Esquivel, Andrea Lancichinetti, Jevin D West, and Renaud Lambiotte. Memory in network flows and its effects on spreading dynamics and community detection. *Nature communications*, 5:4630, 2014

transformation of the original graph. The edges of the original graph become the nodes of the network and they are connected to each other if the original edges shared a node in the network. Figure 34.13 is a reproduction of Figure 6.4 to remind you of the building procedure. Line graphs will pop up also in overlapping community discovery (Section 38.5).

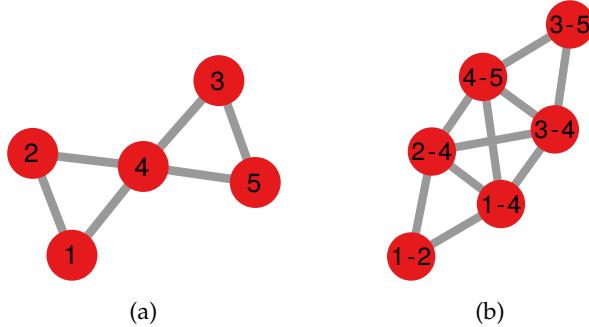


Figure 34.13: (a) A graph. (b) Its linegraph version.

You can model third order dynamics by making a more complicated version of a line graph, in which nodes stand in for paths of length three. You can see how memory graphs are a generalization of line graphs, and they start looking extremely similar to the HON model. In fact, once you have built your memory network with the desired order, then the simple memoryless Markov processes on the memory network describe the high-order processes, of the order you used to build the network in the first place. The adjacency matrix of a memory network of second order is a non-backtracking matrix (Section 11.2), provided that the memory network has no self-loops. Non-backtracking random walks are another example of high order network analysis.

The difference between memory networks and HONs is that HONs are a bit more flexible, because they allow you to have nodes of any order mixed together in the same structure. In the memory network, all nodes represent transitions of the same order.

34.3 Embedding Memory into the Algorithm

The alternative approach to deal with high order dependencies is to leave your structure alone and to embed the high order logic directly into your algorithm. In practice, you “hide” both the pre- and post-process from the previous section into your analysis. This way, you don’t have to deal with the complexity yourself.

There is a bit more diversity in this category of solutions, due to the many different valid ways one could incorporate high orders into network analysis.

Motif Dictionary

The first approach I consider is the one building motif dictionaries. In this approach, one realizes that there are different motifs of interest that have an impact on the analysis. For instance, one could focus specifically on triangles. Once you specify all the motifs you're interested in, you take a traditional network measure and you extend it to consider these motifs.

This move isn't particularly difficult once you realize that low-order network measures still work with the same logic. It's just that they exclusively focus on a single motif of the network: the edge. An edge is a network motif containing two nodes and a connection between them. Once you realize that a triangle is nothing else but a motif with three nodes and three edges connecting them, then you're in business.

To make this a bit less abstract, let's consider a specific instance of this approach^{14,15,16}. In the paper, the idea is to use motifs to inform community discovery, the topic of Part X. I'm going to delve deeper into the topic in that book part, for now let's just say that we're interested in solving the 2-cut problem (Section 11.5): we want to divide nodes in two groups such that we minimize the number of edges connecting nodes in different groups.

In the classical problem we want to make a normalized cut such that the number of edges flowing from one group to the other is minimum, considering some normalization factor. So we have a fraction that looks something like this:

$$\phi(S) = \frac{|E_{S,\bar{S}}|}{\min(|E_S|, |E_{\bar{S}}|)},$$

where S is one of the two groups – i.e. a set of nodes on one side of the cut –, \bar{S} is its complement – i.e. $\bar{S} = V - S$, the set of nodes on the other side of the cut. E_x is the set of edges in the set x , and $E_{x,y}$ is the set of edges established between a node in x and a node in y . In practice, let's find S such that $\phi(S)$ is minimum. We do so by minimizing the number of edges between S and its complement, normalized by their sizes (so that we don't find trivial solutions by cutting off simply a dangling leaf node).

But here we say: *No! Not the number of edges! We are interested in higher order structures! We want to minimize the number of triangles between groups!* How would that look like? Exactly the same. We just count not the number of the edges, but the number of arbitrary motifs M spanning between S and non- S :

$$\phi(S, M) = \frac{|M_{S,\bar{S}}|}{\min(|M_S|, |M_{\bar{S}}|)}.$$

¹⁴ Austin R Benson, David F Gleich, and Jure Leskovec. Higher-order organization of complex networks. *Science*, 353(6295):163–166, 2016

¹⁵ Hao Yin, Austin R Benson, Jure Leskovec, and David F Gleich. Local higher-order graph clustering. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 555–564. ACM, 2017

¹⁶ Charalampos E Tsourakakis, Jakub Pachocki, and Michael Mitzenmacher. Scalable motif-aware graph clustering. In *Proceedings of the 26th International Conference on World Wide Web*, pages 1451–1460, 2017

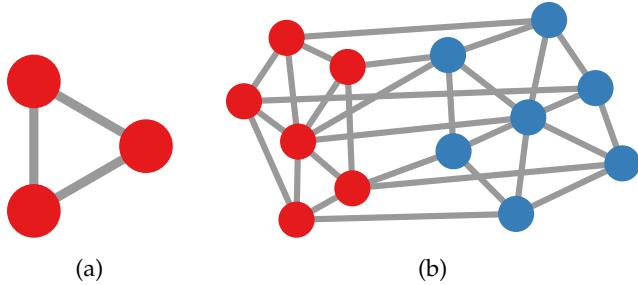


Figure 34.14: (a) The motif we want to minimize the cut for. (b) A high order normalized cut solution, which I represent via the node's color.

Boom. By finding an S minimizing this specific $\phi(S, M)$ we just made a high-order normalized cut. This can be done exactly like finding the “normal” normalized cut, by examining the eigenvectors of a specially constructed Laplacian¹⁷. Figure 34.14 shows an example. In the figure, the two groups still have tons of edges going from one group to another, this is hardly a solution for a regular normalized cut. But there are few triangles between S and \bar{S} , thus this is a proper solution for the higher order normalized cut. The shape of this solution can be applied to many other problems.

Processes with Memory

In this category of solutions you find all the approaches who take the equation of the Markov process and add a temporal factor. For instance, suppose that we’re observing random walks (Chapter 11). We have a vector of probability p_k that tells us the status of the random walkers at time k , namely the probability of the walkers to be in a node (p_k is a vector of length $|V|$). Now, if we were doing a normal random walk and perform one step, we’d know that the next step is simply given by the stochastic adjacency matrix, i.e. $p_{k+1} = p_k D^{-1} A$ (assuming A is the normal adjacency matrix, and D the degree diagonal matrix). This is a *discrete* random walk on a graph.

But what if we want higher order dynamics? What if we want to look at the result of the process after t steps? Well, the idea is to add the temporal information to the equation: $p_{k+t} = p_k T_t$. So what’s T_t ? T_t is a matrix telling us the transition probability from u to v after t steps. If you followed the linear algebra math in Chapter 11 you know that, to perform a random walk of length t , you can simply raise $D^{-1}A$ to the power of t : $(D^{-1}A)^t$.

The limitation here is that time ticks discretely, i.e. the random walker makes one move per timestep. In many cases, you might want to simulate the passage of time as a continuous flow¹⁸. If you want to do it, you need to derive a different T_t . First, rewrite $D^{-1}A$ as $-D^{-1}L$. This changes the random walk from discrete to continuous¹⁹. This allows us to let time flow and take the result of the random

¹⁷ Austin R Benson, David F Gleich, and Jure Leskovec. Tensor spectral clustering for partitioning higher-order network structures. In *Proceedings of the 2015 SIAM International Conference on Data Mining*, pages 118–126. SIAM, 2015

¹⁸ Michael T Schaub, Renaud Lambiotte, and Mauricio Barahona. Encoding dynamics for multiscale community detection: Markov time sweeping for the map equation. *Physical Review E*, 86(2):026112, 2012b

¹⁹ Renaud Lambiotte, J-C Delvenne, and Mauricio Barahona. Laplacian dynamics and multiscale modular structure in networks. *arXiv preprint arXiv:0812.1770*, 2008

walk at time t : $T_t = e^{-tD^{-1}L}$. If we set $t = 1$, we recover exactly the transition probabilities of a one-step random walk. But now we're free to change t at will, to get second, third, and any higher order Markov processes.

Other Approaches

There is a whole bunch of other approaches to introduce high order dynamics into your network structure. More than I can competently cover. I provide here some examples with brief, but overly simplistic, explanations.

One way is to use tensors (Section 5.3). In practice, you create a multidimensional representation of the topological features of the network. Each added dimension of the tensor represents an extra order of relationships between the nodes^{20,21}. Then, by operating on this tensor, you can solve any high order problem: in the papers I cite the problem the authors focus on is graph matching.

Another general category of solutions is a collection of techniques to take high order relation data and transform it into an equivalent first order representation^{22,23}. The first order solution in this structure then translates into the high order one, much like in the HON and memory network approach. The cited techniques are also generally applied to the problem of finding a high order cut in the network.

Being able to study high order interactions can help you making sense of many complex systems. For instance, they have been used to explain the remarkable stability of biodiversity in complex ecological systems²⁴. Other application examples include the study of infrastructure networks²⁵ – how to track the high order flow of cars in a road graph –, and aiding in solving the problem of controlling complex systems²⁶ – which we introduced in Section 21.4.

34.4 Summary

1. Classical network analysis is single-order: only the direct connections matters. But many phenomena they represent are high-order: team collaborations are many-to-many relationships or, in a flight passenger network, your next step in the trip is influenced by all the steps you took in the recent past.
2. We can use simplicial complexes to model how many-to-many relationships make synchronization easier, create new potential failure types in infrastructures, and make epidemics more infective than we would normally expect.

²⁰ Michael Chertok and Yosi Keller. Efficient high order matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(12):2205–2215, 2010

²¹ Olivier Duchenne, Francis Bach, In-So Kweon, and Jean Ponce. A tensor-based algorithm for high-order graph matching. *IEEE transactions on pattern analysis and machine intelligence*, 33(12):2383–2395, 2011

²² Hiroshi Ishikawa. Higher-order clique reduction in binary graph cut. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2993–3000. IEEE, 2009

²³ Alexander Fix, Arman Gruber, Endre Boros, and Ramin Zabih. A graph cut algorithm for higher-order markov random fields. In *2011 International Conference on Computer Vision*, pages 1020–1027. IEEE, 2011

²⁴ Jacopo Grilli, György Barabás, Matthew J Michalska-Smith, and Stefano Allesina. Higher-order interactions stabilize dynamics in competitive network models. *Nature*, 548(7666):210, 2017

²⁵ Jan D Wegner, Javier A Montoya-Zegarra, and Konrad Schindler. A higher-order crf model for road network extraction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1698–1705, 2013

²⁶ Abubakr Muhammad and Magnus Egerstedt. Control using higher order laplacians in network topologies. In *Proc. of 17th International Symposium on Mathematical Theory of Networks and Systems*, pages 1024–1038. Citeseer, 2006

3. Generating networks using simplicial complex dynamics can reproduce many real world properties, hinting at many-to-many interactions being a potential mechanism behind natural interacting processes.
4. There are two approaches to handle high-order processes with memory: you can modify the network structure so that it explicitly codes its memory; or you give to the algorithm the task of remembering previous moves.
5. In High Order Networks you split each node to represent all the paths that lead to it. In memory networks you instead model second order dynamics with a line graph, third order dynamics with the line graph of the line graph, and so on.
6. Embedding memory in the analysis can be done by either building dictionaries of temporal motifs, adding a temporal factor in the classical Markov equation (e.g. for random walks), and other approaches.

34.5 Exercises

1. Calculate the distribution of the $k_{2,0}$ and $k_{2,1}$ degrees of the network at <http://www.networkatlas.eu/exercises/34/1/data.txt>. Assume every clique of the network to be a simplex.
2. Run an SI model with simple contagion on the simplicial complex from the previous exercise. The seed node is node 0. Run it for all possible combinations of β_1 from 0.1 to 0.9 (in 0.1 increments) and for $\beta_2 = \{0.0, 0.25, 0.5\}$. Make 100 independent runs and average the results. Visualize the ratio of infected nodes after 3 steps for each process – like in Figure 34.7.
3. Generate the two-order line graph of the network at <http://www.networkatlas.eu/exercises/34/3/data.txt>, using the average edge betweenness of the edges as the edge weight.
4. Assume that the edge weight is proportional to the probability of following that edge. Which 2-step node transitions became more likely to happen in the line graph compared to the original network? (For simplicity, assume that the probability of going back to the same node in 2-steps is zero for the line graph)

Part X

Communities

35

Graph Partitions

We have reached the part of network analysis that has probably received the most attention since the explosion of network science in the early 90s: community discovery (or detection). To put it bluntly, community discovery is the subfield of network science that postulates that the main mesoscale organization of a network is its partition of nodes into communities. Communities are groups of nodes that are very related to each other. If two nodes are in the same community they are more likely to connect than if they are in two distinct communities. This is an overly simplistic view of the problem, and we will decompose this assumption when the time comes, but we need to start from somewhere.

So the community discovery subfield is ginormous. You might ask: “Why?” Why do we want to find communities? There are many reasons why community discovery is useful. I can give you a couple of them. First, this is the equivalent of performing data clustering in data mining, machine learning, etc. Any reason why you want to do data clustering also applies to community discovery. Maybe you want to find similar nodes which would react similarly to your interventions. Another reason is to condense a complex network into a simpler view, that could be more amenable to manual analysis or human understanding.

More generally, decomposing a big messy network into groups is a useful way to simplify it, making it easier to understand. The reason why there are so many methods to find communities – which, as we’ll see, rarely agree with each other – is because there are innumerable ways to simplify a network.

It is difficult to give you a perspective of how vast this subfield of network science is. Probably, one way to do it is by telling you that there are so many papers proposing a new community discovery algorithm or discussing some specific aspect of the problem, that making a review paper is not sufficient any more. We are in need of making review papers of review papers of community discovery.

This is what I'm going to attempt now.

I think that we can classify review works fundamentally into four categories, depending on what they focus on the most. Review papers on community discovery usually organize community discovery algorithms by:

- **Process.** In this subtype of review paper, the guiding principle is how an algorithm works^{1,2,3,4,5,6,7}. Does it use random walks rather than eigenvector decomposition? Does it use a propagation dynamic or a Bayesian framework? Does it aim at describing the network as is, or at modeling what underlying latent process could have generated what we observe⁸?
- **Performance.** Here, all that matters is how well the algorithm works in some test cases^{9,10,11,12,13}. The typical approach is to find many real world networks, or creating a bunch of synthetic benchmark graphs (usually using the LFR benchmark – see Section 18.2) and rank methods on how well they can maximize a quality function.
- **Definition.** More often than not, the standard community definition I gave you earlier (nodes in the same community connect to each other, nodes in different communities rarely do so) isn't exactly capturing what a researcher wants to find. We'll explore later how this definition fails. Some review works acknowledge this, and classify methods according to their community definition^{14,15,16}. Different processes might be based in different definitions, so there's overlap between this category and the first one I presented, but that is not always the case.
- **Similarity.** Finally, there are review works using a data-driven approach to figure out which algorithms, on a practical level, return very similar communities for the same networks^{17,18,19,20}. This is similar to the performance category, with the difference that we're not interested in what performs *better*, only in what performs *similarly*.

The **process** approach is the most pedagogical one, because it focuses on us trying to understand how each method works. Thus, it will be the one guiding this book part the most. However, I'll pepper around the other approaches as well, when necessary. This book part will necessarily be more superficial than what one of the excellent surveys out there can do in each specific subtopic of community discovery, so you should check them out.

In this chapter I'll limit myself discussing the most classical view of community discovery, the one sheepishly following the classical definition. We're going to take a historical approach, exploring how

¹ Santo Fortunato. Community detection in graphs. *Physics reports*, 486(3-5):75–174, 2010

² Santo Fortunato and Darko Hric. Community detection in networks: A user guide. *Physics Reports*, 659:1–44, 2016

³ Srinivasan Parthasarathy, Yiye Ruan, and Venu Satuluri. Community discovery in social networks: Applications, methods and emerging trends. In *Social network data analytics*, pages 79–113. Springer, 2011

⁴ Mason A Porter, Jukka-Pekka Onnela, and Peter J Mucha. Communities in networks. *Notices of the AMS*, 56(9):1082–1097, 2009

⁵ Mark EJ Newman. Detecting community structure in networks. *The European Physical Journal B*, 38(2):321–330, 2004b

⁶ Leon Danon, Albert Diaz-Guilera, Jordi Duch, and Alex Arenas. Comparing community structure identification. *Journal of Statistical Mechanics: Theory and Experiment*, 2005(09):P09008, 2005

⁷ Natali Gulbahce and Sune Lehmann. The art of community detection. *BioEssays*, 30(10):934–938, 2008

⁸ Tiago P Peixoto. *Descriptive vs. inferential community detection in networks: Pitfalls, myths and half-truths*. Cambridge University Press, 2023

⁹ Jure Leskovec, Kevin J Lang, and Michael Mahoney. Empirical comparison of algorithms for network community detection. In *WWW*, pages 631–640. ACM, 2010b

¹⁰ Zhao Yang, René Algesheimer, and Claudio J Tessone. A comparative analysis of community detection algorithms on artificial networks. *Scientific Reports*, 6:30750, 2016a

¹¹ Steve Harenberg, Gonzalo Bello, L Gjeltema, Stephen Ranshous, Jitendra Harlalka, Ramona Seay, Kanchana Padmanabhan, and Nagiza Samatova. Community detection in large-scale networks: a survey and empirical evaluation. *Wiley Interdisciplinary Reviews: Computational Statistics*, 6(6):426–439, 2014

¹² Günce Keziban Orman and Vincent Labatut. A comparison of community detection algorithms on artificial networks. In *DS*, pages 242–256. Springer, 2009

¹³ Andrea Lancichinetti and Santo Fortunato. Community detection algorithms: a comparative analysis. *Physical review E*, 80(5):056117, 2009

the concept of network communities came to be as we intend it today. Later chapters will complicate the picture. Buckle up, this is going to be a wild ride.

35.1 Stochastic Blockmodels

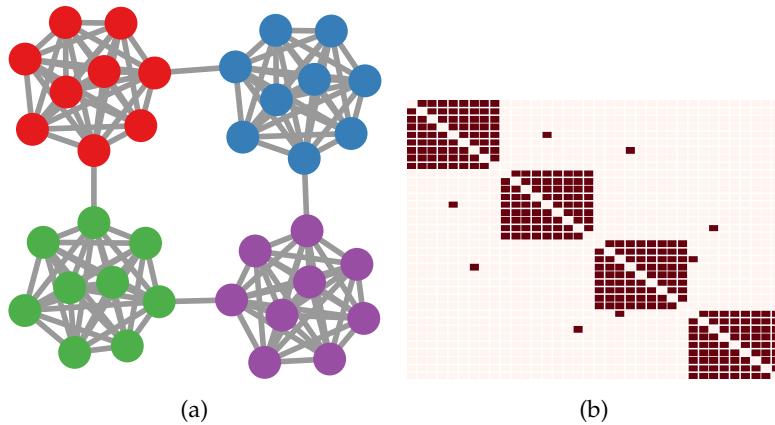
Classical Community Definition

When people started mapping complex systems as networks, they realized that the edges didn't distribute randomly across nodes. We already saw that deviating from random expectation is a source of interest when we talked about degree distributions (Section 9.2). On top of the rich-get-richer effect, researchers realized that edges distributed unevenly among different groups of nodes. Especially, but not only, in social networks, there were lumps of connections, separated by very sparse areas.

The ideal scenario is something resembling Figure 35.1(a). The natural next step was trying to see if we could separate these lumps of connections into coherent groups: communities. This created the first and most commonly accepted definition of a network community:

Communities are groups of nodes densely connected to each other and sparsely connected to nodes outside the community.

I would call this the classical definition of a network community. This definition can be attacked and deconstructed from multiple parts but, for now, let's accept it. Note that, for now, we assume that a node can only be part of a single community. We use the term "partition" to refer to the assignment of nodes to their community.



In the early community discovery days²¹ – before we even had coined the term “community” –, the main approach was using stochastic blockmodels. We already introduced them in Section

¹⁴ Satu Elisa Schaeffer. Graph clustering. *Computer science review*, 1(1):27–64, 2007

¹⁵ Michele Coscia, Fosca Giannotti, and Dino Pedreschi. A classification for community discovery methods in complex networks. *SADM*, 4(5):512–546, 2011

¹⁶ Fragkiskos D Malliaros and Michalis Vazirgiannis. Clustering and community detection in directed networks: A survey. *Physics Reports*, 533(4):95–142, 2013

¹⁷ Nguyen Xuan Vinh, Julien Epps, and James Bailey. Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance. *J. Mach. Learn. Res.*, 11(Oct):2837–2854, 2010

¹⁸ Vinh-Loc Dao, Cécile Bothorel, and Philippe Lenca. Estimating the similarity of community detection methods based on cluster size distribution. In *International Workshop on Complex Networks and their Applications*, pages 183–194. Springer, 2018

¹⁹ Vinh-Loc Dao, Cécile Bothorel, and Philippe Lenca. Community structure: A comparative evaluation of community detection methods. *arXiv preprint arXiv:1812.06598*, 2018

²⁰ Amir Ghasemian, Homa Hosseini-mardi, and Aaron Clauset. Evaluating overfit and underfit in models of network community structure. *TKDE*, 2019

Figure 35.1: (a) A representation of a classical ideal community structure. I encode with the node color the community to which the node belongs. (b) The adjacency matrix of (a).

²¹ Paul W Holland, Kathryn Blackmond Laskey, and Samuel Leinhardt. Stochastic blockmodels: First steps. *Social networks*, 5(2):109–137, 1983

[18.2](#) as a method to generate a synthetic graph. How do we apply them to the problem of finding communities? The first step in our quest is changing the perspective over the graph from Figure [35.1](#). Figure [35.1\(b\)](#) shows you its corresponding adjacency matrix. The diagonal blocks are the ones referred by the name “blockmodel”.

Maximum Likelihood

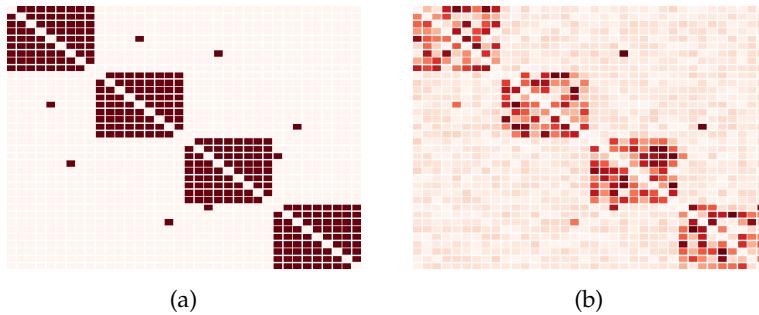


Figure 35.2: (a) An observed adjacency matrix. (b) A possible SBM representation, darker cells indicate edges with higher probability. Finding the most likely (b) explaining (a) is the task of SBM-based community detection.

What we want to do now is to find the SBM model that most accurately reconstructs the adjacency matrix in Figure [35.1\(b\)](#). To do so, we will need to plant a community structure in the SBM model. If the resulting network is similar to the original one, it is very likely that the partition we planted corresponds to the “real” partition in the original graph. As the vignette in Figure [35.2](#) shows, we do so by employing the principle of maximum likelihood^{[22,23](#)} – for details about the likelihood function, see Section [4.3](#). I’m going to give you a very incomplete and simplified view of what that means, prompting you to read more if you’re interested in the topic.

Maximum likelihood estimation means to estimate the parameters of a model that are more likely to generate your observed data. Suppose that you have your parameters in a vector θ . The likelihood function \mathcal{L} tells you how good of a job you made using θ to approximate your observed adjacency matrix A . In practice, you’re after those special parameters $\hat{\theta}$ such that $\mathcal{L}_{\hat{\theta},A}$ is maximum. Mathematically:

$$\hat{\theta} = \arg \max_{\theta \in \Theta} \mathcal{L}_{\theta,A},$$

where Θ is the space of all possible parameters, and A is your given adjacency matrix.

So let’s make an extremely simplified example of what that means when using SBM to find communities. This sketched example has a planted partition, only two parameters, and it exclusively focuses on the simplest definition of community – the one I gave you earlier:

²² Gary King. *Unifying political methodology: The likelihood theory of statistical inference*. University of Michigan Press, 1998

²³ Edwin T Jaynes. *Probability theory: The logic of science*. Cambridge university press, 2003

simple non-overlapping assortative communities. I'm imposing these limitations for pedagogical purposes: in reality, SBMs are much more powerful than this.

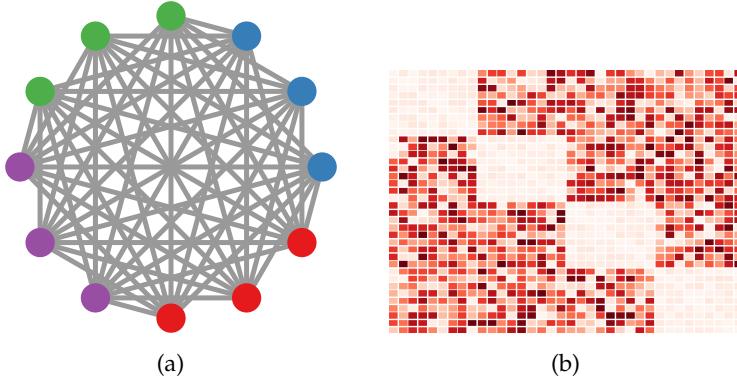


Figure 35.3: (a) A network with a disassortative community structure. (b) A possible disassortative SBM representation, darker cells indicate edges with higher probability.

I already mentioned in Section 18.2 why: SBM can find *disassortative* communities, communities in which the nodes tend to connect to their community mates less than chance. The classical definition I gave you earlier is only for assortative communities. You can compare the disassortative community structure from Figure 35.3(a) with the classical assortative one in Figure 35.1(a). SBMs don't break a sweat in this scenario: you can simply flip the parameters and – presto! – you switch from the classical model in Figure 35.2(b) to the disassortative model in Figure 35.3(b). I haven't even started explaining you the first community discovery method and we already found the first hole in the classical definition!

However, let's take it slow. To understand the SBM basics it is better to start with a cartoonishly simplified example and then introduce the features you can add to SBMs every time it'll be relevant in the book.

As mentioned, in the simplest SBM, we only have three parameters. The first two are the probabilities of two nodes connecting if they are – or are not – in the same community: θ_1 and θ_2 , respectively. The third (hyper)parameter θ_3 is the planted partition: it contains all node pairs that are in the same community.

Let's have the simplest possible \mathcal{L} function. Let's define a helper variable defined for a pair of nodes:

$$l_{\theta, A, u, v} = \begin{cases} \theta_1 - 1, & \text{if } A_{uv} = 1 \text{ and } (u, v) \in \theta_3 \\ \theta_2 - 1, & \text{if } A_{uv} = 1 \text{ and } (u, v) \notin \theta_3 \\ -\theta_1, & \text{if } A_{uv} = 0 \text{ and } (u, v) \in \theta_3 \\ -\theta_2, & \text{if } A_{uv} = 0 \text{ and } (u, v) \notin \theta_3. \end{cases}$$

In practice, if two nodes are connected, we subtract one from their

probability of connections in the SBM – since $\theta_1 > \theta_2$, this means we reward the case in which we put the two nodes in the same community. If they are not, we penalize ourselves proportionally to how sure we were they were connected: in this case we get the lowest penalty if we assumed them not being in the same community.

Then, we go over every node pair to check whether we assigned a high probability of edge existence to two nodes that were indeed connected, and we aggregate the scores:

$$\mathcal{L}_{\theta,A} = \sum_{u,v \in A} l_{\theta,A,u,v}.$$

Let's fix $\theta_1 = 1$ and $\theta_2 = 0.01$ for simplicity, since we're only interested in the partition (θ_3). If we apply \mathcal{L} to the perfect partition, the one following the colors in Figure 35.1(a), we get $\mathcal{L}_{\theta,A} = -8.78$ (I'm ignoring the diagonal because I don't allow self loops). Why?

We get zero contribution from the blocks in the diagonal, since they're fully connected and also in the same community (case 1 in the formula). We get $4 \times (\theta_2 - 1)$ contribution from the edges connecting nodes in different communities (case 2 in the formula) – note that I don't double count because I assume the network is undirected. We get zero contribution from case 3, as there are no disconnected nodes that we put in the same community. Finally, we get $482 \times -\theta_2$ contribution from the 482 disconnected node pairs in different communities (case 4).

Any other partition would return a lower likelihood. For instance, having only two communities, each fully including two blocks, has (following the four cases): $\mathcal{L}_{\theta,A} = 0 + (2 \times (\theta_2 - 1)) + (160 \times -\theta_1) + (322 \times -\theta_2) = -165.2$.

One of the problems of this approach is that the space of all possible partitions (θ_3) is huge. How do we know which one is best? Besides, how do we even know what's the correct number of communities? A reliable way to estimate maximum likelihoods in presence of unknowns such as these, is the Expectation Maximization algorithm²⁴.

Note that here I showed you an intuitive, but cartoonish, likelihood function. This will fail in most, if not all, real world networks. There are smarter alternatives out there, taking into account the degree distributions²⁵, and more that we will see in later chapters. There are also better heuristics to find the best partition²⁶.

Don't be deceived by the fact that the SBM approach I just described is clunky. I introduced it as a historic approach because it was the one used by sociologists before the network science renaissance of the late nineties. However, as I mentioned, SBMs are more sophisticated than this.

²⁴ Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1):1–22, 1977

²⁵ Brian Karrer and Mark EJ Newman. Stochastic blockmodels and community structure in networks. *Physical review E*, 83(1):016107, 2011

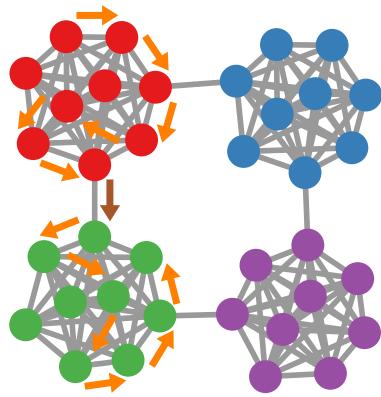
²⁶ Tiago P Peixoto. Efficient monte carlo and greedy heuristic for the inference of stochastic block models. *Physical Review E*, 89(1):012804, 2014a

Many community detection methods use different approaches, and some can be shown as equivalent to SBMs. One such key algorithm is modularity maximization. However, since this is closely related with evaluating the quality of a partition, I'm going to defer presenting modularity maximization methods to Section 36.1.

Another approach uses the spectrum of the graph^{27,28}. Well-separated communities will show large gaps in the eigenvalues. The eigenvectors are simply a coordinate system that you can use to place nodes in a multidimensional space and then use a standard clustering algorithm to find communities, such as k-means. I already covered the main concepts for this approach in Section 11.5, so I won't go into details here. Hereafter, we explore alternatives to the problem of discovering communities.

35.2 Random Walks

A common approach for detecting communities looks at network processes. Random walks (Chapter 11) are a popular choice. These walks tell us a lot about the community structure of the network. Consider the example in Figure 35.1(a). In the network, there are eight "border nodes" with a connection to a different community, out of 36 nodes. And only one edge out of nine they have points outside the community. So we know that the probability for a random walker to get out of the community it is visiting is very low.



²⁷ Ulrike Von Luxburg. A tutorial on spectral clustering. *Statistics and computing*, 17(4):395–416, 2007

²⁸ Mark EJ Newman. Finding community structure in networks using the eigenvectors of matrices. *Physical review E*, 74(3):036104, 2006

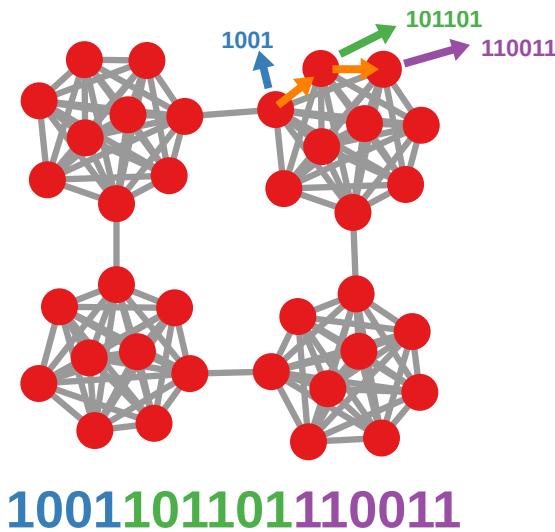
Figure 35.4: A random walk in a network with communities. The node color indicates to which community a node belongs. The arrows show each step of the random walker. The orange arrows are transitions between nodes in the same community. The brown arrow is a transition between nodes in different communities.

Appearing together in a random walk is a strong indication that two nodes are in the same community. Figure 35.4 shows why. When we perform our random walk, only one step out of the 13 we took crossed communities. Let's do some napkin math. The probability of being a "border node" is $8/36 = 0.\bar{2}$. The probability of picking the one edge going outside of the community if you are in a border node is $1/9 = 0.\bar{1}$, because the node has degree nine and only one edge

going out. So the probability of transitioning between communities in a random walk is $(8/36) \times (1/9) \sim 0.025$. Pretty low!

The guiding principle of detecting communities with random walks is that a random walk is likely to be trapped inside a community and to keep visiting nodes belonging to the same community. This has been exploited in a number of different ways^{29,30,31,32,33}. These are only a few examples of the many papers using this approach – you’re going to hear this excuse from me a lot, to save myself from citing literally everything and making this a book about community discovery.

The most known and best performing of these approaches is usually considered the map equation approach, or Infomap^{34,35}. The map equation is what you use to encode the random walk information with the minimum possible number of bits – i.e. minimizing the “code length”. Suppose that you give each node a binary ID. Since we have 36 nodes, we need around 5 bits, but we can save a little if we give shorter codes to central nodes (they are going to be visited more often). Then the cost of describing the random walk is simply the length of the code of the node multiplied by the number of times we’re going to see it in a random walk, which is given by the stationary distribution (Section 11.1). In the example from Figure 35.5, the orange walk is fully described by the bits in the figure: the node id sequence.



²⁹ Stijn Dongen. A cluster algorithm for graphs. 2000

³⁰ Pascal Pons and Matthieu Latapy. Computing communities in large networks using random walks. *J. Graph Algorithms Appl.*, 10(2):191–218, 2006

³¹ Vinko Zlatić, Andrea Gabrielli, and Guido Caldarelli. Topologically biased random walk and community finding in networks. *Physical Review E*, 82(6):066109, 2010

³² Venu Satuluri and Srinivasan Parthasarathy. Scalable graph clustering using stochastic flows: applications to community discovery. In *SIGKDD conference*, pages 737–746. ACM, 2009

³³ E Weinan, Tiejun Li, and Eric Vanden-Eijnden. Optimal partition and effective dynamics of complex networks. *PNAS*, 105(23):7907–7912, 2008

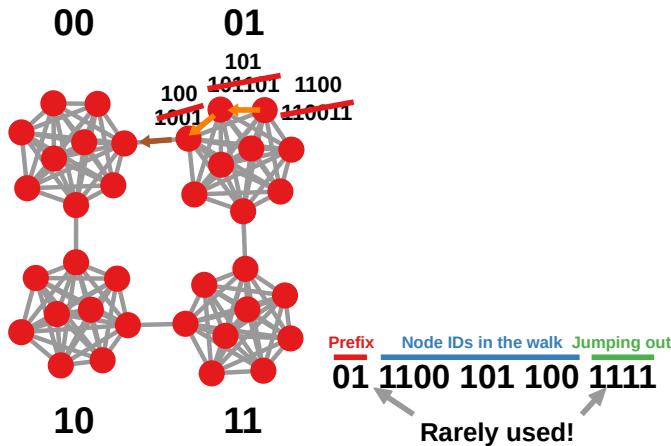
³⁴ Martin Rosvall and Carl T Bergstrom. Maps of random walks on complex networks reveal community structure. *PNAS*, 105(4):1118–1123, 2008

³⁵ Ludvig Bohlin, Daniel Edler, Andrea Lancichinetti, and Martin Rosvall. Community detection and visualization of networks with the map equation framework. In *Measuring Scholarly Impact*, pages 3–34. Springer, 2014

Figure 35.5: A binary node ID schema you’d use to encode a random walk. Each colored arrow points to the ID of the three nodes involved in the orange walk.

Infomap saves bits by using community prefixes. Nodes in the same community get the same prefix. So now we need fewer bits to uniquely refer to each node, because we can prepend the community code the first time and then omit it as long as we are in the same community. Since a community contains, in this case, 9 nodes instead

of 36, we can use shorter codes. We need to add an extra code that allows us to know we're jumping out of a community. This is an overhead, but the assumption here is that a random walker will spend most of its time in the community, so this community prefix and jump overhead is rarely used. Figure 35.6 shows this re-labeling process.



If the partition is good, we can compress the random walk information by a lot. Consider the example in Figure 35.7. Without communities we have no overhead, but we need to fully encode our 36 nodes. The path in orange is simply the sequence of node IDs and can be stored in 72 bits. If we have community partitions, we add the community prefixes and the jump overhead (for the community jump in brown), but the node IDs are shorter. The encoding of the same walk is 56 bits, and we can see that the overhead parts are tiny compared with the rest.

If my explanation still makes little sense, you can try out an interactive system showing all the mechanics of the map equation approach³⁶. Infomap has been adapted to numerous scenarios. Many involve hierarchical, multilayer and overlapping community detection, which we will explore in later chapters. Other modifications include adding some “memory” to the random walkers^{37,38} – effectively using higher order networks (Chapter 34).

This means that the walker is not randomly selecting destinations any more, but it follows a certain logic. Consider a network of flight travelers. If you fly from New York to Los Angeles, your next leg trip isn't random. You're much more likely, for instance, to come back to New York, since you were in LA just for a vacation or visiting family.

Some approaches do not use vanilla random walks, but also consider the information encoded in node attributes in the map equation³⁹.

Figure 35.6: The large two-digit codes are the IDs of the communities. Each node gets a new shorter ID, given that IDs need to be community-unique, rather than network-unique. Now the random walk uses the prefix (in red) to indicate in which community it is, then the new shorter node IDs (in blue) and finally adds an extra ID to indicate it's jumping out of a community (in green).

³⁶ <http://www.mapequation.org/apps/MapDemo.html>

³⁷ Michael T Schaub, Renaud Lambiotte, and Mauricio Barahona. Encoding dynamics for multiscale community detection: Markov time sweeping for the map equation. *Physical Review E*, 86(2):026112, 2012b

³⁸ Martin Rosvall, Alcides V Esquivel, Andrea Lancichinetti, Jevin D West, and Renaud Lambiotte. Memory in network flows and its effects on spreading dynamics and community detection. *Nature communications*, 5:4630, 2014

³⁹ Laura M Smith, Linhong Zhu, Kristina Lerman, and Allon G Percus. Partitioning networks with node attributes by compressing information flow. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 11(2):15, 2016

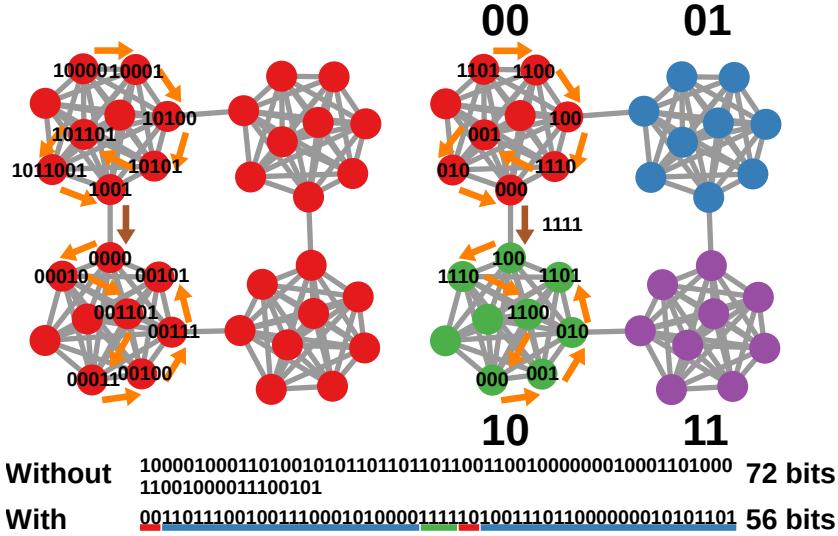


Figure 35.7: The cost of encoding a random walk with the naive scheme (left) compared with the Infomap scheme (right). In the Infomap scheme, I underline in red the community prefixes, in green the inter-community jump overhead, and in blue the node ID encoding.

Since these methods are based on a fundamentally random process – random walks – they tend to be non-deterministic. This means that running the same algorithm on the same input twice might return different results.

35.3 Label Percolation

Random walks are running a dynamic process to detect communities, meaning that we're performing some sort of event on the network to uncover the community structure. Another very popular dynamic approach is having nodes deciding for themselves to which community they belong by looking at their neighbors' community assignments.

We use node labels to indicate to which community each node belongs. We start with a network whose node labels are scattered randomly. Then each node looks at its neighbors and adopts the most common labels it sees (if there is a tie, it will choose a random one among the most popular). As a result, the labels will percolate through the network until we reach a state in which no more significant changes can happen. The assumption is that, in a community, nodes will end up surrounded by nodes with the same label. That is why this class of solutions is usually known as “label percolation” (or propagation).

At the beginning, nodes will switch their labels randomly. However, by chance, some nodes will eventually adopt the same label. If they are in the same community, all of a sudden this label is the only one with two nodes in the cluster. It starts becoming the majority label for many nodes in the cluster, and thus it will be eventually

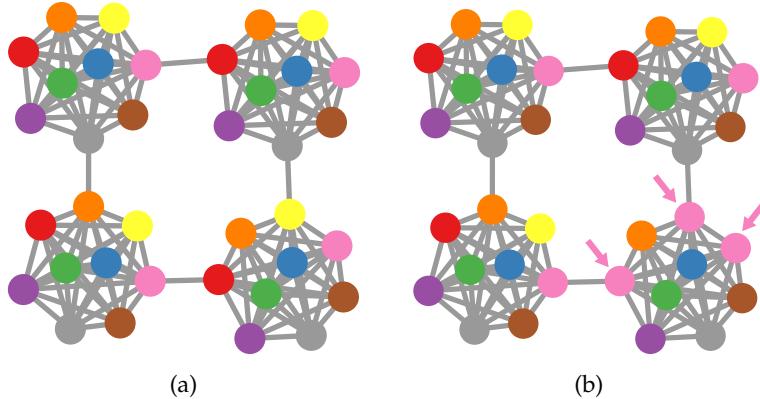


Figure 35.8: (a) Starting condition of label propagation algorithm, with labels distributed randomly in the network. (b) After some iterations, randomly, some neighbors will adopt the same label. The highlighted pink nodes will take over the community at the next time step.

adopted by everybody, as Figure 35.8 shows.

This approach is fairly straightforward and computationally simple. In fact, one of the claims to fame of such an algorithm is its time complexity: it runs linearly in terms on the number of edges in the network. You just have to iterate over your edge list a few times before convergence.

The most important dimension along which the many papers implementing label propagation community detection differ is in the strategy they employ for the nodes to look around their neighborhood. We can classify them in three classes: asynchronous⁴⁰ – which is also the original formulation of the label propagation principle –, semi-synchronous⁴¹, and synchronous⁴².

The asynchronous case uses the labels that the nodes had at the previous iteration. For instance, at the i th iteration a node will decide which label to adopt by looking at the majority label between its neighbors at the $(i - 1)$ th iteration. If some neighbors have, in the mean time, updated their label, this information is ignored, and will be used only at the $(i + 1)$ th iteration.

In the synchronous approach this is not the case: you always use the most up-to-date information you have. If some of your neighbors already changed their label, you look at their i th iteration label. Otherwise, you look at their $(i - 1)$ th iteration label. The semi-synchronous case is, as you might expect, a combination of the two.

Note that, just like in the random walk case, the label propagation algorithms are typically non-deterministic. The random choices nodes make when breaking ties among the most popular labels around them can lead to differences in the detected communities.

⁴⁰ Usha Nandini Raghavan, Réka Albert, and Soundar Kumara. Near linear time algorithm to detect community structures in large-scale networks. *Physical review E*, 76(3):036106, 2007

⁴¹ Gennaro Cordasco and Luisa Gargano. Community detection via semi-synchronous label propagation algorithms. In *2010 IEEE International Workshop on: Business Applications of Social Network Analysis (BASNA)*, pages 1–8. IEEE, 2010

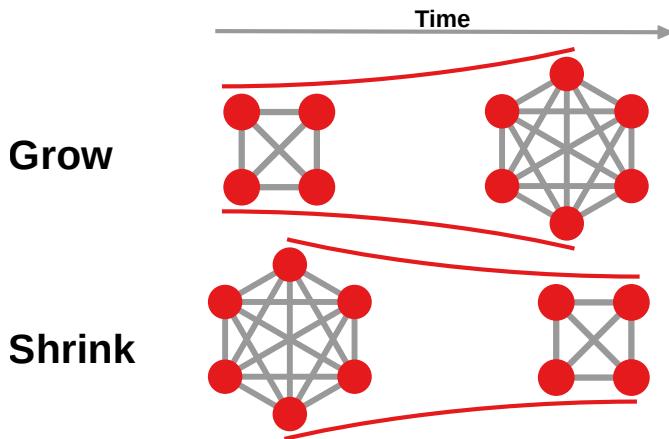
⁴² Jierui Xie, Boleslaw K Szymanski, and Xiaoming Liu. Slpa: Uncovering overlapping communities in social networks via a speaker-listener interaction dynamic process. In *2011 IEEE 11th International Conference on Data Mining Workshops*, pages 344–349. IEEE, 2011

35.4 Temporal Communities

Evolutionary Clustering

So far I've framed the community discovery problem as essentially static. You have a network and you want to divide it into densely connected groups. However, we saw in Section 7.4 that many of the graphs we see are views of a specific moment in time. Networks evolve and you might want to take that information into account. A couple of good review works^{43,44} focus on dynamic community discovery and can help you obtaining a deeper understanding of this problem. Let's explore what can happen to your communities over time.

One possibility is that the community will grow: it will attract new nodes that were previously unobserved. The other side of the coin is shrinking: nodes that were part of the community disappear from the network. Figure 35.9 shows visual examples of these events.



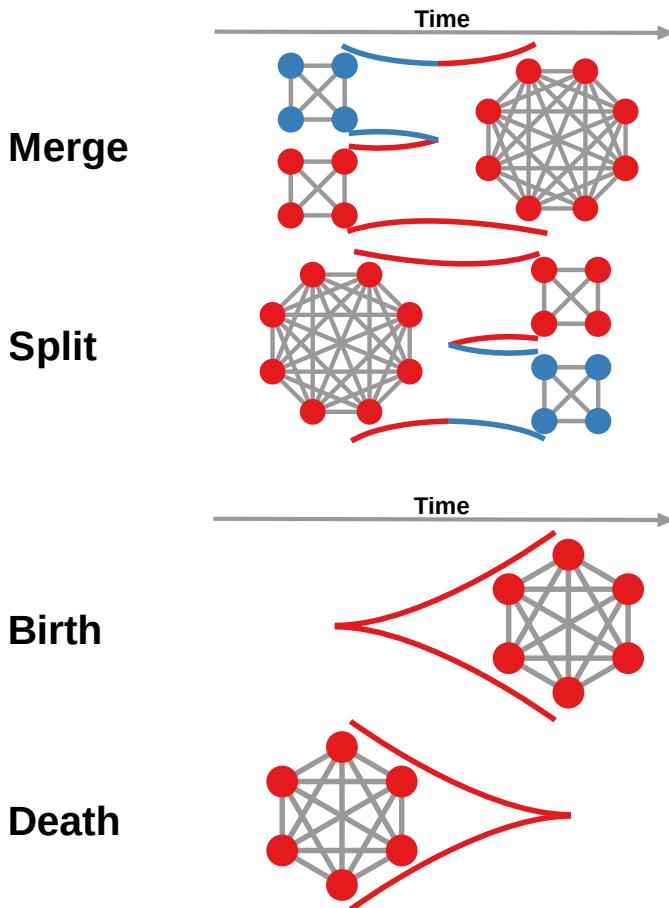
⁴³ Qing Cai, Lijia Ma, Maoguo Gong, and Dayong Tian. A survey on network community detection based on evolutionary computation. *IJBIC*, 8(2):84–98, 2016

⁴⁴ Giulio Rossetti and Rémy Cazabet. Community discovery in dynamic networks: a survey. *ACM Computing Surveys (CSUR)*, 51(2):35, 2018

Figure 35.9: Two things that can happen to your communities in an evolving network: growing and shrinking.

Another way for a community to grow is by merging with other communities. The difference with the previous case is that in the growth case the added nodes to the community were not previously observed. In this case they were, and they were classified in a different community. “Grow” happens with the addition of nodes, while “merge” usually happen with the addition of edges. Again, we can have the opposite case: a community which splits into two or more new communities, due to the loss of edges (rather than nodes, as it was the case in the “shrink” scenario). Figure 35.10 shows visual examples of these events.

Communities can also arise from nothing. This is like “grow”, except that none of the nodes forming the community were previously observed in the network. The converse is community death: every node which was part of it disappears from the network. Figure 35.11



shows visual examples of these events.

Note that, as everything else in this chapter, also this description is cartoonish. What happens in real world networks is way messier. Communities grow, shrink, split, and merge at the same time. Merges could also be partial, with communities “stealing” nodes from each other, resulting in an evolution that is not nearly as neat as the one I depicted here. Don’t assume that you’re going to be able to say, unequivocally, something like “community C split in C_1 and C_2 at time t ”.

How do you detect communities in an evolving graph? One possible approach could be to define a series of network snapshots, apply a community discovery algorithm to each of these snapshots, and then combine the results into a single clustering^{45,46,47}. This is fine in some scenarios, but is generally not advisable, as it makes the quiet assumption that snapshots are sort of independent.

It becomes challenging to link the community discovery of each snapshot to the next. For each community at time t you’re trying to find the community at time $t + 1$ that is the most similar to it,

Figure 35.10: Two things that can happen to your communities in an evolving network: merging and splitting.

Figure 35.11: Two things that can happen to your communities in an evolving network: birth and death.

⁴⁵ John Hopcroft, Omar Khan, Brian Kulis, and Bart Selman. Tracking evolving communities in large linked networks. *Proceedings of the National Academy of Sciences*, 101(suppl 1):5249–5253, 2004

⁴⁶ Sitaram Asur, Srinivasan Parthasarathy, and Duygu Ucar. An event-based framework for characterizing the evolutionary behavior of interaction graphs. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 3(4):16, 2009

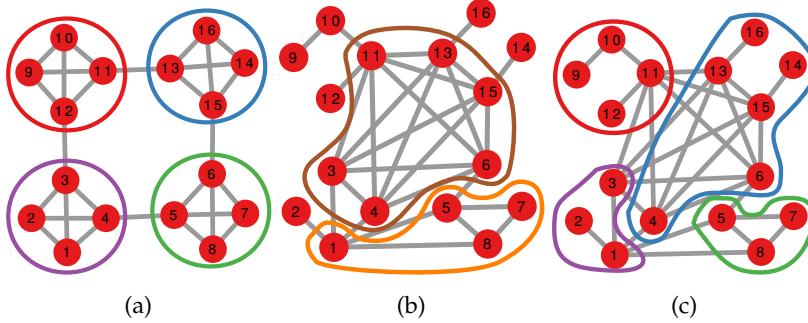
⁴⁷ Gergely Palla, Albert-László Barabási, and Tamás Vicsek. Quantifying social group evolution. *Nature*, 446(7136):664, 2007

and say that the most recent community is an evolution of the older one. A possible similarity criterion would be calculating the Jaccard coefficient.

A better solution is performing evolutionary clustering⁴⁸. This means that we add a second term to whatever criterion we use to find communities in a snapshot – a procedure sometimes called “smoothing”. Suppose you’re using Infomap. The aim of the algorithm, as I presented earlier, is to encode random walkers with the lowest number of bits. Let’s say this is its quality function – which is known as code length (CL).

In evolutionary clustering you don’t just optimize CL . You have CL as a term in your more general quality function Q . The other term in Q is consistency. For simplicity sake, let’s just assume it is some sort of Jaccard coefficient between the partitions at time t and the partition at time $t - 1$. To sum up, a very simple evolutionary clustering evaluates the partition p_t at time t as:

$$Q_{p_t} = \alpha CL_{p_t} + (1 - \alpha) J_{p_t, p_{t-1}}.$$



Here, α is a parameter you can specify which regulates how much weight you want to give to your previous partitions. For $\alpha = 1$ you have standard clustering, while for $\alpha = 0$ the new information is discarded and you only use the partition you found at the previous time step. Figure 35.12 shows you that maximizing CL_{p_t} might yield significantly different results than maximizing a temporally-aware Q_{p_t} function.

This is only one – the simplest – of the many ways to perform smoothing, which the other review works I cited describe more in details. However, all these methods (and the ones that follow) have something in common: they are all at odds with the classical definition of community that I gave you earlier. That is because, at time $t + 1$, we’re not simply trying to group nodes in the same community according to the density of their connections. Eventually, we’re going to end up with a partition with many edges running

⁴⁸ Deepayan Chakrabarti, Ravi Kumar, and Andrew Tomkins. Evolutionary clustering. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 554–560. ACM, 2006

Figure 35.12: (a) The community partition of a graph at time t . (b) A partition of the graph at time $t + 1$ exclusively optimizing the code length, using Infomap. (c) A partition of the graph at time $t + 1$ balancing a good code length and consistency with the partition at time $t + 1$.

between communities, which is against the traditional definition of community. Together with the ability of SBMs to find disassortative communities, these are yet more cracks appearing in the classical community detection assumption of assortative communities.

Smoothing is not necessarily applied to adjacent snapshots: you can have a longer memory looking at $t - 2$, $t - 3$, and so on^{49,50}. In alternative approaches, you can skip the smoothing altogether. You can identify a “core-node” which is the center of the community and will identify it for all snapshots. You then find communities around that node^{51,52}.

Other Approaches

Alternatives to evolutionary clustering exist. You could find an optimal partition only for the very first snapshot of your network. As you receive a new snapshot, rather than starting from scratch and then smoothing, you can adapt the old communities to the new network, whether you do it via global optimization⁵³, or using a specific set of rules to update the old communities^{54,55,56}.

Another approach consists in defining a dynamic null model: a null version of your evolving network which has no communities⁵⁷, much like a random graph. Then you look at deviations from this expected null model in the network as the potential sources of dynamic communities.

You can use SBMs in this case as well – you can use SBMs for *any case*, really. SBMs tend to be more principled than evolutionary clustering approaches, because they model temporal communities directly⁵⁸, rather than chasing communities around as your network evolves. As an example, you could use a tensor representation in which each slice of the tensor is a snapshot of the network. Since a tensor is nothing more than a high dimensional matrix, and SBMs understand matrices, you can make a tensor-SBM⁵⁹. One nice thing about the SBM approach is that it allows to estimate from data the timescale at which the community structure changes – which is inferred by the coupling between snapshots. This is nice because then you don’t need to decide yourself the granularity of the temporal observation. Basing your inferences on data rather than taking a guess is always a plus!

A final approach is not to consider the different snapshots as separate, but taking the entire structure of the network as input all at once, as if it were a single structure. For instance, you can split each node v into many meta-nodes v_{t_1}, v_{t_2}, \dots connected to each other by special edges^{60,61,62,63,64}. This is similar to performing multi-layer community discovery, which we’ll see later.

⁴⁹ Mark Goldberg, Malik Magdon-Ismail, Srinivas Nambirajan, and James Thompson. Tracking and predicting evolution of social communities. In *SocialCom*, pages 780–783. IEEE, 2011

⁵⁰ Matteo Morini, Patrick Flandrin, Eric Fleury, Tommaso Venturini, and Pablo Jensen. Revealing evolutions in dynamical networks. *arXiv preprint arXiv:1707.02114*, 2017

⁵¹ Zhengzhang Chen, Kevin A Wilson, Ye Jin, William Hendrix, and Nagiza F Samatova. Detecting and tracking community dynamics in evolutionary networks. In *ICDMW*, pages 318–327. IEEE, 2010

⁵² Yi Wang, Bin Wu, and Xin Pei. Comm-tracker: A core-based algorithm of tracking community evolution. In *ADMA*, pages 229–240. Springer, 2008b

⁵³ K Miller and Tina Eliassi-Rad. Continuous time group discovery in dynamic graphs. Technical report, LLNL, 2010

⁵⁴ Giulio Rossetti, Luca Pappalardo, Dino Pedreschi, and Fosca Giannotti. Tiles: an online algorithm for community discovery in dynamic social networks. *Machine Learning*, 106(8):1213–1241, 2017

⁵⁵ Yizhou Sun, Jie Tang, Jiawei Han, Manish Gupta, and Bo Zhao. Community evolution detection in dynamic heterogeneous information networks. In *MLGraphs*, pages 137–146. ACM, 2010

⁵⁶ Lei Tang, Huan Liu, Jianping Zhang, and Zohreh Nazeri. Community evolution in dynamic multi-mode networks. In *SIGKDD*, pages 677–685. ACM, 2008

⁵⁷ Danielle S Bassett, Mason A Porter, Nicholas F Wymbs, Scott T Grafton, Jean M Carlson, and Peter J Mucha. Robust detection of dynamic community structure in networks. *Chaos Journal*, 23(1):013142, 2013

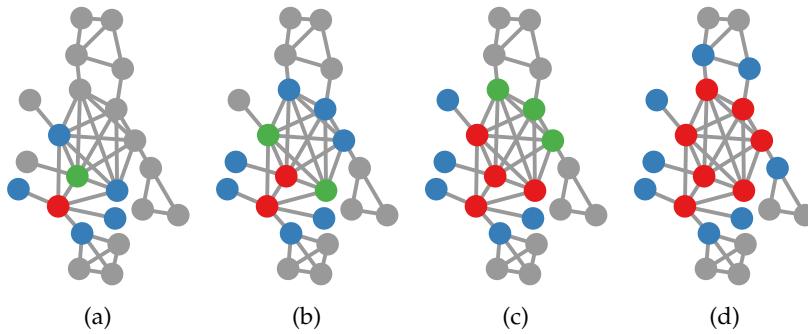
⁵⁸ Tiago P Peixoto. Inferring the mesoscale structure of layered, edge-valued, and time-varying networks. *Physical Review E*, 92(4):042807, 2015

⁵⁹ Marc Tarrés-Deulofeu, Antonia Godoy-Lorite, Roger Guimera, and Marta Sales-Pardo. Tensorial and bipartite block models for link prediction in layered networks and temporal networks. *Physical Review E*, 99(3):032307, 2019

35.5 Local Communities

In some cases, you are not interested in grouping every node into a community. I'm not just referring to allowing nodes to be part of no communities – a feature included in many algorithms, regardless of their guiding principle. Sometimes, you want to find local communities: you're interested in knowing the communities around a specific (set of) node(s), regardless of the rest of the network. This makes sense if the network is very large and some nodes are just too far to ever influence the results on your specific objectives. Or you cannot analyze it fully because it would take too much memory. Or it might take too much time to access the entire network, imposing you to sample it (see Chapter 29).

This is usually done by exploring the graph one node at a time, putting nodes into different bins according to their exploration status – and their community affiliation. For instance, you start from a seed node v_0 , which by definition is part of your local community \mathcal{C} . All of its neighbors are part of the unexplored node set \mathcal{U} .



You iterate over all members of \mathcal{U} , trying to find the one that would maximize some community quality function. For instance, it could be simply the number of edges connecting inside \mathcal{C} – with ties broken randomly. We add to \mathcal{C} the v_1 node with the most edges to the local community. Then, \mathcal{U} is updated with the new neighbors, the ones v_1 has but v_0 did not.

We continue until we have reached our limit: we explored the number of nodes we wanted to test, or we ran out of time, or we actually explored all nodes in the component to which v_0 belongs. Figure 35.13 shows some steps of the process. As you can see, we can terminate after we explore a certain set of nodes. At that point, we detected the local community of node v_0 , without exploring the entire network. We did not explore the blue nodes – although we know they exists – and we're absolutely clueless about the existence of the grey nodes.

⁶⁰ Jimeng Sun, Christos Faloutsos, Spiros Papadimitriou, and Philip S Yu. Graphscope: parameter-free mining of large time-evolving graphs. In *SIGKDD*, pages 687–696. ACM, 2007a

⁶¹ Laetitia Gauvin, André Panisson, and Ciro Cattuto. Detecting the community structure and activity patterns of temporal networks: a non-negative tensor factorization approach. *PloS one*, 9(1):e86028, 2014

⁶² Leto Peel and Aaron Clauset. Detecting change points in the large-scale structure of evolving networks. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015

⁶³ Amir Ghasemian, Pan Zhang, Aaron Clauset, Christopher Moore, and Leto Peel. Detectability thresholds and optimal algorithms for community structure in dynamic networks. *Physical Review X*, 6(3):031005, 2016

⁶⁴ Tiphaine Viard, Matthieu Latapy, and Clémence Magnien. Computing maximal cliques in link streams. *Theoretical Computer Science*, 609:245–252, 2016

Figure 35.13: The process of discovering local communities. Red: nodes in \mathcal{C} . Blue: nodes in \mathcal{U} . Green: nodes maximizing the number of edges in \mathcal{C} and therefore being added to the community.

The algorithm I just described is one⁶⁵ of the many possible^{66,67,68}. All these algorithms are variations of this exploration approach. More alternatives have been proposed, for instance using random walks like Infomap to explore the network^{69,70}. You can explore the literature more in depth using one of the survey papers I cited at the beginning of the chapter.

35.6 Using Clustering Algorithms

I barely started scratching the complex landscape of different approaches to community discovery. We're going to have time in the next chapters to explore even more variations. However, a question might have already dawned on you. *If there are such different approaches to detecting communities, how do I find the one that works for me?* And, *how do I maximize my chances of finding high quality communities?* As you might expect, the answers to these questions are difficult and often tend to be subjective.

Let's start from the second one: designing a strategy to ensure you find close-to-optimal communities. In machine learning, we discovered a surprising lesson. If you want to improve accuracy, designing the most sophisticated method in the world usually helps only up to a certain point. Having many simple methods and averaging the results could potentially yield better results.

This observation is at the basis of what we know as “consensus clustering” (or ensemble clustering)⁷¹. This strategy has been applied to detecting communities⁷² in the way you'd expect. Take a network, run many community discovery algorithms on it, average the results. Figure 35.14 shows an example of the procedure. Note how none of the methods (Figure 35.14(a-c)) found the best communities, which is their consensus: Figure 35.14(d). Note also how the third method finds rather absurd and long stretched sub-optimal communities. However, its evident blunders are easily overruled by the consensus between the other two methods, and its tiebreaker improves the overall partition.

This is a fine strategy, but you should not apply it blindly. You have to make sure the ensemble of community detection algorithms you're considering is internally consistent. In particular, the methods should have a coherent and compatible definition of what a community is. Limiting ourselves to the small perspective on the problem from this chapter – ignoring all that is coming next – combining a dynamic community discovery with a static local community discovery will probably not help.

Even mashing together superficially similar algorithms might result in disaster. For instance, the flow-based communities Infomap

⁶⁵ Aaron Clauset. Finding local community structure in networks. *Physical review E*, 72(2):026132, 2005

⁶⁶ James P Bagrow. Evaluating local community methods in networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(05):P05001, 2008

⁶⁷ Feng Luo, James Z Wang, and Eric Promislow. Exploring local community structures in large networks. *Web Intelligence and Agent Systems: An International Journal*, 6(4):387–400, 2008

⁶⁸ Symeon Papadopoulos, Andre Skusa, Athena Vakali, Yiannis Kompatsiaris, and Nadine Wagner. Bridge bounding: A local approach for efficient community discovery in complex networks. *arXiv preprint arXiv:0902.0871*, 2009

⁶⁹ Lucas GS Jeub, Prakash Balachandran, Mason A Porter, Peter J Mucha, and Michael W Mahoney. Think locally, act locally: Detection of small, medium-sized, and large communities in large networks. *Physical Review E*, 91(1):012821, 2015

⁷⁰ Pasquale De Meo, Emilio Ferrara, Giacomo Fiumara, and Alessandro Provetti. Mixing local and global information for community detection in large networks. *Journal of Computer and System Sciences*, 80(1):72–87, 2014

⁷¹ Alexander Strehl and Joydeep Ghosh. Cluster ensembles—a knowledge reuse framework for combining multiple partitions. *Journal of machine learning research*, 3(Dec):583–617, 2002

⁷² Andrea Lancichinetti and Santo Fortunato. Consensus clustering in complex networks. *Scientific reports*, 2:336, 2012

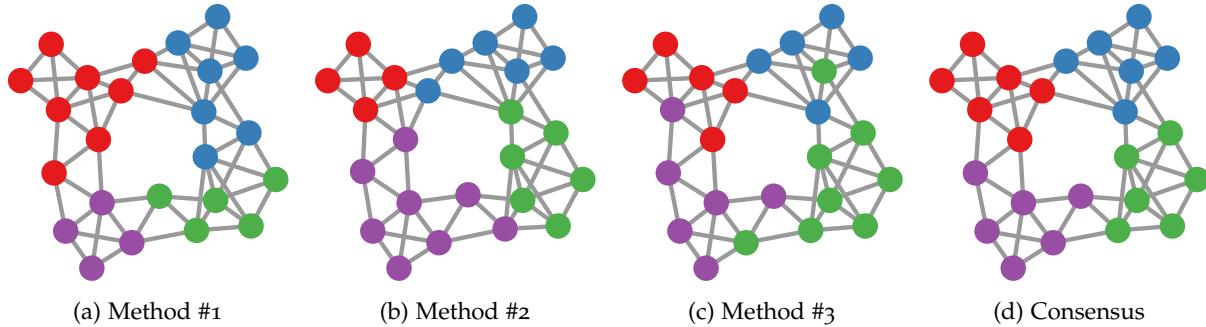


Figure 35.14: An example of consensus clustering. Node color represents the community. (a-c) The results of three independent methods. (d) Their majority vote.

returns aren't extremely compatible with the density optimization algorithms we'll see in the next chapter, even if the community definitions on which they are based don't look too dissimilar.

So, how do you go about choosing which algorithms to include in your ensemble? In a paper of mine⁷³, I explore the relationship between around 70 algorithms, comparing how similar their resulting partitions are. This results in an algorithm similarity network, which has distinct communities: groups of algorithms returning potentially interchangeable results that are significantly different from algorithms in a different group.

Figure 35.15 shows the result. Note that, in there, I label each algorithm with a tag. Chapter 53 contains a map from the tag to a resource where to recover the specific algorithm.

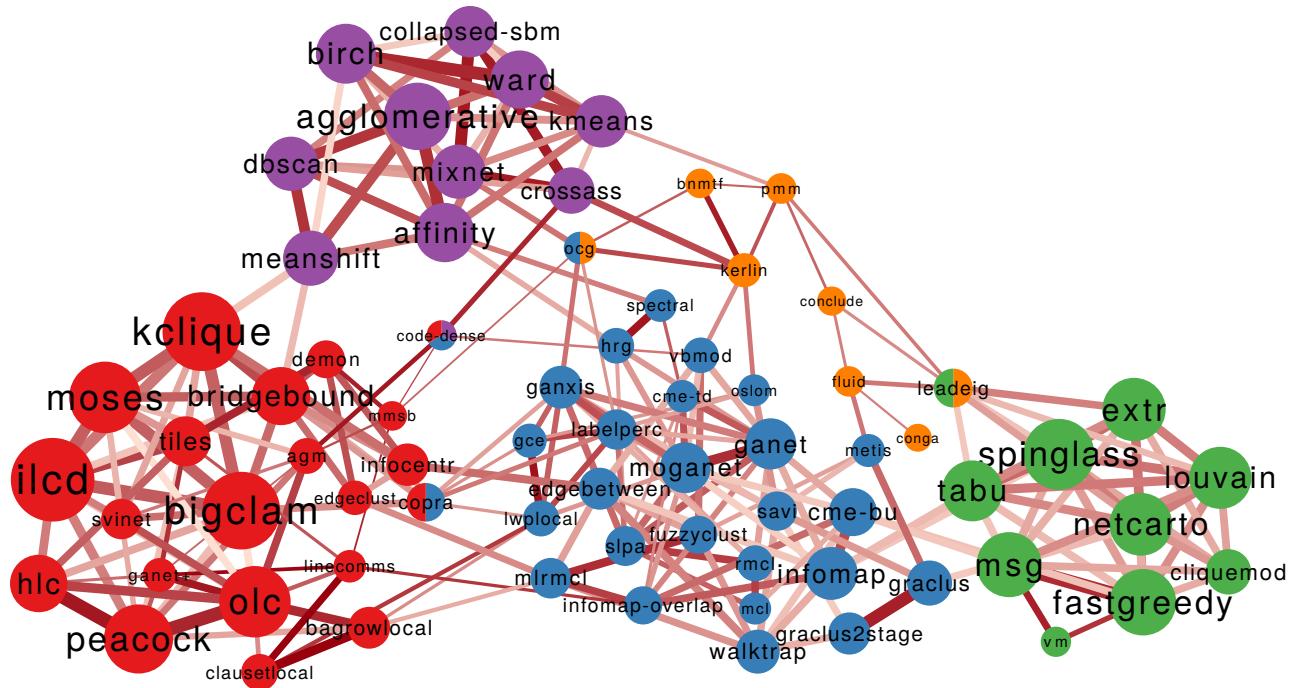
Now what? Well, we could... aehm... detect... communities... on this – I love being meta. These communities of community discovery can drive you in choosing your ensemble set. To find them, I use a version of Infomap allowing nodes to be part of multiple communities. This is the so-called overlapping community discovery, the topic of Chapter 38. Most algorithms allowing overlapping communities are in the red community in the figure – along with the local clustering approach I presented in Section 35.5.

The blue community contains Infomap and the label propagation approaches I explained earlier (Sections 35.2 and 35.3). This shows the close relationship between the two. The purple community includes methods departing from the classical “internal density” definition, to use a “neighbor similarity” approach. These will be covered in detail in Section 39.4.

A very popular approach is using a quality measure to evaluate how good a partition is, and then finding a smart strategy to optimize it. Most methods applying this strategy are in the green community, which is the one we're exploring in the next chapter.

Before moving to it, let me highlight an important lesson that we learn from this algorithm similarity network. Its communities

⁷³ Michele Coscia. Discovering communities of community discovery. In *Proceedings of the 2019 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, pages 1–8, 2019



are well-defined. This means that there are different and mutually exclusive notions of what a community is. This is yet another proof that the naive definition of community commonly accepted without criticism must be only one of the many possible. In fact, we can go deeper than this. The notion that there is a golden partition of the network is a utopia. As I mentioned at the beginning of the chapter, community discovery is more useful than that: it decomposes a network and simplifies it. Since there are innumerable ways – and reasons why – to simplify a network, then there are innumerable approaches to community discovery, and they are all valid even if they don't chase the mythical golden pot of "true" communities at the end of the rainbow.

35.7 Summary

1. According to the classical definition, communities in complex networks are groups of nodes densely connected to each other and sparsely connected to the rest of the network. They are one of the most common mesoscale organizations of real world networks.
2. One of the oldest approaches in community detection is to assume a planted partition of nodes in the network and then finding the distributions of nodes in communities that has the highest

Figure 35.15: The community detection algorithm similarity network. Each node is a community discovery algorithm. They are connected if the two algorithms return similar partitions. I use the node color to represent the algorithm's community. Multicolored nodes belong to multiple communities.

likelihood of explaining the observed connections.

3. A random walker would tend to be trapped in a community, because most of the neighbors of a node are part of its same community. By the same principle, we can detect communities by letting nodes assume the community label that is most common among their neighbors.
4. Networks evolve and so do communities. One can track the evolution of communities by having a two-part community quality function. One part tells us how well we're partitioning the network, the other tells us how compatible the new communities are with the old ones.
5. Sometimes your input network is too big or you have no interest in partitioning all of it. In that case, you can perform local community detection, detecting communities only in the neighborhoods of one or more query nodes.
6. There are hundreds of community detection algorithms. To choose one, you need to know what type of communities it returns. Alternatively, you can perform ensemble clustering, averaging out the results of multiple algorithms.
7. The classical community definition is assortative. Disassortative communities can exist, where nodes don't like to connect to members of the same group. Temporal communities are also not always assortative. This shows that there are more types of communities than the one assumed by the classical definition and they are all valid objectives you can follow to simplify your network.

35.8 Exercises

1. Find the communities in the network at <http://www.networkatlas.eu/exercises/35/1/data.txt> using the label propagation strategy. Which nodes are in the same community as node 1?
2. Find the local communities in the same network using the same algorithm, by only looking at the 2-step neighborhood of nodes 1, 21, and 181.
3. Suppose that the network at <http://www.networkatlas.eu/exercises/35/3/data.txt> is a second observation of the previous network. Perform the label propagation community detection on it and use the Jaccard coefficient to determine how different the communities containing nodes 1, 21, and 181 are.

36

Community Evaluation

How do you know if you found a good partition of nodes into communities? Or, if you have two competing partitions, how do you decide which is best? In this chapter, I present to you a battery of functions you can use to solve this problem. Why a “battery” of functions? Doesn’t “best” imply that there is some sort of ideal partition? Not really. What’s “best” depends on what you want to use your communities for. Different functions privilege different applications. So we need a quality function per application and you need to carefully choose your evaluation strategy to match the problem definition you’re trying to solve with your communities.

Think about “evaluating your communities” more as a data exploration task than a quest to find the ultimate truth. Since there is no one True partition – and not even one True definition of community as I suggested in the previous chapter –, there also cannot be one True quality function. You have, instead, multiple ways to see different kinds of communities, some of which might be more or less useful given the network you have and the task you want to perform.

In the first two sections, I start by focusing on functions that only take into account the topological information of your network. In this case, the only thing that matters are the nodes and edges – at most we can consider the direction and/or the weight of an edge.

In the latter two sections I move to a different perspective. First, we consider the network as essentially dynamic and we use communities as clues as to which links will appear next, under the assumption the communities tend to densify: it is much more likely that a new link will appear between nodes in the same community. Finally, we look at metadata that could be attached to nodes, which might be providing some sort of “ground truth” for the actual communities in which nodes are grouped into in the real world.

36.1 Modularity

As a Quality Measure

When it comes to functions evaluating the goodness of a community partition using exclusively topological information, there is one undisputed queen: modularity¹. You shouldn't be fooled by its popularity: modularity has severe known issues that limits its usefulness. We'll get to those in the second half of this section.

Modularity is a measure following closely the classical definition of community discovery. It is all about the internal density of your communities. However, you cannot simply maximize internal density, as the partition with the highest possible density is a degenerate one, where you simply have one community per edge – two connected nodes have, by definition, a density of one.

¹ Mark EJ Newman. Modularity and community structure in networks. *Proceedings of the national academy of sciences*, 103(23):8577–8582, 2006b

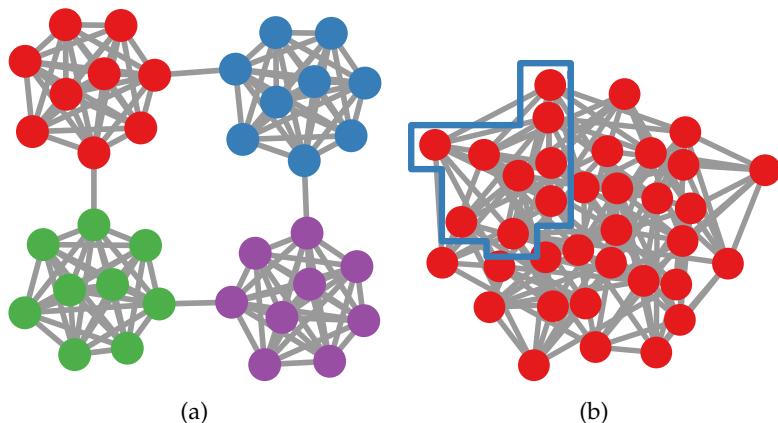


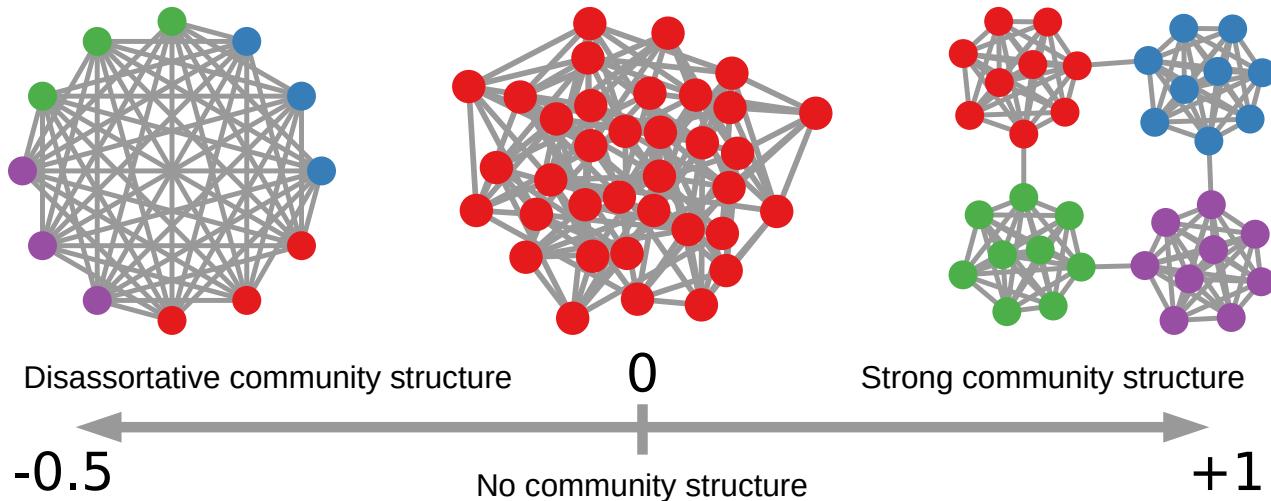
Figure 36.1: (a) A network with a community structure. The node colors represent the community partition. (b) A configuration model of (a), preserving the number of nodes, edges, and degree distribution. The blue outline identifies the nodes that were grouped in the blue community in (a).

Modularity solves this issue by comparing the observed network with a random expectation. For instance, consider the network in Figure 36.1(a). If we were to create a randomized version of it, it'd look like the graph in Figure 36.1(b). In Figure 36.1(b), each node has the very same degree that it has on the left. However, the edges are shuffled around. It is clear that this random network has no community structure. The difference between the two networks is that the communities of nine nodes have many more links inside them than any grouping of nine nodes in Figure 36.1(b).

At an abstract level, modularity is the comparison between the observed number of edges inside a community and the expected number of edges. The expectation is based on a null model of a random graph with the same degree distribution as the observed graph (i.e. a configuration model, Section 18.1). In Figure 36.1, we see that this number is positive: there are more edges in the community structure network than in its randomized version. The blue community

in Figure 36.1(a) contains 36 edges – all communities in the figure do. Picking those nodes from Figure 36.1(b) results in finding only 17 edges among them.

The domain of the modularity function is thus defined between $+1$ and -0.5 , as Figure 36.2 shows. A positive modularity happens when our partition finds nodes whose number of edges exceeds null expectation. When expectation exactly matches the number of edges in our community partition, modularity is zero. You can achieve negative modularity by trying to group nodes together that connect to each other less than chance. This can be a reasonable scenario: for instance, if you have disassortative communities (see Section 30.2). Note that, in the leftmost graph in Figure 36.2, nodes of the same color do not connect with each other.



The question is: how do we build this expectation, mathematically? This boils down to estimating the connection probability between any two nodes u and v . If this were a true random graph ($G_{n,p}$) it'd be easy: the connection probability is $p = 2|E|/|V|$. But we have the constraint of keeping the degree distribution. Each node v has a number of connection opportunities equal to its degree. The number of possible wirings we can make in the network is twice the number of edges. In a configuration model, the probability of connecting u and v is $(k_u k_v)/2|E|$.

Now we have all we need to build the modularity formulation:

$$M = \frac{1}{2|E|} \sum_{u,v \in V} \left[A_{uv} - \frac{k_u k_v}{2|E|} \right] \delta(c_v, c_u),$$

where A is our adjacency matrix, and δ is the Kronecker delta: a function return one if u and v are in the same community ($c_u = c_v$),

Figure 36.2: The domain of modularity, with example partitions returning a given value, from -0.5 (disassortative communities) to $+1$ (assortative communities, the most common case), passing via 0 (random graph with no communities).

zero otherwise.

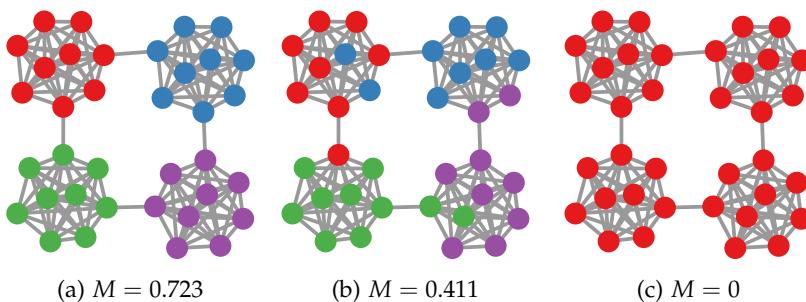
Modularity's formula is scary looking, but it ought not to be. In fact, it's crystal clear. Let me rewrite it to give you further guidance:

$$M = \frac{1}{2|E|} \sum_{u,v \in V} \left[A_{uv} - \frac{k_v k_u}{2|E|} \right] \delta(c_v, c_u),$$

which translates into: for every pair of nodes in the same community subtract from their observed relation the expected number of relations given the degree of the two nodes and the total number of edges in the network, then normalize so that the maximum is 1.

Modularity and Stochastic Blockmodels are related. Optimizing the community partition following modularity is proven to be equivalent to a special restricted version of SBM². Specifically, you need to use the degree-correlated SBM – since it fixes the degree distribution just like the configuration model does (which is the null model on which modularity is defined). Then, you must fix p_{in} and p_{out} – the probabilities of connecting to nodes inside and outside their community – to be the same for all nodes.

In general, you can use both to evaluate the quality of your partition, but there are subtle differences. SBM is by nature generative: it gives you connection probabilities between your nodes. Modularity doesn't. On the other hand, modularity has this inherent test against a null graph which you don't really have in SBMs. In fact, you can easily extend modularity in such way that you can talk about a statistically significant community partition, one that is sufficiently different from chance³.



² Mark EJ Newman. Equivalence between modularity optimization and maximum likelihood methods for community detection. *Physical Review E*, 94(5):052315, 2016a

³ Brian Karrer, Elizaveta Levina, and Mark EJ Newman. Robustness of community structure in networks. *Physical review E*, 77(4):046119, 2008

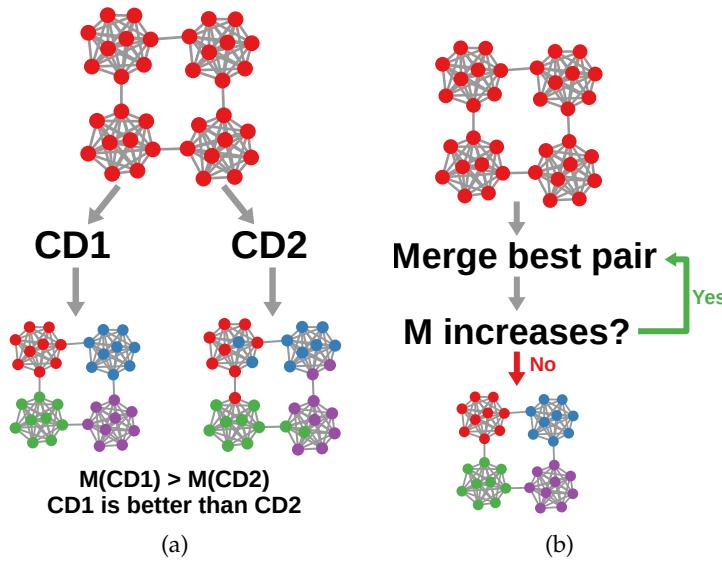
Figure 36.3: A network with a community structure. The node colors represent the community partition. (a) Optimal partition. (b) Sub optimal partition. (c) Partition grouping all nodes in the same community.

Modularity also gives us an intuition about whether a partition is better than another, without the need of calculating the likelihood, which is a more generic tool that was not developed with networks in mind – in fact, I introduced it in Section 4.3. We can compare partitions and see that a higher modularity implies a better partition, as Figure 36.3 shows. Moving nodes outside the optimal partition lowers modularity (compare the scores in the captions of Figures 36.3(a) and 36.3(b)). If we do not do community discovery and return

a single partition (Figure 36.3(c)), modularity will be equal to zero.

As a Maximization Target

As I mentioned earlier, modularity can be used in two ways. So far, we've seen the use case of evaluating your partitions. You start from a graph, you try two algorithms (or the same algorithm twice) and you get two partitions. The one with the highest modularity is the preferred one – see Figure 36.4(a).



The alternative is to directly optimize it: to modify your partition in a smart way so that you'll get the highest possible modularity score – see Figure 36.4(b). For instance, your algorithm could start with all nodes in a different community. You identify the node pair which would contribute the most to modularity and you merge it in the same community. You repeat the process until you cannot find any community pair whose merging would improve modularity^{4,5}. Most approaches following this strategy return hierarchical communities, recursively including low level ones in high level ones, and I cover them in detail in Chapter 37.

But there are other ways to optimize modularity. One strategy is to progressively condense your network such that you preserve its modularity⁶. Or using modularity to optimize the encoding of information flow in the network, bringing it close to the Infomap philosophy⁷. Another approach is using genetic algorithms^{8,9} or extremal optimization¹⁰: an optimization technique similar to genetic algorithms, which optimizes a single solution rather than having a pool of potential ones.

Other approaches include, but are not limited to:

Figure 36.4: (a) The workflow of using modularity as a quality criterion for your partitions. (b) The workflow of using modularity as an optimization target to find the best community partition.

⁴ Mark EJ Newman. Fast algorithm for detecting community structure in networks. *Physical review E*, 69(6):066133, 2004

⁵ Aaron Clauset, Mark EJ Newman, and Cristopher Moore. Finding community structure in very large networks. *Physical review E*, 70(6):066111, 2004

⁶ Alex Arenas, Jordi Duch, Alberto Fernández, and Sergio Gómez. Size reduction of complex networks preserving modularity. *New Journal of Physics*, 9(6):176, 2007

⁷ Azadeh Nematzadeh, Emilio Ferrara, Alessandro Flammini, and Yong-Yeol Ahn. Optimal network modularity for information diffusion. *Physical review letters*, 113(8):088701, 2014

⁸ Clara Pizzuti. Ga-net: A genetic algorithm for community detection in social networks. In *International conference on parallel problem solving from nature*, pages 1081–1090. Springer, 2008

⁹ Clara Pizzuti. A multiobjective genetic algorithm to find communities in complex networks. *IEEE Transactions on Evolutionary Computation*, 16(3):418–430, 2012

¹⁰ Jordi Duch and Alex Arenas. Community detection in complex networks using extremal optimization. *Physical review E*, 72(2):027104, 2005

- Progressively merging cliques, under the assumption that a clique is a structure that has the highest possible modularity¹¹;
- Performing the merging of communities I describe earlier allowing multiple communities to merge at the same time and then refining the results by allowing single nodes to move at the end¹²;
- Using simulated annealing¹³, integrated by using spinglass dynamics¹⁴. This technique can also take into account whether your network is signed¹⁵ – i.e. it has positive and negative connections (see Section 7.2), a special and simpler case of multilayer community discovery, which I'll cover in details in Chapter 40;
- Using Tabu search, another optimization technique related to simulated annealing and working mostly using local information¹⁶;
- Even including geospatial terms in the definition, when your nodes live into an actual geometric space¹⁷.

As you can gather from the number of references, we network folks really like to optimize our modularities.

Expanding Modularity

Unfortunately, modularity is not the end-all be-all of community detection as it initially appeared to be. There are several issues with it. We start by looking at the less problematic – but still annoying – ones. If you go back to the formula, you'll recognize that it is *a little bit too simple*.

The standard definition of modularity works exclusively with undirected, unweighted, disjoint partitions. We'll take care about extending modularity to cover the overlapping case in Chapter 38. For now, let's see what we can do when our graphs have directed edges and/or weighted ones.

The most straightforward way to extend modularity when your graphs have directed edges is simply modifying your expected number of edges between two nodes¹⁸. If in the undirected case we simply used the degree for both nodes u and v , now we have to use their in- and out-degree alternatively. So the expectation turns from $(k_u k_v) / 2|E|$ into $(k_u^{in} k_v^{out}) / |E|$ for the $v \rightarrow u$ edges. Modularity thus becomes:

$$M = \frac{1}{2|E|} \sum_{u,v \in V} \left[A_{uv} - \frac{k_v^{out} k_u^{in}}{|E|} \right] \delta(c_v, c_u).$$

Since we're here, why stopping at directed unweighted graphs? Let's add weights! Say that the (u, v) edge has weight w_{uv} , and that w_u^{out} is the sum of all edge weights originating from u (with w_u^{in} defined similarly for the opposite direction). Then:

¹¹ Bowen Yan and Steve Gregory. Detecting communities in networks by merging cliques. In *2009 IEEE International Conference on Intelligent Computing and Intelligent Systems*, volume 1, pages 832–836. IEEE, 2009

¹² Philipp Schuetz and Amedeo Caflisch. Efficient modularity optimization by multistep greedy algorithm and vertex mover refinement. *Physical Review E*, 77(4):046112, 2008

¹³ Roger Guimera and Luis A Nunes Amaral. Functional cartography of complex metabolic networks. *nature*, 433(7028):895, 2005

¹⁴ Jörg Reichardt and Stefan Bornholdt. Statistical mechanics of community detection. *Physical Review E*, 74(1):016110, 2006

¹⁵ Vincent A Traag and Jeroen Bruggeman. Community detection in networks with positive and negative links. *Physical Review E*, 80(3):036115, 2009

¹⁶ Alex Arenas, Alberto Fernandez, and Sergio Gomez. Analysis of the structure of complex networks at different resolution levels. *New journal of physics*, 10(5):053039, 2008b

¹⁷ Paul Expert, Tim S Evans, Vincent D Blondel, and Renaud Lambiotte. Uncovering space-independent communities in spatial networks. *Proceedings of the National Academy of Sciences*, 108(19):7663–7668, 2011

¹⁸ Elizabeth A Leicht and Mark EJ Newman. Community structure in directed networks. *Physical review letters*, 100(11):118703, 2008

$$M = \frac{1}{2|E|} \sum_{u,v \in V} \left[w_{uv} - \frac{w_v^{out} w_u^{in}}{\sum_{u,v \in V} w_{uv}} \right] \delta(c_v, c_u).$$

Note that this simple move makes optimizing modularity a tad more complicated – you should check out the original paper to see why.

This is all well and good, since there aren't competing definitions of directed/weighted modularity. What's that? I'm being told there are. Oh boy. For instance, an alternative is to look at a directed network as if it were a bipartite network¹⁹, where each node v can be seen as two nodes v^{in} and v^{out} .

It has also been pointed out that, while this generalized modularity gives out different results than the standard modularity, it actually doesn't really distinguish the $u \rightarrow v$ and $v \rightarrow u$ cases very well²⁰. In this case, the proposed solution is to use the PageRank of nodes u and v as an expectation of their connection strength. And, once you do that, you open the floodgates of hell, as any directed measure can be now used to determine your expectation, generating hundreds of different modularity versions, each with its own community definition.

Known Issues

But the issues raised so far are only child's play. Let's take a look at the *real* problematic stuff when it comes to modularity. There are three main grievances with modularity. The first is that random fluctuations in the graph structure and/or in your partition can make your modularity increase²¹. However, I already mentioned that modularity can be extended to take care of statistical significance.

A harder beast to tame is the infamous resolution limit of modularity. To put it bluntly, modularity has a preferred community size, relative to the size of the graph. This means that a partition that a human would consider the natural partition of the network could be rejected by modularity maximization as it is not at the preferred resolution²². Empirically, it has been shown that modularity maximization approaches tend to find $\sqrt{|E|}$ communities in the network – a number of communities that seems to be common for many other partitioning algorithms²³.

For instance, consider the network and partition in Figure 36.5 (top). Modularity is positive, thus this is a good partition. However, the partition to the bottom of Figure 36.5 is better, even if a human would probably disagree. This is because, when we have small communities relative to the number of edges of the network,

¹⁹ Roger Guimerà, Marta Sales-Pardo, and Luís A Nunes Amaral. Module identification in bipartite and directed networks. *Physical Review E*, 76(3):036102, 2007

²⁰ Youngdo Kim, Seung-Woo Son, and Hawoong Jeong. Finding communities in directed networks. *Physical Review E*, 81(1):016103, 2010

²¹ Roger Guimera, Marta Sales-Pardo, and Luís A Nunes Amaral. Modularity from fluctuations in random graphs and complex networks. *Physical Review E*, 70(2):025101, 2004

²² Santo Fortunato and Marc Barthelemy. Resolution limit in community detection. *Proceedings of the National Academy of Sciences*, 104(1):36–41, 2007

²³ Amir Ghasemian, Homa Hosseini-mardi, and Aaron Clauset. Evaluating overfit and underfit in models of network community structure. *TKDE*, 2019

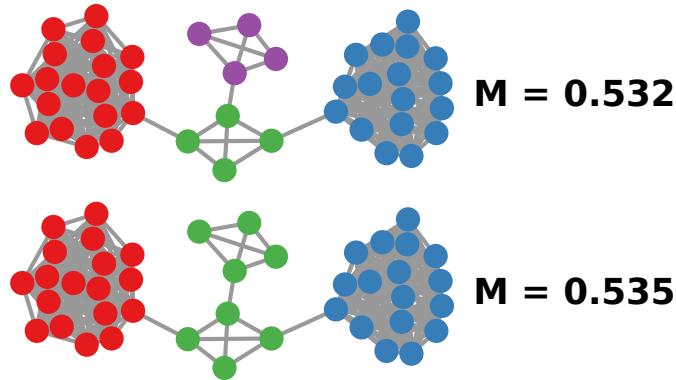


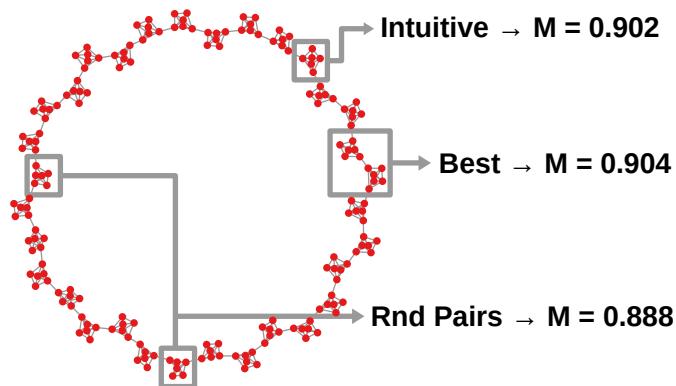
Figure 36.5: The resolution limit of modularity. For the same network, I propose two different partitions in communities, using the node color.

for modularity it is better to merge them, even if they are clearly and intuitively distinct. This is the resolution limit of modularity: it accepts partitions only of a comparable size with the size of the network.

Mathematically speaking, it's not too hard to extend modularity so that it can work at multiple resolutions. The common strategy is to add a resolution parameter^{24,25}. This can be interpreted as adding a bunch of self-loops to each node, such that the number of edges in a small community can still be considerable, due to the presence of such self loops. However, now you're not only optimizing modularity, you also have to search for the optimal value of this parameter. Uff.

This can get really tricky. Consider Figure 36.6 as another example. Here we have a ring of cliques, a classical caveman model. How would you partition this network? It seems natural to just have one community per clique. *Silly human, modularity says, the best partition is instead merging two neighboring cliques.* You look at modularity, puzzled by this sentence. But she is not done: *however, we could also put random clique pairs in the same community, even if they're not adjacent.* That's a good partition as well.

Go home, modularity, you now say, you're drunk.



²⁴ Alex Arenas, Alberto Fernandez, and Sergio Gomez. Analysis of the structure of complex networks at different resolution levels. *New journal of physics*, 10(5):053039, 2008b

²⁵ Jianbin Huang, Heli Sun, Yaguang Liu, Qinbao Song, and Tim Weninger. Towards online multiresolution community detection in large-scale networks. *PloS one*, 6(8):e23829, 2011a

Figure 36.6: A ring of cliques, showing another side of the resolution limit problem of modularity.

Joking aside, this is related to another well known problem of modularity: the field of view limit²⁶. Modularity cannot “see” long range communities. If your communities are very large and span across multiple degrees of separation, modularity will overpartition them. This means that, if a proper community has nodes whose shortest connecting path is more than a few edges long, you might end up splitting them in different groups. This field of view limit is shared with other community discovery approaches using Markov processes (random walks). Even vanilla Infomap tends to overpartition such communities. In fact, in the paper I cited in this paragraph, the authors show how you could interpret the modularity formula as a one-step Markov process.

All of this is to say that optimizing modularity is NP-hard and the heuristics have a hard time finding the best partitions because the space of possible solutions is crowded by high values that look very different. This is the third grievance with modularity: the degeneracy of good solutions^{27,28}. It’s easy to get stuck in local maxima even when the partition you’re returning makes no sense. A classical solution is to summon consensus clustering²⁹ – we saw it in Section 35.6. Hopefully, strategies based on perturbation will converge to different local maxima, and the mistakes will cancel each other out, leading you to a global maximum.

36.2 Other Topological Measures

Given these issues, it’s no wonder that researchers have looked elsewhere for alternative quality measures. The ones I’m mentioning are by no means perfect, and they have been scrutinized less than modularity, so the absence of known issues should be taken with a grain of salt.

The likelihood measure introduced for SBMs is an obvious candidate as modularity alternative, and I wrote about it in details in Section 35.1. The obvious downside here is that it needs your community partition to be a “generative” one: it has to give you a model of connection probabilities among nodes. Without that, you cannot estimate how likely your model is to generate the observed network.

There is also a quality measure lurking behind Infomap (Section 35.2). The “code length” Infomap is trying to minimize is the number of bits you need to encode the random walks using your partition. There is, in principle, no issue in generating a community partition using something else than Infomap, and then testing it with code length. Thus that is also a valid quality measure. You have to be careful, because the standard code length is not normalized. Two networks with different sizes in number of nodes will have a different

²⁶ Michael T Schaub, Jean-Charles Delvenne, Sophia N Yaliraki, and Mauricio Barahona. Markov dynamics as a zooming lens for multiscale community detection: non clique-like communities and the field-of-view limit. *PLoS one*, 7(2):e32210, 2012a

²⁷ Benjamin H Good, Yves-Alexandre De Montjoye, and Aaron Clauset. Performance of modularity maximization in practical contexts. *Physical Review E*, 81(4):046106, 2010

²⁸ Andrea Lancichinetti and Santo Fortunato. Limits of modularity maximization in community detection. *Physical review E*, 84(6):066122, 2011

²⁹ Pan Zhang and Christopher Moore. Scalable detection of statistically significant communities and hierarchies, using message passing for modularity. *Proceedings of the National Academy of Sciences*, 111(51):18144–18149, 2014

expected code lengths. Thus a better partition in a larger network could have a worse (higher) code length than a bad partition in a small network.

A direct evolution of modularity which aims at being a more general version of it is stability^{30,31,32}. In modularity, you see the graph as static. Nodes u and v contribute to modularity only insofar as they are directly connected or not. In stability, you see your graph as a flow. You take into account the amount of time it would take to reach u from v and vice versa. In practice, modularity is equivalent to stability when you only look at immediate diffusion.

There's a battery of other quality measures, conveniently grouped in a review paper³³. I'm going to cover a few here. In all cases, I use a generic $f(C)$ to refer to the function taking the community C as an input and returning its evaluation of the quality of that community.

Conductance

The idea behind conductance is that communities should not be conductive: whatever falls into – or originates from – them should have a hard time getting out. In practice, this translates in comparing the volume of edges pointing outside the cluster^{34,35,36,37}. Here we assume that $C \subseteq V$ is a set of nodes grouped in a community. Mathematically speaking, let's define two sets of edges. The first set of edges, E_C is the number of edges fully inside the community C . That means $E_C = \{(u, v) : u \in C, v \in C\}$. The second set of edges is the boundary of C : the edges attached to one node in C and one node outside C : $E_{B,C} = \{(u, v) : u \in C, v \notin C\}$. We can now define conductance as:

$$f(C) = \frac{|E_{B,C}|}{2|E_C| + |E_{B,C}|}.$$

Note that we want to minimize this function, namely we want to find the partition of G such that the average conductance across all communities is minimal. Figure 36.7 shows two examples of communities with different levels of conductance.

Figure 36.7(a) (in red) is a low conductance community. It is a clique of seven nodes, thus we know that $|E_C| = 7 \times 6/2 = 21$. It has four edges in its boundary ($|E_{B,C}| = 4$), which gives us $4/((2 \times 21) + 4) \sim 0.087$. Figure 36.7(b) (also in red), on the other hand, has a higher conductance. Being a clique of five nodes, $|E_C| = 5 \times 4/2 = 10$. From the figure we see that $|E_{B,C}| = 7$, giving us a conductance of $7/((2 \times 10) + 7) \sim 0.26$.

Note how conductance doesn't care too much about internal density, as one would expect from the classical definition. It cares, instead, about external sparsity: making sure that the community

³⁰ Renaud Lambiotte, J-C Delvenne, and Mauricio Barahona. Laplacian dynamics and multiscale modular structure in networks. *arXiv preprint arXiv:0812.1770*, 2008

³¹ J-C Delvenne, Sophia N Yaliraki, and Mauricio Barahona. Stability of graph communities across time scales. *Proceedings of the national academy of sciences*, 107(29):12755–12760, 2010

³² Jean-Charles Delvenne, Michael T Schaub, Sophia N Yaliraki, and Mauricio Barahona. The stability of a graph partition: A dynamics-based framework for community detection. In *Dynamics On and Of Complex Networks, Volume 2*, pages 221–242. Springer, 2013

³³ Jure Leskovec, Kevin J Lang, and Michael Mahoney. Empirical comparison of algorithms for network community detection. In *WWW*, pages 631–640. ACM, 2010b

³⁴ Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *Departmental Papers (CIS)*, page 107, 2000

³⁵ Ravi Kannan, Santosh Vempala, and Adrian Vetta. On clusterings: Good, bad and spectral. *Journal of the ACM (JACM)*, 51(3):497–515, 2004

³⁶ Jure Leskovec, Kevin J Lang, Anirban Dasgupta, and Michael W Mahoney. Statistical properties of community structure in large social and information networks. In *Proceedings of the 17th international conference on World Wide Web*, pages 695–704. ACM, 2008

³⁷ Jure Leskovec, Kevin J Lang, Anirban Dasgupta, and Michael W Mahoney. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. *Internet Mathematics*, 6(1):29–123, 2009

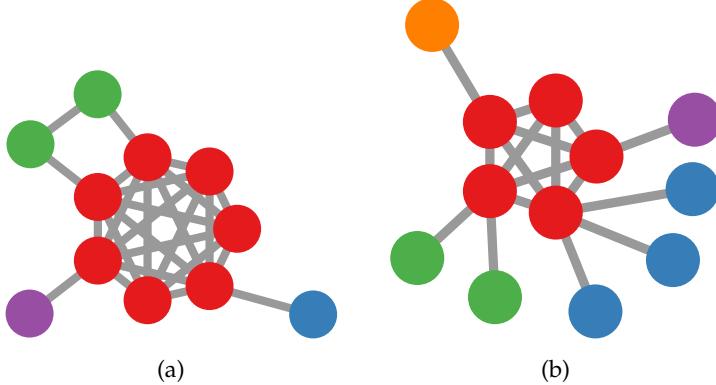


Figure 36.7: Two examples of communities at different conductance levels. I represent the community as the node color. In the text, I focus on the red community.

is as isolated as possible from the rest of the network. Here, both communities are cliques, the densest possible structure. But, since the one in Figure 36.7(b) also has a lot of connections to the rest of the network, the resulting conductance is almost three times higher than the value we get from the community in Figure 36.7(a).

Finally, be aware that you cannot build a community discovery algorithm that simply minimizes conductance – the same way you'd try to maximize modularity. That is because there's a trivial community with zero conductance: the one including all nodes in your network.

Internal density

The other side of the conductance coin is the internal density measure. This is exactly what you'd think it is: how many edges are inside the community over the total possible number of edges the community could host³⁸. Borrowing E_C from the previous section:

$$f(C) = \frac{|E_C|}{|C|(|C| - 1)/2}.$$

So you can see that, in this case, both communities in Figure 36.7 have an internal density of 1, since they're cliques. Thus, internal density is unable to distinguish between them, which we would like since community Figure 36.7(b) is clearly “weaker”, given its high number of external connections.

You can appreciate the paradoxical result of internal density by looking at Figure 36.8. Here, one might be tempted to merge the red and blue communities, since their nodes are so densely connected to each other and to not much else. Yet, the red nodes are a 5-clique and the blue nodes are a 4-clique, while the red and blue nodes are not a 9-clique. Thus, the best way to maximize internal density is to split these clearly very related nodes.

³⁸ Filippo Radicchi, Claudio Castellano, Federico Cecconi, Vittorio Loreto, and Domenico Parisi. Defining and identifying communities in networks. *Proceedings of the National Academy of Sciences*, 101(9):2658–2663, 2004

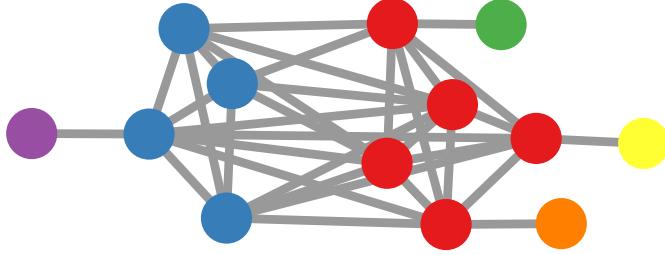


Figure 36.8: The best internal density partition of this core community. I encode the node's community with its color.

Neither conductance nor internal density fully capture the classical definition of community discovery I provided in the previous chapter. The definition wants communities to be both internally dense and externally sparse. Each of the two measures only satisfies one of the two requirements. Thus, if you're using either of them to evaluate your communities, you're practically having a different definition of what a community *is*.

Just like conductance, don't try to blindly maximize internal density, as you're only going to find cliques in your networks.

Cut

Originally, we define the cut ratio as the fraction of all possible edges leaving the community. The worst case scenario is when every node in C has a link to a node not in C . There are $|C|$ nodes in C and $(|V| - |C|)$ nodes outside C , so there can be $|C|(|V| - |C|)$ such links. Thus:

$$f(C) = \frac{|E_{C,B}|}{|C|(|V| - |C|)}.$$

This is usually what gets minimized when solving the mincut problem (Section 11.5). Again, this is a measure easy to game. That is why we often modify it to be a “normalized” mincut:

$$f(C) = \frac{|E_{B,C}|}{2|E_C| + |E_{B,C}|} + \frac{|E_{B,C}|}{2(|E| - |E_C|) + |E_{B,C}|}.$$

The most attentive readers already noticed that the first term in this equation is conductance. The second term is also a conductance of sorts. If the first term is the conductance from the community to the rest of the network, the second term is the conductance from the rest of the network to the community. The two are not the same, because the number of edges in C is $|E_C|$, while the number of edges outside C is $|E| - |E_C|$.

Out Degree Fraction

The out degree fraction (ODF), as the name suggests, looks at the share of edges pointing outside the cluster. It follows a strategy similar to conductance. The difference lies in a normalizing factor. While conductance normalizes with the total number of edges in the community, in the out degree fraction you normalize node by node. In the original paper³⁹, the authors present a few variants of the same philosophy.

In the Maximum-ODF, you simply pick the node which has the highest number of edges pointing outside the community (relative to its degree) as your yardstick:

$$f(C) = \max_{u \in C} \frac{|(u, v) : v \notin C|}{k_u}.$$

The idea here is that, in a good community partition, there shouldn't be *any* node with a significant number of edges pointing outside the community. We can tolerate if a node has a large *number* of edges pointing out, only if the node is a gigantic hub with a humongous degree k_u .

Requiring that there is absolutely no node with a large out degree fraction might be a bit too much. So we also have a relaxed Average-ODF:

$$f(C) = \frac{1}{|C|} \sum_{u \in C} \frac{|(u, v) : v \notin C|}{k_u}.$$

In this case, we're ok if, on average, nodes tend not to connect relatively much to neighbors outside the cluster. If there is one node doing so, the presence of many other nodes without external connections will overwhelm it.

Finally, Flake et al. in their paper propose a further variant of the same idea:

$$f(C) = \frac{1}{|C|} |\{u : u \in C, |(u, v) : v \in C| < k_u/2\}|.$$

For each node u in C , we count the number of edges pointing outside the cluster. If it's more than half of its edges, we mark the node as "bad", because it connects more outside the community than inside. A node shouldn't do that! The measure tells you the share of bad nodes in C , which is something you want to minimize.

Figure 36.9 shows an example community, which we can use to understand the difference between the various ODF variants. In the Maximum-ODF, we're looking for the node with the relative highest out degree. That is node 1 as its degree is just three, and two of those edges point outside the community. Thus, the Maximum-ODF

³⁹ Gary William Flake, Steve Lawrence, C Lee Giles, et al. Efficient identification of web communities. In *KDD*, volume 2000, pages 150–160, 2000

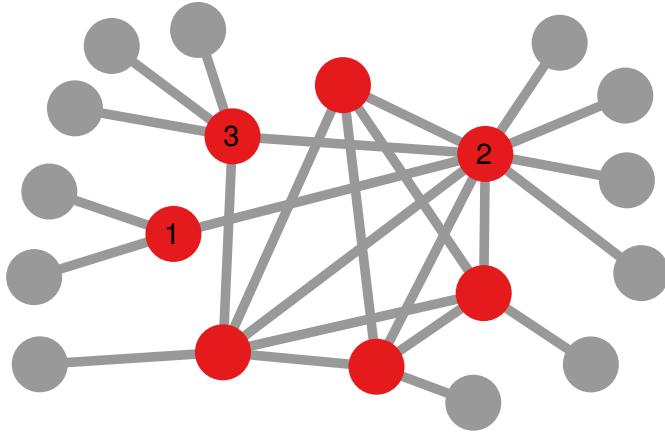


Figure 36.9: An example of community (in red).

is $f(C) = 2/3$. Both nodes 2 and 3 have a higher out-community degree, but they also have a higher degree and thus they don't count at all for the community quality. You can see how Maximum-ODF is a blunt tool which disregards lots of information.

For Average-ODF we have (clockwise starting from node 1):

$$f(C) = (2/3 + 3/5 + 0 + 4/10 + 1/5 + 1/5 + 1/6)/7 \sim 0.319.$$

This is awfully close, but not quite, conductance – which is $12/(2 \times (13) + 12) \sim 0.316$. Finally, we only have two nodes with more links going outside the community than inside: these are nodes 1 and 3.

Thus, Flake-ODF is $f(C) = 2/7$.

36.3 Link Prediction

If you have a temporal network, you gain a new way to test the quality of your communities. After all, communities are dense areas in the network, thus they tell you something about where you expect to find new links. In a strong assortative community partition, there are more links between nodes in the same community than between nodes in different communities. Otherwise, your communities would be weak – or there won't be communities at all⁴⁰.

Thus you can use your communities to have a prior about where the new links will appear in your dynamic network. This sounds familiar because it is: it is literally the definition of the link prediction problem (Part VII). In this approach of community evaluation, you use the community partition as your input. You use it to estimate the likelihood of connection between any pair of nodes in the network, and then you can design the experiment (Chapter 25) and use any link prediction quality measure as your criterion to decide which community partition is better. The higher your AUC, the better looking your ROC curve, the better your partition is.

⁴⁰ Or so the classical definition of community says. I already started tearing it apart, and I'll continue doing so, but in this specific test you base your assumption on this classical definition. If you have a different definition of community, don't use this test.

The classical way to create a $score(u, v)$ is having a simple binary classifier: 1 if u and v are in the same community, 0 otherwise. This is a bit clunky, so you usually want to add a bit of information: how well embedded are the nodes in the network? This also works in the case of overlapping community discovery (Chapter 38), when nodes can be part of multiple communities. In that case, $score(u, v)$ can be the number of communities they have in common. This seemingly innocuous operation has some rather interesting repercussions, which we will see dubbed as the “overlap paradox” in Section 38.7.

A method that works naturally well to be evaluated via link prediction is finding communities via SBMs. This is because, in the general SBM, every pair of nodes receives a p_{in} or a p_{out} connection probability given the planted partition. Only in the simplest SBM techniques these values are the same for every pair of nodes. In more sophisticated approaches they are personalized for each node pair, and thus serve as a natural $score(u, v)$ function.

Remember that, when looking at communities, you could have a disassortative community partition, where nodes tend to connect to other nodes *outside* their own community. You can still use this approach, now penalizing in your $score(u, v)$ function nodes part of the same community. You could even do more fun stuff, by creating a community-community similarity score, in which nodes that are in more interconnected communities receive a higher $score(u, v)$ value.

You know from Chapter 25 that you can still evaluate your link prediction also in presence of static network data, via k-fold cross validation. This would allow you to evaluate your communities as input for link prediction even lacking a temporal network, making this a more general evaluation tool.

36.4 Normalized Mutual Information

Your network might not be temporal, but you could have additional information about the nodes, besides to which other nodes they connect (Section 7.5). In this context, node attributes are usually referred to as “node metadata”. There is a widespread assumption in community discovery: if you have good node metadata, some of them have information about the true communities of the network. Nodes with similar values, following the homophily assumption (Chapter 30), will tend to connect to each other. Therefore there should be some sort of agreement between the community partition of the network and the node metadata⁴¹.

For instance, a classical paper⁴² analyzed a network whose nodes were cellphones, connected together if they made a significant number of calls to each other. The network showed three well-separated

⁴¹ Jaewon Yang and Jure Leskovec. Defining and evaluating network communities based on ground-truth. *Knowledge and Information Systems*, 42(1): 181–213, 2015

⁴² Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment*, 2008 (10):P10008, 2008

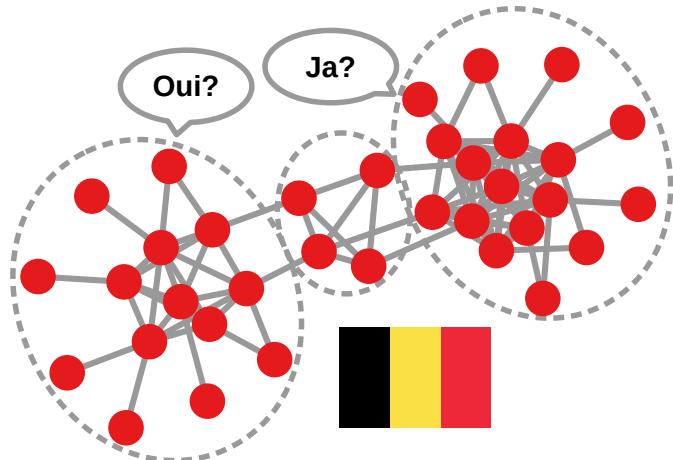


Figure 36.10: A simplification of the Belgian cellphone call graph, highlighting three communities (with the dashed gray outline).

communities. Figure 36.10 shows an extreme simplification of that (very large) graph.

Why was that the case? Why were there gigantic communities? It all becomes clear when I tell you that the country they studied was Belgium, where roughly half of the population is French-speaking and the other half Dutch-speaking, and so they do not call each other. The intersection in the middle is the capital Brussels, where the two populations have to interact. Knowing which language you speak should have almost a one-to-one correspondence with the network community in this case.

How would you calculate such agreement? We can re-use a concept we encountered early on: mutual information (Section 3.5). To recap briefly: you can consider each node in the network as being an entry in two vectors. In the first vector, the node is associated with its metadata: the language the person speaks or whether she lives in Brussels. In the second vector, the node is associated to its community.

Mutual information tells you the number of bits of information you gather about one vector by knowing the other vector. Figure 36.11 (a reprisal of Figure 3.14) should help you understanding what mutual information means: having a set of rules that allow you to infer the values in one vector by knowing the other with better-than-chance odds. The Rand Index^{43,44} provides a similar measure, by counting the number of node pairs agreeing between the community partition and the ground truth, without the information theoretic properties of mutual information.

In Section 23.7 we used mutual information for link prediction, meaning that we didn't care much about comparing different networks, as everything happened in the same network. However, when evaluating community partitions, you need a standardized yardstick

⁴³ William M Rand. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical association*, 66(336):846–850, 1971

⁴⁴ Lawrence Hubert and Phipps Arabie. Comparing partitions. *Journal of classification*, 2(1):193–218, 1985

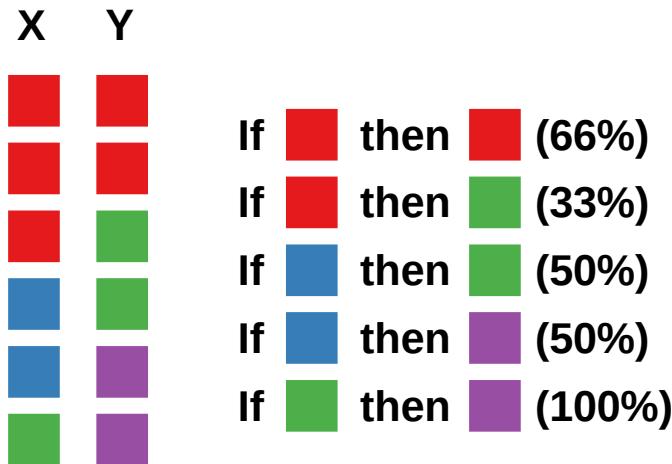


Figure 36.11: An illustration of what mutual information means for two vectors. Vector y has equal occurrences for its values (there is one third probability of any colored square). However, if we know the value of x we can usually infer the corresponding y value with a higher-than-chance confidence.

to know whether a network has communities more tightly knit than another – or if it has communities at all! But mutual information is dependent on the amount of bits you need to encode the vectors in the first place. A longer vector needs more bits to be encoded. Thus, the same value of mutual information can mean different things when you have 100 nodes or 100,000.

That is why we often use a *normalized* mutual information (NMI). This is a simple normalization that forces mutual information to take a value between zero and one. This is generally achieved by dividing mutual information by some combination of the entropy of the two vectors (the community partition and the node metadata)^{45,46}.

While normalizing mutual information so that it's comparable across networks is nice, that is not the full story. Remember that our end here is knowing whether there is a relationship between the communities we found and the node attributes. The problem of mutual information is that it is always non-zero. This means that there will be always a little mutual information between two vectors, even if they are both completely random!



Consider the vectors in Figure 36.12. I generated them by extracting ten random elements, with three possible values. This is done uniformly at random and with independent draws – pinky promise! Yet, if you calculate their NMI values, you're going to obtain around 0.09: a non-zero mutual information from vectors that literally have nothing to do with each other. This is not good.

That is why researchers developed a new normalization for mutual information: Adjusted mutual information (AMI)^{47,48}. In this

⁴⁵ Alexander Strehl and Joydeep Ghosh. Cluster ensembles—a knowledge reuse framework for combining multiple partitions. *Journal of machine learning research*, 3(Dec):583–617, 2002

⁴⁶ William H Press, Saul A Teukolsky, William T Vetterling, and Brian P Flannery. *Numerical recipes 3rd edition: The art of scientific computing*. Cambridge university press, 2007

Figure 36.12: Two random vectors with a positive NMI even if generated completely independently from one another.

⁴⁷ Marina Meilă. Comparing clusterings—an information based distance. *Journal of multivariate analysis*, 98(5):873–895, 2007

⁴⁸ Nguyen Xuan Vinh, Julien Epps, and James Bailey. Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance. *J. Mach. Learn. Res.*, 11(Oct):2837–2854, 2010

case, we subtract from mutual information the amount of bits we would expect to obtain about a vector by pure chance. We can do the same thing for the Rand index I mentioned before, generating the Adjusted Rand Index⁴⁹ (ARI). In this, AMI and ARI are similar to modularity: you're comparing the observed value with the one you'd get from some sort of null model. AMI and ARI are defined to be equal to zero when you get nothing more than you'd expect by just tossing coins. At this point, any positive value starts getting interesting. These indexes can be negative, AMI is for the two vectors in Figure 36.12. A negative AMI means that your clustering isn't good.

It can also mean another thing. You see, so far I grounded this section on a key assumption: that node metadata go hand in hand with the network structure. That... is not always the case. Researchers have seen how much the two notions can diverge⁵⁰. Node metadata and structural network communities are rarely the same thing. Nodes can share attribute values and not being connected to each other due to a variety of reasons.

In fact, the assumption that communities and node metadata go hand in hand rests on shaky ground. It seems to give some sort of importance and status to the node metadata because it calls it "meta" data. But, at the end of the day, in real observed networks metadata is just data. Real metadata is like the planted community in an LFR model (Section 18.2), but when you do data gathering there's no such a thing as metadata: what you find is often – if not always – incomplete, irrelevant, or wrong to a certain degree.

We can call this a "data" problem. Which is yet another issue with the classical definition of communities. Wanting to find a structural way to group nodes with the same attributes – especially when we don't know their values and we want to infer them – is a totally valid aim on which to base your community definition. It's just that it doesn't correlate well with the notion of communities made by densely connected nodes. In this scenario, we might even have disconnected communities, made by multiple components without paths leading from one node in the community to another node in the same community. This is absolutely verboten in the classical view of community discovery.

The assumption that we can infer the "real" communities from node attributes rests on data availability: we have some metadata and we assume that the network follows it. But the network could be wired following multiple different attributes and the interactions between them. Figure 36.13 shows a simplified example of that. The communities found by node colors are "good" to approximate one attribute, but they are terrible for the classical notion of community.

⁴⁹ Douglas Steinley. Properties of the hubert-arable adjusted rand index. *Psychological methods*, 9(3):386, 2004

⁵⁰ Darko Hric, Richard K Darst, and Santo Fortunato. Community detection in networks: Structural communities versus ground truth. *Physical Review E*, 90(6):062805, 2014

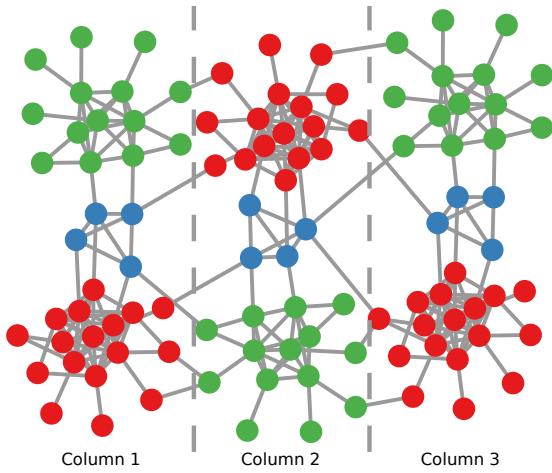


Figure 36.13: A network with a community structure made by conflicting attributes. One node attribute is the color, while the other is its horizontal positioning. Nodes connect most likely with nodes of the same color *and* in the same “column”.

There's another problem with ground truth and community discovery, a more theoretical one. As many other facets of life, in community discovery there is no free lunch⁵¹. This means that community discovery is a large problem with many different network types and valid community definitions. There is no single algorithm who is going to work reliably better than average in all these scenarios.

We are going to see yet more ways in which the classical definition of communities break. But this is a good moment to have a brief pause and collect our thoughts. The real definition of community depends on what the network represents: if you're looking at a social network some definitions of communities make sense, but if you're looking at an infrastructure network they do not. Communities depend on what you're looking for: whether you're trying to approximate a real world property or compress your network. And they also depend on what's your criterion of success: nobody says you cannot use modularity, or NMI, or anything else, as long as it is a motivated choice.

I want you to learn a lesson, my dear reader. You didn't have to go and look for a definition of community: the real definition was inside you all along.

36.5 Summary

1. The most common function used to evaluate community partitions is modularity. Modularity compares the number of edges inside the communities you detected with the expected number of edges in a configuration model which has, by definition, no communities.

⁵¹ Leto Peel, Daniel B Larremore, and Aaron Clauset. The ground truth about metadata and community detection in networks. *Science advances*, 3(5):e1602548, 2017

2. You can also use modularity for something more than evaluating the communities you found: it can be an optimization target. Your algorithm will operate on your communities until it cannot find any additional move that would increase modularity.
3. Standard modularity is defined for undirected and unweighted graphs. There are extensions of the measure to deal with directed and/or weighted graphs, however such extensions are not unique: there are multiple competing versions.
4. Modularity has been extensively studied and we know it has several issues. The main one is resolution limit, where communities have to be of similar size. There is also degeneracy: many partitions are close to optimal, even if they are very different from each other.
5. Many other quality functions have been defined. Conductance aims at minimizing edges flowing out of a community. Internal density aims at maximizing the edges inside the community.
6. One could use communities as the basis of link prediction, since nodes in the same community are expected to connect to each other. Thus a better partition is one that would be more accurate in predicting new links.
7. Normalized mutual information is another way to evaluate your partition when you have metadata about your nodes – if you assume that communities should be used to recover latent node attributes. Be aware, though, that not always nodes with similar attributes connect to each other.

36.6 Exercises

1. Detect communities of the network at <http://www.networkatlas.eu/exercises/36/1/data.txt> using the asynchronous and the semi-synchronous label propagation algorithms. Which one does return the highest modularity?
2. Find the communities of the network at <http://www.networkatlas.eu/exercises/36/2/data.txt> using label propagation and calculate the modularity. Then manually create a new partition by moving nodes 25, 26, 27, 31 into their own partition. Recalculate modularity for this new partition. Did this move increase modularity?
3. Repeat exercise 1, but now evaluate the difference in performance of the two community discovery algorithms by means of conductance, cut size, and normalized cut size.

4. Assume that <http://www.networkatlas.eu/exercises/36/4/nodes.txt> contains the “true” community partition of the nodes from the network at <http://www.networkatlas.eu/exercises/36/1/data.txt>. Determine which algorithm between the asynchronous and the semi-synchronous label propagation achieves higher Normalized Mutual Information with such gold standard.

Hierarchical Community Discovery

When talking about the issues of modularity as a quality measure for network partitions, we touched on an important subject which deserves to be explored more deeply. When doing community discovery, you might have a situation where the network can be divided in different ways and they're all valid partitions. For instance, in Figure 37.1, the obvious assortative partition (blue outlines) will divide scientists into their fields, and laymen into their own communities. However, it is also reasonable to enlarge the definition of a field and say that there is a scientific community, which incorporates all of its subfields (purple outline), and a non-scientific community, which incorporates all non-scientific subcommunities (green outline).

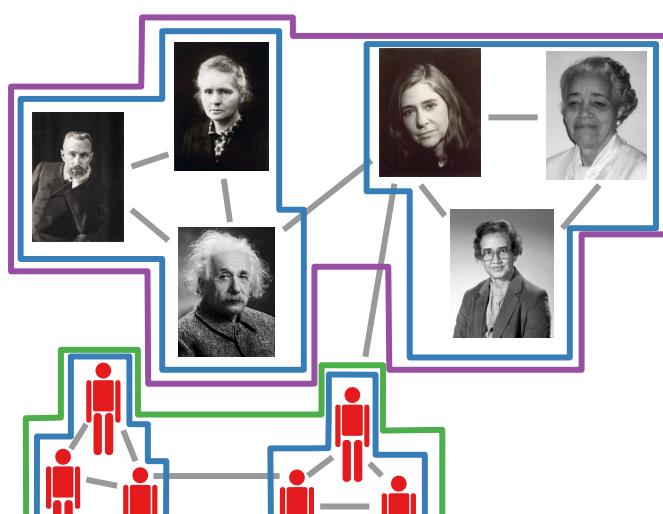


Figure 37.1: Two possible valid partitions of this network. Gray lines are the edges. The blue outlines identify strong, narrow communities. The purple line outlines the scientific community, opposed to the green outline: the laymen community.

To generalize the issue, once you find tightly knit communities, you might realize that some communities are more related to each other than others. And so there is a second level on the partition you can impose on the network. This means that, after finding communities of nodes, you want to find communities of communities. And

communities of communities of communities. And so on. This is the “hierarchical” community discovery problem: how to create a hierarchy of communities that best describes the structure of your network.

I’m going to present some general approaches to hierarchical community discovery. As usual, be aware that there are more than I can cover here^{1,2}. However, once you know them, it’s easy for you to see that you can redefine many standard community discovery algorithms to find hierarchical communities. For instance, the Infomap algorithm I described in the previous chapter has a natural hierarchical version³.

37.1 Recursive Approaches

You can transform any community discovery algorithm into a hierarchical community discovery algorithm by applying it recursively to coarsened views of your network. The easiest way to do so is by following this simple meta algorithm:

1. Apply your algorithm to G and find the optimal communities;
2. Condense your graph by collapsing all nodes belonging to community C into a meta-node C ;
3. Connect all C s with each other, according to how many edges there were between the nodes they include;
4. You now have a new graph G' , so you can go back to step 1.

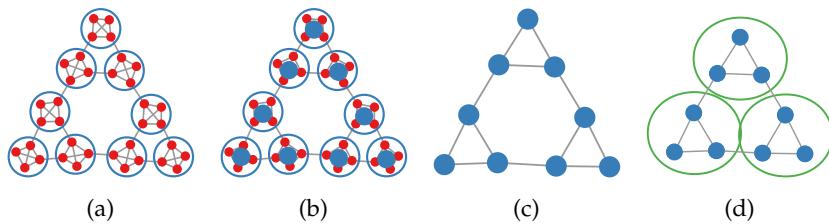


Figure 37.2 shows a graphical example of this meta algorithm. You can modify step 3 in case your algorithm was an overlapping algorithm, which allows communities to share nodes. In that case, you can count the number of shared nodes between the communities⁴, rather than the number of edges connecting them.

However, such approach is just a hack on top of non-hierarchical community discovery. In reality, we want to have a hierarchy-aware

¹ Spiros Papadimitriou, Jimeng Sun, Christos Faloutsos, and S Yu Philip. Hierarchical, parameter-free community discovery. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 170–187. Springer, 2008

² Jianbin Huang, Heli Sun, Jiawei Han, Hongbo Deng, Yizhou Sun, and Yaguang Liu. Shrink: a structural clustering algorithm for detecting hierarchical communities in networks. In *Proceedings of the 19th ACM international conference on Information and knowledge management*, pages 219–228. ACM, 2010

³ Martin Rosvall and Carl T Bergstrom. Multilevel compression of random walks on networks reveals hierarchical organization in large integrated systems. *PLoS one*, 6(4):e18209, 2011

Figure 37.2: (a) The optimal node partition. (b) Collapsing each community into a meta-node. (c) Connecting meta-nodes made by nodes which were originally connected to each other. (d) Finding the second level partition.

⁴ Michele Coscia, Giulio Rossetti, Fosca Giannotti, and Dino Pedreschi. Uncovering hierarchical and overlapping communities with a local-first approach. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 9(1):6, 2014

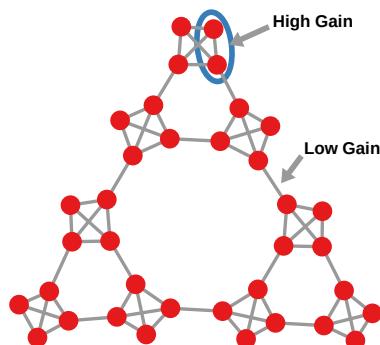
approach that was built with this feature in mind from the beginning, rather than adding it as an afterthought. There are two meta-approaches for baking in hierarchies in your community discovery: merging and splitting.

Merging

In the merging approach, you start from a condition where all your nodes are isolated in their own community and you create a criterion to merge communities. This is a bottom-up approach. It is similar to the meta-algorithm from earlier, but it's not really the same. Let's take a look at how it works, highlighting where the differences with the meta-algorithm are.

The template I'm using to describe this approach is the Louvain algorithm⁵. This is one of the many heuristics used to recursively merge communities with the aim of maximizing modularity^{6,7}, which happens to be among the fastest and most popular.

The Louvain algorithm starts with each node in its own community. It calculates, for each edge, the modularity gain one would get if they were to merge the two nodes in the same community. Then it merges all edges with a positive modularity gain. Now we have a different network for which the expensive modularity gains need to be recomputed. However, this network is smaller, because of all the edge merges. You repeat the process until you have all nodes in the same community. Figure 37.3 shows an example of this process.



The Louvain algorithm is particularly smart and optimized to find the best merges by minimizing the amount of computation needed. Its first step is expensive, because for every edge you have to know what's the modularity gain of merging the nodes. However, once you start merging, it's fast because you only need to update the gains of the nodes directly connected to the new partition. Finally, there is no need to go all the way: we know that putting all nodes in the same community has modularity zero, so at some point there are no

⁵ Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment*, 2008(10):P10008, 2008

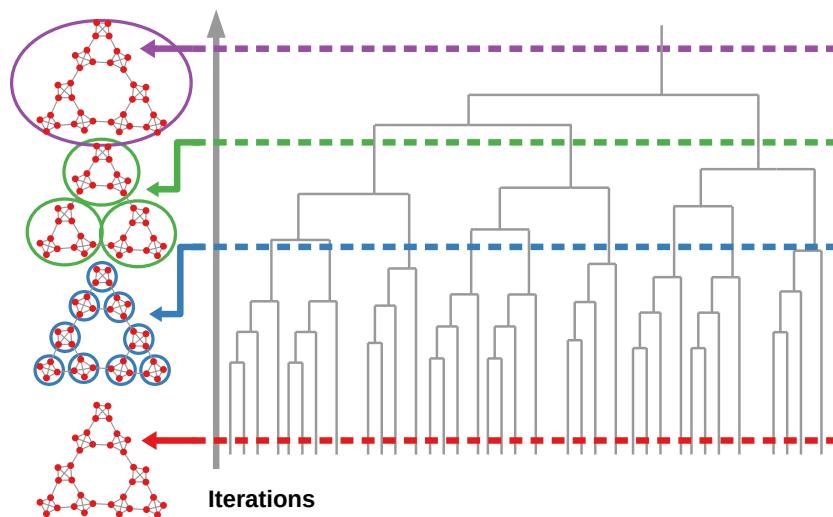
⁶ Marta Sales-Pardo, Roger Guimera, André A Moreira, and Luís A Nunes Amaral. Extracting the hierarchical organization of complex systems. *Proceedings of the National Academy of Sciences*, 104(39):15224–15229, 2007

⁷ Tiago P Peixoto. Hierarchical block structures and high-resolution model selection in large networks. *Physical Review X*, 4(1):011047, 2014c

Figure 37.3: An example of the first step of the Louvain algorithm. All in-clique edges (like the representative I highlight in blue) are merged, while all out-clique edges (like the representative I point to with a gray arrow) are ignored.

moves that can improve modularity, and we can stop. The algorithm inspiring it⁸, for instance, only made one merge per modularity gain and thus had to perform the expensive modularity gain calculation more often for larger networks.

What the algorithm does, in practice, is building a dendrogram of communities from the bottom up. Each iteration brings you further up in the hierarchy. We start with no partition: each node is in its own community. And then we progressively make larger and larger communities, until we have only one. Figure 37.4 shows an example of this approach. This is the crucial difference between the merging approach and what I discuss previously. In the meta algorithm, you don't perform all the merges, you make lots of them at once when you run your step 1 to find the initial communities.



⁸ Aaron Clauset, Mark EJ Newman, and Cristopher Moore. Finding community structure in very large networks. *Physical review E*, 70(6):066111, 2004

Figure 37.4: The dendrogram building from the bottom up typical of a “merging” approach in hierarchical community discovery.

Splitting

In the splitting approach, you do the opposite of what I described so far. You start with all nodes in the same community and you use a criterion to split it up in different communities. For instance by identifying edges to cut. This is a top-down approach.

Historically speaking, the first algorithm using this approach used edge betweenness as its criterion to split communities^{9,10}. That is not to say there aren't valid alternatives as your splitting criterion, including – but not limiting to – edge clustering¹¹ and information centrality¹². However, given its historical prominence, I'm going to allow the edge betweenness Girvan-Newman algorithm to have its place under the limelight.

The first step of the algorithm is to calculate the edge betweenness

⁹ Michelle Girvan and Mark EJ Newman. Community structure in social and biological networks. *Proceedings of the national academy of sciences*, 99(12):7821–7826, 2002

¹⁰ Mark EJ Newman and Michelle Girvan. Finding and evaluating community structure in networks. *Physical review E*, 69(2):026113, 2004

¹¹ Filippo Radicchi, Claudio Castellano, Federico Cecconi, Vittorio Loreto, and Domenico Parisi. Defining and identifying communities in networks. *Proceedings of the National Academy of Sciences*, 101(9):2658–2663, 2004

¹² Santo Fortunato, Vito Latora, and Massimo Marchiori. Method to find community structures based on information centrality. *Physical review E*, 70(5):056104, 2004

of each edge in the network, that is the normalized number of shortest paths passing through it (Section 14.2). The assumption is that edges between assortative communities will have a systematically higher edge betweenness value than edges inside the communities. Figure 37.5 shows an example. All edges inside the communities have a low value because there are many alternative paths you can take, since all nodes are connected to everybody else in the community. On the other hand, if you want to go from one node in one community to a node in another, there's only one edge you can use. As a result, its edge betweenness value skyrockets.

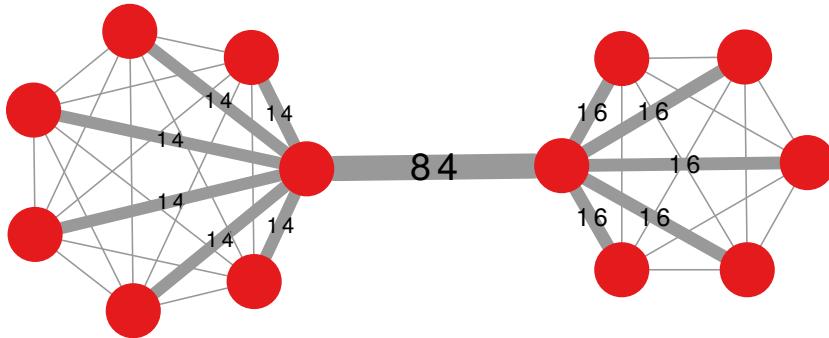


Figure 37.5: Two cliques connected by an edge. I label links with edge betweenness higher than one with the number of shortest paths passing through them.

The second step of the algorithm is to cut the edge with the highest edge betweenness. The final aim is to break the network down into multiple components. Each component of the network is a community.

Unfortunately, after each edge deletion you have to recalculate the betweennesses. Every time you alter the topology of the network you change the distribution of its shortest paths. This makes edge betweenness extremely computationally heavy. Calculating the edge betweenness for all edges takes an operation per node and per edge ($O(|V||E|)$) and you have to repeat this for every edge you delete, resulting in a crazy complexity of $O(|V||E|^2)$. You cannot apply this naive algorithm to anything but trivially small networks.

You can now see the parallels with the Louvain method I described earlier. The difference is that you are exploring the dendrogram of communities from the top down, rather than bottom up. Each iteration brings you further down in the hierarchy. At the very top you start with a network with a single connected component. As you delete edges, you find different connected components. As you continue, you end up with more and more. At the last iteration, each node is now isolated.

Differently from the Louvain algorithm, in the Girvan-Newman method you do not calculate modularity gains as you explore the

dendrogram. Thus, the algorithm will normally perform all the possible splits and returns you the full structure, rather than the cut that maximizes modularity. Thus you will have to calculate the modularity of each split yourself, something similar to what you see in Figure 37.6.

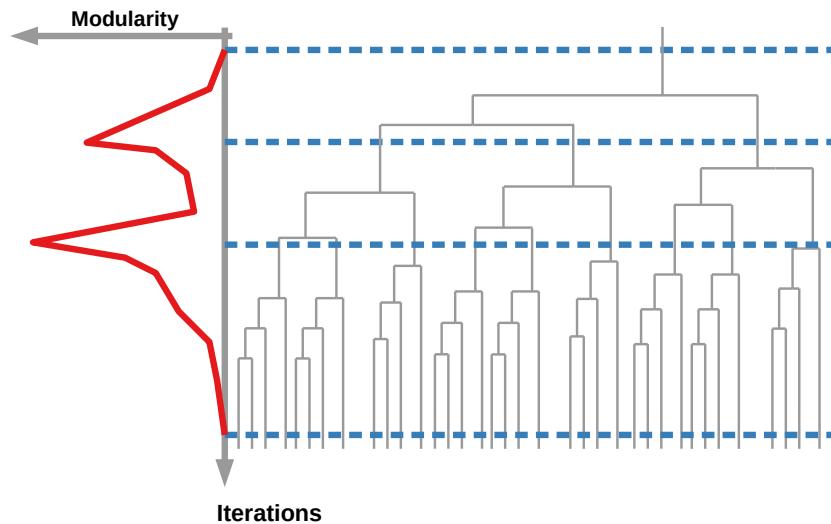


Figure 37.6: The dendrogram building from the top down typical of a “splitting” approach in hierarchical community discovery. The left panel shows the modularity values of each possible cut.

Higher modularities are better partitions, thus better cuts. The good cuts will appear as peaks in the modularity profile of the dendrogram. Thus they are the natural points for us to cut it and get the best partition. Multiple peaks are a clue of a hierarchical organization, because they identify good partitions with a very different number of communities.

The aforementioned edge clustering and information centrality variants use the same algorithm, changing the criterion to determine which edge to cut. In edge clustering the assumption is that edges with high clustering are embedded in communities, because all their neighboring nodes are connected to each other. Note that in this case we also modify the definition of local clustering coefficient so that it applies to edges rather than nodes. The guiding principle is to cut the edges with the lowest clustering first. This is computationally more efficient than using edge betweenness, because when you cut an edge you only change the clustering of the neighboring edges: in edge betweenness all values need to be recomputed.

The information centrality variant has no such benefit, because it simply uses a different definition of edge betweenness. Specifically it uses the edge current flow centrality¹³. It is still more computationally efficient, because it uses random walks rather than shortest paths, which will treat your CPU with more respect.

¹³ Ulrik Brandes and Daniel Fleischer. Centrality measures based on current flow. In *Annual symposium on theoretical aspects of computer science*, pages 533–544. Springer, 2005.

37.2 Hierarchical Random Graphs: Part 2

This section is a throwback to Section 23.5, where I introduced the usage of Hierarchical Random Graphs to solve the problem of link prediction. If you remember, the idea was to divide the graph into a hierarchical organization, under the assumption that nodes part of the same hierarchical branch are more likely to connect to each other. If we start from this assumption, then the natural consequence is also that these nodes *already are* densely connected. Thus they are proper communities!

Note that, however, HRG is more flexible than that: it can also uncover a disassortative community structure where nodes are *less* likely to connect to their community mates.

In Section 23.5 I simply said that “we create a hierarchical representation of the observed connections that fits the data,” which is a rather mysterious non-explanation of how we actually group nodes into communities. The way the algorithm works is by creating a dendrogram to fit the data. In the dendrogram, the $|V|$ leaf nodes are the nodes of the network. The other nodes of the dendrogram are so-called “internal nodes”, and we need $|V| - 1$ of them to properly build the full structure.

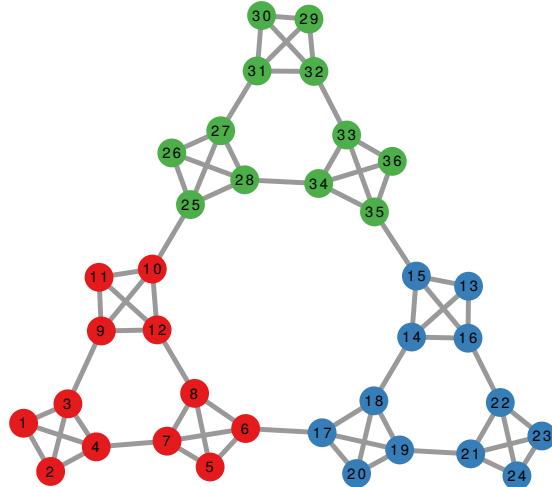
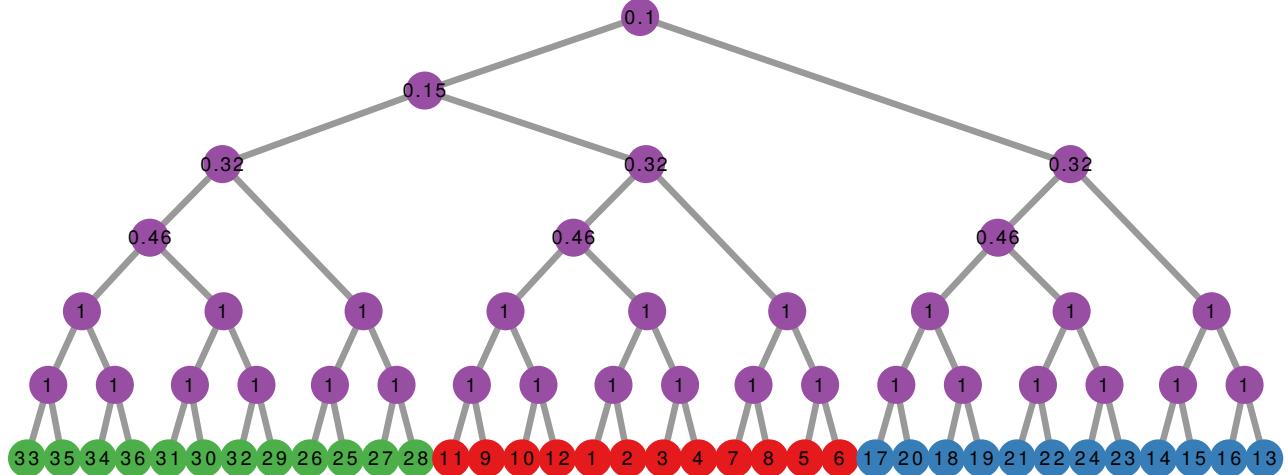


Figure 37.7: A graph with hierarchical communities (node color according to the community partition at one level of the hierarchy).

Figure 37.7 shows a network with a hierarchical community structure. Figure 37.8 is a possible HRG representation of Figure 37.7. Each internal node i has an associated probability p_i . This is the probability of connecting two nodes u and v that are in the branch attached to i . So our aim is to assign to all internal nodes the proper p_i probabilities and to shuffle the leaf nodes properly so that the dendrogram we end up with is the one most likely to describe the real data.



If this sounds familiar, you're not wrong. This looks like a special hierarchical version of stochastic blockmodels. You are basically assigning to each node pair a different p_i probability of connecting, depending on which is their most proximate common internal node ancestor i .

At this point, finding the dendrogram that most likely fits the data is just a choice of how you want to explore the space of all possible dendrograms. In the original paper, authors use a Markov chain Monte Carlo method, where each dendrogram is sampled proportionally to its likelihood value.

Note that the dendrogram in Figure 37.8 is not the only possible good description of Figure 37.7. For instance, I could have grouped nodes 1 and 3 together, at the lowest level, rather than 1 and 2. The resulting dendrogram would have been equally likely, and would generate an equally good hierarchical representation.

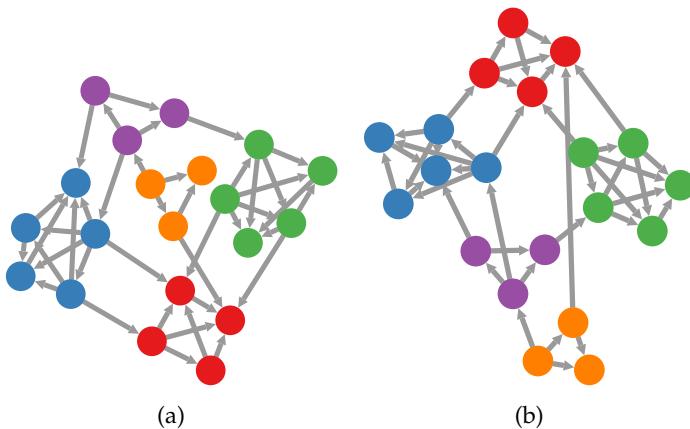
37.3 *Directed Communities*

HRGs allow us to transition to a slightly different way of interpreting hierarchical community discovery. HRGs still group communities into super communities, but takes a more statistical approach rather than applying a recursive merge/split operation. Pushing further this idea, we can get to the scenario of directed community discovery: to find communities in directed networks. Here we totally reject the idea that we should group communities into super communities and we simply establish that the communities we found organize in a hierarchy. Some communities are at the bottom because they prevalently point to other communities – here pointing to is interpreted as

Figure 37.8: A likely HRG representation of Figure 37.7. Purple nodes are internal nodes. I label them according to their p_i probability, which is the number of edges between nodes in the branches over all possible number of edges between them.

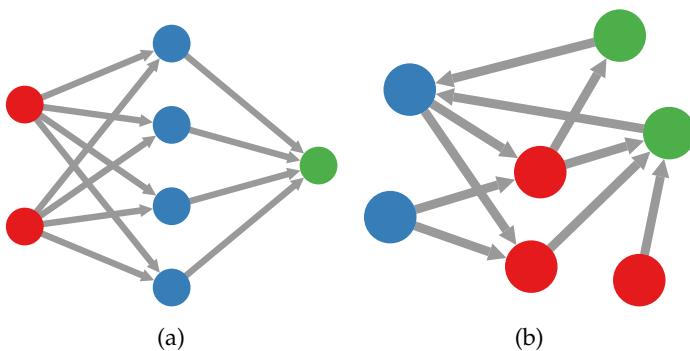
being below in the hierarchy.

One approach¹⁴ is relatively similar to HRGs: we're still doing a degree corrected SBM. However, as Figure 37.9, our objective is not to merge communities, but to figure out how to arrange them vertically so that most links point upwards. Note how in Figure 37.9(b) all edges going from one community to another always point upwards. The algorithm will find the vertical positioning of the communities most appropriate to guarantee this feature the best way it can. As a result, you can consider the top community as the one dominating the network, in a hierarchical fashion.



¹⁴ Tiago P Peixoto. Ordered community detection in directed networks. *Physical Review E*, 106(2):024305, 2022

Of course, this is not the only way to intend directed community discovery. I already showed you in Section 36.1 that you could use (one of the many competing versions of) directed modularity to do the job. In general, directed networks open the possibility of finding communities with a logic that is different from the usual one based on density. We might want to identify groups of nodes that follow a certain pattern¹⁵. For instance, Figure 37.10(a) shows that nodes can be grouped not because they connect to each other – in the classical density case – but because they are pointed by (or point to) the same nodes. In this case, red points to blue and blue points to green.



¹⁵ Darong Lai, Christine Nardini, and Hongtao Lu. Partitioning networks into communities by message passing. *Physical review E*, 83(1):016115, 2011

Figure 37.10: The node color identifies the node's community. (a) A pattern-based directed community partition. (b) A flow-based directed community partition.

This does not necessarily imply the hierarchical directionality we just saw. You can have a headless flow-based pattern like in Figure 37.10(b) that still satisfies the definition I just gave, without allowing you to find any hierarchical organization. In this case, blue points to red, red to green and green to blue, so there's no head to this cycling pattern.

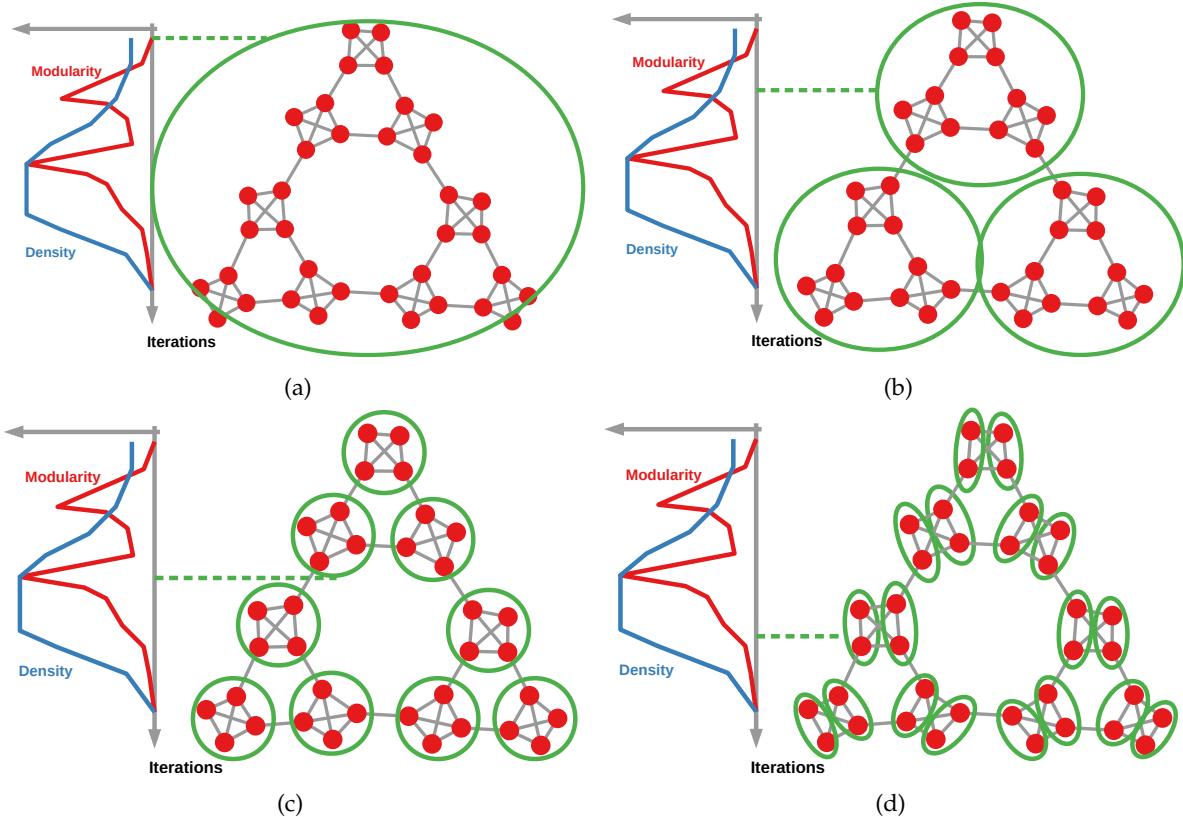
37.4 Density vs Hierarchy

Hierarchical community discovery poses some issues with our traditional community definition based on density. If there is a partition with maximum internal density in the network, then a partition at a different hierarchical level must have a lower density – by definition. Is that still a valid partition of the network? When you perform hierarchical community discovery you'd say yes, and that would be totally valid. There are valid analytic scenarios where you can divide up a social network at multiple levels. For instance the example I made at the beginning, dividing it up into a layman and scientific community, and then breaking down the scientific community in fields and subfields. But that is in direct contradiction with our classical community definition.

This is particularly tricky since even modularity, which should be defined as in direct correspondence with this density-based definition, actually disagrees with it. In other words, the density profile changes differently from the modularity profile. When we group everything in a community, there's some density even if modularity is zero (Figure 37.11(a)). At the top hierarchical level we have high modularity but low internal density (Figure 37.11(b)). At the best partition we have agreement (Figure 37.11(c)). But density is still high even with low modularity for a partition that puts together connected node pairs (Figure 37.11(d)).

37.5 Summary

1. You can find communities at different scales in a network. Meaning that there are communities of nodes, communities of communities, communities of communities of communities, and so on. The process to find such structures is hierarchical community discovery.
2. You can find hierarchical communities with either a top-down or a bottom-up approach. In the bottom-up or merging approach, you start with each node in its own community and then you recursively merge communities optimizing a quality function.



3. In the top-down or splitting approach, you start with the network encapsulated in a single community and you recursively split it following some guiding principle, e.g. removing the edges with the highest betweenness.
4. The third alternative is to model your network as a hierarchical system, and then find the hierarchical organization that is the most likely explanation of your observed data.
5. Not all communities at all levels of the hierarchy maximize the internal density and/or the external sparsity of your communities. Thus, even if totally valid, hierarchical communities defy our classical definition of communities based on edge density.

37.6 Exercises

1. Use the edge betweenness algorithm to find hierarchical communities in the network at <http://www.networkatlas.eu/exercises/37/1/data.txt>. Since the algorithm has high time complexity, perform only the first 10 splits. What is the split with the highest modularity?

Figure 37.11: The contrast between modularity and density at different cuts in the hierarchical community organization: (a) every node in the same community; (b) sub-optimal high-level partition; (c) optimal low-level partition; (d) maximal density but low modularity partition.

2. Change the splitting criterion of the algorithm, using the inverse edge weight rather than edge betweenness. Since this is much faster, you can perform the first 20 splits. Do you get higher or lower modularity relative to the result from exercise 1?
3. Use the maximum edge weight pointing to a community as a guiding principle to merge nodes into communities using the bottom-up approach. (An easy way is to just condense the graph by merging the two nodes with the maximum edge weight, for every edge in the network)
4. Using the algorithm you made for exercise 3, answer these questions: What is the latest step for which you have the average internal community edge density equal to 1? What is the modularity at that step? What is the highest modularity you can obtain? What is the average internal community edge density at that step?

38

Overlapping Coverage

Let's go back for a moment to the classical definition of communities in networks:

Communities are groups of nodes densely connected to each other and sparsely connected to nodes outside the community.

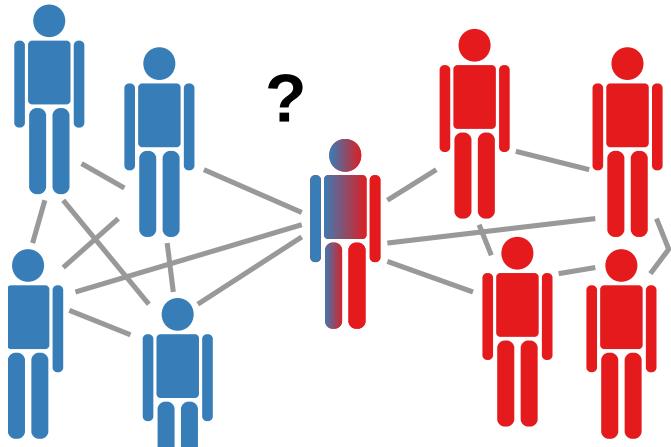


Figure 38.1: A social network with an individual having an equal number of relationships distributed among two communities.

This seems to imply that communities are a clear cut case. Nodes have a majority of connections to other nodes in their community. However real world networks do not have to conform to this expectation, and in fact often they don't. There are numerous cases in which nodes belong to multiple communities: to which community does the center person in Figure 38.1 belong? The red or the blue one?

The classical community definition forces us to make a choice. Regardless of the choice we make – red or blue – it wouldn't be a satisfying solution. The more reasonable answer is “she belongs to both”. For instance, a person can very well be part of one community because it is composed by the people they went to school with. And she can be part of a work community too, of people she works with. Some of these people could be the same, but usually they are not.

The problem is that none of the methods seen so far allow for such a consideration. For instance, the basic stochastic blockmodels only allows you to plant a node in a community, not multiple. Modularity also has issues, because of the Kronecker delta: since this is going to be 1 for multiple communities for a node, there will be double-counting and the formula breaks down.

This is where the concept of *overlapping* community discovery was born. We need to explicitly allow for overlapping communities: communities that can share nodes. There are many ways to do this, which have been reviewed in several articles^{1,2} dedicated especially to this sub problem of community detection (itself a sub problem of network analysis: it's communities all the way down).

Here we explore a few of the most popular approaches.

38.1 Evaluating Overlapping Communities

Before we delve deep into overlapping community discovery, let's amend Chapter 36 to this new scenario. We can have a few options when we try to evaluate how well we divided the network into overlapping communities.

Normalized mutual information expects you to put nodes into a single category. However, there are ways to make it accept an overlapping coverage^{3,4}. The obstacle is that NMI wants to compare the vector of metadata with the vector containing the community partition. The vector can only have one value per node but, in an overlapping coverage, it can have multiple values. Thus we don't compare the vectors directly. We compare two bipartite matrices.

Suppose you found \mathcal{C} communities, and you have \mathcal{A} node attributes. You can describe the overlapping coverage in communities with a $|V| \times \mathcal{C}$ binary matrix, whose u, C entry is equal to 1 if node u is part of community C . The node attribute matrix is similarly defined. Figure 38.2 shows an example of this procedure. Now you can calculate the mutual information between the two matrices by pairing the columns such that we assign to each column on one side the ones on the other side that is the most similar to it.

We can normalize this mutual information in different ways. In fact, the papers I cited earlier propose six alternatives, providing different motivations for each of those. These overlapping NMIs share with their original counterpart the issue of non-zero values for independent vectors – although they try to mitigate the issue with different strategies.

Some researchers have pointed out a few biases in the overlapping extensions of NMI and similar measures⁵. They propose a unified framework that can evaluate disjoint, overlapping, and even hierar-

¹ Jierui Xie, Stephen Kelley, and Boleslaw K Szymanski. Overlapping community detection in networks: The state-of-the-art and comparative study. *Acm computing surveys (csur)*, 45(4):43, 2013

² Alessia Amelio and Clara Pizzuti. Overlapping community discovery methods: A survey. In *Social Networks: Analysis and Case Studies*, pages 105–125. Springer, 2014

³ Andrea Lancichinetti, Santo Fortunato, and János Kertész. Detecting the overlapping and hierarchical community structure in complex networks. *New Journal of Physics*, 11(3):033015, 2009

⁴ Aaron F McDaid, Derek Greene, and Neil Hurley. Normalized mutual information to evaluate overlapping community finding algorithms. *arXiv preprint arXiv:1110.2515*, 2011

⁵ Alexander J Gates, Ian B Wood, William P Hetrick, and Yong-Yeol Ahn. Element-centric clustering comparison unifies overlaps and hierarchy. *Scientific reports*, 9(1):8574, 2019

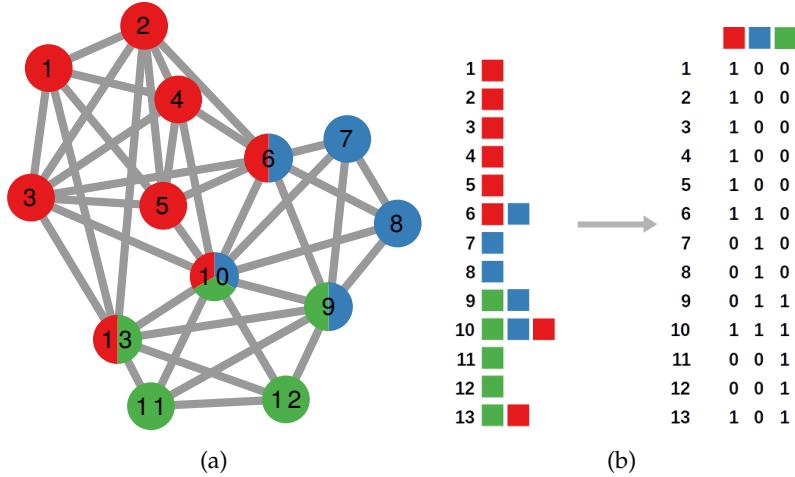


Figure 38.2: (a) A network with three overlapping communities, encoded by the node's color. (b) Transforming the overlapping coverage into a binary affiliation matrix, which we can use as input for the overlapping version of NMI

chical communities. This is achieved by creating a node-community bipartite affiliation graph and then project it so that you obtain a new node-node unipartite graph – just like the original network you started with. However, now, the relations between the nodes are due to their common cluster affiliations. The similarity between two community coverages is estimated by looking at the similarity of the stationary distributions of the projected affiliation graphs.

An alternative measure, called Omega Index, attempts to be the overlapping equivalent of the adjusted mutual information: the one correcting for chance⁶. This is an extension of the adjusted Rand index (Section 36.4) for non-disjoint clusters. The Rand index is simply the number of times two partitions agree over all possible pairs of nodes. The adjusted-for-chance version establishes the probability of agreeing by chance and uses that to normalize the index. The solution again passes through a procedure similar to what we explained for the overlapping version of NMI – but note that there is no universal null model to correct for, and the one you assume has a severe impact on the results you'll be seeing⁷.

What about modularity? Of course there should be a way to extend it to work with multiple clusters! How hard can that be? It isn't at all. In fact, it is so easy that there are multiple conflicting ways to extend modularity for the overlapping case. Because woe to the scientific community that can agree on something.

Solutions span from replacing the binary Kronecker delta with a continuous node similarity measure based on the product^{8,9} or the average¹⁰ of “belonging” coefficients (i.e. how much a node really belongs to a community); to simply calculating the average modularity of all communities¹¹; to a version incorporating both overlap and directed edges¹².

⁶ Linda M Collins and Clyde W Dent. Omega: A general formulation of the rand index of cluster recovery suitable for non-disjoint solutions. *Multivariate Behavioral Research*, 23(2):231–242, 1988

⁷ Alexander J Gates and Yong-Yeol Ahn. The impact of random models on clustering similarity. *The Journal of Machine Learning Research*, 18(1):3049–3076, 2017

⁸ Tamás Nepusz, Andrea Petróczi, László Négyessy, and Fülöp Bazsó. Fuzzy communities and the concept of bridgeness in complex networks. *Physical Review E*, 77(1):016107, 2008

⁹ Hua-Wei Shen, Xue-Qi Cheng, and Jia-Feng Guo. Quantifying and identifying the overlapping community structure in networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2009(07):P07042, 2009

¹⁰ Shihua Zhang, Rui-Sheng Wang, and Xiang-Sun Zhang. Identification of overlapping community structure in complex networks using fuzzy c-means clustering. *Physica A: Statistical Mechanics and its Applications*, 374(1):483–490, 2007b

¹¹ Anna Lázár, Dániel Ábel, and Tamás Vicsek. Modularity measure of networks with overlapping communities. *EPL*, 90(1):18001, 2010

¹² Vincenzo Nicosia, Giuseppe Manzoni, Vincenza Carchiolo, and Michele Malgeri. Extending the definition of modularity to directed graphs with overlapping communities. *Journal of Statistical Mechanics: Theory and Experiment*, 2009(03):P03024, 2009

Mentioning “belonging” coefficients allows me to make a distinction here. You can perform overlapping community discovery in two different ways. The first is by saying that nodes fully belong to multiple communities – i.e. that all the communities they belong to are equal for them. There is no way to say that a node is “more” part of one community or another. This is in contrast with fuzzy clustering¹³, in which nodes have such belonging coefficients and thus can tell you whether they really feel like they’re strongly part of a community or not.

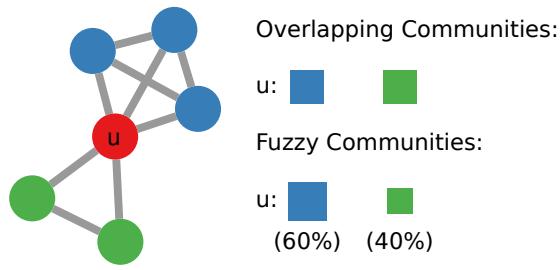


Figure 38.3 depicts this distinction. Fuzzy communities are more difficult to calculate, but they are also more precise. However, you should be careful in not relying on the coefficients you get from fuzzy clustering too much. Is there really a difference in saying that 49% of a node “belongs” to a community, versus saying that 51% of the node belongs to it? In some cases it might be, but you need to have trust in the fact that your data allows you such level of confidence.

38.2 Adapted Approaches

On the basis of having an overlapping version of modularity, overlapping community discovery needs not to be a separated problem with specialized solutions. We already have delineated a procedure to solve the problem: trying to maximize a target function. So we can take all the algorithms which maximize modularity, and make them maximize overlapping modularity instead.

This move can be applied to multiple other algorithms. For instance, we have an adaptation of Infomap¹⁴. If you remember Section 35.2, in Infomap we’re looking for a smart way to encode random walks. We do so by using short binary codes to identify nodes inside modules and special codes to indicate when the random walk crosses between communities.

In such strategy, the communities are disjoint, because if a node is part of two communities you would have to use the code for crossing between communities when you visit it. However, there is a way to make this encoding compatible with overlapping communities. That

¹³ James C Bezdek. *Pattern recognition with fuzzy objective function algorithms*. Springer Science & Business Media, 2013

Figure 38.3: Comparing overlapping and fuzzy clustering for node u : the size of the square is proportional to u ’s “belonging” coefficient, in share of number of u ’s edges connected to the community.

¹⁴ Alcides Viamontes Esquivel and Martin Rosvall. Compression of flow can reveal overlapping-module organization in networks. *Physical Review X*, 1(2): 021025, 2011

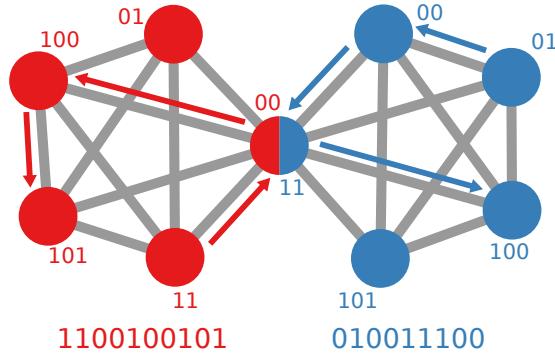


Figure 38.4: The encoding of two random walks in the overlapping version of Infomap. Note that in neither the red nor the blue path we're crossing community boundaries, so we don't use the community crossing code.

is giving to a node part of multiple communities a different code per community. Figure 38.4 shows an example.

For instance, if the Reykjavik airport is part of both the North American and the European community, it will have two codes. We would use one code if the random walk approaches Reykjavik from a North American airport, and the other code if it approaches Reykjavik from a European one. You would then use the community crossing code only if you actually transition between clusters in the next step.

Finally, let's consider label propagation. There is a way to extend the classical label propagation algorithm to allow for overlapping communities¹⁵. The idea is that nodes will not just adopt the single most common label among their neighbors. They will adopt all labels, each weighted by a "belonging coefficient", which is the weighted average of the belonging coefficient of that label across all neighbors.

Labels that are below a specific belonging coefficient threshold are removed from the node at each step, preventing the nodes to converge to a single global status where all nodes belong to all communities at the same level of belonging.

Figure 38.5 shows a small run of this principle. Let's suppose that we set our minimum belonging coefficient to 0.5. In the first step (from Figure 38.5(a) to 38.5(b)), the two fully red nodes stay fully red, because they both receive 0.5 of the red label, 0.33 of the blue and 0.16 of the purple label. The only label clearing the 0.5 threshold is the red one, thus they become fully red. The half red half purple node becomes red because that's the only label around it. The central node is also red, receiving 0.5 red, 0.25 blue and 0.25 orange. From Figure 38.5(b) to 38.5(c) though, the central node will correctly split between red and blue, because they both contribute half of its neighborhood.

¹⁵ Steve Gregory. Finding overlapping communities in networks by label propagation. *New Journal of Physics*, 12(10):103018, 2010

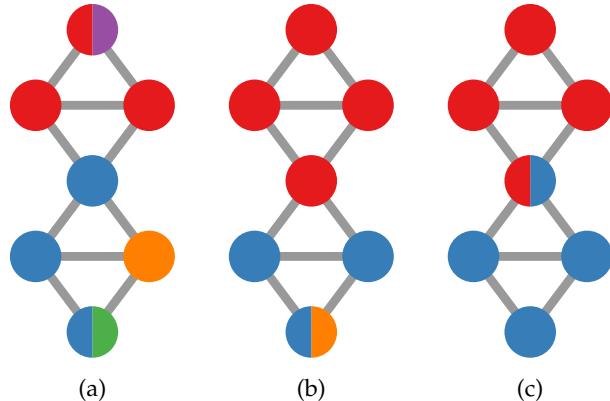


Figure 38.5: A simple run of the overlapping label propagation. The node's color is a pie chart representing the belonging coefficient of the node to a label.

38.3 Explicit Structural Approaches

In the class of structural approaches we find methods that have a definition of what an overlapping community should look like, and try to find such a structure in the network. The idea is different from modularity maximization, because it is not primarily driven by the optimization of a function. I add the word “explicit” because these approaches exclusively look at the structure as it is, and try to find the communities there. In the next section we’ll see “latent” structural approaches which assume that the observed structure is the result of a hidden one driving the connections.

The explicit structural approach is historically the oldest solution to overlapping community discovery. I can think of fundamentally two subclasses: the famous clique percolation approach, and node splitting.

Clique Percolation

Clique percolation starts from the observation that communities should be dense. What is the densest possible subgraph? The clique. In a clique, all nodes are connected to all other nodes. So the problem of community discovery more or less reduces to the problem of finding all cliques in the network. However, this is a bit too strict: there are subgraphs in the network that, while being very dense and close to being a clique, are not fully connected. It would be a pity to split them into many small substructures.

Thus researchers developed the more sophisticated k -clique percolation algorithm¹⁶. Clique percolation says that communities must be cliques of at least k nodes, with k being a parameter you can freely set. In the first step, the algorithm finds all cliques of size k , whether they are maximal or not. Then, it attempts to merge two communities in the same community if the two communities share at least a $k - 1$

¹⁶ Imre Derényi, Gergely Palla, and Tamás Vicsek. Clique percolation in random networks. *Physical review letters*, 94(16):160202, 2005

clique.

For instance, consider the example in Figure 38.6, setting the parameter $k = 5$. The blue and green 5-cliques only share two nodes, so it cannot be a 4-clique. But the green and purple do share a 4-clique, so they are merged (top row). And there is another purple 5-clique that can now be merged with the green community (bottom row).

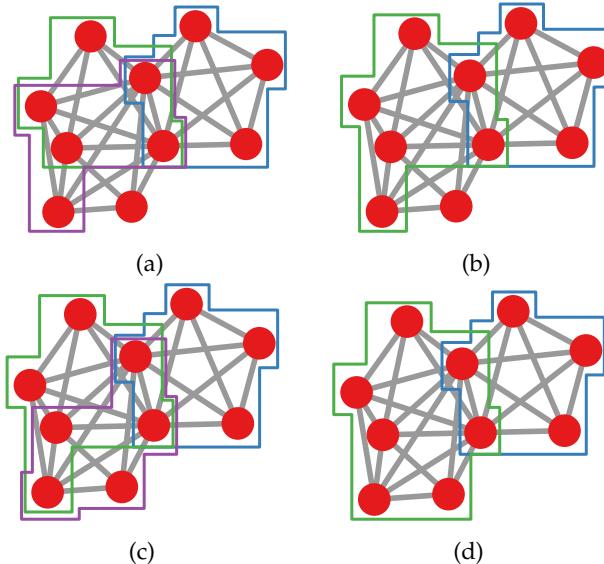


Figure 38.6: An example of clique percolation. 5-Cliques are highlighted by outlines. Green cliques percolate with purple cliques because they share $k - 1 = 4$ nodes.

This is generally implemented via the creation of a clique graph¹⁷. The nodes of a clique graph are the cliques in the original graph. We connect two cliques if they share nodes. For instance, if we only connect cliques sharing $k - 1$ nodes, then we can efficiently find all communities by finding all connected components in the clique graph.

This algorithm works well in practice. It has been used to study overlapping friendship patterns in school systems¹⁸ – due to classroom being quasi-cliques: pupils have rare but significant friendships across classes –, and in metabolic networks¹⁹. However, it has a couple of downsides.

First, finding all cliques in a network is computationally expensive. One could fix this problem by setting k to be relatively high. If we set $k = 5$ we know that nodes with degree three or less cannot be in any community, because they need at least four edges to be part of a 5-clique. Since most networks have broad degree distributions (Section 9.3), this means that we can safely ignore the vast majority of the network, thus reducing the number of operations we need to find communities. This is a suboptimal solution because it implies that one will not classify most nodes into communities. For this reason,

¹⁷ Tim S Evans. Clique graphs and overlapping communities. *Journal of Statistical Mechanics: Theory and Experiment*, 2010(12):P12037, 2010

¹⁸ Marta C González, Hans J Herrmann, J Kertész, and Tamás Vicsek. Community structure and ethnic preferences in school friendship networks. *Physica A*, 379(1):307–316, 2007

¹⁹ Shihua Zhang, Xuemei Ning, and Xiang-Sun Zhang. Identification of functional modules in a ppi network by clique percolation clustering. *Computational biology and chemistry*, 30(6):445–451, 2006

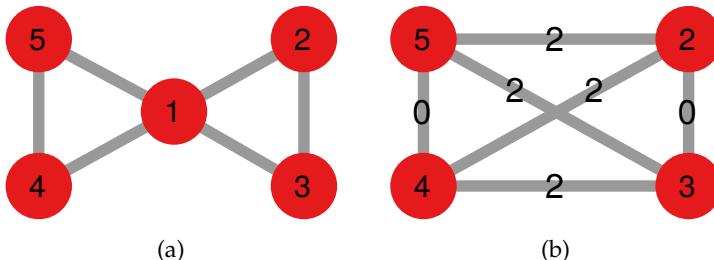
there are developments of this algorithm^{20,21} that are a bit more computationally efficient.

Second, it has limited coverage for sparse networks. That means that it might end up being unable to classify nodes in networks because they are not part of any clique. If you set your k relatively low, e.g. $k = 4$, all nodes with degree equal to one cannot be part of any community. This is because a node with degree equal to one cannot be part of a 3-clique. Thus it will never be merged into any 4-clique, which are the basis of our communities.

Node Splitting

Another approach is to simply recognize that a node is part of multiple communities if it has different identities. This is extremely similar to the approach of overlapping Infomap. In that case we represented the two identities of the node by giving it two different codes: one per community to which it belongs. Here we literally split it in two. We modify the structure of the network in such a way that, when we are done, by performing a normal non-overlapping community discovery we recover the overlapping clusters. In the resulting structure we have multiple nodes all referring to the same original one.

If we want to split nodes, we need to answer two questions: which nodes do we split and how. First we identify the nodes most likely to be in between communities. If you remember the definition of betweenness, you'll recollect that nodes between communities are the gatekeepers of all shortest paths from one community to the other. So they are the best candidates to split. There are many ways to perform the split, but I'll focus on the one that involves calculating a special betweenness: pair betweenness^{22,23}. Pair betweenness is a measure for a pair of edges: the number of shortest paths that use both of them.



For instance, consider the graph in Figure 38.7(a). The most central node is node 1. To try and split it, we build its split graph. Meaning that we remove node 1 and we connect all nodes that were connected by 1. Each edge has a weight: the number of shortest paths in the

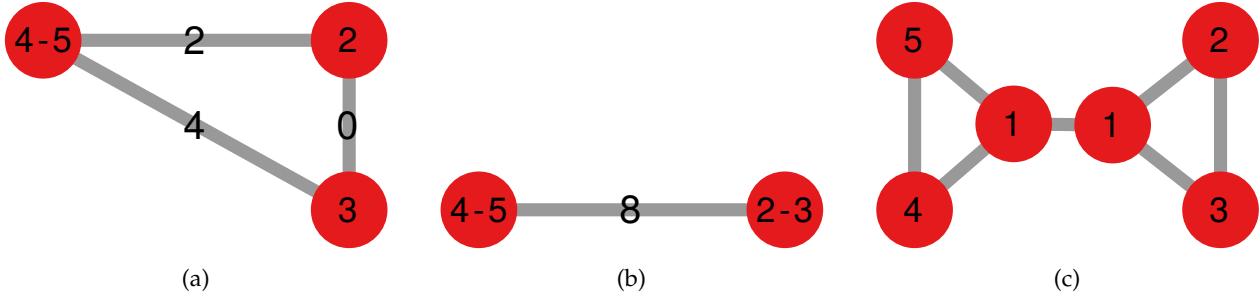
²⁰ Jussi M Kumpula, Mikko Kivelä, Kimmo Kaski, and Jari Saramäki. Sequential algorithm for fast clique percolation. *Physical Review E*, 78(2):026109, 2008

²¹ Fergal Reid, Aaron McDaid, and Neil Hurley. Percolation computation in complex networks. In *2012 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, pages 274–281. IEEE, 2012

²² Steve Gregory. An algorithm to find overlapping community structure in networks. In *European Conference on Principles of Data Mining and Knowledge Discovery*, pages 91–102. Springer, 2007

²³ Steve Gregory. Finding overlapping communities using disjoint community detection algorithms. In *Complex networks*, pages 47–61. Springer, 2009

Figure 38.7: Attempting to find the best node to split in (a). Selecting node 1 as the candidate, we build a pair betweenness graph (b). I label each edge in (b) with its pair betweenness.



original graph that passed through node 1. In this case, there are two shortest paths using the $(4, 1)$ and $(1, 3)$ edges: the one going from node 4 to node 3 and the one going from node 3 to node 4. We can represent the pair betweenness of all neighbors of node 1 with a weighted clique (Figure 38.7(b)).

To find the split we use a simple algorithm: we identify the edges with the lowest pair betweenness and we merge the nodes connected by those edges (Figures 38.8(a-b)). At each merge, we sum up the pair betweennesses of all edges that got merged together by the merging of the node. Once we have one remaining edge, the resulting split is the best one. The reason is that edges with low pair betweenness are likely to be in the same community. Once you identify the split (Figure 38.8(c)), it is easy to find disjoint communities and then merge them into overlapping.

38.4 Latent Structural Approaches

In this section we have a collection of methods that make an assumption: the observed community division of the network is the result of a latent structure. In this latent structure, we have nodes assigned to communities. Then, the probability of observing an edge between two nodes is proportional to the number of communities the two nodes have in common in the latent structure. The two classes of approaches in this category I consider are Mixed Membership Stochastic Blockmodels (MMSB) and the community affiliation graph.

Mixed Membership Stochastic Blockmodels

The description of the latent structural approaches I just wrote should turn on a light bulb in your head. The idea that the probability of connecting two nodes is dependent on a latent community partition is not new: it is exactly the starting point of the stochastic blockmodel approach. In SBM, we assume there is a partition of nodes and we assign a higher probability of connection between

Figure 38.8: (a-b) Merging the nodes connected by the weakest pair betweenness edges. (c) The resulting split in the original graph.

nodes in the same partition than the one between nodes in different partitions. The little problem we need to solve now is how to make this mathematical machinery work when we want to allow nodes to be part of multiple communities. That solution constitutes the Mixed Membership Stochastic Blockmodels²⁴, an object that I already mentioned in Section 18.2.

The trick here is that we represent each node's membership as a vector. The vector tells us how much the node belongs to a given community. Then, we also have a community-community matrix, that tells us the probability of a node belonging to community c_1 to connect to a node belonging to a community c_2 . These are the two ingredients that replace the simple community partition in the regular SBM. From this moment on, you attempt to find the set of community affiliation vectors and the community-community probability matrix that are most likely to reproduce your observed data, exactly as you do in SBM.

Just like we saw in Section 35.4, we can have dynamic MMSB, adding time to the mix^{25,26,27,28}: the community affiliation vectors and the community-community matrix can change over time. There is also a hierarchical (Chapter 37) variant of MMSB, allowing a nested community structure²⁹.

Community Affiliation Graph

Affiliations graphs have been often used to describe the overlapping community structure of real world networks³⁰. In a community affiliation graph you assume that you can describe your observed network with a latent bipartite network. In this bipartite network, the nodes of one type are the nodes of your observed network. The other type, the latent nodes, represent your communities. Nodes are connected to the communities they belong to. This is the community affiliation graph, because it describes the affiliations to communities of your nodes.

Figure 38.9 shows a representation of a community affiliation graph. Of course, you can build such a graph easily once you already know to which communities the nodes belong. The hard part is finding out the best representation. There are a few ways to do so, usually relying on the expectation maximization algorithm that is also at the basis of the MMSB. One such approach is BigClam³¹, which uses non-negative matrix factorization as the guiding principle (see Section 5.6 for a refresher).

²⁴ Edoardo M Airoldi, David M Blei, Stephen E Fienberg, and Eric P Xing. Mixed membership stochastic blockmodels. *Journal of Machine Learning Research*, 9(Sep):1981–2014, 2008

²⁵ Wenjie Fu, Le Song, and Eric P Xing. Dynamic mixed membership blockmodel for evolving networks. In *Proceedings of the 26th annual international conference on machine learning*, pages 329–336. ACM, 2009

²⁶ Eric P Xing, Wenjie Fu, Le Song, et al. A state-space mixed membership blockmodel for dynamic network tomography. *The Annals of Applied Statistics*, 4(2):535–566, 2010

²⁷ Qirong Ho, Le Song, and Eric Xing. Evolving cluster mixed-membership blockmodel for time-evolving networks. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 342–350, 2011

²⁸ Kevin S Xu and Alfred O Hero. Dynamic stochastic blockmodels: Statistical models for time-evolving networks. In *International conference on social computing, behavioral-cultural modeling, and prediction*, pages 201–210. Springer, 2013

²⁹ Tracy M Sweet, Andrew C Thomas, and Brian W Junker. Hierarchical mixed membership stochastic blockmodels for multiple networks and experimental interventions. *Handbook on mixed membership models and their applications*, pages 463–488, 2014

³⁰ Jae Dong Noh, Hyeong-Chai Jeong, Yong-Yeol Ahn, and Hawoong Jeong. Growing network model for community with group structure. *Physical Review E*, 71(3):036131, 2005

³¹ Jure Leskovec and Jaewon Yang. Overlapping community detection at scale: a nonnegative matrix factorization approach. In *Proceedings of the sixth ACM international conference on Web search and data mining*, pages 587–596. ACM, 2013

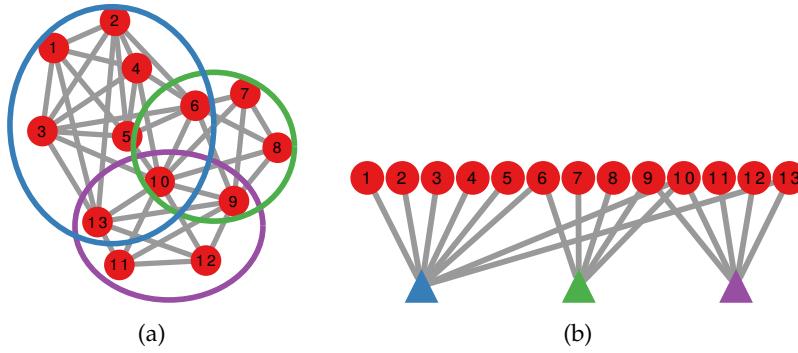


Figure 38.9: (a) A graph with overlapping communities indicated by the colored outlines. (b) Its corresponding community affiliation graph. The community latent nodes are triangular and their color corresponds to the color used in (a).

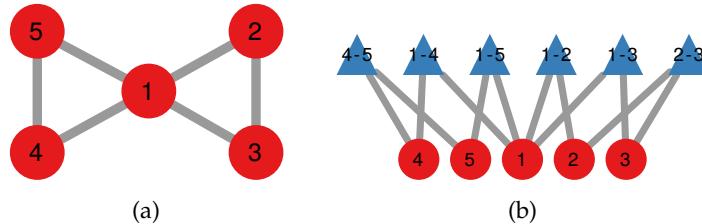
38.5 Clustering Links

An alternative approach is to look at edges. Why looking at edges? Because people can be in multiple communities, as we saw: work mates, school mates, etc. However, a link usually is created for one single reason: you met that person in one situation and that's the defining characteristic of your relationship. You originally met u as a work colleague, and v as a school mate. So one can cluster links with a non-overlapping method and say that a person is part of all the communities to which their edges belong.

There can be many similarity measures and link communities can be found by adapting and optimizing such functions to the link community case.

Line Graphs

To cluster the edges rather than the nodes we can transform the network into its corresponding line graph³². In a line graph, as we saw in Section 6.1, the edges become nodes and they are connected if they're incident on the same node. A way to do so is to generate a weighted line graph.



³² TS Evans and Renaud Lambiotte. Line graphs, link partitions, and overlapping communities. *Physical Review E*, 80(1):016105, 2009

Figure 38.10: (a) A simple graph. (b) A bipartite version of (a) connecting each node to its edges.

To create a line graph you first transform the network into bipartite connecting the nodes to the edges they are connected to, as Figure 38.10 shows. Then you project this network over the edges. The most important thing to define is how to weight the edges in

the line graph. Different weight profiles will steer the community discovery on the line graph in different directions.

You could use any of the weighting schemes I discussed in Chapter 26, but the researchers proposing this method also have their suggestions. The reason you might need a special projection is because you want nodes that are part of an overlap to give their edges lower weights, because their connections are spread out in different communities.

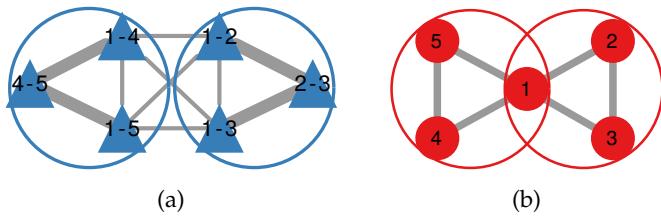


Figure 38.11: (a) The blue circles are disjoint communities in the line graph. (b) The red circles are the corresponding overlapping communities in the original graph.

At that point, a disjoint community discovery will downplay the weak edges and find communities with strong edge weights, as Figure 38.11(a) shows. Once we bring back the communities to the other side of the projection, as I do in Figure 38.11(b), we have overlapping ones.

Other approaches in this class exist, including random walks on line graphs³³.

Hierarchical Link Clustering

In Hierarchical Link Clustering³⁴ (HLC), the first step is to calculate a measure of similarity between two edges. We only calculate it for edges sharing one node: edges (u, k) and (v, k) share node k . If two edges share no node, their similarity is zero. If the edges share a node, their similarity is the Jaccard coefficient of the neighborhoods of the two non-shared nodes:

$$S_{(u,k),(v,k)} = \frac{|N_u \cap N_v|}{|N_u \cup N_v|}.$$

The edges with the highest S value are merged in the same community. For instance, in Figure 38.12, edges (1,2) and (1,3) have a high S value: the neighborhoods of nodes 2 and 3 are identical, thus $S_{(1,2),(1,3)} = 1$. On the other hand, edges (4,7) and (7,8) only have one node in the numerator, thus: $S_{(4,7),(7,8)} = 1/6$.

Then, the merging happens recursively for lower and lower S values, building a full dendrogram, as we saw in Chapter 37 for hierarchical community discovery. We then need a criterion to cut the dendrogram. We cannot use modularity, because these are link communities, not node communities.

³³ Xiaoheng Deng, Genghao Li, and Mianxiong Dong. Finding overlapping communities with random walks on line graph and attraction intensity. In *International Conference on Wireless Algorithms, Systems, and Applications*, pages 94–103. Springer, 2015

³⁴ Yong-Yeol Ahn, James P Bagrow, and Sune Lehmann. Link communities reveal multiscale complexity in networks. *nature*, 466(7307):761, 2010

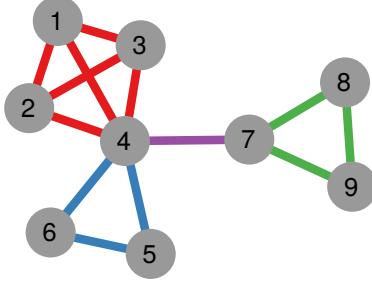


Figure 38.12: A graph and its best link communities. The color of the edge represents its community. Nodes are part of all communities of their links. For instance, node 4 belongs to three communities: red, blue, and purple.

The original authors develop a new quality measure called “partition density”. For each link community c , we have $|E_c|$ as the number of edges it contains, and $|V_c|$ as the number of nodes connected to those edges. Its density D_c is

$$D_c = \frac{|E_c| - (|V_c| - 1)}{|V_c|(|V_c| - 1)/2 - (|V_c| - 1)},$$

which is the number of links in c , normalized by the maximum number of links possible between those nodes ($|V_c|(|V_c| - 1)/2$), and its minimum $|V_c| - 1$, since we assume that the subgraph induced by c is connected. Note that, if $|V_c| = 2$, we simply take $D_c = 0$. All D_c scores for all c s in your link partition are aggregated to find the final partition density, which is the average of D_c weighted by how many links are in c : $D = \frac{1}{|E|} \sum_c |E_c| D_c$.

Ego Networks

Assuming that links exists for one primary reason works usually well, but it is a problematic assumption. Let's look back at the case of work and school communities. What would happen if you were to end up working in the same company and play in the same team of a former schoolmate? Is it still fair to say that the link between the two of you exists for only one predominant reason?

Modeling truly overlapping communities can get rid of this problem. There are many ways to do it, but we'll focus on one that is easy to understand. The starting observation is that networks have large and messy overlaps. However, just like in the assumption of clustering links, here we realize that the neighbors of a node usually are easier to analyze. It is easy for a node to look at a neighbor and say: “I know this other node for this reason (or set of reasons)”.

The procedure³⁵ works as follows, and I use Figure 38.13 to guide you. First, we extract the ego network of a node, removing the ego itself. This creates a simpler network to analyze. In the figure, I start by looking at node 1 on the top right. This is a graph with two

³⁵ Michele Coscia, Giulio Rossetti, Fosca Giannotti, and Dino Pedreschi. Demon: a local-first discovery method for overlapping communities. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 615–623. ACM, 2012.

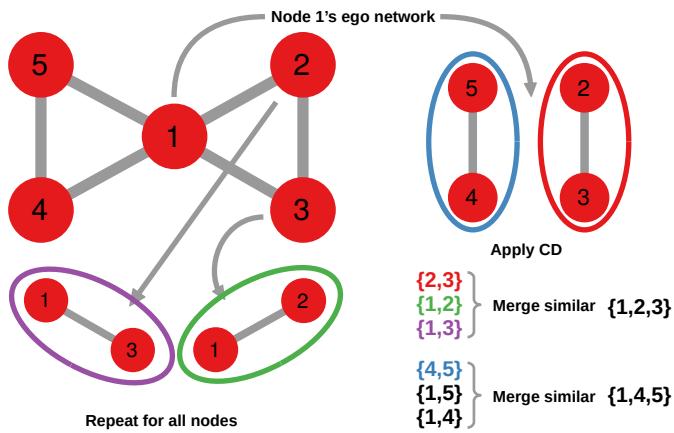


Figure 38.13: The process of community discovery via the breaking down of the network into ego networks.

connected components: one connecting nodes 2 and 3, the other connecting nodes 4 and 5.

Then, we apply a disjoint community discovery algorithm to the ego network. The ego network is easier to analyze and often has easily distinguishable communities. In the example, these are the blue and red outlines, which find two trivial communities. We repeat the process for all nodes in the network, extracting all ego networks: in the figure I show the ego networks of nodes 2 and 3, with the communities I find in those cases, the green and purple outlines, respectively. Note that I omit the ego networks for nodes 4 and 5, but you can hopefully see where this is going.

Once we're done, we have a set of communities, and we can merge them according to some criterion. In the case of the figure, we merge communities if they share at least one node that is not part of too many communities. So we merge $\{2,3\}$ to $\{1,2\}$ and $\{1,3\}$ on the basis of them sharing nodes 2 and 3. Same reasoning for merging $\{4,5\}$ to $\{1,4\}$ and $\{1,5\}$. We don't merge $\{1,2,3\}$ and $\{1,4,5\}$, because the only node they have in common is 1, which is in common with all communities in the network.

The original paper uses label propagation as the community discovery algorithm to apply to each network, but this needs not to be the case. We can instead apply a naive overlap algorithm. This move allows us to solve the problem of the assumption we mentioned before: we're allowing the links to have multiple origins, thus we don't necessarily force each link to be present for exclusively one reason.

38.6 Other Approaches

An interesting and more general approach to overlapping community discovery is the Order Statistics Local Optimization Method³⁶

³⁶ Andrea Lancichinetti, Filippo Radicchi, José J Ramasco, and Santo Fortunato. Finding statistically significant communities in networks. *PLoS one*, 6(4): e18961, 2011

(OSLOM). In reality, OSLOM is a bit of a Swiss army knife, in the sense that it isn't limited by finding overlapping communities: it can deal with edge weights and directions, hierarchical and evolving communities. Its basic philosophy is extremely similar to modularity: it builds an expected number of edges in a community by means of a configuration model.

Differently from modularity, OSLOM attempts to establish the statistical significance of the partition. That is, it asks how likely it is to observe the given community subgraphs in a configuration model. The less likely a vertex is to be included in a community in a null model, the more likely it is that we should add it to the community. Thus, these p values can be used as a mean of ranking the next move, a move being adding a node to a community.

One can see how OSLOM is also a hierarchical community discovery method of the merge type: it assumes all nodes being on their own community at the beginning, and then it progressively merges them. Differently from classical hierarchical CD, a node is still a merge candidate even after it has been added to a community, allowing overlap. Moreover, OSLOM can be used as a post-processing strategy, to refine the communities you already found using another method. In fact, one could use OSLOM to transform a hard disjoint partition into an overlapping coverage.

38.7 The Overlap Paradox

Let's go back to the beginning of this chapter. Let's reiterate the definition of a community in a network:

Communities are groups of nodes densely connected to each other and sparsely connected to nodes outside the community.

As we've been seeing, overlapping communities make a paradox arise in our definition of community. If we have no overlap the "denser inside" and "sparser outside" assumption works well. However, if we add overlap, we cannot have it both ways.

If there are nodes in between communities either of two things will happen. The nodes in the overlap could connect with all nodes in both communities but not to each other, to maintain the "external sparsity" condition. But doing so contradicts the "internal density" part, because the overlap nodes do no connect to each other even though they belong to the same community. Figure 38.14 provides an example for this scenario.

In Figure 38.14(a) we have a graph made by four 5-cliques and four sets of four nodes overlapping between two neighboring cliques. The overlap nodes don't connect to each other. Figure 38.14(b) shows

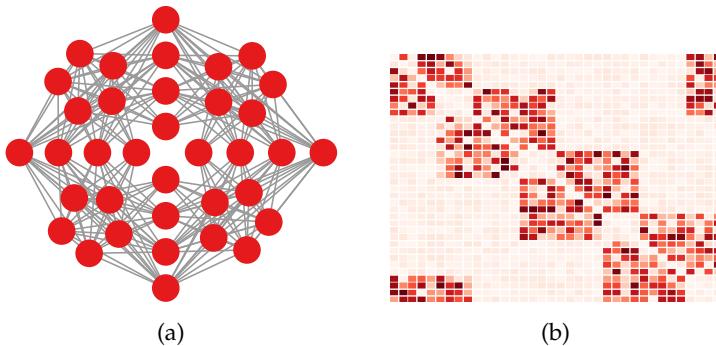


Figure 38.14: A graph (a) and its stochastic blockmodel (b).

how a stochastic blockmodel would interpret such a structure. You can clearly see that there are “holes” in the communities where the overlap nodes should be. If the overlap nodes don’t connect to each other, they have low connection probability, which contradicts the fact that they are part of the same community.

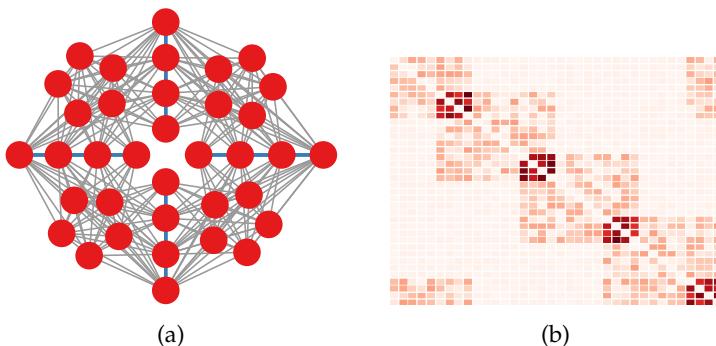


Figure 38.15: (a) The same graph as the one in Figure 38.14, but the added edges are highlighted in blue. (b) The corresponding stochastic blockmodel.

If, on the other hand, we maintain the “internal density” condition, since these nodes share not one but two communities, then they are more likely to connect to each other than nodes sharing only one community. In doing so, we end up with the opposite problem: breaking external sparsity. The overlap, which by definition is between the two communities, is denser than the community itself³⁷!

In Figure 38.15(a) we have such a scenario, with a graph similar to the one from Figure 38.14(a). But here all the overlap nodes are connected to each other, and they are connected more strongly than non-overlap nodes, given that they share more communities with each other. The corresponding stochastic blockmodel (Figure 38.15(b)) now shows that the communities themselves look weaker than the overlap.

This is another reason why our golden rule, the standard definition of communities in complex networks, isn’t as shiny as we originally thought.

³⁷ Jaewon Yang and Jure Leskovec. Community-affiliation graph model for overlapping network community detection. In *2012 IEEE 12th international conference on data mining*, pages 1170–1175. IEEE, 2012

38.8 Summary

1. In real world networks, communities can overlap, meaning that nodes can be part of multiple communities at the same time. For instance, you're part of both the community of your high school friends, and of your university colleagues.
2. Many quality measures like modularity or mutual information cannot deal with overlapping communities. Thus we have several extensions that allow them to take into account node-sharing communities.
3. Disjoint community discovery algorithms can be adapted to find overlapping communities, for instance by means of fuzzy clustering: each node is given a "belonging coefficient" for each community, and can have multiple coefficients larger than zero.
4. Explicit structural approaches define the structure of an overlapping community and attempt to find it in the network. For instance, by percolating cliques, or splitting nodes so that each of their copies can belong to different communities.
5. Alternatively, one can divide edges into communities rather than nodes. In such approaches, the nodes belong to all communities to which their connections belong.
6. Overlapping communities put our classical community definition in crisis: if it is true that the more communities two nodes share the more likely they are connected, then the overlap of multiple communities is denser than the communities themselves, i.e. there are more links going outside communities than inside.

38.9 Exercises

1. Use the k-clique algorithm to find overlapping communities in the network at <http://www.networkatlas.eu/exercises/38/1/data.txt>. Test how many nodes are part of no community for k equal to 3, 4, and 5.
2. Compare the k-clique results with the coverage in <http://www.networkatlas.eu/exercises/38/2/comms.txt>, by using any variation of overlapping NMI from <https://github.com/aaronmcdaid/Overlapping-NMI>. For which value of k do you get the best performance?
3. Implement the ego network algorithm: for each node, extract its ego minus ego network and apply the label propagation algorithm,

then merge communities with a node Jaccard coefficient higher than 0.1 (ignoring singletons: communities of a single node). Does this method return a better NMI than k-clique percolation for $k = 3$?

39

Bipartite Community Discovery

So far we have extended the community discovery problem by adding or discussing features of the output: do we want communities to share nodes or not? Do we want to have some sort of hierarchical optimization? In this and in the next chapter we instead discuss advancements on the other direction: what if our input is special? Here we deal with the case of bipartite networks, leaving multilayer networks for the next chapter.

We start by briefly amending modularity to the bipartite case. The bulk of the chapter is dedicated to alternative and more specialized ways to find bipartite communities. As usual, you can find a specialized survey of bipartite community detection methods¹.

39.1 Evaluating Bipartite Communities

Since it has been a looming presence across this entire book part, let's start again with modularity, the elephant in the room of community discovery. Network scientists in the community detection business love modularity. If there is a scenario in which modularity doesn't work, they panic and start amending it to hell, until it works again. We've seen this with directed and overlapping community discovery, and we're seeing it again.

There are a couple of alternatives when it comes to define a modularity that works for bipartite networks. If you remember the original version of the modularity, it hinges on the fact that we want the partition to divide the network in communities that are denser than what we would expect given a null model – the configuration model. Thus, extending modularity means to find the right formulation of a null model for bipartite networks^{2,3}.

This is not that difficult, the only thing to keep in mind is that the expected number of edges in a bipartite network is different than in a regular network. So, while in the traditional modularity

¹ Taher Alzahrani and Kathy J Horadam. Community detection in bipartite networks: Algorithms and case studies. In *Complex systems and networks*, pages 25–50. Springer, 2016

² Michael J Barber. Modularity and community detection in bipartite networks. *Physical Review E*, 76(6):066102, 2007

³ Roger Guimerà, Marta Sales-Pardo, and Luís A Nunes Amaral. Module identification in bipartite and directed networks. *Physical Review E*, 76(3):036102, 2007

the configuration model connection probability was $\frac{k_u k_v}{2|E|}$, here it is instead $\frac{k_u k_v}{|E|}$, with the added constraints that u and v needs to be nodes of unlike type. The sum of modularity is made only across pairs of nodes of unlike types, otherwise we would have negative modularity contributions from nodes that cannot be connected, which would make the modularity estimation incorrect.

To see why this is the case, suppose that we're checking u and v and they are of the same type. Since they are of the same type and we're in a bipartite network, they cannot connect to each other, so $A_{uv} = 0$. But they are both part of the network, thus $k_u \neq 0$ and $k_v \neq 0$. Thus $\frac{k_u k_v}{|E|} > 0$, meaning that $A_{uv} - \frac{k_u k_v}{|E|} < 0$. Negative modularity contribution.

Once you have a proper bipartite modularity you can use any of the modularity maximization algorithms to find modules in your network, or even specialized ones⁴.

39.2 Via Projection

As we saw in previous sections, bipartite networks have nodes of two different types, and edges are established exclusively between nodes of different types. In Netflix, we have users watching movies. It's natural to want to find communities in these networks. You want to know which movies are similar to each other so you can suggest them to users that are similar to each other.

In this case there is an easy obvious strategy. You take the bipartite network, you project it to unipartite using one of the techniques we saw in Chapter 26 – simple or hyperbolic weighting, random walks, etc – and then you apply a normal unipartite community discovery to the result. Then you can project on the other set of nodes and find the other communities.

There are a couple of issues with this strategy. The first is that by projecting you're losing information. You connect movies with a weighted edge, which carries a quantitative information. But the bipartite network had qualitative information: a structure of users watching different things. That information is lost.

This is related to the second issue: once you project your network on your two types of edges and find their communities, you have movies grouped together because watched by the same users. But you don't know who those users are. Same with communities of users: which are their common movies? You have to go back to the bipartite network to know. A way to solve this issue is to use the dual projection approach⁵. In dual projection, you project the

⁴ Stephen J Beckett. Improved community detection in weighted bipartite networks. *Royal Society open science*, 3(1): 140536, 2016

⁵ Martin G Everett and Stephen P Borgatti. The dual-projection approach for two-mode networks. *Social Networks*, 35(2):204–210, 2013

bipartite networks into its two unipartite versions and then you analyze them at the same time with specialized techniques. This dual projection approach has been applied to community discovery⁶, with encouraging results.

⁶ David Melamed. Community structures in bipartite networks: A dual-projection approach. *PLoS one*, 9(5): e97823, 2014

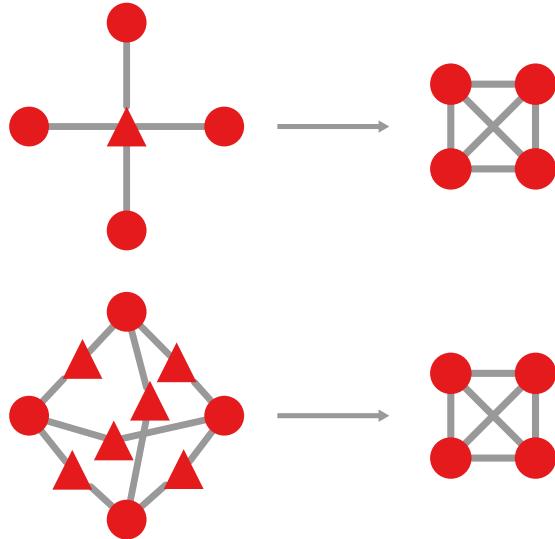


Figure 39.1: Examples showing different bi-structures projecting to the same uni-structures. Both a 1,4-clique (top) and six 1,2-cliques (bottom) project into a 4-clique.

There is an additional, more subtle, issue with this strategy. There are a number of different structures in bipartite networks that projects into the same unipartite graphs. This means that a proper bipartite community discovery algorithm and its hypothetical unipartite version will return different results, even if someone runs the mirror algorithm on the projection of the bipartite graph. Figure 39.1 shows two different bipartite networks that project to the same result. As the figure shows, projecting always means losing information. If we were to perform such projection we wouldn't be able to distinguish between the two cases Figure 39.1 shows, while a proper bipartite community detection algorithm could.

39.3 Direct Bipartite Module Detection

Bi-Clique Percolation

The solution is to perform the community discovery directly on the bipartite structure. Here, we use the concept of bi-clique we saw earlier. Remember that a clique is a set of nodes in which all possible edges are present. A bi-clique is the same thing, considering that some edges in a bipartite network are not possible. For instance, a 5-clique in a unipartite network is a graph with five nodes and ten edges. In a bipartite network, a 2,3-clique has two nodes of type 1, three nodes of type 2, and all nodes of type 1 are connected to nodes

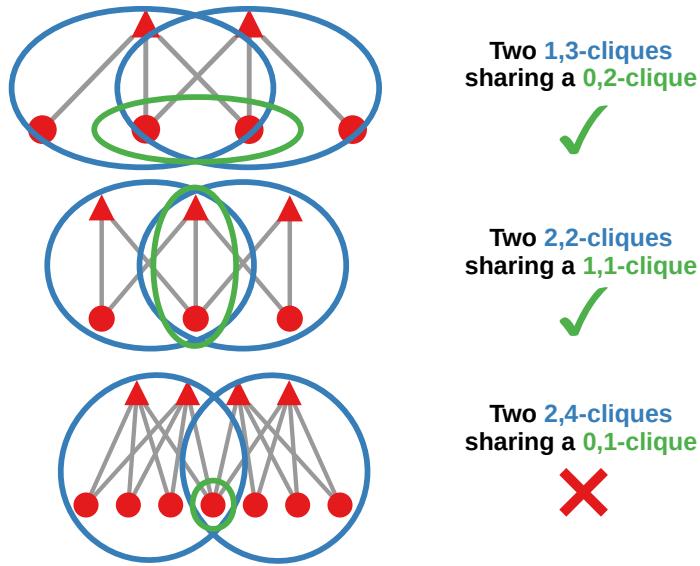


Figure 39.2: Examples of bi-clique percolation. The top two examples will lead to percolation because they satisfy the $n - 1, m - 1$ constraint. The last example on the bottom does not.

of type 2 – six edges in total. This is the starting point of bipartite community discovery.

Bi-clique percolation works in the same way as the unipartite k -clique percolation algorithm – described in Section 38.3 –, with the added headache of having two numbers of nodes to keep track, because now it's a biclique. Communities are n,m -cliques (again, n and m are parameters) and they get merged if they share an $(n - 1)(m - 1)$ -clique. Figure 39.2 shows some examples of bi-clique percolation. Note that any combination of n nodes of the same type can be a $0,n$ -clique, thus allowing percolation.

The bi-clique percolation algorithm inherits from its predecessor the ability of returning overlapping communities. Thus in this case we get not only bipartite communities, but these communities can also share nodes.

Adapting Classical Approaches

As we already got used to see, many of the classical approaches to community discovery can be adapted to take into account complications in the network structure. Bipartite community discovery is no exception. So let's see what we can do to transform random walks, label propagation, and stochastic blockmodels to the bipartite case.

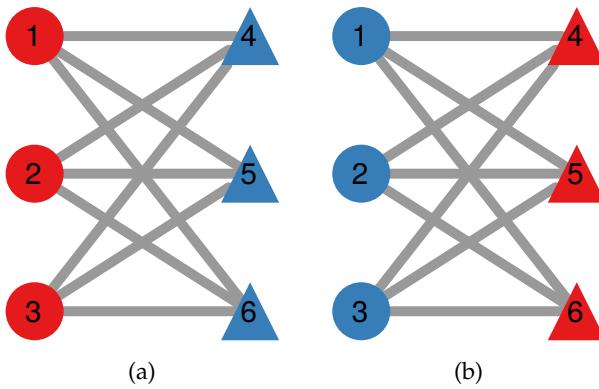
There are a couple of ways to exploit random walks and find bipartite communities^{7,8}. We already considered one: simply project the network as unipartite and perform the random walks normally. The issue is that the abundance of links will lower the power of random walkers, because the network will be too dense and the

⁷ Masoumeh Kheirkhahzadeh, Andrea Lancichinetti, and Martin Rosvall. Efficient community detection of network flows for varying markov times and bipartite networks. *Physical Review E*, 93(3):032309, 2016

⁸ Taher Alzahrani, Kathy J Horadam, and Serdar Boztas. Community detection in bipartite networks using random walks. In *Complex Networks V*, pages 157–165. Springer, 2014

boundaries between communities will be difficult to find. That is why you might also want to perform network backboning (Chapter 27).

Alternatively, you could perform high-order random walks. We saw in Chapter 34 that we can add a parameter to a random walker, telling it how much time it passes between one step and another. You can effectively model 2-steps random walks this way, which will allow you to find the communities for nodes of one type – and then of the other type, by repeating the process with a different starting point.



Classical label percolation community discovery can be tricky. The first reason is that synchronous label percolation has one problem known as a label oscillation. Figure 39.3 provides an example. After a few iterations, it might be that a community of nodes has a majority label on one side and a different majority label on the other side. The algorithm will be then stuck oscillating the labels at each time step. In such a scenario, it will never converge⁹.

One could solve the issue in various ways. First, one could simply use asynchronous updating. However, traditional asynchronous updating will update nodes in a random order. This might impact the stability of the resulting partition, because the order in which nodes are updated matters. Two subsequent runs of the algorithm could yield very different results. Moreover, it might impact convergence time, because there are many node orders which will still result in label oscillation. The most sure way to prevent label oscillation is to update first all nodes of one type and then all nodes of the other.

There are a few other ways to prevent oscillation. First, one could integrate label propagation with the modularity approach¹⁰. In this scenario, one doesn't run label propagation until convergence, but only for a few steps. Then they would refine the communities by maximizing bipartite modularity. Alternatively, one could put constraints on how we allow labels to propagate. For instance, we

Figure 39.3: (a) Communities from a synchronous label propagation at an hypothetical time t (the node color is the community label). Each node of circle type has a blue label majority in its neighborhood, and each triangle has a red majority. (b) At time $t + 1$ the labels oscillate according to the label majority at time t . The system will oscillate forever between (a) and (b) without converging.

⁹ Lovro Šubelj and Marko Bajec. Robust network community detection using balanced propagation. *The European Physical Journal B*, 81(3):353–362, 2011

¹⁰ Xin Liu and Tsuyoshi Murata. Community detection in large-scale bipartite networks. *Transactions of the Japanese Society for Artificial Intelligence*, 25(1):16–24, 2010

could force communities to be of comparable sizes in number of nodes or edges¹¹.

Finally, we can adapt stochastic blockmodels to the bipartite case, creating a biSBM¹². This has some similarities with the MMSB we saw for overlapping community discovery in Section 38.4. First, we don't look directly at the $|V_1| \times |V_2|$ biadjacency matrix B . It is more convenient to look at its adjacency matrix equivalent:

$$A = \begin{pmatrix} 0 & B \\ B^T & 0 \end{pmatrix}$$

The zeros on the main diagonal mean that nodes of the same type cannot connect to each other, enforcing the bipartite structure (see Section 8.1).

Then, just like in the overlapping case, we can have a special community-community matrix that tells us the probability of nodes in two distinct communities to connect to each other. The special condition here is that communities grouping nodes of the same type will have zero probability of connecting to each other, respecting the bipartite constraint.

Once we have these two special structures in place, one can proceed finding the most likely blockmodel that explains the observed data, which is the one with the best community partition, with the same strategies as in vanilla SBM. Note that this method can be trivially extended to multi-partite networks, modifying the fundamental structures accordingly.

The biSBM clusters the two modes separately, so you get a mixing matrix that tells you how the groups in the V_1 nodes interact with the groups in the V_2 nodes. In contrast, bipartite modularity and some other approaches will produce mixed groups, which contain nodes from both V_1 and V_2 . This makes biSBM a co-clustering method, like the ones we'll see in Section 39.4. The difference with those methods is that they find communities discovery via neighbor similarity, which is not the philosophy of biSBM.

A related method uses matrix factorization¹³. It starts by noticing that zeroes in A have different meanings. The zeroes in the main diagonal block represent impossible connections, connections that shouldn't be penalized. The zeroes in the off-diagonal block instead represent edges that *could* exist. Thus, the authors define a mask matrix M , with the same dimensions as A , with zeros on the main diagonal blocks and ones in the off diagonal blocks. By factorizing the product of M and A together with our best guess at the community organization of A , we obtain a function we can maximize to find the best community partition, knowing that we're only penalizing zeroes corresponding to connections that could exist.

¹¹ Michael J Barber and John W Clark. Detecting network communities by propagating labels under constraints. *Physical Review E*, 80(2):026129, 2009

¹² Daniel B Larremore, Aaron Clauset, and Abigail Z Jacobs. Efficiently inferring community structure in bipartite networks. *Physical Review E*, 90(1):012805, 2014

¹³ Zhong-Yuan Zhang and Yong-Yeol Ahn. Community detection in bipartite networks using weighted symmetric binary matrix factorization. *International Journal of Modern Physics C*, 26(09):1550096, 2015

Redefining the Clustering Coefficient

One reasonable way to find communities in unipartite networks is by exploiting the clustering coefficient. Nodes with high clustering coefficients are supposedly well embedded in a community, because all their neighbors are connected to each other. Thus, an algorithm cutting low local clustering coefficient areas could perform well. If you recall Section 37.1, one such strategy is to derive an edge clustering measure from the local node clustering, and then use it to determine which edges to cut to find a hierarchical community structure.

Our problem here is that the local clustering coefficient in bipartite networks is zero for all nodes. This is because in bipartite networks there cannot be triangles, which are at the basis of the computation of the local clustering coefficient (Section 12.2). A triangle, by definition, connects three nodes together. However, in bipartite networks, the edge closing the triangle cannot exist, because it would connect two nodes of the same type.

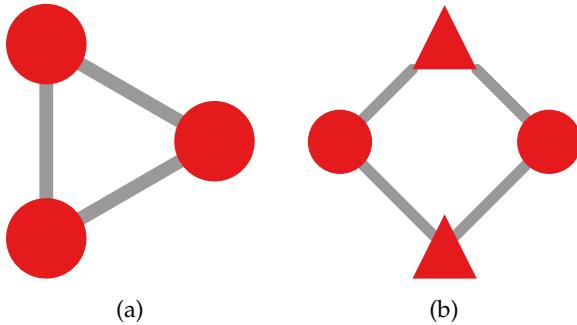


Figure 39.4: (a) The structure at the basis of the clustering coefficient of unipartite networks: the triangle. (b) Its equivalent in bipartite networks: the square.

This needs not to worry us. We can redefine the clustering coefficient to make sense in a bipartite network. In a unipartite network, the triangle is the smallest non-trivial cycle, the one that does not backtrack using the same edge, as you can see in Figure 39.4(a). We can also have a smallest non-trivial cycle in bipartite networks. It involves four nodes, as Figure 39.4(b) shows. So we can say that the local clustering coefficient of a node in a bipartite network is the number of times such cycles appear in its neighborhood, divided by the number of times they could appear given its degree¹⁴.

Let us assume that we want to know the local square clustering coefficient of node z . If we say that nodes u , v , and z are involved in s_{uvz} squares, then contribution of nodes u and v to the square clustering coefficient of z is:

$$C4_{u,v}(z) = \frac{s_{uvz}}{s_{uvz} + (k_u - \eta_{uvz}) + (k_v - \eta_{uvz})},$$

¹⁴ Peng Zhang, Jinliang Wang, Xiaojia Li, Menghui Li, Zengru Di, and Ying Fan. Clustering coefficient and community structure of bipartite networks. *Physica A: Statistical Mechanics and its Applications*, 387(27):6869–6875, 2008

with $\eta_{uvz} = 1 + s_{uvz}$. In practice, the number of possible squares (in the denominator) is the number of actual squares plus how many additional squares you could have given u 's and v 's free edges, edges not involved in any square. Here, u and v are the nodes of the same type.

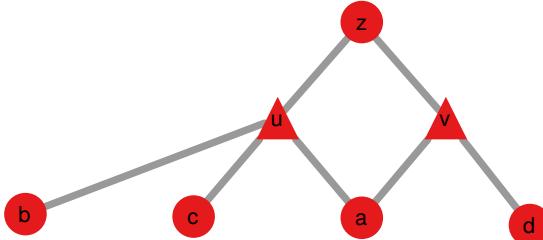


Figure 39.5: An example of bipartite network on which we can calculate the local clustering coefficient of node z .

Consider Figure 39.5. In the figure, $s_{uvz} = 1$, because nodes u , v , and z are involved in one square. As a consequence, $\eta_{uvz} = 2$. Since $k_u = 4$ and $k_v = 3$, we know that there could be $s_{uvz} + (k_u - \eta_{uvz}) + (k_v - \eta_{uvz}) = 1 + (4 - 2) + (3 - 2) = 4$ squares between the nodes: $uvza$ (which exists), $uvzb$, $uvzc$, and $uvzd$ (which do not exist). Since z has no additional neighbors, its local clustering coefficient is thus 1/4.

Once you have a properly defined local node clustering measure, you can use it to derive the corresponding edge clustering measure. Then, you can apply the same edge splitting algorithm we explored in the hierarchical community discovery case to find hierarchical bipartite communities.

39.4 Neighbor Similarity

A wholly different category of approaches tries to look at the adjacency matrix of a bipartite graph under a different perspective. The point of a community in a bipartite network is not that the nodes connect densely to each other, but that they connect to the same neighbors – nodes of the other type. Thus it's not much about internal density as it is about structural similarity (Section 15.2). Some approaches use some sort of common neighbor approach¹⁵.

Figure 39.6 shows a way to perform such a mental pivot. In a regular unipartite network – Figure 39.6(a) – we're looking at a community as a *set of nodes that connect to each other*. In a bipartite network – Figure 39.6(b) – we're looking at a community as a *set of nodes that connect to the same nodes*. Thus, in a bipartite community, we don't require two nodes that are part of the same community to connect to each other.

This is a fully valid definition of community that can be translated

¹⁵ Simone Daminelli, Josephine Maria Thomas, Claudio Durán, and Carlo Vittorio Cannistraci. Common neighbours and the local-community-paradigm for topological link prediction in bipartite networks. *New Journal of Physics*, 17(11):113037, 2015

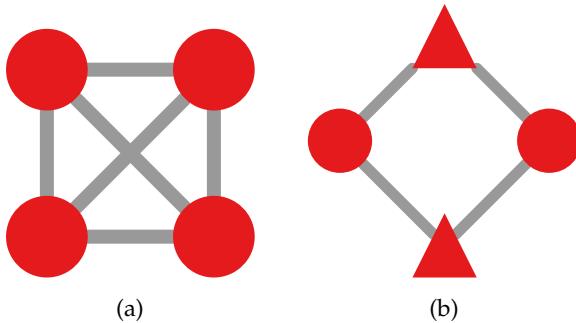


Figure 39.6: (a) A classical unipartite community. (b) A bipartite community.

to unipartite networks, which generates another way to look at the community discovery problem. I have showed this as a valid “community” of community discovery algorithms in Section 35.6.

The change in perspective might seem small at first, but it opens up a sea of possibilities. When you look at an adjacency matrix as a simple set of feature vectors, you can perform data clustering on it. Meaning that you can see a node as a point in a multidimensional space, a space with as many dimensions as there are nodes of the other type in the network. Then you can pick any of your favorite machine learning algorithms, spanning from k-means^{16,17,18} to dbscan¹⁹, and interpret their clusters as communities.

You should also be encouraged to look at bi-clustering (or co-clustering) techniques^{20,21,22}. The advantage of such techniques is that they will cluster the rows and the columns of your matrix at the same time. In this way, you don’t have to manually map the clusters you found by looking at the $|V_1| \times |V_2|$ matrix with the ones you found in the $|V_2| \times |V_1|$ matrix.

The way these methods find clusters is usually by estimating the pairwise distance (Euclidean or otherwise) between data points. Then clusters are sets of points that lump together in this complex space. Hopefully, boundaries between clusters are clear, as few points are equidistant from multiple cluster centers. In community discovery, your data points are the nodes. Figure 39.7 shows an example.

Besides old data clustering techniques, there is a lot of excitement for the application of neural network approaches to community discovery²³. However, usually, adjacency matrices are too sparse and constrained to provide a proper input to neural networks. Thus, most of the deep learning attacks to community detection use more sophisticated representations of the relations between nodes in the graph, in the form of graph embeddings, which we’ll explore more in depth later on (in Chapter 42).

To wrap up this chapter, let’s recall again our definition of communities in complex networks:

¹⁶ Hugo Steinhaus. Sur la division des corp matériels en parties. *Bull. Acad. Polon. Sci.*, 1(804):801, 1956

¹⁷ James MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Berkeley symp on math statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA, 1967

¹⁸ Stuart Lloyd. Least squares quantization in pcm. *IEEE transactions on information theory*, 28(2):129–137, 1982

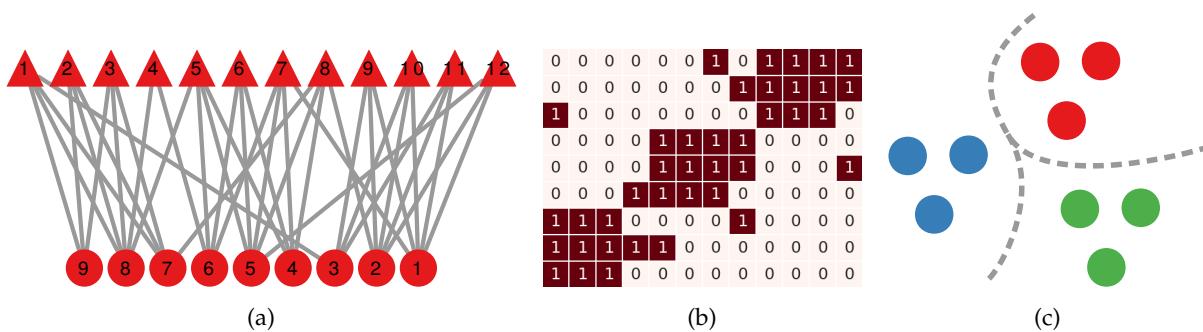
¹⁹ Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231, 1996

²⁰ Inderjit S Dhillon. Co-clustering documents and words using bipartite spectral graph partitioning. In *SIGKDD*, pages 269–274, 2001

²¹ Inderjit S Dhillon, Subramanyam Mallela, and Dharmendra S Modha. Information-theoretic co-clustering. In *SIGKDD*, pages 89–98, 2003

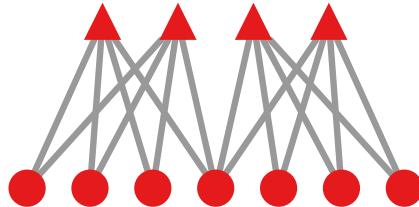
²² Yuval Kluger, Ronen Basri, Joseph T Chang, and Mark Gerstein. Spectral biclustering of microarray data: coclustering genes and conditions. *Genome research*, 13(4):703–716, 2003

²³ Joan Bruna and X Li. Community detection with graph neural networks. *stat*, 1050:27, 2017



Communities are groups of nodes densely connected to each other and sparsely connected to nodes outside the community.

The bipartite community discovery introduces another issue with the standard definition of community based on density. On the one hand, nodes of the same type cannot connect in a bipartite graph – so they have density of zero –, but they can and will be part of the same community. Figure 39.8 has many missing links, but it is still a valid bipartite community. On the other hand, we are looking at a community definition that is more based on neighborhood similarity than on internal density. So again this criterion of internal density is a bit flaky.



39.5 Summary

1. In bipartite networks, nodes of the same type cannot connect to each other. However, they could still be in the same community, because they have lots of common neighbors. Thus we need to adapt modularity and other community quality measures to take this into account.
2. One way to perform bipartite community discovery is by projecting the bipartite network into unipartite form and then perform community discovery there. However, the resulting network will be too dense and we will lose information in the projection.
3. We can adapt clique percolation by percolating bi-cliques. We can perform random walks by making them perform two steps

Figure 39.7: (a) A bipartite network. (b) Its adjacency matrix. (c) A 2D spatial representation of the circular nodes, using their adjacencies to determine the position. Node color is its cluster, as identified by spatial clustering (dashed line).

Figure 39.8: A possible bipartite community.

at a time. We can also adapt label propagation by performing it asynchronously and refining its results.

4. We can also redefine the local clustering coefficient, so that it is based not on the number of triangles around a node (which cannot exist in a bipartite network), but on the number of its surrounding squares.
5. Alternatively, one could simply infer node similarity measures by looking at their neighbors, or perform simple data clustering. However, all these strategies show how our classical definition of communities in networks cannot really capture the bipartite case.

39.6 Exercises

1. The network at <http://www.networkatlas.eu/exercises/39/1/data.txt> is bipartite. Project it into unipartite and find five communities with the Girvan-Newman edge betweenness algorithm (repeat for both node types, so you find a total of ten communities). What is the NMI with the partition proposed at <http://www.networkatlas.eu/exercises/39/1/nodes.txt>?
2. Now perform asynchronous label propagation directly on the bipartite structure. Calculate the NMI with the ground truth. Since asynchronous label propagation is randomized, take the average of ten runs. Do you get a higher NMI?
3. Consider the bi-adjacency matrix as a data table and perform bi-clustering on it, using any bi-clustering algorithm provided in the `scikit-learn` library. Do you get a higher NMI than in the previous two cases?

40

Multilayer Community Discovery

The last chapter of community discovery, at least for this book, focuses on multilayer networks. In multilayer networks, nodes can belong to different layers and thus they can connect for different reasons. In multilayer networks we want to find communities that span across layers. For example, we want to figure out communities of friends even if your friends are spread across multiple social media platforms.

There are a few review works you can check out to have a more in-depth exploration of the topic^{1,2}. Here, I go over briefly the main approaches and peculiar problems of community discovery in multilayer networks.

40.1 Flattening

Similar to the bipartite case, there is a relatively simple solution. You can flatten the network by collapsing nodes across layers³, meaning that you reduce the multilayer network to a network with a single layer and weighted edges. The weights on the edges depend on the multilayer connections. In practice, you're collapsing a qualitative information – in which layer a connection appears – into a quantitative one – an edge weight. This assumes that every edge type is equally important. Then you can perform a normal mono-layer community discovery. Figure 40.1 shows an example.

There are a few choices for your edge weights. The simplest one could be to simply count the number of layers in which the connection between the nodes appear. However, you might want to take into account some interplay between the layers. For instance, you can count the number of common neighbors that two nodes have and use that as the weight of the layer, under the assumption that a layer where two nodes have many common neighbors should count for more when discovering communities. Or you could use “differential flattening⁴”: flatten the multilayer graph into the single

¹ Jungeun Kim and Jae-Gil Lee. Community detection in multi-layer graphs: A survey. *ACM SIGMOD Record*, 44(3):37–48, 2015

² Obaida Hanteer, Roberto Interdonato, Matteo Magnani, Andrea Tagarelli, and Luca Rossi. Community detection in multiplex networks, 2019

³ Michele Berlingario, Michele Coscia, and Fosca Giannotti. Finding and characterizing communities in multidimensional networks. In *2011 International Conference on Advances in Social Networks Analysis and Mining*, pages 490–494. IEEE, 2011a

⁴ Jungeun Kim, Jae-Gil Lee, and Sungsu Lim. Differential flattening: A novel framework for community detection in multi-layer graphs. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 8(2):27, 2017

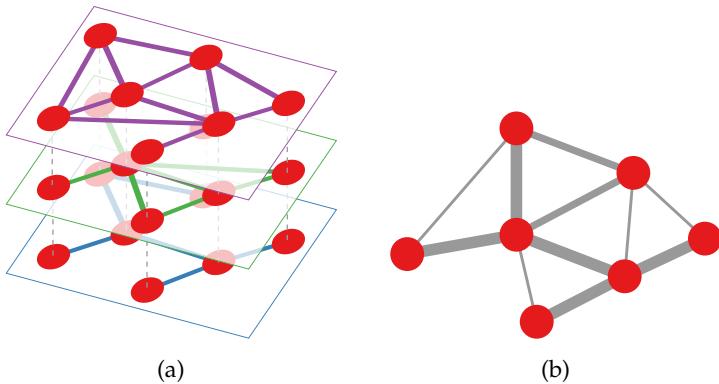


Figure 40.1: (a) A multilayer network. (b) A weighted flattening of (a). An edge's width is proportional to its weight.

layer version of it such that its clustering coefficient is maximized.

As with the bipartite case, simplistic solutions create problems. The issue is the same: we lose information. When we translate a layer into a weight, we don't know any more which type of links we're looking at. Some types of links might contribute differently to the communities. Moreover, flattening is not always possible, or at least not straightforward. If a node in one layer is coupled with multiple nodes in another layer, how do we represent it in the flattened network?

40.2 Layer by Layer

There is another solution that is slightly more sophisticated than flattening a multilayer network but that at the same time doesn't require you to go fully multidimensional in your analysis. It is performing community discovery separately on each layer of your network and then somehow combine your results. This is sort of like the approach of dynamic community discovery where you perform your detection on each snapshot of the network and then you aggregate the results (Section 35.4).

In fact, one could see each snapshot of the network as a layer – or vice versa: each layer as a snapshot. Thus anything we do on an evolving network we could also do on a multilayer one. And – why not? – we could even have *evolving* multilayer networks, putting the two together⁵. The solution is not ideal, though, because there are many assumptions you have on a dynamic networks that you might break on a multilayer network – and vice versa. For instance, in a dynamic graph you have some sort of continuity assumption: one snapshot should be, in principle, similar to the next. There is no requirement of similarity between layers: in fact, they can even be strongly anti-correlated.

For these reasons, you need some specialized community discovery approaches. In one approach, one could build a matrix where

⁵ Marya Bazzi, Mason A Porter, Stacy Williams, Mark McDonald, Daniel J Fenn, and Sam D Howison. Community detection in temporal multilayer networks, with an application to correlation networks. *Multiscale Modeling & Simulation*, 14(1):1–41, 2016

each row is a node and each column is the partition assignment for that node in a specific layer⁶. This is then a $|V| \times |C|$ matrix. Then one could perform kMeans on it, finding clusters of nodes that tend to be clustered in the same communities across layers.

A similar approach⁷ uses frequent pattern mining, a topic we'll see more in depth in Section 41.4. For now, suffice to say that we again perform community discovery on each layer separately. Each node can then be represented as a simple list of community affiliations. We then look for sets of communities that are frequently together: these are communities sharing nodes across layers.

| Node | L1 | L2 | L3 |
|------|------|------|------|
| 1 | C1L1 | C1L2 | C1L3 |
| 2 | C1L1 | C1L2 | C1L3 |
| 3 | C1L1 | C1L2 | C1L3 |
| 4 | C2L1 | C1L2 | C1L3 |
| 5 | C2L1 | C1L2 | C2L3 |
| 6 | C2L1 | C1L2 | C3L3 |
| 7 | C1L1 | C2L2 | C3L3 |
| 8 | C2L1 | C2L2 | C3L3 |
| 9 | C2L1 | C2L2 | C3L3 |
| 10 | C1L1 | C2L2 | C2L3 |

(a)

| MLComm | SLComms |
|--------|------------------|
| MLC1 | C1L1, C1L2, C1L3 |
| MLC2 | C2L1, C1L2 |
| MLC3 | C2L2, C3L3 |

(b)

⁶ Lei Tang, Xufei Wang, and Huan Liu. Community detection via heterogeneous interaction analysis. *Data mining and knowledge discovery*, 25(1):1–33, 2012

⁷ Michele Berlingero, Fabio Pinelli, and Francesco Calabrese. Abacus: frequent pattern mining-based community discovery in multidimensional networks. *Data Mining and Knowledge Discovery*, 27(3):294–320, 2013c

| MLComm | Nodes |
|--------|---------|
| MLC1 | 1, 2, 3 |
| MLC2 | 4, 5, 6 |
| MLC3 | 7, 8, 9 |

(c)

Figure 40.2 shows an example. In Figure 40.2(a) we have the communities found for each layer for each node. Then we decide that we want to merge communities if they have at least three nodes in common, i.e. they appear in at least three rows of the table.

Figure 40.2(b) shows the multilayer communities mapping and there are many interesting things happening. First, we only want maximal sets, meaning that we aren't interested in returning C1L1 by itself if we also find it in a larger set of communities. Second, we are ok if a community gets merged in different sets – i.e. the multilayer communities can overlap –: C1L2 is part of two maximal sets, MLC1 and MLC2. Figure 40.2(c) shows the final output: the multilayer community affiliation. A node is part of a multidimensional community if it is part of all communities composing it.

Node 10 is an example of a final interesting thing: it is part of no multidimensional community because its affiliation is a weird combination of communities. We can decide to let it be without community affiliation, or to allow it to be part only of its non-multilayer communities.

There are other algorithms solving the same problem and inspired by frequent pattern mining^{8,9}.

Figure 40.2: (a) The communities found in each layer of each node. (b) The merged multilayer communities. (c) The final node-community affiliation.

⁸ Arlei Silva, Wagner Meira Jr, and Mohammed J Zaki. Mining attribute-structure correlated patterns in large attributed graphs. *Proceedings of the VLDB Endowment*, 5(5):466–477, 2012

⁹ Zhiping Zeng, Jianyong Wang, Lizhu Zhou, and George Karypis. Coherent closed quasi-clique discovery from large dense graph databases. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 797–802. ACM, 2006

The last solution for this section is inspired by ensemble clustering. Again, we have a community per layer. Then we use the same strategy I outlined in Section 35.6: we consider each community partition in each layer as a valid clustering of the same underlying relationship via different datasets¹⁰. We then find the “true” clustering, which is the partition that is the closest one to the combination of all partitions.

Aggregating communities across layers has some benefits. For instance, it might solve the resolution problem of modularity¹¹ that I discussed in Section 36.1. However, all these methods have the downside of relying more or less on the same assumption: that the layers are correlated to each other. While this might not be a bad assumption to start with¹², disassortative layers exist and might represent a problem.

40.3 Multilayer Adaptations

Multilayer Modularity

I already mentioned how obsessed networks scientists are with modularity, so you know what’s coming next: multilayer modularity¹³. Suppose we’re using the Louvain method, which grows communities node by node. If we found a triangle in a layer, can we extend it by taking a node in a different layer? Intuitively yes, the edge should count because the node is the same. However, if we were to represent this as a flat network, the new node is not densely connected to the rest of the triangle: a node couples only with itself, not with its community fellows. So the coupling edges have to count in some special way.

In practice, standard modularity works well in each layer separately. Consider Figure 40.3: in modularity, the part testing for the density of the community is $A_{uv} - \frac{k_u k_v}{2|E|}$. If we use this same part for the inter-layer coupling, we would end up with a case in which the community cannot be expanded across layers, because there are only sparse connections between layers. A node couples only with itself in a different layer, not connecting to its community members, making a multi-layer community sparser than it actually is. So we need to add something that will allow us to count the coupling links, so that we don’t end up with the trivial result of all mono-layer communities.

The full formulation of multilayer modularity is the following:

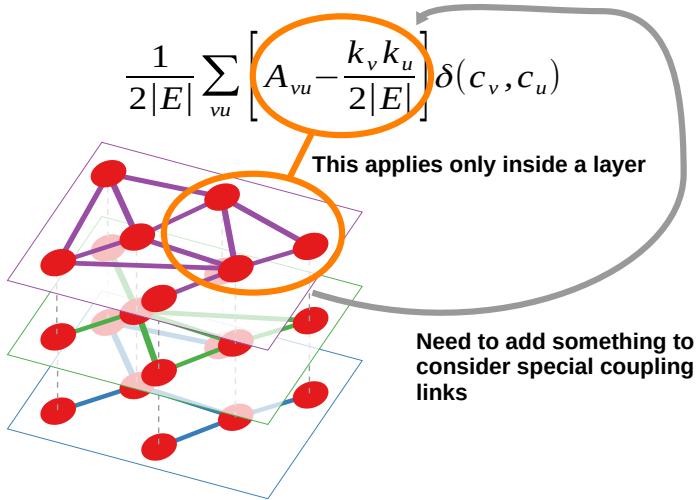
$$\frac{1}{2(|E| + |C|)} \sum_{vusr} \left[\left(A_{vus} - \gamma_s \frac{k_{vs} k_{us}}{2|E_s|} \right) \delta_{sr} + C_{vsr} \delta_{uv} \right] \delta(c_{us}, c_{vr}).$$

¹⁰ Andrea Tagarelli, Alessia Amelio, and Francesco Gullo. Ensemble-based community detection in multilayer networks. *Data Mining and Knowledge Discovery*, 31(5):1506–1543, 2017

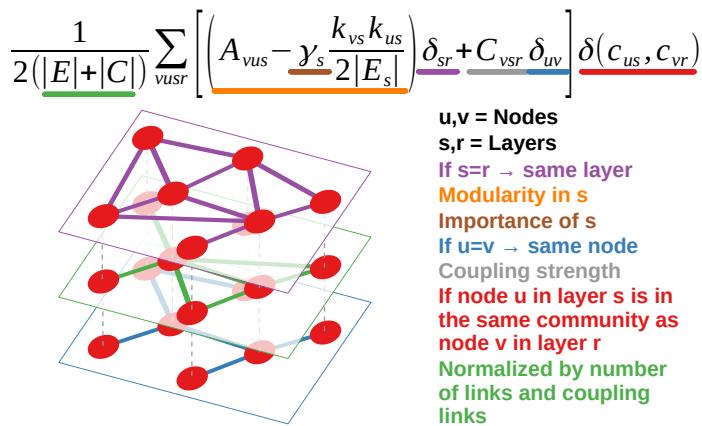
¹¹ Dane Taylor, Saray Shai, Natalie Stanley, and Peter J Mucha. Enhanced detectability of community structure in multilayer networks through layer aggregation. *Physical review letters*, 116(22):228301, 2016

¹² Desislava Hristova, Mirco Musolesi, and Cecilia Mascolo. Keep your friends close and your facebook friends closer: A multiplex network approach to the analysis of offline and online social ties. In *Eighth International AAAI Conference on Weblogs and Social Media*, 2014

¹³ Peter J Mucha, Thomas Richardson, Kevin Macon, Mason A Porter, and Jukka-Pekka Onnela. Community structure in time-dependent, multiscale, and multiplex networks. *science*, 328(5980):876–878, 2010



Let's break it down – and you can check Figure 40.4 for a graphical representation and you can always go back to Section 36.1 to read more about the notation of classical modularity and compare it with this formulation. E and C are the sets of (intra-layer) edges and (inter-layer) couplings. A_{vus} is our multilayer adjacency tensor, it is equal to 1 if nodes u and v are connected in layer s , and it is 0 otherwise. k_{us} and k_{vs} are the degrees of u and v in layer s , respectively. $|E_s|$ is the number of edges in s .



The major complication in multilayer modularity is that we have many Kronecker deltas (δ). The first is δ_{sr} : its role is to make sure that standard modularity is applied inside a layer (when $s = r$, $\delta_{sr} = 1$, because s and r are the same layer). The second is δ_{uv} and it checks whether we are looking at two nodes that are coupled across layers: δ_{uv} is 1 if $u = v$. Note that δ_{sr} and δ_{uv} are mutually exclusive. If $s = r$ we are in the same layer and so it must be that $u \neq v$, because we don't have self loops. On the other hand, if $u = v$ then

Figure 40.3: The issues in applying modularity to the multilayer case. The part looking at the community density only applies inside a single layer, thus we need to add something to it to consider the special coupling edges.

Figure 40.4: The adaptation of modularity to the multilayer setting. Each part of the formula is underlined with a color corresponding to its interpretation.

$s \neq r$, because by definition coupling connections go across layers, thus s and r must refer to different layers. Both deltas can be zero if we're looking at uncoupled nodes in different layers. The final delta, $\delta(c_{us}, c_{vr})$ is the same as in standard modularity, it is equal to 1 only if we are looking at nodes inside the same community, i.e. $c_{us} = c_{vr}$.

With γ_s we can regulate how important each layer s is for the community. In practice, γ is a vector of weights, one per layer s of the network.

C_{vsr} is the strength of the coupling link, which is a parameter just like γ is: you can decide how strong the layer couplings should be. It matters only when we're looking at the same node connected by a coupling link across layer ($u = v$), and so δ_{uv} is 1. In this case, nothing else matters, because δ_{sr} is 0 (because $s \neq r$), so the standard modularity part cancels out.

Just like in standard modularity, only nodes in the same community contribute to the sum, so when node u in layer s is in the same community as node v in layer r (meaning that $\delta(c_{us}, c_{vr}) = 1$). This is normalized by the number of edges ($|E|$) across all layers plus all coupling links ($|C|$).

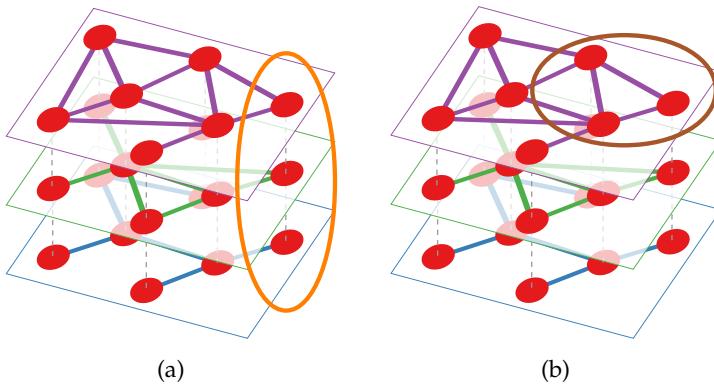


Figure 40.5: The effect of the C_{vsr} parameter on multilayer modularity. (a) High values imply pillar communities. (b) Low values imply flat communities (right).

If you decide that your inter layer couplings are very strong, you'll end up with "pillar communities" where nodes tend to favor grouping with themselves across layers: the inter layer couplings trump any intra-layer regular edge. If your inter layer couplings are weak (low C_{vsr}) then you'll end up with "flat communities" as nodes prefer to group with other nodes in the same layer. I show an example in Figure 40.5.

Instead, γ allows you to indicate some layers as more important than others, as I show in Figure 40.6. If the purple layer is more important than the green one, multilayer modularity will group in the community a node that is not connected with the two nodes in the green layer. If we flip the γ values to make green more important than purple, the situation is reversed, and modularity will return

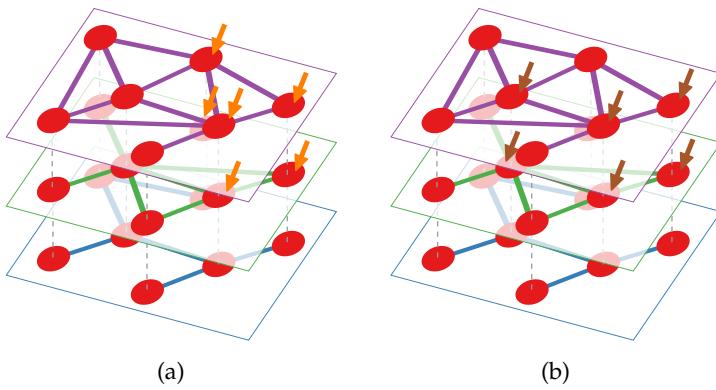


Figure 40.6: The effect of the γ_s parameter on multilayer modularity. (a) High γ for the top (purple) community and low for the mid (green) community. (b) Low γ for the top (purple) community and high for the mid (green) community.

different communities.

An alternative way to adapt modularity maximization to multilayer networks is to adapt the Louvain algorithm (see Section 37.1) to handle networks with multiple relation types¹⁴.

Other Approaches

Modularity is not the only algorithm that can be adapted to multilayer networks. Following the same strategy we applied for bipartite networks, we can adapt the kitchen sink of community discovery to multilayer structures.

The random walks approach can be multilayer^{15,16}. In practice, we have a random walker that normally selects edges in the same layer, but it has a special rule that sometimes allows it to go through a coupling edge, and then resume its normal random walk. Figure 40.7 shows an example of such a move. Many ways have been proposed to expand random walks to multilayer networks, and a special set of algorithms focuses on local-first community discovery. We center our focus on a specific (set of) query nodes and we find the communities surrounding them^{17,18}.

Similarly, one could propagate labels across layers with special

¹⁴ Inderjit S Jutla, Lucas GS Jeub, and Peter J Mucha. A generalized louvain method for community detection implemented in matlab. URL <http://netwiki.amath.unc.edu/GenLouvain>, 2011

¹⁵ Zhana Kuncheva and Giovanni Montana. Community detection in multiplex networks using locally adaptive random walks. In ASONAM, pages 1308–1315. ACM, 2015

¹⁶ Manlio De Domenico, Andrea Lancichinetti, Alex Arenas, and Martin Rosvall. Identifying modular flows on multilayer networks reveals highly overlapping organization in interconnected systems. *Physical Review X*, 5(1):011027, 2015a

¹⁷ Lucas GS Jeub, Michael W Mahoney, Peter J Mucha, and Mason A Porter. A local perspective on community structure in multilayer networks. *Network Science*, 5(2):144–163, 2017

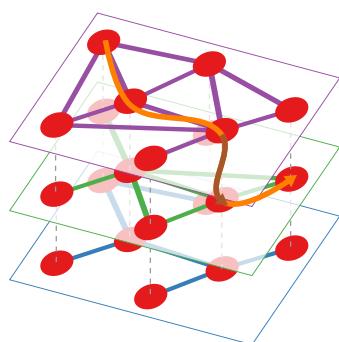
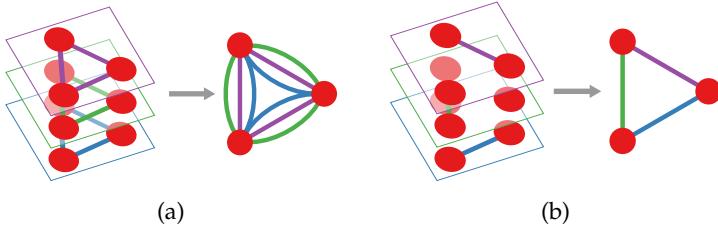


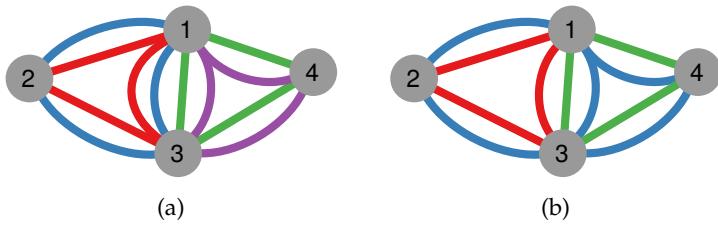
Figure 40.7: In multilayer random walk community discovery, the random walker (orange) has a certain probability to perform a layer jump (brown).

rules¹⁹, and thus adapt the fast label propagation algorithm to find multilayer communities. First, we cannot use synchronous label propagation: like in the bipartite case, also for multilayer networks we could be stuck with label oscillation (Section 39.3), this time across layers. Second, the authors define a quality function that regulates the propagation of labels. This is done because there might be layers that are relevant for a community and layers that are not. We do not want a community, which is very strong in some layers, to “evaporate away” just because in most layers the nodes are not related.

Next on the menu is k-clique percolation²⁰. In this scenario, we need to redefine a couple of concepts, particularly what a clique is in a multilayer network, and how we determine when two multiplex cliques are adjacent. For the first case, we need to talk about k-l-cliques: a set of k nodes all connected through a specific set of l layers. Moreover, there are two ways for nodes to be all connected via the layers: all pairs of nodes could be connected in all layers at the same time, or they could be connected in only one layer at a time. The first type of clique is an k-l-AND-clique, the second type is a k-l-OR-clique. Figure 40.8 shows an example.



It becomes clear that two k-l-cliques might share $(k - 1)$ nodes across different layers. In such a case, we need some care in defining a parameter to regulate percolation. We need a minimum number m of shared layers to allow the percolation. If the two cliques do not share at least m layers, even if they share $k - 1$ nodes they are not considered adjacent. Figure 40.9 shows an example.



The final adaptation we consider is the stochastic blockmodels^{21,22}. Just like we saw for overlapping and bipartite SBMs, we need to add an additional matrix into our expectation maximization frame-

¹⁸ Roberto Interdonato, Andrea Tagarelli, Dino Ienco, Arnaud Sallaberry, and Pascal Poncelet. Local community detection in multilayer networks. *DMKD*, 31(5):1444–1479, 2017

¹⁹ Oualid Boujemai and Mohamed Bouguessa. Mining community structures in multidimensional networks. *TKDD*, 11(4):51, 2017

²⁰ Nazanin Afsarmanesh and Matteo Magnani. Finding overlapping communities in multiplex networks. *arXiv preprint arXiv:1602.03746*, 2016

Figure 40.8: (a) A 3-3-AND-clique. (b) A 3-3-OR-clique.

²¹ Caterina De Bacco, Eleanor A Power, Daniel B Larremore, and Cristopher Moore. Community detection, link prediction, and layer interdependence in multilayer networks. *Physical Review E*, 95(4):042317, 2017

²² Natalie Stanley, Saray Shai, Dane Taylor, and Peter J Mucha. Clustering network layers with the strata multilayer stochastic block model. *IEEE transactions on network science and engineering*, 3(2):95–105, 2016

Figure 40.9: Two 2,2-cliques sharing a 1,2-clique. The edge color represents the edge layer. If $m = 1$ (a) does NOT percolate because the rightmost clique does not share a layer with the leftmost clique; (b) DOES percolate, since the two cliques share the blue layer.

work. For overlapping and hierarchical SBMs it was a community-community matrix telling us how strongly communities connect to each other. In this case, instead, we have a layer-layer matrix telling how likely it is for two layers to have the same edges.

This is neat, because it allows us to model assortative, disassortative, and non-assortative layer relationships. In the first case, being connected in a layer increases the chances of being connected in the other layer: it is more likely to be friends on Facebook if we are also friends in real life. The second case is the opposite: a relation in one layer makes it harder to be connected on another: if you attack me in an online game it's less likely that we'll be friend. The non-assortative case covers the scenario in which being connected in a layer gives us no information on whether we're connected in the other.

All these cases – the special layer-layer jump probability in random walks, the quality function in label propagation, and the layer-layer probability matrix in SBM – allow us to select the relevant layers for a multilayer community. Thus, we can keep a node group together even if in many layers the nodes don't connect to each other, probably because those layers were disassortative with the layers in which the community appears.

Still an adaptation of already existing methods, but more properly multilayer, is the approach of multilink similarity²³. This is heavily inspired by the hierarchical link clustering we saw in the overlapping case (Section 38.5). The objective is the same: to define a link-link similarity measure that we can use to progressively merge link communities in a hierarchical fashion.

The similarity measure for multilayer networks is:

$$S_{(u,k),(v,k)} = \epsilon z^{\beta_{uk,vk}} + (1 - \epsilon) |P_{uv\bar{k},2}|.$$

Here, ϵ and z are parameters between 0 and 1 you can set. The real work is made by $\beta_{uk,vk}$ and P . $\beta_{uk,vk}$ takes values between 0 and 1, and it is one minus the share of layers in common connecting uk and vk . So, for instance, if the node pairs uk and vk connect in mutually exclusive layers – i.e. no layers in common – then $\beta_{uk,vk} = 1$. On the other hand, if they connect in exactly the same layers and no other layer, then $\beta_{uk,vk} = 0$. Since our parameter z is between 0 and 1, the whole term tells us how much we weight in the link-link similarity the absence of layers, because mutually exclusive layer set will just be $z^1 = z$.

$|P_{uv\bar{k},2}|$ is instead the count of paths of length 2 between u and v that do not pass through k , plus the number of layers in which u and v connect directly to each other. This is normalized by the lowest degree between u and v , excluding all connections going to k . Thus,

²³ Raul J Mondragon, Jacopo Iacovacci, and Ginestra Bianconi. Multilink communities of multiplex networks. *PLoS one*, 13(3):e0193821, 2018

in practice, the parameter ϵ regulates the weight we want to give to the number of the shared layers of edges uk and vk , over the local multilayer clustering of nodes u and v . If $\epsilon = 1$ we only care about clustering together node pairs that connect through the same sets of layers.

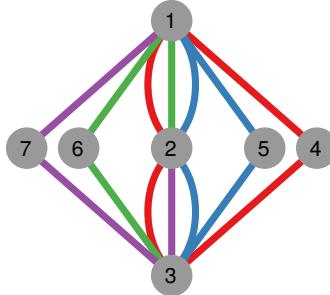


Figure 40.10: A multilayer network, with the edge color encoding the layer in which it appears.

Let us consider Figure 40.10 and attempt to estimate the similarity of node pairs $(1, 2)$ and $(1, 3)$. The pairs share two out of four possible layers, thus $\beta_{uk,vk} = 0.5$. There are also four other paths going from 1 to 3 that do not use node 2. Both node 1 and 3 have degree equal to four – remember we don't count the connections going through node 2 –, thus $|P_{uv\bar{k},2}| = 4/4 = 1$. If we were to set $z = 0.6$ and $\epsilon = 0.4$, then $S_{(u,k),(v,k)} = (0.4 \times 0.6^{0.5}) + ((1 - 0.4) \times 1) \sim 0.91$.

Other adaptations I'm not going to discuss in details are an extension of local community discovery to multilayer network²⁴. This is based on redefining the simple concepts of degree and neighborhood for the multilayer case, and then apply a classical local exploration approach, as the one we saw in Section 35.5.

Another class of solutions include representing multilayer graphs in a lower dimensional space with a technique known as “Grassmann manifold”²⁵.

40.4 Multilayer Density

We have talked about how to find communities in multilayer networks. Implicitly, we're resting on our definition, which is based on density. But what actually *is* multilayer density? This turns out to be an ambiguous concept.

Let's go back to Figure 40.8 for a moment. Is a group of nodes “multilayer densely” connected when they are connected in all layers (Figure 40.8(a))? Or is it that you need to look at all connections across layers (Figure 40.8(b))? To use a different perspective, let's represent multilayer networks with a labeled multigraph. In our first example, we have a multigraph with connections in all labels. In the second example we sill have a triangle, so the multigraph is

²⁴ Manel Hmimida and Rushed Kanawati. Community detection in multiplex networks: A seed-centric approach. *NHM*, 10(1):71–85, 2015

²⁵ Xiaowen Dong, Pascal Frossard, Pierre Vandergheynst, and Nikolai Nefedov. Clustering on multi-layer graphs via subspace analysis on grassmann manifolds. *IEEE Transactions on signal processing*, 62(4):905–918, 2013

dense. But the two concepts are not the same. Which of the two are we looking at?

This is another case when one has to make their own judgment. The answer depends on the type of analysis, and the type of data, you are looking at. In some cases, you want connections in all layers. In some others, you are ok with looking at all layers to find communities. You cannot rely on a fixed definition of communities based on density, because it cannot apply to all scenarios.

Thus, you need to have measures to determine when you are in one case and when you are in another. I proposed a couple in a paper of mine²⁶. We decided to call them “redundancy” and “complementarity”. Note that this redundancy here has nothing to do with the cousin of the local clustering coefficient we talked about in Section 12.2.

Redundancy is the easiest of the two. To consider a set of nodes to be densely connected in a multilayer network, we require that the edges appear in *all* the layers we are interested in. If we have a community c containing a set of nodes, and we test it over the set of layers L , redundancy is simply the share of actual edges over all the edges we would need to connect every pair of nodes through every layer:

$$\rho_c = \sum_{u,v \in P_c} \frac{|\{l : \exists(u,v,l) \in E\}|}{|L| \times P_c},$$

where P_c is the set of all node pairs in c .

Complementarity is a little trickier, because it is the intersection of three concepts: variety, exclusivity, and homogeneity. Variety is through how many different layers nodes in community c connect to each other. This is simply the number of layers in c divided by the number of layers in the network: $\frac{|L_c| - 1}{|L| - 1}$. Exclusivity counts how many pairs of nodes connect in just one layer within c : if $\bar{P}_{c,l}$ is the number of node pairs in c which connect only in layer l , the exclusivity is $\frac{\sum_{l \in L} \bar{P}_{c,l}}{|P_c|}$. Finally, homogeneity estimates how uniformly the edges in c distribute across layers, which is $1 - \frac{\sigma_c}{\sigma_c^{\max}}$. Here, σ_c is the standard deviation of the distribution of the edges in c across layers, and it is normalized by its theoretical maximum.

Let's see some examples to put all these Greek letters to good use. Let's consider Figure 40.11, assuming that the network has a total of three layers. In Figure 40.11(a) we have a high redundancy case. Since the community includes all layers of the network, the numerator of redundancy is simply the count of edges: 18. The denominator is 3 (the number of layers) times the number of node

²⁶ Michele Berlingario, Michele Coscia, and Fosca Giannotti. Finding redundant and complementary communities in multidimensional networks. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, pages 2181–2184, 2011b

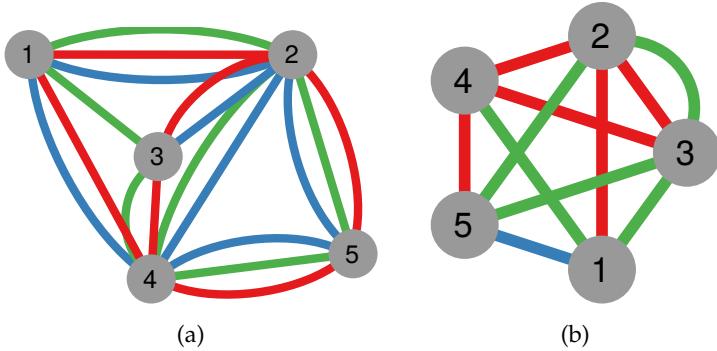


Figure 40.11: Two examples of different types of multilayer density. The edge color encodes the layer in which the edge appears. (a) A high redundancy case. (b) A high complementarity case.

pairs in the community, which is 10, since we have 5 nodes. Thus the redundancy is $18/30 = 0.6$.

Figure 40.11(b) is instead a high complementary case. Variety is 1 by definition, since the community contains all layers of the network. Exclusivity is $9/10$, because there is one pair of nodes (nodes 2 and 3) which is connected in two layers, and thus it is not counted. Finally, the standard deviation of the distribution of the edges in c across layers is the standard deviation of the vector $[5, 1, 5]$, since there are five edges in the red and green layer, and only one in the blue layer. This is ~ 1.88 , which is exactly two thirds of the maximum possible, leaving us with a total homogeneity of 0.33. Thus, complementarity is $1 \times 0.9 \times 0.33 = 0.297$, penalized by the low representation of the blue layer in the community.

40.5 Mopping Up Community Discovery

We have finally reached the end of this extremely simplified trip through community discovery. It all started with a simple, nothing-up-my-sleeve definition of what communities are in complex networks:

Communities are groups of nodes densely connected to each other and sparsely connected to nodes outside the community.

Yet, as we progressed in this journey, we realized that there is a gazillion ways in which such definition needs to be stretched, it is not the full story, or simply does not work. To sum up our list of grievances:

- The standard definition implies assortative communities: nodes in the same communities tend to connect to each other more than random. However, disassortative communities are a thing as well: nodes in a disassortative community tend to connect to nodes in different communities (Section 35.1).

- If our network is evolving, communities are evolving too. The information from past communities should be taken somehow into account, making the communities at time $t + 1$ a compromise between their density and their similarity with the communities at time t (Section 35.4).
- Communities can be local (Section 35.5), meaning that we might prevent ourselves from discovering all members of a community and thus leaving out some parts of the network that would make the community denser.
- Measures defined with the idea of maximizing density and external sparsity have counter-intuitive behavior, for instance modularity has resolution limit, degeneracy of good solutions, and limits in its field of vision (Section 36.1).
- We want communities to be interpretable and/or to tell us something about the real world properties of the entities we are grouping. This can be achieved by finding the communities maximizing the normalized mutual information of some other data we have about nodes. While this is a worthwhile task for many real world applications, it can – and most often does – clash with the internal density requirement (Section 36.4). This is because node “meta”data is just data, and it doesn’t necessarily have any relevance to the edge creation process of your network.
- Many networks have a hierarchical community structure, where we can find communities of communities. However, by definition, these communities must be more sparsely connected than the communities forming them. This should not make them any less valid, as they are a useful tool to explain many natural structures (Chapter 37).
- It is equally a fact that many real world networks have overlapping communities: nodes can be part of multiple communities at the same time. But if it is true that the more communities two nodes have in common the more likely they are to connect, we end up with networks in which the overlap of communities is denser than the communities themselves, which contradicts our original definition (Chapter 38).
- We can find communities in bipartite networks as well. However, by definition, these communities will be somewhat sparse, given that we forbid connections between nodes of the same type. We need to modify our definition of density accordingly (Chapter 39).
- However, there are proper ways to find bipartite communities – and even regular unipartite communities – by adapting classical

data clustering algorithms from data mining. These algorithms will simply take as input the adjacency matrix of the graph as if it was an attribute table. The meaning of the communities found this way would change, though: these are not any more nodes densely connected to each other, but rather nodes connected to the same neighbors (Section 39.4).

- Finally, as we saw in this chapter, we need to adapt density to the multilayer case as well. However, this is necessarily an ambiguous operation with multiple valid alternatives, as multilayer density can be intended both in a “redundancy” and in a “complementary” sense.

The moral of this story is that you can intend “communities” in complex networks in a thousand different ways. Performing community discovery is not a neutral operation you would do like adding two numbers. It is a complex problem that starts from the question: what is a community *for me*? Or for *my data*? What am I *actually* looking for? Just picking an algorithm because someone tells you so, or because it is the most used, is guaranteed to misfire.

This is also why, if you encounter somebody who uncritically tells you the classical definition or uses it in their paper without at least a mention of these caveats, you should tell them they’re wrong, otherwise Michele would have written these 100 pages for nothing.

40.6 Summary

1. Multilayer community discovery is the adaptation of community discovery for networks with multiple edge types. The simplest approach is to flatten the multilayer structure by collapsing all layers into a weighted simple graph.
2. Alternatively, you can find communities using a non-multilayer algorithm on each of your layers separately. Then, you would merge the results.
3. One can adapt modularity to multilayer networks by adding a term that takes into account the inter-layer connection strength, binding the nodes in the communities across layers. There are similar adaptations for random walks and label percolation approaches.
4. The concept of “multilayer density” is intrinsically ambiguous. One could intend it as the requirement of all nodes connected through all layers at the same time, or in a single different layer for each node pair.

40.7 Exercises

1. Take the multilayer network at <http://www.networkatlas.eu/exercises/40/1/data.txt>. The third column tells you in which layer the edge appears. Flatten it twice: first with unweighted edges and then with the count of the number of layers in which the edge appears. Which flattening scheme returns a higher NMI with the partition in <http://www.networkatlas.eu/exercises/40/1/nodes.txt>? Use the asynchronous label percolation algorithm (remember to pass the edge weight argument in the second case).
2. Using the same network, perform label percolation on each layer separately. Build the $|V| \times |C|$ table, perform kMeans (setting $k = 4$) on it to merge the communities. Does this return a higher NMI when compared with the ground truth?
3. Calculate the redundancy of each community you found for the previous exercises.

Part XI

Graph Mining

41

Frequent Subgraph Mining

Graph mining is a category of network analysis including the application of machine learning and data mining techniques to the analysis of complex networks. You got your machine learning primer back in Chapter 4, which you should use as a reference in case you find something here you don't understand. This chapter focuses on one of the traditional ways to do graph mining: to find frequent patterns in the graph. In further chapters, I will then cover more recent techniques which connect graph mining with the developments in neural networks: first by transforming the graph in a less complex format that can be understood by traditional neural network architectures (Chapters 42 and 43); and then by creating new neural network architectures that can exploit the complex structure of a graph directly (Chapter 44).

The idea in this chapter is to describe your graph by counting all the different motifs that compose it – sometimes called graphlets¹. I will start with a brief introductory section connecting machine learning and network science, and then delve deeper into why and how we perform frequent subgraph mining.

41.1 Machine Learning with Graphs

The idea of applying machine learning to network science is to train a model to find something interesting about your network. To train your model you need to give it some features in some sort of numerical form. This is important because one key thing you need to do during training (and then during the evaluation) is to *quantify* how well the model is doing. The problem with networks is that they are structures, and not numbers. It is difficult to quantify something with them. You need to transform the structure into a (set of) number(s) – and do it in a smart way.

Traditionally, you would just collect a bunch of characteristics of your network (or nodes, or edges) and feed them to a classifier. In

¹ Nino Shervashidze and Karsten Borgwardt. Fast subtree kernels on graphs. *Advances in neural information processing systems*, 22, 2009

Figure 41.1, we give the nodes' degree and betweenness centrality to a simple logistic regression model. The model can figure out that blue nodes have high centrality but low degree, so that unclassified node we have in grey is probably blue. This is how Rolx worked, all the way back in Section 15.1 – but here we take a more general approach than simply inferring node structural roles.

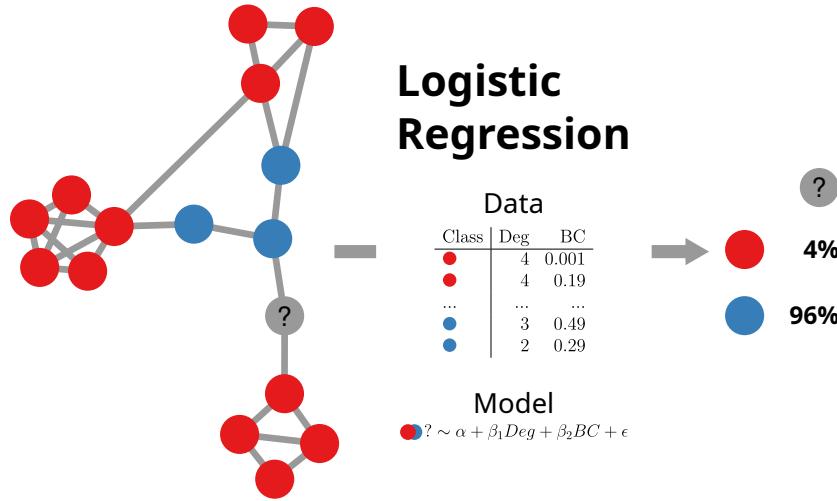


Figure 41.1: An example of classical machine learning on graphs. The node colors represent the node's classes.

We can shift the perspective from learning something about the nodes to learning something about the network in its entirety. The same approach of computing statistics for a classifier can be used here as well, for instance to estimate the similarity between two networks (Section 48.1). You can use the degree distribution, the global clustering coefficient, etc. You can also use a “bag of nodes” approach, in which a network is described by the list of individual properties of its nodes. Finally, and more relevantly for this chapter, you can enumerate all subgraph motifs and their relative occurrences to compare two networks, boiling down a complex network to the counts of its substructures.

41.2 Network Motifs

Our experience with modeling real world networks tells us that they are not random: they are an expression of complex dynamics shaping their topology. Networks will tend to have overexpressed connection patterns. Nodes and edges will form different shapes much more – or less – often than what you'd expect if the connections were random. For instance, the clustering coefficient analysis tells us that you're going to find more triangles than expected given the number of nodes or edges.

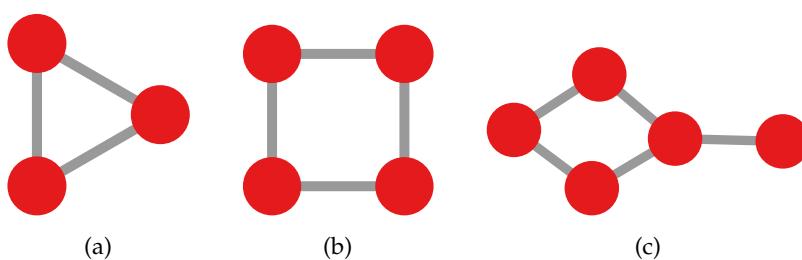
Frequent subgraph mining is the branch of network science that attempts to find these overexpressed patterns, when they are more complex than a simple triangle. Want to know whether a square with a dangling edge appears more often than chance? You have to perform subgraph mining! In frequent subgraph mining we have a wealth of techniques to systematically and efficiently enumerate all possible subgraphs and finding the ones that occur more often in your networks.

We start by defining the building blocks of complex networks. These are network motifs^{2,3,4,5}. A network motif is a subgraph of your original network with a given topology. A triangle is a motif, a square is a motif, the five nodes with the connection pattern in Figure 41.2(c) is a motif.

Generally, one wants to know which motifs are relevant for a network and which ones aren't. So the standard technique is to follow a relatively simple procedure. First, you count how many times each motif appears in your network. Second, you define a null model of your network, keeping its relevant properties fixed – maybe just the degree distribution. Third, you count the expected number of occurrences of the motifs in the null model. Finally, you compare with your observation, so that you can build an idea of the statistical significance of the motif.

We use network motifs for many different applications. For instance, and I won't get tired of bringing this up, we use them for community discovery⁶. Of course nobody forces you to have exclusively the simple motifs I depict in Figure 41.2. One can extend the concept of network motifs to encompass temporal networks^{7,8} – so time-evolving motifs –, and multilayer networks^{9,10}.

You might have noticed that the examples I show in Figure 41.2 are all very small. They include only a handful of nodes and edges. There's a reason for that. Finding motifs in a large network is a hard problem. There are clever techniques to enumerate specific small motifs which are reasonably fast^{11,12}. However, in general, one has to solve the scary graph isomorphism problem, which is the topic of the next section.



² Ron Milo, Shai Shen-Orr, Shalev Itzkovitz, Nadav Kashtan, Dmitri Chklovskii, and Uri Alon. Network motifs: simple building blocks of complex networks. *Science*, 298(5594):824–827, 2002

³ Shai S Shen-Orr, Ron Milo, Shmoolik Mangan, and Uri Alon. Network motifs in the transcriptional regulation network of *Escherichia coli*. *Nature genetics*, 31(1):64, 2002

⁴ Uri Alon. Network motifs: theory and experimental approaches. *Nature Reviews Genetics*, 8(6):450, 2007

⁵ Jukka-Pekka Onnela, Jari Saramäki, János Kertész, and Kimmo Kaski. Intensity and coherence of motifs in weighted complex networks. *Physical Review E*, 71(6):065103, 2005

⁶ Alex Arenas, Alberto Fernandez, Santo Fortunato, and Sergio Gomez. Motif-based communities in complex networks. *Physics A*, 41(22):224001, 2008a

⁷ Lauri Kovanen, Márton Karsai, Kimmo Kaski, János Kertész, and Jari Saramäki. Temporal motifs in time-dependent networks. *Journal of Statistical Mechanics: Theory and Experiment*, (11):P11005, 2011

⁸ Ashwin Paranjape, Austin R Benson, and Jure Leskovec. Motifs in temporal networks. In *WSDM*, pages 601–610. ACM, 2017

⁹ Federico Battiston, Vincenzo Nicosia, Mario Chavez, and Vito Latora. Multi-layer motif analysis of brain networks. *Chaos*, 27(4):047404, 2017

¹⁰ Manlio De Domenico, Vincenzo Nicosia, Alexandre Arenas, and Vito Latora. Structural reducibility of multilayer networks. *Nature communications*, 6:6864, 2015b

¹¹ Ali Pinar, C Seshadhri, and Vaidyanathan Vishal. Escape: Efficiently counting all 5-vertex subgraphs. In *WWW*, pages 1431–1440, 2017

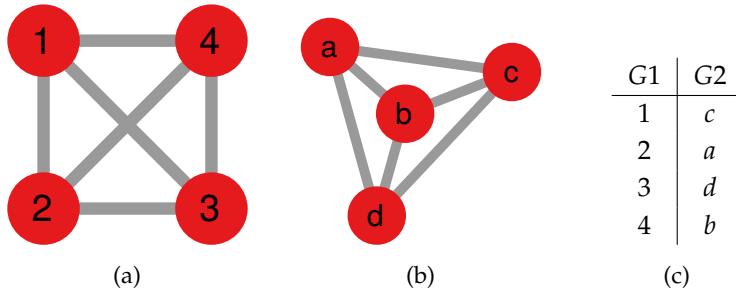
Figure 41.2: Three examples of motifs in complex networks.

41.3 Graph Isomorphism

General Idea

Colloquially speaking, we can state the graph isomorphism problem as follows: given two graphs, decide whether they are the same graph. Two graphs are the “same” if they have the same topology. More formally, graph isomorphism is the search of a function which maps each node of a graph to each node of the other graph, such that they have the same neighbors – identically mapped nodes¹³.

Are the graphs in Figure 41.3(a-b) isomorphic? Table 41.3(c) attempts to answer positively: it relabels nodes from Figure 41.3(a) into nodes from Figure 41.3(b). Since all nodes are connected to their identically labeled neighbors, the answer is yes, the graphs are isomorphic – in fact they’re both 4-cliques.



This example is simple enough, but the problem gets very ugly very soon when we start considering non-trivial graphs. Subgraph isomorphism is an NP-complete problem¹⁴: a type of problem where a correct solution requires you to try all possible combinations of labeling. This grows exponentially and requires a time longer than the age of the universe even for simple graphs of a few hundreds nodes.

That is why we’re looking for efficient algorithms to solve graph isomorphism. Recently, a claim of a quasi-polynomial algorithm shook the world¹⁵ – well, parts of it. However, this is more of a theoretical find, which cannot be used in practice. Given how hard the problem is, we can either try to solve it quickly by paying the price of getting it wrong sometimes, or more slowly but having an exact solution. I’ll give an example for both strategies, since they are both quite important for the rest of this book part.

Approximate Solutions

One of the most well-known ways to approximate a solution to graph isomorphism is to use the Weisfeiler-Lehman algorithm¹⁶. Part of the

¹² Leo Torres, Pablo Suárez-Serrato, and Tina Eliassi-Rad. Non-backtracking cycles: length spectrum theory and graph mining applications. *Applied Network Science*, 4(1):41, 2019

¹³ Brendan D McKay et al. *Practical graph isomorphism*. Department of Computer Science, Vanderbilt University Tennessee, USA, 1981

Figure 41.3: (a, b) Two graphs, with their nodes labeled with their ids. (c) A function connecting the node ids from graph (a) to the node ids of graph (b).

¹⁴ Scott Aaronson. P=?NP. *Electronic Colloquium on Computational Complexity (ECCC)*, 24:4, 2017. URL <https://eccc.weizmann.ac.il/report/2017/004>

¹⁵ László Babai. Graph isomorphism in quasipolynomial time. In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, pages 684–697. ACM, 2016

¹⁶ Boris Weisfeiler and Andrei Leman. The reduction of a graph to canonical form and the algebra which appears therein. *nti, Series*, 2(9):12–16, 1968

reason why Weisfeiler-Lehman is popular comes from the fact that it approximates the behavior of message-passing neural networks – which we'll see in Chapter 44 –, providing a nice connection between isomorphism and deep learning on graphs. The Weisfeiler-Lehman approach is intuitively simple, and I show an example in Figure 41.4.

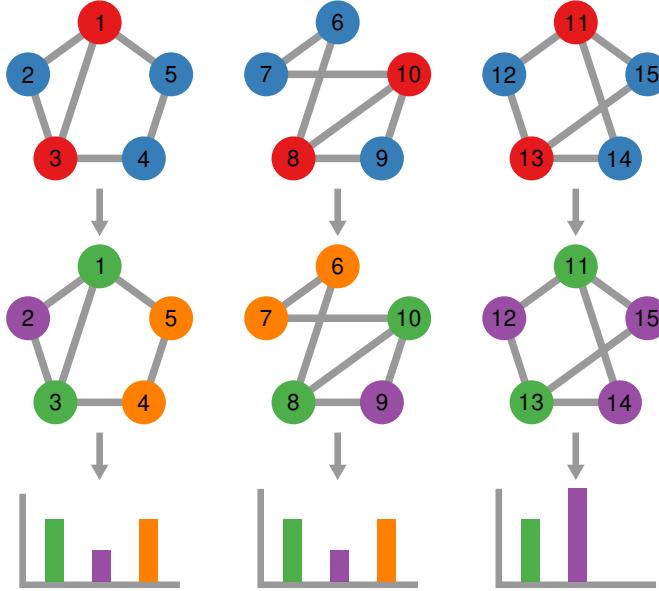


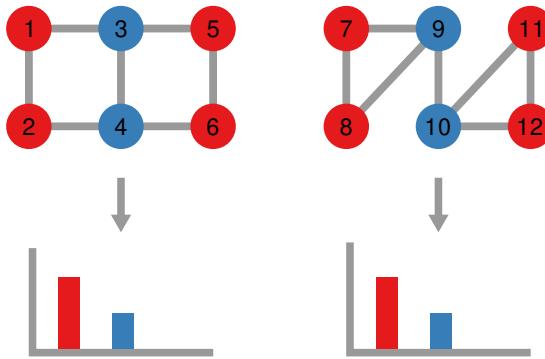
Figure 41.4: The Weisfeiler-Lehman algorithm. The node colors represent the node's classes, based on the degree. In the first step (top) red means degree 3 and blue means degree 2 for all networks. In the second step each color changes according to a consistent rule within a given network. Finally, we have the histogram of node colors.

We start by giving a color to a node – classically, each node with the same degree will get the same color. Then, each node will get a new color by combining its color with the colors of all its neighbors. For instance, in the second step of Figure 41.4, the left and middle networks encode three new colors. Green means “I'm red and I'm connected with two blue nodes and one red node”, purple means “I'm blue and I'm connected with two red nodes”, and orange means “I'm blue and I'm connected with one red and one blue node”. The network on the right is different, because it is not isomorphic with the other two, so its colors are determined with different rules – which I'm sure you can figure out.

In technical computer science terms, these sentences I used to determine the new colors are called a “hash function”. A hash function gives a unique output to each different input, that is to say we end up with a new color that is uniquely determined by all the colors that went in. At some point, there are no more changes, the graph reaches stability and there is no point in continuing. Note that, for the network on the right in Figure 41.4 that actually happened at step one, so step two wasn't necessary – I included it for aesthetic purposes. At this point a network can be described by a the color histogram of its nodes, which is the number of nodes with a given color – the

bottom row of Figure 41.4. Two networks with the same color histograms are isomorphic: they have the same number of nodes with the same colors. Indeed, the networks on the left and in the middle are isomorphic to each other, while the one on the right isn't.

This works well for most networks, and it is guaranteed to finish after at most $|V|$ steps¹⁷. However, you're a smart reader and you know there's a catch. The test sometimes fails. You can see in Figure 41.5 a case in which two non-isomorphic graphs result in the same color histogram. You can tell that the graphs are not isomorphic, because one has two triangles and the other has none.



¹⁷ László Babai and Ludik Kucera. Canonical labelling of graphs in linear average time. In *20th annual symposium on foundations of computer science (sfcs 1979)*, pages 39–46. IEEE, 1979

Figure 41.5: The Weisfeiler-Lehman algorithm. The node colors represent the node's classes, based on the degree. In both cases, red means “one red and one blue neighbor” and blue means “two red and one blue neighbor”.

There are ways to fix this issue, for instance by aggregating not only the information from direct neighbors, but from nodes at k -hops away¹⁸. However, as you might expect, this greatly increases the computational complexity of the approach.

Note that here I use simple colors based on the degree for simplicity. Nothing would change in my explanation if you were to have more complex labels – for instance some sort of node attributes. As long as you hash those attributes into a label uniquely identify by a specific set of attribute values, the algorithm works the same and will produce the same results shown in this section.

Another popular solution in this class is by using DFS codes, which I'm going to treat extensively in Section 41.4 due to their convenient connections to itemset mining. This property of DFS codes make them more useful in many practical contexts than Weisfeiler-Lehman. Specifically, we'll see how you can save computations with DFS codes if you are growing motifs by adding nodes and edges, while Weisfeiler-Lehman doesn't allow you to do so.

Exact Solutions

If one needs to solve graph isomorphism exactly, to the best of my knowledge, the current practical state of the art to solve graph iso-

¹⁸ Jin-Yi Cai, Martin Furer, and Neil Immerman. An optimal lower bound on the number of variables for graph identification. *Combinatorica*, 12(4):389–410, 1992

morphism is the VF2 algorithm^{19,20} – recently evolved to VF3^{21,22}. Since I’m just going to give the general idea of the backtracking process it implements, the sophisticated differences between the various approaches aren’t all that important. Most of the power of these algorithms come in a series of heuristics they can use to make good guesses, but here I’m just interested in talking about the general idea.

“Heuristics” means to do your darnedest not to actually run the algorithm itself, or to do it in a way that you expect it to finish more quickly. For instance, the first step of VF2 is to make all easy checks that don’t really require much thought. As an example, two graphs cannot be isomorphic if they have a different number of nodes or a different number of edges. If this check fails, you can safely deny the isomorphism. Once all these heuristic all have confirmed that the graphs could be isomorphic, then you have to begrudgingly actually look at the network, which is to say that you apply the following procedure:

- Step #1: match one node in G_1 with a node in G_2 ;
- Main loop: try to match the n -th node in G_1 with the n -th node in G_2 . If the match is unsuccessful, recursively step back your previous matches and try a new match;
- End loop, case 1: you explored all nodes in G_1 and G_2 , then the graphs are isomorphic;
- End loop, case 2: you have no more candidate match, then the graphs are not isomorphic.

Most of the heavy lifting is made in the main loop, when checking whether a match is successful or not, but most of the cleverness is in step #1: prioritizing good matches that are more likely to succeed – for instance never trying to match two nodes with different degrees. An illustrated example with a simple graph would probably be helpful. Consider Figure 41.6. VF2 attempts to explore the tree of all possible node matching (Figure 41.6(c)). It starts from the empty match – the root node.

The first attempted match always succeeds, as any node can be matched to any other node – in this case matching node 1 with node a . For the second match to succeed, we need that the two matched nodes are connected to each other. Since node a connects to node b and node 1 connects to node 2, then the $1 = a$ and $2 = b$ match is a success.

However, attempting to match $3 = c$ fails, because while node 2 is connected to node 3, node b (matched with 2) isn’t connected to c (matched to 3). Thus VF2 backtracks: it undoes the last matches

¹⁹ Luigi Pietro Cordella, Pasquale Foggia, Carlo Sansone, and Mario Vento. An improved algorithm for matching large graphs. In *3rd IAPR-TC15 workshop on graph-based representations in pattern recognition*, pages 149–159, 2001

²⁰ Luigi P. Cordella, Pasquale Foggia, Carlo Sansone, and Mario Vento. A (sub) graph isomorphism algorithm for matching large graphs. *IEEE transactions on pattern analysis and machine intelligence*, 26(10):1367–1372, 2004

²¹ Vincenzo Carletti, Pasquale Foggia, Alessia Saggese, and Mario Vento. Introducing vf3: A new algorithm for subgraph isomorphism. In *Graph-Based Representations in Pattern Recognition: 11th IAPR-TC-15 International Workshop, GbRPR 2017, Anacapri, Italy, May 16–18, 2017, Proceedings* 11, pages 128–139. Springer, 2017b

²² Vincenzo Carletti, Pasquale Foggia, Alessia Saggese, and Mario Vento. Challenging the time complexity of exact subgraph isomorphism for huge and dense graphs with vf3. *IEEE transactions on pattern analysis and machine intelligence*, 40(4):804–818, 2017a

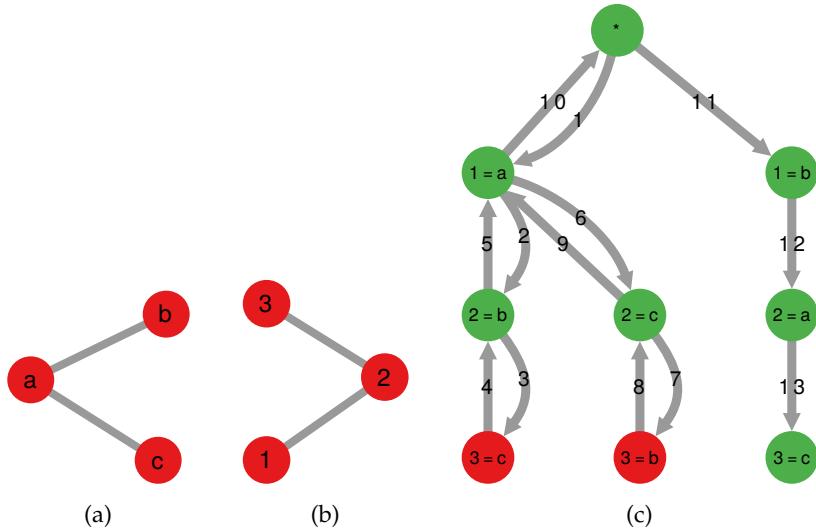


Figure 41.6: (a, b) Two graphs, with their nodes labeled with their ids. (c) The inner data structure used by the VF2 algorithm to test for isomorphism. I label each node with the attempted match. The node color tells the result of the match (green = successful, red = unsuccessful). I label the edges to follow the step progression of the algorithm.

and starts from the last successful match – provided that there are possible matches to try. In this case there aren't²³, so it backtracks again.

Trying to set $2 = c$ and $3 = b$ fails again, for the same reason as before. So VF2 has to give up also on the $1 = a$ match and start from scratch. Luckily, there's another possible move: $1 = b$. When we go down the tree all matches are successful, until we touched all nodes in the graph. At that point, we can safely conclude the two graphs are isomorphic. Note that Figure 41.6(c) doesn't include the branches that VF2 never tries in this case, for instance the $1 = c$ branch.

You can see here what I meant by smart heuristics that can help making the algorithm faster. The $1 = a$ starting move was particularly stupid, because node 1 and node a don't have the same degree, so they could never have matched. With the heuristic of "only try to match nodes with the same degree" you could have saved a lot of computation by starting directly with the $1 = b$ move – and that's what VF2 does.

As expected, multilayer networks provide another level of difficulty. One can perform graph isomorphism directly on the full multilayer structure²³, or give up a bit of the complexity and represent them as labeled multigraphs^{24,25}.

41.4 Transactional Graph Mining

So far we've been dealing with network motifs on a "top-down" approach. We have some motifs of interest and we ask ourselves whether they are overexpressed or underexpressed. This implies that you have to start with your motifs already in mind. This might not be

²³ Mikko Kivelä and Mason A Porter. Isomorphisms in multilayer networks. *IEEE Transactions on Network Science and Engineering*, 5(3):198–211, 2017

²⁴ Vijay Ingallali, Dino Ienco, and Pascal Poncelet. Sumgra: Querying multigraphs via efficient indexing. In *International Conference on Database and Expert Systems Applications*, pages 387–401. Springer, 2016

²⁵ Giovanni Micale, Alfredo Pulvirenti, Alfredo Ferro, Rosalba Giugno, and Dennis Shasha. Fast methods for finding significant motifs on labelled multi-relational networks. *Journal of Complex Networks*, 2019

possible. Sometimes, you need a “bottom-up” approach: you want an algorithm telling you the frequencies of all possible simple network motifs. This is usually the task of frequent subgraph mining.

We split frequent subgraph mining in two: transactional and simple graph mining. Transactional graph mining was developed first, because single graph mining introduces some non-trivial problems. It’s best to start by explaining transactional graph mining, and we’ll deal with the additional obstacles of single graph mining later (in Section 41.5).

Frequent Itemset Mining

Transactional graph mining is inspired by frequent itemset mining, a classical problem in data mining^{26,27,28}. In frequent itemset mining, your input is a collection of sets. Each set includes different objects. The objective is to find the subsets that appear more often in your collection of sets. The number of times each subset appear is called support. We increase support by one each time we find a subset inside a set in the collection. Figure 41.7 provides an example.

From Figure 41.7 you can see that this problem explodes in complexity very soon. Even with just five itemsets and five items, the number of possible subsets can get very high. Thus the crucial problem in frequent itemset mining is how to efficiently explore the search space. There are many algorithms to do it, but I’ll focus on the old and legendary Apriori²⁹, given its simplicity and its didactic potential.

| Data | Frequency |
|---------------------|-----------|
| ● ● ● | 4 |
| ● ● | 5 |
| ● ● ● ● | 3 |
| ● ● ● ● ● | 2 |
| ● ● ● ● ● ● | 1 |
| ● ● ● ● ● ● ● | 2 |
| ● ● ● ● ● ● ● ● | 1 |
| ● ● ● ● ● ● ● ● ● | 1 |
| ● ● ● ● ● ● ● ● ● ● | 1 |
| ● ● ● ● ● ● ● ● ● ● | 1 |

The first thing you do is giving up on the idea of finding *all* subsets. You only want to find the *frequent* ones. Thus you establish a support threshold: if a subset fails to occur in that many sets, then you don’t want to see it. This allows you to prune the search space. If subset A is not frequent, then none of its extensions can be: they have to contain it so they can be at most as frequent as A is³⁰. Thus, once you rule out subset A , none of its possible extensions should

²⁶ Jiawei Han, Jian Pei, and Yiwen Yin. Mining frequent patterns without candidate generation. In *ACM sigmod record*, volume 29, pages 1–12. ACM, 2000

²⁷ Jian Pei, Jiawei Han, Runying Mao, et al. Closet: An efficient algorithm for mining frequent closed itemsets. In *ACM SIGMOD workshop on research issues in data mining and knowledge discovery*, volume 4, pages 21–30, 2000

²⁸ Jiawei Han, Hong Cheng, Dong Xin, and Xifeng Yan. Frequent pattern mining: current status and future directions. *Data mining and knowledge discovery*, 15(1):55–86, 2007

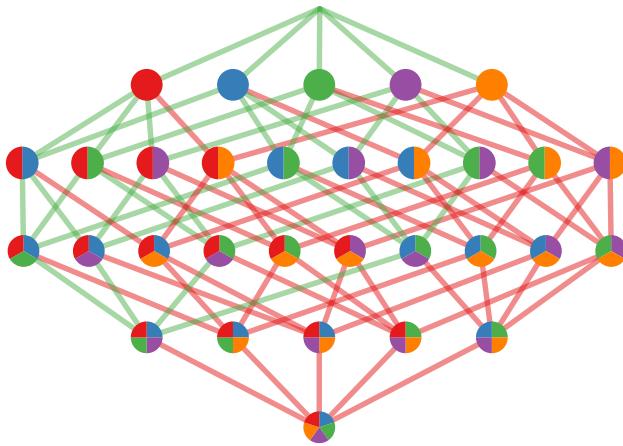
²⁹ Rakesh Agrawal, Ramakrishnan Srikant, et al. Fast algorithms for mining association rules. In *Proc. 20th int. conf. very large data bases, VLDB*, volume 1215, pages 487–499, 1994

Figure 41.7: An example of frequent itemset mining. The original data is on the left, one line per set of items (itemset). We calculate the frequency of each itemset, including all subsets (right).

³⁰ That is, the support function is anti-monotonic: it can only stay constant or shrink as your set grows in size.

even be considered, since none can be frequent. This usually allows to perform much fewer tests than the possible ones, and still return all frequent subsets.

For instance, in Figure 41.7, the orange circle only occurs once. If the support threshold is 2, we know we don't need to check the red-orange, purple-orange, and red-purple-orange subsets. With one check, we prevented three.



Actually, we prevented many more. Figure 41.8 shows the entire search space in a dataset with five different items. As you can see, if we have an item which does not pass the support threshold, the search space crumbles. Apriori explores this graph and marks all nodes with an orange item as infrequent, checking only the itemsets without red connections. This doesn't even take into account the other infrequent subsets from Figure 41.7.



As a small aside, note that, once you know the frequencies of all sets, you can build what we call "association rules". What you want to do is to find all rules in the form of: "If a set contains the objects A , then it is likely to also contain object b ". Figure 41.9 shows a simple example of the problem we're trying to solve.

Suppose that, in your data, you see 100 instances of sets containing objects a_1 , a_2 , and a_3 . And let's say that, among them, 80 also contain object b . Then you can say, with 80% confidence, that the following rule applies: $\{a_1, a_2, a_3\} \rightarrow b$. The $\{a_1, a_2, a_3\}$ part is the antecedent of

Figure 41.8: Apriori's search space. Each node represents a subset and bears the colors of the items it contains. Given that the orange item is not frequent, Apriori marks as red the links it needs not to follow, because they lead to a subset containing the infrequent orange item, which cannot make the support threshold.

Figure 41.9: An example of association rule mining. Assume the frequencies of each itemset are the ones from Figure 41.7. We generate rules recording the relative frequency of observing two itemsets. Note that these frequencies are not symmetric! While the green item only occurs 60% of the times a blue item occurs, every time green occurs we also have the blue item.

the rule, while b is the consequent.

You can also correct your confidence for chance, if you know b 's overall frequency in the data, and the size of the dataset. This is the “lift” measure. Let's say that, in our example, b appears in 120 sets. Also, our dataset contains a total of 400 sets. The lift of the rule is the relative frequency (support) of $\{a_1, a_2, a_3, b\}$ ($80/400$) over the product of the support of the antecedent and the consequent ($100/400 \times 120/400$). This latter quantity gives us the probability of the antecedent and the consequent to co-appear randomly. Doing the math: $.2/(.25 \times .3) = 2.6$. This means that the rule appears more than twice as much as we would expect if there was no relationship between the antecedent and the consequent. A lift lower than one indicates items appearing less often than they would do at random: a sign that the rule is unlikely to be interesting.

Lift is one of those Swiss army knives that has been independently invented in multiple fields. For instance it is also known as Relative Risk in statistics³¹, and Revealed Comparative Advantage in trade economics³².

When you replace itemsets with motifs, you obtain the GERM algorithm: that is why I introduced it as graph association rule mining in the link prediction chapters (Sections 23.6 and 24.2). How to go from itemsets to network motifs is the topic of the rest of this section.

From Itemsets to Network Motifs

Transactional graph mining applies all this machinery to graphs. Rather than looking at simple sets of items, we look at graph patterns: triangles, 4-cliques, bi-cliques... any possible combination of nodes and edges. So we have a database of many different graphs and we ask in how many graphs the motif appears. This is our definition of support, as Figure 41.10 shows.

The big problem is how to enumerate all possible graphs efficiently. We want to avoid trying to count the occurrences of a graph pattern G'' if it contains a pattern G' , which we already counted and found not frequent enough. Since G'' is an extension of G' , we already know it cannot be frequent enough: a larger graph can at most be as frequent as the least frequent of its subgraphs.

There are many ways to do this. An incomplete list of approaches includes FFSM³³, Gaston³⁴, Moss³⁵, etc. You can find relevant literature for a historic quantitative comparison of these methods³⁶. Just like for frequent itemset mining, I'll focus on a specific method, gSpan^{37,38}, given its historical and didactic relevance.

Graphs are more complex structures than itemsets, so building a

³¹ Jun Zhang and F Yu Kai. What's the relative risk?: A method of correcting the odds ratio in cohort studies of common outcomes. *Jama*, 280(19):1690–1691, 1998

³² Bela Balassa. Trade liberalisation and “revealed” comparative advantage 1. *The manchester school*, 33(2):99–123, 1965

³³ Jun Huan, Wei Wang, and Jan Prins. Efficient mining of frequent subgraphs in the presence of isomorphism. In *Third IEEE International Conference on Data Mining*, pages 549–552. IEEE, 2003

³⁴ Siegfried Nijssen and Joost N Kok. The gaston tool for frequent subgraph mining. *Electronic Notes in Theoretical Computer Science*, 127(1):77–87, 2005

³⁵ Christian Borgelt and Michael R Berthold. Mining molecular fragments: Finding relevant substructures of molecules. In *2002 IEEE International Conference on Data Mining, 2002. Proceedings.*, pages 51–58. IEEE, 2002

³⁶ Marc Wörlein, Thorsten Meini, Ingrid Fischer, and Michael Philippsen. A quantitative comparison of the subgraph miners mofa, gspan, fsm, and gaston. In *European Conference on Principles of Data Mining and Knowledge Discovery*, pages 392–403. Springer, 2005

³⁷ Xifeng Yan and Jiawei Han. gspan: Graph-based substructure pattern mining. In *2002 IEEE International Conference on Data Mining, 2002. Proceedings.*, pages 721–724. IEEE, 2002

³⁸ Xifeng Yan and Jiawei Han. Close-graph: mining closed frequent graph patterns. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 286–295. ACM, 2003

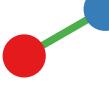
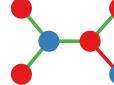
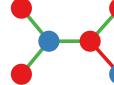
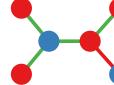
| Motif | Graph DB | | | | Support | |
|---|---|---|---|---|---------|---|
|  |  | ✓ | ✗ | ✓ | ✓ | 3 |
|  |  | ✓ | ✗ | ✓ | ✓ | 3 |
|  |  | ✗ | ✗ | ✓ | ✗ | 1 |

Figure 41.10: For each of the patterns on the left we check whether a graph in the database (on top) contains it (green checkmark) or not (red cross). The number of graphs in the database containing the motifs is its support.

search space like the one Apriori creates (Figure 41.8) is tricky. If you cannot explore the search space like Apriori does, it's even harder to prune it by avoiding exploring patterns you already know they are not frequent. gSpan solves the problem by introducing a graph lexicographic order (which I'm going to dumb down here, the full details are in the paper).

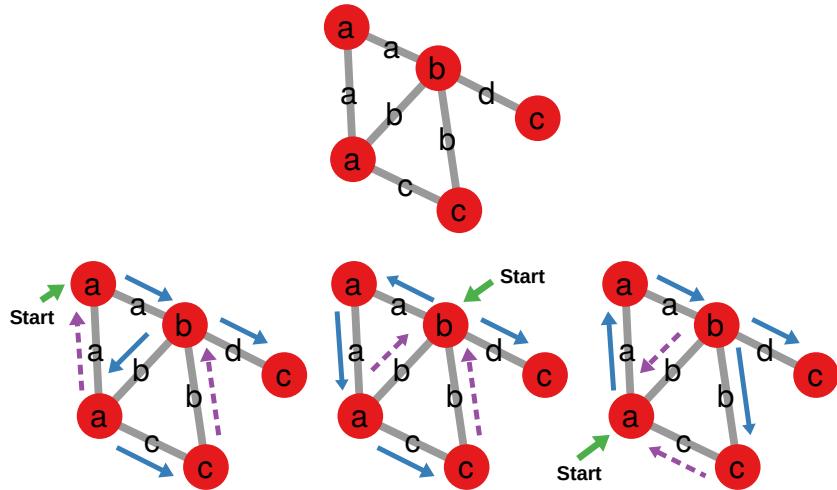


Figure 41.11: Three possible DFS explorations of the graph on top. Blue arrows show the DFS exploration and purple dashed arrows indicate the backwards edges (pointing to a node we already explored). Remember that DFS backtracks to the last explored node, not to where the backward edges points.

Suppose you have a graph, as in Figure 41.11 (top). You can explore it using a DFS strategy. Actually, you can have many different DFS paths: you can start from node a , from node b , ..., then you can move through a different edge any time. We can encode each DFS

exploration with a DFS code: a sequence of quintuples (id of source node, id of target node, label of source node, label of edge, label of target node). Figure 41.12 shows the DFS codes for the three explorations in Figure 41.11. Note that, every time we explore a node with backward edges, we insert them in the code immediately, before continuing with the DFS exploration.

| Order | DFS1 | DFS2 | DFS3 |
|-------|-----------------|-----------------|-----------------|
| 0 | (0, 1, a, a, b) | (0, 1, b, a, a) | (0, 1, a, a, a) |
| 1 | (1, 2, b, b, a) | (1, 2, a, a, a) | (1, 2, a, a, b) |
| 2 | (2, 0, a, a, a) | (2, 0, a, b, b) | (2, 0, b, b, a) |
| 3 | (2, 3, a, c, c) | (2, 3, a, c, c) | (2, 3, b, b, c) |
| 4 | (3, 1, c, b, b) | (3, 0, c, b, b) | (3, 0, c, c, a) |
| 5 | (1, 4, b, d, c) | (0, 4, b, d, c) | (2, 4, b, d, c) |

Once you have all DFS codes for a graph, you can find the *minimum* DFS code, by simply sorting them alphanumerically. The minimum DFS code – in our example the third one (DFS3 in Table 41.12) – is a canonical representation for a graph. Two graphs with the same minimum DFS code are isomorphic, and if you encounter a non-minimum DFS code in your search space you can safely ignore it. It is necessary to remind you what I said in Section 41.3: DFS codes provide an approximate solution to the graph isomorphism problem, so you might get it wrong sometimes. However, it is an occurrence rare enough not to impact you in practice most of the times.

We're ok paying this price of approximation because DFS codes allow you to reduce the graph mining problem to a frequent itemset problem. What before I called items a_1 , a_2 , and a_3 are now items like (0, 1, a, a, a), (1, 2, a, a, b), and (2, 0, b, b, a). If you always find them together with the additional (2, 3, b, b, c), you have built a graph association rule! So, with this canonical graph representation, you can solve the frequent subgraph mining problem with any frequent itemset algorithm (like Apriori).

41.5 Single Graph Mining

Transactional graph mining is great because it's a very similar problem to frequent itemset mining and has some interesting applications. For instance, your graph database could contain thousands of different chemical compounds, and you want to find the most common substructures among all those molecules. However, there's a great deal of network data that doesn't fit this mold.

For instance, if you want to mine all frequent patterns in a social network, you typically have a single large graph. In those cases, you cannot have as definition of support the "number of graphs in which

Figure 41.12: The DFS codes for the DFS explorations in Figure 41.11. DFS exploration edges in blue. For backward edges (purple) the node id of the source is higher than the node id of the target.

the pattern appears" as in the transactional setting. That number is always going to be either zero – the pattern doesn't appear – or one. What you want to do, instead, is to count the number of times the patterns appear in the single network.

However, such naive support definition won't work. To see why, consider the example in Figure 41.13. It's obvious that the red node motif appears only once: there's only one red node. However, naively, we could say that motif m_2 appears twice. This is unacceptable, because it breaks the anti-monotonicity rule: a motif can only occur at most as much as the least frequent of its sub-motifs. Since m_2 contains m_1 , it cannot appear more often than m_1 .

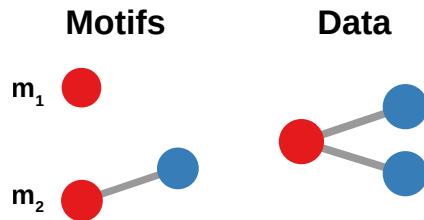


Figure 41.13: Two motifs (left) and our graph data (right). Motif m_1 appears only once. How many times does motif m_2 appear?

If we were to accept it to have higher support than its submotifs, we could not prune the search space using Apriori's strategy, meaning that we would have to explore an exponentially growing set of possibilities. This would make single graph mining impractical in all but trivial scenarios.

So the main quest for single graph mining is the one for an anti-monotonic support definition that is sufficiently easy to compute – otherwise we don't gain much time – and that hopefully makes intuitive sense. I'm going to show you three alternatives: using ego networks (Section 30.1), harmful overlap, and the minimum image support.

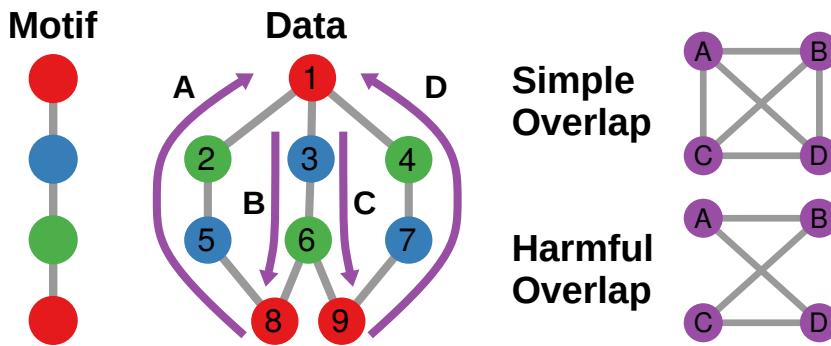
Ego Networks

One option is to bring back the problem into familiar territory. One can split the single graph in many different subgraphs and then apply any transactional graph mining technique. For example, one could take the ego networks of all nodes in the network. The support definition would then be the number of nodes seeing the pattern around them in the network.

Harmful Overlap

Another option starts from recognizing that the entire problem of non-monotonicity is due to the fact that motif m_2 appears twice only because we allow the re-use of parts of the data graph when counting

the motif's occurrences. In Figure 41.13, we use the red node in the data graph twice to count the support of m_2 . In practice, the two patterns supporting m_2 overlap: they have the single red node in common. We could forbid such overlap: we don't allow the re-use of nodes when counting a motif's occurrences. With such a rule, m_2 would appear only once in the data graph. If we applied the rule, we would have an anti-monotone support definition³⁹: larger motifs would only appear fewer times or as many times as the smaller motifs they contain.



To see how, consider Figure 41.14. The motif appears four times, but each of these four occurrences share at least one node. We can create an “overlap graph” in which each node is an occurrence in the data graph, and we connect occurrences if they share at least one node. If we forbid overlaps, we only want to count “complete” and non-overlapping occurrences. This is equivalent of solving the maximum independent set problem (see Section 12.3) on the overlap graph: finding the largest set of nodes which are not connected to any other member of the set. In this case, we have four independent sets all including a single node – because the overlap graph is a clique – and thus the pattern occurs only once.

This is the simple overlap rule and it is usually too strict. There are some overlaps between the occurrences that do not “harm” the anti-monotonicity requirement for the support definition⁴⁰. To find harmful overlaps you need to do two things. First, you look at which nodes are in common between two occurrences. For instance, in Figure 41.14, $A \cap B = \{1, 8\}$, A and B share nodes 1 and 8. Second, you need to make sure that these nodes that are in common between the two occurrences are not required to map the same nodes at the same time. In the case of A and B they are, because the only way to map the top red node in the motif is to use node 8 for A and 1 for B . This is a harmful overlap.

The non-harmful overlaps are the ones in which this doesn't

³⁹ Michihiro Kuramochi and George Karypis. Finding frequent patterns in a large sparse graph. *Data mining and knowledge discovery*, 11(3):243–271, 2005

Figure 41.14: From left to right: a pattern, the graph dataset, and its corresponding simple and harmful overlap graphs. I label each occurrence of the motif with a letter, which also labels the corresponding node in the overlap graph.

⁴⁰ Mathias Fiedler and Christian Borgelt. Support computation for mining frequent subgraphs in a single graph. In *MLG*, 2007

happen. For instance A and C are not overlapping harmfully: the only node in common between A and C is node 1. However, we do not need node 1 to map the same node in the motif in A and C : when we use node 1 in A we use node 9 in C , when we use node 1 in C we use node 8 in A . Node 1 is also the only node in common between A and D , but in this case the overlap is harmful, because we use it to map the same node in the motif: the bottom red node.

The simple and harmful overlap build different overlap graphs, but then they count occurrences in the same way, using the maximum independent set problem. In the example from Figure 41.14 this leads to different support values: for the harmful support, the motif occurs twice in the network – you have two independent sets of size two (A, C and B, D).

Minimum Image Support

The problem of simple and harmful overlap is that they have to solve the maximum independent set problem for every motif we search in a possibly very large overlap graph, which is a hard problem. Thus, researchers proposed a new definition which skips this computation: the minimum image support⁴¹. In this definition, what matters is that we do not re-use the same node in the network to play the same role in the motif.

⁴¹ Björn Bringmann and Siegfried Nijssen. What is frequent in a single graph? In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 858–863. Springer, 2008

| | <i>A</i> | <i>B</i> | <i>C</i> | <i>D</i> | Count |
|--|----------|----------|----------|----------|-------|
| | 8 | 1 | 1 | 9 | 3 |
| | 5 | 3 | 3 | 7 | 3 |
| | 2 | 6 | 6 | 4 | 3 |
| | 1 | 8 | 9 | 1 | 3 |

Figure 41.15: (a) The motif (b) The image table for the minimum image support definition, with the motif's nodes as rows and all the occurrences of the motif as columns. Each cell records the node id we use for the mapping.

In practice, we look at which node in the network we use to map each node in the motif. To do so, we build an “image” table. Figure 41.15 shows the image table for the example in Figure 41.14. In the table, we record the node in the network playing the role of a specific node in the motif. Thus, the support of the motif is the minimum number of distinct row values – two identical values in a row stand for an incompatible pair of occurrences.

The aforementioned Moss method is able to deal with multigraphs, thus it can be used for some multilayer graph mining as well – assuming your multilayer network can be represented as a multigraph. Otherwise, Muxviz⁴² allows for multilayer motif counting, but employs a naive support definition and thus cannot be used for graph mining, due to the break of the anti-monotonicity requirement.

41.6 Summary

1. Machine learning can give you powerful insights about your networks, but first you need to find a way to transform the complex structure of a network into numerical values that can be handled by machine learning algorithms. One way to do so is to count network motifs.
2. Network motifs are small simple graphs that you can use to describe the topology of a larger network. For instance, you can count the number of times a triangle or a square appears in your network.
3. To do so, you need to solve the problem of “graph isomorphism”, which is the task of determining whether two graphs have the exact same topology. Graph isomorphism is a computationally heavy problem to solve, which can be tackled quickly in an approximate way, or slowly in an exact way.
4. Frequent subgraph mining is the graph equivalent of frequent itemset mining: to efficiently find all the graph motifs that appear in your network, avoiding to perform the expensive graph isomorphism problem for patterns that you already discovered not being frequent.
5. Frequent subgraph mining comes in two flavors. Transactional mining analyzes many small networks and counts the number of networks containing the motif we’re counting. Single graph mining analyzes a single large graph and counts the number of times the motif appears.
6. Unfortunately, simply counting motif appearances in a single graph cannot support an efficient exploration of the search space, because larger motifs might appear more often than smaller motifs, i.e. the counting function is not-monotonic.
7. We have different ways of counting the frequency of a motif in a single graph that are anti-monotonic. They are all based on the concept that we should not count twice patterns that are overlapping, for different definitions of what “overlapping” means.

⁴² Manlio De Domenico, Mason A Porter, and Alex Arenas. Muxviz: a tool for multilayer analysis and visualization of networks. *Journal of Complex Networks*, 3(2):159–176, 2015c

41.7 Exercises

1. Test whether the motifs in <http://www.networkatlas.eu/exercises/41/1/motif1.txt>, <http://www.networkatlas.eu/exercises/41/1/motif2.txt>, <http://www.networkatlas.eu/exercises/41/1/motif3.txt>, and <http://www.networkatlas.eu/exercises/41/1/motif4.txt> appear in the network at <http://www.networkatlas.eu/exercises/41/1/data.txt>.
2. How many times do the motifs from the previous question appear in the network? <http://www.networkatlas.eu/exercises/41/1/motif2.txt> is included in <http://www.networkatlas.eu/exercises/41/1/motif3.txt>: is the latter less frequent than the former as we would require in an anti-monotonic counting function?
3. Suppose you define a new type of clustering coefficient that is closing <http://www.networkatlas.eu/exercises/41/1/motif3.txt> with <http://www.networkatlas.eu/exercises/41/1/motif4.txt>. What would be the value of this special clustering coefficient in the network?

42

Shallow Graph Learning

In the next chapters we're going to fully explore the recent and quickly evolving field of Graph Neural Networks (GNNs). I already mentioned some basics of neural networks in Section 6.4 and ideas on what they can do on graphs in Section 15.3. Now it's time to explain exactly how you can use neural networks on graphs to perform those – and more – tasks.

There are a few good books if you want to get into this topic more in depth^{1,2}, which you should check out. Some of the surveys in the literature³ also come with code⁴ that you can use to follow along the explanations.

For the time being, we're focusing on building "embeddings". A small terminology note: building embeddings and using them in GNNs can go under different terms depending on where the focus is. For instance, GNNs can be considered a special case of the more general geometric learning that goes beyond the use of Euclidean space in data analysis⁵. We'll see more on the use of complex non-Euclidean geometries in Chapter 47. Another term you could see is "collective classification" field: the attempt of classifying nodes by looking at how they relate to the rest of the network⁶.

Finally, one thing you need to know to read these chapters critically. In the next couple of chapters, I am adopting a purely structural approach to build node embeddings. This means that they are built exclusively by looking at their connections. It is only in Chapter 44 that I'm going to give you a fuller picture that takes the node attributes into account as well. So don't assume that looking at the connections is the only way to build a node embedding!

42.1 What is a Node Embedding?

A node embedding is a vector of numbers that describes the node's role in the complex network structure. To get a node embedding you need a function that maps each node of your network to a vector

¹ William L Hamilton. *Graph representation learning*. Morgan & Claypool Publishers, 2020

² Michael M Bronstein, Joan Bruna, Taco Cohen, and Petar Veličković. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. *arXiv preprint arXiv:2104.13478*, 2021

³ Palash Goyal and Emilio Ferrara. Graph embedding techniques, applications, and performance: A survey. *Knowledge-Based Systems*, 151:78–94, 2018

⁴ <https://github.com/palash1992/GEM>

⁵ Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017

⁶ Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Gallagher, and Tina Eliassi-Rad. Collective classification in network data. *AI magazine*, 29(3):93–93, 2008

of numbers. Figure 42.1 shows a stylized example of what a node embedding is. We transform the original graph (Figure 42.1(a)) into a set of two-dimensional numerical vectors for each of its nodes (Figure 42.1(b)). These vectors have a spatial relationship reflecting some of the graph's properties (Figure 42.1(c)).

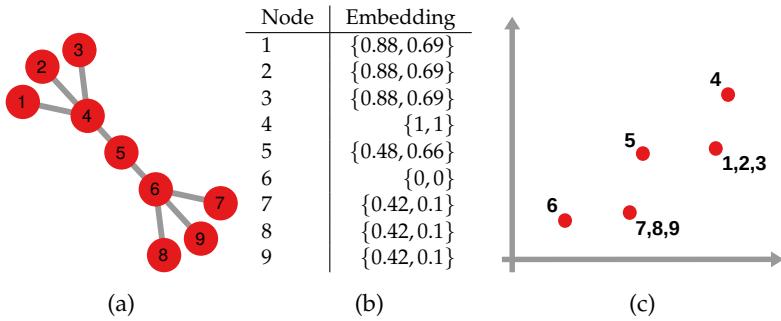


Figure 42.1: (a) An example graph. (b) One of the possible embeddings of (a), assigning a two dimensional vector to each node. (c) The scatter plot representation of (a)'s embeddings.

Properties of Good Embeddings

Not all functions turning a node into a vector of numbers are good. To be useful, a node embedding should satisfy at least a few basic properties:

1. An embedding should be small, meaning that it should have a low dimensionality, few entries. If your network has $|V|$ nodes, then the vector representing each node should have fewer than $|V|$ entries.
2. An embedding should be dense, you don't want most of your entries in most of your node embeddings to be zeroes – otherwise, why having those dimensions in the first place, if they don't give you much extra information?
3. An embedding should be permutation invariant: we shouldn't be getting different embedding depending on the order in which we're exploring the nodes.
4. An embedding should be a faithful representation of the topology of your network. Nodes that are “similar” should be represented by vectors at a low distance – as Figure 42.2 shows.

The aim of the embedding function is to embed your nodes into a low dimensional space in a way that respects at least those properties. You can think of your nodes as the points of a scatter plot: points that are spatially close to each other are similar. If this low dimensional representation is any good, you can then analyze this scatter plot and discover interesting properties of your nodes. This is helpful, because

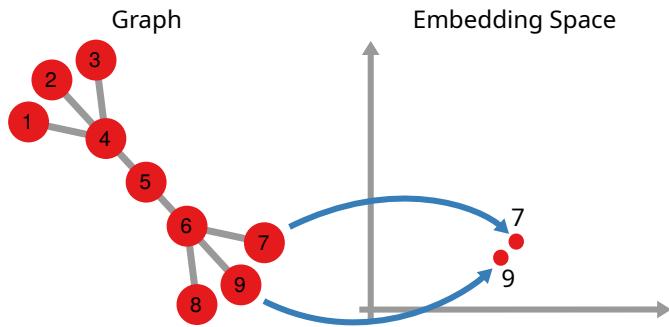


Figure 42.2: A graph and its embedding space, where the embedding function represented by the blue arrows places structurally equivalent nodes nearby in a 2D embedding space.

usually the scatter plot is (i) easier to analyze than a graph, and (ii) a more common data structure than a graph on which you can apply a more diverse set of algorithms that were not developed with graphs in mind.

The real meat that makes embeddings interesting is the last constraint – which connects similarity of embeddings with the similarity of the nodes. But before we dive into that, there might be a question lingering in your mind: why are we bothering with graph embeddings? We can already represent easily a node with a vector. In fact, this is something you taught me since Chapter 8! A node is nothing more than a row in the adjacency matrix of the graph. Thus it is a vector. Why can't we use that as our “embedding”?

The reason is that A fails each of the first three constraints:

1. It is not low-dimensional. If you slice the adjacency matrix, your nodes will be represented by a vector of length $|V|$, which doesn't save you any dimensions.
2. It is binary and sparse, therefore not information-dense. Most algorithms that you want to apply on your embeddings don't work well with this sort of input data.
3. It is not permutation invariant. You will get a different embedding depending on how you sort the rows and columns of your matrix. So you could get two isomorphic graphs with different embeddings!

I show you an example of the last problem in Figure 42.3. You can tell that the two graphs defined by the two matrices I show there are isomorphic, because one matrix is the rotation of the other – quite literally, I was too lazy to make another figure from scratch, so I just rotated it in latex.

In fact, you want something even stronger than permutation invariance. You want permutation equivariance: not only the embeddings of a node should not be changed by the order in which you visit

Figure 42.3 consists of two 7x7 binary matrices labeled (a) and (b). Matrix (a) has a checkerboard pattern where every second row and column contains a 1. Matrix (b) has a similar pattern but with some additional 1s in the diagonal and off-diagonal positions.

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 | 1 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 |

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 | 0 | 1 |
| 0 | 0 | 0 | 1 | 1 | 1 | 0 |

Figure 42.3: (a) An adjacency matrix. (b) The adjacency matrix of a graph isomorphic to the one represented by the adjacency matrix in (a).

the graph, but nodes in the exact same position in the graph should always get the same embeddings.

Embeddings and Node Similarity

Let's now focus on the most interesting of the constraints I mentioned: similar nodes should get similar embeddings. This means that we need to decide how to calculate the similarity of nodes and of embeddings.

The similarity of embeddings is the easiest, because they are just numerical vectors, so let's start from there. We can store all our embeddings in a matrix, let's call it Z . Z is a $|V| \times d$ matrix, where each node $v \in V$ gets a d dimensional vector. One can estimate the similarity between two node embeddings Z_u and Z_v by their dot product $Z_u^T Z_v$. Great.

What about node similarity? The whole point of making embeddings was that we cannot easily quantify a network structure, so aren't we lost in recursion? Well, actually there are many ways to estimate node similarity: we saw a few when we talked about different roles and structural equivalences in Chapter 15. The fact that there are no unique ways to define node similarity shouldn't scare you, but empower you: you can define different embeddings depending on your notion of node similarity, which depends on what you want to do with your embeddings. In other words, you can build and optimize embeddings for many specific tasks – Figure 42.4 shows you a perfectly valid alternative embedding for the network from Figure 42.1.

The example I show in Figure 42.1 is one you'd use if you wanted your embeddings to help you with community discovery or some spreading event on the network. However, it would be poor when it comes to estimate, for instance, structural equivalence (Section 15.2), which Figure 42.4 instead captures.

Note that you have an additional degree of freedom: not only you can decide which function you use to create the embedding, you can also decide the shape of the space in which you're creating

(a)

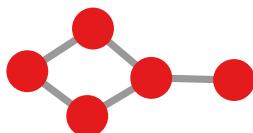
| Node | Embedding |
|------|--------------|
| 1 | {0.01, 0.01} |
| 2 | {0.01, 0.01} |
| 3 | {0.01, 0.01} |
| 4 | {0.01, 1} |
| 5 | {0.33, 0.5} |
| 6 | {0.05, 0.9} |
| 7 | {0.05, 0.05} |
| 8 | {0.05, 0.05} |
| 9 | {0.05, 0.05} |

(b)

the embedding. For simplicity, in Figures 42.4(a) and 42.4(b) I use an Euclidean space: each dimension has equal importance and the distance between two points is determined by the length of the straight line between the points. This is not the only choice. For instance, some researchers use a hyperbolic space^{7,8,9}, where the distance between two points is not determined by a straight line, but by the branch of a hyperbole.

Classical Embeddings

At this point in the chapter, you might feel a bit daunted by the concept of embeddings. It ostensibly looks simple: to embed is to transform a node in a vector of numbers. But you might think that, in order to encode the complexity of what a node does in a potentially large structure you might have to apply ridiculously sophisticated and advanced algorithms. In this subsection, I want to convince you that this isn't true. In fact, you have seen me using node embeddings – without me telling you that I was – ever since Section 2.7! Let me show you the first case of node embeddings you saw in the book, stripped away of some unnecessary complexity: Figure 42.5 is a simplified reproduction of Figure 2.7 – literally the first network you saw in the book.



I can hear you thinking: *Michele, what the hell are you talking about?* But bear with me for a second. In Figure 42.5 I need to decide where to put the nodes on a 2D plane. That means I need to figure out a pair of x and y coordinates so that the network looks understandable. The way I decide the values of x and y for each node depends on their connections in the network. This is equivalent to say that I use the structure of the network to assign to each node a 2D numerical

Figure 42.4: (a) A different valid embedding of the graph in Figure 42.1(a), assigning a two dimensional vector to each node. (b) The scatter plot representation of (a)'s embeddings.

⁷ Dmitri Krioukov, Fragkiskos Papadopoulos, Maksim Kitsak, Amin Vahdat, and Marián Boguná. Hyperbolic geometry of complex networks. *Physical Review E*, 82(3):036106, 2010

⁸ Ginestra Bianconi and Christoph Rahmede. Emergent hyperbolic network geometry. *Scientific reports*, 7:41974, 2017

⁹ Maksim Kitsak, Ivan Voitalov, and Dmitri Krioukov. Link prediction with hyperbolic geometry. *Physical Review Research*, 2(4):043113, 2020

Figure 42.5: A simple graph whose nodes are located on a 2D space interpreting their 2D embeddings as spatial coordinates.

vector. In turn, this also means exactly that I embed the network structure in a 2D space.

It would be ugly if there were edge crossings or the nodes overlapped, so the network structure constrains how I build these vectors. Every time you try to display your network in the clearest form possible, you are creating 2D (sometimes 3D) node embeddings. In fact, there are a bunch of specialized algorithms that layout your networks – we'll see them in Chapter 51. They are a relatively simple way to create node embeddings – but, crucially, it is a totally *valid* way to do it.

42.2 From Neural Networks to Graph Neural Networks

The main reason people nowadays can't get enough of making new ways of building node embeddings is because node embeddings are a perfect way to boil down the complexity of a graph in a way that classical neural network architectures can understand it. To see what I mean, we shall take another look at Figure 6.12, which I reproduce here as Figure 42.6.

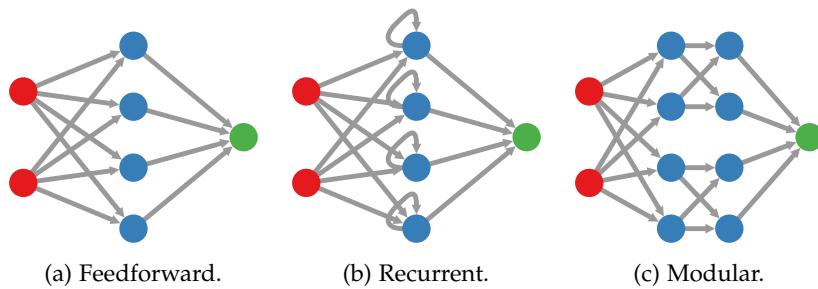


Figure 42.6: Different neural networks. The node color determines the layer type: input (red), hidden (blue), output (green).

From the figure, you can see the general architecture of a neural network. The input goes into the input layer, it is processed by one or more hidden layers, until the neural network produces an output in the output layer. Now, as I mentioned in Chapter 41, the problem here is that the input layer is a vector of numbers, but networks aren't vectors of numbers. If they were, analyzing them would be easy and this book wouldn't exist. So we need to find a way to transform them into something a neural network understands. Which is embeddings, as I mentioned.

We generally call this approach “shallow learning” because we use the network only to build the embeddings. However, once we have the embeddings, we forget about the network and we don't use it any more – there are no more edges in the vectors of numbers we feed into the algorithm. As I already outlined, when we get to Chapter 44, we'll deal with “deep learning”. In this case, we use

the edges themselves as the neural network, which then become a central part of the learning process. This will require us to define new graph neural network algorithms, rather than re-using generic neural network approaches that were not defined specifically for graphs.

However, I like to think that the two approaches aren't so different. This is just my personal view but, if you squint hard at it, many of the classical neural network approaches are actually graph neural networks. In Section 43.1 I'll give an example from natural language processing, but here I'll focus on another classical neural network approach: image convolution. Figure 42.7 shows a depiction of what a simple convolution operation looks like at an abstract level.

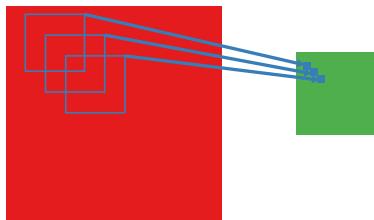


Figure 42.7: A convolution operation. Color determines the layer of the neural network: red = input, blue = hidden, green = output.

The figure shows what one normally does in image convolution: we chunk the image in a collection of (overlapping) pixels. Then the neural network learns – across multiple layers – relationships between adjacent chunks of pixels. Eventually it will figure out that these chunks here look like an eye, those over there look like a mouth, and so on until it classifies the image as a face.

But consider a perspective shift, which I represent in Figure 42.8. Here, each square is a (collection of) pixels. What the neural network does is to look at each pixel's neighbors and use them to update the pixel's representation. This is literally the same as having a simple network – a regular grid of nodes – as the input of your neural network architecture. So, in this simplified example, there are two differences between a neural network and a graph neural network. A non-graph neural network:

1. Only accepts simple graphs as inputs; and
2. It requires the simple graphs to be *ordered* – we know which nodes come “before” and “after” the current one, for instance the ones representing the pixel on the left and on the right.

A complex network doesn't have such a canonical node ordering. So, to move from neural networks to graph neural networks, we “simply” have to figure out how to handle arbitrarily complex unordered graphs.

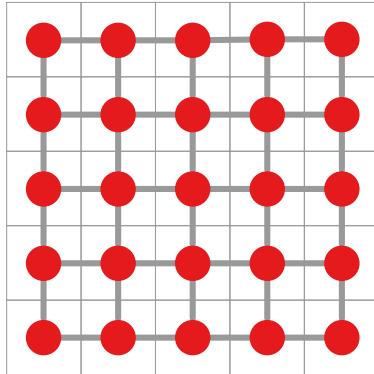


Figure 42.8: A way of thinking about image convolution as a graph neural network problem. Each square in the grid represents a pixel. Each pixel can be thought of as a node connected to its adjacent pixels.

We now take a look at specialized ways to build graph embeddings. I decided to focus on two main families of approaches. Here we focus on spectral embeddings. Then, in the next chapter, we'll see a random walk embeddings, which has proven to be a popular and powerful way of building embeddings.

42.3 Spectral Embeddings

This is the oldest category of graph embeddings techniques. The objective of the researchers working in this early definition of the problem was simple dimensionality reduction. In other words, they were mainly interested in having small vectors representing the topology around a node without having to look at the entirety of its neighborhood – that is to say, its adjacency vector. In practice, they were trying to have some sort of network-aware Principal Component Analysis (Section 5.6).

Let's repeat the general idea of embedding nodes: if nodes u and v are connected to each other by a strong link A_{uv} , then their embeddings Z_u and Z_v should be similar. In spectral embeddings you want to find a good loss function that approximates well your objective which can be optimized by solving an eigenvector problem. In practice, you want to find the right matrix representation of your graph corresponding to that specific loss function.

These matrix representations are sometimes transformations of A , but most often of the Laplacian L . Let's look at a simple example to warm up. Figure 42.9(a) shows a graph. Here, I choose to transform A into $D^{-1/2}AD^{-1/2}$ – see Figure 42.9(b)). The first step to get the embedding is to calculate the Single Value Decomposition (SVD) of that transformed matrix – see Section 5.6.

Remember that SVD produces three outputs – the singular values and two matrices with the singular vectors. For now, I only show you the singular values in Figure 42.9(c), which give you an idea of the

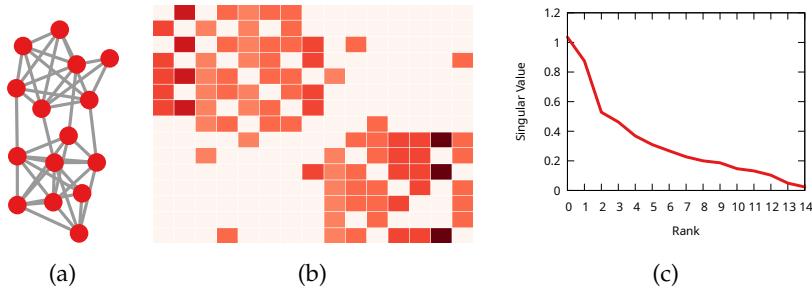


Figure 42.9: (a) A graph. (b) Its $D^{-1/2}AD^{-1/2}$ matrix. (c) The singular values of (b), sorted on the x axis in descending order.

explanatory power of a given singular vector. You can see that, to describe $D^{-1/2}AD^{-1/2}$ the first two single vectors are more powerful than the rest. This is equivalent in saying that you can represent each node with a 2D embedding, composed by the values corresponding to that node in the first two singular vectors.

Figure 42.10 shows you how those 2D embeddings look like, highlighting the difference between nodes in one community and the ones in the other. In practice, if you decided you want 2D embeddings, you can take the first two singular values and those will make your Z embedding matrix.

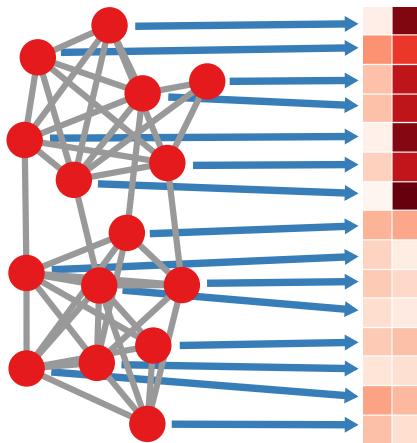


Figure 42.10: On the right you have the two singular vectors corresponding to the top two singular values. Each node corresponds to a row in this matrix.

This could be familiar to you, it is practically equivalent to solving min-cut by using the eigenvectors of the Laplacian – something we saw all the way back in Section 11.5. So that's why L is a popular solution here. But building spectral embeddings is more general than just finding the mincut solution. A community structure is not necessary to create spectral embeddings. You can do that with arbitrary networks, provided you have a good loss function. Different loss functions will lead to slightly different transformations of L before its decomposition. As far as I can tell, this is the main thing differentiating spectral embedding approaches.

Let's take a look at a couple of examples. There are other ap-

proaches, most notably ways to include node attributes in the construction of the embedding¹⁰. But, once you got a few basic examples laid down, you're equipped to think about spectral embeddings techniques in general.

Locally Linear Embedding

In Locally Linear Embedding¹¹, assume that you have your Z embeddings. Then, the loss function you want to minimize is:

$$\sum_u \left| Z_u - \sum_v Z_v A_{uv} \right|^2.$$

Here, the straight bars mean that we are taking the length of the vector $Z_u - \sum_v Z_v A_{uv}$ and then we square it. In practice, your loss is the difference between Z_u and Z_v , weighted by how strongly they are connected in the graph (A_{uv}). If u and v are connected by a high A_{uv} link strength, we better have a low $Z_u - Z_v$ difference to cancel it out! It turns out that finding Z , the matrix containing the embeddings Z for all our nodes can be solved as an eigenvector problem. Specifically, you can take the smallest eigenvectors of the matrix $(I - A)^T(I - A)$ – but discarding the actual smallest one –, with I being the identity matrix.

Laplacian Eigenmaps

Laplacian Eigenmaps¹² change the loss function, which means the objective is still the eigenvector decomposition of a matrix, but the matrix itself is built differently. In this case, the loss function is:

$$\frac{1}{2} \sum_u |Z_u - Z_v|^2 A_{uv},$$

which gives more weight to the $Z_u - Z_v$ difference than before – because it squares it before multiplying it with the A_{uv} link strength. Also in this case you can solve the problem by taking the smallest eigenvector of the normalized Laplacian $D^{-1/2}LD^{-1/2}$, with D being the degree diagonal matrix of the graph G .

42.4 Pooling

You might have noticed that so far I have exclusively talked about generating node embeddings. However, in many cases, you don't want to have node embeddings. Maybe you want to have edge embeddings to do link prediction – trying to figure out if there is a key function distinguishing the embedding of edges that exists from

¹⁰ Xiaotong Zhang, Han Liu, Xiao-Ming Wu, Xianchao Zhang, and Xinyue Liu. Spectral embedding network for attributed graph clustering. *Neural Networks*, 142:388–396, 2021

¹¹ Sam T Roweis and Lawrence K Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326, 2000

¹² Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *Advances in neural information processing systems*, pages 585–591, 2002

edges that don't, which will tell you which non-existing links are likely to exist. Or you might want to have an embedding for the entire network. A classical case is trying to classify molecules to predict some of their characteristics. A molecule is a network of atoms, so you want to embed the entire structure.

To do this, you need to perform what we call "pooling". The role of a pooling function is to take a bunch of embeddings and return one that is a good representation of all of them. Pooling for edges is relatively straightforward, since we only have two node embeddings to deal with and not much complexity. For edge (u, v) , you have embeddings Z_u and Z_v . To get to a Z_{uv} embedding you can average them ($Z_{uv} = (Z_u + Z_v)/2$) or multiply them element-wise ($Z_{uv} = Z_u \times Z_v$), or take the L1 or L2 norm, you get the idea. To give you an example, I take the node embeddings from Figure 42.1(b) and I create a bunch of edge embeddings out of them in Figure 42.11.

| Edge | AVG Pool | Prod Pool | L1 Norm | L2 Norm |
|--------|--------------|--------------|--------------|--------------|
| (1, 4) | {0.94, 0.84} | {0.88, 0.69} | {1.88, 1.69} | {1.33, 1.21} |
| ... | ... | ... | ... | ... |
| (4, 5) | {0.74, 0.83} | {0.48, 0.66} | {1.48, 1.66} | {1.11, 1.20} |
| (5, 6) | {0.24, 0.33} | {0.00, 0.00} | {0.48, 0.66} | {0.48, 0.66} |
| (6, 7) | {0.21, 0.05} | {0.00, 0.00} | {0.42, 0.10} | {0.42, 0.10} |
| ... | ... | ... | ... | ... |

Pooling information for graphs is... a bit more complicated. The structure of a single edge is trivial. The structure of a graph is not. It could have an arbitrary number of nodes and edges, distributed in an arbitrary way. A good pooling function needs to take this into account to build the final embedding. We're going to see a few methods briefly, and you can read some surveys out there^{13,14} to gain a deeper understanding.

Flat Pooling

The algorithms that do flat pooling for graphs all think: *nah, Michele is wrong, pooling graphs ain't that different from pooling edges*. In this strategy, "flat" means that all vectors are aggregated in a manner that doesn't really take the network into account. So you could apply any of the functions that I mentioned before to all the node embeddings of a graph.

Slightly more sophisticated methods try to weight some of these embeddings differently depending on some characteristic that can be discovered with a neural network, using the attention mechanism¹⁵ – which I haven't explained to you, yet. Another approach¹⁶ sorts the node embeddings in increasing order and then performs a convolu-

Figure 42.11: A few different ways of pooling the node embeddings from Figure 42.1(b) into edge embeddings.

¹³ Daniele Grattarola, Daniele Zambon, Filippo Maria Bianchi, and Cesare Alippi. Understanding pooling in graph neural networks. *IEEE transactions on neural networks and learning systems*, 2022

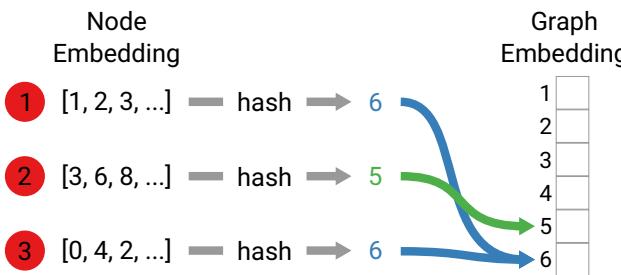
¹⁴ Chuang Liu, Yibing Zhan, Jia Wu, Chang Li, Bo Du, Wenbin Hu, Tongliang Liu, and Dacheng Tao. Graph pooling for graph neural networks: Progress, challenges, and opportunities. *arXiv preprint arXiv:2204.07321*, 2022

¹⁵ Takeshi D Itoh, Takatomi Kubo, and Kazushi Ikeda. Multi-level attention pooling for graph neural networks: Unifying graph representations with multiple localities. *Neural Networks*, 145: 356–373, 2022

¹⁶ Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. An end-to-end deep learning architecture for graph classification. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018b

tion until you get to the desired number of dimensions – solving the issue that networks with a different number of nodes would generate a graph embedding with different dimensions.

Other clever methods involve hashing the node embedding so that it becomes an index and then update the corresponding index in the graph embedding¹⁷. Figure 42.12 shows how this happens in practice. Nodes 1 and 3 happen to get the same hash and so update the same part of the graph embedding. This is fine and expected – if you have a graph embedding with 256 entries, but a graph with two million nodes, you’re expecting a ton of hash collisions.



The downside of this approach is that, if you have a small graph, the resulting graph embedding will be sparse. You can see it in the figure, where more than half of the resulting embedding is not updated.

There are more approaches in this class, but you get the idea: here the structure of the network doesn’t matter that much. Let’s see a couple of examples where instead it matters a lot.

Node Clustering Pooling

You can see the flat pooling approach I presented in the previous section using a different perspective. Averaging all node embeddings is equivalent to say that there exists a “virtual node” connected to all nodes in the network, whose embedding is the average of all its neighbors¹⁸ – i.e. all the nodes of the network. That will give you the same result as averaging all node embeddings. You can see this perspective in Figure 42.13(a).

Once you adopt this perspective, you see immediately how to make a smarter choice. What if we decide to have not one but a fixed number of virtual nodes? In that way, we can have a fixed size graph embedding. If we choose those nodes wisely, we get a good embedding that is representative of the structure of the network. This is what we’re doing in this section, and Figure 42.13(b) shows you how to visually think in these terms.

Specifically, we choose virtual nodes via node clustering^{19,20,21,22},

¹⁷ David K Duvenaud, Dougal Maclaurin, Jorge Iparragirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. In *NIPS*, pages 2224–2232, 2015

Figure 42.12: The pooling approach determining the index of the graph embedding to be updated by hashing the node embedding.

¹⁸ Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*, 2015

¹⁹ Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *NIPS*, pages 3844–3852, 2016

²⁰ Martin Simonovsky and Nikos Komodakis. Dynamic edge-conditioned filters in convolutional neural networks on graphs. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3693–3702, 2017

²¹ Filippo Maria Bianchi, Daniele Grattarola, and Cesare Alippi. Spectral clustering with graph neural networks for graph pooling. In *International conference on machine learning*, pages 874–883. PMLR, 2020

²² Hao Yuan and Shuiwang Ji. Structpool: Structured graph pooling via conditional random fields. In *Proceedings of the 8th International Conference on Learning Representations*, 2020

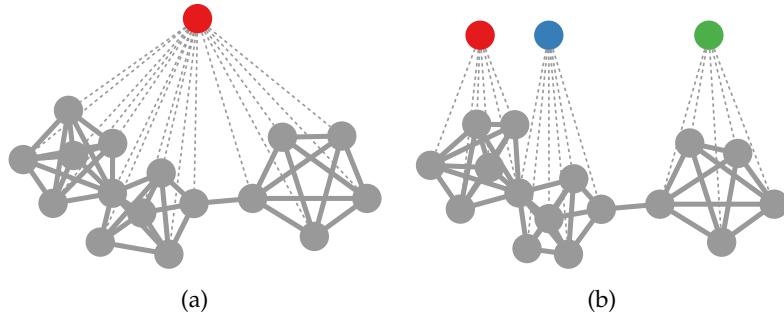


Figure 42.13: (a) Flat pooling via a virtual node (in red) connected with all nodes in the network by virtual edges (thin dashed lines). (b) Node clustering pooling with a fixed number of virtual nodes (in red, blue, and green) connected to dense subgraphs.

connecting each virtual node with a dense subgraph of the network – as we expect those node embeddings to be coherent with each other. The criterion to choose to which nodes a virtual node connects seems awfully familiar: it's quite literally community discovery (Part X). In fact, many special things you can do to find smarter communities might help you in finding better virtual nodes.

For instance, virtual nodes can share neighbors. You can see that the red and blue virtual nodes in Figure 42.13(b) share one, because it is a node well connected to two clusters. If you want to allow that, you can pick an overlapping community discovery algorithm (Chapter 38). And you don't have to give up the idea of having a single embedding for the entire graph. If you adopt a hierarchical community discovery mindset (Chapter 37), you can make virtual nodes aggregating information from virtual nodes – see Figure 42.14.

This is essentially what DiffPool²³ does, with the difference that DiffPool doesn't use an off-the-shelf hierarchical community discoverer, but learns the hierarchical structure together with the node embeddings. This is advantageous, because the properties you want to capture with the embeddings might not be all that correlated with the structure. For instance, if you want spectral embeddings, you should coarsen your graph with a technique that preserves its spectrum²⁴.

²³ Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. *Advances in neural information processing systems*, 31, 2018b

²⁴ Chenhui Deng, Zhiqiang Zhao, Yongyu Wang, Zhiru Zhang, and Zhuo Feng. Graphzoom: A multi-level spectral approach for accurate and scalable graph embedding. *arXiv preprint arXiv:1910.02370*, 2019

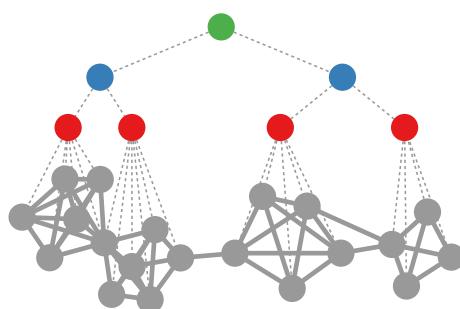


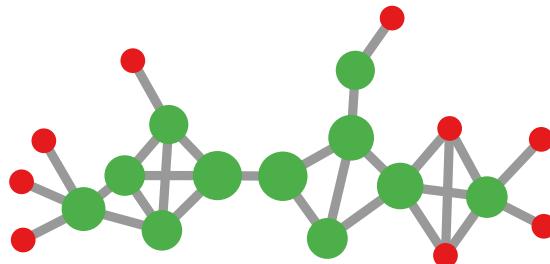
Figure 42.14: A hierarchical node clustering pooling.

Node Drop Pooling

Another way to eventually end up with a fixed-size embedding is to apply a three step strategy:

1. Score nodes by how relevant they are for the structure;
2. Select the least relevant nodes;
3. Coarsen the graph by dropping the nodes you just selected.

You stop when you have the desired number of nodes. Step #2 is usually trivial, you just select the k nodes with the lowest scores. Most of the work is done in step #1: we need to find good scoring rules. In Figure 42.15 I show an example of a simple scoring rule, by dropping nodes with the lowest betweenness centrality.



A general difference in how to calculate the scores is whether you're looking at the graph holistically in a single pass²⁵, or whether you do multiple passes. In the latter case, a typical approach is to do two passes, to score the relevance of nodes for their local structure (e.g. a community) and then globally for the whole network²⁶. For instance, a bridge nodes might be completely irrelevant locally because it's sparsely connected to the community, but crucial globally because it is the sole node keeping the community connected with the rest of the network.

While most attention is devoted to step #1, occasionally step #3 can be important. For instance, there are approaches that do not only drop the selected nodes. They will also drop some of the non-selected nodes, under the assumption that the selected nodes are "weird" by definition, and we want to build the graph embedding with a sample that is "representative", whatever representative means for our current analysis²⁷.

42.5 Summary

1. A node embedding is a vector of numbers representing a node. It should be small (with length $d \ll |V|$), dense, not dependent

Figure 42.15: A step of node drop pooling. The size of each node is proportional to its betweenness centrality. Nodes in red have betweenness centrality of zero, so we will drop them.

²⁵ Hongyang Gao and Shuiwang Ji. Graph u-nets. In *international conference on machine learning*, pages 2083–2092. PMLR, 2019

²⁶ Hongyang Gao, Yi Liu, and Shuiwang Ji. Topology-aware graph pooling networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(12):4512–4518, 2021

²⁷ Juanhui Li, Yao Ma, Yiqi Wang, Charu Aggarwal, Chang-Dong Wang, and Jiliang Tang. Graph pooling with representativeness. In *2020 IEEE International Conference on Data Mining (ICDM)*, pages 302–311. IEEE, 2020

on the order in which you explore the nodes, and representative of the node attributes and position in the network – similar nodes should have spatially close embeddings.

2. Depending on how you build them, embeddings can have multiple meanings and facilitate different analyses. For instance, you can use embeddings to determine node communities, or identify structurally equivalent nodes.
3. We want good node embeddings because they allow us to use any machine learning techniques to learn patterns in a complex network, which cannot be fed directly into those types of algorithms.
4. One way to build embeddings is to formulate a loss function that can be minimized by finding the eigenvectors of a corresponding transformation of the adjacency (or Laplacian) matrix of the graph. This is spectral embeddings.
5. To get edge embeddings you can perform simple combinations of the embeddings of the nodes they connect. To get graph embeddings you need to pool information from all the nodes in the network.
6. Pooling options for graph are: flat, where you don't look too hard at the structure of the graph; node clustering, where you collapse nodes into virtual nodes in a smart way; and node drop, where you recursively remove nodes that aren't salient for the network.

42.6 Exercises

1. Build a 2D node embedding for the network at <http://www.networkatlas.eu/exercises/42/1/data.txt>. You should build it by taking the first two singular values of the $D^{-1/2}AD^{-1/2}$ matrix.
2. Compare the embeddings you obtained from the previous exercise with the ones you get by taking the second and third eigenvectors of $(I - A)^T(I - A)$. Which one has the lowest loss according to $\sum_u \left(Z_u - \sum_v Z_v A_{uv} \right)^2$?
3. Compare the embeddings you obtained from the previous two exercises with the ones you get by taking the first two eigenvectors of $D^{-1/2}LD^{-1/2}$. Which one has the lowest loss according to $\frac{1}{2} \sum_u (Z_u - Z_v)^2 A_{uv}$?

4. Make 2D edge embeddings by pooling each of the three node embeddings from the previous exercises. Use the product as the pooling function. Plot them on a scatter plot, coloring existing vs non existing edges with two different colors.
5. Make a graph embedding of the network by hierarchically aggregating the node embeddings in two layers: first from the nodes into two virtual nodes corresponding to the two clusters of the network (which you have to find somehow) and then aggregating the two virtual nodes into a single final virtual node. Show the difference between these embeddings and flat embedding.

43

Random Walk Embeddings

In this chapter we continue looking at how to build node and graph embeddings. This time, we focus on doing so via random walks. I split this chapter out from the previous one, because there are a ton of random walk embedding methods. The reason is that making embedding via random walks hits three seducing spots. First, it is easy to do, there isn't much easier than doing a random walk. Second, it is flexible: there are many ways to bias your random walks, which can lead to meaningfully different embeddings. Finally, random walk embeddings work really well, they are very expressive. Let's take a look at them.

43.1 Basic Idea

The general approach of the embedding methods in the random walk category is to generate embeddings of a node by performing several – you guessed it – random walks. The embedding of node v is created by starting a bunch of random walks from v and noting down which nodes appear in these random walks.

We know this is useful, because it works in natural language processing (NLP). In the famous Word2Vec approach^{1,2} we want to make embeddings of words. To embed the word v we note down the other words used in sentences where we use v . Two words, u and v have similar embeddings if they are used around the same words – that means they must share some semantic or syntactic similarity. The reason why I'm describing Word2Vec is that this NLP technique can be translated directly into network terms. In fact, if you were a network supremacist (I know I am), you might even say that Word2Vec is a network algorithm, and the NLP people are quiet about it – just like in Section 42.2 we saw that image convolution is a network algorithm specifically designed for simple grids. Let's look at Figure 43.1.

In Word2Vec you are basically saying that each word is a node

¹ Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013a

² Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, 26:3111–3119, 2013b

Complex network analysis is easy

Network science is hard

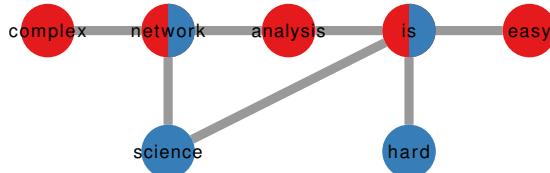


Figure 43.1: Representing two sentences as a graph, showing the connection between Word2Vec and network science.

and each sentence is a random walk. The random walk happens on a network we don't have, the one connecting each word to each other because of their semantic and syntactic relationships. But not having the network doesn't matter, because all you need to create the embeddings are the random walks. Similarly to the random walk approach in community discovery (Section 35.2), we base this on the assumption that random walks of two similar nodes will hit the same neighbors.

Once you realize this, you can apply Word2Vec to your networks and get node embeddings. This is quite literally what DeepWalk³ does. You simply perform a bunch of random walks – normally a certain number of random walks starting at a given node, with a given length, and so on. The more two words co-appear in a random walk, the more similar they are. Therefore, whatever your Z embeddings are, they need to be a good approximation of the co-appearance frequencies.

You can write this in mathematical form, all you need is to find the right loss function (Section 4.3) – which in this case is the log-likelihood. I'm going to simplify DeepWalk a bit, to help intuition. Let's assume \mathcal{W} is the set of our random walks. We can say that $|\mathcal{W}_{u,v}|$ is the number of random walks in which u and v co-appear. Then, we can consider this log-likelihood function:

$$\mathcal{L} = - \sum_{u,v \in \mathcal{W}} |\mathcal{W}_{u,v}| \log(p(u|v)),$$

where $p(u|v)$ is our estimation of the probability of observing node u in a walk that touched node v . This is a likelihood function because we're summing logs of probabilities, which is equivalent to multiplying the probabilities themselves – see Section 4.3 for a refresher with a simple case with coin tosses. We multiply each probability with $|\mathcal{W}_{u,v}|$ because we want a probability to weigh more if the two nodes co-appear in a random walk a lot of times. If that happens – meaning $|\mathcal{W}_{u,v}|$ is high – we want $p(u|v)$ to be high as well. This is what Figure 43.2 shows for a random network.

³ Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710. ACM, 2014.

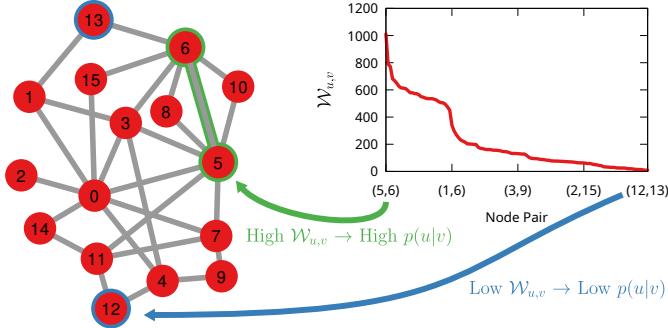


Figure 43.2: (Left) A graph. (Right) The graph's frequency of node co-appearance in a random walk (y axis) per node pair (x axis) sorted in descending order. Node pairs on the left of the frequency plot should get as similar embedding as possible.

Ok, so the question left unanswered is: how do we estimate $p(u|v)$? That's the job of our embeddings Z . If we find a really good Z , we can get high $p(u|v)$ for each high $|\mathcal{W}_{u,v}|$. We already know that the embedding similarity of u and v is $Z_u^T Z_v$, but the unfortunate thing is that this isn't a probability. No problem, we have the softmax function (Section 4.2) whose superpower is to transform anything into a probability. So:

$$p(u|v) = \text{softmax}(Z_u, Z_v) = \frac{e^{Z_u^T Z_v}}{\sum_{w \in V} e^{Z_u^T Z_w}}.$$

Note that this is not computationally efficient. In the softmax function, for the denominator, we need to loop over all the nodes in the network. To avoid that, we can approximate the softmax normalization via negative sampling, i.e. estimating the similarity of two nodes that co-appear in a random walk with a sample of the similarities of the nodes that don't co-appear with them. Since we're sampling, we can choose a suitably low number of samples, reducing computational complexity. Since this is all about random walks, you can perform the negative sampling with a random walk as well (Section 29.3), which basically means to pick nodes in the negative sample with a probability proportional to their degree.

While using negative sampling technically gives you a function that is not softmax, it approximates it well enough to be considered the same⁴.

Note that approaches in this family require you to set a lot of parameters. You need to decide how many walks per node you want to do, how long the random walks should be, etc. I don't know about you, but I don't like to think and to make lots of decisions. Some methods cover your back and can learn the optimal values for those parameters for you⁵.

⁴ Yoav Goldberg and Omer Levy. word2vec explained: deriving mikolov et al.'s negative-sampling word-embedding method. *arXiv preprint arXiv:1402.3722*, 2014

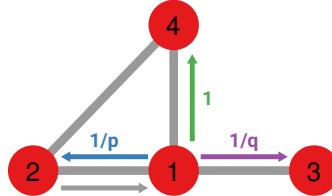
⁵ Sami Abu-El-Haija, Bryan Perozzi, Rami Al-Rfou, and Alexander A Alemi. Watch your step: Learning node embeddings via graph attention. *Advances in neural information processing systems*, 31, 2018

43.2 Variants on Simple Graphs

If we wanted to improve over DeepWalk, what would we do? For now, let's keep things simple and assume we only deal with simple graphs – graphs with a single edge type. We deal with heterogeneous / multilayer network later. We have a few options, the most common being to deploy high order random walks. This is popular because, depending on how the high order is implemented, you can get node embeddings with radically different meanings and properties.

Node2Vec

Node2vec⁶ notices that DeepWalk uses uniform random walks, picking the next step of the walk completely at random. However, there is something to gain by performing higher order random walks (Chapter 34). Figure 43.3 shows you an example.



In a high order random walk what matters is not only the node you're currently in, but also the node you come from. Node2Vec uses two parameters to exploit this information, p and q . In practice, it tries to modulate between a BFS-like strategy and a DFS-like one (Section 13.1). In Figure 43.3 you just followed the gray arrow to go from node 2 to node 1. How do you pick the next step?

Parameter p tells us how much we hate to backtrack. The step bringing you back to node 2 is weighted $1/p$. So if p is high, $1/p$ is low, and so it is unlikely we backtrack. But if p is below 1, it is more likely to backtrack, which is what a BFS exploration would do. Parameter q , instead, tells us how much we hate to explore parts of the network we haven't seen yet. Among all of node 1's neighbors, those that are not also neighbor with node 2 will get a $1/q$ weight to be explored. With low q we're trying to get as far from node 2 as possible – which is how DFS would behave. With high q we're more likely to explore a common neighbor between nodes 1 and 2.

DeepWalk is equal to Node2Vec when we set $p = q = 1$, so each neighbor of node 1 is treated equally. But setting p and q with various combinations leads to different random walks and, as a consequence, different embeddings Z which privilege some information about the network over some other.

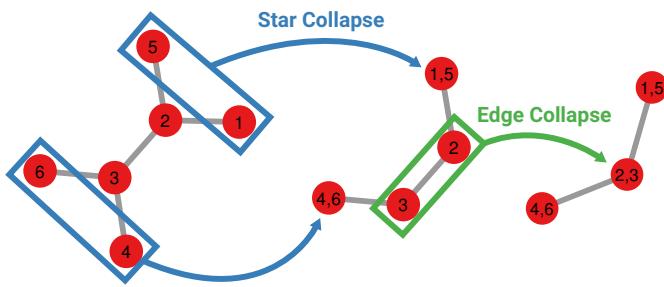
⁶ Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864. ACM, 2016.

Figure 43.3: The biased random walk implemented in Node2Vec. The gray arrow is the step we just took. The colored arrows show the reweighting of the various possibilities, with the p and q parameters. Blue for backtrack, green for common neighbor, purple for not shared neighbors.

HARP

HARP⁷ is a meta-strategy that one can apply to improve any other random walk embedding method, including DeepWalk and Node2Vec. The idea is to employ a smarter way to initialize the weights of the function summarizing your nodes – before you start performing your random walks. The way HARP does it is basically as a sort of “reverse pooling”. If you remember Section 42.4, node clustering pooling takes node embeddings and combines them hierarchically to obtain a graph embedding. Here, instead, we coarsen the graph hierarchically *before* we calculate the embeddings. We then calculate the embeddings on the coarsened graph. Then we refine the embeddings for each node on the original graph.

The coarsening is done with two approaches: edge and star collapsing, which I show in Figure 43.4. The idea is to preserve both first order relationship (edges) and second order relationships (paths of length two).



First one does star collapsing, where pairs of structurally equivalent nodes of degree one are merged. This means that, if you have k of them, you perform $\log_2 k$ merges. Then, you can do edge collapsing. First, you create set $E' \subset E$ such that no two edges in E' share a node. Then you collapse each edge in E' into a single node. You have to do star collapsing first because, if you did not, you'd have to perform k merges here instead of $\log_2 k$. So this ordering of operations is mostly a convenience to reduce computational complexity. On the other hand, the order in which one explores the graph matters for the final result, as the same graph can generate different E' sets depending on how you explore it. However, this turns out not to be too consequential, as the resulting embeddings normally aren't that sensitive to this choice.

An alternative approach tries to include in the random walk the global structure of the graph by raising the stochastic matrix to several powers and using these k step transition matrix to build the embedding⁸. You can also integrate node attributes in your representation^{9,10}.

⁷ Haochen Chen, Bryan Perozzi, Yifan Hu, and Steven Skiena. Harp: Hierarchical representation learning for networks. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018

Figure 43.4: The star and edge collapse processes in HARP. Nodes in the blue outlines merge during the star collapse phase, nodes in the green outlines merge in the edge collapse phase.

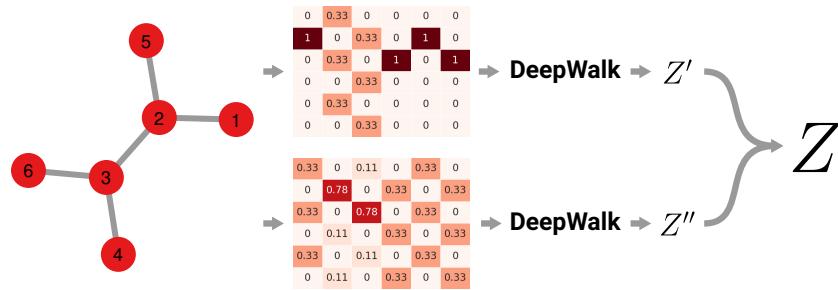
⁸ Shaosheng Cao, Wei Lu, and Qiongkai Xu. Graep: Learning graph representations with global structural information. In *Proceedings of the 24th ACM international conference on information and knowledge management*, pages 891–900, 2015

⁹ Zhilin Yang, William W Cohen, and Ruslan Salakhutdinov. Revisiting semi-supervised learning with graph embeddings. *arXiv preprint arXiv:1603.08861*, 2016b

¹⁰ Shirui Pan, Jia Wu, Xingquan Zhu, Chengqi Zhang, and Yang Wang. Tri-party deep network representation. *Network*, 11(9):12, 2016

LINE

LINE¹¹ is another approach worth mentioning. Just like in HARP and Node2Vec, LINE realizes that one needs to take into account both first and second order relationships between nodes, a feature that is absent from DeepWalk. I depict the general idea of LINE in Figure 43.5.



¹¹ Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. Line: Large-scale information network embedding. In *Proceedings of the 24th international conference on world wide web*, pages 1067–1077. International World Wide Web Conferences Steering Committee, 2015a

Figure 43.5: The general shape of LINE’s framework. From left to right the graph has two matrix representations for first and second order, on which we perform the random walks.

LINE represents a graph with two matrices: one with the first order and one with the second order proximities. LINE has its own definition of what these matrices should be, but simplifying a bit one could think that the first order proximities can be represented with the stochastic adjacency matrix A , since it gives you the transition probability via a random walk. Then, the second order proximities could be the squared stochastic adjacency matrix, A^2 , which gives the probability of moving between the nodes with a random walk of length two.

Once you have these two matrices, you can use the same loss function of DeepWalk, twice. In the first pass, you create embeddings minimizing the loss over the first order proximities, and in the second pass you minimize the loss over the second order proximities. Then, you combine the embeddings.

Struct2Vec

Struct2Vec¹² follows the guiding principle of structural similarity. The idea is to guide the random walker to explore nodes that are structurally similar. The key here is that there could be structurally similar nodes that are very far apart in the network – and therefore unlikely to appear in a naive random walker. How do we fix the problem?

The idea here is to build a multilayer graph G' out of your original G . Figure 43.6 shows how it looks like. This multilayer graph has k layers, where k is the diameter of G . Each layer contains the same nodes as G . In each layer, all pairs of nodes in G are connected

¹² Leonardo FR Ribeiro, Pedro HP Saverese, and Daniel R Figueiredo. struc2vec: Learning node representations from structural identity. In *SIGKDD*, pages 385–394, 2017

– although some edges are allowed to have a weight of zero and so appear non-existent, I explain later why that is the case. The embeddings are built using DeepWalk on G' rather than G .

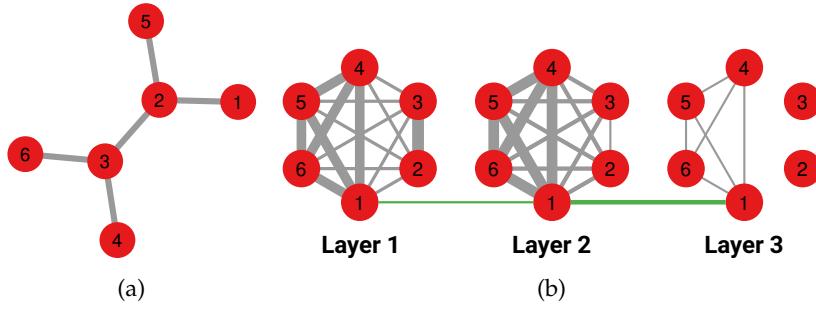


Figure 43.6: (a) The original graph. (b) The multilayer network Struct2Vec builds. I only show one set of interlayer couplings, in green, for node 1. The line thickness is proportional to the edge's (and coupling's) weight.

You might think: *what's the point of doing random walks on a giant clique?* Well, it all comes down to the weights of those edges, which guide the random walker to explore structurally similar nodes. A u, v edge in layer i has a weight proportional to the structural similarity of u and v at distance i . In practice, you take the set of all nodes at i distance to u and you compare it with the set of all nodes at i distance to v . Struct2vec compares them using the degrees of these nodes, enforcing a structural similarity function that says that two nodes are i -structurally similar if the nodes at distance i from them have similar degree sequences. Note that, in Figure 43.6, some nodes appear disconnected: they are actually connected with an edge weight of zero, because of their low structural similarity.

G' is more useful than G because, when the random walk transitions to any layer that is not the first, it can explore nodes that are far apart more easily. Once the random walker reaches the last layer k , in one step it can transition between the two farthest apart nodes in G – because that layer represents the diameter.

The random walker can transition to a different layer via interlayer couplings – in Figure 43.6 I show only two of them, in green. These couplings connect node u to itself in the neighboring layers – so u in layer i connects to itself in layers $i + 1$ and $i - 1$. The strength of the coupling is proportional to u 's average similarity to all other nodes in layer i . If u is similar to all layer i nodes, then it's more informative to go up one layer to $i + 1$.

43.3 Heterogeneous Graph Embeddings

Graph embedding techniques, regardless of their chosen approach, need to be adapted to be able to handle heterogeneous¹³ and multilayer networks, networks where nodes and/or edges can be of

¹³ Shiyu Chang, Wei Han, Jiliang Tang, Guo-Jun Qi, Charu C Aggarwal, and Thomas S Huang. Heterogeneous network embedding via deep architectures. In SIGKDD, pages 119–128, 2015

multiple different types. For instance, one could adopt the metapath approach¹⁴ we saw in Section 24.2 when talking about link prediction in multilayer networks. The problem with heterogeneous networks is that there are some node types that are more dominant – i.e. connected – than others. Their representations would then be extremely noisy. In metapath2vec, the problem is solved by switching one’s attention from nodes to metapaths.

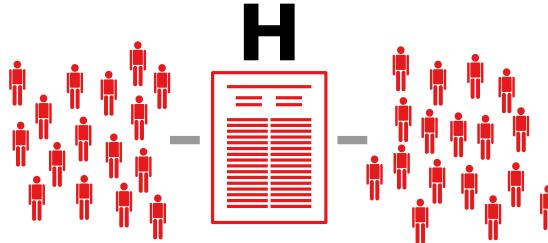


Figure 43.7 shows a stylized depiction of the issue. The paper reporting the discovery of the Higgs boson has 5,154 co-authors. Every pair of co-authors is a valid path in the co-authorship network. As a result, the embedding of the node representing the paper is extremely noisy, as it could be visited by 10^7 walks of length 2, not even counting the ones that could lead you back to your origin node. But by instead focusing on each of the metapaths (Section 24.2), we will have a much cleaner signal.

Another way to deal with heterogeneous networks is to infer different embeddings for different edge types¹⁵. Let’s say you have two nodes of different types: a conference venue c and a topic t . They are connected to the same node a , an author, who published in that conference and in that topic, but not at the same time. Thus, c and t are not connected and not similar to each other. Then a will be equidistant from them. However, when we focus on the “author-topic” edge type, a will be closer to t , and when we focus on the “author-conference” edge type, a will be closer to c .

Knowledge Graph Embedding¹⁶ is particularly popular and well-suited for these techniques and that’s why I dedicate some space to it. This is the application of graph embedding techniques on knowledge graphs. Knowledge graphs are graphs of entities related to each other by a certain semantic. *De facto*, these are special heterogeneous networks. Examples of knowledge graphs are, for instance, DBpedia¹⁷, Wikidata, and more. What makes them special is their massive size and the rich semantics behind the connections.

In such cases, graph embedding techniques acquire a special meaning. We are now establishing relations between concepts, building a way to determine that, for instance, knives are to cooks what cameras are to movie directors. This allows us to create automatically new

¹⁴ Yuxiao Dong, Nitesh V Chawla, and Ananthram Swami. metapath2vec: Scalable representation learning for heterogeneous networks. In *SIGKDD*, pages 135–144, 2017

Figure 43.7: A troubling set of connections in an heterogeneous network: a paper with hundreds of co-authors.

¹⁵ Yu Shi, Qi Zhu, Fang Guo, Chao Zhang, and Jiawei Han. Easing embedding learning by comprehensive transcription of heterogeneous information networks. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2190–2199, 2018

¹⁶ Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. A three-way model for collective learning on multi-relational data. In *Icml*, volume 11, pages 809–816, 2011

¹⁷ Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick Van Kleef, Sören Auer, et al. Dbpedia—a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic Web*, 6(2):167–195, 2015

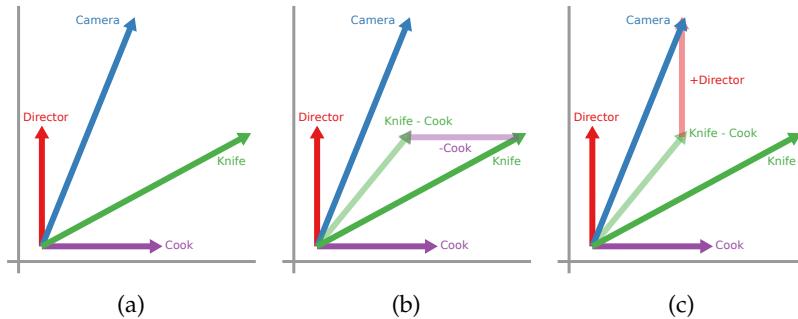


Figure 43.8: (a) Some node embeddings we learned from a knowledge graph. (b-c) Manipulating node embeddings by adding-subtracting their vector representations can uncover new knowledge.

connections, previously unknown relationships, in the knowledge graph. Figure 43.8 shows an example.

In the figure, I create the new abstract concept of “knife minus cook”. By adding “movie director” to such an abstract concept, I discover the camera-director relationship as equivalent to knife-cook. Thus the “knife minus cook” is a useful concept, which we might use for all professions.

Such operations can be done directly from text using Natural Language Process techniques. What makes this special, is that the embeddings were created not by looking at natural text, but at the structure of a knowledge graph. To do this, we need specialized tools^{18,19,20}. As usual, you can find more details on this problem in dedicated survey papers^{21,22}.

43.4 Applications

So, what can you do with node embeddings? To cut a story short: practically everything. We already saw a couple of applications, in this chapter and in others. Without repeating myself too much, I’ll spend only a couple of words about the applications we’ve already seen in the book:

- Community discovery (Part X): if you build embeddings encoding the similarity of neighborhoods of nodes (like in Figure 42.1), then you can feed them into any data clustering algorithm such as kMeans or DBSCAN and get communities;
- Structural equivalence (Section 15.2): if you build embeddings encoding the structural similarity of nodes (like in Figure 42.4) the same data clustering algorithms will identify structurally equivalent nodes;
- Link prediction²³ (Part VII): by comparing the embeddings of edges that exist against those of edges that do not, you can figure out which edges should exist, and be predicted;

¹⁸ Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. Knowledge graph embedding by translating on hyperplanes. In *Twenty-Eighth AAAI conference on artificial intelligence*, 2014b

¹⁹ Yankai Lin, Zhiyuan Liu, Maosong Sun, Yang Liu, and Xuan Zhu. Learning entity and relation embeddings for knowledge graph completion. In *Twenty-ninth AAAI conference on artificial intelligence*, 2015

²⁰ Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *European Semantic Web Conference*, pages 593–607. Springer, 2018

²¹ Maximilian Nickel, Kevin Murphy, Volker Tresp, and Evgeniy Gabrilovich. A review of relational machine learning for knowledge graphs. *Proceedings of the IEEE*, 104(1):11–33, 2015

²² Quan Wang, Zhendong Mao, Bin Wang, and Li Guo. Knowledge graph embedding: A survey of approaches and applications. *IEEE Transactions on Knowledge and Data Engineering*, 29(12):2724–2743, 2017a

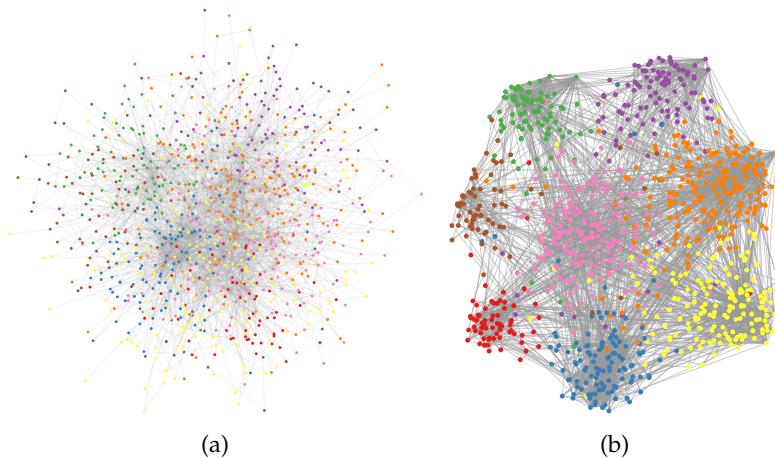
²³ Mingdong Ou, Peng Cui, Jian Pei, Ziwei Zhang, and Wenwu Zhu. Asymmetric transitivity preserving graph embedding. In *SIGKDD*, pages 1105–1114, 2016

- Node role detection (Chapter 15): you can build embeddings to conform to your expectation of specific roles in the networks, and classify all nodes based on that.

Here I want to unpack the topic of network visualization, since I teased it in Section 42.1. We’re going to do a deep dive on network visualization in Part XIII, but hopefully Section 42.1 did a good job to make you understand why embeddings are useful in this case. One could reduce each node into two or three dimensions, and then simply use them as the (x, y, z) coordinates to display them spatially. After all, this is what is already being done with traditional dimensionality reduction techniques: it is the whole selling point of the quasi-magical t-distributed stochastic neighbor embedding²⁴ (t-SNE) or UMAP²⁵.

In fact, the standard approach includes two steps²⁶. First, you use your favorite graph embedding technique to reduce all nodes to vectors of length d . Then, you apply a dimensionality reduction technique to reduce those d dimensions to two and you plot the result. Figure 43.9 shows what happens to a relatively simple LFR benchmark with a high mixing parameter – i.e. communities which tend to share lots of inter-community edges. In Figure 43.9(a) you see what a classical network layout would do, while Figure 43.9(b) shows you a combination of Node2Vec embeddings, reduced with UMAP. In the latter case, the community structure is more evident. Imagine what you could do if you could make a more advanced node embedding, taking into account node attributes, edge weights, and more!

Another natural application of node embeddings we haven’t seen so far is graph summarization. We’re going to examine the problem more in details in Chapter 46. For now, suffice to say that, if two



²⁴ Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9 (Nov):2579–2605, 2008

²⁵ Leland McInnes, John Healy, and James Melville. Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*, 2018

²⁶ Jian Tang, Jingzhou Liu, Ming Zhang, and Qiaozhu Mei. Visualizing large-scale and high-dimensional data. In *WWW*, pages 287–297, 2016

Figure 43.9: An LFR benchmark. The node color represents its community affiliation. (a) Force directed layout (Section 51.1). (b) layout based on node embeddings via random walks.

nodes have very similar embeddings, we could just collapse them into the same node. The benefit would be to have a smaller structure to analyze – less memory and time consumption for your algorithm – while still preserving the general properties of the network as a whole.

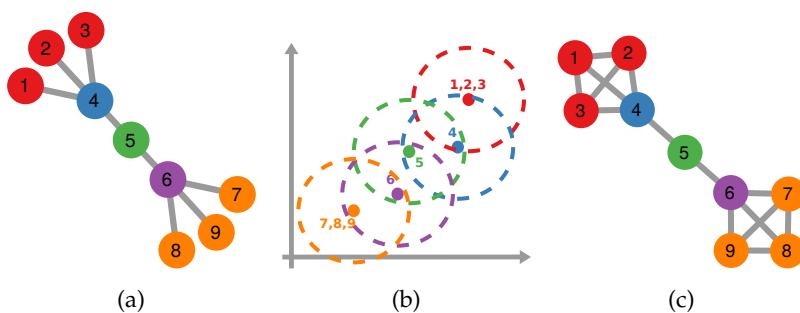
One common way is to do the following. First, take your graph and calculate its node embeddings. These, as we saw, are spatial representations in d dimensions. Then, calculate the pairwise distance between all these points. You can use the Euclidean distance or whatever floats your boat. At this point, you can establish a certain distance k . Nodes that are closer than k get connected together, otherwise they remain disconnected. This is a graph reconstructed from the embeddings. You can compare the reconstructed graph with the original one. The more similar they are, the better the embedding worked. Figure 43.10 shows an example of this procedure.

You can consider closer points as the same node and summarize your graph this way. For instance, nodes 7, 8, 9 have the same embedding and thus they could be considered to be the same node, just like nodes 1, 2, 3.

Other classical applications of graph embeddings are node ranking²⁷, solving classical combinatorial problems like the traveling salesman problem²⁸, and network alignment²⁹, the problem of figuring out which nodes from two distinct networks might refer to the same real world entity. This is still limited to the realm of network analysis for network analysis' sake, but we know we can use networks – and, therefore, graph embeddings – to solve many more problems. Some include natural language processing³⁰, computer vision³¹, and bioinformatics³², to cite a few pointers.

43.5 Limitations

The classical node embeddings approaches had a lot of issues: they didn't capture structural similarity, they ignored node and edge attributes, etc. These issues were fixed with various new methods,



²⁷ Namyong Park, Andrey Kan, Xin Luna Dong, Tong Zhao, and Christos Faloutsos. Estimating node importance in knowledge graphs using graph neural networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 596–606, 2019.

²⁸ Elias Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. In *Advances in Neural Information Processing Systems*, pages 6348–6358, 2017

²⁹ Mark Heimann, Haoming Shen, Tara Safavi, and Dana Koutra. Regal: Representation learning-based graph alignment. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, pages 117–126, 2018

³⁰ Zhouhan Lin, Minwei Feng, Cicero Nogueira dos Santos, Mo Yu, Bing Xiang, Bowen Zhou, and Yoshua Bengio. A structured self-attentive sentence embedding. *arXiv preprint arXiv:1703.03130*, 2017

³¹ Sijie Yan, Yuanjun Xiong, and Dahua Lin. Spatial temporal graph convolutional networks for skeleton-based action recognition. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

³² Marinka Zitnik, Monica Agrawal, and Jure Leskovec. Modeling polypharmacy side effects with graph convolutional networks. *Bioinformatics*, 34(13):i457–i466, 2018.

Figure 43.10: (a) An example graph. (b) One of the possible embeddings of (a). The dashed circles encapsulate the radius inside which we consider the nodes as connected together. (c) The version of graph (a) reconstructed through its two dimensional embedding.

which we have explored in the past two chapters. There remains one fundamental issue with the majority of the most popular node embedding approaches. Throwing some jargon at you, the issue is that these methods are not inductive, but transductive. What does that mean?

Figure 43.11 shows an example. Here we’re in a fairly classical scenario, where we try to train our algorithm by dividing nodes into a training set V_t and a validation set V_v , to figure out whether we’re overfitting. If you remember Section 4.1, these two sets are disjoint: all nodes in the validation set are not part of the training set, or $V_t \cap V_v = \emptyset$. How does that affect your node embeddings?

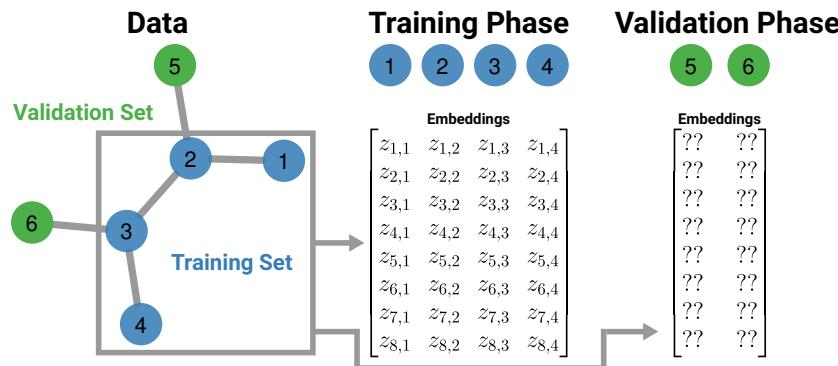


Figure 43.11: The original network is divided in training nodes (in blue) and validation nodes (in green). We create the embedding (grey outline) using only the training nodes and their connections.

If you’re using a spectral approach, you can only calculate the eigenvectors on the matrix including only the nodes in the training set. This means Z is a $V_t \times d$ matrix. In the Figure 43.11, we have eight dimensional embeddings, so Z is 4×8 . Here you have no embeddings for the nodes in the validation set!

You might be tempted to perform validation on the full network. But adding the nodes from the validation set means recalculating *all* the embeddings, even those of the nodes in the training set – meaning you just trained on embeddings that are different than the ones you use in the validation. Oops. The same holds for random walk embeddings: the presence/absence of new nodes will change the random walk content, by making some node pairs all of a sudden much more (or less) likely to appear together in a random walk.

This issue is not limited to the mere practicality of making training and validation sets, though. This limitation also makes it technically hard to deal with a network that evolves over time. Every time some nodes/edge can come in – or out – of your structure, all your embeddings will change and need to be recalculated from scratch. There are ways to go around this issue³³, but in most cases we’ll need to dive into different techniques designed specifically for dynamic

³³ Jundong Li, Harsh Dani, Xia Hu, Jiliang Tang, Yi Chang, and Huan Liu. Attributed network embedding for learning in a dynamic environment. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pages 387–396, 2017a

graphs^{34,35}, or to solve a specific subproblem.

In the next chapter we'll see how deep learning can give us more natural inductive methods that do not suffer from this issue.

43.6 Summary

1. The basic idea of random walk embeddings is to perform many random walks on a network and to minimize the loss function describing those random walks. Two nodes have similar embeddings if they co-appear often in the same random walk.
2. You can get different and meaningful embeddings by modifying the random walk strategy. You can have high order random walks exploring the graph either in depth or in width, you can collapse the network in various ways to explore structural similarities, and more.
3. With heterogeneous graphs your strategy needs to take into account that nodes and edges of different types might require special rules to be traversed.
4. Knowledge graphs are a special case of heterogeneous networks: they express relations between real world concepts. In this scenario, embeddings can help us to uncover previously unknown meanings – i.e. groups of nodes in the knowledge graph.
5. Node embeddings can be applied to many problems we saw in the book: community discovery, link prediction, and more. Visualization is another natural one: by building 2D embeddings we can interpret them as spatial coordinates to display our network.
6. Many common node embedding methods can only build and use embeddings for nodes that are already present in the graph. If you add new nodes – because of a temporal graph or a train-validation split – you need to recompute all embeddings from scratch.

43.7 Exercises

1. Perform 10,000 random walks of length 6 in the network at <http://www.networkatlas.eu/exercises/43/1/data.txt>. Build embeddings with $d = 32$ using Word2Vec (I suggest using the gensim implementation). Reduce them to two dimensions using `sklearn.manifold.TSNE`. Visualize the network by using the 2D embeddings as spatial coordinates. Use <http://www.networkatlas.eu/exercises/43/1/nodes.txt> to determine the node colors.

³⁴ Seyed Mehran Kazemi, Rishabh Goel, Kshitij Jain, Ivan Kobyzev, Akshay Sethi, Peter Forsyth, and Pascal Poupart. Representation learning for dynamic graphs: A survey. *Journal of Machine Learning Research*, 21(70):1–73, 2020

³⁵ Meng Liu, Yue Liu, Ke Liang, Wenxuan Tu, Siwei Wang, Sihang Zhou, and Xinwang Liu. Deep temporal graph clustering. *ICLR*, 2024

2. Use the Word2Vec embeddings reduced to 2D with tSNE you found in the previous exercise to find 4 clusters in the data using any clustering algorithm (if kMeans, set $k = 4$, otherwise you can use DBSCAN and find the eps parameter giving you 4 clusters). What is the NMI (use the `sklearn` function to calculate it) of the clusters with the ground truth you can find at <http://www.networkatlas.eu/exercises/43/1/nodes.txt>?
3. Compare the NMI score from the previous exercise to the one you would get from a classical community discovery like label propagation. Note: both methods are randomized, so you could perform them multiple times and see the distributions of their NMIs.
4. Use the Word2Vec embeddings reduced to 2D with tSNE you found in the previous exercise as a link prediction score (if Z_u and Z_v are node embeddings, then the score for edge u, v is $Z_u^T Z_v$). Draw the ROC curve of your predictions, assuming that the true new edges are the ones you can find in <http://www.networkatlas.eu/exercises/43/4/newedges.txt>.
5. Is the AUC you get from the previous question better or worse than the one you'd get from a classical link prediction like Jaccard, Resource Allocation, Preferential Attachment, or Adamic-Adar?

44

Message-Passing & Graph Convolution

In this chapter we start dealing with the development of deep learning methods that work on a graph. The difference with what we've seen so far is that the embedding methods from the previous chapters encode the graph's structure so that we can use non-graph deep learning methods to learn something about the graph. Here we scrap that approach and try to adapt the deep learning methods to be able to work and learn directly on the graph. Which is why the organization of this chapter and the next has one section per deep learning approach, explaining how that specific deep learning approach can be performed on a graph.

For these two chapters to make most sense and to understand most of the lingo, you'd be required to know more about what deep learning is. This is outside the scope of this book, as it deserves a (couple of) book(s) on its own. For this reason, I suggest you some readings^{1,2,3}, which might help you figuring out better what is going on.

A note about terminology. Formally speaking, graph convolutional networks and other more sophisticated approaches such as graph attention networks are all part of the family of message-passing graph neural networks, which is the most general model and it includes the two. *However*, to take things step by step and make them more digestible, I'll slightly abuse the terminology. For now on, "message-passing graph neural network" in this book refers not to the *general* class, but to the *basic* class. This makes it easier to introduce graph convolutional networks for the special thing they do. But, technically speaking, there is no difference between graph convolutional networks and message-passing graph neural networks, because graph convolutional networks are a type of message-passing graph neural networks.

¹ Li Deng, Dong Yu, et al. Deep learning: methods and applications. *Foundations and Trends® in Signal Processing*, 7(3-4):197–387, 2014

² Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015

³ Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016

44.1 Message-Passing Graph Neural Networks

Before I start throwing math at you, I'll start with a simple graphical example of what a message-passing graph neural network (MPGNN) is. The reason is that this type of architecture is the basis of all the variants that will follow. Understanding this will provide the foundation to figure out what happens in graph convolution networks and in other models. The example I give here is pretty cartoonish, but it should help understanding. It is based on the early attempts to define graph neural networks^{4,5,6}, which have been constantly refined over time⁷.

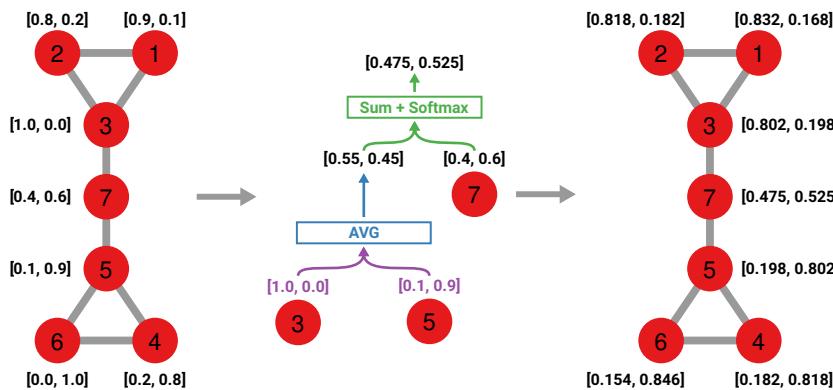
One key component in this framework is that the nodes start with some feature matrix, which we will call H^0 . H^0 is a $|V| \times d$ matrix, associating each node with a vector of length d of features. If your graph doesn't have node attributes, you can always use some structural properties as attributes, for instance their degree.

The key process of a message-passing graph neural network is... message passing. At each iteration – which we call layer – we do three things:

1. Each node formulates a *message* based on its current node attributes and passes it to all its edges;
2. Each node receives the messages from its neighbors and *aggregates* them;
3. Each node *updates* its own attributes based on the aggregated messages.

Figure 44.1 shows this process graphically for a node, and the final result for the whole network.

Note how I emphasized the words “message” “aggregate” and “update”. That is because those are the functions that power the



⁴ Franco Scarselli, Sweah Liang Yong, Marco Gori, Markus Hagenbuchner, Ah Chung Tsoi, and Marco Maggini. Graph neural networks for ranking web pages. In *The 2005 IEEE/WIC/ACM International Conference on Web Intelligence (WI'05)*, pages 666–672. IEEE, 2005

⁵ Marco Gori, Gabriele Monfardini, and Franco Scarselli. A new model for learning in graph domains. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 2, pages 729–734. IEEE, 2005

⁶ Christian Merkwirth and Thomas Lengauer. Automatic generation of complementary descriptors with molecular graph networks. *Journal of chemical information and modeling*, 45(5):1159–1168, 2005

⁷ Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *ICML*, pages 1263–1272. JMLR.org, 2017

Figure 44.1: A potential message-passing graph neural network architecture. From left to right: a network with 2D node attributes; the message-passing procedure for node 7; the result of the message-passing for all nodes. The different colors in the message-passing procedure highlight the three steps: message composition and passing in purple, aggregation in blue, and update in green.

special juice of MPGNNs. Depending on how you compose the messages, aggregate them, and use them to update a node, you can lead your graph neural network to achieve different objectives. In the figure I decided that the message composition was just the identity, i.e. the node attributes are passed “as they are”. The aggregation function was the element-wise average, so nodes 3 and 5 aggregate to $[(1 + 0.1)/2, (0 + 0.9)/2] = [0.55, 0.45]$. Then the updating function was sum and then softmax – to keep the property that the node attributes for each node sum to 1. So node 7 updates as $\text{softmax}([0.4 + 0.55, 0.6 + 0.45]) = [0.475, 0.525]$. Of course you can use any arbitrary combination of functions here.

One round of messaging, aggregating, and updating constitutes one layer of the MPGNN. In the figure, that results in building a new attribute matrix. We call it H^1 , because it is the result of the first layer applied to H^0 . The power of MPGNNs is that you can pile up multiple layers. Figure 44.2 shows you how we get H^2 from the second layer, which boils down to re-applying the same aggregate and update functions to H^1 .

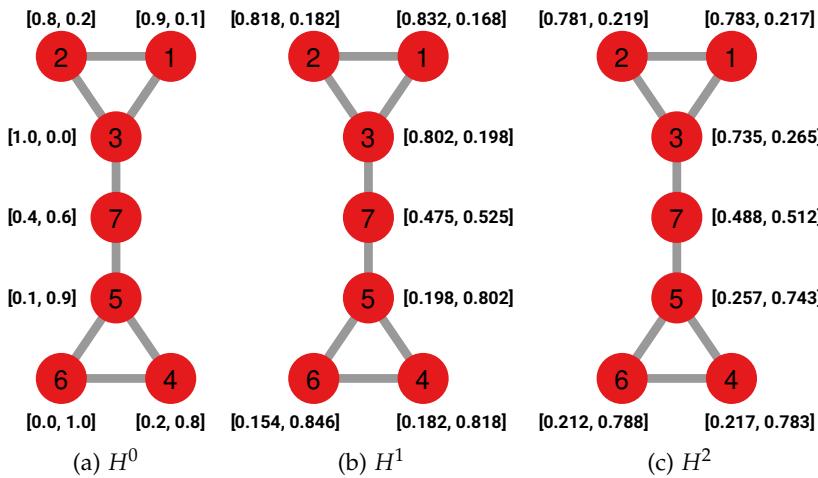


Figure 44.2: (a) The original network. (b) The result of applying the first layer of message-passing. (c) The result of applying the second layer.

You can see how this can be powerful: in H^2 we’re starting to delineate very clearly not only the clustered structure of the network, but also which nodes are in which position in this clustered structure, whether on the border or deeply embedded in the community.

Why do we want to have multiple layers? Because in this way, for each node v , we can pool information from v ’s extended neighborhood. At the first layer, we only get information from its direct neighbors. At the second layer, however, those neighbors have embedded some information from *their* neighbors, which is now passed to v as well. At layer l , H_v^l encodes information at l hops from v . More interestingly, this “information” that is passed is not merely about the

features, but it depends on the structure of the network as well. H^1 is not dependent only on H^0 but also on the structure of G ! The same H^0 on a different G will produce a different H^1 . Isn't that neat?

Before I move on to the sections showing where this neatness ends, a few concluding remarks. So far we have worked with simple graphs. However, you know that networks come in various flavors, and your MPGNN should take that into account. For instance, if you're working with multilayer networks, you need your aggregate and update functions to run over the various different edge types^{8,9}. However, a more flexible approach is to realize that the edge type is nothing more than a feature of the edge. So we can treat this as any other feature, and including it in the aggregate function that we use to pass the message to the nodes¹⁰.

Note also that there's no one stopping from having edges passing messages to each other, or to summarize a graph embedding by updating at each layer the graph's embedding with a combination of its node and edge embeddings^{11,12}.

44.2 Limits of the Message-Passing Approach

Relations to Isomorphism Test

It's worth to make a quick pause and point back to Section 41.3, where I explained the Weisfeiler-Lehman isomorphism test. In that section I told you that this isomorphism test is important because it approximates the behavior of MPGNNs. If you go and read this section again, you'll realize that Weisfeiler-Lehman is exactly a message-passing network, with specific aggregating and updating functions. In fact, there are proofs showing that this style of message-passing network can distinguish – at best – the same structures of Weisfeiler-Lehman¹³. That is to say that, in the cases where Weisfeiler-Lehman fails the isomorphism test, a MPGNN would fail to distinguish structures – returning the same H^l for different G s. Bummer.

The full proof is sophisticated and a bit much to include in this book, but I'll give you a superficial intuition about it. Consider Figure 44.3(a). Here I show you the networks I used in Figure 41.5 to show you a case in which Weisfeiler-Lehman gets it wrong: it says the graphs are isomorphic when they aren't. I use as simple 1D embedding the node degrees. The aggregate and update functions of the MPGNN are the sum. As you can see, once we pass through a layer, both networks generate the same embeddings, even though the structures are different and one would expect them to lead to different embeddings.

On the other hand, in Figure 44.3(b) we have the graphs I used in

⁸ Diego Marcheggiani and Ivan Titov. Encoding sentences with graph convolutional networks for semantic role labeling. *arXiv preprint arXiv:1703.04826*, 2017

⁹ Komal K Teru and William L Hamilton. Inductive relation prediction on knowledge graphs. *ICML, Virtual*, 2020

¹⁰ Koustuv Sinha, Shagun Sodhani, Jin Dong, Joelle Pineau, and William L Hamilton. Clutr: A diagnostic benchmark for inductive reasoning from text. *arXiv preprint arXiv:1908.06177*, 2019

¹¹ Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018

¹² Pablo Barceló, Egor V Kostylev, Mikael Monet, Jorge Pérez, Juan Reutter, and Juan-Pablo Silva. The logical expressiveness of graph neural networks. In *8th International Conference on Learning Representations (ICLR 2020)*, 2020

¹³ Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018a

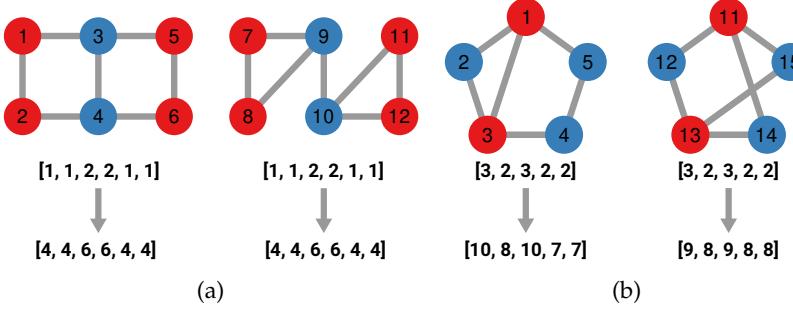


Figure 41.4 to show when Weisfeiler-Lehman gets it right. As you can see, using the same 1D embeddings and the sum as the aggregate and update functions, the different structures generate different embeddings, as one would expect.

There are ways to fix this problem. A simple one involves extending H^0 with additional features, specifically features that associate each node with a unique ID¹⁴. In this way, the MPGNN can identify when two nodes share a neighbor, which will allow it to distinguish structures Weisfeiler-Lehman cannot. It's relatively simple to do this: just concatenate to H^0 the $|V| \times |V|$ identity matrix I . This is all fine and dandy, but the problem is that now we lose the permutation invariance requirement, since now we're forcing the graph to have a specific node ordering. This can be solved by aggregating information from all possible orderings, basically running the MPGNN for all possible permutation of nodes. While this will lead to permutation invariance, one should note that trying all permutations of nodes is something you can do only for small graphs with few nodes.

More tractable solutions exploit high order dynamics (Chapter 34). For instance, in the k-MPGNN approach¹⁵ one realizes that what I just described is merely a 1-MPGNN, where nodes get messages individually from their neighbors. But we can imagine a 2-MPGNN where each pair of nodes receives messages from its neighboring pairs of nodes – pairs with which they share a node. Then you can have as many k-MPGNN as you want, with each set of k nodes communicating with all other sets of k nodes with which they share $k - 1$ nodes. Figure 44.4(a) shows an example, where you can see the two graphs should lead to different embeddings, as the node pair in blue gets updates from a different number of adjacent node pairs – in green – in the two cases.

Another high-order tactic is to use simplicial complexes¹⁶. To cut a long story short, the idea here is that messages not only pass through edges, but also through simplicial complexes, with their own customized aggregate and update functions. This solves many problems because, as you can see in Figure 44.4(b), even if Weisfeiler-Lehman

Figure 44.3: Two non-isomorphic networks with 1D embeddings passing through one layer of a simple MPGNN.
 (a) Leading to the same result.
 (b) Leading to different results.

¹⁴ Ryan Murphy, Balasubramaniam Srinivasan, Vinayak Rao, and Bruno Ribeiro. Relational pooling for graph representations. In *International Conference on Machine Learning*, pages 4663–4673. PMLR, 2019

¹⁵ Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and lehman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4602–4609, 2019

¹⁶ Cristian Bodnar, Fabrizio Frasca, Yuguang Wang, Nina Otter, Guido F Montufar, Pietro Lio, and Michael Bronstein. Weisfeiler and lehman go topological: Message passing simplicial networks. In *International Conference on Machine Learning*, pages 1026–1037. PMLR, 2021

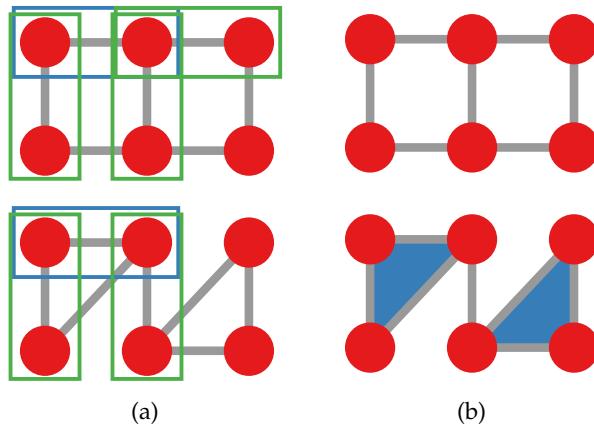


Figure 44.4: (a) A 2-MPGNN where I identify with a green outline all node pairs adjacent to the node pair in the blue outline. (b) A MPGNN with simplicial complexes of three nodes, highlighted with a blue fill.

thinks that the two graphs are isomorphic, a simplicial MPGNN does not, because it sees that one has two simplicial complexes that the other does not. This approach of using high order structures connects to a more general “topological” learning, of which graph neural networks are a special case.

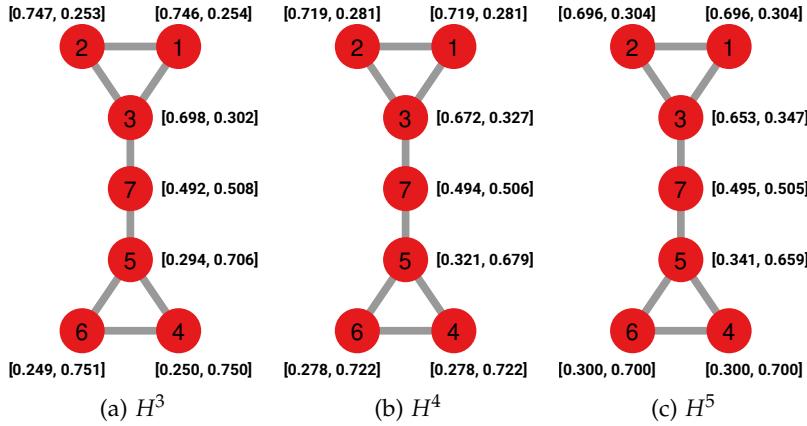
Smoothing Problems

There's another issue with MPGNNS. As I told you, more layers allow you to pool information from farther away in the network. One thing you might be tempted to do is to pool information from the entire network. That is to say, you could decide to have l layers so that l is the diameter of the network. This way you know that, in the end, even the farthest away pair of nodes will exchange some information. However, you start to run into troubles, a specific kind of trouble we call "smoothing".

To understand smoothing, let's take the process we started in Figure 44.2 and let's keep unfolding it. In Figure 44.2 we stopped at the second layer, so in Figure 44.5 I start from the third and I continue for a few layers. What do you observe?

All embeddings are getting more and more similar to each other. That is, the MPGNN is losing its capability to distinguish the nodes in the network. A few more layers, and each node will get the same embeddings. In MPGNN, you either stop early or you run long enough to become a vector of constants. You knew this was going to happen, because this is exactly the same dynamics I showed you back in Section 11.6, when we talked about reaching a consensus in the network. From that section you know that the eigenvalues of the Laplacian will tell you when you will reach the consensus, so you should stop far before that point.

Well, actually there are other things you can do to prevent smooth-



ing. One of them is to use the skip-connection approach. The simplest way to do it is by not blindly accepting the message arriving from the neighbors. For each node v , you can run your update function on the message v receives plus v 's representation in the previous layer¹⁷. In practice, v remembers what it was and only updates a little bit with the new information. This makes information propagate more slowly in the MPGNN which, given that our problem is over-smoothing, it's a feature, not a bug. The way you combine the new message with the old representation is up to you, you can figure out which function works best for your case¹⁸. Since you're using a node's embedding to update itself, this is a type of recurrent neural network¹⁹.

Another neat thing you can do is the so-called jumping knowledge network²⁰. In this approach, the final embedding for node v in your MPGNN is not the final layer, but a concatenation of v 's intermediate embeddings from all layers. If you had d dimensional embeddings and l layers, the final embedding for v has not length d , but length ld .

You could also add layers that do not pass messages among nodes²¹, such as ones coming from more classical neural network designs.

Over-Squashing

So, adding too many layers will cause smoothing, which is a bad news. The even worse news is that it won't even help with the original problem, which is that there are some nodes in the periphery whose messages are going to be largely ignored. Even if you have l layers in a network with diameter l , when the message from the peripheral node arrives to its most distant nodes, it has changed so much it became unrecognizable. Figure 44.6 shows you why: the original message – the pure blue color representing its content – has

Figure 44.5: The result of applying several more layers to the network in Figure 44.2: (a) third, (b) fourth, and (c) fifth layer.

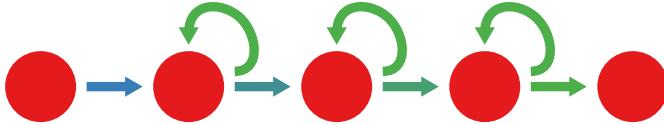
¹⁷ Andreas Veit, Michael Wilber, and Serge Belongie. Residual networks are exponential ensembles of relatively shallow networks. *arXiv preprint arXiv:1605.06431*, 1(2):3, 2016

¹⁸ Trang Pham, Truyen Tran, Dinh Phung, and Svetha Venkatesh. Column networks for collective classification. In *Proceedings of the AAAI conference on artificial intelligence*, volume 31, 2017

¹⁹ Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. In *EMNLP*, pages 1724–1734, 2014.

²⁰ Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. In *International conference on machine learning*, pages 5453–5462. PMLR, 2018b

²¹ Jiaxuan You, Zhitao Ying, and Jure Leskovec. Design space for graph neural networks. *Advances in Neural Information Processing Systems*, 33:17009–17021, 2020



passed through many “aggregate” steps – in green –, each of them changing it with the current node’s attributes.

It’s like a game of telephone that has ran for too long. By the end, the message passed to the final recipient node is completely green: the intermediate nodes have erased all the blue information contained in the original message. Central hubs with high betweenness are the culprit and can “squash” virtually all messages with their own attributes, resulting in less expressive embeddings at the end²².

44.3 Convolution

It is now time to go into the dreaded math I foreshadowed at the beginning of the chapter. There are many reasons why this is a good idea. One of them is to realize that most of what I said so far is nothing more than a bunch of matrix multiplications. This gives you an idea why there has been an explosion of popularity with neural network approaches: it’s not because these are fancy algorithms. Matrix multiplication ain’t fancy but it works, and it has the convenient property that you can do it blazingly fast on a GPU. It is aligning the neural network approaches with how the hardware works that led to the gains.

Message-Passing as Matrix Multiplications

Ok, so how is a MPGNN really a matrix multiplication? Remember the key ingredient: we have a $|V| \times d$ matrix H^0 with d dimensional node features. Then we want each node v to look at the features of all its neighbors. What matrix can tell you the neighbors of a node? The good ol’ adjacency matrix A . So the most barebone graph neural network possible can be expressed simply as:

$$H^l = AH^{l-1},$$

that is to say: to get the values for layer l , you take those from layer $l - 1$ and you aggregate them by multiplying H^{l-1} by the adjacency matrix A . You know from Section 5.2 that the result of multiplying a $|V| \times d$ matrix (H^0) to a $|V| \times |V|$ matrix (A) is going to be another $|V| \times d$ matrix – so H^1 will be just another node attribute matrix. Figure 44.7 shows an abstract representation of how this happens.

Figure 44.6: An example of oversquashing. Horizontal arrows represent the passed message. Loopy arrows represent the aggregate step of the receiving node. The color of the arrow represent what’s being passed.

²² Jake Topping, Francesco Di Giovanni, Benjamin Paul Chamberlain, Xiaowen Dong, and Michael M Bronstein. Understanding over-squashing and bottlenecks on graphs via curvature. *arXiv preprint arXiv:2111.14522*, 2021

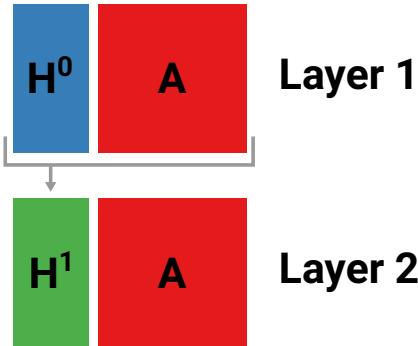


Figure 44.7: The basic architecture of a MPGNN in matrix form. The blue rectangle represents the matrix for node features, red rectangle for the adjacencies, and green rectangle for the hidden features.

Note how in the figure I use different colors for H^0 and H^1 , stressing that the former is data while the latter is a representation of what we've learned in the first layer. This process – and equation – describes the aggregation step of the MPGNN I described at the beginning of Section 44.1 – except that here the aggregation function is the sum and not the average.

There are some issues with this formulation which you can solve by implementing the update step of the MPGNN. If $H^k = H^{l-1}A$, then each node v completely forgets its $l - 1$ features – which is a problem especially for the first layer, since in that case the H^0 features we're forgetting are literally node v 's actual real features. To fix this issue we add self loops in the update step, assuming that each node v is a neighbor of itself. This is also a matrix operation! It is equivalent to adding the identity matrix I to A . So now we have:

$$H^l = (I + A)H^{l-1}.$$

Using a different perspective, this is equivalent to sum to the result H^{l-1} to $H^{l-1}A$. This would lead to $H^l = H^{l-1} + AH^{l-1}$, and we can group the H^{l-1} terms, since $H^{l-1}I = H^{l-1}$ – with I being the identity matrix.

Great, but we're not done reproducing the basic MPGNN I showed you at the beginning of the chapter. The first thing we need to solve is that, in my original MPGNN, the aggregation function was not the sum, but the average. Why might we prefer the average over the sum? Well, if you have a lot of layers, constantly summing features will lead them to eventually blow up in scale. All values will diverge to infinity. You might want to keep each H^l within the same scale. You can actually see this problem happening in the example from Figure 44.3, with the resulting embedding being much larger than the original one.

You might think to solve this problem by using the adjacency matrix, normalized by the degree – yet another thing you can do with matrix multiplications: $D^{-1}A$. However, sometimes a more

sophisticated approach can help you. For instance, the symmetric normalization $D^{-1/2}AD^{-1/2}$ still produces a stochastic matrix, but here you're normalizing the values coming from your neighbors, taking into account how many neighbors they themselves have²³. So:

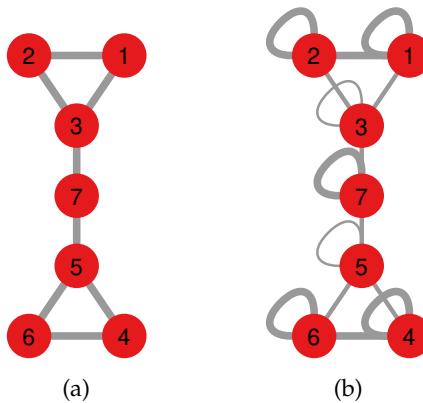
$$H^l = \hat{D}^{-1/2}(I + A)\hat{D}^{-1/2}H^{l-1},$$

with $\hat{D} = D + I$, i.e the degree matrix which takes into account the fact that you added self loops to A . Note that you don't have to always normalize: in fact, normalizing by the degree reduces the number of structures you can distinguish (Section 44.2) because now all of a sudden you can't distinguish nodes with different degrees any more.

We're so close to reproduce the initial MPGNN, we are only missing one step: the final activation function in the update step. You might want to apply any activation function σ to the result – see Section 4.2 for some options. The reason is that matrix multiplications can only allow you to learn linear functions – this is linear algebra, after all! However, more often than not, you might want to learn nonlinear functions. An activation function such as ReLU or the sigmoid can allow you to inject nonlinearity in the system. The result is:

$$H^l = \sigma\left(\hat{D}^{-1/2}(I + A)\hat{D}^{-1/2}H^{l-1}\right).$$

This is a big pile of linear algebra already, but we can still demystify it a bit with a graphical representation. That formula is equivalent to the operation I depict in Figure 44.8.



²³ Thomas Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017

Now we're getting somewhere, but this somewhere still isn't very far. Up until this moment we have reproduced the MPGNN framework. It is now time to really go deep.

Figure 44.8: (a) The original network. (b) The network after the $\hat{D}^{-1/2}(I + A)\hat{D}^{-1/2}$ operation. The thickness of an edge is proportional to the weight it gets as a result of the transformation.

From Message-Passing to Convolution

The key ingredient to really start with deep learning on graphs looks innocuous. We just add another matrix multiplication to the framework. We call this mysterious family of matrices W , and we have a different W^l matrix for each layer l . So:

$$H^l = \sigma \left(\hat{D}^{-1/2}(I + A)\hat{D}^{-1/2}H^{l-1}W^l \right).$$

This is starting to become a handful, isn't it? Yet, it's still matrix multiplications all the way down. And we know what each piece means, except W and what its role is, so let's take it step by step. Fundamentally, W does three things:

1. By multiplying it to the other matrices it weighs the results – that's why it's called W , W for Weights.
2. It changes the dimensions of the embeddings, as we're free to decide one of its dimensions.
3. It allows to the neural network to learn in a supervised task.

In other words, W is the primary ingredient of the secret sauce that allows to move from MPGNN to actual Graph Convolutional Networks (GCNs). The first point is the least interesting: multiplying W to any matrix will change the matrix, over- or under-weighting parts of it – unless W is the identity matrix.

How do we achieve the second point? Well, W is a $d \times d'$ matrix. We're multiplying it to a $|V| \times d$ matrix, so you know that you're going to end up with a $|V| \times d'$ matrix in the end. Unless $d = d'$, you will end up with a set of node features that is different than the original set. Stripping away complexity that is not necessary to get to the point, Figure 44.9 shows you how this dimensionality change happens.

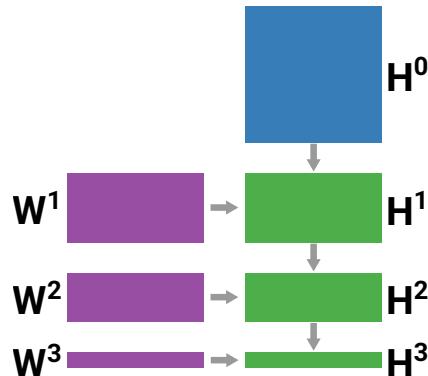


Figure 44.9: The dimensionality reduction that one can apply to H^l by multiplying H^{l-1} to W^l . The length of the sides of the rectangles are proportional to the dimensions of their matrices.

Each W^l must share one dimension with H^l and will determine the shape of H^{l+1} . In most cases, you want $d' < d$, for instance because the original set was sparse and you want it to be denser. Often, for the last layer, you want $d' = 1$, so that you end up with a single number per node, and use it to classify the nodes.

If these two things – weighting and dimensionality reduction – were all that W does, you could still technically remain within the bounds of MPGNNS. If we only have fixed weights, it is as if our graph was weighted from the beginning. And as long as you always keep $d = d'$, we still have our familiar MPGNN. But the burning question you might have, and the one which will mark our permanent departure to GCNs is: what's actually inside W ? We know what H , A , I , and D contain. But W seems mysterious.

To understand W we need to unpack its third role in the GCN: to allow the neural network to learn. In this scenario, we have some sort of ground truth we want to predict, so we're performing a supervised learning task. Let's assume that we have this ground truth in matrix T – for simplicity assuming it is merely a $|V| \times 1$ vector. Ideally, we want the final embeddings H^l to also be $|V| \times 1$, and to be identical to T – plus/minus a constant multiplicative factor. If that's the case, we can perfectly predict the T values with H^l .

A MPGNN cannot do this because it doesn't have W and so it can't learn. That is to say, with given starting node features H^0 , you're going to get the same H^l as output. It might be good to predict T , but switch T with T' – a different ground truth, a different set of new node features we want to predict – and the MPGNN won't be able to adapt to this new task. But a GCN can, because it has W . At each layer l , a GCN can calculate how different H^l is to any arbitrary T . This difference will be estimated using any loss function (Section 4.3) we think is appropriate.

Then the GCN can perform one trick, which is so powerful we gave it a cool name: backpropagation^{24,25}. The GCN can backpropagate the errors. Basically this means that, after each layer l has computed the value of its loss function, it can call up layer $l - 1$ and tell it: "you dummy! Look what a bad job you did! Do it better so that my loss function is lower!" And layer $l - 1$, very embarrassed, will get to work to minimize the loss function. How can it do it? By changing the only thing it can change: W^{l-1} .

Backpropagation is a bit more complicated than this: it is a clever way to figure out how the loss function is changing given the changes in the parameters – in technical terms you call the landscape of such changes "gradients" and you want to "descend" them, i.e. to find the minimum value of the loss function. One way to do it is by stochastic gradient descent²⁶. However, you only care about these

²⁴ Seppo Linnainmaa. *The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors*. PhD thesis, Master's Thesis (in Finnish), Univ. Helsinki, 1970

²⁵ David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986

²⁶ Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951

details if you’re doing machine learning, so we skip some nitty-gritty here. This is more or less the general picture you need as a network scientist.

There might still be one last thing worrying you. Backpropagation is cool, but it only tells you how to update W to go from a high to a low loss. It doesn’t tell you how to initialize W . To initialize W you need to apply the strategy I depict in Figure 44.10.

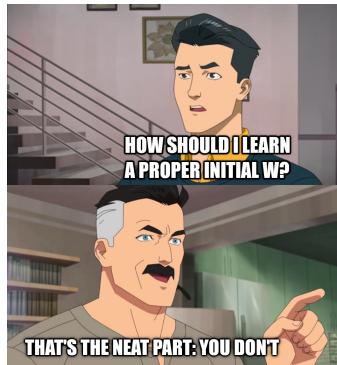


Figure 44.10: The initialization algorithm for W in a graph convolutional network.

That is to say: you can start with a random W and backpropagation will lead any random W to predict T with minimal loss²⁷. Sure, some people will tout the idea of pre-training, which is to do a training phase to get a great W before you actually run your GCN. But that’s not necessarily going to help.

To give you an example of the flexibility of GCNs against MPGNNs consider Figure 44.11. Here we assume the nodes are papers, connections are citations, and nodes have features connecting them to the words in their abstracts. Our T here might be a binary matrix, telling us whether a paper is about a specific subfield. This is a fairly classical test case of graph neural networks. And MPGNNs can do well here, as the figure shows. Since papers citing each other also have fairly similar abstracts, the messages will be able to detect this synergy between the network’s structure and the node embeddings.

Note how the GCN can produce 1D embeddings even if we started with 2D ones, which here works fine because we only have two classes to predict, so we can place the node in a spectrum. One difference is how much more confident the GCN is with respect to the MPGNN: the scores are much closer to zero and to one, which means the GCN is sure about the label assignment. Also, the GCN doesn’t distinguish between nodes that are structurally equivalent, such as nodes 4 and 6, even if they started with different embeddings.

But these are mere details: the MPGNN here worked equally as well as the GCN. But suppose that the same network – same nodes, same citations, same abstracts – now wants to predict a different T' :

²⁷ Petar Velickovic, William Fedus, William L Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. Deep graph infomax. *ICLR (Poster)*, 2(3):4, 2019

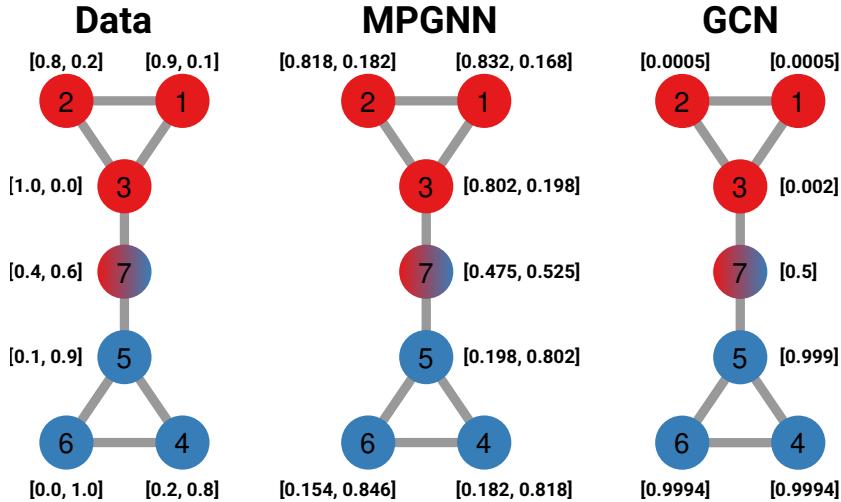
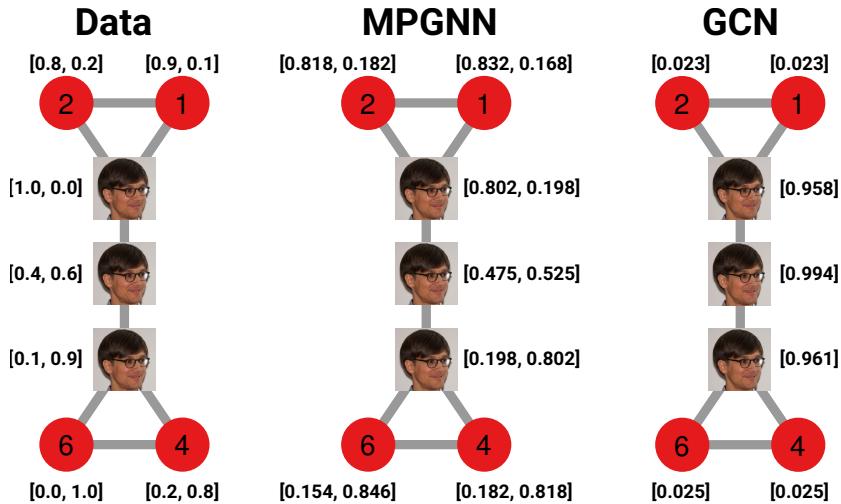


Figure 44.11: (Left) The initial data: nodes are papers connected by undirected edges if the papers cite one another. Each node starts with two attributes. The node's color tells us the field of the paper it represents, with node 7 belonging to both the red and the blue class. (Right) The numbers next to the node represent the node's embeddings obtained from the last layer of the MPGNN and GCN.

a binary vector telling us whether the paper was authored by Jure Leskovec. A MPGNN is condemned to run the same process, and would probably perform poorly – as Jure is an eclectic researcher. A GCN, instead, can make its W into an efficient Leskovec-detector – a “lesk2vec” if you may²⁸ –, and spot him in no time.



²⁸ I'm begrudgingly crediting Sune Lehmann for the best joke of the chapter.

Figure 44.12: (Left) The same network as Figure 44.11, however this time we want to predict the node's picture rather than its color. (Right) The embeddings produced by the MPGNN and the GCN.

Figure 44.12 shows you that there is no connection between the MPGNN embeddings and whether the node is a Leskovec node, while the GCN only gives high values to the Leskovec node and low values to all other nodes – a radically different embedding even if the starting point was exactly the same. Backpropagation lead it to correctly modify W to adapt to this new T' .

As a final note, so far I talked about deep graph neural networks

for supervised tasks, where we have a ground truth we want to predict. This is not the only option. There are unsupervised approaches that can be driven by a maximization of mutual information between local and global graph properties²⁹.

44.4 Spectral Convolution

Before moving on to even more sophisticated methods, I want you to consider a slight shift in perspective. If you look again at the most recent pile of matrix multiplications that define a GCN, you'll notice that it is only dependent on A , the adjacency matrix:

$$H^l = \sigma \left(\hat{D}^{-1/2} (I + A) \hat{D}^{-1/2} H^{l-1} W^l \right).$$

Even if H^{l-1} aggregates information from potentially many steps away, to determine the new embedding of node v we're still operating exclusively on the information that reached the direct neighbors of v . This is what we call a "spatial" approach³⁰. This is powerful and has many advantages. One of them is that the information to determine the status of a node is essentially local: you don't need to know the entire topology of the graph to learn a specific node's embedding. There are many popular GCN methods that are spatial^{31,32}.

This, by the way, solves the transductivity problems of shallow learning we saw in Section 43.5. If you get a new node in your network, you can determine its new embeddings by only running the local part of the graph neural network.

However, spatial approaches are not the only game in town, just like they weren't when we dealt with shallow learning. In shallow learning, random walk-based methods are also spatial, and I showed you there are alternatives, namely by using a spectral approach. In GCNs we can have a spectral approach as well^{33,34}. If in the spatial approach we learn the embedding of v only by looking at v 's neighbors, in the spectral approach v 's updated embedding depends on the entire graph as a whole.

This is because we see the embedding as a signal that is processed by the entire graph. To make sense of this sentence you need to understand a few things about signal processing, the Fourier transform, and its discrete version on graphs. This is a huge topic, very complex, so I'm going to give you an extremely simplified and incomplete version of what's going on, and you'd do better reading more advanced texts on the subject³⁵.

Basically, the Fourier transform is a way to decompose any periodic signal into a weighted sum of frequencies that make it up. If, for any moment in time, you sum the value of each frequency with

²⁹ Petar Veličković, William Fedus, William L Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. Deep graph infomax. *arXiv preprint arXiv:1809.10341*, 2018

³⁰ Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2008

³¹ James Atwood and Don Towsley. Diffusion-convolutional neural networks. In *NIPS*, pages 1993–2001, 2016

³² Mathias Niepert, Mathias Ahmed, and Konstantin Kutzkov. Learning convolutional neural networks for graphs. In *ICML*, pages 2014–2023, 2016

³³ Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. In *ICLR*, 2014

³⁴ Renjie Liao, Zhizhen Zhao, Raquel Urtasun, and Richard S Zemel. Lanczosnet: Multi-scale deep graph convolutional networks. *arXiv preprint arXiv:1901.01484*, 2019b

³⁵ David Shuman, Sunil Narang, Pascal Frossard, Antonio Ortega, and Pierre Vandergheynst. The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *IEEE Signal Processing Magazine*, 3(30):83–98, 2013

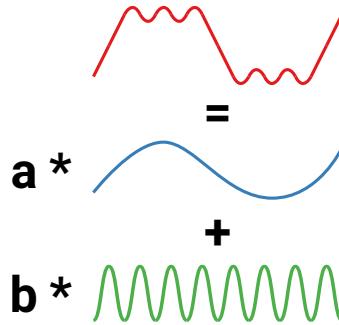


Figure 44.13: The Fourier transform. The original signal in red can be thought as the weighted sum of frequencies (in blue and green).

the weight you learned by the Fourier transform, you'll get the exact value of the signal. Figure 44.13 shows you how to think about this visually.

In the graph Fourier transform, you want to do the same, but your signal is H^{l-1} . How to do it? Well, one key part of the regular Fourier transform is to calculate the difference between the value of the signal at a point and the values of the signal in the neighboring points. This is done with the Laplace operator and you should have alarm bells ringing in your head right now. The Laplacian does exactly the same thing on a graph: the -1 entry for each edge gives you exactly the difference of a node with its neighbor.

The way to use L in the graph Fourier transform is to exploit the following equivalence: $L = \Phi \Lambda \Phi^{-1}$ – this is the eigendecomposition we saw in Section 5.6. What are Φ and Λ ? Λ is L 's eigenvalue diagonal matrix:

$$\Lambda = \begin{pmatrix} \lambda_0 & \dots & 0 \\ 0 & \ddots & 0 \\ 0 & \dots & \lambda_n \end{pmatrix}$$

where $0 = \lambda_0 < \lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_{|V|-1}$ are the eigenvalues of L sorted in increasing order (as usual, we assume G is connected). Φ is the matrix of L 's eigenvectors, in the same order as Λ . We do this because in this way we have established that the “frequencies” we were talking about in the normal Fourier case are the eigenvalues of L . As a consequence, we can reconstruct any signal on G as the weighted sum of the eigenvalues of L , as Figure 44.14 depicts.

This means that you can use Φ as a convolution operator instead of A , and Φ will give you information about how the entire graph at once processes a single node v 's embeddings. The obvious downside here is that calculating Φ is computationally expensive. So it is no surprise that people have worked on ways to perform this update locally³⁶. Then there are ways to render this approach more sophisticated and expressive^{37,38}.

³⁶ Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *NIPS*, pages 3844–3852, 2016

³⁷ Ron Levie, Federico Monti, Xavier Bresson, and Michael M Bronstein. Cayleynets: Graph convolutional neural networks with complex rational spectral filters. *IEEE Transactions on Signal Processing*, 67(1):97–109, 2018

³⁸ Filippo Maria Bianchi, Daniele Grattarola, Lorenzo Livi, and Cesare Alippi. Graph neural networks with convolutional arma filters. *IEEE transactions on pattern analysis and machine intelligence*, 44(7):3496–3507, 2021

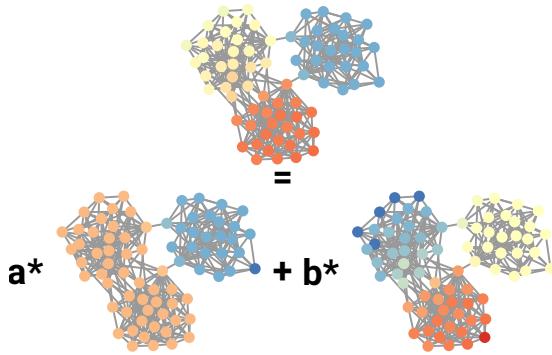


Figure 44.14: The Fourier transform. The original signal (top) in red can be thought as the weighted sum of eigenvectors of the Laplacian (bottom).

44.5 Summary

1. Deep learning on graphs uses the message-passing graph neural network (MPGNN) framework: each node aggregates messages from its neighbors' features and uses them to update its own features. One can repeat this aggregation-update operation many times, each of which being a layer in the MPGNN.
2. Limitations of this approach come from the fact that there are structures MPGNNs aren't able to distinguish, that piling up too many layers will usually result in all nodes having the same embeddings, and that messages from peripheral nodes are usually lost.
3. A MPGNN can be described as a series of matrix operations in linear algebra, for instance: $H^l = \sigma(\hat{D}^{-1/2}(I + A)\hat{D}^{-1/2}H^{l-1})$, which means to add self loops, normalize edge weights, then pass messages through edges, and update with a non-linear σ function such as ReLU.
4. In graph convolutional networks (GCNs), you add another matrix family W^l to the multiplication. W^l can weight how much a node's embedding depends on each of its neighbors, and it can reduce the dimensions of your embeddings.
5. W^l can also allow you to predict specific node features you're interested in. At the beginning, W^l is random, but training it via backpropagation can lead to find the W^l that minimizes your loss with any prediction target you might have.
6. One can go from a spatial approach with A to a spectral approach with Φ – the matrix of eigenvectors of the Laplacian – performing a graph Fourier transform, processing each embedding via the entire graph rather than only the immediate neighborhood.

44.6 Exercises

1. Implement the MPGNN described in Section 44.1. You need to define an “aggregate” function which takes a list of nodes and their embeddings and returns the element-wise mean of those embeddings. You need to define an “update” function which takes two vectors, sums them, and return their softmax. Finally, you need a message-passing function which loops over all nodes of the network, applies aggregate to its neighbors, and applies update with the result of the aggregation and the node’s embedding. Run a single layer of it on the network at <http://www.networkatlas.eu/exercises/44/1/network.txt>, with node features at <http://www.networkatlas.eu/exercises/44/1/features.txt>. Do you get the same results as Figure 44.1?
2. Run the MPGNN you designed in the previous exercise. Make a scatter plot of the node embeddings using the two dimensions as x and y coordinates at the first, fifth, tenth, and twentieth layer. What do you observe?
3. Implement a MPGNN as a series of matrix operations, implementing $H^l = \sigma(\hat{D}^{-1/2}(I + A)\hat{D}^{-1/2}H^{l-1})$, with σ being softmax. Apply it to the network at <http://www.networkatlas.eu/exercises/44/3/network.txt>, with node features at <http://www.networkatlas.eu/exercises/44/3/features.txt>. Compare its running time with the MPGNN you implemented in the first exercise, running each for 20 layers and making several runs noting down the average running time.

45

Deep Graph Learning Models

45.1 Attention

In standard machine learning, “attention” is a strategy that allows you to have embeddings that adapt to the situation¹. The classical case is natural language processing. Each word must have an embedding, but problems arise when you have words with multiple meanings. You have an example in this book part, with the awkwardness of the word “network”, which we use in this book to refer to an object – for instance a *social* network – that is rather different from a *neural* network. A world where the same word “network” can have more than one embedding would be wonderful. Attention allows you to do that.

In practice, with attention, the presence of a word in the same sentence can change the embedding of the words that follow. That makes sense: if I write “social” before “network” you immediately know what I’m talking about, and you know it is different than the network I’d be talking about had I written “neural” instead. You can see this happening graphically in Figure 45.1: attention is nothing more than taking a generic concept pointing to the general idea of “network” and multiplying a different vector depending on which words preceded “network”. The result is to point in a slightly different direction, which is the more concrete embedding of which meaning of the word we are actually referring to.

Mathematically this means that the H^l embeddings we use to calculate the H^{l+1} ones are not really really the H^l embeddings: they are $\alpha^l(H^l)$, i.e. the result of applying the attention function α to them at layer l .

The same principle that works in NLP works in networks too. When you receive messages from your neighbors, you might want to pay attention more to some over others. For instance, in a paper citation network, you might have hubs that are cited by everyone, or interdisciplinary papers whose citations are ambiguous – because

¹ Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014

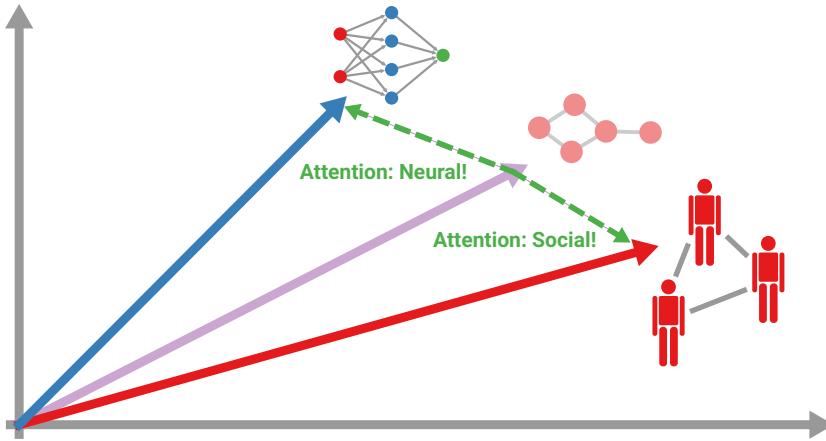


Figure 45.1: The mechanism of attention. Vectors represent the embeddings of words, pointing to their location in a 2D embedding space. The dashed green arrow translates the original generic embedding in purple to more specialized concepts, depending on which word preceded “network”.

they can come from multiple fields, and so might confuse you when trying to predict fields with citations. In those cases, you want to pay less attention to them.

In Graph Attention Networks^{2,3} (GATs) the way to implement this is by looking at the $D^{-1}A$ transformation that we have in MPGNNS. This is sort of an attention mechanism: it is the trivial one where each neighbor gets the same amount of attention. In this simplified example, each neighbor of node v gets exactly $1/k_v$ attention, with k_v being v 's degree. In this framework, v 's attention only depends on v 's characteristics, specifically its degree.

GATs unlock a new degree of freedom allowing v 's attention to depend also on the characteristics of the neighboring node u as well⁴. GCNs could sort of do the same, because the symmetric $\hat{D}^{-1/2}(I + A)\hat{D}^{-1/2}$ normalization also depends on the neighbor u 's degree, but this attention is always fixed to be $1 / (\sqrt{k_v} \sqrt{k_u})$. Instead, GATs try to learn – again with backpropagation – a different attention value for each of v 's neighbors. Since we introduced this learnable function as α , the GAT formula is:

$$H^l = \sigma(H^{l-1} \alpha^l W^l).$$

There is quite some freedom in how to define α for GATs. The key thing is that α takes as input both H^{l-1} and W^l , which is what allows it to be so flexible. So, in summary, GCNs are a special type of GATs: they are GATs that have a fixed non-learnable α . Therefore, GATs are a generalization of GCNs. In Figure 45.2 I call back to Figure 44.8. GCNs allow us to go from Figure 45.2(a) to Figure 45.2(b), preventing some issues that would be caused by using only A . GATs allow us to go from Figure 45.2(b) to Figure 45.2(c), realizing we're not forced to always use the same transformation, but we can actually earn the

² Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017

³ Shaked Brody, Uri Alon, and Eran Yahav. How attentive are graph attention networks? *arXiv preprint arXiv:2105.14491*, 2021

⁴ Shyam A Tailor, Felix L Opolka, Pietro Lio, and Nicholas D Lane. Adaptive filters and aggregator fusion for efficient graph convolutions. *arXiv preprint arXiv:2104.01481*, 2021

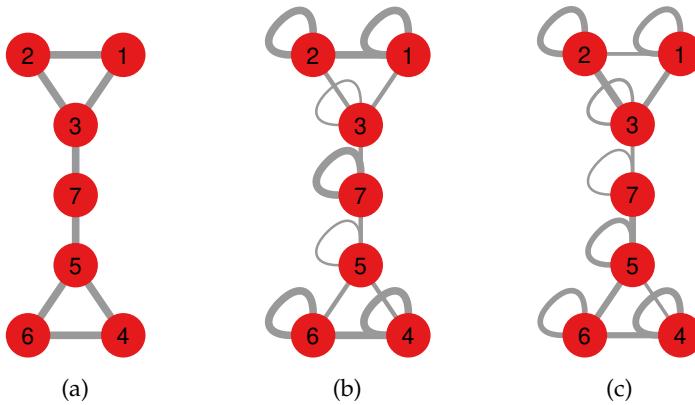


Figure 45.2: (a) The original network. (b) The network after the $\hat{D}^{-1/2}(I + A)\hat{D}^{-1/2}$ operation, leading to a GCN. The thickness of an edge is proportional to the weight it gets as a result of the transformation. (c) The network after an arbitrary transformation determined by a learned α function, leading to a GAT.

best possible one to minimize our loss.

Using attention, you realize why you might want to have multiple layers. Once you change the embedding of the word “network” because you saw the word “social” before, then every word following “network” might want to pay more (or less) attention to it.

Further refinements of this model are able to tackle heterogeneous networks with different node and edge types⁵, deal with data embedded in space and time⁶, with signed networks⁷ and more.

45.2 Transformers

Ever since a seminal paper came out⁸, transformers became all the rage in neural networks, even though I’ve been playing with them since the 80s⁹. In fact, the T in the popular Chat-GPT algorithm actually stands for Transformer. So, of course the graph neural network literature took a good look at them.

Conceptually, the important part of a transformer is not all that difficult. It is merely a parallel attention mechanism. Instead of having a single α^l function per layer, you have many: $\alpha_1^l, \alpha_2^l, \dots, \alpha_k^l$. These are all trained independently, so they are free to learn different things. Then, the representation you get can be any combination of these newly learned representations – more often than not it is a linear combination and/or you stitch them together by concatenating the various matrices. If you want to learn the lingo, each of the α_k^l independent attention is called an attention “head”.

One can apply this strategy exactly as written to GATs, leading to graph transformers. In fact, the paper proposing GATs for the first time – which I described in the previous section – already defines graph transformers. Figure 45.3 shows you what happens in each layer l of a graph transformer. The various attention heads are trained independently, they each operate on the H^{l-1} and W^l

⁵ Xiao Wang, Houye Ji, Chuan Shi, Bai Wang, Yanfang Ye, Peng Cui, and Philip S Yu. Heterogeneous graph attention network. In *The world wide web conference*, pages 2022–2032, 2019

⁶ Chenhan Zhang, JQ James, and Yi Liu. Spatial-temporal graph attention networks: A deep learning approach for traffic forecasting. *IEEE Access*, 7: 166246–166256, 2019

⁷ Junjie Huang, Huawei Shen, Liang Hou, and Xueqi Cheng. Signed graph attention networks. In *ICANN 2019: Workshop and Special Sessions*, pages 566–577. Springer, 2019

⁸ Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017

⁹ I’m now told that puppet robots don’t count in machine learning, although I’m pretty sure Megatron is too evil not to be the product of some sort of rogue AI.

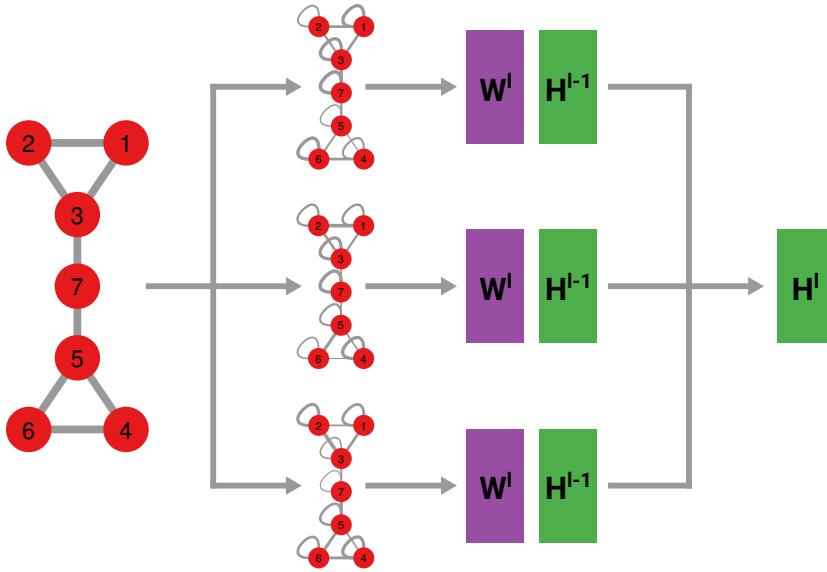


Figure 45.3: What happens in the layer l of a graph transformer.

matrices, then the result is combined to produce H^l .

This can be seen as a further generalization away from GCNs. In GATs we decided we wanted to learn the attention weights with a function, rather than having fixed ones like in GCNs. In graph transformers we ask: why limiting ourselves to learning a single attention function? And so we learn an arbitrary number of them.

Of course there are different ways to make more sophisticated graph transformers^{10,11}. And, following Section 44.4, one can ditch the spatial approach to GATs and graph transformers to embrace the spectral one¹².

45.3 Deep Generative Graph Models

One cool thing you can do with graph neural networks is to learn how to generate a graph that looks like a real one. This is arguably something we already discussed in Part V, so this section could fit in there – and I gave you a teaser of some deep generative approaches in Section 18.4. However there are a couple of reasons why this section is here. First, it relies on advanced graph neural network concepts that we could not introduce before Part V. More importantly, the mechanism here is rather different. Rather than fixing a handful of key parameter and let a fixed algorithm – such as preferential attachment – to do the rest, here we actually learn from many graphs how the algorithm should wire the network. Exponential random graphs and the configuration model get closer to this approach, but they too only learn from a single graph.

¹⁰ Seongjun Yun, Minbyul Jeong, Rae-hyun Kim, Jaewoo Kang, and Hyunwoo J Kim. Graph transformer networks. *Advances in neural information processing systems*, 32, 2019

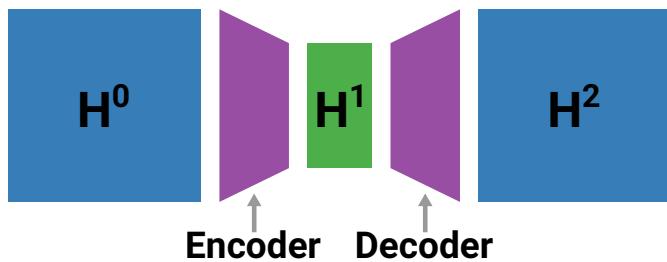
¹¹ Erxue Min, Runfa Chen, Yatao Bian, Tingyang Xu, Kangfei Zhao, Wenbing Huang, Peilin Zhao, Junzhou Huang, Sophia Ananiadou, and Yu Rong. Transformer for graphs: An overview from architecture perspective. *arXiv preprint arXiv:2202.08455*, 2022

¹² Devin Kreuzer, Dominique Beaini, Will Hamilton, Vincent Létourneau, and Prudencio Tossou. Rethinking graph transformers with spectral attention. *Advances in Neural Information Processing Systems*, 34:21618–21629, 2021

Variational Autoencoders

To understand variational autoencoders¹³ we need to start by breaking down their name into its component parts: we start by understanding what an “encoder” is, what it means for it to be “auto”, and what “variational” means.

The *encoder* part is the easiest. An encoder is a function that takes an input and produces a code – a more succinct representation of the same data that went in. You’ve already seen a bunch of encoders up until now. Every time you produce a node embedding you’re encoding it into a different representation. An *autoencoder* aims to reconstruct the original data by passing it through an encoder and then recovering it. This means we need a decoder that reconstructs the original data. Figure 45.4 shows you the general structure of an autoencoder.



Both the encoder and the decoder can be learned, so that the loss in reconstructing the original data is minimal. You might think it’s pointless to try and reconstruct the original data – after all you have it, so why bother regenerating it? – but the value is that now your encoder and decoder have learned the key features of the data. At this point, you don’t need the original data any more and you can generate as much data as you want.

Why would we need a *variational* part? That’s what introduces randomness. If you always reconstruct the input features H^0 , then you’re going to always get the same embeddings from the autoencoder, as you can see in Figure 45.5 – where I unpack the values inside H^1 . In practice, the objective of an autoencoder is to learn directly the embeddings and use them to reconstruct the original adjacency matrix A .

However, what you can do instead is to learn not the embeddings themselves, but the *distribution* of their values. For instance, you can assume the embeddings distribute normally, and then you use two neural networks to learn the mean and standard deviations of those distributions. This means that you treat the embeddings as random variables, which can be more succinctly described by the parameters of their distribution. Once you have learned such

¹³ Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013

Figure 45.4: The structure of an autoencoder: (left to right) the input data (blue) is encoded (purple) in an embedding (green) then decoded (purple) to a second layer embedding (blue) which is as similar as possible to the original data.

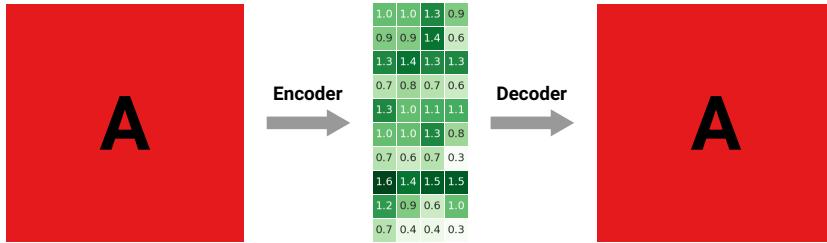


Figure 45.5: An autoencoder working on an adjacency matrix (in red). The encoder learns the H^1 embeddings (in green) and the decoder reconstructs the adjacency matrix.

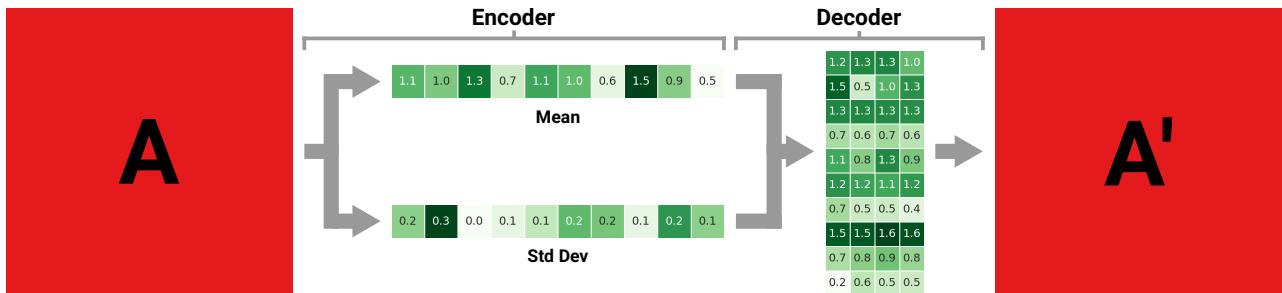


Figure 45.6: A variational autoencoder working on an adjacency matrix (in red). The encoder learns the mean and standard deviations behind the true H^1 embeddings (in green) and the decoder first samples a random embedding with those means and standard deviations and then uses it to reconstruct the adjacency matrix.

parameters – in our case the mean and standard deviation – you can generate random embeddings with those parameters. You’re going to get a new random embedding for each node which looks like the original one – because it has the same average and standard deviation – but it’s different enough that can lead you to reconstruct a different A .

You can see in Figure 45.6 how the variational part makes the encoder and decoder more complex. The encoder cannot simply reconstruct the embeddings, but needs to learn their distributions. The decoder has the task of sampling from these distributions and then use the sampled embeddings to reconstruct the data.

When it comes specifically to graphs, the main approaches are whether you want to learn the parameters of the distributions for each node separately^{14,15} – and so reconstruct node embeddings from their own means and standard deviations – or if you want to pool everything together and learn directly at the level of the entire graph¹⁶ – reconstructing the entire adjacency matrix all at once.

Depending on how you build the embeddings, the autoencoder can learn different things. For example, you can try to use the autoencoder to reconstruct the information present in random walks¹⁷, and if you go high-order you can learn the first and second order relationships between nodes at the same time¹⁸.

¹⁴ Thomas N Kipf and Max Welling. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*, 2016

¹⁵ Aditya Grover, Aaron Zweig, and Stefano Ermon. Graphite: Iterative generative modeling of graphs. In *ICML*, pages 2434–2444, 2019

¹⁶ Martin Simonovsky and Nikos Komodakis. Graphvae: Towards generation of small graphs using variational autoencoders. In *ICANN*, pages 412–422. Springer, 2018

¹⁷ Shaosheng Cao, Wei Lu, and Qiongkai Xu. Deep neural networks for learning graph representations. In *AAAI*, 2016

¹⁸ Daixin Wang, Peng Cui, and Wenwu Zhu. Structural deep network embedding. In *SIGKDD*, pages 1225–1234, 2016a

Generative Adversarial Networks

The idea behind Generative Adversarial Networks (GANs) is relatively simple^{19,20}: we employ a two step strategy. First, we create a generator – of any type, with any algorithm – that can generate some realistic-looking data. Then we pass it to a classifier, whose task is to learn how to distinguish real data from randomly generated one. The two are the adversaries: the generator is trying to fool the classifier, and the classifier is trying to spot the generator’s outputs.

Graph versions of GANs are what you expect them to be. Both the generator and the classifier need to be some sort of neural network architecture that can generate adjacency matrices and use them as input for the classification^{21,22}.

Autoregressive Models

Autoregressive models can be used in combination with VAEs and GANs to augment them – because they can be used as decoder for a VAE or as the generator for a GAN. But they can also be their own thing, just generating the networks without a subsequent step in the VAE or GAN architecture.

Their main objective is to relax a key assumption VAEs and GANs make: that edges are generated independently. We know that’s not true: high clustering coefficients mean that the friend of my friend is more likely to be my friend. This means that edge generation is not independent. Autoregressive models solve the problem by reducing the generation of the adjacency matrix as the generation of a linear sequence of zeroes and ones. The next entry on the sequence is dependent on the sequence we have generated so far – so here’s the dependency.

I already explained how one of these methods work – without using too much deep learning lingo – in Section 18.4, where I showed you GraphRNN²³. In this case, we’re still operating on an edge-by-edge basis – even if we use all the previously generated edges to make a new one. An alternative approach looks at general descriptive statistics of a graph – such as degree distribution, the spectrum of the Laplacian, and more – and use those as points of comparison to generate the new graph²⁴.

The advantage of this approach is that it can be generic – working on properties that any graph has – but it can also be specific. If you know what these graphs represent and what peculiar statistics they might have – for instance, some physical and chemical properties that must be respected when generating graphs representing molecules²⁵ –, you can integrate that information in the generation process.

¹⁹ Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014

²⁰ Antonia Creswell, Tom White, Vincent Dumoulin, Kai Arulkumaran, Biswa Sengupta, and Anil A Bharath. Generative adversarial networks: An overview. *IEEE signal processing magazine*, 35(1): 53–65, 2018

²¹ Nicola De Cao and Thomas Kipf. Molgan: An implicit generative model for small molecular graphs. *arXiv preprint arXiv:1805.11973*, 2018

²² Aleksandar Bojchevski, Oleksandr Shchur, Daniel Zügner, and Stephan Günnemann. Netgan: Generating graphs via random walks. In *International Conference on Machine Learning*, pages 609–618, 2018

²³ Jiaxuan You, Rex Ying, Xiang Ren, William Hamilton, and Jure Leskovec. Graphrnn: Generating realistic graphs with deep auto-regressive models. In *International Conference on Machine Learning*, pages 5694–5703, 2018

²⁴ Renjie Liao, Yujia Li, Yang Song, Shenlong Wang, Will Hamilton, David K Duvenaud, Raquel Urtasun, and Richard Zemel. Efficient graph generation with graph recurrent attention networks. *Advances in neural information processing systems*, 32, 2019a

²⁵ Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Junction tree variational autoencoder for molecular graph generation. In *International conference on machine learning*, pages 2323–2332. PMLR, 2018

45.4 Alternative Message-Passing Techniques

So far I've only discussed the same type of message-passing technique. This is the case where the messages from all nodes are aggregated in discrete steps and summed up in a single-valued summary. For instance, consider Figure 45.7. Here node v aggregates the messages from the neighbors by averaging them with its own embedding. There are a couple of ways to challenge this approach.

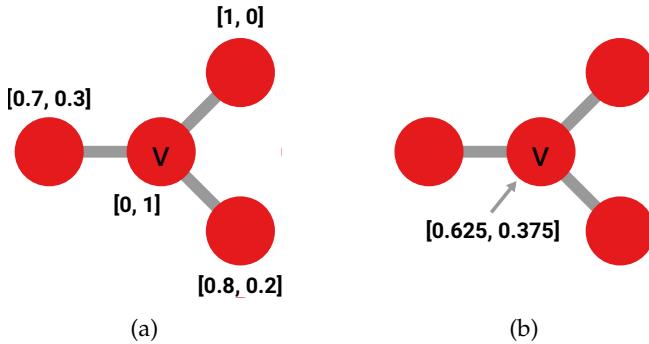


Figure 45.7: (a) A network with 2D node embeddings next to their corresponding nodes. (b) The resulting embedding for node v , using average as the aggregation function.

The first is by using set aggregators. In practice, this replaces the average function we just used. You take the three embeddings as a set and you use a function that, given a set, produces an output. One example of such function is the hash we used for the Weisfeiler-Lehman isomorphism test. To give you an idea of how it might work, Figure 45.8(a) shows you a stupid set aggregator. Here we simply say that the result of the message-passing for node v is a list of two unordered sets. Unordered means that, in this case, $\{0, 1, 0.8, 0.7\}$ would be identical to $\{0.7, 1, 0, 0.8\}$ – and any possible permutation. We need this property because the message-passing still needs to be a permutation-invariant operation – it doesn't matter the order in which we look at the neighbors.

There are, of course, smarter way to define set aggregators^{26,27}. And, with Janossy pooling, you can actually use a function that is not

²⁶ Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep sets. *NeurIPS*, 30, 2017

²⁷ Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *CVPR*, pages 652–660, 2017

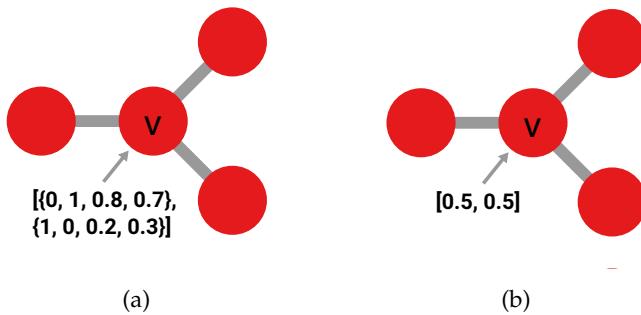


Figure 45.8: (a) The resulting embedding for node v from Figure 45.7(a), using a set aggregation function. (b) Same as (a), but by using a continuous aggregation function.

permutation invariant, and then average it over all permutations²⁸. Normally, there will be too many permutations to be able to do this exhaustively. So you can randomly sample some permutations, or deploy a canonical ordering of nodes – so there is only one order and this operation becomes permutation invariant.

The second alternative is to use normal aggregation functions, but refusing to run them in a discrete way. You can simulate a continuous flow of information and then discretize it at an arbitrary moment. This is a technique that we call continuous message passing^{29,30,31,32} and has relations with the more generic form of geometric deep learning – of which graph neural networks are a special, discrete, case.

To give an oversimplified example, in Figure 45.8(b) I only pass half of the messages from the neighbors, which results in node v preserving more of its own original embedding.

45.5 Practical Considerations

Computational Efficiency

In practically all cases, you don't want to implement deep graph neural network architectures by naively applying the operations I explained to you so far. Consider the message-passing model: a node can only send the same message to the MPGNN architecture. If you have a hub with thousands of connections, you're going to repeat the same operation thousands of times, to achieve the same result. In a GCN, since you're doing a single matrix multiplication, you don't have this problem, but you have another one: if your graph is huge, so are your matrices, and they might not fit in memory.

One thing you can do is to apply the sampling and mini-batching strategies^{33,34,35,36} (Section 4.4). In a graph you need specific ways of sampling. You know the intricacies of network sampling from Chapter 29. As a refresher, Figure 45.9(a) shows that sampling nodes randomly will most likely lead to disconnected samples where any message-passing strategy won't have anything to work with.

In the figure we were even lucky that the randomness lead to somewhat connected patches, but each patch cannot communicate with the others and we have wasted the time to sample and to compute the messages to essentially receive no information whatsoever. One minibatching strategy – which I show in Figure 45.9(b) – will get the messages for node v from a given number of nodes from v 's neighborhood, and then recursively sample the neighbors of the neighbors and so on, until we reached the desired depth. In the figure, we want to receive messages from up to two neighbors, up to a

²⁸ Ryan L Murphy, Balasubramaniam Srinivasan, Vinayak Rao, and Bruno Ribeiro. Janossy pooling: Learning deep permutation-invariant functions for variable-size inputs. *arXiv preprint arXiv:1811.01900*, 2018

²⁹ Ben Chamberlain, James Rowbottom, Maria I Gorinova, Michael Bronstein, Stefan Webb, and Emanuele Rossi. Grand: Graph neural diffusion. In *International Conference on Machine Learning*, pages 1407–1418. PMLR, 2021a

³⁰ Benjamin Chamberlain, James Rowbottom, Davide Eynard, Francesco Di Giovanni, Xiaowen Dong, and Michael Bronstein. Beltrami flow and neural diffusion on graphs. *Advances in Neural Information Processing Systems*, 34:1594–1609, 2021b

³¹ Cristian Bodnar, Francesco Di Giovanni, Benjamin Chamberlain, Pietro Lio, and Michael Bronstein. Neural shear diffusion: A topological perspective on heterophily and oversmoothing in gnns. *Advances in Neural Information Processing Systems*, 35:18527–18541, 2022

³² Fabrizio Frasca, Emanuele Rossi, Davide Eynard, Ben Chamberlain, Michael Bronstein, and Federico Monti. Sign: Scalable inception graph neural networks. *arXiv preprint arXiv:2004.11198*, 2020

³³ Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 974–983, 2018a

³⁴ Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *NIPS*, pages 1024–1034, 2017

³⁵ Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. Graphsaint: Graph sampling based inductive learning method. *arXiv preprint arXiv:1907.04931*, 2019

³⁶ Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 257–266, 2019

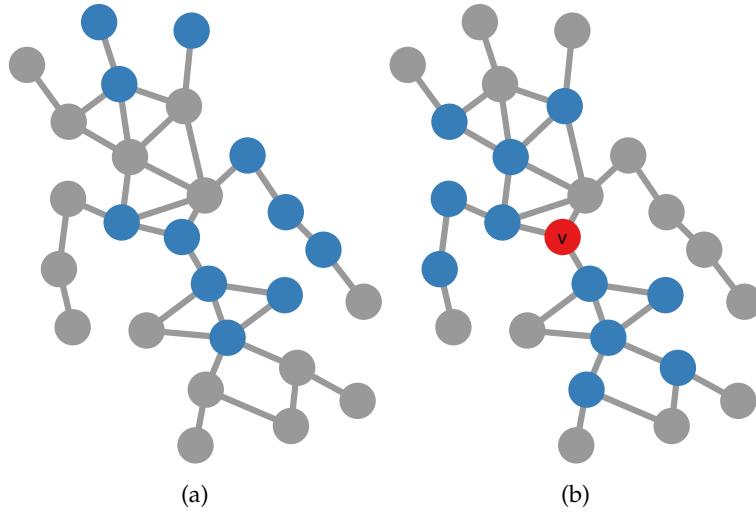


Figure 45.9: Two network mini-batching approaches. The gray nodes are not sampled and the blue nodes are sampled. (a) Random sampling, (b) Mini-batching around the red focus node.

distance of three steps.

Regularization

As I mentioned in Section 4.1, one of the greatest woes of machine learning is overfitting. Since you’re learning from a lot of data bottom-up without imposing a theory top-down, you might end up just learning the special quirks of the data you use for training. In neural networks there are a few techniques to avoid overfitting, which we normally call “regularization”. A classical strategy is called “dropout”: you take your matrix W and you randomly set to zero some entries or entire rows^{37,38}.

When working with a graph neural network, you can do a special variation of this: edge dropout³⁹. Since you also have A besides W , you can do dropout in A as well. This means to randomly remove a few edges between layers, which will make it harder for your framework to overfit to the original A .

Augmentation

Augmentation is something you’d do if your input graph is too sparse, which means information might have a hard time propagating through the layers. Curiously, in this case you’d do the exact opposite of the regularization strategy we just discussed: you’d try to add virtual edges – or virtual nodes⁴⁰. It might be difficult sometimes to add proper virtual edges to actually improve the situation rather than adding noise, but in some cases things can be easier. For instance, if you deal with a bipartite network, it’s kind of natural to add virtual edges between nodes of the same type, if they have a lot

³⁷ Li Wan, Matthew Zeiler, Sixin Zhang, Yann Le Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In *International conference on machine learning*, pages 1058–1066. PMLR, 2013.

³⁸ Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

³⁹ Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *European Semantic Web Conference*, pages 593–607. Springer, 2018.

⁴⁰ Johannes Gasteiger, Aleksandar Bojchevski, and Stephan Günnemann. Predict then propagate: Graph neural networks meet personalized pagerank. *arXiv preprint arXiv:1810.05997*, 2018.

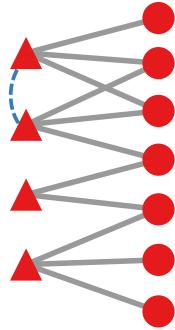


Figure 45.10: A bipartite network, augmented with the virtual edge in the blue dashed line.

of common neighbors – as Figure 45.10, where the nodes sharing two neighbors are connected with a virtual edge.

Applications

When deeply embedded in this book part, it is easy to lose oneself in the mountain of ever-evolving details that make up one of the most prolific publication grounds of the last half decade or so. One should not lose sight that GNNs are always means to an end. I already mentioned a few applications of shallow embedding learning in Section 43.4, so I think it is high time we refresh our minds with a few successful applications of GNNs out there.

Popular application of deep learning include in general those tasks that are difficult to approach from a purely theoretical standpoint. They are mostly scenarios when you'd normally make judgment calls like "I'll know it when I see it". The GNN can try to do it for you. For instance we have:

- Recommender systems: mostly this is link prediction in a bipartite network of users consuming some sort of product (movie, book, song, etc). This is normally done by learning which user embeddings are more likely to connect with which product embedding.
- Misinformation detection on social media: hoping to spot differences in the embeddings of real and fake news, or accounts that spread misinformation from accounts that do not^{41,42}.
- Network medicine: identifying organisms that could serve as new antibiotics by figuring out how they would interact with given pathogens⁴³.

And more.

⁴¹ Federico Monti, Fabrizio Frasca, Davide Eynard, Damon Mannion, and Michael M Bronstein. Fake news detection on social media using geometric deep learning. *arXiv preprint arXiv:1902.06673*, 2019

⁴² Yi Han, Shanika Karunasekera, and Christopher Leckie. Graph neural networks with continual learning for fake news detection from social media. *arXiv preprint arXiv:2007.03316*, 2020

⁴³ Jonathan M Stokes, Kevin Yang, Kyle Swanson, Wengong Jin, Andres Cubillos-Ruiz, Nina M Donghia, Craig R MacNair, Shawn French, Lindsey A Carfrae, Zohar Bloom-Ackermann, et al. A deep learning approach to antibiotic discovery. *Cell*, 180(4):688–702, 2020

45.6 Summary

1. Graph Attention Networks (GATs) allow you to weight the contribution of each of node v 's neighbors with a learnable function, which can help you further minimize your loss function. This makes them a more general version of GCNs, which have fixed attention weights.
2. Graph transformers are GATs with multiple independently initialized and learned attention functions. The results of these functions are then combined to produce the layer's output.
3. Graph variational autoencoders can help you generate random graphs. They first learn the parameters behind the distributions of the embeddings and then use these parameters to generate realistic random embeddings, from which they can reconstruct a new random adjacency matrix.
4. In Generative Adversarial Networks we have a generator and a detector. The generator tries to make synthetic data as similar to the real data as possible. The detector tries to distinguish synthetic data from real one. This can lead to better and better generators.
5. Autoregressive models allow you to represent and learn dependencies between edges when generating a graph, by treating it as a sequence of zeros and ones whose next entries depend on the previously generated ones.
6. You can have more sophisticated message-passing techniques that operate on sets rather than numbers, and run the message-passing process in a continuous way rather than in discrete steps.
7. Minibatching by sampling the nodes' neighborhood can help with computational efficiency, removing random edges can help avoiding overfitting, and adding virtual edges can help when your network is too sparse for effective message-passing.

45.7 Exercises

1. Implement a simple attention mechanism by replacing the $\hat{D}^{-1/2}(I + A)\hat{D}^{-1/2}$ term from the function you developed in the exercises of the previous chapter. The new α term comes from the edge weights in the third column of the network at <http://www.networkatlas.eu/exercises/45/1/network.txt>. The features are at <http://www.networkatlas.eu/exercises/45/1/features.txt>. Then run it on that network. (For the purposes of this and the following exercises, you can use a completely random W)

2. Implement a simple transformer by repeating the attention operation from the previous exercise with the alternative weights in the third, fourth, and fifth column of <http://www.networkatlas.eu/exercises/45/1/network.txt>. Combine them with a final layer averaging all the attention heads.
3. Implement a simple graph variational autoencoder. Learn the embeddings with the transformer you just implemented on the network from the previous exercise. Then calculate each node embedding's mean and standard deviation. Then generate ten random embeddings with the same average and standard deviations.
4. Implement a basic continuous message-passing. Each node averages its own embedding with the average of its neighbors times a factor k . Try $k = 0.5$, $k = 1$, and $k = 1.33$ with the network from the previous exercises.

Part XII

Holistic Network Analysis

46

Graph Summarization

Graph summarization is a data mining class of algorithms that take an input graph and reduce its size – summarizing it – returning a smaller graph as an output. The output graph should respect the salient characteristics of the input graph, so that analyses performed on the output return results that can be used to reconstruct what the whole input graph would return¹.

There are many reasons why one would want to perform graph summarization. The main ones are four:

- Algorithmic speedup: this is the classic motivation². If your original graph has, like Facebook, $\sim 10^9$ nodes, even the most elementary algorithms will take a long time to run. This might be a problem if, for instance, you want to perform online analysis and return results in real time. If you manage to reduce the size of your network to $\sim 10^6$ nodes, this would buy you a lot of time and reactivity.
- Storage facilitation: hard disks might be cheap nowadays, but they ain't free. Again using the Facebook example, it might not be a problem storing $\sim 10^9$ nodes, but if all those nodes perform several activities per day, you might start to get into trouble. Moreover, you will have to hit some physical limits: even if you can create a system with several petabytes of storage capability, if you want to *use* those petabytes of data you have to move them around, and all of a sudden the speed of light seems so slow.
- Noise reduction: noise creeps up inside your data at every twist and turn. Storing and using your full network as it is measured might not be a good idea. Graph summarization can help you to smooth out the noise using information theoretic techniques, reconstructing the underlying signal.
- Visualization: as we will see in Chapter 51, plotting a network is hard and takes a lot of time. No one will ever visualize directly

¹ Yike Liu, Tara Safavi, Abhilash Dighe, and Danai Koutra. Graph summarization methods and applications: A survey. *ACM Computing Surveys (CSUR)*, 51(3):1–34, 2018b

² Tomás Feder and Rajeev Motwani. Clique partitions, graph compression and speeding-up algorithms. *Journal of Computer and System Sciences*, 51(2): 261–272, 1995

a $\sim 10^9$ node network. If you summarize it so that all its visual features are respected, you might then be able to have something meaningful to show.

Before going on the specific methods, I need to clarify what this chapter is about and what it isn't about. This chapter focuses on four main approaches to summarize graphs:

- Aggregation (Section 46.1): collapsing nodes – and the edges connecting them – into super nodes;
- Compression (Section 46.2): finding a set of rules enabling you to encode your network by using fewer bits than its adjacency matrix;
- Simplification (Section 46.3): tossing “unimportant” nodes and edges;
- Influence-based (Section 46.4): finding the smallest possible graph which is still able to describe propagation events in the same way as the original one.

They all have one thing in common: they attempt to reduce a graph by lowering the number of nodes and edges such that the output is another – smaller – graph which represents the entire structure, only simplified.

In this sense, graph summarization is *not* network sampling, and I will also not use space in this chapter for other similar methods such as low-rank approximations.

Graph summarization is fundamentally distinct from network sampling (Chapter 29) even if both branches start from the same point: networks are too large and we cannot take them all in at once. However, in network sampling, you explore a *part* of the network and you operate on the observed structure *directly*. Neither is true in graph summarization. In summarization you want to analyze and understand the *whole* structure, and you do so *indirectly* by manipulating it. There are methods that attempt summarization-by-sampling^{3,4}, but I'm not going to cover them.

Conversely, low-rank approximations seek to reduce the data size with low reconstruction error⁵. This is practically a Principal Component Analysis technique that is specialized to work on the adjacency matrices of a large sparse graph, rather than on generic attribute tables. Thus one could consider this more akin to matrix factorization techniques, for which I invite you to refer to Section 5.6.

Finally, a word about how to evaluate your graph summary. Many methods come with their own quality measure that they are trying to optimize. Thus you could use one of these measures to decide

³ Marc Najork, Sreenivas Gollapudi, and Rina Panigrahy. Less is more: sampling the neighborhood graph makes salsa better and faster. In *Proceedings of the Second ACM International Conference on Web Search and Data Mining*, pages 242–251, 2009

⁴ Jihoon Ko, Yunbum Kook, and Kijung Shin. Incremental lossless graph summarization. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 317–327, 2020

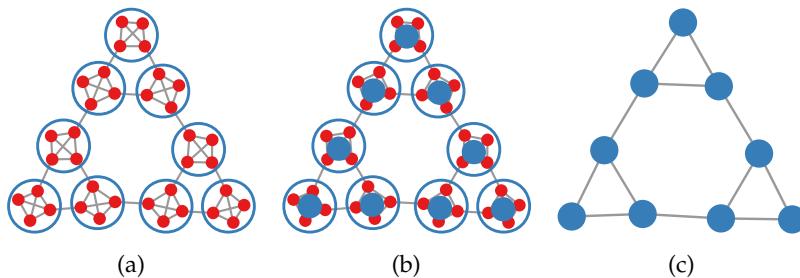
⁵ Jimeng Sun, Yinglian Xie, Hui Zhang, and Christos Faloutsos. Less is more: Compact matrix decomposition for large sparse graphs. In *Proceedings of the 2007 SIAM International Conference on Data Mining*, pages 366–377. SIAM, 2007b

whether you have obtained a good summary or not. For instance, in the compression class we're trying to minimize the number of bits needed to describe the graph.

However, Part X of this book – on community discovery – should have drilled something in your head: the real quality criterion you want to run is dependent on what you want to do with your graph summary. Thus the ideal test should be something as closely related to your final analysis task as possible. For instance, you might want to preserve some properties of interest – e.g. the clustering coefficient. The more you depart from the original value, the more poorly your method is performing.

46.1 Aggregation

In the aggregation approach, the idea is to take the original structure and start aggregating nodes – or edges – into superstructures that stand in for the observed ones. This process is normally guided by some function that determines the quality loss of each aggregation operation. The function must be designed with some application in mind (e.g. community discovery, link prediction, or others).



We already saw a flavor of this approach when we discussed hierarchical community discovery in Section 37.1. Figure 46.1 is a reprise of Figure 37.2 and is a great example of the aggregation approach. In it, we use a community discovery technique to highlight densely connected modules, and we then collapse each into a “super node”.

Another example from the past comes from Section 33.2, where we discussed graph condensation, another summarization technique. In the case of condensation, it involves neither communities nor cliques – clique-reduction⁶ is another aggregation method –, but strongly connected components in directed networks.

There's also an approach from a future section of this book. When visualizing networks, one thing you need to decide is the positioning of the nodes onto a 2D plane. This is the problem of finding a good layout for your graph, and I'll talk in depth about this in Chapter

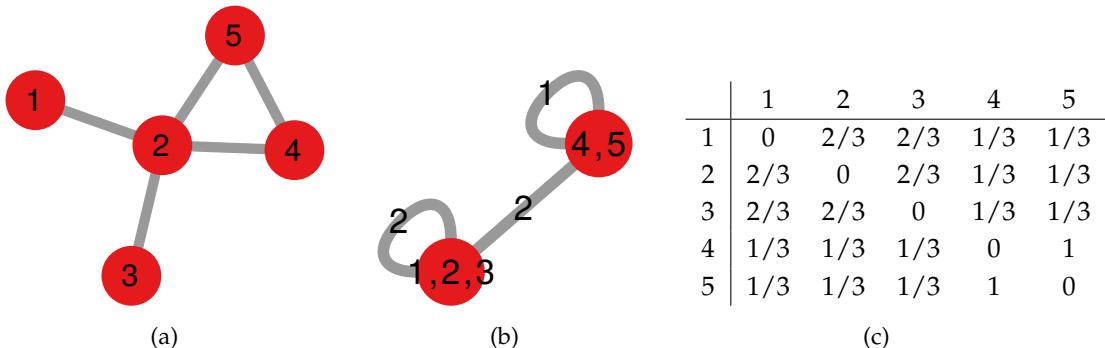
Figure 46.1: (a) An input graph with highlighted cliques. (b) Aggregation step: we consider each clique as a “supernode” (in blue). (c) The summarized graph.

⁶ Hiroshi Ishikawa. Higher-order clique reduction in binary graph cut. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2993–3000. IEEE, 2009.

51. In that chapter, you'll see that one of the biggest problems is that nodes sometimes snuggle together a bit too closely, overlapping with each other. One could identify such nodes that tend to occupy the same position in space and simply aggregate them into a super node⁷, and use this information to bundle up edges as well⁸ – edge bundling is a classic visualization improvement I'll discuss in Section 51.3.

Of course, community discovery, graph condensation, or visualization were not originally developed with summarization in mind. Thus it is possible to design node aggregation methods that are specialized for summarization, even if inspired by other related approaches. One is Grass^{9,10}. In Grass one performs the node aggregation in such a way that the errors in reconstructing the original adjacency matrix are minimized.

Suppose you condensed the graph in Figure 46.2(a) into the graph in Figure 46.2(b). Now all you have is Figure 46.2(b), but you might want to know what is the probability that nodes 1 and 4 are connected. You can reconstruct Figure 46.2(a)'s adjacency matrix via Figure 46.2(b)'s – and keeping track of the original number of edges inside and between each super node.



For instance, if two nodes u and v are in the same super node a , then their expected connection probability is $|E_a|/(|V_a|(|V_a| - 1))$, namely the number of edges collapsed inside a over all edges a could contain. Vice versa, if u and v are in different super nodes a and b , then their connection probability is $|E_{ab}|/(|V_a||V_b|)$, again: number of edges between a and b over all the possible edges that there could be. Thus, the reconstructed adjacency matrix of the original graph is the one in Figure 46.2(c). The quality function guiding this process is mutual information: the higher the mutual information between the original and the reconstructed matrix, the better the aggregation is.

Other approaches compress structurally equivalent nodes¹¹ – see Section 15.2 for a refresher on structural equivalence.

Respecting the adjacency of nodes is not necessarily the only

⁷ Emden R Gansner and Yifan Hu. Efficient node overlap removal using a proximity stress model. In *International Symposium on Graph Drawing*, pages 206–217. Springer, 2008

⁸ Emden R Gansner, Yifan Hu, Stephen North, and Carlos Scheidegger. Multi-level agglomerative edge bundling for visualizing large graphs. In *2011 IEEE Pacific Visualization Symposium*, pages 187–194. IEEE, 2011

⁹ Kristen LeFevre and Evinaria Terzi. Grass: Graph structure summarization. In *Proceedings of the 2010 SIAM International Conference on Data Mining*, pages 454–465. SIAM, 2010

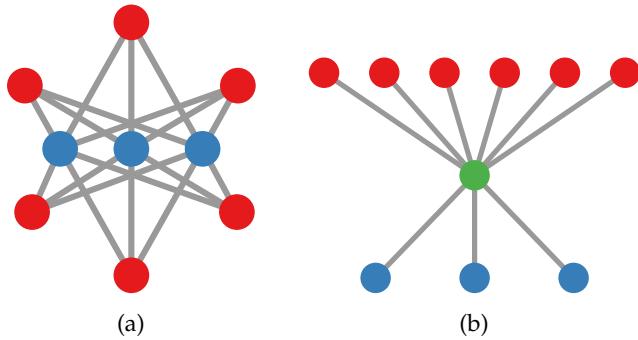
¹⁰ Matteo Riondato, David García-Soriano, and Francesco Bonchi. Graph summarization with quality guarantees. *Data mining and knowledge discovery*, 31(2):314–349, 2017

Figure 46.2: (a) An input graph. (b) Aggregation of (a). Node labels report the nodes collapsed into the super node. Edge labels record the number of edges inside or between super nodes. (c) The adjacency matrix of (a) as reconstructed via (b).

¹¹ Hannu Toivonen, Fang Zhou, Aleksi Hartikainen, and Atte Hinkka. Compression of weighted graphs. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 965–973, 2011

reasonable guiding principle for your aggregation. As I mentioned previously, one might want to perform summarization to aid visualization. In this case, one might want to just simplify complex motifs that would tangle up your visualization¹².

Since we're shifting perspectives, we might as well keep shifting them. So far, we have assumed that aggregation involves the collapse of nodes into super nodes. However, we could very well collapse *edges* instead. In this case, the edge is aggregated into what we call a "compressor", or virtual node. The idea is that high degree nodes, especially those embedded in very dense parts of the network, are at the center of a lot of redundant information.



¹² Cody Dunne and Ben Shneiderman. Motif simplification: improving network visualization readability with fan, connector, and clique glyphs. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 3247–3256, 2013

Figure 46.3: (a) An input graph. Nodes in red have low degree, nodes in blue have high degree. (b) Its summarization via edge aggregation into a compressor (in green).

Take Figure 46.3(a) as an example: its structure can be summarized with a very simple formula – all red nodes connect to all blue nodes. We can compress this information in a node that represent all red-blue connections. The resulting graph – in Figure 46.3(b) – describes exactly the same structure, but does so with nine edges instead of 18, at the price of adding a single node. This looks a bit like the community affiliation graph we saw in Section 38.4, and in fact I'll have you notice that – in this case – we're practically just compressing a 6,3-clique.

46.2 Compression

If you're familiar with zipped archives and programs like WinRAR, you already know what the guiding principle of this branch of graph summarization is. The idea here is to compress the original structure so that we minimize the number of bits we need to encode it. Suppose you're compressing a text in ASCII format. Each letter will cost you seven bits. However, by looking at the text, you realize that many ASCII characters never appear. Thus you can re-encode all characters using shorter codes: if you only use 40 distinct characters, you only need a bit more than five bits per character, reducing a 1MB text into $\sim 760kB$. You can do better than that, realizing that most of the

times the character “h” follows specific other characters and so on. In practice, you’re modeling your text with a model M . Encoding M takes some bits, but it saves many more. This is following the same philosophy as the Infomap community discovery approach we saw in Section 35.2.

Translating this into graph-speak, you want to construct a model M of your graph G so that the length L describing both is minimal, or: $\min L(G, M) = L(M) + L(G|M)$. An example¹³ creates M using a two-step code: (i) each super node in the summary is connected, in the original graph, to all nodes in all super nodes adjacent to it in the summary; and (ii) we correct every edge mistake with an additional instruction.

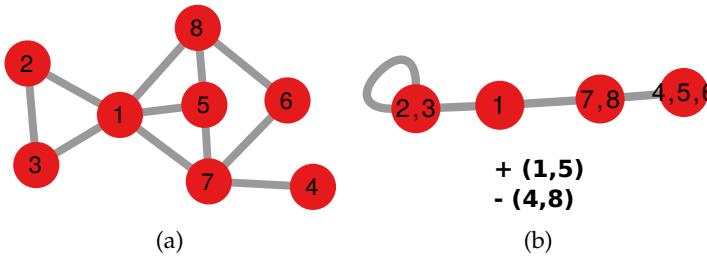


Figure 46.4 shows the approach. The original graph in Figure 46.4(a) can be compressed in the graph in Figure 46.4(b). However, the summary is not perfect. It assumes the existence of an edge that does not really exist, and it misses another edge that exists. We add these two rules to the model M and now the summary is a perfect reconstruction of Figure 46.4(a). In Figure 46.4(b) we say that nodes 7 and 8 are connected to nodes 4, 5, and 6. This is mostly accurate, but not completely correct: we need the additional rule that nodes 4 and 8 do not connect to each other.

The objective now is to find the best combination of summary and additional rules that uses as little information as possible, given or take a margin of error you can set as parameter. There are many information-theoretic approaches in this category^{14,15,16}.

A natural domain of application for compression-based summarization is in the description of evolving networks. In practice, one has many snapshots of the same network and they are trying to reconstruct what the whole network looks like¹⁷. The idea is to find the model that is able to best represent all the snapshots you collected.

You might feel like aggregation and compression are basically the same category. After all, if you look at Figure 46.4, what you’re seeing is basically an aggregation strategy. The fundamental difference between the two categories is the existence of the model M . In aggregation, there is no M : we simply brute force our way through

¹³ Saket Navlakha, Rajeev Rastogi, and Nisheeth Shrivastava. Graph summarization with bounded error. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 419–432, 2008

Figure 46.4: (a) An input graph. (b) Its summarization via minimization of description length. I label each super node with the list of nodes it contains. On the bottom, the additional rules we need to reconstruct (a).

¹⁴ Paolo Boldi and Sebastiano Vigna. The webgraph framework i: compression techniques. In *Proceedings of the 13th international conference on World Wide Web*, pages 595–602, 2004

¹⁵ Sebastian E Ahnert. Power graph compression reveals dominant relationships in genetic transcription networks. *Molecular BioSystems*, 9(11):2681–2685, 2013

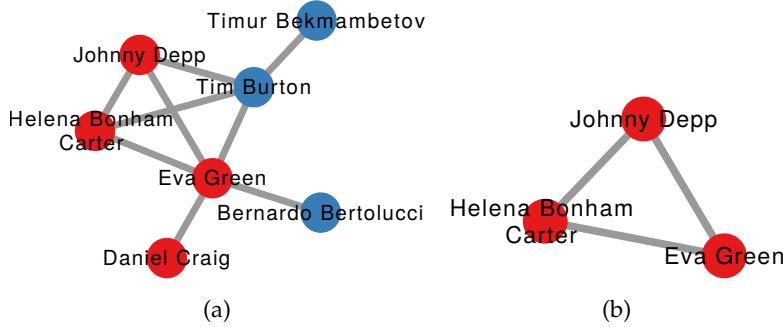
¹⁶ Danai Koutra, U Kang, Jilles Vreeken, and Christos Faloutsos. Vog: Summarizing and understanding large graphs. In *Proceedings of the 2014 SIAM international conference on data mining*, pages 91–99. SIAM, 2014

¹⁷ Neil Shah, Danai Koutra, Tianmin Zou, Brian Gallagher, and Christos Faloutsos. Timecrunch: Interpretable dynamic graph summarization. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1055–1064, 2015

the graph to save every node or edge we can, regardless whether we're uncovering common patterns or not. The existence of M in the compression category, instead, forces us only to perform an aggregation if it results in a leaner and more elegant M . As a consequence, M itself is an important result of the procedure, because it contains information about the common patterns you can find in your original structure.

46.3 Simplification

In this class we group solutions that are the lovechildren of network sampling (Chapter 29) and backboning (Chapter 27). The idea here is not to create “super nodes” like in the previous two classes. Here, we look at the original structure. However, we simplify it by removing nodes and edges that we consider “unimportant”. Sampling and backboning techniques can be considered simplification strategies that are purely structural: they only use information coming from the graph’s topology.



Those are not the only valid approaches. If the graph also has metadata attached to nodes – or edges – we can exploit them. For instance, Ontovis¹⁸ allows the simplification of the graph via the specification of a set of attribute values we’re interested in studying. For instance, the graph in Figure 46.5(a) can be simplified into the one in Figure 46.5(b), if we’re only interested in knowing the relationships between actors (node type value) working with Tim Burton (topology attribute).

Ontovis finds the best way to simplify the graph, primarily focusing on its visual characteristics when plotted in 2D: it is first and foremost a visualization-aiding tool. Ontovis focuses on node attributes, but one could also switch their focus to edges¹⁹.

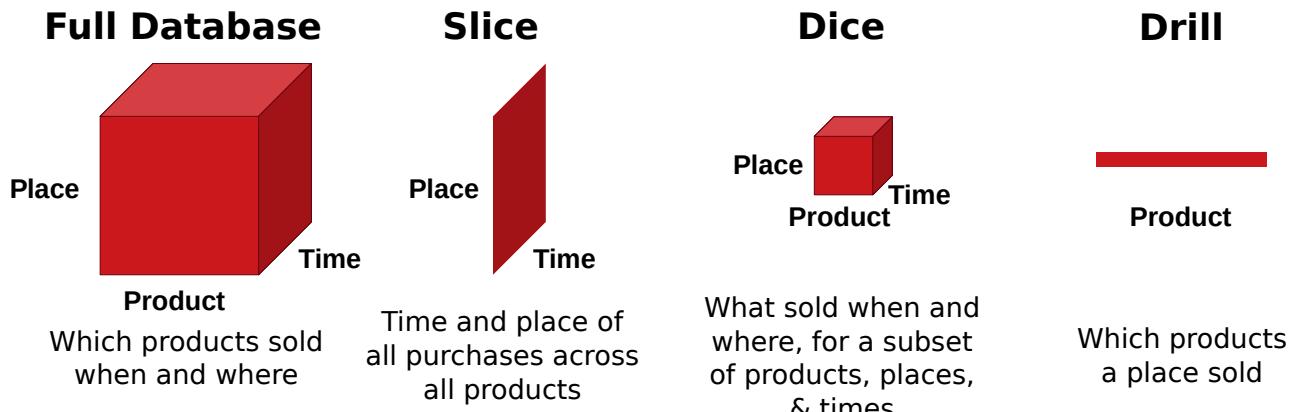
Note, also, that another difference with sampling is that, in graph simplification, we are not really interested in preserving any specific property of the original graph. This is, instead, a core focus of

Figure 46.5: (a) An input graph: directors (in blue) and actors (in red) connected if they collaborated with each other. (b) Its simplification via the selection of only actor-type nodes directly connected to Tim Burton.

¹⁸ Zeqian Shen, Kwan-Liu Ma, and Tina Eliassi-Rad. Visual analysis of large heterogeneous social networks by semantic and structural abstraction. *IEEE transactions on visualization and computer graphics*, 12(6):1427–1439, 2006

¹⁹ Cheng-Te Li and Shou-De Lin. Ego-centric information abstraction for heterogeneous social networks. In *2009 International Conference on Advances in Social Network Analysis and Mining*, pages 255–260. IEEE, 2009

network sampling.



When not necessarily focusing on visualization, the simplification approach uses a database metaphor to help the user navigate between different “views” of the network data. A classical database infrastructure is OLAP²⁰, which stands for OnLine Analytical Processing. In an OLAP database you have data that is inherently multidimensional, for instance sales can happen in different shops, at different times, via different product categories, and customer classes, etc. OLAP allows you to represent this with a “data cube” that you can slice and dice to aggregate the dimensions. Figure 46.6 shows you a visual representation of what different operations look like – and mean.

Graph OLAP^{21,22,23} is fundamentally the same thing applied to networks, using node/edge attributes and characteristics to drive the simplification procedure. While traditional graph OLAP works best with categorical node attributes, there are also ways to dice and slice your graph using numeric attributes²⁴, allowing you to also consider node properties such as the degree.

Another related category of approaches is “graph sketches”^{25,26,27}.

46.4 Influence Based

In influence-based summarization one is interested in having a summarized graph in which influence events follow the same dynamics as in the original graph. That is: if something is flowing through the graph, percolating node to node, the percolation in the summarized graph follows the same dynamics as in the original graph.

There are a few ways to wrap your head around this concept. I feel that I can provide two very different approaches for you, that should serve different types of understanding graph dynamics. The

Figure 46.6: An example of operations on an OLAP database.

²⁰ Surajit Chaudhuri and Umeshwar Dayal. An overview of data warehousing and olap technology. *ACM Sigmod record*, 26(1):65–74, 1997

²¹ Yuanyuan Tian, Richard A Hankins, and Jignesh M Patel. Efficient aggregation for graph summarization. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 567–580, 2008

²² Chen Chen, Xifeng Yan, Feida Zhu, Jiawei Han, and S Yu Philip. Graph olap: Towards online analytical processing on graphs. In *ICDM*, pages 103–112. IEEE, 2008

²³ Peixiang Zhao, Xiaolei Li, Dong Xin, and Jiawei Han. Graph cube: on warehousing and olap multidimensional networks. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pages 853–864, 2011

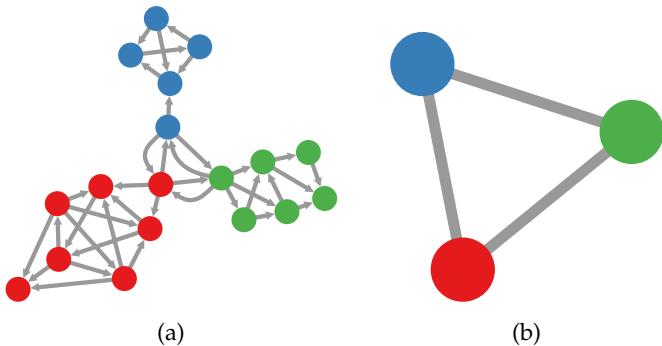
²⁴ Ning Zhang, Yuanyuan Tian, and Jignesh M Patel. Discovery-driven graph summarization. In *ICDE*, pages 880–891. IEEE, 2010

²⁵ Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Graph sketches: sparsification, spanners, and subgraphs. In *SIGMOD-SIGACT-SIGAI*, pages 5–14, 2012

²⁶ Edo Liberty. Simple and deterministic matrix sketching. In *SIGKDD*, pages 581–588, 2013

²⁷ Mina Ghashami, Edo Liberty, and Jeff M Phillips. Efficient frequent directions algorithm for sparse matrices. In *SIGKDD*, pages 845–854, 2016

first rests on a data-driven approach. Suppose you have a social network, where nodes are people. Some users are early adopters of a new product. As they perform an action, some of their friends will see it and will imitate them. This means that you can detect “tribes” of people reacting with the same timing to the same stimuli. This is basically like doing community discovery, with the difference being that you’re not maximizing internal density, but simply the synchronization of an action. What I described is, for instance, what GuruMine detects, and you can use Section 21.2 as a refresher.



Visually, this would look like Figure 46.7. The central hubs influence each other, and each is responsible for influencing their branch. Thus one could summarize the graph as a clique of interacting tribes. Of course, you don’t have to use GuruMine for this. In some cases, researchers have used special adaptations to estimate community-level influence²⁸.

There’s a completely different way to interpret summarization by influence preservation. We have seen that the spectrum of the Laplacian can be used to partition a graph – solving the cut problem (Section 11.5). This is related to diffusion processes: the reason why the eigenvectors of the Laplacian help you with cutting is because they are a sort of simulation of a diffusion process, and the edges to cut are the bottlenecks through which things cannot flow efficiently. For now, let’s take this for granted, but we’ll see more about this relationship in Section 47.2, where we’ll talk about using the Laplacian to estimate distances between sets of nodes by releasing a flow from the nodes in the origin to the nodes in the destination.

If we want to summarize the graph to preserve these diffusion properties, we can use the Laplacian to guide our process. What we’re after, in the simplest possible terms, is a smaller Laplacian matrix, with fewer rows and columns, that has the same eigenvectors²⁹.

Among other interesting approaches there is SPINE³⁰. In SPINE, one analyzes many influence events in the network. Then, SPINE only keeps in the network the edges that are able to better explain

Figure 46.7: (a) An influence graph. The edge direction tells you who influences whom. The node color tells you the detected “tribes”. (b) The summary graph, compressing all tribes into a node, to preserve influence dynamics.

²⁸ Yasir Mehmood, Nicola Barbieri, Francesco Bonchi, and Antti Ukkonen. Csi: Community-level social influence analysis. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 48–63. Springer, 2013

²⁹ Manish Purohit, B Aditya Prakash, Chanhyun Kang, Yao Zhang, and VS Subrahmanian. Fast influence-based coarsening for large networks. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1296–1305, 2014

³⁰ Michael Mathioudakis, Francesco Bonchi, Carlos Castillo, Aristides Gionis, and Antti Ukkonen. Sparsification of influence networks. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 529–537, 2011

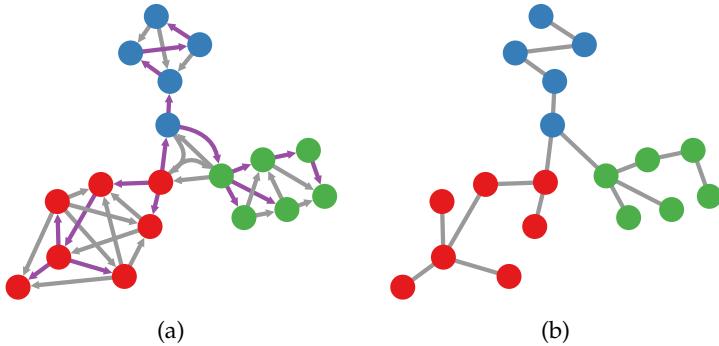


Figure 46.8: (a) An influence graph. The edge color represents the path taken by an hypothetical spreading event. (b) The summary graph, including only the edges used in the spreading process.

the paths of influence you observe. You might realize that an edge is never used to transport information, and thus you could remove it from the structure without hampering your explanatory power. Figure 46.8 shows an example of this.

46.5 Summary

1. Graph summarization is the task of reducing the size of your graph so that you can facilitate different operations, such as analysis, storage, data cleaning, and/or network visualization.
2. There are many ways to perform summarization. One is to do so via aggregation: you find coherent modules in your network and you collapse them into the same node, aggregating all incoming and outgoing edges.
3. A second category is compression: you similarly try to aggregate the graph, but this time you record your operation in a model. The model also has to be encoded, and thus you have to find the simplest possible model that best represents your original graph.
4. The third approach is simplification. This is especially relevant for visualization: you want to simplify the graph so that motifs that would clutter its representation are reduced and do not cross each other.
5. Finally, you might have influence in mind: some process is spreading on your network and you want to keep only the connections that are most likely responsible for that process to percolate through the nodes.
6. Making a summary of a chapter about summarization is delightfully meta and I'm having a hell of a good time.

46.6 Exercises

1. Perform label percolation community discovery on the network at <http://www.networkatlas.eu/exercises/46/1/data.txt>. Use the detected communities to summarize the graph via aggregation.
2. The table at <http://www.networkatlas.eu/exercises/46/2/diffusion.txt> contains the information of which node (first column) influenced which other node (second column). Use it to summarize the graph by keeping only the edges used by the spreading process.
3. Summarize the summary you generated answering question #1 with the data from question #2. Do you still obtain a connected graph?

47

Node Vector Distance

In Chapter 13 we saw a way to determine the distance between two nodes: the number of edges you need to cross in the graph to go from one to the other. Alternatively, one could use the hitting time (Section 11.3): how long it will take for a random walk to hit both nodes. However, sometimes you don't want the distance between a pair of nodes. Sometimes you want to ask: what is the distance between a *group* of nodes and another, given that some nodes might weigh more than others?

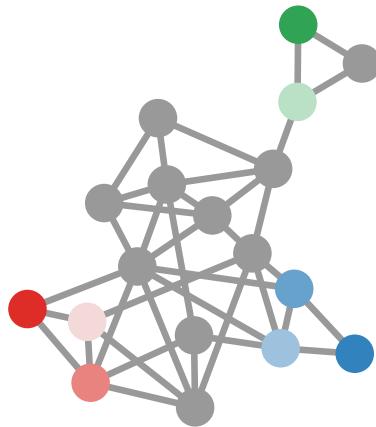


Figure 47.1 provides an intuitive example to understand this question. Is the total red hue distributed closer to the blue color, or to the green? How much does it matter that the darker nodes carry more weight in estimating such distance?

I call this problem the Node Vector Distance, and it has many applications:

- In computer vision^{1,2}, we represent an image as a graph of points of interest, with different values, proportional to how much light or color is in them. Two images can then be compared by estimating how much "light" we have to transport from the interest

Figure 47.1: A graph with different highlighted groups of nodes. The intensity of the color is proportional to how much weight is on the node.

¹ Shmuel Peleg, Michael Werman, and Hillel Rom. A unified approach to the change of resolution: Space and gray-level. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(7):739–742, 1989

² Yossi Rubner, Carlo Tomasi, and Leonidas J Guibas. The earth mover's distance as a metric for image retrieval. *International journal of computer vision*, 40(2):99–121, 2000

points of one to the interest points of the other. Small amounts will indicate that the images are similar.

- In economics^{3,4}, you can represent products as nodes, connected if there is a significant number of countries that are able to co-export significant quantities of them. A country occupies the products in this network it can export. From one year to another, the country will change its export basket, by shifting its industries to different products. How dynamic is the country's export basket?
- In epidemics^{5,6,7}, a disease occupies the nodes in a social network it has infected. Across time, the disease will move from a set of infected individuals to another. Similarly, in viral marketing, product adoption can be modeled as a disease.

All these cases can be represented by the same problem formulation. You have a network G . Then you have two vectors: an origin vector p and a destination vector q . Both p and q tell you how much value there is in each node. p_u tells you how much value there is in node u at the origin, and q_v tells you how much value there is in node v at the destination.

All you want to do is to define a $\delta(p, q, G)$ function. Given the graph and the vectors of origin and destination, the function will tell you how far these vectors are. There are many ways to do so, which are organized in a survey paper⁸, on which this chapter is based. Before we jump into the network distances, it is probably wise to have a refresher on non-network distances, since it will allow us to introduce concepts that will be helpful later.

47.1 Non-Network Distances

How to estimate node vector distances on networks is a new and difficult problem. Let's take it easy and first have a quick refresher on the many ways we can estimate distances of vectors *without* a network. The easiest way to do it is by assuming a vector of numbers just represents a set of coordinates in space. If you're on Earth, with three numbers you can establish your latitude, longitude and altitude. That is enough to place you on a position in a three dimensional space. Another person might be at a different latitude, longitude and altitude than you. What is the distance between you and your friend? Easy! You throw a straight rope between you and your friend and its length is the distance between you. This is the Euclidean distance.

In Section 5.4 I did my very best to connect this intuitive idea in real life with linear algebra operations. The pain you felt back then should pay off now. To sum up, if p and q are the vectors defining your two positions in space, the Euclidean distance is

³ Ricardo Hausmann, César A Hidalgo, Sebastián Bustos, Michele Coscia, Alexander Simoes, and Muhammed A Yildirim. *The atlas of economic complexity: Mapping paths to prosperity*. Mit Press, 2014

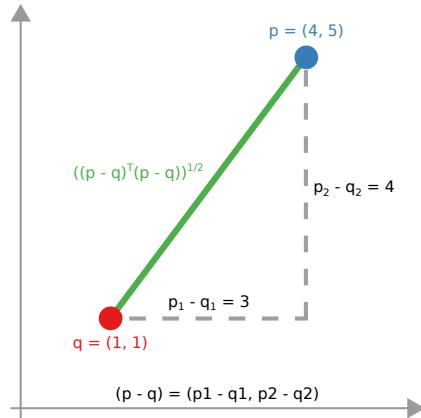
⁴ César A Hidalgo, Bailey Klinger, A-L Barabási, and Ricardo Hausmann. The product space conditions the development of nations. *Science*, 317(5837):482–487, 2007

⁵ Vittoria Colizza, Alain Barrat, Marc Barthélémy, and Alessandro Vespignani. The role of the airline transportation network in the prediction and predictability of global epidemics. *Proceedings of the National Academy of Sciences of the United States of America*, 103(7):2015–2020, 2006

⁶ Ayalvadi Ganesh, Laurent Massoulié, and Don Towsley. The effect of network topology on the spread of epidemics. In *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, volume 2, pages 1455–1466. IEEE, 2005

⁷ Romualdo Pastor-Satorras and Alessandro Vespignani. Epidemic dynamics and endemic states in complex networks. *Physical Review E*, 63(6):066117, 2001

⁸ Michele Coscia, Andres Gomez-Lievano, James McNerney, and Frank Neffke. The node vector distance problem in complex networks. *ACM Computing Surveys*, 2020



$((p - q)^T I(p - q))^{1/2}$, where I is the identity matrix. Figure 47.2 should help refreshing your intuition behind the Euclidean distance.

You can put any matrix in this formulation instead of I , as long as they are positive semi-definite matrices. Why would you want to put any other matrix in there? Remember that matrices are nothing more than spatial transformations: they tell you how to bend and warp your space. So putting something else than I will make the $(p - q)$ vector bend in special ways. What are these ways? Well, the role of I in the Euclidean distance is to tell us that each dimension of the vector makes the exact same contribution to the distance. So the vectors $(0, 1, 0)$ and $(0, 0, 1)$ are exactly equidistant from $(1, 0, 0)$.

However, in some cases, we might notice that some dimensions are correlated: they change together. So, if the vectors differ in a direction opposite from what we would expect, this change should count more than the one we would expect. We know we can calculate the covariance of two variables (Section 3.4), so we can store the relations between the dimensions of p and q in their covariance matrix $cov(p, q)$ and, since we want to count more when we have a change going in the opposite direction from the expected, we invert the matrix: $((p - q)^T cov^{-1}(p, q)(p - q))^{1/2}$. This is the Mahalanobis distance.

Figure 47.3 shows you an example. The two dimensions of the

Figure 47.2: Euclidean distance in $m = 2$ dimensions. We build the special $p - q$ vector to have, at its i th entry, the difference between the i th entries of p and q .

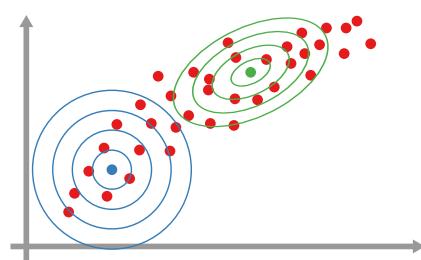


Figure 47.3: An example of two correlated variables. The blue concentric circles represent Euclidean distances centered on the blue point. The green concentric ellipses represent Mahalanobis distances centered on the green point.

scatter plot are clearly correlated. This means that moving in a direction orthogonal to the correlation line counts for more distance covered: it is an unexpected move. That is what the Mahalanobis distance, in green, is capturing. The Euclidean distance is oblivious to this and, for it, all directions are equally important. Crossing the same number of lines means covering the same distance: while the direction you choose in the Euclidean case doesn't matter, it does in the Mahalanobis distance.

In the next section we'll see how we can use a graph's topology to weight the dimensions of our vector in such a way that the difference between p and q is constrained to happen in the space described by our network. This boils down to find a smart positive semi-definite matrix to put in the place occupied by I or $\text{cov}^{-1}(p, q)$.

Are we done with non-network vector distances? Of course not: we haven't even started yet. I already mentioned a few other distances when I talked about network projections in Section 26.2. We have correlation distances, cosine distances, etc. I need not to go into details on how each of these measures work. I will only explain the cosine distance just to make a point: the Euclidean way of estimating distances is not the only proper way.

What do I mean by this? Consider the rope example I made before. For some distance measures, the length of the rope between you and your friend is not the most important thing. You can have two points requiring a longer rope to connect that could be closer to each other than points requiring a shorter rope. This is the case of the widely used cosine distance.

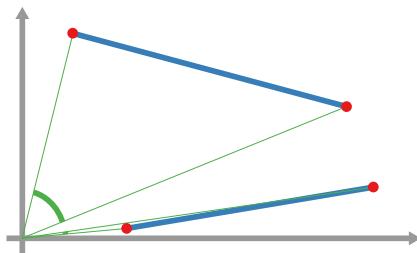


Figure 47.4: The difference between the Euclidean distance between two points (in blue) and their cosine distance (thick green arc tracing the angle between the two points).

In cosine distance you look at the *angle* made by the vectors connecting the two points, as Figure 47.4 shows with a thick green line. The distance between them is one minus the cosine of that angle. This is useful, because the cosine is 1 for angles of zero degrees and 0 for angles at ninety degrees. Two points on the same straight line will have a distance of zero, even if they're infinitely farther apart on such a line. For instance, the two points at the bottom of Figure 47.4 are at a considerable Euclidean distance, but practically neighbors when it comes to cosine distance. Sometimes in life it doesn't matter where

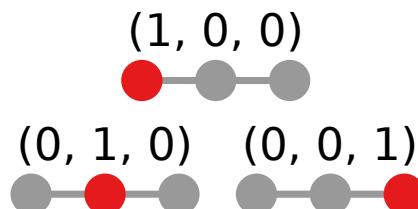
you are, as long as you're going in the same direction.

If you're curious about why we need the cosine distance, the reason is to deal with high-dimensional data. When you have lots of dimensions, data points tend to all roughly be at the same Euclidean distance and it's very difficult to tell differences between them⁹. This is what we normally call the "curse of dimensionality." By refusing to calculate Euclidean distances between the points and considering only the angle they make with each other – i.e. by using cosine distances – we can tame the curse of dimensionality, at least partially.

The importance of the cosine distance is in reminding us that a distance measure does not have to necessarily respect the triangle inequality like the Euclidean distance does. The triangle inequality says that the distance between a and b is always lower than or equal to the distance between a and c and b and c . This is how things work in the real physical world. But it is not the only way things *can* work.

47.2 Generalized Euclideans

In this section we're going to generalize the Euclidean distance by replacing the identity matrix I with some matrix dependent on the topology of the network we're working with. In practice, this means that we are constraining the diffusion process to happen through the edges of the network. We don't want the diffusion to jump between any two pairs of nodes. If the nodes are far apart in the network, such jump should be considered differently than the one happening between two nodes that are directly connected. For instance, in Figure 47.5, I make the case in which the $(0, 1, 0)$ and $(0, 0, 1)$ vectors should not be considered equidistant from $(1, 0, 0)$, because the leftmost and rightmost nodes in this network are not connected, and they are thus farther apart.



⁹ Kevin Beyer, Jonathan Goldstein, Raghu Ramakrishnan, and Uri Shaft. When is "nearest neighbor" meaningful? In *Database Theory—ICDT'99: 7th International Conference Jerusalem, Israel, January 10–12, 1999 Proceedings* 7, pages 217–235. Springer, 1999

Figure 47.5: The graph representation of three vectors. I highlight in red the node corresponding to the entry equal to 1 in the vector.

With its reliance on I , the Euclidean distance considers these cases as equidistant. So we need to replace I with something else. We'll look at three alternatives.

Laplacian

In the Laplacian version¹⁰, we look at the Laplacian of the adjacency

¹⁰ Michele Coscia. Generalized euclidean measure to estimate network distances. In *Proceedings of the International AAAI Conference on Web and Social Media*, volume 14, pages 119–129, 2020

matrix. Remember from Section 8.4, that the Laplacian $L = D - A$, with D being the degree matrix and A the adjacency matrix. Since the smallest eigenvalue of L is zero, L is positive semi-definite.

We use the graph Laplacian because the Laplace operator describes mathematically statuses of equilibrium¹¹. In practice, if you have a liquid on a container making waves – i.e. being out of equilibrium – the Laplace operator will tell you how the diffusion of the liquid will behave when transitioning to its equilibrium state, where there are no more waves.

Just like in the Mahalanobis case, L like cov tells us how close together nodes are. Thus, we need to invert it. If you recall Section 11.4, L is singular and singular matrices – by definition – cannot be inverted. But we're not too picky about what “inverting” means, so we take the Moore-Penrose pseudoinverse.

So, to sum up, the Laplacian's δ function is:

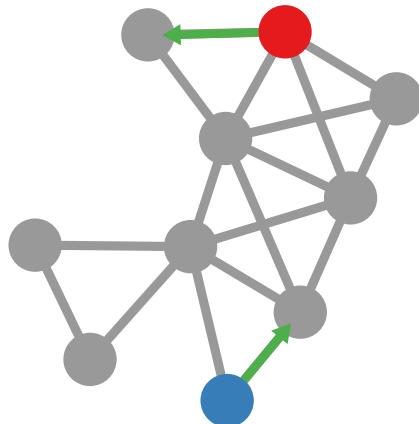
$$\delta(p, q, G) = ((p - q)^T L^\dagger (p - q))^{1/2},$$

with L^\dagger being the pseudoinverse of the graph Laplacian of G .

Markov Chain

Random walks are helpful to estimate node-node distances (Section 14.4). If we are in the situation of Figure 47.6, we could estimate the distance between the red and blue node by simply asking how long it will take for a random walker to go from one node to the other. Here, we generalize this idea to groups of nodes.

In the Markov Chain distance we start from the assumption that, given a starting point p , by looking at G we can construct an expected “next step”, which is $E(q) = Ap$. This expected behavior follows a simple random walk (which is a Markov process, see Section 2.7). In other words, we expect that q should be the result of a



¹¹ Lawrence C Evans. Partial differential equations and monge-kantorovich mass transfer. *Current developments in mathematics*, 1997(1):65–126, 1997

Figure 47.6: To estimate the distance between the red and the blue node, we could release random walkers (in green) from them and calculate how much time it will take for them to arrive at the other node.

one step diffusion of p via random walks. For this to be the case, A needs to be the stochastic adjacency matrix. Here, we also set the diagonal of the adjacency matrix to be equal to one before we transform it in its stochastic version. This is equivalent to add a self loop to all nodes in the network: we want to allow the diffusion process to stand still in the nodes it already occupies.

For each node u in the network, we can calculate the expected occupancy intensity in the next time step by unrolling the previous formula: $E(q_u) = \sum_{v \in V} A_{u,v} p_v$. This is helpful, because it allows us to calculate the standard deviation of this expectation, $\sigma_{u,v}$, which we can do by making a few assumptions on the distribution of this expectation (which are spelled out in the original paper¹²). For now, suffice to say that such deviation is $\sigma_{u,v} = (p_v A_{u,v} (1 - A_{u,v}))$.

Since we now have an expectation and a deviation, we can calculate the z-score of the observation, which is a measure of how many standard deviations your observation is distant from the expectation. We calculate a z-score for each node in the network and place it in its corresponding spot in a diagonal matrix:

$$[Z]_{u,u} = \sum_{v \in V} \sigma_{u,v}^2.$$

Now, the problem of this formulation is that it'd make this distance not symmetric, because to build $\sigma_{u,v}$ we only used p as the origin of the diffusion. So, if $\sigma_{u,v}$ is the deviation of the diffusion from v to u , we can also calculate a deviation of the diffusion from u to v : $\sigma_{v,u}$. This is done as above, switching p 's and q 's places. Then the u,u entry in Z 's diagonal is the sum of $\sigma_{u,v}^2$ and $\sigma_{v,u}^2$.

Finally we can write our distance as:

$$\delta_{p,q,G} = ((p - q)^T Z^{-1} (p - q))^{1/2}.$$

In this distance measure you can tune the propagation duration. Maybe you don't want to make one-step random walks by using simply the stochastic matrix A . Maybe you want to have two steps random walks. In this case, you would use A^2 . To ensure that the two farther apart nodes can still reach each other, you could consider to use A^ℓ , with ℓ being the diameter of the network (Section 13.3). If you were to take A^∞ , then you'd be using the stationary distribution (Section 11.1). In that case, the topology of G doesn't matter any more: the only thing that makes two nodes closer or farther is their degree.

¹² Michele Coscia, Andres Gomez-Lievano, James McNerney, and Frank Neffke. The node vector distance problem in complex networks. *ACM Computing Surveys*, 2020

Annihilation

Let's take that last thought a bit further. If you let your p and q vectors to diffuse via random walks for an infinite amount of time, they will distribute themselves to all nodes of G proportionally to their degree, because they will both tend to approximate the stationary distribution. In the wavy water basin I mentioned before, p and q are simply two different waves conditions, while the stationary distribution is... well ... the stationary distribution: a waveless basin where the water is at the same level everywhere. Mathematically, this means that $\sum_{k=0}^{\infty} A^k q$ and $\sum_{k=0}^{\infty} A^k p$ are the same thing, or $\sum_{k=0}^{\infty} A^k(p - q) = 0$.

Now, the interesting bit is for which value of k this is true or, put in another words, how fast will p and q cancel out. If they cancel each other out quickly, it means that they were already pretty similar to begin with. In fact, that equation would be true at $k = 0$ if $p = q$. So we're interested in the *speed* of that equation. This is given us by the following formula:

$$\delta_{p,q,G} = ((p - q)^T \sum_{k=0}^{\infty} A^k(p - q))^{1/2}.$$

You shouldn't be scared of the infinite sum $\sum_{k=0}^{\infty} A^k$: it converges and there is a proper solution to it, which is in the survey paper I've been citing in this chapter.

47.3 Shortest Path Based

The solutions based on shortest paths start from the assumption that the problem of establishing distances between sets of nodes can be generalized from solving the problem of finding the distance between pairs of nodes. This is a well understood and solved problem: using a shortest path algorithm – for instance Dijkstra's¹³ – one can count the number of edges separating node u to v .

If we take Figure 47.7 as an example, we'd start by collecting a bunch of distances: [2, 3, 3] when starting from node 5 or node 8, and

¹³ Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959

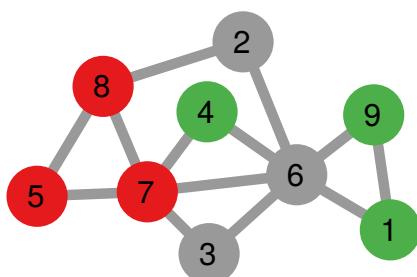


Figure 47.7: A network where the red nodes represent the origins and the green nodes represent the destinations.

$[1, 2, 2]$ when starting from node 7. We then want to aggregate these distances with a given strategy, to define several functions solving the problem.

Since this section deals with shortest paths, a useful convention is to refer to all possible paths between origins and destinations as $P_{p,q}$. This is to avoid to calculate all shortest paths between all pairs of nodes in the network, which is computationally expensive and not necessary, since we don't need the distances between nodes that have a zero value in both p and q . A path length $|P_{u,v}| \in P$ is the minimum number of edges required to cross to move from node u to node v .

There are two subcategories in this group: methods which try to optimize the paths from p to q , and methods which do not. We start from the latter.

Non-Optimized

Here we show a set of possible aggregations of shortest path distances between the nodes in p and q , by taking hierarchical clustering as an inspiration. There are of course more strategies than the ones listed here, but I can't really list them all – and most haven't really been researched yet.

When performing hierarchical clustering, there are three common ways to merge clusters according to their distance¹⁴: single, complete, and average linkage. Single linkage (green in Figure 47.8) means that the distance between two clusters is the distance between their two closest points. On the other hand, complete linkage (purple in Figure 47.8) considers the distance of the two farthest points as the cluster distance. In average linkage (orange in Figure 47.8), one calculates the average distance between all pairs of points in the two clusters as the distance between the clusters.

¹⁴ Gabor J Szekely and Maria L Rizzo. Hierarchical clustering via joint between-within distances: Extending ward's minimum variance method. *Journal of classification*, 22(2):151–183, 2005

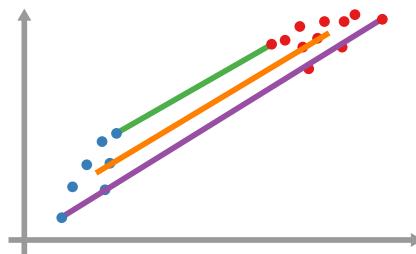


Figure 47.8: Different linkage strategies to estimate the distances between clouds of points: single (green), complete (purple), and average (orange).

Similarly, our aim is to reach the destination from the origin in the minimum distance possible. In the single linkage strategy, the “cost” of reaching a destination node is the distance of it from the closest possible origin node. First, we need to make sure that $\sum p = \sum q$. If that isn't the case, we rescale up the vector with the smallest sum so that this equation is satisfied. For instance, if q had a lower sum, we

transform it: $q' = (\sum p / \sum q)q$.

Then we start a loop. At each iteration, we want to find the pair of closest nodes ($\arg \min_{u,v} |P_{u,v}|$) that can exchange the largest possible value. How much value can two nodes exchange? A node can only give what they have, so we will exchange the minimum of the two values: $\min(p_u, q_v)$. The contribution of this move to the distance is $|P_{u,v}| \min(p_u, q_v)$: we move $\min(p_u, q_v)$ across $|P_{u,v}|$ edges. Once the value is exchanged, we update p_u and q_v to reflect the successful transaction: $p_u = p_u - \min(p_u, q_v)$ and $q_v = q_v - \min(p_u, q_v)$. Eventually, all values would have been transferred, because we ensured that the two vectors sum to the same value, and the iterations will stop.

The complete linkage uses the very same operation: the only difference is using at each step $\arg \max_{u,v} |P_{u,v}|$ instead of $\arg \min_{u,v} |P_{u,v}|$.

This means that we preferentially exchange value between the node pairs that are *farthest*, not closest.

The average linkage is conceptually simpler: it is the weighted average path distance between all $u \in p$ and all $v \in q$:

$$\delta_{p,q,G} = \frac{\sum_{\forall v \in q} \sum_{\forall u \in p} p_u q_v |P_{u,v}|}{\sum p}.$$

Here it doesn't matter what we put in the denominator, since we already ensured that p and q sum to the same value.

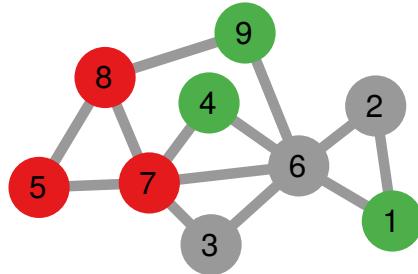


Figure 47.9: A network where the red nodes represent the origins and the green nodes represent the destinations.

Looking at Figure 47.9, we can now estimate the different distances between red and green nodes. In single linkage, we try to find the shortest path to the closest destination from each origin. Origin 8 goes to destination 9 because they are directly connected, and so does origin 7 with destination 4. Origin 5 has to take a path of length 3 to reach destination 1: $5 \rightarrow 7 \rightarrow 6 \rightarrow 1$. Thus, for single linkage, $\delta_{p,q,G} = 1 + 1 + 3 = 5$. If we normalize the vectors beforehand, each step counts for $1/3$, and thus the distance would be $5/3 = 1.\bar{6}$.

The average linkage looks at all nine shortest paths, and calculates an average. The total length of all shortest paths is 18. It then normalizes with the total moved weight: $\sum p = 3$. Thus the average linkage estimates the distance as $18/3 = 6$. If we normalized the

vectors, we would again count each path as contributing one third, i.e. $(18/3)/3 = 2$.

In complete linkage, we perform a similar operation as in single linkage, but looking at the farthest destination for each origin. The farthest destination is $5 \rightarrow 1$, at three steps; then $8 \rightarrow 4$ and $7 \rightarrow 9$ at two steps each. Thus, complete linkage will return $3 + 2 + 2 = 7$ as distance. If we normalized the vectors, we would again count each path as contributing one third, i.e. $7/3 = 2\bar{3}$.

Optimized

Here we try to be a bit smarter than the aggregation strategies we saw so far. In this branch of approaches, we try to optimize this aggregation such that the number of edge crossing is minimized.

If there are no further constraints in this optimization problem, we are in the realm of the Optimal Transportation Problem (OTP) on graphs¹⁵. In its original formulation¹⁶, OTP focuses on the distance between two probability distributions without an underlying network. However, it has been observed how this problem can be applied to transportation through an infrastructure, known as the multi-commodity network flow¹⁷. Specifically, one has to simply specify how distant two dimensions in the vector are. The distance needs to be a metric, and the number of edges in the shortest path between two nodes satisfies the requirement.

In its most general form, the assumption is that we have a distribution of weights on the network's nodes, and we want to estimate the minimal number of edge crossings we have to perform to transform the origin distribution into the destination one. This is a high complexity problem, which has lead to an extensive search for efficient approximations^{18,19,20,21,22,23,24,25}. For what concerns us, all these methods are equivalent: they all solve OTP and the difference between them is how they perform the expensive optimization step. Thus, they all return a very similar distance given p , q and G – plus or minus some approximation due to their optimization strategy –, and fall in the same category.

More formally, in OTP we want to find a set of movements M such that:

$$M = \arg \min_{m_{p_u,q_v}} \sum_{p_u} \sum_{q_v} m_{p_u,q_v} d_{u,v},$$

where p_u and q_v are the weighted entries of p and q , respectively; m_{p_u,q_v} is the amount of weights from p_u that we transport into q_v ; and d_{p_u,q_v} is the distance between them. Then:

¹⁵ Andrew McGregor and Daniel Stubbs. Sketching earth-mover distance on graph metrics. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 274–286. Springer, 2013

¹⁶ Gaspard Monge. Mémoire sur la théorie des déblais et des remblais. *Histoire de l'Académie Royale des Sciences de Paris*, 1781

¹⁷ Frank L Hitchcock. The distribution of a product from several sources to numerous localities. *Studies in Applied Mathematics*, 20(1-4):224–230, 1941

¹⁸ Ira Assent, Andrea Wenning, and Thomas Seidl. Approximation techniques for indexing the earth mover's distance in multimedia databases. In *Data Engineering, 2006. ICDE'06. Proceedings of the 22nd International Conference on*, pages 11–11. IEEE, 2006

¹⁹ Matthias Erbar, Martin Rumpf, Bernhard Schmitzer, and Stefan Simon. Computation of optimal transport on discrete metric measure spaces. *arXiv preprint arXiv:1707.06859*, 2017

²⁰ Montacer Essid and Justin Solomon. Quadratically-regularized optimal transport on graphs. *arXiv preprint arXiv:1704.08200*, 2017

²¹ George Karakostas. Faster approximation schemes for fractional multicommodity flow problems. *ACM Transactions on Algorithms (TALG)*, 4(1):13, 2008

²² Jan Maas. Gradient flows of the entropy for finite markov chains. *Journal of Functional Analysis*, 261(8):2250–2292, 2011

²³ Ofir Pele and Michael Werman. A linear time histogram metric for improved sift matching. In *European conference on computer vision*, pages 495–508. Springer, 2008

²⁴ Ofir Pele and Michael Werman. Fast and robust earth mover's distances. In *Computer vision, 2009 IEEE 12th international conference on*, pages 460–467. IEEE, 2009

²⁵ Justin Solomon, Raif Rustamov, Leonidas Guibas, and Adrian Butscher. Continuous-flow graph transportation distances. *arXiv preprint arXiv:1603.06927*, 2016

$$\delta_{p,q,G} = \frac{\sum_{p_u q_v} m_{p_u, q_v} d_{u,v}}{\sum_{p_u q_v} m_{p_u, q_v}},$$

where the m_{p_u, q_v} movements come from the M we found at the previous step. The differences between the methods cited before almost exclusively lie in the strategy to find the optimal M . The thing left to determine in the δ formula is the distance function $d_{u,v}$ between pairs of nodes. As mentioned previously, we choose this to be the length of the shortest path in G between u and v , or: $d_{u,v} = |P_{u,v}|$. This is zero if $u = v$.

There *could* be additional constraints to this optimized many-to-many distance. For instance, we could have the constraint that, while we are moving something from node u to node v via the edge that connects them, nothing else can pass through that edge. In practice, we are simulating an actual physical transportation system, in which edges and nodes have capacities. If we want to move two values at the same time through the same edge, we need to re-route one of them, because the edge – or the node – is occupied. How to find the optimal way to solve this problem is the realm of Multi-Agent Path Finding (MAPF)^{26,27}.

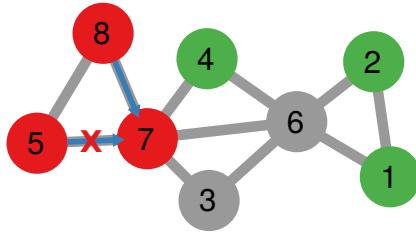


Figure 47.10 shows an example of this problem: we want to go from node 8 to node 2 and from node 5 to node 1. Unfortunately, the shortest paths for these two objectives both involve passing through node 7 at the same time. This cannot happen, since node 7 can host only a single walker at a time. Thus, either the walker in 8 or the walker in 5 needs to wait for a bit until node 7 is clear again.

In general MAPF, you have multiple robots occupying one node at a time and they each have a specific node as their intended destination²⁸. This is slightly different from our problem, where each weight in p can potentially reach any other destination in q . So one has to determine, before running MAPF, which $u \in p$ should go to which $v \in q$. One solution is to simply use the same strategy we used for single linkage in the non-optimized shortest path category: we look for the shortest path length $|P_{u,v}|$ carrying the largest possible weight $\min(p_u, q_v)$.

²⁶ Oded Goldreich. Finding the shortest move-sequence in the graph-generalized 15-puzzle is np-hard., 2011

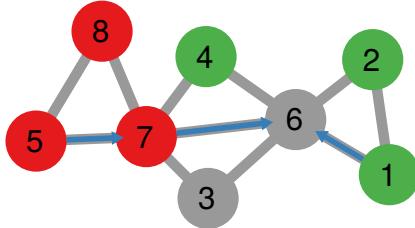
²⁷ Jingjin Yu and Daniela Rus. Pebble motion on graphs with rotations: Efficient feasibility tests and planning algorithms. In *Algorithmic Foundations of Robotics XI*, pages 729–746. Springer, 2015

Figure 47.10: Attempting to find a MAPF solution from red nodes to green nodes. Blue arrows show two attempted moves that cannot be executed at the same time.

²⁸ Klaus-Tycho Foerster, Linus Groner, Torsten Hoefer, Michael Koenig, Sascha Schmid, and Roger Wattenhofer. Multi-agent pathfinding with n agents on graphs with n vertices: Combinatorial classification and tight algorithmic bounds. In *International Conference on Algorithms and Complexity*, pages 247–259. Springer, 2017

Moreover, since in MAPF robots cannot be in the same node at the same time, you still have a problem. Say that we assigned a robot to go from u to v in our preprocessing. If $p_u \neq q_v$, then either u or v has some unallocated weight. Thus we would need to add at least a second robot that can either start in u or terminate in v . But this violates MAPF. The way we solve the issue is by running a sequence of MAPF sessions. In each session, we attempt to move all the weights that were left over during the previous session. We keep running smaller and smaller sessions until all weights have been allocated – which we can guarantee by normalizing either p or q so that they sum to the same value, as we did in the non-optimized solutions.

There are many algorithms to solve MAPF^{29,30,31,32,33,34,35,36,37,38,39}, each of them providing a different solution to NVD with our preprocessing strategy.



Another variant to OTP is pursuit-evasion games. In these games, we populate a space with a set of robots. Some robots are pursuers and they aim at capturing the other robots, the evaders. In discrete pursuit-evasion (DPE) we force the robots to move through the nodes and edges of a graph, rather than in a Euclidean space⁴⁰. Many algorithms have been proposed to model different strategies and constraints both on the pursuer and on the evader side.

One can see how it is possible to adapt DPE to solve our distance problem. First, we set the pursuers as p and the evaders as q . Then we run any DPE solving algorithm. Alternatively, both p and q are sets of pursuers and try to capture each other, with no evasion. Figure 47.11 is an example: here nodes 5 and 1 are trying to capture each other. Every time pursuers capture each other, the one carrying the $\min(p_u, q_v)$ weight disappears and the other one carries its own weight minus $\min(p_u, q_v)$. The amount of time/moves it takes for all weights to disappear is the distance between p and q . Since p and q sum to the same value – either by normalization or by the usual expansion strategy –, the system will terminate.

There are a number of solutions to DPE, satisfying a vast number of different constraints^{41,42,43,44,45,46}.

²⁹ Julio E Godoy, Ioannis Karamouzas, Stephen J Guy, and Maria Gini. Adaptive learning for multi-agent navigation. In *Int Conf on Autonomous Agents and Multiagent Systems*, pages 1577–1585. International Foundation for Autonomous Agents and Multiagent Systems, 2015

³⁰ Jamie Snape, Jur Van Den Berg, Stephen J Guy, and Dinesh Manocha. The hybrid reciprocal velocity obstacle. *IEEE Transactions on Robotics*, 27(4):696–706, 2011

³¹ Glenn Wagner and Howie Choset. Sub-dimensional expansion for multirobot path planning. *Artificial Intelligence*, 219:1–24, 2015

³² Andrew Dobson, Kiril Solovey, Rahul Shome, Dan Halperin, and Kostas E Bekris. Scalable asymptotically-optimal multi-robot motion planning. In *2017 International Symposium on Multi-Robot and Multi-Agent Systems (MRS)*, pages 120–127. IEEE, 2017

Figure 47.11: Attempting to find a pursue solution from red nodes to green nodes. Blue arrows show attempted moves.

³³ Hang Ma, TK Satish Kumar, and Sven Koenig. Multi-agent path finding with delay probabilities. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017

³⁴ Thayne T Walker, David M Chan, and Nathan R Sturtevant. Using hierarchical constraints to avoid conflicts in multi-agent pathfinding. In *Int Conf on Automated Planning and Scheduling*, 2017

³⁵ Konstantin Yakovlev and Anton Andreychuk. Any-angle pathfinding for multiple agents based on sipp algorithm. In *Twenty-Seventh International Conference on Automated Planning and Scheduling*, 2017

³⁶ Thayne T Walker, Nathan R Sturtevant, and Ariel Felner. Extended increasing cost tree search for non-unit cost domains. In *IJCAI*, pages 534–540, 2018

³⁷ Jiaoyang Li, Pavel Surynek, Ariel Felner, Hang Ma, TK Satish Kumar, and Sven Koenig. Multi-agent path finding for large agents. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 7627–7634, 2019

³⁸ Anton Andreychuk, Konstantin Yakovlev, Dor Atzmon, and Roni Sternr. Multi-agent pathfinding with continuous time. In *IJCAI*, volume 19, 2019

47.4 Graph Fourier Transform

There are other ways to solve the node vector distance problem in networks that do not start either from shortest paths nor from a generalization of Euclidean distances. They are generally linked to the spectrum of the graph, since the spectrum can be used to describe diffusion processes on the network, and the node vector distance is a type of diffusion process.

In the signal processing literature, a common scenario is one where the analyst has a battery of sensors, whose readings are correlated with each other. In order to extract the actual signal \hat{s} from the noisy and correlated signal data s , these relationships between sensor outputs have to be taken into account. The relationships can be modeled with a network G connecting related sensors. Then, the outputs are smoothed using the Graph Fourier Transform $\hat{s} = \Phi^T s^{47,48}$.

I explained what the Graph Fourier Transform is in Section 44.4. For the purposes of this section, suppose that our signal – the vector assigning each node to a value – is p . Then the corrected signal is equal to: $\hat{p} = \Phi^T p$ – remember that Φ is the matrix of L 's eigenvectors. With this transformation, \hat{p} tells us how much each of the eigenvectors contributes to p . In other words, we are changing our representation from the “spatial nodes” (p) to the “frequency modes” (\hat{p}). We can now weight the modes so that we take into account the topology of the graph. This is usually achieved by filtering the signal in the spectral domain, multiplying it with the diagonal matrix of the Laplacian's eigenvectors Λ .

Once we apply this transformation to both p and q , we have encoded G 's topology in the vectors. The Euclidean distance between them is the node vector distance that we are looking for:

$$\delta_{p,q,G} = \text{Euclidean}(p\Lambda\Phi^T, q\Lambda\Phi^T).$$

Note that this is not one, but a family of measures. One could replace the Euclidean distance with any other off-the-shelf measure (cosine, correlation, etc) to estimate the distance between the filtered p and q , because they already contain G 's topology in their values.

These approaches can be used to establish the distance between two different signals on a graph. However, this is but one of the applications of graph signal processing. Other scenarios include signal cleaning⁴⁹, frequency analysis⁵⁰, sampling⁵¹, interpolation⁵², and trend filtering⁵³, to cite a few. This also means that the transformation proposed here might not be the optimal one, and it is for sure not the only one.

³⁹ Minghua Liu, Hang Ma, Jiaoyang Li, and Sven Koenig. Task and path planning for multi-agent pickup and delivery. In *Int Conf on Autonomous Agents and MultiAgent Systems*, pages 1152–1160. IFAAMAS, 2019

⁴⁰ Torrence D Parsons. Pursuit-evasion in a graph. In *Theory and applications of graphs*, pages 426–441. Springer, 1978

⁴¹ Saeed Akhoondian Amiri, Lukasz Kaiser, Stephan Kreutzer, Roman Rabinovich, and Sebastian Siebertz. Graph searching games and width measures for directed graphs. In *LIPics-Leibniz International Proceedings in Informatics*, volume 30. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2015

⁴² Brian Alspach. Searching and sweeping graphs: a brief survey. *Le matematiche*, 59(1, 2):5–37, 2006

⁴³ Fedor V Fomin and Dimitrios M Thilikos. An annotated bibliography on guaranteed graph searching. *Theoretical computer science*, 399(3):236–245, 2008

⁴⁴ Flaminia L Lucio. Intruder capture in sierpinski graphs. In *FUN*, pages 249–261. Springer, 2007

⁴⁵ Victor Gabriel Lopez Mejia, Frank L Lewis, Yan Wan, Edgar N Sanchez, and Lingling Fan. Solutions for multiagent pursuit-evasion games on communication graphs: Finite-time capture and asymptotic behaviors. *IEEE Transactions on Automatic Control*, 2019

⁴⁶ Nicholas M Stiffler and Jason M O'Kane. Pursuit-evasion with fixed beams. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4251–4258. IEEE, 2016

⁴⁷ David K Hammond, Pierre Vandergheynst, and Rémi Gribonval. Wavelets on graphs via spectral graph theory. *Applied and Computational Harmonic Analysis*, 30(2):129–150, 2011

⁴⁸ David I Shuman, Benjamin Ricaud, and Pierre Vandergheynst. Vertex-frequency analysis on graphs. *Applied and Computational Harmonic Analysis*, 40(2):260–291, 2016

⁴⁹ Patric Hagmann, Leila Cammoun, Xavier Gigandet, Reto Meuli, Christopher J Honey, Van J Wedeen, and Olaf Sporns. Mapping the structural core of human cerebral cortex. *PLoS biology*, 6(7):e159, 2008

⁵⁰ Aliaksei Sandryhaila and Jose MF Moura. Discrete signal processing on graphs: Frequency analysis. *IEEE Trans. Signal Processing*, 62(12):3042–3054, 2014

47.5 Statistics on a Network Space

While reading this chapter, you might have been wondering something. If we have defined an analogue to the Euclidean distance when the network represents the space in which our observations live, can we do the same thing for other statistical measures? The answer turns out to be yes. I know of two works in this direction. One redefines the concept of variance for node vectors, while the other works with a generalization of normalized co-variance – also known as the Pearson linear correlation.

Network Variance

We introduced the concept of variance all the way back in Section 3.1. We said it quantifies how much we expect any random observation to differ from the average (squared). One way to write this formula is⁵⁴:

$$\text{var}(x) = \frac{1}{2} \sum_{u,v} x_u x_v d_{uv}^2,$$

assuming u and v are all possible pairs of observations in our distribution x – and so x_u is the value of x for observation u . Here, d_{uv} is the Euclidean distance between observations u and v . But what if u and v are nodes in a graph G ? Then their distance should not be the Euclidean distance, but a graph distance. So, if for d_{uv} we take a graph distance, we have just defined a notion of network variance.

Luckily, it is very easy to define a graph distance: d_{uv} could simply be the shortest path between u and v . A better option, though, is to use the effective resistance between the two nodes (Section 11.4). This is because the effective resistance is a proper metric and it uses random walks rather than shortest paths. The latter is desirable, because it makes effective resistance less sensitive to small changes in the structure – such as the addition or the removal of an edge – which can dramatically change the shortest path distance.

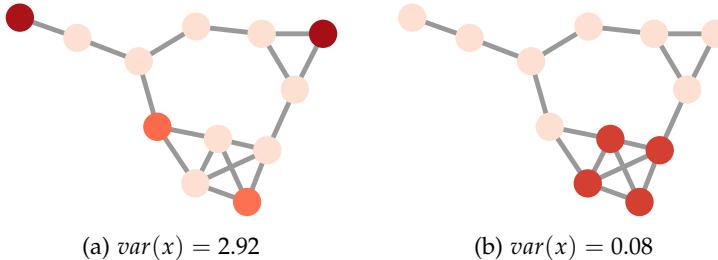


Figure 47.12 shows an example, which should help you have a visual understanding of what it means to calculate a network variance. We have the same graph, but two different variables defined on its nodes. Intuitively, the variable I show in Figure 47.12(a) has

⁵¹ Aamir Anis, Akshay Gadde, and Antonio Ortega. Towards a sampling theorem for signals on arbitrary graphs. In *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*, pages 3864–3868. IEEE, 2014

⁵² Sunil K Narang, Akshay Gadde, Eduard Sanou, and Antonio Ortega. Localized iterative methods for interpolation in graph structured data. In *Global Conference on Signal and Information Processing (GlobalSIP), 2013 IEEE*, pages 491–494. IEEE, 2013

⁵³ Yu-Xiang Wang, James Sharpnack, Alex Smola, and Ryan J Tibshirani. Trend filtering on graphs. *Journal of Machine Learning Research*, 17(105):1–41, 2016b

⁵⁴ Karel Devriendt, Samuel Martin-Gutierrez, and Renaud Lambiotte. Variance and covariance of distributions on graphs. *SIAM Review*, 64(2):343–359, 2022

Figure 47.12: Two different attributes on the same network. The darker the node, the higher the value on that node for a given attribute.

a high variance, because it is scattered around the periphery of the graph. On the other hand, the variable I show in Figure 47.12(b) has a low variance, because it clusters in a small subpart of the graph. The network variance estimated via effective resistance confirms this intuition, giving a high value for Figure 47.12(a) and a low one for Figure 47.12(b) – which I report in the subcaptions.

Network Correlation

We can use a similar trick to extend the familiar Pearson correlation coefficient (Section 3.4) to variables defined on a network⁵⁵. Let's suppose that we have two vectors, x and y , each assigning a value to a node in the graph G . Each node v in G contributes something to the Pearson correlation, and that something is the relation between its corresponding values x_v and y_v . If, across all v s, every time we have a high x_v value we also have a high y_v value, and vice versa, then the Pearson correlation is positive and high.

But, wait, if we are in a graph G , why should we only weigh v 's values to determine the correlation coefficient? The basic assumption that is at the basis of literally everything in network science is that the network structure influences the behavior of the nodes – and vice versa. With this assumption we must conclude that v 's neighbors influence the x_v value, and therefore they should play a role in the correlation.

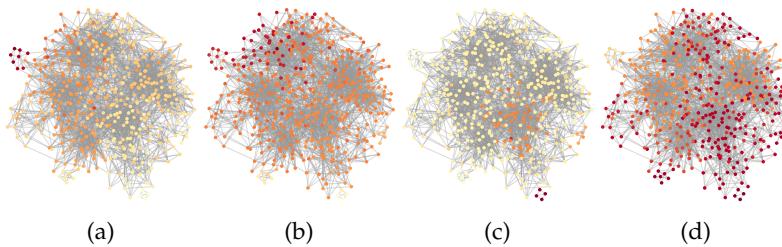


Figure 47.13 gives you a visual understanding of what a network correlation means for a node vector. For a given node vector v in Figure 47.13(a), I show you node vectors with positive (Figure 47.13(b)), no (Figure 47.13(c)), and negative (Figure 47.13(d)) network correlations.

The neighbors' role should be inversely proportional to their distance from v : farther nodes exert a weaker influence – indirectly through their neighbors, and neighbors of neighbors and so on. This distance is usually estimated via – again – the effective resistance matrix as it was the case for network variance. In this case one should not use the shortest path lengths, because they are not a proper metric on a network and they could lead to mathematical errors⁵⁶.

⁵⁵ Michele Coscia. Pearson correlations on complex networks. *Journal of Complex Networks*, 9(6):cnab036, 2021

Figure 47.13: Four different node vectors on the same network. The node's color represents the value in x , from dark red (high) to bright yellow (low). (a) A node vector v . (b) A node vector with a positive network correlation with v . (c) A node vector with no correlation with v . (d) A node vector with an anti-correlation with v .

⁵⁶ Michele Coscia and Karel Devriendt. Pearson correlations on networks: Corrigendum. *arXiv preprint arXiv:2402.09489*, 2024

47.6 Summary

1. The node vector distance problem is the quest for finding a way to estimate a network distance between two vectors describing the degree of occupancy of the nodes in the network. If at time t I occupy nodes 1, 2, and at time $t + 1$ I occupy nodes 3, 4, 5, how much did I move in the network?
2. The Euclidean distance can be used to estimate distances between vectors in a homogeneous space. Thus, a family of solution focuses on “warping” the space so that it is described by the topology of the network. At that point, you can use the Euclidean distance on such a warped space.
3. Another family of solutions uses shortest paths: you calculate all shortest paths between all nodes of origin and destination, and you aggregate the results somehow. Alternatively, you can try to find only those shortest paths minimizing the resulting distance.
4. You can add several constraints to the optimized shortest path strategy. For instance, you could model a real infrastructure network: nodes and edges have finite capacity.
5. Finally, you can use signal cleaning techniques. You can see your network as describing sets of sensors that return correlated results. Thus, two “signals” are far apart if they are reported by uncorrelated sensors, which are not connected to each other.
6. You are not limited to calculating distances between node vectors. One can also extend the basic concepts of variance and correlation to the case in which the vector lives in the complex space defined by a graph.

47.7 Exercises

1. Calculate the distance between the node vectors in <http://www.networkatlas.eu/exercises/47/1/vector1.txt> and <http://www.networkatlas.eu/exercises/47/1/vector2.txt> over the network in <http://www.networkatlas.eu/exercises/47/1/data.txt>, using the Laplacian approach. The vector files have two columns: the first column is the id of the node, the second column is the corresponding value in the vector. Normalize the vectors so that they both sum to one.
2. Calculate the distance using the same data as the previous question, this time with the average linkage shortest path approach. Normalize the vectors so that they both sum to one.

3. Calculate the distance using the same vectors as the previous questions, this time on the <http://www.networkatlas.eu/exercises/47/3/data.txt> network, with both the average linkage shortest path and the Laplacian approaches. Are these vectors closer or farther in this network than in the previous one?
4. Calculate the variances of the vectors used in the previous exercises on both networks used in the previous exercises.

48

Topological Distances

In the previous chapter we learned how to estimate the distance between two vectors describing the occupancy of sets of nodes in the same network. In that problem, you get two vectors with $|V|$ entries, and you calculate their distance on the *same* network topology. We called this “node vector distance”, a certain type of “network distance”. There are other ways one could interpret the term “network distance”. I group them all in this chapter.

Specifically I talk about:

- Network similarity (Section 48.1): how to tell if two graphs G_1 and G_2 have a similar topology;
- Network alignment (Section 48.2): finding nodes in G_1 that are similar to nodes in G_2 , so that we could couple them and consider G_1 and G_2 as two layers of a multilayer network;
- Network fusion (Section 48.3): given multiple observations of a network, combine them to create a summary that is the most similar to all observations.

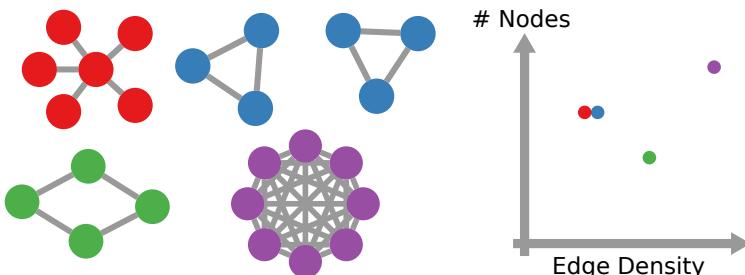
48.1 Network Similarity

By far, the most common and popular way to intend the term “network distance” is as the opposite of the similarity between two networks. The term “network similarity” is, unfortunately, rather ambiguous, and you might find papers dealing with very different problems but using the same terminology. For instance, one could intend “network similarity” as a measure of how similar two nodes are (see Section 15.2). Or one could be talking about “similarity networks”, which are ways to express the similarities between different entities by connecting the ones that are the most similar to each other – something you might do via bipartite projections (Chapter 26).

Here, we focus on a different problem. The idea here is simple: we have two networks G_1 and G_2 and we want to know how similar the two are. Namely, how easy it is to mistake G_1 as G_2 by looking at their edges. There are many ways to do this, and I'll try to give a general overview.

Most of the applications of these techniques are in biology and chemistry. The idea is to compare networks describing specific pathways. However, there are also more peculiar applications, for instance in malware detection¹ and image recognition². I am going to include the approaches used mostly for practical problems in computer science. However, there are many more distance measures that have a more distinctively “mathy” flavor. The bible for this kind of things is for sure the Encyclopedia of Distances³. Some examples of distance measures you can find there are the Chartrand-Kubicki-Schultz distance⁴, the rectangle distance⁵, and many more others.

At a practical level, all the methods that follow have one thing in common. Comparing two networks using a handful of summary statistics means to define a low-dimensional space in which every network is a point. You can visualize that as a scatter plot: Figure 48.1 makes a super simple one where I decide to classify networks by their number of nodes and edge density. However, networks are a high-dimensional object, and the summary statistics commonly used in network science are usually non orthogonal – differently from what you’d get from, for instance, Principal Component Analysis (Section 5.6) –: in my case, from Section 12.1 you know that the number of nodes is usually negatively correlated with edge density.



The main issue is that we still don’t know which set of network statistics is sufficient to cover the space of all possible networks. Whatever dimensions you use to organize your networks will collapse many – possibly dissimilar – networks into the same place in your scatter plot. This happens in Figure 48.1, where a star (in red) is confused with a set of unconnected cliques (in blue). This is not necessarily a bad thing! If the summary statistics you chose are meaningful to you in some fundamental way, this is a feature. However, if you’re hunting for “universal” patterns, this approach could mislead

¹ Neha Runwal, Richard M Low, and Mark Stamp. Opcode graph similarity and metamorphic detection. *Journal in computer virology*, 8(1-2):37–52, 2012

² Sébastien Sorlin and Christine Solnon. Reactive tabu search for measuring graph similarity. In *International Workshop on Graph-Based Representations in Pattern Recognition*, pages 172–182. Springer, 2005

³ Michel Marie Deza and Elena Deza. Encyclopedia of distances. In *Encyclopedia of distances*, pages 1–583. Springer, 2009

⁴ Gary Chartrand, Grzegorz Kubicki, and Michelle Schultz. Graph similarity and distance in graphs. *Aequationes Mathematicae*, 55(1-2):129–145, 1998

⁵ Christian Borgs, Jennifer Chayes, László Lovász, Vera T Sós, Balázs Szegedy, and Katalin Vesztergombi. Graph limits and parameter testing. In *Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*, pages 261–270, 2006

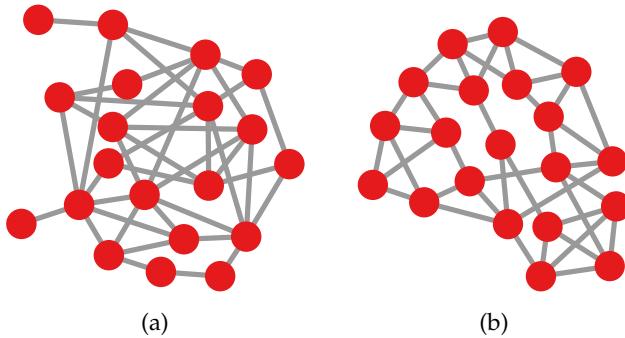
Figure 48.1: On the left we have four graphs, each identified by the color of its nodes. On the right, I make a two dimensional projection by recording each graph’s node count (y axis) and edge density (x axis). The similarity between two graphs is the inverse of their distance in this space.

you.

Global Property Comparison

The most basic way to tell whether two networks are similar is by looking at their global properties⁶. If two networks have the same degree distribution, the same average path length, the same clustering coefficient, the same average degree, and so on... Well, doesn't that mean that these two networks are.... the same?

This is a seducing option because, as we'll see, estimating the similarity between two networks by looking at their topology is computationally very hard. It is related to the graph isomorphism problem, and we saw that graph isomorphism is a though nut to crack in Section 41.3. On the other hand, estimating many global properties is trivial and instantaneous in many cases, and well studied and optimized in others.



⁶ Geng Li, Murat Semerci, Bulent Yener, and Mohammed J Zaki. Graph classification via topological and label attributes. In *Proceedings of the 9th international workshop on mining and learning with graphs (MLG), San Diego, USA*, volume 2, 2011

Figure 48.2: Two graphs of which we want to estimate the similarity.

Of course, you need to be extremely careful in considering two things. First, what are the global properties you're looking at? Second, how do you aggregate the differences between these properties to end up with a single measure of similarity? These are important questions, because you might end up considering as similar two networks that are very different. Consider Figures 48.2(a) and 48.2(b). The two networks have a lot in common: same number of nodes and edges (thus the average degree and density are the same as well). They have almost identical degree distributions, approximated by a Gaussian. They have the same diameter and a very similar average path length (2.1 vs 2.4). Up until now, you'd consider them practically equivalent. And yet, they're still relatively different, as they were generated using two very different processes. Figure 48.2(a) is a $G_{n,m}$ random graph, while Figure 48.2(b) is a small-world graph. The crucial factor I forgot to check is the clustering coefficient, which is low for $G_{n,m}$ graphs (0.17 in this instance) and high for small-world networks (0.41 here).

Pairwise Node Similarity

A common approach is the estimation of all possible combinations of node similarities. This is a relatively popular way to attack the problem, which underlies many other techniques. The reason is that it is a natural way to think about network similarity: two networks are similar if they have the same nodes and these nodes connect to the same neighbors. Estimating all the pairwise node similarities is the first step to tell which nodes are the same. More often than not, that is the end goal of estimating network similarity: we might be less interested in how similar two networks are and more in which nodes from one network are the same nodes in the other. This is the problem of network alignment and we'll see it more in depth in Section 48.2.

We have seen dozens of ways to tell how similar two nodes are, both in Section 15.2 and in Chapter 23. The general idea is to try and estimate the structural equivalence of all nodes in the two graphs. Then you can either average all the node-node similarities you calculated, or find the best way to map nodes: for each u_1 in G_1 you find the best corresponding u_2 in G_2 such that, when you mapped all nodes, the average similarity is maximized.

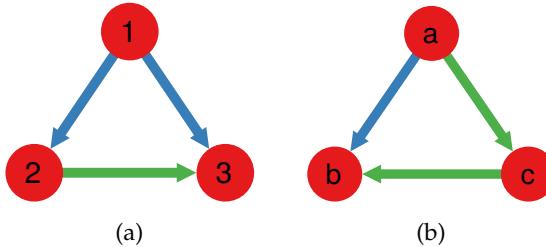


Figure 48.3: Two graphs of which we want to estimate the similarity. Edge color represents its type.

Just to get a better intuition on how this might work, consider Figure 48.3. We can estimate the networks' similarity by looking at the structural equivalence of their nodes. Nodes 1 and a are very similar: they both have outdegree of two and they point to the same neighborhood – two nodes connected by a single green edge. The only difference is in the label of one of their edges. Nodes 2 and c are also of relatively high similarity, given their equal in- and out-degree with again the sole difference of the edge color. Nodes 3 and b are, on the other hand, almost structurally identical, with the sole difference being not between them, but between their neighbors. We can conclude, then, that the two graphs are extremely similar, since we just made a node mapping among very similar nodes.

This is a simplification of real approaches^{7,8}. The hard part is defining an efficient technique to find such mappings.

⁷ Sergey Melnik, Hector Garcia-Molina, and Erhard Rahm. Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In *Proceedings 18th International Conference on Data Engineering*, pages 117–128. IEEE, 2002

⁸ Laura A Zager and George C Verghese. Graph similarity scoring and matching. *Applied mathematics letters*, 21(1):86–94, 2008

Graph Edit Distance

The strictest possible criterion to establish the similarity between two networks is by solving the graph isomorphism problem. If two graphs are literally the same, their similarity is equal to one. Of course, the graph isomorphism test is binary, thus it is too strict. A single edge difference would net you a zero similarity. We can transform this test into something more useful by counting the number of edge differences between the two graphs. This is akin to define a “graph edit distance”.

The edit distance between objects a and b is an estimation of the number of edits you need to make on a in order to transform it into b . Perhaps the most known and used edit distance is the string edit distance, of which the most famous is the Levenshtein distance⁹: this tells you how far apart two strings are. Variants of it are widely used, for instance, by search engines and word processors: when you mistype a word, the software will look up what are the properly spelled words that are at the smallest edit distance from what you typed, and it will suggest them to you. This works well because, usually, you won’t make more than one or two mistakes in typing something – unless you’re me and you’re trying to retype “Levenshtein” from memory.

String and graph edit distances work with the same principles^{10,11}. In strings you are allowed to perform three operations: character insertion, deletion, and replacement. In graphs you have the same three operations, but you can apply them to either nodes or edges, for a total of six operations. Your nodes and edges might have labels, so you want to be able to flip the label values as well.

⁹ Vladimir I Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, volume 10, pages 707–710, 1966

¹⁰ Xinbo Gao, Bing Xiao, Dacheng Tao, and Xuelong Li. A survey of graph edit distance. *Pattern Analysis and applications*, 13(1):113–129, 2010

¹¹ Kaspar Riesen and Horst Bunke. Approximate graph edit distance computation by means of bipartite graph matching. *Image and Vision computing*, 27(7):950–959, 2009

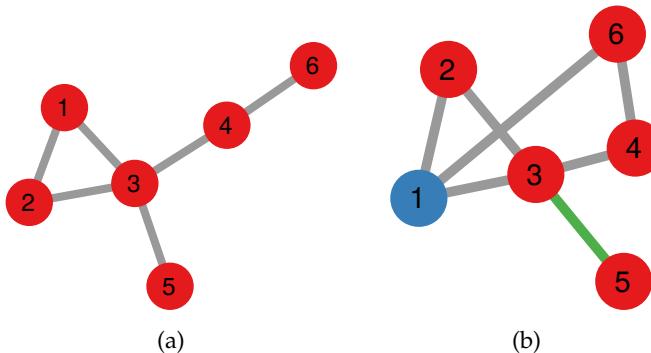


Figure 48.4: Two graphs of which we want to estimate the similarity. Node and edge color represents their type.

Figure 48.4 can help you to visualize the process. Here, we want to know how many operations we need to go from the graph in Figure 48.4(a) to the graph in Figure 48.4(b). Starting from node 1, we need to change its label (from red to blue) and to add the edge connecting it to node 6. Node 2 is fine, but node 3 needs to replace its edge to

node 5 with one labeled in green. There are no more edits we need to do, so the distance between the two graphs is three.

Of course, the hard part of graph edit distance is finding the minimum set of edits, so there are a bunch of ways to go about it, ranging from Expectation Maximization to Self-Organizing Maps, to subgraph isomorphism¹². Special network types deserve special approaches, for instance in the case of Bayesian networks¹³ (see Section 6.4) and trees^{14,15}.

The prototypical graph edit distance metric¹⁶ is relatively simple to understand. It is based on the maximum common subgraph. Given two graphs G_1 and G_2 , first you find the largest common subgraph G_s : the largest collection of nodes and edges that is isomorphic in both graphs. Then, the distance between G_1 and G_2 is simply the number of nodes and edges that remain outside G_s :

$$\delta_{G_1, G_2} = |E_1 - E_s| + |E_2 - E_s| + ||V_1| - |V_2||,$$

with V_x and E_x being the set of nodes and edges of graph G_x . This is actually a metric, as it respects the triangle inequality. An evolution of this approach tries to find the maximum common edge subgraph¹⁷, which is found in the line graph representation of G .

The problem gets significantly easier when the networks are aligned. Two networks are aligned if we have a known node correspondence between the two. This means that we know that node u in one network is the same as node v in the other. How to align two networks is an interesting problem in and of itself, and we're going to look at it in Section 48.2. For now, we just take for granted that the two networks we're comparing are already aligned.

In this case, we don't have to go looking for maximum subgraphs. We can just iterate over all the nodes and edges in the two networks and note down every time we find an inconsistency: a node or an edge that is present in one network and absent in the other. Simply counting won't do much good, though, because some differences should count more than others, if they are significantly affecting the local or global properties of the network. Consider Figure 48.5: both Figure 48.5(b) and Figure 48.5(c) are just one edge away from Figure 48.5(a). However, since Figure 48.5(b) breaks down in multiple connected components, its difference should be counted as higher.

There are a few strategies to estimate these differences. Deltacon is based on some sort of node affinity estimation¹⁸. One could also do vertex rank comparison¹⁹: if the most important nodes in two networks are the same, then the networks must be similar, to some extent. The same authors propose other ways to estimate network similarity, for instance via shingling: reducing the networks to sequences and then applying a sequence comparing algorithm.

¹² Horst Bunke. On a relation between graph edit distance and maximum common subgraph. *Pattern Recognition Letters*, 18(8):689–694, 1997

¹³ Richard Myers, RC Wilson, and Edwin R Hancock. Bayesian graph edit distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(6):628–635, 2000

¹⁴ Davi de Castro Reis, Paulo Braz Golgher, Altigran Soares Silva, and Alberto F Laender. Automatic web news extraction using tree edit distance. In *Proceedings of the 13th international conference on World Wide Web*, pages 502–511, 2004

¹⁵ Mateusz Pawlik and Nikolaus Augsten. Rted: a robust algorithm for the tree edit distance. *arXiv preprint arXiv:1201.0230*, 2011

¹⁶ Vladimír Baláž, Jaroslav Koča, Vladimír Kvasnička, and Milan Sekanina. A metric for graphs. *Časopis pro pěstování matematiky*, 111(4):431–433, 1986

¹⁷ John W Raymond, Eleanor J Gardiner, and Peter Willett. Rascal: Calculation of graph similarity using maximum common edge subgraphs. *The Computer Journal*, 45(6):631–644, 2002

¹⁸ Danai Koutra, Joshua T Vogelstein, and Christos Faloutsos. Deltacon: A principled massive-graph similarity function. In *Proceedings of the 2013 SIAM International Conference on Data Mining*, pages 162–170. SIAM, 2013

¹⁹ Panagiotis Papadimitriou, Ali Dasdan, and Hector Garcia-Molina. Web graph similarity for anomaly detection. *Journal of Internet Services and Applications*, 1(1):19–30, 2010

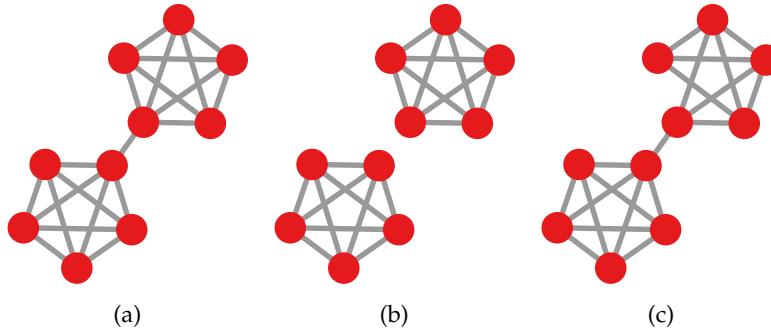


Figure 48.5: Three graphs of which we want to estimate the similarity. Note how (b) misses the edge connecting the cliques, and (c) misses an edge inside the top clique.

These latter approaches are specialized to find changes in the same time-evolving network.

The big caveat for using graph edit distances is that they only work for specific data generating processes. For instance, remember the $G_{n,p}$ uniform random graphs from Chapter 16? Two $G_{n,p}$ graphs with the same n and (low) p are similar, in the sense that they are realizations of the same process. However, since edges are independent and the graphs are sparse, they will have almost no edge in common. As a consequence, their edit distance is large! So what you're looking for when using edit distances is for a generating process that has strong dependencies between edges: the fact that two nodes are connected implies the presence/absence of other edges in their neighborhood. You should discard graph edit distance measures as soon as you think that the edges inside your networks are independent from each other.

Substructure Comparison

Substructure comparison^{20,21} is similar to graph edit distance. In this class of methods, you describe the network as a dictionary of motifs and how they connect to each other. Usually, you'd find the motifs by applying frequent subgraph mining (Chapter 41). In practice, graph edit distance is equivalent to a simple substructure comparison, where the only substructure you're focusing on is the edge. There is not much to say about this class, given its similarity with the previous one: the same considerations and warnings that applied there also apply here.

When it comes to applications of substructure similarity, the classical scenario is estimating compound similarities at the molecular level in a biological database²². But there are more fun scenarios, such as an analysis of Chinese recipes²³.

²⁰ Xifeng Yan, Philip S Yu, and Jiawei Han. Substructure similarity search in graph databases. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 766–777, 2005.

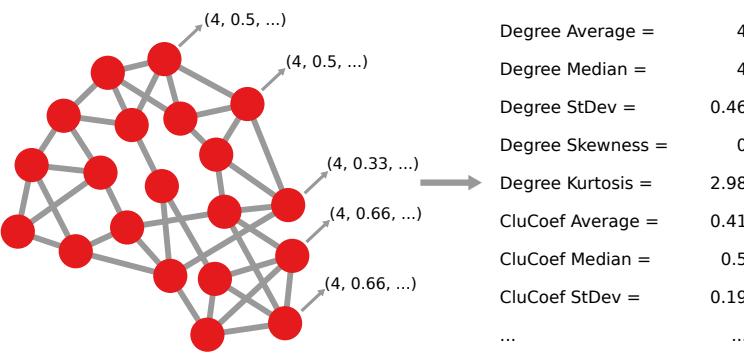
²¹ Haichuan Shang, Xuemin Lin, Ying Zhang, Jeffrey Xu Yu, and Wei Wang. Connected substructure similarity search. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 903–914, 2010.

²² Thomas R Hagadone. Molecular substructure similarity searching: efficient retrieval in two-dimensional structure databases. *Journal of chemical information and computer sciences*, 32(5):515–521, 1992.

²³ Liping Wang, Qing Li, Na Li, Guozhu Dong, and Yu Yang. Substructure similarity measurement in chinese recipes. In *Proceedings of the 17th international conference on World Wide Web*, pages 979–988, 2008a.

Holistic Approaches

In the holistic category I group a series of approaches that are a mixture of the four previous strategies. Meaning that they use parts of all global properties, node similarities, and edit distance, to build a more general similarity measure for networks. The idea is to build a “signature vector”: a numerical vector that describes the relevant aspects of the topology of G . Then, the similarity between two graphs is simply the similarity between their signature vectors. In practice, one could think this class to include a sort of “graph embeddings” (Chapter 42).



NetSimile²⁴ is one of the many algorithms in this class. I represent its workflow in Figure 48.6. First, NetSimile calculates seven features for each node of the graph: degree, local clustering coefficient, average neighbor degree, average neighbor local clustering coefficient, number of edges among neighbors, etc. Then, these features are aggregated across nodes, i.e. NetSimile calculates their summary statistics like average, standard deviation, etc. This is the signature vector of the graph, which can now be used to compare G with any other graph. Any distance measure discussed so far in the book – cosine, Euclidean, ... – can be used to perform the comparison. The authors focus specifically on the Camberra distance²⁵.

Similar approaches are graph hashes²⁶, designed for optimizing graph similarity searches in a graph database possibly containing thousands of graphs; and approaches that are more rooted in social theories²⁷. The latter case is an evolution of NetSimile. Rather than including a laundry list of all the measures we think we can use to compare graphs, we pick the ones that are theoretically motivated. We define which are the criteria of similarity based on different theories, and we discard the rest. The objective is to be able to better interpret the similarity scores.

A close cousin of holistic approaches is the one of graph kernels^{28,29,30}. Just like in NetSimile and in graph embeddings, a graph

Figure 48.6: The workflow of NetSimile. The rightmost column of number is the beginning of the graph’s signature vector.

²⁴ Michele Berlingerio, Danai Koutra, Tina Eliassi-Rad, and Christos Faloutsos. Netsimile: A scalable approach to size-independent network similarity. *arXiv preprint arXiv:1209.2684*, 2012

²⁵ Godfrey N Lance and William T Williams. Computer programs for hierarchical polythetic classification (“similarity analyses”). *The Computer Journal*, 9(1):60–64, 1966

²⁶ Xiaohong Wang, Jun Huan, Aaron Smalter, and Gerald H Lushington. G-hash: towards fast kernel-based similarity search in large graph databases. In *Graph Data Management: Techniques and Applications*, pages 176–213. IGI Global, 2012

²⁷ Michele Berlingerio, Danai Koutra, Tina Eliassi-Rad, and Christos Faloutsos. Network similarity via multiple social theories. In *Proceedings of the 2013 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, pages 1439–1440, 2013b

²⁸ Thomas Gärtner, Peter Flach, and Stefan Wrobel. On graph kernels: Hardness results and efficient alternatives. In *Learning theory and kernel machines*, pages 129–143. Springer, 2003

²⁹ SVN Vishwanathan, Karsten M Borgwardt, Nicol N Schraudolph, et al. Fast computation of graph kernels. In *NIPS*, volume 19, pages 131–138, 2006

³⁰ U Kang, Hanghang Tong, and Jimeng Sun. Fast random walk graph kernel. In *Proceedings of the 2012 SIAM international conference on data mining*, pages 828–838. SIAM, 2012

kernel is the reduction of a complex high-dimensional graph into a vector of numbers. These vectors are then fed to a machine learning algorithm that is able to learn the shape of the space in which these vectors live and thus the similarity between them. Just like with many graph embeddings techniques, these kernel are usually created by means of some sort of random walk process.

Information Theory

A radically different approach works directly with the adjacency matrix of a graph. The idea here is to generalize the Kullback-Leibler divergence (KL-divergence) so that it can be applied to determining the distance between two graphs. The KL-divergence is a cornerstone of information theory and linked with the concept of information entropy – see Section 3.5 for a refresher.

The KL-divergence is also known as “relative entropy”. From Section 3.5, you learned that the information entropy of a vector X is the number of bits per element you need to encode it. Now, of course when you try to encode a vector, you try to be as smart as possible. You create a codebook that is specialized to encode that particular vector. If there is an element that appears much more often than the others, you will give it a short code: you will have to use it more often and, if it is shorter, every time you use it you will save bits. This is the strategy used by Infomap to solve community discovery – see Section 35.2.

Now suppose you have another vector, Y . You want to know how similar Y is to X . One thing you could do is to encode Y using the code book you optimized to encode X . If $X = Y$, then the codebook is as good encoding X as it is encoding Y : you need no extra bits. As soon as there are differences between X and Y , you will start needing extra bits to encode Y , because X ’s codebook is not perfect for Y any more. The KL-divergence boils down to the number of extra bits you need to encode Y using X ’s codebook.

| X | X | Y | Y |
|-----|--------|-----|----------|
| ■ | ■ = 0 | ■ | ■ = 0 |
| ■ | 0 | ■ | 11 bits |
| ■ | 0 | ■ | 6 values |
| ■ | ■ = 10 | ■ | ■ = 10 |
| ■ | 10 | ■ | 10 |
| ■ | ■ = 10 | ■ | = 9 / 6 |
| ■ | 10 | ■ | = 11 / 6 |
| ■ | ■ = 11 | ■ | ■ = 11 |
| ■ | 11 | ■ | 11 |
| | | | |
| (a) | | (b) | |

Figure 48.7: An example of the spirit of KL-divergence. The code we use for X (a) requires additional bits to encode Y (b).

Figure 48.7 presents a rough outline of the idea behind the KL-divergence – simplified to help intuition. The X vector in Figure 48.7(a) requires 1.5 bits per element. Using its codebook to encode Y

in Figure 48.7(b) increases the requirement to 11 total bits instead of the original 9.

In its original formulation, the KL-divergence is defined for pairs of vectors. However, one can expand it to allow it to consider different inputs³¹. One can say that entries in the vectors are dependent on each other. Thus, if you consider a graph as a series of $|V|$ variables, one per node, you can express the pairwise dependencies as the edges of the graph. This approach has applications in chemistry³².

Another way of comparing networks by means of analyzing their adjacency matrices comes from comparing the eigenvectors of their Laplacians³³. Similar networks will experience similar spreading patterns, which are reflected in their spectra.

48.2 Network Alignment

Earlier in the chapter, I mentioned what aligned networks are: two networks are aligned if we have a node to node mapping, i.e. for each node in network G_1 we have a corresponding node in G_2 representing the same entity. Many networks are naturally aligned. The most typical case of aligned networks are time-evolving networks. Two snapshots of a structure are simply two different networks: since we have the same ids on the nodes, we have the alignment for free. Another example could be networks describing brain scans: we divide the brain in different areas for all individuals, and these areas might interact differently between individuals. The areas are the nodes, thus their identities are known, while the interactions are the edges, which might change. In general, any multilayer network (Section 7.2) could be seen as a collection of aligned networks: each layer is a network and the inter layer couplings are the mappings from one layer to another.

However, you might not be as lucky as in the case of evolving networks: sometimes you have two observations that you think you should be able to align, but you actually do not have neither consistent node ids, nor a reliable node mapping. For instance, you might have collected a bunch of data from different social media. You know that people have profiles in different platforms, but these platforms will use different and mutually incompatible identifiers. Thus, you will need to figure out who is who in all the networks you collected. This is the network alignment problem³⁴. A classical application of network alignment is the attempt to map the protein-protein interaction of different organisms³⁵, discovering that many biological pathways are preserved across species.

Figure 48.8 shows an example. Given two graphs as input with V_1 and V_2 as their node sets, you want to produce a $|V_1| \times |V_2|$ matrix

³¹ David J Galas, Gregory Dewey, James Kunert-Graf, and Nikita A Sakhnenko. Expansion of the kullback-leibler divergence, and a new class of information metrics. *Axioms*, 6(2):8, 2017

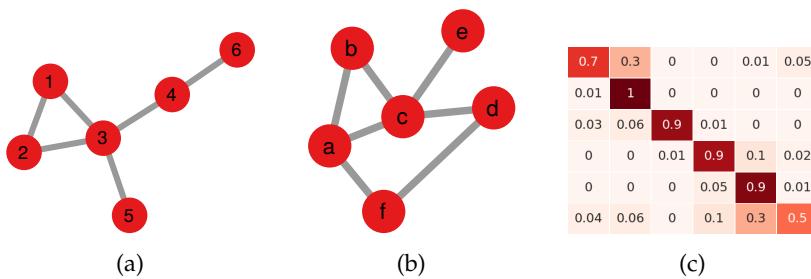
³² Christopher L McClendon, Lan Hua, Gabriela Barreiro, and Matthew P Jacobson. Comparing conformational ensembles using the kullback-leibler divergence expansion. *Journal of chemical theory and computation*, 8(6):2115–2126, 2012

³³ Anirban Banerjee. Structural distance and evolutionary relationship of networks. *Biosystems*, 107(3):186–196, 2012

³⁴ Huynh Thanh Trung, Nguyen Thanh Toan, Tong Van Vinh, Hoang Thanh Dat, Duong Chi Thang, Nguyen Quoc Viet Hung, and Abdul Sattar. A comparative study on network alignment techniques. *Expert Systems with Applications*, 140:112883, 2020

³⁵ Brian P Kelley, Roded Sharan, Richard M Karp, Taylor Sittler, David E Root, Brent R Stockwell, and Trey Ideker. Conserved pathways within bacteria and yeast as revealed by global protein network alignment. *Proceedings of the National Academy of Sciences*, 100(20):11394–11399, 2003

telling you the probability of each node from the first graph to be the node in the second graph. The way this matrix is built can rely on any structural similarity measure, we saw a few in different parts of this book. Then, the idea is to pick the cells in this matrix so that the sum of the scores is maximized and, at the same time, we match as many nodes as possible³⁶. If $|V_1| \neq |V_2|$ you will have to face a choice: either you do not map some nodes or you allow nodes from one network to map to multiple nodes in the other. This common approach can be extended, for instance, by calculating multiple versions of this matrix using different measures and then seeking a consensus matrix which is a combination of all the similarity measures.



³⁶ Oleksii Kuchaiev and Nataša Pržulj. Integrative network alignment reveals large regions of global network similarity in yeast and human. *Bioinformatics*, 27(10):1390–1396, 2011

Figure 48.8: Two graphs of which we want to discover the alignment. (c) assigns to each node pair from (a) and (b) an alignment probability.

One could also find just a few node mappings with extremely high confidence and then expand from that seed³⁷, assuming that the neighborhoods around these high confidence nodes should look alike. Of course, a large portion of network alignment solutions rely on solving the maximum common subgraph problem: if you find isomorphic subgraphs in both networks, chances are that the nodes inside these subgraphs are the same, and thus should be aligned to each other³⁸. Other approaches rely on the fact that isomorphic graphs have the same spectrum, thus similar values in the eigenvectors of the Laplacian imply that the nodes are relatively similar³⁹.

Another approach uses a dictionary of networks motifs. Each node is described by counting the number of motifs it is part of. We can then describe the node as a numerical count vector. Two nodes with similar vectors are similar⁴⁰. This approach has to solve the graph isomorphism problem as well, but it needs to do so only for small graph motifs rather than for – supposedly – large common subgraphs. This way, it can be more efficient.

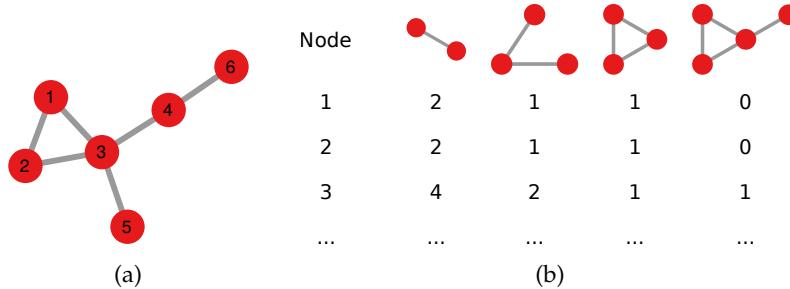
Figure 48.9 shows an example: here I choose a relatively small set of motifs, which generate a short vector. However, one could define as many motifs as they are relevant for a specific application, and obtain much more precise vectors describing the nodes. A final approach I mention is MAGNA, which uses a genetic algorithm approach: it tries aligning by exploring the search space of all possible node mappings, allowing the best matches to survive and evolve, and

³⁷ Giorgos Kollias, Shahin Mohammadi, and Ananth Grama. Network similarity decomposition (nsd): A fast and scalable approach to network alignment. *IEEE Transactions on Knowledge and Data Engineering*, 24(12):2232–2243, 2011

³⁸ Gunnar W Klau. A new graph-based method for pairwise global network alignment. *BMC bioinformatics*, 10(1):S59, 2009

³⁹ Rob Patro and Carl Kingsford. Global network alignment using multiscale spectral signatures. *Bioinformatics*, 28(23):3105–3114, 2012

⁴⁰ Tijana Milenković, Weng Leong Ng, Wayne Hayes, and Nataša Pržulj. Optimal network alignment with graphlet degree vectors. *Cancer informatics*, 9: CIN-S4744, 2010



dropping the worst matches⁴¹.

48.3 Network Fusion

The final problem related to network distance/similarity is network fusion. Network fusion is a relatively old term and branch of computer science that up until recently had little to do with network science proper. It was introduced a few decades ago in the field of neural networks^{42,43}. The idea was that you trained a neural network on some data. The network has grown to be able to capture as much of the variation as possible. If you use the same algorithm to train on different data, you might end up with a similar, but not identical, configuration in your neural network. Some connections are stronger, others are weaker. Most of the variation between the same neural networks trained on different data is due to overfitting. Thus you want to build yet another neural topology, smoothing out all the noise. This is network fusion, because effectively you want to fuse together all the neural networks that you have trained. This might be an old idea, but is still an area of active research⁴⁴.

Figure 48.10 is the easiest mental picture you need to understand the principle of network fusion. We have two aligned networks in Figure 48.10(a) and Figure 48.10(b). We decide that we want to fuse them together by calculating the average edge weight. We also decide that we keep a connection in the fused network only if its resulting average weight is higher than 2. Figure 48.10(c) is the result. Of course, real network fusion algorithms are much smarter and more sophisticated than this.

Slowly but surely, network fusion crept into network science and found applications that go beyond increasing the performance and applicability of neural networks. For instance, consider genomic data. You can collect samples of interactions from many individuals. These are similar, but not always the same. You might want to combine them to create a prototypical interaction network⁴⁵. Alternatively, it could be that some of these samples are incomplete, and you can use

Figure 48.9: (a) A graph. (b) A node-motif table, counting how many times each node is part of a given motif.

⁴¹ Vikram Saraph and Tijana Milenković. Magna: maximizing accuracy in global network alignment. *Bioinformatics*, 30(20):2931–2940, 2014

⁴² Lars Kai Hansen and Peter Salamon. Neural network ensembles. *IEEE transactions on pattern analysis and machine intelligence*, 12(10):993–1001, 1990

⁴³ Sung-Bae Cho and Jin H Kim. Multiple network fusion using fuzzy logic. *IEEE Transactions on Neural Networks*, 6(2):497–501, 1995

⁴⁴ Xianzhi Du, Mostafa El-Khamy, Jungwon Lee, and Larry Davis. Fused dnn: A deep neural network fusion approach to fast and robust pedestrian detection. In *2017 IEEE winter conference on applications of computer vision (WACV)*, pages 953–961. IEEE, 2017

⁴⁵ Bo Wang, Aziz M Mezlini, Feyyaz Demir, Marc Fiume, Zhuowen Tu, Michael Brudno, Benjamin Haibe-Kains, and Anna Goldenberg. Similarity network fusion for aggregating data types on a genomic scale. *Nature methods*, 11(3):333, 2014

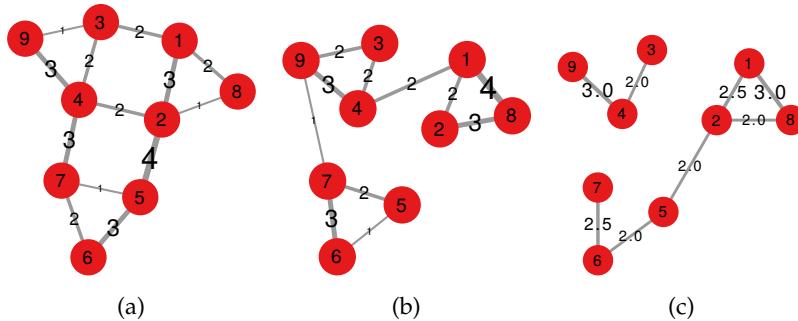


Figure 48.10: An example of network fusion: (c) is the result of the fusion of (a) and (b). Edge thickness proportional to its weight.

their fusion as the complete genomic data.

48.4 Summary

1. In this chapter we explore different ways to estimate the similarity/distance between the topologies of two networks: given two graphs, quantitatively estimate how much of their topologies are the same. Related applications are network alignment and fusion.
2. Network similarity can be done in many ways. One could compare the global properties of the network such as degree distribution and clustering; or aggregate all the node pairwise similarities; or estimate the number of edits needed to go from one network to the other.
3. One can combine all those approaches in a holistic one, determining which elements of the similarities between networks are more relevant, based on what the networks represent. Additionally, one could see adjacency matrices as signals and calculate the mutual information entropy between them.
4. Network alignment is the problem of finding a node-to-node mapping between two networks. We hypothesize that the two networks represent the connections between the same real world entities and we need to re-identify them by looking exclusively at the network topologies.
5. Network fusion is the process of taking multiple versions of the same network and reconstructing the underlying structure. The idea is that each observation might be noisy or incomplete, while their combination should represent the ideal structure.

48.5 Exercises

1. Estimate the similarity between the networks at <http://www.networkatlas.eu/exercises/48/1/data1.txt>, <http://www.networkatlas.com>.

- [eu/exercises/48/1/data2.txt](#), and <http://www.networkatlas.eu/exercises/48/1/data3.txt>, by comparing their average degree, average clustering coefficient, and density (average their absolute differences). Which pair of networks are more similar to each other?
2. Calculate the structural similarities of all pairs of nodes for all pairs of networks used in the previous question. Derive a network similarity value by averaging the node-node similarities. Since the networks are aligned, the node-node similarity is the Jaccard coefficient of their neighbor sets, and you should only calculate them for pairs of nodes with the same id. Which pair of networks are more similar to each other?
 3. Calculate the graph edit distances between the networks used in the previous questions. Remember that the networks are aligned, thus you just need to iterate over nodes and compare their neighborhoods. Which pair of networks are more similar to each other?
 4. Fuse the three networks together to produce a consensus network. You can keep an edge in the consensus network only if it appears in two out of three networks – assume that their are aligned and that nodes with the same id are the same node.

Part XIII

Visualization

49

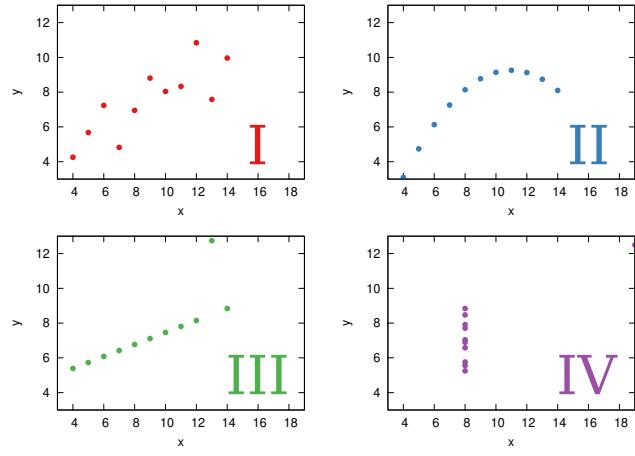
Node Visual Attributes

Data visualization is at the core of any data analysis undertaking – call it statistics, data science, or whatever else. There are two reasons why: gathering insights in exploratory data analysis, and presenting your results.

Let's start from gathering insights. It is sometimes – not always! – easier to spot patterns when you look at them with a proper visualization, rather than relying on summary statistics. The classical case for this position is the Anscombe quartet¹. In Figure 49.1(a) you have four datasets of two variables.

| I | | II | | III | | IV | |
|----|-------|----|------|-----|-------|----|------|
| x | y | x | y | x | y | x | y |
| 10 | 8.04 | 10 | 9.14 | 10 | 7.46 | 8 | 6.58 |
| 8 | 6.95 | 8 | 8.14 | 8 | 6.77 | 8 | 5.76 |
| 13 | 7.58 | 13 | 8.74 | 13 | 12.74 | 8 | 7.71 |
| 9 | 8.81 | 9 | 8.77 | 9 | 7.11 | 8 | 8.84 |
| 11 | 8.33 | 11 | 9.26 | 11 | 7.81 | 8 | 8.47 |
| 14 | 9.96 | 14 | 8.1 | 14 | 8.84 | 8 | 7.04 |
| 6 | 7.24 | 6 | 6.13 | 6 | 6.08 | 8 | 5.25 |
| 4 | 4.26 | 4 | 3.1 | 4 | 5.39 | 19 | 12.5 |
| 12 | 10.84 | 12 | 9.13 | 12 | 8.15 | 8 | 5.56 |
| 7 | 4.82 | 7 | 7.26 | 7 | 6.42 | 8 | 7.91 |
| 5 | 5.68 | 5 | 4.74 | 5 | 5.73 | 8 | 6.89 |

(a)



(b)

In all datasets, these variables have the same mean and standard deviation, the same correlation and even drawing a regression line between the two variables leads to the same result. You'd think that the four datasets are identical and no clear patterns distinguish them. However, simply *seeing* how these datasets look like – as I show in Figure 49.1(b) with a humble scatter plot – immediately tells you that there's something interesting going on.

¹ Francis J Anscombe. Graphs in statistical analysis. *The american statistician*, 27(1):17–21, 1973

Figure 49.1: (a) Four datasets with x and y coordinates. (b) Data visualization of (a) in the form of a scatter plot.

Then there is result communication. When you write a paper, you must state your results clearly and in an intuitive manner. In many cases, it is true that showing a picture of them is necessary. Thus, you need to be proficient in data visualization techniques, so that you won't accidentally trick your reader – or you won't trick yourself while reading a paper from a malicious miscommunicator.

The classical building blocks of network visualization are nodes and edges. Traditionally, we represent nodes as circles or dots, and edges as lines connecting dots. Then, nodes are scattered around so that the ones not connected to each other tend to be far apart. Instead, edges tend to be as short as possible, so connected nodes appear in close spatial proximity. This is so ingrained in network science visualization that you saw me using this approach throughout most of the examples I presented so far.

There are reasons why rules and best practices exist. They work in most scenarios. They also build familiarity: if you're exposed to the same strategies over and over again, you become literate and know immediately what's going on. In the majority of what follows I will align myself with these conventions. However, the first thing that needs to be highlighted is something that might appear obvious. Even if we represent them that way, nodes aren't dots, and edges aren't lines. The dot-line diagram is a *map*, not the *territory*. The reality that lurks behind a graph can take many forms and some will communicate better your intentions than others.

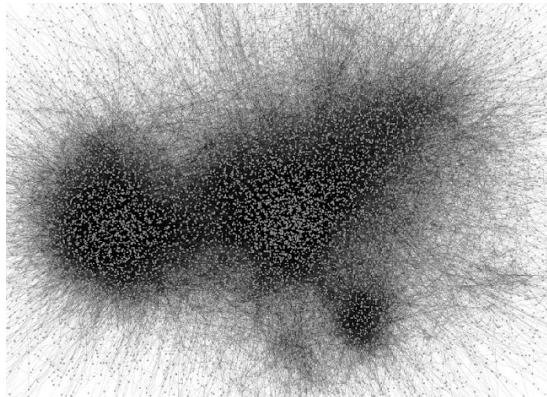


Figure 49.2: Hello hairball, my old friend. I've come to talk with you again, because a vision softly creeping left its seeds while I was sleeping and the vision that was planted in my brain still remains.

In practice, my aim is to give you the best practices and then empower you to break them when you feel they get in the way of your network visualization. Our journey is a fight against the nemesis of every network scientist who dares visualizing her own networks: the hairball. A hairball, or spaghettigraph, is something that looks like the example in Figure 49.2.

We already saw how the hairball can get in your way when you're

analyzing network data in Part VIII. Here, we're trying to defeat it in the realm of communicating to others what your network contains. If we trust the node-link diagram convention too much, we end up with visualizations that are cluttered like in Figure 49.2 and do not communicate much besides "it's complicated".

So, if there's something you will take away from this part, it is how to make hairballs less hairbally. We start in this chapter with node visual attributes, then we move on to edge visual attributes (Chapter 50) and network layouts (Chapter 51), with a small carousel of peculiar examples.

My software of choice is usually Cytoscape^{2,3}, which is the one I'm most proficient with. Most of the examples in this part will be based on Cytoscape and can be achieved by using it without any real programming skill. A popular alternative would be Gephi^{4,5}.

Finally, I should say that what follows is all practical knowledge of me messing up with Cytoscape for ten years and learning myself what looks good and what doesn't. I'm not an expert on data visualization and visual communication in general. If you want a more in-depth dive into proper visualization techniques – which go beyond simple network visualization – you're best served with one of the many awesome books and papers out there^{6,7,8,9,10}. You should also consider keeping an eye on some conferences on data visualizations such as IEEE Visualization Conference and the ACM Computer-Human Interaction conference.

49.1 Size

The first thing you might want to modify about a node is its size. This should be used for **quantitative** attributes, measuring some sort of importance of the node. They can be directly calculated from the graph properties (such as number of connections, PageRank, etc) or they can be provided as quantitative metadata. For instance, in a network where nodes are traffic junctions, it could be the number of cars that can pass through a street crossing per unit of time. It seems natural to encode the node's importance directly on its size, as I show in Figure 49.3.

The reason for using sizes – and other visual features as we will see – is to facilitate perception of the quantities and hence facilitate inference and insight. You want to make quantitative distinctions so that the variables you're visualizing stand out. If you cannot tell the difference between two different node sizes because the variations are too subtle, you're not communicating anything to your viewer. Thus, you have to have enough diversity in your visual features: just the amount that the human eye can perceive. This is one of the reasons

² Paul Shannon, Andrew Markiel, Owen Ozier, Nitin S Baliga, Jonathan T Wang, Daniel Ramage, Nada Amin, Benno Schwikowski, and Trey Ideker. Cytoscape: a software environment for integrated models of biomolecular interaction networks. *Genome research*, 13(11):2498–2504, 2003

³ <https://cytoscape.org/>

⁴ Mathieu Bastian, Sébastien Heymann, Mathieu Jacomy, et al. Gephi: an open source software for exploring and manipulating networks. *Icwsm*, 8(2009): 361–362, 2009

⁵ <https://gephi.org/>

⁶ Ben Shneiderman. The eyes have it: A task by data type taxonomy for information visualizations. In *Proceedings 1996 IEEE symposium on visual languages*, pages 336–343. IEEE, 1996

⁷ Alberto Cairo. *The Functional Art: An introduction to information graphics and visualization*. New Riders, 2012

⁸ Isabel Meirelles. *Design for information: an introduction to the histories, theories, and best practices behind effective information visualizations*. Rockport publishers, 2013

⁹ Edward Tufte and P Graves-Morris. *The visual display of quantitative information*; 1983, 2014

¹⁰ Tamara Munzner. *Visualization analysis and design*. AK Peters/CRC Press, 2014

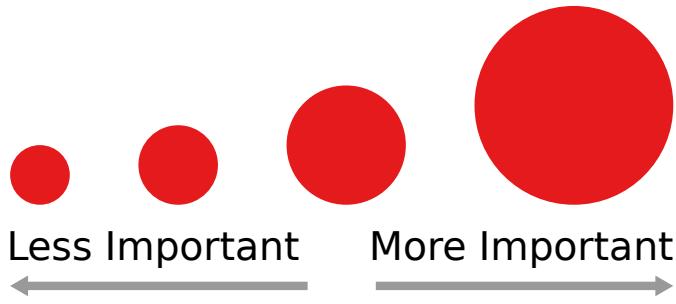


Figure 49.3: The natural scale with which we can use node size – meaning: its area – to confer the idea of its importance.

why it is so hard to create visualizations. Finding the right scale to encode a quantity into a size is hard, especially when you’re dealing with continuous variables and you have to bin them yourself.

Taking Figure 49.4 as an example, in Figure 49.4(b) you cannot really tell who is boss by simply looking at the node sizes. As soon as we exaggerate the size difference – Figure 49.4(c) –, it becomes clearer and the visualization becomes more informative and, arguably, visually more pleasing. Differences have to jump to the eye: making subtle changes is not going to communicate much to the viewer. In other words, to facilitate the viewer’s perception of the differences in the quantities mapped, your visualization has to have enough “action”, differences, it has to say something.

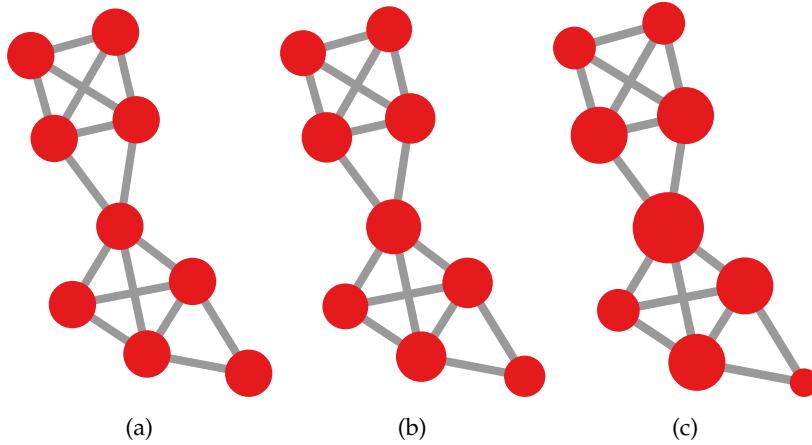
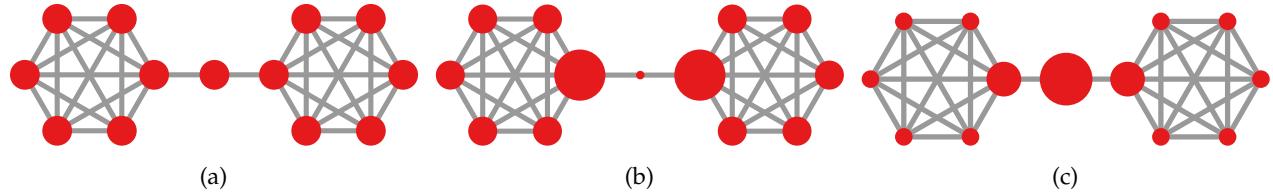


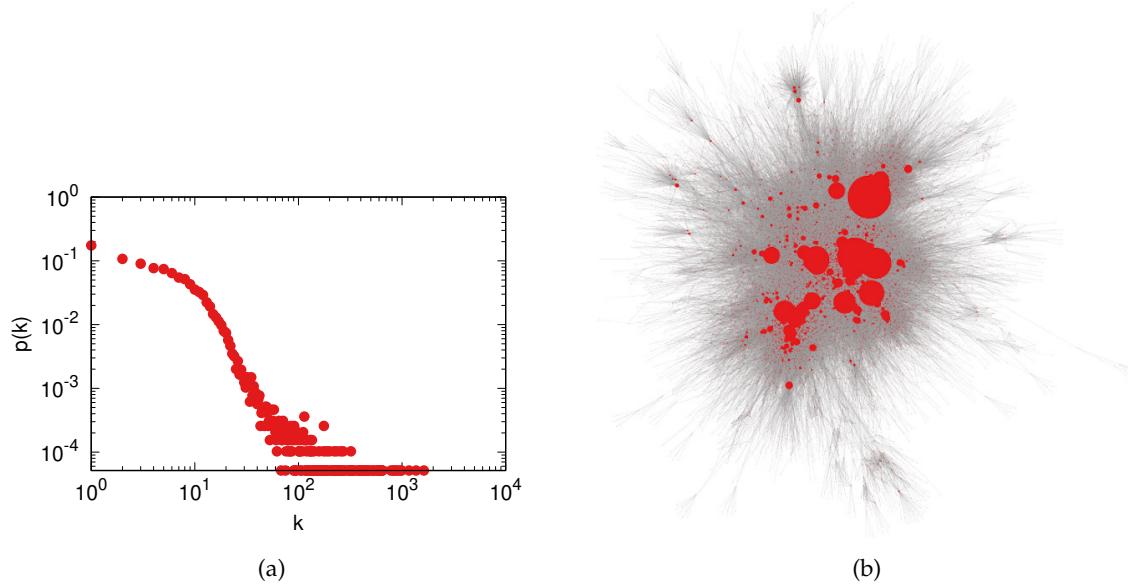
Figure 49.4: (a) A graph in which all nodes have the same size. (b) A graph in which the node’s degree determines its size, with subtle variations. (c) Same as (b), but exaggerating the node size variation.

You cannot simply take away the message that any quantitative measure of node importance is an equally good choice for your node size. Some of those measures will not highlight what you want to highlight. For instance, the degree is not always the right choice. Consider Figure 49.5(a): would you think to use the node’s degree as a measure of its size? If you do, you end up with Figure 49.5(b) where the node playing arguably the strongest role in keeping the network together almost disappears. A much better choice, in this case, is betweenness centrality (Figure 49.5(c)).



It shouldn't surprise you – after all the network analysis we've done – to hear that many variables of interest in a network have very broad distributions. Degree and betweenness centrality, the two examples cited so far, follow (quasi) power laws, with few gigantic hubs and many nodes with minimum values. This means that linear size scales are not going to work very well: everything is going to be tiny and then BAM! One huge node, the hub.

Figure 49.5: (a) A graph in which all nodes have the same size. (b) A graph in which the node's degree determines its size. (c) Same as (b), but using betweenness centrality instead of the degree for the nodes' size.



Consider Figure 49.6. Here, we use the degree to determine the node size and we use a linear scale. Since this is an example of comic book characters, we expect the ones appearing with many other characters in the same comic book to be the most important. And they are: the largest nodes are the ones you would expect. But... they are too much the ones you expect. Their size swamps everything else. As a result, the visualization might be *truthful*, but it's not *informative*. It doesn't show you any new information. You already knew all you can gather from it.

To counteract this, you need to apply a quasi-logarithmic scaling. If you're creating your visualizations programmatically you can have an actual log scale, although you probably will still have to manually

Figure 49.6: (a) Degree distribution of the Marvel social network example. (b) Visualizing the network with a linear node size map, where the degree directly determines the node size.

tweak it a bit to make the result more pleasing. The idea is to have diminishing returns to the contribution of the degree to the node size. The differences in size from the minimum degree, to the average degree – which can be quite low – are big but, from that point on, the contribution to the node's size plateaus.

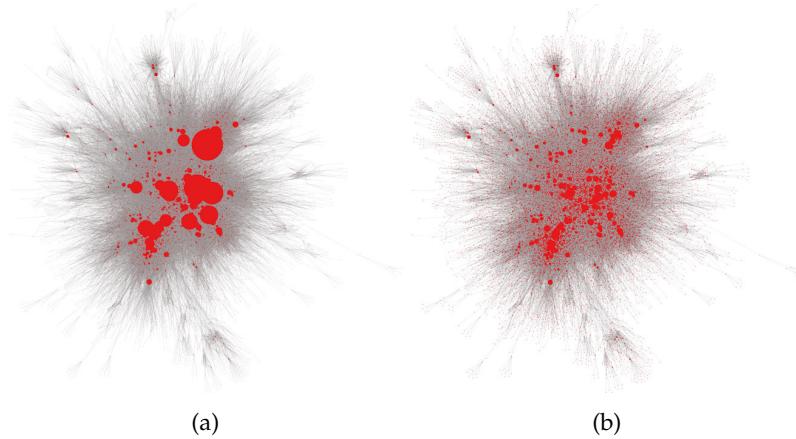
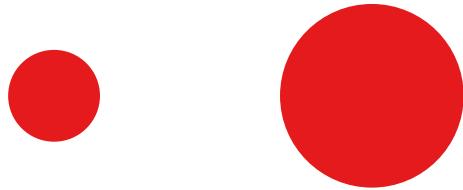


Figure 49.7: (a) The comic book social network using degree directly for node size. (b) Same network using a quasi-logarithmic scaling for the node size.

If you do so, you can find new clusters that were previously cluttered by the huge nodes, or that had a low degree and so they did not pop up. You can compare the two hairballs in Figures 49.7(a) and 49.7(b). Note that the visualization is still truthful: we're never going to make nodes with lower degree larger than nodes with higher degree. That would be bad and land you in the naughty corner. We're just making the visualization more useful.

This is probably a good place to stop and make a disclaimer. Even if eyes are the highest bandwidth sensors we have, it doesn't mean they are flawless. Nor that our monkey brain is able to use the information they gather in a perfect way. Human perception is flawed and you cannot expect that something a computer understands will appear obvious to your viewers as well. In the case of node size this takes the form of the confusion between radii and areas.

Unless otherwise specified by the software/program of choice, you are going to decide the *radius* of the node when determining its size. This can be trouble if you don't handle this choice properly. The reason is that, when you increase the radius, you are substantially performing a linear increase: you think that, if the degree increases by one unit, you should increase the radius by one unit. Unfortunately, what a viewer will perceive is you changing the *area* of the circle. The crux of the problem is that a radius is a one dimensional quantity, and it should never be used for controlling a two dimensional one such as an area – which is what your readers perceive. You think you're increasing something linearly, but you're actually



| | |
|--------------|--------------------|
| Degree = 1 | Degree = 2 |
| Radius = 1 | Radius = 2 |
| Area = π | Area = 4π (!!) |

Figure 49.8: A human perceives a node with radius one as being of size π , its area. So she will also perceive a node of radius two as being of size 4π : double degree, but four times as large!

raising that increase by the power of two.

Figure 49.8 shows you why you need to be well aware of the difference. What you think is a small increase can seem humongous to your reader.

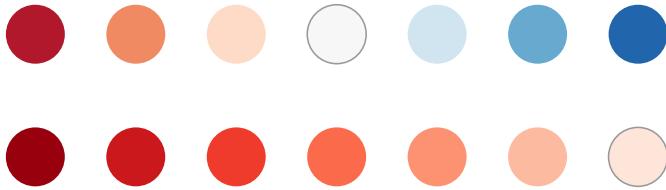
49.2 Color

The second obvious feature to manage for your nodes is their color. If we routinely use node sizes for quantitative attributes, we primarily use node color for **qualitative** ones. The reason is that, while humans perceive size as quantitative, color hue is not perceived in the same way. Cleveland and McGill¹¹ distinguish between different data types and how much different graphical features are effective for each data type. Color is good for nominal attributes – categories that cannot be compared/sorted, like “apple” vs “orange”. Color could be used for ordinal attributes, that are still categories but can be compared – for instance days of the week, Monday comes before Tuesday. Color is terrible for quantitative attributes, for which areas are a more effective tool. As always, what follows is based on my experience and, if you want or need more in-depth explanations, you should check out the paper.

When it comes to network visualization, this implies that we put nodes into classes and we use colors to emphasize that different nodes are in different classes. Classical examples can be the node’s community – Part X –, or its role – Chapter 15. I already mentioned nodes can have metadata, and these metadata could be categorical. For instance, in a network connecting online shopping products because they are co-purchased together you could use the color to determine their category (outdoors, rather than kitchen, rather than electrical appliances).

You could still use node color for ordinal attributes, and maybe for quantities as well, provided that you have clear and intuitive bins.

¹¹ William S Cleveland and Robert McGill. Graphical perception: Theory, experimentation, and application to the development of graphical methods. *Journal of the American statistical association*, 79(387):531–554, 1984



The way one would use colors for quantities is by implementing a gradient. A classical one is a blue-red spectrum for temperatures: this is a diverging scale that can be useful, e.g., if you have some sort of correlation data. You have a very precise and semantically meaningful middle point, and nodes can diverge in either of two directions, as I show in Figure 49.9 (top). Otherwise, if we're talking of a more classical intensity – say how much money a customer spent in your online shop – you want a simple sequential gradient, just like the one in Figure 49.9 (bottom).

There are many things you need to take into account when using colors. One of the trickiest ones is cultural associations¹². When you visualize something, your visualizations come after centuries – if not millennia – of other people using colors for different tasks. These usages ingrained in our mind a quick way to decode information. For instance, we associate red with danger, yellow with caution, green with “good to go”. Black is death, and – stereotypical – blue is for boys and pink is for girls. But blue is also Democrat against red Republican if we're talking about elections in the US – which, interestingly, is the opposite of the left-right wing spectrum for other countries in which red is on the left. It all depends on the context in which you're visualizing. Color can aid you in making your visualization quicker to decode, but if you're instead using it differently from a convention it can make things harder.

This doesn't even take into consideration the deficits in the physical perception by humans. Just to repeat myself – we are very limited when it comes to distinguish colors. If you ask your laptop how many colors there are out there, a popular reaction would be counting the number of possible RGB combinations and to reply: 16 millions! That would be very wrong for any human with a hint of common sense. In fact, what I would say is that you should never use more than nine colors in your visualization, and I'm sure that a few of my data designer friends are already gasping in horror to the extent of my liberalism. Nine, for them, is already way too much.

It's not just about the quantity of colors, though, it is also about how to choose and use them. How many colors would you say I used for the nodes in Figure 49.10(a)? If you guessed 16 – which is the correct answer – you're very lucky, or you have some Truman

Figure 49.9: (top) A gradient palette for diverging quantities and a meaningful middle point of the spectrum. (bottom) An intensity gradient, useful to go from zero to a maximum value without a meaningful middle point.

¹² Samuel Silva, Beatriz Sousa Santos, and Joaquim Madeira. Using color in visualization: A survey. *Computers & Graphics*, 35(2):320–333, 2011

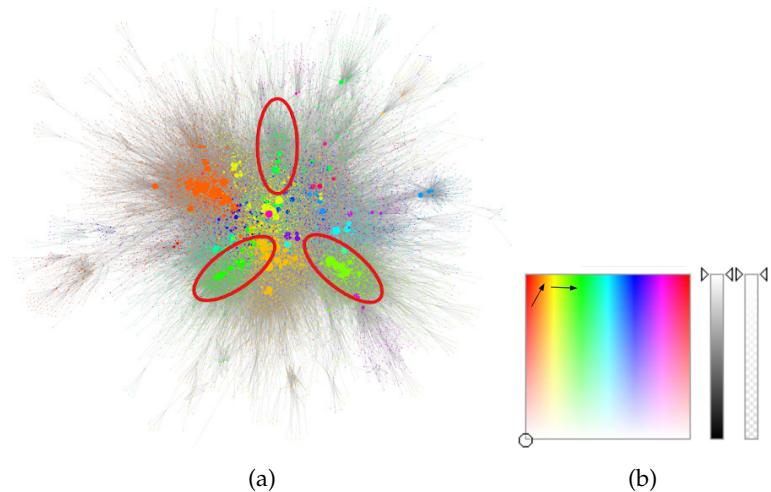


Figure 49.10: (a) The comic book social network using communities for the node's color. (b) An example of the RGB color space. The black arrows indicate equal distance movements in this space, connecting colors at different human perceptive distances.

Capote levels of pattern recognition. In the network I highlighted three groups of nodes. These have different colors, believe me or not. Few – if any – people would be able to tell without scanning the figure for more than a handful of seconds. Requiring this level of effort from your viewer means to lose them.

Why does Figure 49.10(a) fail? Because it assumes that RGB is a perceptive color space. In a perceptive color space, if you move by a given amount, you get to a color that will be perceived differently. This is wrong, as Figure 49.10(b) shows: the two black arrows in it make two movements in the space and show that the same RGB space distance can connect either two virtually identical colors – virtually identical for our monkey brains – or two very distinct ones.

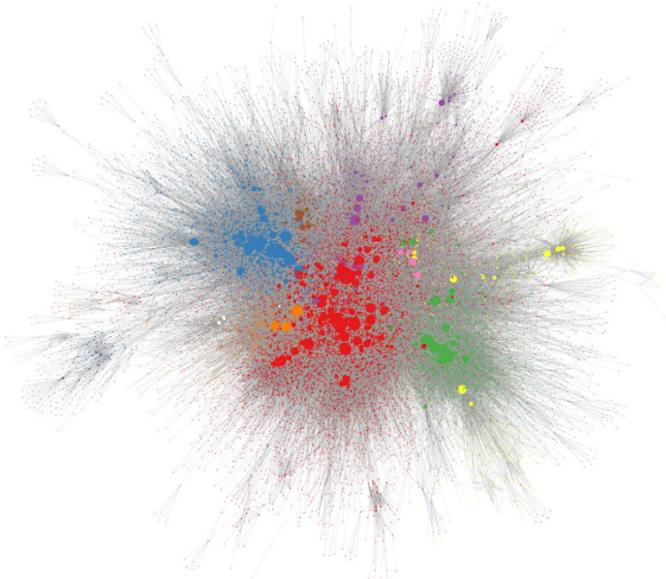
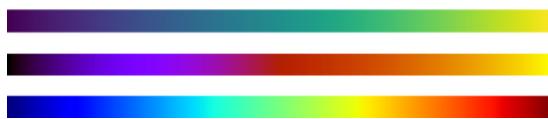


Figure 49.11: A version of Figure 49.10(a) with fewer colors and based on a more sane color space than RGB.

If there is one message I wish I could ingrain in you after reading this material is this one: RGB is a terrible terrible terrible color space for information visualization and nobody should use it for anything related to data design ever. There is some research backing color palettes that align better with human perception – also including the case of people with color blindness¹³. A good resource you can use is the Color Brewer interactive tool¹⁴, which will generate the palettes for you¹⁵. Color Brewer is embedded in many software/programming packages that you might already use for your visualizations, including R¹⁶, Cytoscape (since version 3.7.1, for earlier version you need the Color Cast plugin), QGis, Python (Matplotlib and Seaborn, for instance), and Matlab.

Figure 49.11 uses the Color Brewer space and fixes one of the many problems of Figure 49.10(a). In Figure 49.11 we use also fewer colors – just nine – which is always good.

Color Brewer and RGB are not the only possible color spaces you could use. If you are creating visualization for printing, you should use a CMYK color space. This is similar to RGB, but RGB is an *additive* color space, while CMYK is *subtractive*. Additive color spaces describe how different wavelengths of light add to each other, which is how computer screens work. Subtractive color spaces, instead, describe how ink combines on the page, which is why it'll show better how things will look in print. HSV and HSL are alternative color spaces which transform RGB to be more perceptually-relevant: we as humans don't really perceive colors as combinations of red-blue-green, but as variation in hue, saturation and lightness, which is what HSL stands for.



There are other options for linear color gradients – which I show in Figure 49.12. Some of these palettes were systematically compared across a series of tasks viewer might want to perform¹⁷, as well as different issues your readers might have with perceiving colors. For instance, to tackle the aforementioned issue of color blindness, you could transform these palettes into their correspondent black and white version and see how they look like to a person unable to distinguish colors but only relying on lightness. I do exactly this in Figure 49.13 and show that, for instance, the jet palette in Matlab performs poorly because the two ends of the spectrum become indistinguishable. Of course, testing for color blindness and other human vision deficiencies is much more complex than this, and you

¹³ Cynthia A Brewer. Color use guidelines for mapping. *Visualization in modern cartography*, 1994:123–148, 1994

¹⁴ Mark Harrower and Cynthia A Brewer. Colorbrewer.org: an online tool for selecting colour schemes for maps. *The Cartographic Journal*, 40(1):27–37, 2003

¹⁵ <http://colorbrewer2.org/>

¹⁶ Erich Neuwirth and R Color Brewer. Colorbrewer palettes. *R package version*, pages 1–1, 2014

Figure 49.12: Some linear color palettes you'll find available in different software. From top to bottom: viridis (Matplotlib), Gnuplot default, and jet (Matlab).

¹⁷ Yang Liu and Jeffrey Heer. Somewhere over the rainbow: An empirical assessment of quantitative colormaps. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, pages 1–12, 2018



should delve deeper in the literature I cited at the beginning of the chapter.

Note that here I used node colors for communities. It seems that I'm suggesting that you shouldn't find more than nine communities in your networks. That is not exactly what I'm saying. Of course, when it comes to the *analysis*, you will find the number of communities that you will find. The sky is the limit there. It's when it comes to *visualizing* them that you should never show more than nine. If you try to break that limit, you may as well not visualize anything. A famous motto in data visualization is: "emphasizing everything means to emphasize nothing". So you need to find a different solution, maybe showing smaller extracts of your data.

As with node sizes, also in node colors – if you're applying a gradient – you're best served using a (quasi)logarithmic scale to highlight differences better and make color variance more meaningful.

49.3 Other Features

To wrap up this chapter, let's see a few more things you can do to your nodes. They both stem from the same idea: your nodes represent something, and so you want to communicate this to your viewers.

The first strategy involves **node labels**. If you want the audience to know something, you simply tell them. You plaster some text on top of your nodes and you call it a day. In my opinion, this is a desperate move and it should be avoided if possible. Just as in movies, also in data visualizations it's better to "show, not tell". In other words, nobody wants to *read* your network. They want it to speak to them.

That is not to say that sometimes a good choice of node labels can enhance your visualization. You can practically transform your network into a glorified word cloud. I don't love it, but I grudgingly admit that sometimes it works. An example could be the one in Figure 49.14 – although in this case one should choose a less saturated color for the nodes, because the current red goes in the way of the readability of the label. My rule of the thumb is that the node label font size should have a one-to-one correspondence to the node size. It would look weird to have a gigantic label on top of a tiny node, and vice versa.

The second visual attribute you could play with is the node's

Figure 49.13: The linear color palettes from Figure 49.12, transformed in a grayscale. From top to bottom: viridis (Matplotlib), Gnuplot default, and jet (Matlab).

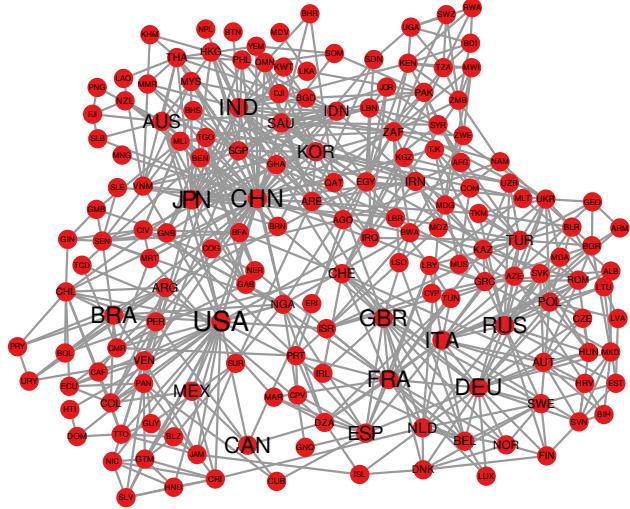


Figure 49.14: A network with node labels conveying information about a node's importance. In this trade network, it is the country's Gross Domestic Product.

border. This is an interesting one, because it could be used for quantitative and qualitative attributes at the same time. For the node border, you can both decide the color *and* the thickness. Again, you should really ask yourself whether you really need to do it. Personally I almost never touch node borders – I'd say that in 99% of my visualization the border is invisible. If you already have node sizes and colors, adding a border of a different size and color would just cause information overload in your reader's brain. You should only do it if there are extremely clear patterns in your network, which involve no more than a handful distinct values, and that can be easily parsed.

For instance, in Figure 49.15, we could have two nodes of same size and color – perhaps these are two plants in the same country (color) and employing the same number of people (size). However, they process different products (border color) and they have different throughput in number of products processed per day (border thickness).



Figure 49.15: Two nodes with same color and size, but with borders of different thickness and color.

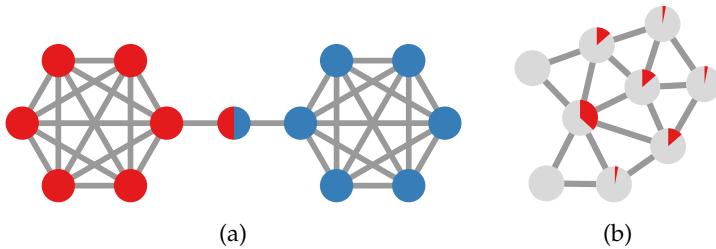
Another strategy is more creative – and for this reason you should apply tons of caution if you want to go this way. It involves xenographic¹⁸. This translates to “weird visualizations”, stuff that has very specific and almost unique use cases, and thus it's likely to choose a style that people haven't seen before. You can be creative with what you put on your nodes, as long as you don't abuse it and

¹⁸ <https://xeno.graphics/>

it has a meaningful relationship with your message.

One obvious way you can communicate differences in kind when it comes to a node would be to represent it not as a dot, but as a figure. The classical case is by transforming the node's shape. I already used this approach in this book for bipartite networks. A classic way to visualize them is to use one node shape for V_1 nodes, and another for V_2 nodes. For instance they can be circles vs squares. Another way is to use symbols. For instance, you could have a network of dogs and use a silhouette of the dog's appearance to encode its breed – this is inspired by the beautiful “Top Dog” visualization¹⁹.

My favorite use case, instead, keeps the node's shape constant, but transforms it into a chart in itself. This involves the often-maligned pie charts. Pie charts get a bad rep, often deservedly so, but can be rather useful in specific instances. You can use them both in the qualitative and in the quantitative use case, making them more versatile than either node color or size.



¹⁹ <https://www.informationisbeautiful.net/visualizations/best-in-show-whats-the-top-data-dog/>

Figure 49.16: (a) Using pie charts on nodes to signify their allegiance to multiple communities. (b) Using pie charts to represent the relative centrality of each node.

We know that communities in networks can share nodes (Chapter 38). If you're using the node color to encode the community, what do you do if a node belongs to more than one of them? You can use a pie chart for that! Figure 49.16(a) shows an example. This assumes that the node is not part of too many communities but, if you have enough communities in your network to break a pie chart, you shouldn't use colors to encode them to begin with.

In the quantitative case, pie charts are more limited, but still work in case you have “quantitative classes”. With that, I mean that you have a quantitative attribute that can take very specific and very different values, such as a centrality. In that case, distinguishing between few very different pie charts is possible even for the human brain, as you can see in Figure 49.16(b).

A final xenographic touch concerns playing with the alpha channel – i.e., the opacity of the node. In this paragraph, I consider the extreme case of not showing the nodes at all. Normally, there's no point in having invisible nodes. After all, the nodes are what you want to see in a network, so why making it impossible to look at them? However, there are some use cases in which this rule can be

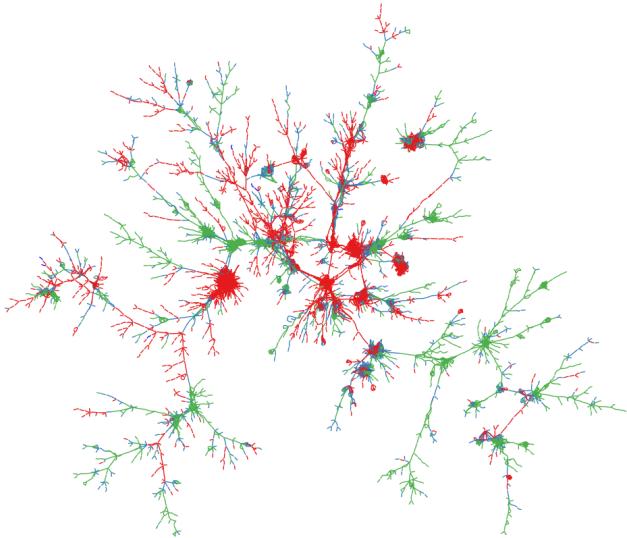


Figure 49.17: A network with fully transparent nodes, where all the topological information is conveyed by the edge colors.

broken. I present one in Figure 49.17. I'm not arguing that the figure is a *good* visualization: what I'm saying is that being able to see the nodes would not make much of a difference, especially since they do not have attributes of interest. Rather, the visualization allows you to see where different types of edges create red and green clumps, and which edge type keeps the network together in which branches. And how to deal with edge visual attributes is exactly the topic of the next chapter.

49.4 Summary

1. The first visual attribute of nodes is their size. Usually, you want to show quantitative attributes via size – the degree, the capacity, etc. Be aware that you should always manipulate the *area* of the node, which is what your viewer perceives. If your software only allows you to control a node's *radius*, keep in mind that your area will change quadratically for each linear change of the radius.
2. Second, you can control a node's color. Usually, this is for qualitative attribute, e.g. community affiliation. Use no more than nine distinct colors, from a perceptual-aware space (*not* RGB rainbows!).
3. Gradients can be used for quantitative attribute: diverging ones for quantities with a clear midpoint – e.g. correlations –, otherwise sequential ones for quantities going from zero to an arbitrary maximum.
4. You can augment your nodes with additional visual elements. Labels could be used – sparingly – and their size should be locked

with the node's area size. You can use pie charts and icons to embed additional information on the nodes.

49.5 Exercises

1. Import the network at <http://www.networkatlas.eu/exercises/49/1/data.txt>, calculate the nodes' degrees and use them to set the node size. Make sure you scale it logarithmically. This can be performed entirely via Cytoscape. (The solution will be provided as a Cytoscape session file)
2. Import the community information from <http://www.networkatlas.eu/exercises/49/2/nodes.txt> and use it to set the node color. (The solution will be provided as a Cytoscape session file)

50

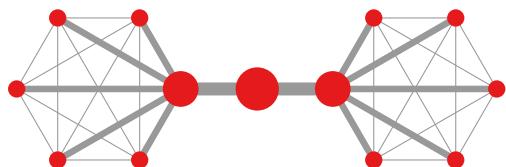
Edge Visual Attributes

When it comes to edge visual attributes, most of the things already mentioned for nodes in Chapter 49 still apply. So this is going to be mostly a recap, with a few additional warnings.

50.1 Classical Visual Elements

Size

The equivalent for edges of node size is the thickness. As in the previous case, this is mostly for quantitative attributes on edges. The most trivial one is the edge's weight: heavy edges usually appear to be more thick. Another common use case is to put edge betweenness as the determinant of the edge thickness. This works well when used in conjunction with nodes sizes following the same semantics. It gives a sense of balance to the visualization, so you can see which edges are contributing to the node's centrality. Figure 50.1 shows an example.



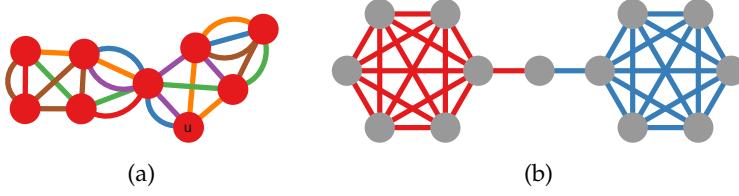
Just like node betweenness, also edge betweenness is unevenly distributed across edges. And, as you already saw, typically edge weights distribute equally broadly – see Chapter 27 for a refresher. So you have to apply the same pseudo log scaling for edge thickness as you did for node size. Lines are considered one dimensional, so you shouldn't worry too much about the square area problem I mentioned for node sizes. It will start to be a problem only if your edges are so large that your eyes start interpreting lines as rectangles, at which point they're probably already too large!

Figure 50.1: In this network the edge thickness is proportional to its edge betweenness value. The node size is proportional to the node betweenness.

Color

When it comes to colors, there are more differences between edges and nodes than we just saw for sizes¹. The fundamental difference between edges and nodes is that there are so many more of the former than of the latter: typically twice or three times as many. Also, dots and circles are much easier to see than lines, especially thin lines. Since most edges will have low weights, most of them will be relatively thin, as we just discussed. Thus, seeing the edges is trickier.

There is another reason, which is even more practical. Very rarely, if at all, you will have good qualitative information about your edges. By their very nature of being the glue connecting things, edges are much more likely to have quantitative information attached to them. We usually classify things, not the glue between things.



The most obvious exceptions are two. You can have qualitative information telling you to which layer and to which community an edge belongs, if you have multilayer networks (Section 7.2) and/or link communities (Section 38.5). For multilayer networks you can use edge colors to represent the layer if you adopt a multigraph visualization – as I do in Figure 50.2(a). However, this will get unwieldy pretty soon, as the number of nodes, edges and layers grows beyond an elementary size. In fact, it's usually best to use dedicated tools for the visualization of multilayer networks² – although the field of visualizing multilayer networks is still in its infancy.

For link communities, keep in mind the same warning I made regarding node communities. It is pointless to try and visualize more than a handful – nine – communities, even more so when it comes to links. Figure 50.2(b) shows an example. Again, it's not that you should always find fewer than nine communities in your analysis. You can and should find however many there are in the network. It's visualizing them that is a problem.

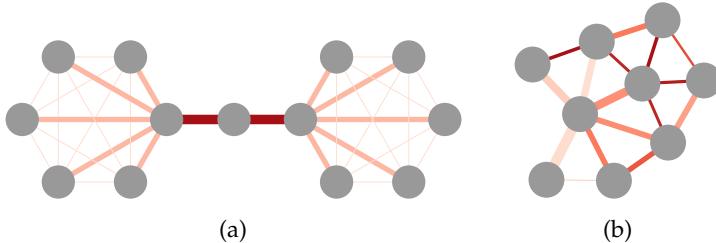
As said, most often you will have quantitative information on your edges. So it is much more common to use gradients on the edges than it is on the nodes. You have gathered that I'm not a great fan of gradients from the previous chapter, but that's what we have to work with. The way I usually fix the problem is to use the same quantitative attribute for both thickness and color, so that the two

¹ Danielle Albers Szafir. Modeling color difference for visualization design. *IEEE transactions on visualization and computer graphics*, 24(1):392–401, 2017

Figure 50.2: (a) Using edge colors to represent the edge's layer. (b) Using edge colors to represent the link community to which they belong.

² Manlio De Domenico, Mason A Porter, and Alex Arenas. Muxviz: a tool for multilayer analysis and visualization of networks. *Journal of Complex Networks*, 3(2):159–176, 2015c

can work in concert and reinforce each other. Two imperfect visual clues can sum and make each other clearer. This is the case for edge betweenness, determining both color and thickness of the edges in Figure 50.3(a).



That is not to say that there aren't good reasons to break this rule. One dimension I play with is usually the one of the edge weight's significance, for instance when doing network backboning – see Chapter 27. In that case, the color can work in contrast with the thickness. I find more natural to use thickness to represent how heavy an edge is, so to assign it to represent the weight. This is under the metaphor that large things generally weigh more. On the other hand, there is no inherent connection between the color of a thing and its weight. So I assign the color to represent the significance. Usually, larger weights tend to connect nodes which have higher average connection weights, so large links will tend to have paler colors than smaller links, which creates a nice contrast in Figure 50.3(b).

A final word of caution about the number of colors in your visualization. You might have noticed that Figures 50.2 and 50.3 use edge colors but choose a single hue for the nodes. This is because in a network you have two visual elements – circles and lines – and allowing both of them to be colored differently effectively doubles the number of visual elements in your visualizations. Already Figure 50.2(a) feels way too busy. If in the previous chapter I told you not to use more than nine colors in the visualization, here I'm giving you a more nuanced guideline: the sum of the distinct number of colors for edges and nodes must be nine or lower. Meaning that, if you have five colors for nodes, you shouldn't have more than four for edges.

Transparency

Transparency is another aspect in which edges diverge from nodes. In the previous chapter I mentioned that nodes should be fully visible, and provided only a single use case in which I believe transparency can add something to the visualization by removing the nodes from sight. When it comes to edges, I usually abuse trans-

Figure 50.3: (a) Using edge colors to reinforce the message conveyed by the thickness: the edge's betweenness centrality. (b) Using edge colors for an orthogonal quantitative information: edge thickness is its weight, while the color represent the weight's significance.

parency lavishly. Most commonly, I make transparencies work together with colors, to reinforce them. Significant links have darker colors *and* are more opaque. The objective of playing with the alpha channel for edges is to create a visual hierarchy, where nodes come to the forefront and edges go to the background.

However, sometimes you can play with edge transparencies even if you don't have any attribute to attach to them at all! This is because of the sheer number of edges: in most real world networks, they are going to overlap to each other, no matter what. Thus edge transparency, even a fixed value, can highlight structure, because there are going to be more overlaps in dense areas of the network than in sparser ones. Thus, you can highlight such clusters even without any edge metadata.

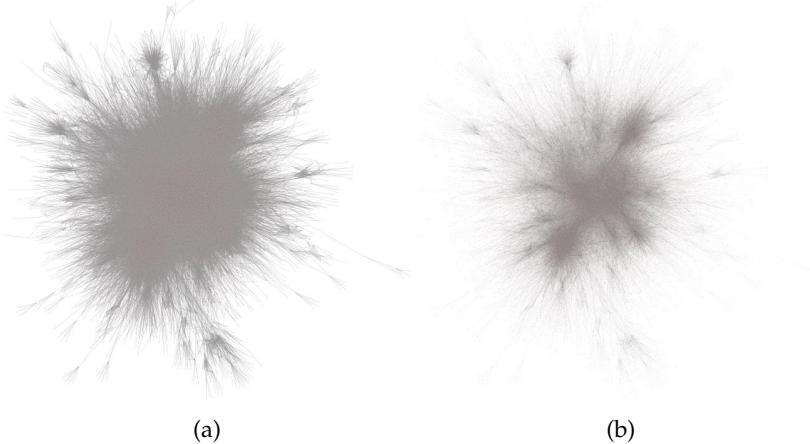


Figure 50.4: (a) A network with solid edges with 100% opacity. (b) The same network, but this time all edges have a 37.5% opacity, no matter their weight.

I do exactly that in Figure 50.4(b). Compared to Figure 50.4(a), the version with edge transparency looks less like a random smudge on the paper and shows a few structures of interest, even if I added literally zero bits of information by removing some edge opacity.

Labels

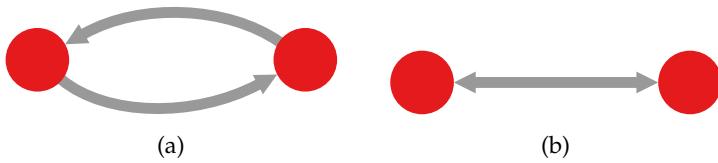
Moving on to edge labels: if I said that node labels have to be rarely used, then use edge labels even less than that. This is an extremely rare use case, you should avoid edge labels at (almost) all costs. Practically, they're only useful for scholastic examples, when explaining simple dynamics of super simple graphs. For instance, I used edge labels in this book for examples of weighted edges in weighted networks, with a grand total of five nodes and five edges. That's more or less stretching the use case of edge labels to the limit.

50.2 Xenographic Elements

Just like with nodes, also with edges you can be... edgy in how you visualize them. There are two fundamental aspects I'm going to mention here: shapes and bends.

The classical edge visualization is as a straight, solid line. This is what you should do in 99% of the cases. However, in many cases, you might want to slightly change this shape. The most common shape change for edges is when you are working with directed connections. In this case, the convention is to add an arrow that indicates the direction of the edge. The arrow points from the originator of the edge to the target.

Directed networks are more challenging to visualize than you might think. The reason is not only that you're doubling the possible number of edges, which is true and it is an issue. But the real trouble is that now you might have a significant number of double edges between the same two nodes: $u \rightarrow v$ and $u \leftarrow v$. This might make your visualization a real mess. One convention you can implement is not to actually draw the two edges. What you can do is to draw a single edge and add to it a second arrow pointing in the opposite direction if that edge is reciprocal. Figure 50.5 shows how this strategy looks like.



Unfortunately, this is not an immediately obvious thing to do with standard network plotting software, so it might take some effort. Moreover, this visualization technique gets significantly more complicated in the case of weighted directed networks. If the two edges, $u \rightarrow v$ and $u \leftarrow v$, have two different weights, it is even less clear how you should handle visualizing them in a single line.

Line shapes can be manipulated in other ways, specifically by altering the line style. One can have dashed, dotted, wavy lines. These are clearly differences in qualities of the connection, and thus should only be used for qualitative attributes. My advice would be to rely on changing the edge line style exclusively for (i) very small networks, and (ii) just in those rare cases you need to print your visualization in black and white and thus cannot use color instead. It is clear that, once you have a lot of connections, they are going to inevitably be drawn one on top of the other. Distinguishing between a dashed and a dotted line if they overlap is nigh impossible.

Figure 50.5: (a) The classical reciprocal edge visualization with two bent edges. (b) Merging the two edges in a single reciprocal edge.

The final odd thing you could do to your edges is to bend them, meaning that the (u, v) edge is not a straight line from u to v any more, but it takes a “detour”. Why would you want to do this? There are fundamentally two reasons. Figure 50.5(a) provides an example: since there are two edges between the nodes, we want the visualization to be more symmetric and pleasant, and thus we bend the two edges.

More often, edge bends are used to make your network layout more clear. You bend edges to bundle together the ones going from nearby nodes to other nearby nodes. Since this is done mostly to clean up the visualization *after* you already decided where the nodes should be placed, I will deal with this topic in the network layout chapter (Chapter 51).

50.3 Network Lifting

Let’s recap all the advice I gave you on node and edge visual attributes and see a case of applying each feature one by one to go from a meaningless hairball to something that conveys at least a little bit of information. Our starting point is the smudge of edges you already saw a couple of times: that’s Figure 50.4(a). Note that this isn’t really the starting point, because we already settled on a network layout, but that will be the topic of the next chapter.

The usual order I apply to my networks after I settled on a layout is the following:

1. Edge transparencies;
2. Edge sizes;
3. Edge colors;
4. Node sizes;
5. Node colors.

So let’s do this.

Edge transparencies. In this network, I do have quantitative information, that is the edge betweenness of each connection. However, I think that it’s better if I limit that to the other edge visual features, so I fix the same edge transparency to all links. The result is Figure 50.4(b).

Edge sizes & colors. We now move on to use edge betweenness. I merge the two steps of edge size and color into one, because using simply the thickness does not make a significant difference with the previous visualization. Compare Figure 50.4(b) with Figure 50.6(a)

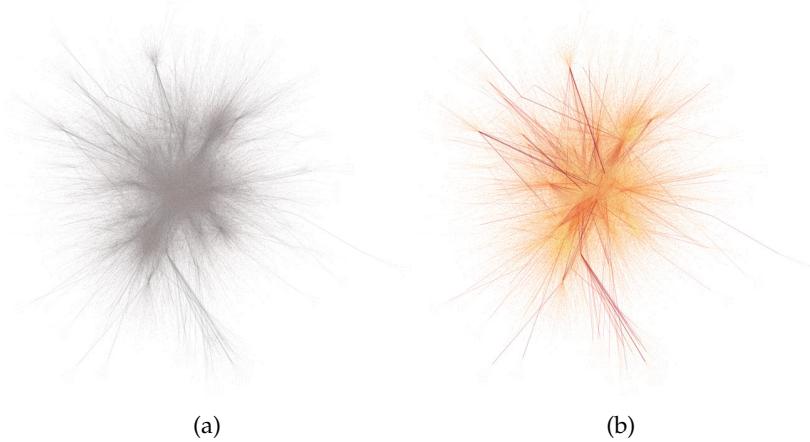


Figure 50.6: (a) Using edge betweenness for the link's thickness of Figure 50.4. (b) Using it for the link's color as well.

and see that not much has changed. So I apply a Color Brewer color gradient to the edges, resulting in Figure 50.6(b). Hopefully now you can see that there are a few very important long connections keeping the network together, connecting very central comic book characters to a sub universe they're almost exclusively part of.

Node sizes. It's time to deal with nodes. There's something to be said for keeping them almost invisible, but that's not what we want to do here. This is a comic book network and so we want to know which characters are tightly connected to the universe of which other characters. So first we need to know who the important fellows are. We use node size for that. As outlined in the previous chapter, this is a job for the degree. The more connections a node has, the more important it is, the larger it should be. And that's what Figure 50.7(a) does. Note the pseudo-logarithmic node size scaling.

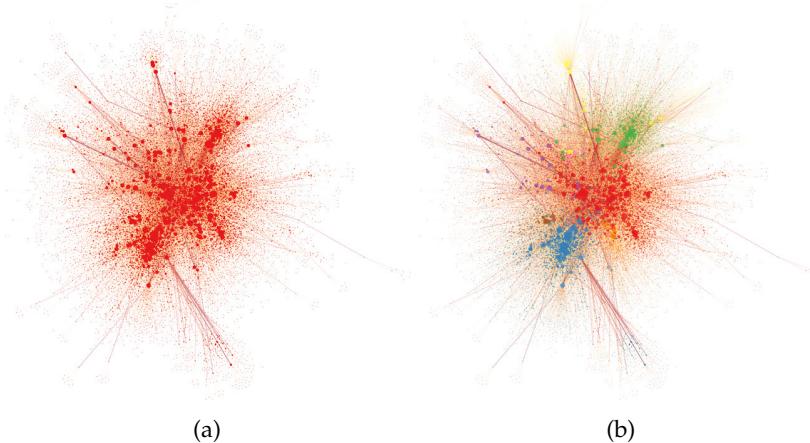


Figure 50.7: (a) Using node degree for the node's size of Figure 50.6. (b) Using communities for the node's color.

Node colors. Finally, we discover groups of characters using a community discovery algorithm – see Part X. I tweak it so that it will return no more than nine communities. Again, I trust the qualitative color palette that Color Brewer provides, I'm partial to Set1 – even if it is not exactly color blind friendly. See Figure 50.7(b) for the final result.

Figure 50.7(b) still has a long way to go before we can call it a good network visualization. But, compared to the starting point in Figure 50.4(a) we can definitely say more things about its structure. Which is exactly what network visualization is for.

A way to improve this picture would be to choose a better layout, and to apply some tweaks to it. That is the topic of next chapter.

50.4 Summary

1. Edge visual elements should be paired, whenever possible, with the same semantics as node visual elements. The thickness of an edge should be proportional to the same – or a similar – variable determining node size. If node size is its betweenness centrality, edge size should be its edge betweenness.
2. Differently from node colors, edge colors are used mostly for quantitative attributes. Usually, there are more edges than nodes in a network, thus it is harder to limit the number of edge colors. Anyhow, you should not have more than nine different colors in total, whether they are node or edge colors. Classical qualitative edge colors choices are layers or link communities.
3. You should try to collapse reciprocal edges in a directed network into a single edge with arrows pointing in both directions. You should use line style very sparingly, only for specific scenarios like black and white printing.
4. A classical workflow to improve your network visualization is to determine edge and node visual attributes in this order: edge transparency, edge width, edge color, node size, and node color.

50.5 Exercises

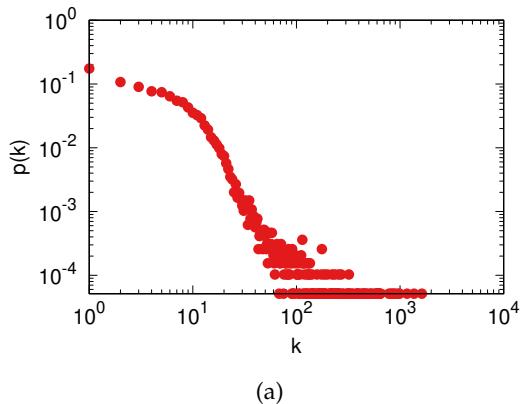
1. Build on top of your visualization from exercise #2 in Chapter 49. Assign to edges a sequential color gradient and a transparency proportional to the logarithm of their edge betweenness (the higher the edge betweenness the more opaque the edge).
2. Import edge data from <http://www.networkatlas.eu/exercises/50/2/edges.txt> and use the attribute for the edge's width.

51

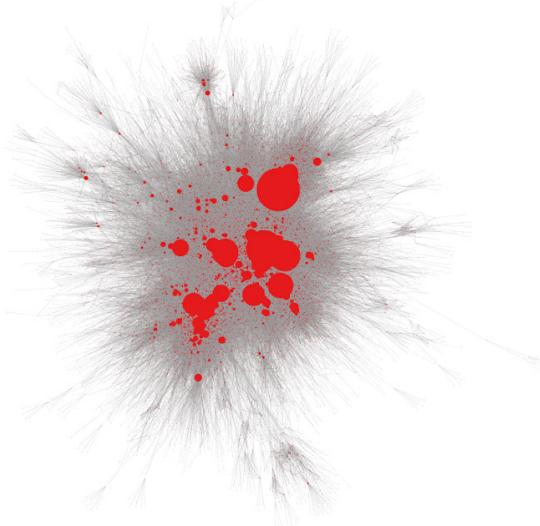
Network Layouts

The network layout is the algorithm that decides where to place each node. This is usually a result of the other nodes to which it is connected. That is why I dub this process “the art of scattering nodes around”. Such art has a queen: Mary Northway¹, the first person who realized – in 1940! – that displaying nodes accurately is a must if one wants to parse social networks.

¹ Mary L Northway. A method for depicting social relationships obtained by sociometric testing. *Sociometry*, pages 144–150, 1940



(a)



(b)

Changing graphical elements as we saw in the previous chapters is good, but network data has a peculiarity that other data types don't have. Networks are a particular data type that allow our representations an additional degree of freedom. This influences the network visualization. To see what I mean consider that, in a scatter plot, you cannot move the points around because that would change the data. But in a network you have connections. What counts is not the “absolute” position of a node, but its “relative” one. Figure 51.1 shows an

Figure 51.1: (a) In a scatter plot, changing the x-y coordinates of a point is forbidden, because that will change the data. (b) In a network, you can move nodes around, as long as you do it with a consistent set of rules to all nodes.

example of what I mean.

The aim of this chapter is to present a few of the classical network layouts, trying to provide a rough guide on what should be used when – although this is sometimes subtle and a matter of personal preferences.

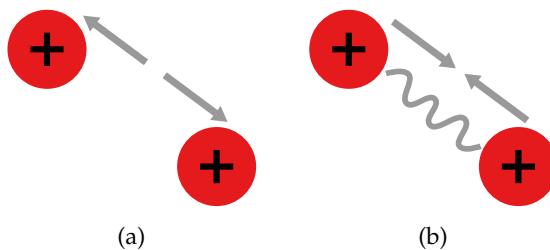
There is a general issue you should be aware of: the vast majority of network layouts were developed with single layer networks in mind. However, they are commonly used for tasks that go beyond these types of networks. For instance, we will see that one common layout principle is the force directed one. People have been applying it to more complex structures such as hypergraphs^{2,3}, or multilayer networks⁴.

Unfortunately, there is not much visual research that I am aware of in these subfields. How to visualize a hypergraph or a multilayer network is a problem with its own challenges and they are both in need of finding their own conventions. The conventions I used when talking about these structures, specifically in Sections 7.2 and 7.3, are a good starting point. Otherwise, you should check out some recent surveys⁵.

Another entry in the “weird network types that we don’t really know how to visualize” is high order networks – see Chapter 34. As far as I know, HoNVis⁶ is the only technique occupying the high order network visualization niche, so you might as well just read that paper.

51.1 Force-Directed

By far, the layout you will see in network science papers most often is the force directed layout. There are many variants of the same basic principle, but here I’ll limit myself to explain the mechanics underlying most of them, and provide four examples you can find implemented in Cytoscape.



The basic principle underlying any force directed method is that nodes should try to repel each other so that they do not overlap, as you can see in Figure 51.2(a). However, connected nodes should be

² Erkki Mäkinen. How to draw a hypergraph. *International Journal of Computer Mathematics*, 34(3-4):177–185, 1990

³ Naheed Anjum Arafat and Stéphane Bressan. Hypergraph drawing by force-directed placement. In *International Conference on Database and Expert Systems Applications*, pages 387–394. Springer, 2017

⁴ Manlio De Domenico, Mason A Porter, and Alex Arenas. Muxviz: a tool for multilayer analysis and visualization of networks. *Journal of Complex Networks*, 3(2):159–176, 2015c

⁵ Fintan McGee, Mohammad Ghoniem, Guy Melançon, Benoît Otjacques, and Bruno Pinaud. The state of the art in multilayer network visualization. In *Computer Graphics Forum*, volume 38, pages 125–149. Wiley Online Library, 2019

⁶ Jun Tao, Jian Xu, Chaoli Wang, and Nitesh V Chawla. Honvis: Visualizing and exploring higher-order networks. In *2017 IEEE Pacific Visualization Symposium (PacificVis)*, pages 1–10. IEEE, 2017

Figure 51.2: (a) Nodes behave like particles of the same charge repelling each other. (b) Edges behave like springs, bringing connected nodes together.

closer to each other, to represent relatedness using spatial proximity. The way this is usually implemented is to consider nodes to have the same magnetic sign – creating the repulsive force. To bring connected nodes together, edges are – rather than strings – springs. The spring wants to be as short as possible and so it will pull connected nodes close together – see Figure 51.2(b). Usually, springs are stronger than the repelling charge force at long distances, but have a minimum length, so that when the nodes are very close together they will not overlap. Of course, as soon as you add more nodes to the mix this gets pretty complicated, as nodes pull other nodes in different directions and so springs overstretch, even if they’re stronger than charges.

Different force-directed network layouts include Prefuse⁷, yFiles organic⁸, regular⁹ and compound spring embedded¹⁰, Fruchterman-Reingold¹¹, simulated annealing¹², GME¹³ and a bunch that I’m probably forgetting. They’re all implemented in either Cytoscape or igraph. They usually differ in the strength and length they give to charges and springs, or in the strategy they apply to find the configuration in which charges and springs are the least stressed, i.e. the system has the minimal possible residual energy.

Of these, I provide a few visualizations of the typical results you’d get in Cytoscape with its default parameters choices. They appear to be quite similar, with a few differences. From the strongest to the weakest charges: spring embedded (Figure 51.3(a)), Prefuse (Figure 51.3(b)), yFiles organic (Figure 51.3(c)), and compound spring embedded (Figure 51.3(d)).

Spring embedded (Figure 51.3(a)) tends to shoot nodes in the stratosphere. It highlights clusters and central backbones better, but nodes tend to overlap. Note how the purple community here looks central. We’ll see it changing its relative position in the other layouts, a sign that even different flavors of force directed can communicate different messages about the centrality of nodes and/or communities.

Prefuse (Figure 51.3(b)) is the bread and butter of network visualization. It isn’t particularly good, but works ok in most cases, and tends to be computationally less expensive than the other force directed alternatives. You will find Prefuse to be the default choice in many cases, in Cytoscape for instance. It deploys a classical balance between charges and springs. Note how the purple community isn’t as central here as it was in Figure 51.3(a).

yFiles organic (Figure 51.3(c)) is very similar to Prefuse force directed. It tends to cluster nodes a bit more snugly, usually works a bit better for more complex networks than the one in Figure 51.3. It is my default choice, unless the network is too big, since the yFiles organic algorithm has a higher computational complexity.

⁷ Jeffrey Heer, Stuart K Card, and James A Landay. Prefuse: a toolkit for interactive information visualization. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 421–430. ACM, 2005

⁸ Roland Wiese, Markus Eglspacher, and Michael Kaufmann. yfiles—visualization and automatic layout of graphs. In *Graph Drawing Software*, pages 173–191. Springer, 2004

⁹ Tomihisa Kamada and Satoru Kawai. A simple method for computing general position in displaying three-dimensional objects. *Computer Vision, Graphics, and Image Processing*, 41(1):43–56, 1988

¹⁰ Ugur Dogrusoz, Erhan Giral, Ahmet Cetintas, Ali Civril, and Emek Demir. A layout algorithm for undirected compound graphs. *Information Sciences*, 179(7):980–994, 2009

¹¹ Thomas MJ Fruchterman and Edward M Reingold. Graph drawing by force-directed placement. *Software: Practice and experience*, 21(11):1129–1164, 1991

¹² Ron Davidson and David Harel. Drawing graphs nicely using simulated annealing. *TOG*, 15(4):301–331, 1996

¹³ Arne Frick, Andreas Ludwig, and Heiko Mehldau. A fast adaptive layout algorithm for undirected graphs (extended abstract and system demonstration). In *International Symposium on Graph Drawing*, pages 388–403. Springer, 1994

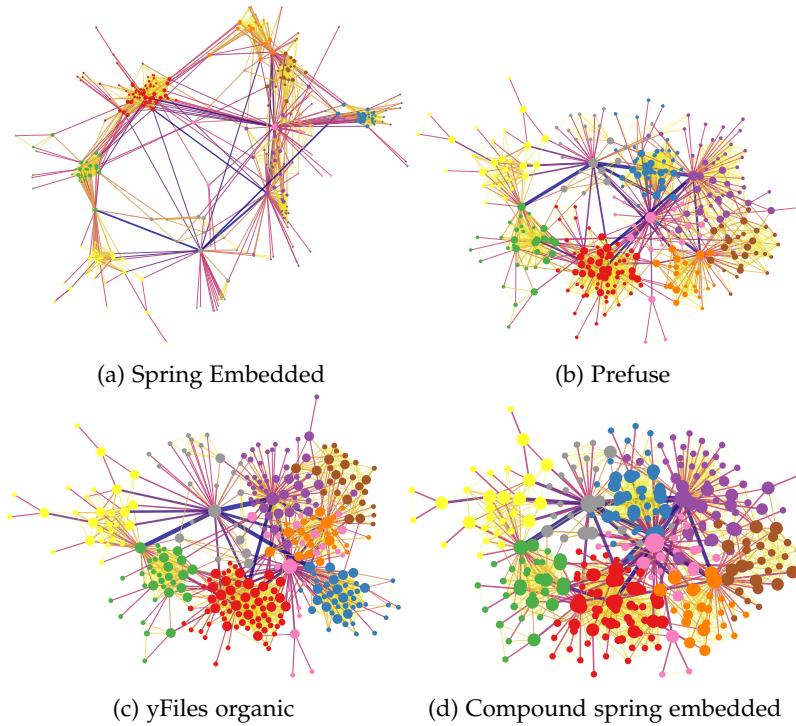


Figure 51.3: The same network displayed with different flavors of force directed layout. Node colors are their communities, node sizes are their degree. Edge colors and sizes are proportional to their betweenness centrality.

Compound spring embedded (Figure 51.3(d)) is at the opposite end of the spectrum when compared to spring embedded. It forces nodes to be more or less equidistant from each other. It is a good option if the nodes are all more or less equivalent, but plays badly once you have diverse node sizes. You can see a lot of nodes overlapping in Figure 51.3(d).

My rule of thumb here is that, the more complex interconnections you have the more you want your clusters – if you have any – to be separated. So you would choose the spring embedded layout. On the other hand, well balanced and separated clusters, with nodes more or less on equal footing, will mean going to the opposite end of the spectrum, to the compound spring embedded.

Note that Cytoscape's implementation of layout algorithms is not the most efficient. However, as most network plotting programs, it will accept you to pass x and y coordinates as attributes of your nodes, which you can use to display them. One trick is to calculate the layout using a more efficient script – for instance igraph in R or C – and then import the result in Cytoscape. Alternatively, remember that you can have a force directed style layout even without applying the force directed algorithm. Section 43.4 taught you how to use node embeddings to determine the placement of nodes on a 2D space.

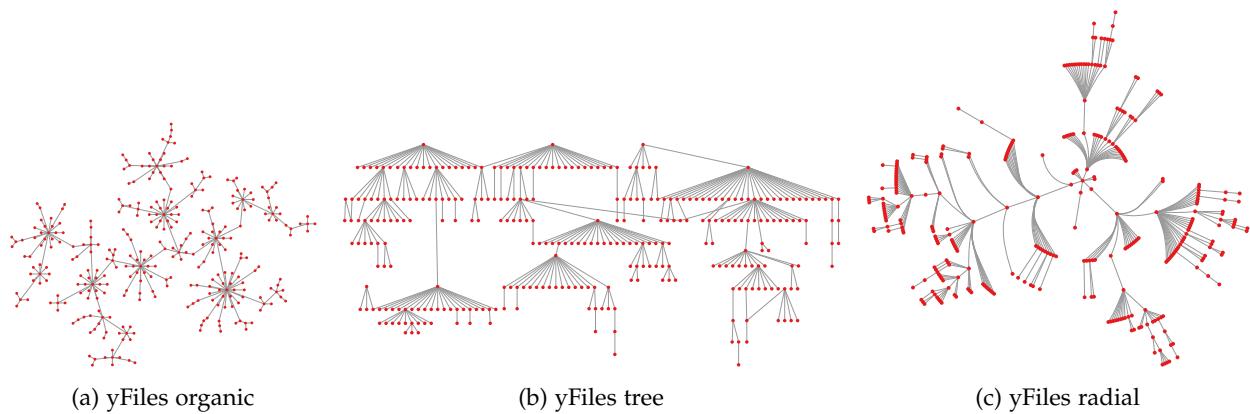
Force directed is a good choice for sparse to medium-sparse networks. It works well when you have well defined clusters: any

additional density coming from quasi-cliques is well handled because the nodes will bunch up in a ball together no matter what. Problems arise when the additional density comes from interconnections spanning across clusters. Also, it tends to fit your network onto a virtual sphere: its layouts tend to be circular. This is not always the best choice, as some networks are going to fit different shapes better.

51.2 Other Node-Link Layouts

Hierarchical

If force directed and its variants are a good default choice, they are not the only way to display your networks. As I concluded in the previous section, they have some pretty limited use cases. What happens when we go out of those use cases?



The first scenario we consider is the one of extremely sparse networks. In that case you can obviously use force directed. However, it can arguably be not the better choice. Consider Figure 51.4(a): even for this extremely sparse graph, the layout still manages to have many awkward node-edge overlaps. This is because very sparse networks are similar to trees, and a tree hardly fits the assumption of a circular layout. A tree has a root and leaves, thus a inherent top-down flow, which doesn't fit well the "in-out" flow of a circle (from the center to the circumference).

In this case you want to use a tree layout as I show in Figure 51.4(b). A popular variant would be a radial layout – Figure 51.4(c). The radial layout is a compromise that still respects the tree-like structure of sparse graphs and, at the same time, has a force-directed "in-to-out" flavor to it.

As soon as your network becomes more dense than a quasi-tree, you should probably stop considering the hierarchical layouts. In

Figure 51.4: The same network displayed with different layouts, force directed and hierarchical.

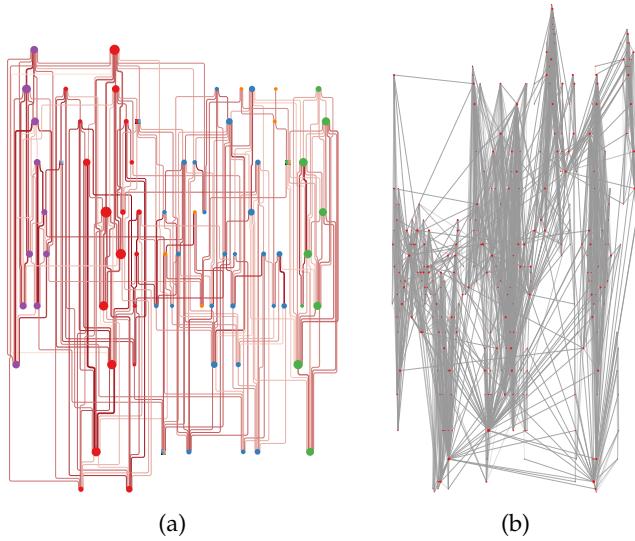


Figure 51.5: (a) A dense network still manageable with a hierarchical layout. (b) An example of a network too complex to be displayed with a hierarchical layout

some extreme cases they can work, if you want to highlight some specific messages. For instance, I would not use it for the network in Figure 51.5(a), but I can see how it can communicate something about the clusters of the network. The layout manages to put nodes in the same community in the same column, showing how some communities have stronger – darker, thicker – connections than others. However, a non-trivial number of nodes and edges would make your network visualization completely unintelligible, as it happens in Figure 51.5(b).

Circular

A second scenario to consider is the case of an extremely dense network. The network could be so dense, that the force directed is not able to pull nodes apart and show structure. In this case, the first step of the solution involves considering a layout that might not seem the best for the job, but has a few tricks up its sleeve: the circular layout. Which is exactly what it sounds: it places nodes on a circle, equidistant from one another.

The first part of the trick in using circular layouts is not to display the nodes in a random order, but choosing an appropriate one. Ideally, you want to place nodes in bunches such that most connections happen across neighbors. Usually this is achieved by identifying the nodes' attribute which groups them best. You can also run a custom algorithm deciding the order and then provide that as the attribute for the circular layout.

Figure 51.6 shows an example. In the figure you can see that the layout still works: it shows how most connections remain within

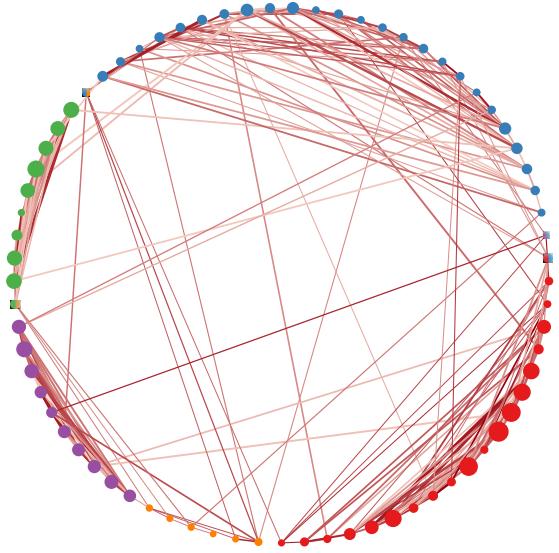


Figure 51.6: A circular layout.

the communities, and clearly points at how many and where the inter-community connections are. In a force directed layout, these connections would stretch long and be forced in the background of denser areas, with the effect of being difficult to appreciate. However, the real kicker for circular layouts happens when you consider the use of an additional visual feature: edge bends.

51.3 Edge Bends

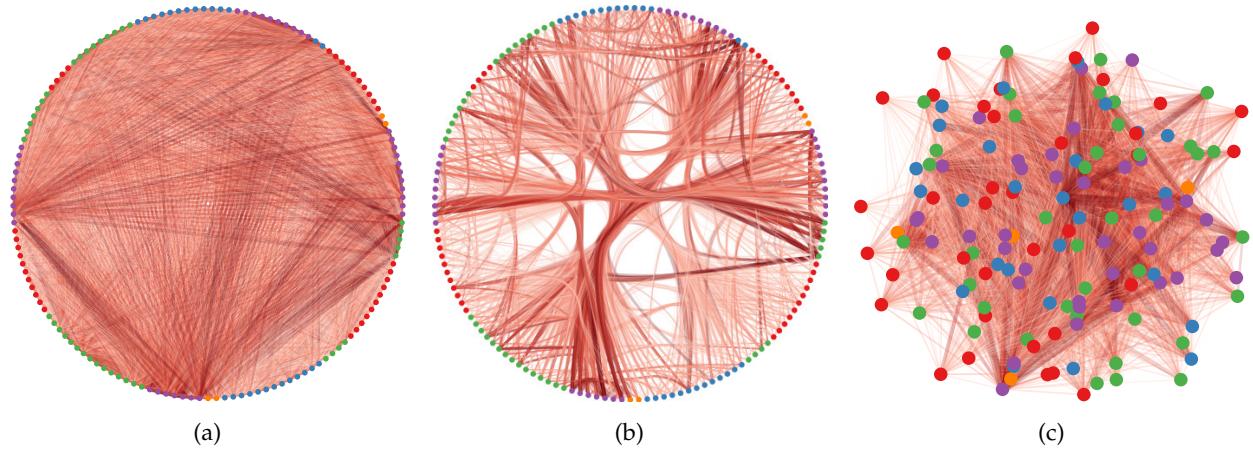
The default choice in network layouts up until recently was to show edges as straight lines. However this is usually not pleasing visually. Go back to Figure 51.3(a) and you will feel how clunky the long thick edges feel. People started playing with edge bends and discovered that they can solve a series of other problems, rather than being a simple cosmetic enhancement.

The first strategy to bend your edges is to bundle together the ones coming/going from/to more or less the same part of the network^{14,15}. This is extremely useful for the very dense networks in circular layouts I teased at the end of the previous section¹⁶. Consider Figure 51.7(a): here the circular layout isn't helping us much in making sense of this extreme density. However, once we bundle edges in Figure 51.7(b), we obtain a much clearer idea of what is going where. Sure, the visualization is still complex and it is still hard to tell which nodes are connected to which other. However, that is a much better situation than the alternative visualization you would get from a force directed without edge bends. Figure 51.7(c) is my proof.

¹⁴ Danny Holten. Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data. *IEEE Transactions on visualization and computer graphics*, 12(5):741–748, 2006

¹⁵ Danny Holten and Jarke J Van Wijk. Force-directed edge bundling for graph visualization. In *Computer graphics forum*, volume 28, pages 983–990. Wiley Online Library, 2009

¹⁶ Emden R Gansner and Yehuda Koren. Improved circular layouts. In *International Symposium on Graph Drawing*, pages 386–398. Springer, 2006



You can use edge bundling in other layouts too. Usually this will reduce clutter and make your visualization clearer and easier to parse. For instance, in a force directed layout communities will reduce to flower bouquets where all connections collapse in the same point. See Figure 51.8(a) for an example. This is as if the community is a hyperedge connecting all the nodes that it groups. This is not exactly the full topological information in the network, but oftentimes it is an acceptable approximation.

A hierarchical network layout benefits from edge bends as well – see Figure 51.8(b). This is because it is difficult to display hierarchies in a tight space, they tend to grow horizontally and vertically. With edge bends you can make your edges take a slightly longer route which increases the picture’s readability. Of course, a bend can decrease the readability if you are then unable to clearly distinguish which edge goes where in a bundle. So there are techniques to make this distinction¹⁷.

Bending edges helps in a second scenario. When squeezing complex multidimensional structures, such as dense networks, onto a two dimensional plane, you’ll often have no good placement choice for your nodes and edges. In many cases this is just awkward and displeasing to the eye, but in many others it can create problems and untruthful visualizations. The most dreaded case you should avoid at all cost is ghost edges.

Consider Figure 51.9. Suppose we know that the network’s topology involving those three nodes is on the left. Node 1 is connected to node 2 which is connected to node 3 in a chain. However, these three nodes don’t live in a vacuum. There are thousands of other nodes and edges around them, pulling and pushing them in different directions. After a few layouts, you might not notice that your three nodes ended up in the configuration to the right.

Figure 51.7: The same network displayed with different layouts: (a) circular with no bends, (b) circular with edge bundling, (c) Prefuse force directed.

¹⁷ Benjamin Bach, Nathalie Henry Riche, Christophe Hurter, Kim Marriott, and Tim Dwyer. Towards unambiguous edge bundling: Investigating confluent drawings for network visualization. *IEEE transactions on visualization and computer graphics*, 23(1):541–550, 2016

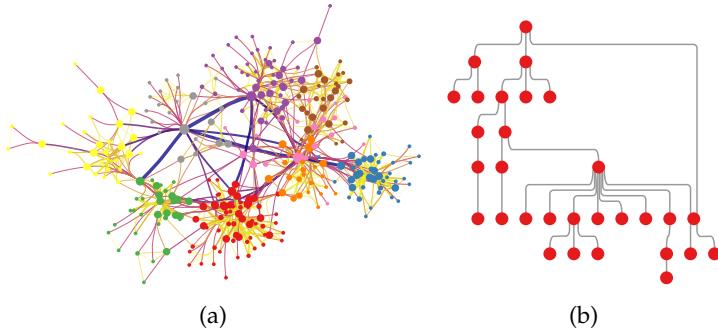


Figure 51.8: Edge bundles in (a) force directed and (b) hierarchical network layouts.

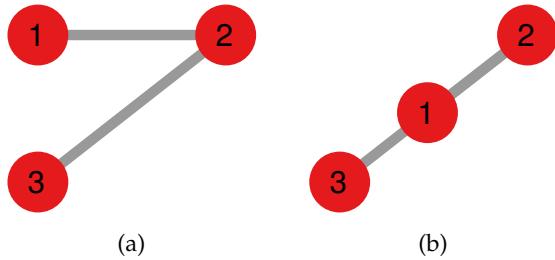


Figure 51.9: A scenario in which you might create ghost edges.

Seeing that configuration, a viewer would instantly assume that node 1 is connected to both node 2 and 3, without a connection between the latter two. This, as we know, is wrong. Worse still, there's no way by looking at the configuration to the right to know what really is going on between these three nodes. It might be that node 1 is not connected to either of those, and ended up there by accidents of your layout. Or that could be a squeezed triangle. Or it might be even true that the three nodes are not connected at all to each other, and that's just a long edge connecting two other nodes out of sight. There are so many ways to lie in a 2D network layout – whether you do it accidentally or on purpose.

Edge bends can partially save you. In particular, the trick is to use organic or orthogonal edge routing¹⁸, implemented by yFiles. To know what it looks like, consider Figure 51.10. In the figure, there are many cases in which the vanilla force directed layout would pass a straight edge across the nodes. As it happens, some of those cases were actually two edges with a node in between. The organic edge router would not change the edge shape in that case. But some edges get a dramatically evident bend, because the vanilla force directed would make you believe they connected nodes while, in reality, they did not.

¹⁸ Tim Dwyer, Kim Marriott, and Michael Wybrow. Integrating edge routing into force-directed layout. In *International Symposium on Graph Drawing*, pages 8–19. Springer, 2006

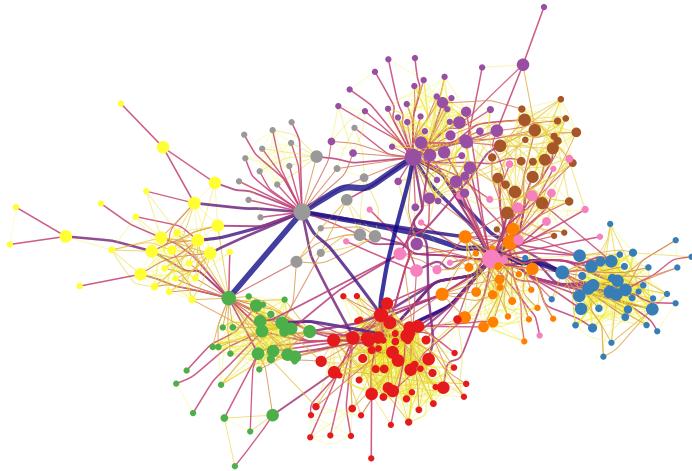


Figure 51.10: A force directed layout with organic edge routing.

51.4 Alternative Layouts

In this section, we break the assumption that nodes are circles and edges are lines. We try to find weird ways to summarize the network topology in a way that is more compact and compelling.

Matrix Layouts

What's the last resource for networks in which the density is too high even for a circular layout plus edge bundles? If your network is so dense, then you don't have a network: you have a matrix and you should visualize it as such. In these cases, what matters more is not really which area is denser than which other, but which blocks of nodes have connections with higher and lower weights.¹⁹

¹⁹ If your network is this dense and *also* unweighted, consider changing job.

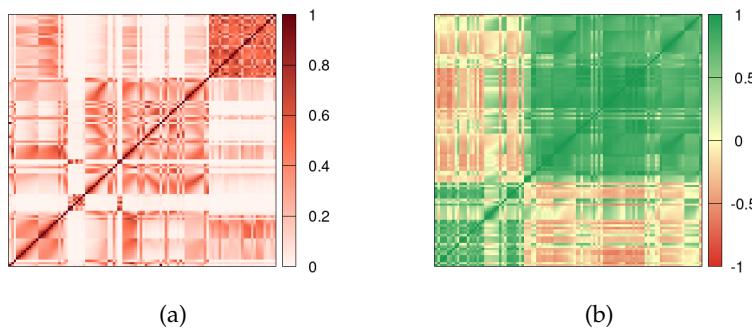


Figure 51.11: (a) A matrix view of a network with progressive edge weights. (b) A matrix view of a network with divergent edge weights, such as correlations.

Color scales should be chosen depending whether your edge weights are defined as progressive or divergent. The first case, in Figure 51.11(a), is the classical case of an edge indicating the intensity of the connection between two nodes, or the cost of edge traversal. The second case, in Figure 51.11(b), is a classical correlation network.

This is a default visualization scenario, as correlations are always defined between any pair of nodes, and thus the network will be complete.

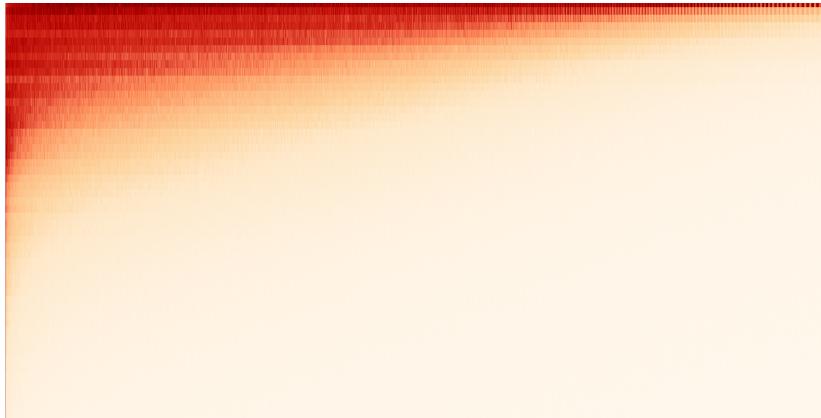


Figure 51.12: A matrix view of a nested network.

That is not to say that this is the only use case of a matrix visualization. Even for sparser networks, sometimes a matrix is worth a thousand nodes. Consider the case of nestedness (Section 32.4), a particular core-periphery structure for bipartite networks where you can sort nodes from most to least connected. The most connected node connects to every node in the network, while the least connected nodes only connect to the nodes that everyone connects to. This sort of linear ordering naturally lends itself to a matrix visualization. While a node-link diagram would make a mess of such a core, rendering the message difficult to perceive, a matrix view is deceptively simple, as I show in Figure 51.12.

This case is a good example of the main problem in visualizing networks as matrices: the order of the rows/columns you choose is the most important thing. You should put your nodes in the sequence that highlights the crucial structural characteristics the best. In the nestedness case, nodes are sorted by degree (or total incoming weight sum). If your network has communities, you want to have nodes next to their community mates. This creates the classical block diagonal matrices. There are other criteria you might want to consider²⁰.

Hive Plots

The issue with all traditional node-link diagrams is that the position in space of a node is arbitrary: it does not reflect its properties, but it is just relative to its connections. As such, layouts are not reproducible, because a small change in the initial conditions in the

²⁰ Michael Behrisch, Benjamin Bach, Nathalie Henry Riche, Tobias Schreck, and Jean-Daniel Fekete. Matrix reordering methods for table and network visualization. In *Computer Graphics Forum*, volume 35, pages 693–716. Wiley Online Library, 2016

placement of a single node will result in a completely different layout. Hive plots²¹ try to fix these issues by providing a way to have a deterministic node layout placement.

The idea is the following: first, the user determines a set of rules. The aim of these rules is to divide nodes into classes. Nodes in the same class will be grouped together on the same axis. Then, the user selects a specific measure, which determines the position of a node on that axis. The hive plot will then attempt to place the axes in such a way to minimize edge crossings. If there are connections between the nodes on the same axis, the axis will be duplicated in order to avoid confusing loops.

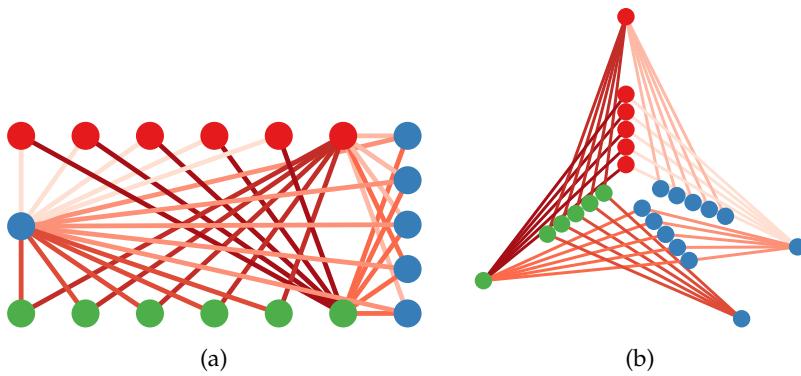


Figure 51.13 shows a simple example. In Figure 51.13(a) we have three node types, which result in three axes in Figure 51.13(b). However, the blue nodes also connect to each other, and have more complex connecting patterns with the other groups. Thus, we duplicate the blue axis, meaning that we have a copy of it at slightly different angles. The blue-blue links flow between the copies, and we divided the connections between blue and other nodes as to minimize the number of crossings.

The position of nodes on the axes is determined by the degree. The nodes with high degree on each axis are the ones on top, separated by the rest. The other nodes have all the same degree, so they are grouped, although we separate them so that the nodes don't overlap with each other. You can choose which measure to use for the node positioning on the axis, you are not forced to use the degree.

Graph Thumbnails

Just like hive plots, also graph thumbnails²² aim to provide a deterministic layout, where two isomorphic graphs result in the same visualization. In graph thumbnails, we decide to give up the ability of analyzing local structures. We are not seeing each individual node: the visualization is a summary of the graph's global structure. The

²¹ Martin Krzywinski, Inanc Birol, Steven JM Jones, and Marco A Marra. Hive plots – rational approach to visualizing networks. *Briefings in bioinformatics*, 13(5):627–644, 2011

Figure 51.13: (a) A graph where I encode with colors the node and edge types. (b) The hive plot version of (a).

²² Vahan Yoghoudjian, Tim Dwyer, Karsten Klein, Kim Marriott, and Michael Wybrow. Graph thumbnails: Identifying and comparing multiple graphs at a glance. *IEEE Transactions on Visualization and Computer Graphics*, 24(12):3081–3095, 2018

idea is to dissect a graph into its main core components in a hierarchical fashion. Each core is then visualized as a circle, whose color tells us its core level. You should use graph thumbnails when you need to compare a large number of graphs in a compact way and you care about the high-level organization of the graph as a whole, rather than the meso-level communities – or the individual nodes.

The decomposition is done via the classical k-core detection – see Section 14.7. Each connected component of a network is part of a 1-core. Then, there could be multiple k-cores around the network. Each k-core is represented as a circle, and it is nestled inside the $(k - 1)$ -core that contains it.

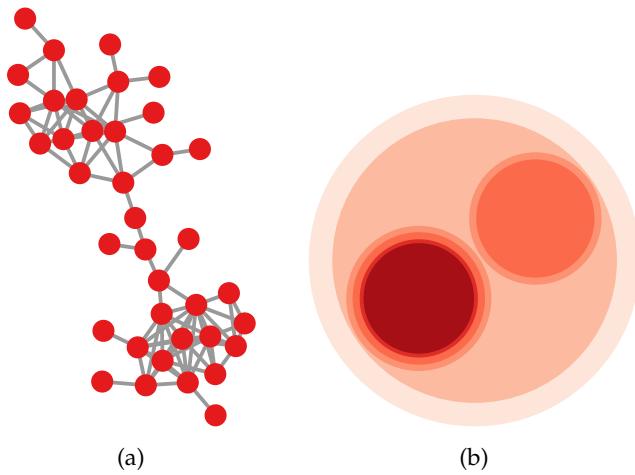


Figure 51.14: (a) A graph and its (b) graph thumbnail visualization. The color of the circle encodes the k value of the k -core, while its size is (loosely) proportional to the number of nodes in that particular core.

Figure 51.14 shows an example. The graph in Figure 51.14(a) has two communities. All nodes in the graph are part of the 1-core because it's a single connected component. Some nodes are not part of the 2-core, but they are only the peripheral dangling ones: both communities are part of the same 2-core. The communities split when we consider the 3-core, that is why the second circle in the graph thumbnail in Figure 51.14(b) contains two subcircles. The smallest community only contains a 4-core, while the largest goes up to a 6-core, explaining why the second circle goes to darker hues.

A disadvantage of the graph thumbnail visualization is that it needs to apply a circle packing algorithm²³. It is impossible to pack circles efficiently inside other circles. In this specific example, both the 1- and the 2-core circles are larger than they should be given the number of nodes they contain. I needed to enlarge them, because otherwise they could not contain properly the two 3-cores of the network.

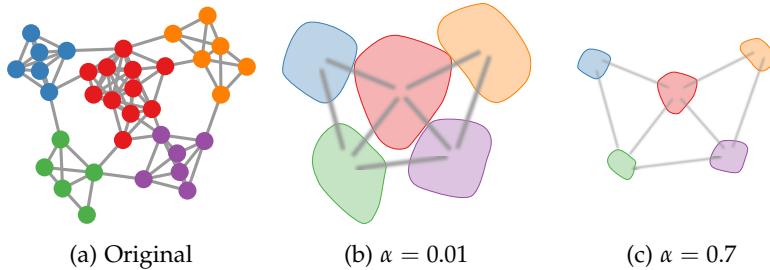
²³ Charles R Collins and Kenneth Stephenson. A circle packing algorithm. *Computational Geometry*, 25(3):233–256, 2003

Probabilistic Layout

Following the same “we can’t visualize all nodes” philosophy of graph thumbnails, we have probabilistic layouts²⁴. This technique is handy when you have a generic guess of where the nodes *should* be, but you cannot draw them all. You should use such layouts especially for very large graphs that you couldn’t visualize otherwise, because they have too many nodes and/or edges.

The idea is as follows. First, you sample the nodes in your network, taking only a few of them. Then you calculate their positions using a deterministic force directed layout. You repeat the procedure multiple times, obtaining, for each node, a good approximation of where it should be. If you have nodes that you never sampled, you can reasonably assume that they are going to be in the area surrounding their neighbors. Since you’re applying the algorithm to a sample, this won’t take much time even if the original network was too large to be analyzed in its entirety.

Now each node is associated to a spatial probability distribution, much like elementary particles in quantum physics. You can assume that, if the node is anywhere, it’ll be somewhere in the area where its probability is nonzero. At this point, you can merge nodes whose spatial probabilities overlap, by detecting and drawing a contour containing them. You should then bend edges and smudge them as well, to reflect the uncertainty of where their endpoints are.



²⁴ Christoph Schulz, Arlind Nocaj, Jochen Goertler, Oliver Deussen, Ulrik Brandes, and Daniel Weiskopf. Probabilistic graph layout for uncertain network visualization. *IEEE transactions on visualization and computer graphics*, 23(1):531–540, 2016

Figure 51.15: A graph and its probabilistic layouts, for different levels of α .

You can specify a parameter α regulating how tight your smudges should be. For low values of α , you get the quickest results at the price of large uncertainties. When $\alpha \sim 1$, your smudges become points, tightening up all nodes belonging to a smudge in the same area. Figure 51.15 shows a toy example, for two levels of α .

Revealing Matrices

One key visualization technique is scatterplot matrices or SPLOMs. When you have multiple variables in your dataset, you might be interested in knowing which one correlates with which other. So you can create a matrix where each row/column is a variable, and each

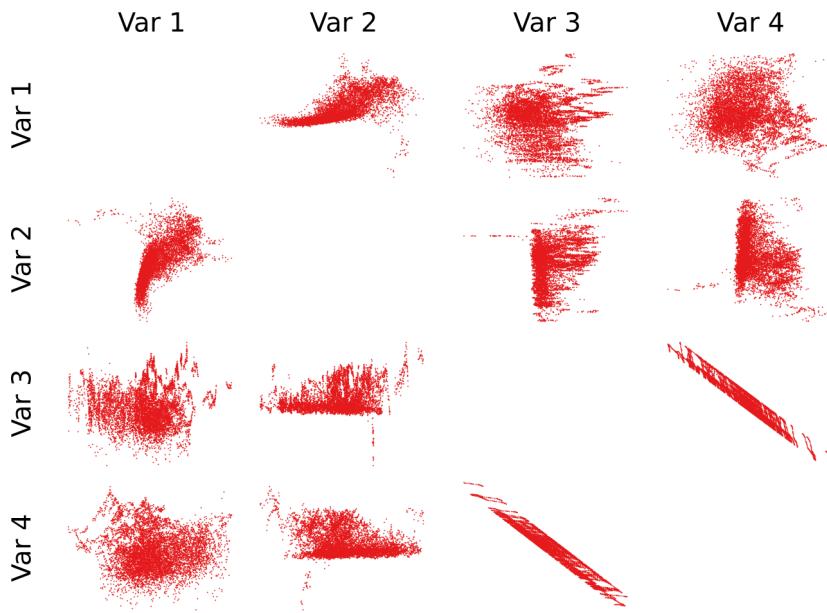


Figure 51.16: An example of SPLOM visualization.

cell contains the scatter plot of the row variable against the column variable. Figure 51.16 shows an example.

The same visualization technique can be applied to networks. In revealing matrices²⁵, each row/column of your matrix is an entity. Then, each cell of the matrix contains a bipartite network, where the nodes of one type are the row entity and the nodes of the other type are the column entity.

One defect of SPLOMs is that the main diagonal of the matrix is a bit awkward. In it, the row variable and the column variable are the same. Thus the scatter plot is meaningless, as it is the same variable on the x and y axes: a straight line. One could modify it by showing some sort of statistical distribution of the variable, but that would mean breaking the axis consistency of the SPLOM. For this reason, the main diagonal of a SPLOM is often omitted.

This defect does not apply to the revealing matrices visualization. The main diagonal in this case is well defined: it is simply the direct relationship between entities of the same type. Thus, it contains a unipartite network per node type in your database.

Timelines

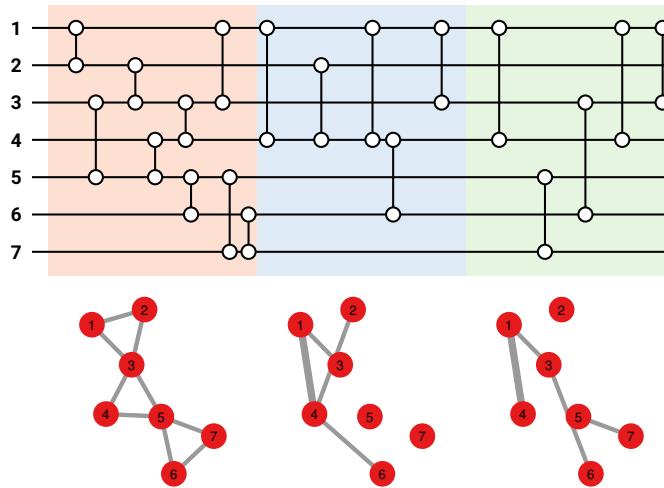
Temporally evolving networks are tricky to visualize. The most natural thing you can do is to use animations, showing a state of the network per frame, but that's unsatisfying. First because you can't put them on a piece of paper. Second because you cannot have

²⁵ Maximilian Schich. Revealing matrices. 2010

an overview of the dynamics all at once; you need to wait for the animation to play out and you might have forgotten what was in the first frame by the time you get to the last.

There is one visualization technique I first saw in 2012²⁶ (but it could be older) that changes fundamentally how we visualize a network to show time in a more natural way. Figure 51.17 shows an example.

²⁶ Petter Holme and Jari Saramäki. Temporal networks. *Physics reports*, 519 (3):97–125, 2012



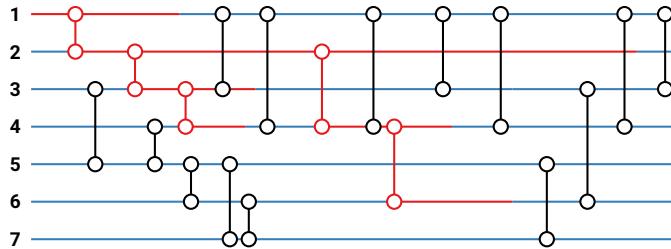
In the figure a node becomes a horizontal timeline. Time flows from left to right, as we normally assume by convention. When two nodes are connected, we join the corresponding timelines with a vertical line. So, in this visualization, a node is a horizontal line and an edge is a vertical line. You can easily see in the figure that nodes 1 and 4 have a high level of activity and node 7 is low activity, something that might be tricky to do otherwise. You may argue that this is a gimmick that only works for a handful of nodes and edges, but so are node-link diagrams, so checkmate there.

One neat thing that this visualization allows you to do is to keep track of events on the network. For instance, consider an SIS model – which I show in Figure 51.18. From the figure you can already infer that, once the disease is exclusively in node 2 it is trapped and has nowhere to go in the future, because node 2 won't interact with any other node. This kind of inference might be harder and/or slower to do visually with a different visualization.

51.5 Case Studies

As it often happens in data visualization, what you want to say with your network visualization might not be supported by any standard visualization technique out there. Sometimes, you need to craft a

Figure 51.17: A network timeline visualization: nodes as horizontal lines, edges as vertical lines. Each shaded area shows one snapshot and the corresponding classical node-link diagram visualization below it.



custom visualization, bending and breaking rules along the way. No one should really follow your workflow, because it applies only to your specific aim with your specific data. However, seeing some of these examples could be helpful in making you realize that you are not mad: sometimes you really do know better than everybody else. The aim of this section is to empower you in being daring: try to look at your data and your communication objective, and create your way to bringing them together.

I touch on two examples I worked on. These are custom ways of displaying a node-link diagram that I found useful. These node-link diagrams have special configurations given the need to highlight specific features of the networks they represent. Of course, there's much more out there, but these are two cases I'm familiar with.

Product Space

The Product Space^{27,28} is a popular example. The Product Space is a network in which each node is a product that is traded among countries in the global market. Two products are connected if the sets of countries exporting them have a large overlap. The idea of this visualization is to show you which products are similar to each other, because if your country can make a given set of products, via the Product Space it can figure out which are the most similar products it should consider trying to export.

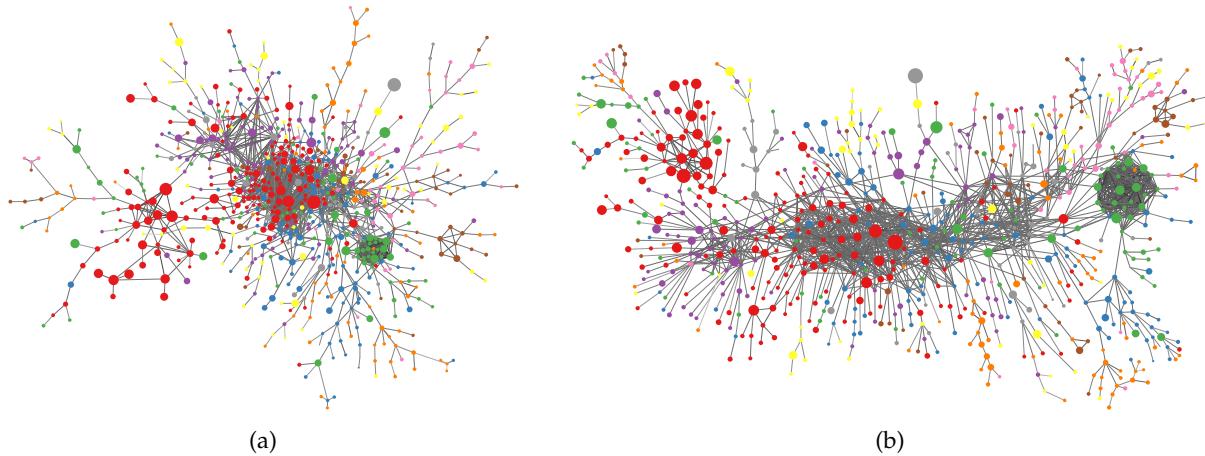
The original way to try and visualize the Product Space was a simple force directed layout, as I show in Figure 51.19(a). However, as I mentioned previously, the force directed layouts have this tendency of forcing your networks on a sphere. This happens to work really poorly in the case of the Product Space. The reason is that not all products are the same. Some products are harder to export than others. This is a key concept in the original research, known as Economic Complexity.

This means that the Product Space has an inherent "direction". Countries want to move from simple to more complex products, as the latter is a more rewarding category to be able to export. However, the circle has no direction. It is a loop: you always get back to where

Figure 51.18: Network timeline visualization of an SIS process. The node's line is red when the node is infected (I) and blue when it is susceptible (S). A contagion happens probabilistically when a red line joins a blue line, when it happens the connection is red. Note node 4 that gets reinfected after recovering.

²⁷ César A Hidalgo, Bailey Klinger, A-L Barabási, and Ricardo Hausmann. The product space conditions the development of nations. *Science*, 317 (5837):482–487, 2007

²⁸ Ricardo Hausmann, César A Hidalgo, Sebastián Bustos, Michele Coscia, Alexander Simoes, and Muhammed A Yildirim. *The atlas of economic complexity: Mapping paths to prosperity*. Mit Press, 2014



you started. The shape of the Product Space in Figure 51.19(a) does not allow us to perceive the development path. That is why it is necessary to stretch out the visualization as I do in Figure 51.19(b): now the Product Space is a (complex, multidimensional) line²⁹ and you can see that there is a clear direction going from right to left, from less to more complex products. It is still a type of force directed layout, but it needed to be customized to remove its inherent circularity.

Cathedral

In another paper of mine, I analyze government networks³⁰. My nodes are government agencies and I establish edges between them if the website of an agency has an hyperlink pointing to the website of another agency. One key question is verifying if this network has a hierarchical organization – see Chapter 33. One obvious way to explore this question is visualizing the network and see if it looks like a hierarchy. Unfortunately, the network is relatively large and dense. So I need to come up with a custom layout. Such layout is useful to visualize dense hierarchical networks, and thus can be considered as an enhancement of the classical hierarchical layout presented earlier, that works only for tree-like structures.

The first step is to group nodes into a 2-level functional classification. This means to assign to each agency the function it performs in the government. For instance, a school is part of the education system (level 1 function) and of the primary & secondary education (level 2 function). Or: a city government is part of general administration (level 1 function) and of the municipal administration (level 2 function).

There are not many level 2 functions so I can collapse all agencies

Figure 51.19: The Product Space. (a) Classical force directed layout. (b) Manually adjusted linear force directed. The node color is a product's Leamer category.

²⁹ Eerily looking like an angel from Neon Genesis Evangelion, with that creepy head with multiple green eyes... Am I the only one seeing it?

³⁰ Stephen Kosack, Michele Coscia, Evann Smith, Kim Albrecht, Albert-László Barabási, and Ricardo Hausmann. Functional structures of us state governments. *Proceedings of the National Academy of Sciences*, 115(46):11748–11753, 2018

into their level 2 function. Then I display these functions in a scatter plot. On the x axis, I report the centrality of the level 1 function. The most central level 1 function is in the middle and, as we get to the edges of the visualization, we get to progressively less central level 1 functions. Now, in this plot, each column contains all level 2 functions belonging to a specific level 1 function. On the y axis I report the centrality of the level 2 function. Functions at the top are more central than functions at the bottom. The result is in Figure 51.20. It looks like a nice hierarchy!

The attack you could do to this visualization is that it might make any network look like a hierarchical network, no matter if it is actually hierarchical or not. That is why I embed a small inset in the top left corner. The network in the inset is the result of a configuration model version of the original network: it has the same number of nodes, edges, and the same degree distribution. The connections are rewired randomly, destroying the hierarchy – if any is present. When I apply the same layout strategy, I obtain the visualization in the inset: there is not a trace of hierarchy any more! This proves that the layout is not showing hierarchies where there are none.

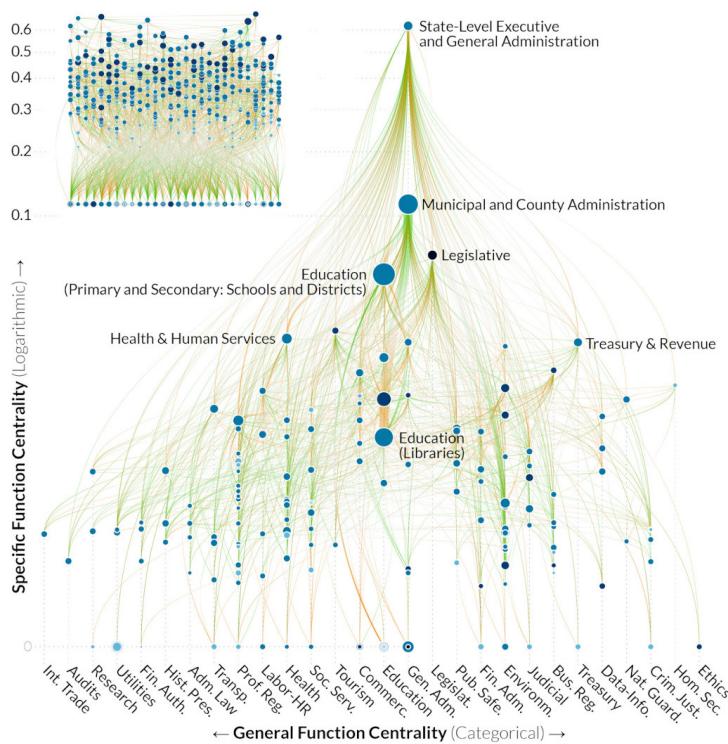


Figure 51.20: The triangular two-level centrality plot I describe in the main text – image by Kim Albrecht.

51.6 Summary

1. A network layout is an algorithm that determines where the nodes of your network visualization should be in a 2D space. The positions of the nodes are determined by the connections between them.
2. The most common principle is the one of the force directed layout. Nodes are charges of the same sign repelling each other and edges are springs trying to keep connected nodes together. This layout works for sparse networks with communities, whose topology fits on a circle.
3. Specialized layouts exist for even sparser networks with a hierarchical organization. Circular layouts can work for denser networks, provided that you use edge bends, bundling edges between nodes located in the same regions of the circle.
4. Edge bends help in many layouts, particularly avoiding the creation of “ghost edges”. These happen when your network layout places a node on top of an edge that is not connected to it, giving the appearance that the node is part of a chain.
5. Node-link diagrams representing nodes as circles and edges as lines are not the only solution. For very dense networks you can show the network as a matrix, as a graph thumbnail via k-core decomposition, or with a probabilistic layout associating a node to a cloud of probability in space.
6. In many cases, your network will have a clear and unique message that has never been visualized before. In those cases, you need to bend rules and create a unique visualization serving your specific communication objective.

51.7 Exercises

1. Which network layout is more suitable to visualize the network at <http://www.networkatlas.eu/exercises/51/1/data.txt>? Choose between hierarchical, force directed, and circular. Visualize it using all three alternatives and motivate your answer based on the result and the characteristics of the network.
2. Which network layout is more suitable to visualize the network at <http://www.networkatlas.eu/exercises/51/2/data.txt>? Choose between hierarchical, force directed, and circular. You might want to use the node attributes at <http://www.networkatlas.eu/exercises/51/2/nodes.txt> to enhance your visualization.

Visualize it using all three alternatives and motivate your answer based on the result and the characteristics of the network.

3. Which network layout is more suitable to visualize the network at <http://www.networkatlas.eu/exercises/51/3/data.txt>? Choose between hierarchical, force directed, and circular. Visualize it using all three alternatives and motivate your answer based on the result and the characteristics of the network.

Part XIV

Useful Resources

52

Network Science Applications

Network science is a vast field, exponentially expanding since the late nineties. There has been so much work on it. This book so far cites more than 1,000 papers, and yet there still an incredible wealth of produced knowledge that did not fit in here. In this chapter I want to give you a taste of what network science can do. The idea is to briefly discuss the main contributions of a handful of classic papers that, for one reason or another, did not find space in the more pedagogical chapters that preceded this one.

Of course, the set of papers discussed here is subjective: it is my own perspective of the field, the papers and contributions on which I stumbled most often while working. Specifically, I am partial to the field of computational social science¹: the use of computer science techniques to study social systems – and humanities in general. I am still deeply embedded in the digital humanities tribe. It is also, ironically, a largely incomplete set. No matter how much effort I pour into this book to make it more exhaustive, it seems that each paper I add simply increases its surface area and makes it less thorough, not more. It's the fractal nature of complex systems, and it's something I will have to live with.

52.1 Network Effects of Innovation

Why is society nudging us to live in cities? Is there an invisible force gluing humans in larger and larger settlements? As a matter of fact, this might very well be true. A research collaboration^{2,3} started investigating these questions by performing a deceptively simple analysis. They took data about as many cities in the world as possible. Then, they made a straightforward plot. They placed the population of the city on the x-axis, and plotted a bunch of other variables in the y-axis.

Then they found a power relation between a city's population and its outcomes. Their plots looked like the ones I show in Figure 52.1.

¹ David Lazer, Alex Sandy Pentland, Lada Adamic, Sinan Aral, Albert Laszlo Barabasi, Devon Brewer, Nicholas Christakis, Noshir Contractor, James Fowler, Myron Gutmann, et al. Life in the network: the coming age of computational social science. *Science (New York, NY)*, 323(5915):721, 2009

² Luis MA Bettencourt, José Lobo, Dirk Helbing, Christian Kühnert, and Geoffrey B West. Growth, innovation, scaling, and the pace of life in cities. *Proceedings of the national academy of sciences*, 104(17):7301–7306, 2007

³ Andres Gomez-Lievano, HyeJin Youn, and Luis MA Bettencourt. The statistics of urban scaling and their connection to zipf's law. *PloS one*, 7(7), 2012

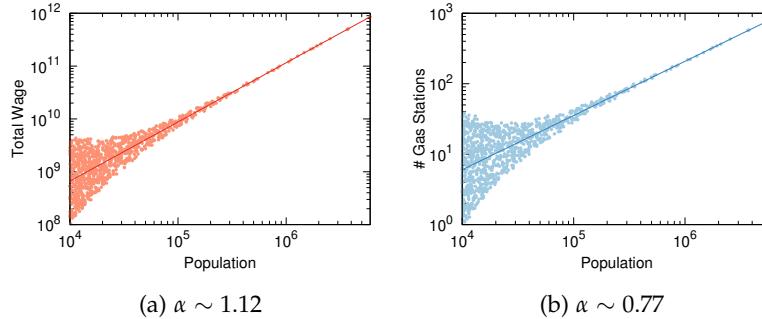


Figure 52.1: (a) Total wage sum (y axis) as a function of a city's population (x-axis). (b) Number of gas stations (y axis) as a function of a city's population (x-axis).

When they looked at their α exponents, they discovered something remarkable. The two plots in Figure 52.1 might seem identical, but they differ in a crucial aspect: the value of the α exponent. Figure 52.1(a) has an $\alpha > 1$, while Figure 52.1(b) has an $\alpha < 1$. This is a much bigger deal than you might think.

The authors found a consistent higher-than-one α for all wealth creation and innovation activities in a city and, at the same time, a consistent lower-than-one α for all activities accounting for infrastructure management. $\alpha > 1$ means that each added individual to the city contributes more than its fair share to the total. If you have a city where each individual publishes a patent per year and you add a new inhabitant to the city, you don't get an additional patent that year: you get that now each individual publishes 1.12 patents! Vice versa, $\alpha < 1$ is a classic "economies of scale" scenario: once you serve 100 people, you can serve an additional person without increasing your effort by 1%.

So far, this is not a network paper: it's just a purely statistical observation. It is when trying to explain such phenomena that you find networks everywhere^{4,5,6}. Networks are a necessary ingredient to explain why an additional node enriches the network in a non linear way. Humans have limited resources, so they can only interact with what they can access. In network terms, these are the other nodes at a maximum distance l from them. Every time you add a neighbor with new connections, lots of new nodes will get closer to you, some closer than l .

Consider Figure 52.2 as an example. Let's assume that node u is selling something, and it has a range: it can only serve up until its neighbors' neighbors. Its original productivity is then 10. Then, node v appears, and it connects to u . If v 's contribution were to be linear, u 's productivity would go up to 11. But v has other neighbors of its own, neighbors that were previously unreachable by u , as they were at distance $l = 3$. Thus, u 's productivity jumps to 15!

This is a sort of combinatorial effect, where each node adds a

⁴ Luís MA Bettencourt. The origins of scaling in cities. *science*, 340(6139):1438–1441, 2013

⁵ Elsa Arcaute, Erez Hatna, Peter Ferguson, Hyejin Youn, Anders Johansson, and Michael Batty. Constructing cities, deconstructing scaling laws. *Journal of The Royal Society Interface*, 12(102):20140745, 2015

⁶ Andres Gomez-Lievano, Oscar Patterson-Lomba, and Ricardo Hausmann. Explaining the prevalence, scaling and variance of urban phenomena. *Nature Energy*, pages 1–9, 2018

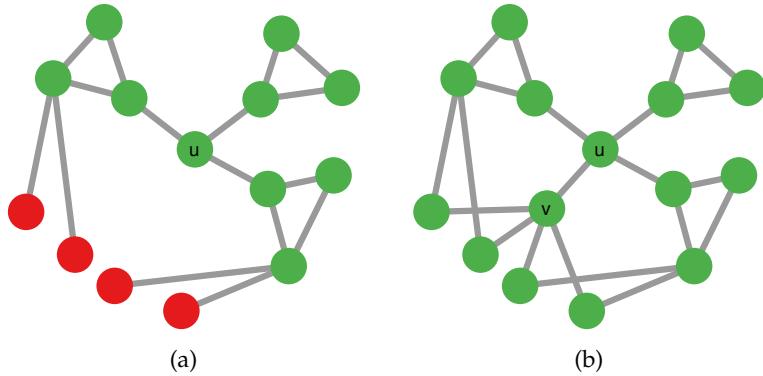


Figure 52.2: An explanation of non-linear node addition contribution. The node color encodes the reachability of a node from u : red = unreachable, green = reachable.

new factor you can use and recombine with all the factors that were already present so far. This is easy to see especially in patents data⁷. Every time someone makes a new invention, that new invention can be combined with all the previous inventions to create a new one, and so on at infinity. Thus, the knowledge added by a new invention potentially *multiplies* itself with the previously accumulated knowledge, rather than just *adding* to it.

52.2 Anonymity in the Age of Social Networks

We live in troubling times when it comes to our privacy. Large organizations have an interest in gathering information about each individual, whether they do it for surveillance – as highlighted by Snowden’s leaks –, or for profit – Facebook tracking is ubiquitous, but by no mean the exception in the private sector. The problem is that the simple usage of technology spreads an uncontrollable amount of information about us: the simple sequences of queries you ask a search engine might be enough to identify you⁸, and the combination of few elementary demographic pieces of data can de-anonymize 80% of people⁹.

If that worries you, consider that queries and demographics are simple unconnected data. When you talk about interconnected information, the problem is much worse. Consider what you see in Figure 52.3. The node 1 in the center of this network might be you. If an attacker has identified some of your connections, they can say a lot about you¹⁰: the only node in this network that has exactly $\{2, 3, 4, 5\}$ as the set of their friends. They might not be able to know your name, but under the assumption of homophily (Chapter 30) they could infer what you like, your sexual orientation, and maybe even health issues. This is not solved by making your profile private¹¹. Even not having a profile at all on social media won’t make you safe: a platform can create a shadow profile of you¹².

⁷ Hyejin Youn, Deborah Strumsky, Luis MA Bettencourt, and José Lobo. Invention as a combinatorial process: evidence from us patents. *Journal of The Royal Society Interface*, 12(106):20150272, 2015

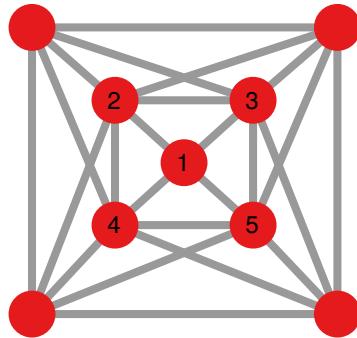
⁸ Lars Backstrom, Cynthia Dwork, and Jon Kleinberg. Wherefore art thou r3579x?: anonymized social networks, hidden patterns, and structural steganography. In *Proceedings of the 16th international conference on World Wide Web*, pages 181–190. ACM, 2007

⁹ Latanya Sweeney. k-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(05):557–570, 2002

¹⁰ Shirin Nilizadeh, Apu Kapadia, and Yong-Yeol Ahn. Community-enhanced de-anonymization of online social networks. In *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security*, pages 537–548, 2014

¹¹ Elena Zheleva and Lise Getoor. To join or not to join: the illusion of privacy in social networks with mixed public and private user profiles. In *Proceedings of the 18th international conference on World wide web*, pages 531–540, 2009

¹² David Garcia. Leaking privacy and shadow profiles in online social networks. *Science advances*, 3(8):e1701172, 2017



De-anonymizing social networks¹³ is feasible, under a wide array of different scenarios – whether the attacker is a government agency, a marketing campaign, or an individual stalker. This is usually done by creating a certain amount of auxiliary information that can then be used to recursively de-anonymize more and more nodes in the network. Counter-measures usually adopt the k -anonymity style: making sure that no individual can be identified by obfuscating enough data to make at least $k - 1$ other individuals identical to her in some respect. For instance, a network is k -degree anonymous if there are at least k nodes with any given degree value¹⁴.

Sometimes, the focus is preventing the disclosure of information about a relationship, i.e. to combat link re-identification¹⁵. You might not want Facebook to know you are friend with someone, which they could do by performing some relatively trivial link prediction – see Part VII. In those cases, you might want to hide some of your relationships, and/or add a few fake connections, to throw off the score function of the link you want to hide.

52.3 Human Connectome

Quite likely, the most famous and studied network in human history is the brain. We have been studying neural networks of many animals, due to their limited size and ease of analysis: cats¹⁶, mice¹⁷, and, of course, the superstar *C. Elegans* worm¹⁸. However, most of this is done with the big prize as the ultimate objective: the human brain. You might have heard of the Human Connectome Project. Proposed in 2005¹⁹, its objective was to create a low-level network map of the human brain: a network where nodes are individual neurons and connections are the synapses between them.

The idea was that applying all the network science artillery to such a network would help us understanding better how our brains work²⁰ – or don't, sometimes. In fact, one of the major lines of research is comparing the brain connection patterns between healthy

Figure 52.3: A network in which I label the identified nodes.

¹³ Arvind Narayanan and Vitaly Shmatikov. De-anonymizing social networks. In *2009 30th IEEE symposium on security and privacy*, pages 173–187. IEEE, 2009

¹⁴ Kun Liu and Evinaria Terzi. Towards identity anonymization on graphs. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 93–106, 2008

¹⁵ Elena Zheleva and Lise Getoor. Preserving the privacy of sensitive relationships in graph data. In *International Workshop on Privacy, Security, and Trust in KDD*, pages 153–171. Springer, 2007

¹⁶ JW Scannell, GAPC Burns, CC Hilgetag, MA O'Neil, and Malcolm P Young. The connectional organization of the cortico-thalamic system of the cat. *Cerebral Cortex*, 9(3):277–299, 1999

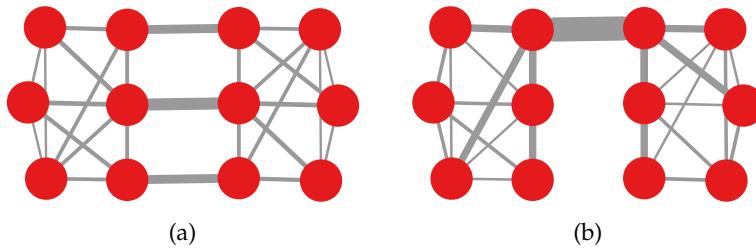
¹⁷ Quanxin Wang, Olaf Sporns, and Andreas Burkhalter. Network analysis of corticocortical connections reveals ventral and dorsal processing streams in mouse visual cortex. *Journal of Neuroscience*, 32(15):4386–4399, 2012b

¹⁸ Siming Li, Christopher M Armstrong, Nicolas Bertin, Hui Ge, Stuart Milstein, Mike Boxem, Pierre-Olivier Vidalain, Jing-Dong J Han, Albar Chesneau, Tong Hao, et al. A map of the interactome network of the metazoan *c. elegans*. *Science*, 303(5657):540–543, 2004

¹⁹ Olaf Sporns, Giulio Tononi, and Rolf Kötter. The human connectome: a structural description of the human brain. *PLoS computational biology*, 1(4), 2005

²⁰ Ed Bullmore and Olaf Sporns. Complex brain networks: graph theoretical analysis of structural and functional systems. *Nature reviews neuroscience*, 10(3):186–198, 2009

and unhealthy individuals, because network analysis should be able to easily allow the identification of significant differences in the structures²¹. For instance, as Figure 52.4 shows, a simple edge betweenness centrality analysis could identify the overload on some synapses caused by structural differences.



There have been many studies of the human brain before the Human Connectome started. For instance, researchers have studied the effect of learning on the connections between brain areas²². The reason why the Human Connectome is so revolutionary is the granularity of the data. Most brain network studies look at brain activity patterns: the high level difference in electrical potential of brain areas. In this case, the nodes are not individual neurons, but larger modules of the brain.

To be clear, one does not exclude the value of the other. In fact, the brain is an extremely complex organ: it is the most important organ of your body²³. This means that it operates at multiple scales²⁴, in a hierarchical fashion: neurons are part of modules²⁵, and there are modules of modules, and so on – check out Chapters 33 and 37 for a few refreshers on hierarchies. In fact, one of the most appropriate models of the brain is multilayer networks²⁶.

52.4 Science of Science and of Success

Unsurprisingly, one of the things that interests scientists the most is... scientists. Network scientists are no exception to this rule. There is a large and healthy literature in analyzing networks of scientists. We already saw many examples of two types of science networks: co-authorship networks, where scientists are connected to each other if they collaborate on the same paper/project; and citation networks, connecting papers if one cites another.

The two can be combined to try and gather a general picture of how science gets done. Science is one of the most important human activities²⁷, because we rely on it to develop new and better ways to improve our everyday life. It's better to understand how it works, so that we can do it better. This is fundamentally the mission statement

²¹ Danielle S Bassett and Edward T Bullmore. Human brain networks in health and disease. *Current opinion in neurology*, 22(4):340, 2009

Figure 52.4: The comparison between (a) a healthy brain and (b) an unhealthy brain. Nodes are neurons, edges are synapses and their thickness is proportional to their edge betweenness centrality.

²² Danielle S Bassett, Nicholas F Wymbs, Mason A Porter, Peter J Mucha, Jean M Carlson, and Scott T Grafton. Dynamic reconfiguration of human brain networks during learning. *Proceedings of the National Academy of Sciences*, 108(18):7641–7646, 2011

²³ According to the brain.

²⁴ Richard F Betzel and Danielle S Bassett. Multi-scale brain networks. *Neuroimage*, 160:73–83, 2017

²⁵ Paolo Bonifazi, Miri Goldin, Michel A Picardo, Isabel Jorquera, A Cattani, Gregory Bianconi, Alfonso Represa, Yechezkel Ben-Ari, and Rosa Cossart. Gabaergic hub neurons orchestrate synchrony in developing hippocampal networks. *Science*, 326(5958):1419–1424, 2009

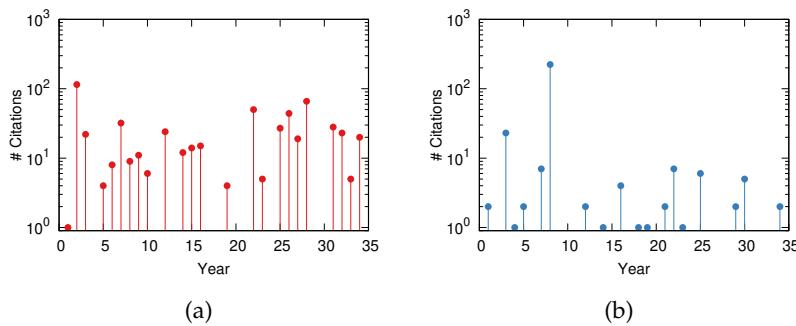
²⁶ Manlio De Domenico. Multilayer modeling and analysis of human brain networks. *Giga Science*, 6(5):gix004, 2017

²⁷ According to scientists.

of the science of science field^{28,29,30}, kickstarted by network scientists and making extensive use of network analysis tools.

One of the most peculiar findings is that the occurrence of the highest impact work of a scientist's career will happen at a random point in time³¹. In other words, there is no way to predict which of your papers will earn you a Nobel prize: it could be your first, it could be your last, or any in between. This is bad news if we want to predict the success of some research, but it's great news for me. The fact that I haven't come even close to making a groundbreaking discovery doesn't mean it won't happen eventually. I simply won't see coming if it does (it won't).

Figure 52.5 shows an example of this concept. In both cases, the breakout paper arrived early, but there is no pattern in how citations come. Moreover, the red scientist (Figure 52.5(a)) is a better scientist on average than the blue one (Figure 52.5(b)), having 23.5 citations per paper against blue's 16.2. They're also more productive (24 vs 18 papers). And yet, it is the blue scientists who published the best paper – with 223 citations, while red's best paper only has 115 citations. Life is unfair this way.



Science of science ended up being a specialized niche of the more broad field of the science of success: the systematic investigation of the gap between one's performance and their success^{32,33}. It is not always the best work of a person that ends up being the most successful. For instance, the Mona Lisa, the most famous painting of the world, is a masterpiece from one of the greatest intellectuals of all time – Leonardo da Vinci – but, among its other breathtaking creations, it is rather unremarkable. So unremarkable, in fact, that it was completely ignored and not even exposed until interest in it exploded after its theft.

This disconnect between performance and success is not an exclusive domain of art. It is a much more universal phenomenon. Another studied example is tennis³⁴: it is not necessarily the tennis

²⁸ Albert-László Barabási, Chaoming Song, and Dashun Wang. Publishing: Handful of papers dominates citation. *Nature*, 491(7422):40, 2012

²⁹ Dashun Wang, Chaoming Song, and Albert-László Barabási. Quantifying long-term scientific impact. *Science*, 342(6154):127–132, 2013

³⁰ Santo Fortunato, Carl T Bergstrom, Katy Börner, James A Evans, Dirk Helbing, Staša Milojević, Alexander M Petersen, Filippo Radicchi, Roberta Sinatra, Brian Uzzi, et al. Science of science. *Science*, 359(6379):eaao0185, 2018

³¹ Roberta Sinatra, Dashun Wang, Pierre Deville, Chaoming Song, and Albert-László Barabási. Quantifying the evolution of individual scientific impact. *Science*, 354(6312):aaf5239, 2016

Figure 52.5: Two examples of career paths of scientists, showing the number of citations (y axis) gathered from papers published in a given year (x axis).

³² Samuel P Fraiberger, Roberta Sinatra, Magnus Resch, Christoph Riedl, and Albert-László Barabási. Quantifying reputation and success in art. *Science*, 362(6416):825–829, 2018

³³ Lu Liu, Yang Wang, Roberta Sinatra, C Lee Giles, Chaoming Song, and Dashun Wang. Hot streaks in artistic, cultural, and scientific careers. *Nature*, 559(7714):396, 2018a

³⁴ Burcu Yucesoy and Albert-László Barabási. Untangling performance from success. *EPJ Data Science*, 5(1):17, 2016

player at the top of the world ranking the one gathering the most Wikipedia page views – or news articles about them, for that matter.

In fact, the performance-success disconnect can and should be applied to science as well. In this section, I equated “success” with citations: a successful paper gathers tons of citations. But is it the *best* (read: highest performing) paper? Not at all! Citations and grant awards correlate with things that are independent of the science/performance itself (e.g., gender³⁵, race³⁶ and how junior a person is³⁷). The world isn’t a perfect meritocracy. Cumulative advantage is not just the pretty story of how you model broad degree distributions in networks (Section 17.3): it is the real unfairness in front of everybody who does not start in the advantaged place/time/gender/race. We should investigate the performance-success disconnect in order to make the world suck a little less. One way to do it is to model science as the interaction between individual characteristics and systemic structures³⁸.

52.5 Human Mobility

A significant portion of network scientists have also worked on issues of human mobility: describing and predicting how individuals and collectives move in the urban and global landscape^{39,40}. There are a few reasons for this. First, there is a strong connection between human mobility and many networked phenomena that network scientists investigate. Just to highlight the example from the previous sections: one can use the “mobility” of scientists between affiliations to predict their success⁴¹. Alternatively, one can use mobility data to augment the de-anonymization process of people in social settings⁴², or to better predict the spread of infectious diseases^{43,44,45}.

Second, complex networks are themselves useful tools to model and analyze mobility patterns. For instance, one can create a better synthetic model of human mobility by using an underlying social network to create realistic motivations for the simulated agents to move in space⁴⁶.

Classically, to predict the number of people moving from area *A* to area *B*, one would use a “gravity model”. This works just like Newton’s gravity law: the mobility relation between two areas is directly proportional to how many people live in them (their “mass”) and inversely proportional to their distance⁴⁷. In other words, there can be many people moving between New York and Chicago because they are huge cities, but Boston might attract more New Yorkers despite being less populous, simply because it’s closer. The gravity model is overly simplistic: it’s deterministic, it requires previous mobility data to fit parameters, it lacks theoretical grounding, and

³⁵ Jonathan R Cole. Fair science: Women in the scientific community. 1979

³⁶ Donna K Ginther, Walter T Schaffer, Joshua Schnell, Beth Masimore, Faye Liu, Laurel L Haak, and Raynard Kington. Race, ethnicity, and nih research awards. *Science*, 333(6045):1015–1019, 2011

³⁷ Robert T Blackburn, Charles E Behmer, and David E Hall. Research note: Correlates of faculty publications. *Sociology of Education*, pages 132–141, 1978

³⁸ Samuel F Way, Allison C Morgan, Daniel B Larremore, and Aaron Clauset. Productivity, prominence, and the effects of academic environment. *Proceedings of the National Academy of Sciences*, 116(22):10729–10733, 2019

³⁹ Marta C Gonzalez, Cesar A Hidalgo, and Albert-Laszlo Barabasi. Understanding individual human mobility patterns. *nature*, 453(7196):779–782, 2008

⁴⁰ Julián Candia, Marta C González, Pu Wang, Timothy Schoenharl, Greg Madey, and Albert-László Barabási. Uncovering individual and collective human dynamics from mobile phone records. *Journal of physics A: mathematical and theoretical*, 41(22):224015, 2008

⁴¹ Pierre Deville, Dashun Wang, Roberta Sinatra, Chaoming Song, Vincent D Blondel, and Albert-László Barabási. Career on the move: Geography, stratification, and scientific impact. *Scientific reports*, 4:4770, 2014

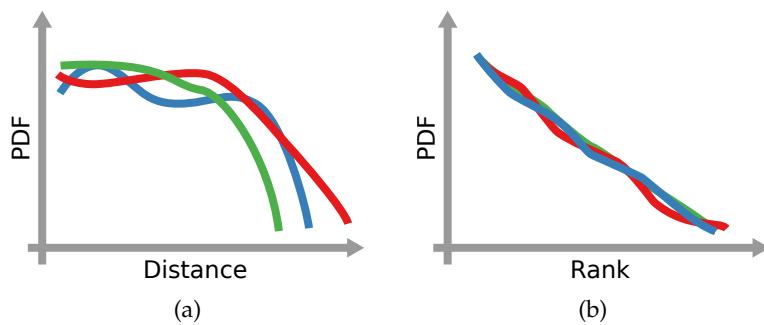
⁴² Yves-Alexandre De Montjoye, César A Hidalgo, Michel Verleysen, and Vincent D Blondel. Unique in the crowd: The privacy bounds of human mobility. *Scientific reports*, 3:1376, 2013

⁴³ Vittoria Colizza, Alain Barrat, Marc Barthelemy, Alain-Jacques Valleron, and Alessandro Vespignani. Modeling the worldwide spread of pandemic influenza: baseline case and containment interventions. *PLoS medicine*, 4(1), 2007

⁴⁴ Duygu Balcan, Vittoria Colizza, Bruno Gonçalves, Hao Hu, José J Ramasco, and Alessandro Vespignani. Multiscale mobility networks and the spatial spreading of infectious diseases. *PNAS*, 106(51):21484–21489, 2009

it simply doesn't predict observations that well. Network scientists have then developed a radiance model to fix these shortcomings⁴⁸.

What do we find? At a collective level, human mobility patterns are surprisingly universal, but *not* when it comes to the covered distance⁴⁹. Figure 52.6(a) shows that the probability of making a trip is only mildly related to distance: there is no function properly approximating the likelihood of you visiting a place given its distance to you, and different cities have different scaling and cutoffs. In other words, it is not true that the farther apart a pizza place is, the least you go to eat there.



It is rather that how frequently you go to eat there is connected to the number – and quality – of the alternatives in between you and the pizza place. This is only correlated with, rather than being caused by, distance. What matters most, is the *rank* of the place in your preferences. Figure 52.6(b) shows that there is a clear and universal function predicting a trip's probability given the popularity rank of the destination – e.g. no matter the city, 20% of trips go to the most popular destination.

The predictability of the collective dynamics, however, does not trickle down to predictability of individuals. Yes, we're animals of habit: we often commute between the same two places – a property one can exploit to infer home and work locations by looking at incomplete mobility data^{50,51}. On average, one can confidently predict 93% of individual mobility⁵². However, there is a large variation between individuals: for some you could predict even better than that, while others are fundamentally unpredictable⁵³.

52.6 Memetics

Who around here doesn't like Internet memes? Cute and funny little pictures, perfect to waste time at work. Of course network scientists love them. However, we need to maintain appearances and pretend that the penguin image we have on our screens is there really for

⁴⁵ Michele Tizzoni, Paolo Bajardi, Adeline Decuyper, Guillaume Kon Kam King, Christian M Schneider, Vincent Blondel, Zbigniew Smoreda, Marta C González, and Vittoria Colizza. On the use of human mobility proxies for modeling epidemics. *PLoS computational biology*, 10(7), 2014

⁴⁶ Mirco Musolesi and Cecilia Mascolo. A community based mobility model for ad hoc network research. In *MOBIHOC*, pages 31–38, 2006

⁴⁷ Dirk Brockmann, Lars Hufnagel, and Theo Geisel. The scaling laws of human travel. *Nature*, 439(7075):462–465, 2006

Figure 52.6: (a) Probability of a trip (y axis) as a function of the distance to the destination (x axis). (b) Probability of a trip (y axis) as a function of the destination's rank (x axis). In both cases, different colors report data from different cities.

⁴⁸ Filippo Simini, Marta C González, Amos Maritan, and Albert-László Barabási. A universal model for mobility and migration patterns. *Nature*, 484 (7392):96–100, 2012

⁴⁹ Anastasios Noulas, Salvatore Scellato, Renaud Lambiotte, Massimiliano Pontil, and Cecilia Mascolo. A tale of many cities: universal patterns in human urban mobility. *PloS one*, 7(5), 2012

⁵⁰ Md Shahadat Iqbal, Charisma F Choudhury, Pu Wang, and Marta C González. Development of origin-destination matrices using mobile phone call data. *Transportation Research Part C*, 40:63–74, 2014

⁵¹ Lauren Alexander, Shan Jiang, Mikel Murga, and Marta C González. Origin-destination trips by purpose and time of day inferred from mobile phone data. *Transportation research part c*, 58:240–250, 2015

⁵² Chaoming Song, Zehui Qu, Nicholas Blumm, and Albert-László Barabási. Limits of predictability in human mobility. *Science*, 327(5968):1018–1021, 2010

⁵³ Luca Pappalardo, Filippo Simini, Salvatore Rinzivillo, Dino Pedreschi, Fosca Giannotti, and Albert-László Barabási. Returners and explorers dichotomy in human mobility. *Nature communications*, 6:8166, 2015

work. We're studying memes, you know? This is for science. There are a few angles with which network scientists attack the study of memes. Some of those already found space elsewhere in the book – e.g. in Section 21.2.

The first is the relationship between the network structure and the probability of a rumor to spread – or the fraction of nodes who will end up hearing a rumor. Theoretical calculations⁵⁴ show the impact of the network's topology: in a random graph, initial spread is slow but it will relentlessly cover the entire network; while for scale free networks the initial speed is fast but, in presence of degree correlations, it might fail to cover the entire network. All of this is very similar to simulations of diseases spreading on a network (see Chapter 20).

Other studies show how the large diversity in the meme success distribution – few memes spread globally while most are immediately forgot – are due to the limited capacity of brains to process information^{55,56,57}. Another key question is whether memes spread following simple or complex contagion: is a single exposure sufficient or does reinforcement play a significant role? It seems that memes indeed obey the complex contagion rules⁵⁸. For a refresher on the concepts, see Chapter 21.

More complex topological features, such as communities, are difficult to treat mathematically, but their impact can be studied using real world data. Figure 52.7 shows a toy example of the role of communities in meme propagation. Memes originating in the overlap between different communities – in red in Figure 52.7 – have a better chance to go viral^{59,60}. Being born well embedded in a community – blue in Figure 52.7 – is bad for propagation, because there are not many paths leading the meme outside of the community.

In general, there are many empirical studies investigating how information propagates through a social network⁶¹, be it memes, rumors, news⁶², videos^{63,64}, or photographs^{65,66}.

Specifically, one study focuses on the dynamics of “following” a

⁵⁴ Maziar Nekovee, Yamir Moreno, Ginestra Bianconi, and Matteo Marsili. Theory of rumour spreading in complex social networks. *Physica A: Statistical Mechanics and its Applications*, 374(1): 457–470, 2007

⁵⁵ Nathan Oken Hodas and Kristina Lerman. How visibility and divided attention constrain social contagion. In *SocialCom*, pages 249–257. IEEE, 2012

⁵⁶ Lilian Weng, Alessandro Flammini, Alessandro Vespignani, and Filippo Menczer. Competition among memes in a world with limited attention. *Scientific reports*, 2:335, 2012

⁵⁷ James P Gleeson, Jonathan A Ward, Kevin P O’sullivan, and William T Lee. Competition-induced criticality in a model of meme popularity. *Physical review letters*, 112(4):048701, 2014

⁵⁸ Bjarke Mønsted, Piotr Sapieżyński, Emilio Ferrara, and Sune Lehmann. Evidence of complex contagion of information in social media: An experiment using twitter bots. *PloS one*, 12(9), 2017

⁵⁹ Lilian Weng, Filippo Menczer, and Yong-Yeol Ahn. Virality prediction and community structure in social networks. *Scientific reports*, 3:2522, 2013

⁶⁰ Lilian Weng, Filippo Menczer, and Yong-Yeol Ahn. Predicting successful memes using network and community structure. In *ICWSM*, 2014

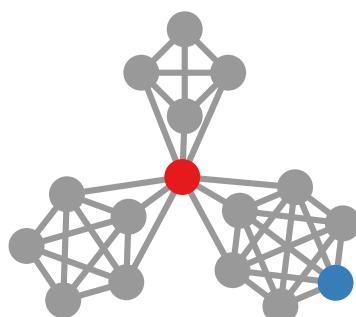


Figure 52.7: A social network. The red and blue nodes are the origin points of two memes.

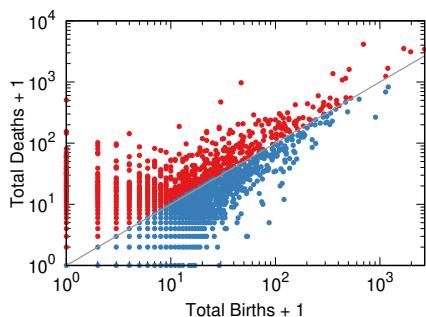
content creator on social media⁶⁷. The common sense thing is that the more people are following you – say on Twitter – the better it is. You can be more influential if more people listen to you. However, experiments show that this is true only to a certain point. What matters most is the engagement of the followers. Just increasing the count is doing you no good if the people clicking on the *Follow* button actually don't read what you produce. Your voice is diluted on the platform and you have less reach if you inflate those numbers.

The structure of the social network is not, however, the only thing that matters. I already mentioned elsewhere in the book (in Section 21.2) that an important factor is also timing: when and how fast you get your appreciation matters a lot in determining whether you are going viral. Alternatively, one could look at the content itself of the meme: the specific image or text associated to it. Studies show how positive valence – a happy meme – are useful for propagation⁶⁸. In my own research, I instead show how innovation is the key: you want to do something that is dissimilar from everything that has been done before^{69,70}.

52.7 Digital Humanities

Digital humanities is an umbrella term, covering a vast set of applications of computational tools to disciplines in the humanities. As a trained digital humanist myself, I cannot end this book without taking a closer look at this field. And, in a sense, I wasn't, because one could argue that the entire network analysis field is one of the largest subfield of digital humanities. In network analysis we have mathematical and computational models – graphs and networks – which are primarily applied to understand social systems. However, among the gigantic bazaar of network science applications, some stand out as poster children of digital humanities.

One is for sure the study of the birth-death network across cultural history⁷¹. Figure 52.8 shows an interesting historical pattern: each



⁶¹ Kristina Lerman and Rumi Ghosh. Information contagion: An empirical study of the spread of news on digg and twitter social networks. In *ICWSM*, 2010

⁶² Soroush Vosoughi, Deb Roy, and Sinan Aral. The spread of true and false news online. *Science*, 359(6380):1146–1151, 2018

⁶³ Meeyoung Cha, Haewoon Kwak, Pablo Rodriguez, Yong-Yeol Ahn, and Sue Moon. I tube, you tube, everybody tubes: analyzing the world's largest user generated content video system. In *SIGCOMM*, pages 1–14, 2007

⁶⁴ Meeyoung Cha, Haewoon Kwak, Pablo Rodriguez, Yong-Yeol Ahn, and Sue Moon. Analyzing the video popularity characteristics of large-scale user generated content systems. *IEEE/ACM Transactions on networking*, 17(5):1357–1370, 2009

⁶⁵ Meeyoung Cha, Alan Mislove, Ben Adams, and Krishna P Gummadi. Characterizing social cascades in flickr. In *Proceedings of the first workshop on Online social networks*, pages 13–18, 2008

⁶⁶ Meeyoung Cha, Alan Mislove, and Krishna P Gummadi. A measurement-driven analysis of information propagation in the flickr social network. In *WWW*, pages 721–730, 2009b

⁶⁷ Meeyoung Cha, Hamed Haddadi, Fabricio Benevenuto, and Krishna P Gummadi. Measuring user influence in twitter: The million follower fallacy. In *ICWSM*, 2010

⁶⁸ Jonah Berger and Katherine L Milkman. What makes online content viral? *Journal of marketing research*, 49(2):192–205, 2012

⁶⁹ Michele Coscia. Average is boring: How similarity kills a meme's success. *Scientific reports*, 4:6477, 2014

Figure 52.8: The number of famous people who were born (x axis) and died (y axis) in a city. The identity line is in gray. Red points above the identity line and blue points below.

point is a city, and for each city we count the number of famous people who were born and who died in that city. In red we can see death attractors: cities who had more famous deaths than births. In blue we have the emitting cities. By analyzing the historical trajectories of cities, we can see how the cultural center of the world moved from Rome to Paris and then to New York, because more and more people die in the city where they work – and notable people work where most notable people are. There are other interesting patterns, for instance the fact that the median distance between the birth and death place is increasing, reflecting technological advancements.

With a similar dataset, researchers built the “notable people portfolio” of cities and nations⁷². The idea is to classify all famous people in the area they contributed the most to humanity. Then, one can visualize in which areas places specialize⁷³. For instance, the largest profession represented in the United States is actors, while it is politicians for Greece. But one could explore other dimensions. For instance, professions that are over-expressed in a country against the rest of the world, like chess players in Armenia (6% of all famous people!). Or explore gender divide: in Canada 27.4% of male famous people were actors against 55.4% female famous people. Finally, you can explore time as well. Before 1700 AD, the most common way to become famous in Italy was to have a career in politics (30.7% of famous people did). Afterward? You’re better off trying as a soccer player (21.6%).

Other digital humanities applications of network science involve archaeology. This mostly involves the use of network visualization techniques to make sense of a complex, interconnected, and often largely incomplete set of evidence⁷⁴. However, it is not necessary to limit ourselves to this: network analysis can be used as a tool to explore evidence. For instance, there are studies of social networks in classical Rome⁷⁵. Other examples of prehistoric social network archaeology focus on pre-hispanic North America⁷⁶. Departing from archaeology, the field of social network analysis in a historic⁷⁷, religious⁷⁸, or anthropological⁷⁹ setting is alive and well.

And since network analysis endows us with powerful tools to study hidden preferences – such as homophily and segregation, see Chapter 30 – it is a natural instrument to use in other humanities fields, such as gender studies. In particular, there are studies showing unequal gender dynamics when it comes to power relations in online collaborative tools such as, e.g., Wikipedia. As you might expect, the majority of Wikipedia contributors are white men.

When it comes to female representation in the content^{80,81}, this gender gap shows. The researchers find that women are equally represented in article numbers – at least in the main six language

⁷⁰ Michele Coscia. Popularity spikes hurt future chances for viral propagation of protomemes. *Communications of the ACM*, 61(1):70–77, 2017

⁷¹ Maximilian Schich, Chaoming Song, Yong-Yeol Ahn, Alexander Mirsky, Mauro Martino, Albert-László Barabási, and Dirk Helbing. A network framework of cultural history. *science*, 345(6196):558–562, 2014

⁷² Amy Zhao Yu, Shahar Ronen, Kevin Hu, Tiffany Lu, and César A Hidalgo. Pantheon 1.0, a manually verified dataset of globally famous biographies. *Scientific data*, 3:150075, 2016

⁷³ <https://pantheon.world/>

⁷⁴ Tom Brughmans. Thinking through networks: a review of formal network methods in archaeology. *Journal of Archaeological Method and Theory*, 20(4):623–662, 2013

⁷⁵ Tom Brughmans. Connecting the dots: towards archaeological network analysis. *Oxford Journal of Archaeology*, 29(3):277–303, 2010

⁷⁶ Barbara J Mills, Jeffery J Clark, Matthew A Peeples, W Randall Haas, John M Roberts, J Brett Hill, Deborah L Huntley, Lewis Borck, Ronald L Breiger, Aaron Clauset, et al. Transformation of social networks in the late pre-hispanic us southwest. *Proceedings of the National Academy of Sciences*, 110(15):5785–5790, 2013

⁷⁷ Claire Lemercier. Formal network methods in history: why and how? In *Social networks, political institutions, and rural societies*, pages 281–310. 2015

⁷⁸ Eleanor A Power. Discerning devotion: Testing the signaling theory of religion. *Evolution and Human Behavior*, 38(1):82–91, 2017

⁷⁹ Jessica C Flack, Michelle Girvan, Frans BM De Waal, and David C Krakauer. Policing stabilizes construction of social niches in primates. *Nature*, 439(7075):426–429, 2006

⁸⁰ Claudia Wagner, David Garcia, Mohsen Jadidi, and Markus Strohmaier. It’s a man’s wikipedia? assessing gender inequality in an online encyclopedia. In *Ninth international AAAI conference on web and social media*, 2015

⁸¹ Claudia Wagner, Eduardo Graells-Garrido, David Garcia, and Filippo Menczer. Women through the glass ceiling: gender asymmetries in wikipedia. *EPJ Data Science*, 5(1):5, 2016

editions of Wikipedia. However, it is the way women are portrayed that is the problem. Women on Wikipedia tend to be more linked to men than vice versa. Moreover, romantic relationships and family-related issues are much more frequently discussed on Wikipedia articles about women than men.

52.8 Political Polarization

Many researchers have used network analysis to study social aspects related to politics. Here I focus specifically on political polarization. Polarization is a complex phenomenon, which can be dissected in different ways from different perspectives. In general, one can say that polarization grows when people get more and more extreme in the political positions or ideologies they subscribe to. If everyone is a centrist, there is no polarization, but splitting camps in extreme left and extreme right indicates high polarization. Then there is the aspect of how you relate to people with different opinions than yours: are you civil, or is a disagreement enough to cause you to use toxic or aggressive language⁸²? Finally, you could consider the polarization of the population at large as different from the one of elites.

It doesn't seem immediately obvious how this relates to networks. After all we're talking about opinions and how people use language. However, networks can be a powerful tool to study polarization. In fact, in the early times of the application of networks to polarization, researchers were mostly making a structural argument⁸³: homophily plays a strong role, people would connect only with people they agree, so we could identify echo chambers^{84,85,86}. We've seen this argument in Section 30.4. It should be noted that the existence and the strength of echo chambers is a far from established fact, there are still questions to be answered^{87,88,89}.

To sum it up, looking at Figure 52.9 one could think of polarization as being three things: more extreme opinions, isolation in homogeneous groups known as echo chambers, and the tendency of those groups not to interact with groups that are too dissimilar from them. You could also add attributes on the edges recording the toxicity of the language used and correlate it with the opinion difference between the two people interacting. If you get more toxicity with a higher opinion difference, then you get higher polarization.

From there, researchers developed a few different ways to quantify how much it is difficult for people with a given opinion to reach people with a different opinion in a social network^{90,91,92}. The harder it is, the more society is polarized. Some popular approaches are based on partitioning the networks into two groups of opposite opinions and see how easy it is for random walks to cross the boundaries of

⁸² Yelena Mejova, Amy X Zhang, Nicholas Diakopoulos, and Carlos Castillo. Controversy and sentiment in online news. *arXiv preprint arXiv:1409.8152*, 2014

⁸³ Michael Conover, Jacob Ratkiewicz, Matthew Francisco, Bruno Gonçalves, Filippo Menczer, and Alessandro Flammini. Political polarization on twitter. In *Proceedings of the international aaai conference on web and social media*, volume 5, pages 89–96, 2011

⁸⁴ Walter Quattrociocchi, Antonio Scala, and Cass R Sunstein. Echo chambers on facebook. Available at SSRN 2795110, 2016

⁸⁵ Eli Pariser. *The filter bubble: What the Internet is hiding from you.* penguin UK, 2011

⁸⁶ Federico Cinus, Marco Minici, Corrado Monti, and Francesco Bonchi. The effect of people recommenders on echo chambers and polarization. In *Proceedings of the International AAAI Conference on Web and Social Media*, volume 16, pages 90–101, 2022

⁸⁷ Alan I Abramowitz and Kyle L Saunders. Is polarization a myth? *The Journal of Politics*, 70(2):542–555, 2008

⁸⁸ Levi Boxell, Matthew Gentzkow, and Jesse M Shapiro. Greater internet use is not associated with faster growth in political polarization among us demographic groups. *Proceedings of the National Academy of Sciences*, 114(40):10612–10617, 2017

⁸⁹ Emily Kubin and Christian von Sikorski. The role of (social) media in political polarization: a systematic review. *Annals of the International Communication Association*, 45(3):188–206, 2021

⁹⁰ Wesley Cota, Silvio C Ferreira, Romualdo Pastor-Satorras, and Michele Starnini. Quantifying echo chamber effects in information spreading over political communication networks. *EPJ Data Science*, 8(1):35, 2019

⁹¹ Matteo Cinelli, Gianmarco De Francisci Morales, Alessandro Galeazzi, Walter Quattrociocchi, and Michele Starnini. The echo chamber effect on social media. *Proceedings of the National Academy of Sciences*, 118(9):e2023301118, 2021

⁹² Bjarke Mønsted and Sune Lehmann. Characterizing polarization in online vaccine discourse—a large-scale study. *PloS one*, 17(2):e0263746, 2022

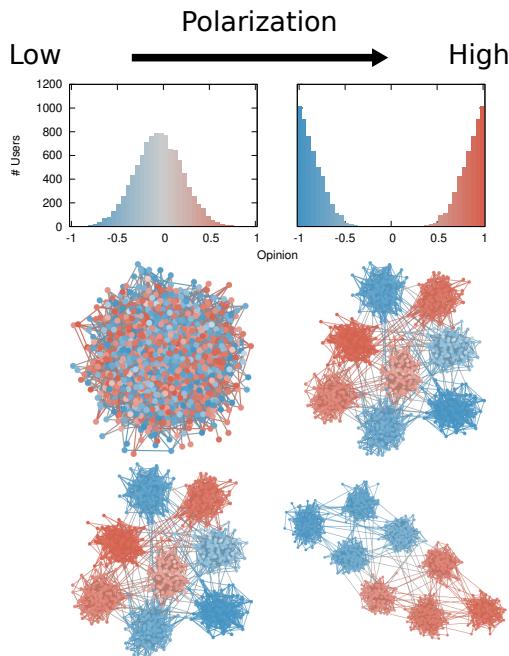


Figure 52.9: The three factors we can consider for ideological polarization. From top to bottom: opinion extremity, echo chambers, and opinion homophily across echo chambers.

this partition⁹³. I personally like to calculate the network distance between differing opinions⁹⁴ using the Generalized Euclidean measure I described in Section 47.2.

Finally, researchers have also created agent based models to see how polarization could arise – and be counteracted by good policies⁹⁵.

⁹³ Kiran Garimella, Gianmarco De Francisci Morales, Aristides Gionis, and Michael Mathioudakis. Quantifying controversy on social media. *ACM Transactions on Social Computing*, 1(1):1–27, 2018

⁹⁴ Marilena Hohmann, Karel Devriendt, and Michele Coscia. Quantifying ideological polarization on a network using generalized euclidean distance. *Science Advances*, 9(9):eabq2044, 2023

⁹⁵ Henrique Ferraz de Arruda, Felipe Maciel Cardoso, Guilherme Ferraz de Arruda, Alexis R Hernández, Luciano da Fontoura Costa, and Yamir Moreno. Modelling how social network algorithms can influence opinion polarization. *Information Sciences*, 588:265–278, 2022

53

Data & Tools

A professional worker is only as good as the tools they have and their mastery of them. Thus, knowing where to find the best tools is the first necessary step for being a good network scientist. The tools aren't going to do the job for you, but without them you're just a person armed with lots of good intentions. The aim of this chapter is to kickstart you to your career. I will give you a brief overview of the software libraries and programs one can use to analyze and visualize networks, point you to useful online resources – especially to find new data sources –, and briefly discuss some of the most famous graph data you will find in the literature.

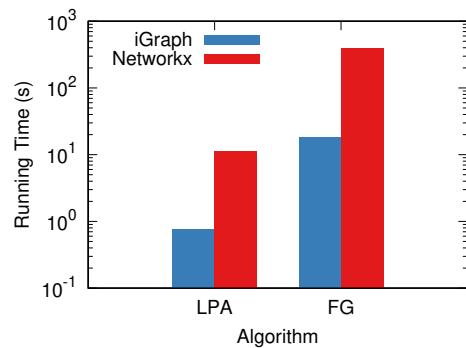
You might have already seen some of these resources here and there mentioned throughout the book. For instance, the vast majority of the exercises rely on you using Networkx, while in Part XIII I heavily relied on the knowledge of what Cytoscape and Gephi can do for network visualization.

53.1 Libraries

It might be my bias as a computer scientist showing, but my opinion is that, if you want to have a career in network science, the first thing you have to look at is libraries that help you programming your custom network analyses. There are many fully fledged software programs, with their graphical interfaces and ready-to-use implemented analyses, but I don't think you're really going to be a complete network scientist if you only rely on them. At some point, you will find out something you cannot do with them, and you'll need to roll up your sleeves and get your hands dirty with programming. If you start by learning the libraries, instead, you can always have the option of lazily use another software for all the trivial tasks that don't need any specific customized contribution.

Networkx

I start by dealing with Networkx^{1,2}. Networkx is a Python library implementing a vast array of network algorithms and analyses. Networkx is – as far as I can tell – the most popular choice for students approaching network analysis tasks. I think it's a generalist tool that is not the best at anything specifically, but good enough at everything. One downside is that it is the library struggling with computational efficiency the most, but it is the most complete and popular. If this were a chapter about cinema, Networkx would be Steven Spielberg: everybody knows him, every movie he makes is good but not really great, but it's a bit of a boring choice as a favorite director.



¹ Aric Hagberg, Pieter Swart, and Daniel S Chult. Exploring network structure, dynamics, and function using networkx. Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008

² <https://networkx.github.io/>

Figure 53.1: The running times for different implementations of community discovery algorithms in Networkx (red) and iGraph (blue).

As I mentioned, the biggest issue of Networkx is efficiency. Networkx is mostly implemented in Python, and it's so large it is impossible for the maintainers to guarantee high code standards throughout the library. Thus you might end up waiting for hours – or days! – for an operation that would take other libraries few seconds to complete. For instance, Figure 53.1 compares the running times – on the same network with $\sim 100k$ nodes and on the same machine – of the label propagation and fast greedy modularity community discovery algorithms between Networkx and iGraph. Note the logarithmic scale on the y axis, and despair. Thus, for any analysis with a high time complexity – for instance frequent pattern mining – you should definitely look somewhere else.

Another downside I stumbled upon is bugginess. You will often find that the most obscure network functions are sometimes not implemented correctly. I found myself having to switch library because the graph isomorphism functions on labeled multigraphs were just returning the wrong results. This is to be expected for a vast library like this: bugs take time to be noticed, and their fixes to be incorporated.

That is not to say that things aren't improving. I hope that the bug I stumbled upon is now corrected. And other examples of slow

implementations are actually on par with state of the art implementation. It used to take impossibly long to generate LFR benchmarks on Networkx, but when I checked on the current version at the time of writing this chapter, I noticed no runtime difference with the C binary provided by the original authors of the paper. Kudos to Networkx on this!

Networkx comes with big strengths as well. First, it is very *pythonic*, which means intuitive and easy to use – well, at least to me and to all who are comfortable with Python’s style of doing stuff. Second, as mentioned, it has really a broad coverage. You can tell this was a tool made by network scientists. The array of functions included in Networkx has no peers in any other library that I know. Finally, it has a relatively decent ecosystem. Of course, not everything can be implemented in Networkx. The developers need to choose what to focus on. But it is a tool on which it is relatively easy to build. Thus you can easily find packages expanding Networkx’s capabilities. For instance, Networkx doesn’t have a way to find temporal communities, but researchers have built a library on top of Networkx to do so.

graph-tool

If you want to stay in the domain of Python, the obvious alternative to Networkx is graph-tool^{3,4} by Tiago Peixoto. Graph-tool is, to some extent, the opposite of Networkx in almost every respect. For this reason, it represents a perfect complementary tool.

Graph-tool has several weaknesses. Many are connected to its strengths. For instance, one major strength of graph-tool is its efficiency: it is really fast in computing almost anything. This is due to the fact that it is one of the very few libraries I know that actually has parallel implementations of the network algorithms. This means that, if you are on a machine with multiple cores – which is to say, your computer is not older than fifteen years –, your analyses are going to run much faster because each of your cores will be involved in the computation. This comes at the downside of requiring some non-trivial technical expertise when dealing with it. I have had students who had to give up on some parts of their projects because they could not install graph-tool.

The other weakness is its incompleteness. Graph-tool is nowhere near Networkx when it comes to offering network analysis tools. This is due to the fact that graph-tool is practically a one man show. Tiago has made a godly amount of work for one person, but he is still one human. On the other hand, this is linked to a strength as well. There might not be many functions implemented in graph-tool, but the

³ Tiago P Peixoto. The graph-tool python library. *figshare*, 2014b

⁴ <https://graph-tool.skewed.de/>

ones that are there benefit from having a single mind behind them. The aforementioned issue I had with the bugs in labeled multigraph isomorphism was solved by simply using graph-tool.

Moreover, getting into Tiago's frame of mind is necessary to use graph-tool. You need to understand the way he does things in order to be able to do them as well. Things like function naming, object types, parameter passing – what you call the interface of the library – are not as pythonic and intuitive as in Networkx.

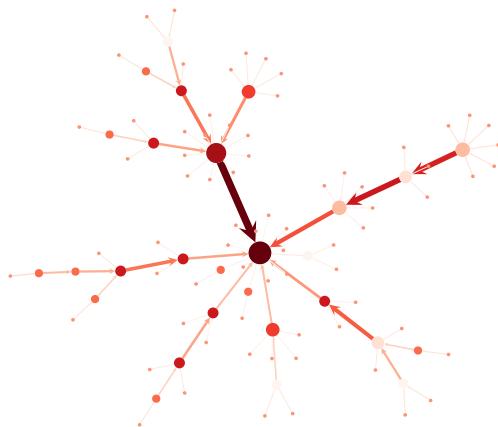


Figure 53.2: A network visualization generated with graph-tool.

The final strength of graph-tool is in visualization. I typically use other programs to visualize graphs, but it is undeniable that graph-tool is lightyears ahead any other library you can think of. Figure 53.2 is an example of what you can do with only minimal effort.

If you want to stay in Python and have performance and a bundle of graph utilities – rather than a hand-holding library who thinks for you, but also limits you – graph-tool is the way to go. Graph-tool is, in my movie director analogy, Werner Herzog. Extremely prolific and productive, but you already know that everything you're going to see will be heavily influenced by his charming accent.

iGraph

Among all the alternatives, iGraph^{5,6} is certainly the most versatile tool. It combines the strengths – and weaknesses – of Networkx and graph-tool. On the one hand, it is a surprisingly complete tool with lots of implemented functions – just like Networkx –, and it is pretty efficiently written – like graph-tool. Other advantages reside in the fact that the library is available on a vast array of platforms: you can use it both in Python and in R. You can even import it directly as a C library. Thus, if you are capable of writing in C, you can probably cook up a customized analysis using the power of iGraph that cannot

⁵ Gabor Csardi and Tamas Nepusz. The igraph software package for complex network research. *InterJournal, Complex Systems*, 1695(5):1–9, 2006

⁶ <https://igraph.org/>

be beaten in terms of running time.

That said, iGraph is not the be all end all of network analysis. As I mentioned, it is available on R. In fact, I'd venture the guess that it was developed primarily for R. And I am personally incompatible with R – some people love it, others like me cannot really understand it. To me, the interface of the library makes no sense. Function names, parameter passing, how things are stored and retrieved from objects: it is all in R style, which my brain unfortunately translates to “incomprehensible randomness”. The fact that it is possible to import iGraph in Python should not fool you: it is not pythonic at all, and the Python code you end up writing while using iGraph doesn't even look like Python (for some it is a plus, but not for me). You've been warned.

An ironic note of merit to the documentation. Writing documents is hard. Reading and understanding them is, at least for a dense researcher like me, even harder. Luckily, sometimes iGraph's documentation brightens your day with timeless comedic gems such as:

`layout_on_sphere` places the vertices (approximately) uniformly on the surface of a sphere, this is thus a 3d layout. It is not clear however what “uniformly on a sphere” means.

Gee, thank you, it's refreshing to see that not even who developed this function knows what the function is doing. Continuing my movie directors analogy, iGraph is David Lynch: probably the only one able to do what he is doing, but good luck knowing what's going on when you look at something made by him.

The fact that I'm badmouthing iGraph so hard and yet I am including it in the book and I use it should really convince you that it is a fundamental tool. If I could live without it – trust me – I would. But I can't, because sometimes it is the only thing that will save you.

Julia

Julia is a more recent alternative to Python. Like Python, it is a general purpose programming language, but it is particularly geared towards numerical analysis and so it is useful for data science. I call this section “Julia” rather than using the specific name of the library because Julia's library design is minimalist and modular. Each library will implement only a very specific set of things and you will find yourself having to import several libraries to make a complete network analysis pipeline. The libraries you'd find yourself using most often for your tasks are:

- `Graphs.jl` (<https://juliagraphs.org/Graphs.jl/stable/>) for basic graph models and operations;

- GraphIO.jl (<https://github.com/JuliaGraphs/GraphIO.jl>) to read/write graphs to/from memory;
- A bunch of libraries for plotting (https://juliagraphs.org/Graphs.jl/stable/first_steps/plotting/);
- A bunch of libraries if you want to have weighted or heterogeneous graphs (<https://juliagraphs.org/Graphs.jl/stable/ecosystem/graphtypes/>);
- Laplacians.jl (<https://danspielman.github.io/Laplacians.jl/dev/>) for some specific advanced linear algebra operations;
- And others that you can find in the JuliaGraphs GitHub repository collection (<https://github.com/orgs/JuliaGraphs/repositories>).

The advantages of Julia are that, in general, code will run faster than Python, all things being equal – I'll show an example later. Also, some of the code that has been implemented in Julia cannot be found anywhere else – at least that I know of. For instance, I know of no other way to get Laplacian solvers than using Laplacians.jl (unless I were to implement them myself, obviously). Julia comes at the disadvantages that it compiles on the fly, so the first time you run a piece of code it might take a long time, while you wait for the compilation. Also, Julia's programming logic is different than Python. If you come from Python you will sometimes be surprised by the behavior of Julia, findings in your variables unexpected values because of how and where they were initialized. Julia for me is like Ari Aster: the hot new kid on the block who's doing amazing stuff, but you wonder whether you've become too old for that.

Torch Geometric

One thing you should always remember is that networks and graphs are, at the end of the day, matrices – remember Chapter 8. I try to ignore this fact as much as I can, but it is an undeniable truth. Sometimes, the best thing you can do is to treat them as such, and to start doing some good old linear algebra. Which means that your toolbox can include specialized software like Matlab or Octave. There are in fact, network scientists who are able to do everything they need to do exclusively in these programming environments. I always look at them in awe, not knowing if I do so out of being fascinated or terrified by them.

If you are using Python, you cannot live without learning at least the basics of Numpy and Scipy^{7,8}, especially when it comes to use sparse matrices. Pandas⁹ is a good tool as well, because you can

⁷ Pauli Virtanen, Ralf Gommers, Travis E Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, et al. Scipy 1.0: fundamental algorithms for scientific computing in python. *Nature methods*, pages 1–12, 2020

⁸ <https://www.scipy.org/>

⁹ <https://pandas.pydata.org/>

use it to pivot effortlessly between dealing with networks as edge lists and as matrices. Networkx can convert to and from its data structures into Numpy, Scipy, and Pandas.

However, the real game changes is the ability of performing matrix operations on GPUs. If the algorithm you're running can be expressed as a series of matrix multiplications, then you should load your graph onto your graphic card using PyTorch Geometric (<https://pytorch-geometric.readthedocs.io/en/latest/>) and get it done stupendously efficiently. The downside is that you need to get the whole GPU computing stack to work: get Torch to work, install your GPU compute libraries (such as CUDA) and so on. This can be daunting, but the rewards are incredible.

To give you an idea, I tried to calculate the effective resistance on a graph of 3,000 nodes in all the frameworks I could. Figure 53.3 shows you how much time they took. I don't include graph-tool because it doesn't have a native way of calculating effective resistance and so you'd do it in numpy/scipy anyway. The runtimes I include here are only for the calculation of the matrix, and they do not take into account the read/write time nor the building of the basic data structure – but it does count the calculation of the Laplacian.

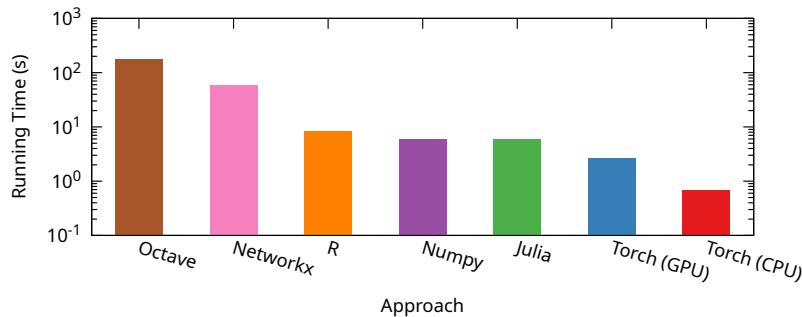


Figure 53.3: The running times (y axis) to compute the effective resistance matrix in different frameworks (x axis and bar color).

Note that the figure has a logarithmic y axis. This shouldn't be taken as a formal benchmark, you know this operation might be a bit weird because the CPU version of torch geometric was faster than the GPU one. Another reason is that, normally, Octave is good for matrix operations, but its pseudoinverse is very slow, taking 3 minutes. However, just to give an idea, Networkx takes 60 seconds, but you can cut that by a factor of ten to 6 second with a pure numpy/scipy operation – or working with Julia, which has a slight edge over numpy. You can cut by another factor of ten to 0.6 seconds (!!) using torch geometric.

So working with torch geometric is like watching an Alex Garland movie: you know you're going into the deep end, Ex Machina style.

Other

Networkx, graph-tool, iGraph, and Torch Geometric are the four horsemen of network analysis: most of the times, if you need a library, one of them will cover you. That is not to say they are the only things you should know. Here I group a bunch of miscellanea that could come in handy sooner or later.

There are a few competing libraries for doing network analysis. Stanford Network Analysis Project^{10,11} (SNAP) comes to mind. This is another C++ library, thus it competes more directly with iGraph. It is owned by a research group at Stanford, which is both a blessing and a curse. It contains mostly implementations of the analyses developed in that research group, which are great and extremely useful. This, however, comes to the cost of completeness. It is very focused.

Specializing on multilayer networks, there's the Multinet package¹². This will help you deal with this complex data format, and it has versions for both R¹³ and Python¹⁴.

Then there are packages that provide specialized utilities. There are probably more out there than I can count, so I'm making only one example to make you aware of the wealth of useful things that you could find. The powerlaw package^{15,16} allows you to perform statistical testing to verify whether your degree distribution is really a power law and cannot be explained by more mundane generating processes. It is what we rely on in Section 9.4.

Other notable libraries include:

- CDlib^{17,18}, specializing on implementing community discovery algorithms. It includes more than 50 of them, plus around 30 quality functions you can use to evaluate them (Chapter 36);
- NDlib^{19,20}, focusing on models of epidemics/spreading events on networks (Part VI);
- DyNetX²¹, which extends networkx adding to it the ability of dealing with temporal/dynamic networks.

53.2 Software

So far, we've seen software libraries: additional packages for programming languages that you can use to code your own solutions to network analysis problems. Sometimes, you instead want a fully-fledged software, possibly with a graphical user interface, to operate a set of standard operations. This is what we deal with in this section.

¹⁰ Jure Leskovec and Rok Sosič. Snap: A general-purpose network analysis and graph-mining library. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 8(1):1–20, 2016

¹¹ <https://snap.stanford.edu/>

¹² Matteo Magnani, Luca Rossi, and Davide Vega. Analysis of multiplex social networks with r

¹³ <https://cran.r-project.org/web/packages/multinet/index.html>

¹⁴ <https://pypi.org/project/uunet/>

¹⁵ Jeff Alstott and Dietmar Plenz Bullmore. powerlaw: a python package for analysis of heavy-tailed distributions. *PLoS one*, 9(1), 2014

¹⁶ <https://github.com/jeffalstott/powerlaw>

¹⁷ Giulio Rossetti, Letizia Milli, and Rémy Cazabet. Cdlib: a python library to extract, compare and evaluate communities from complex networks. *Applied Network Science*, 4(1):52, 2019

¹⁸ <http://cdlib.readthedocs.io>

¹⁹ Giulio Rossetti, Letizia Milli, Salvatore Rinzivillo, Alina Sirbu, Dino Pedreschi, and Fosca Giannotti. Ndlib: a python library to model and analyze diffusion processes over complex networks. *International Journal of Data Science and Analytics*, 5(1):61–79, 2018

²⁰ <http://ndlib.readthedocs.io>

²¹ <http://dynetx.readthedocs.io>

For Visualization

By far, the software I use the most is Cytoscape^{22,23}. Mostly, I use it for network visualization. The visual style of Cytoscape is based on the Protovis Java library²⁴, which is one of the ancestors of D3^{25,26} – and it shows. The visual style of Cytoscape is really good, and you can customize a large quantity of visual attributes relatively easily.

Cytoscape supports some basic network analysis. You can calculate a bunch of node, edge, and network statistics, the ones you'd come up first in your exploratory data analysis phase – nothing too fancy. This analytic capability is mostly there only to allow you to use node and edge statistical properties to augment your visualization. Since version 3.8, Cytoscape shows fewer plots. For instance you cannot see any more the distribution of shortest path lengths. Moreover, I can't seem to be able to show them in a log-log scale – it was possible before. However, the graphical quality of the plots greatly improved, and now they are interactive, allowing you to manipulate them to select nodes in the networks.

Another major strength of Cytoscape is its good selection of plugins. Just like Networkx, it supports a healthy ecosystems of contributors. The Cytoscape community skews heavily on the biological network crowd: protein-protein networks, gene interactions, and the like. Thus, if you're part of that community, you will likely find everything you need for Cytoscape. If you're not part of that community, the coverage is a bit more spotty.

Cytoscape comes with an API interface and a console. This means that you can write your customized piece of code that can use Cytoscape as a utility to augment your visualizations. I haven't tested it myself, but it is a nice option to have, if you're the tinkerer kind of analyst. Moreover, Cytoscape understands relatively advanced graph file formats such as GraphML²⁷ and XGMML²⁸. These are XML dialects specifically developed for graphs. They allow you to store a comprehensive list of node and edge attributes, which you can use to directly encode how your graph is supposed to look like. This means that you can transport your Cytoscape visualizations to any other software which understands them.

Finally, on the downsides, two more quick notes. First, it is a bit annoying to install, because at any given time it will rest on the previous long term support Java version (at the time of writing it is Java 11, but for most of 2020 you needed Java 8, which came out in 2014 – i.e. you were running 6 year old code). Second, I find Cytoscape to be a bit on the buggy side, with random mouse focus fails when selecting/editing text/nodes. This might be my personal experience using it on Linux, which is probably not as well supported as the ver-

²² Paul Shannon, Andrew Markiel, Owen Ozier, Nitin S Baliga, Jonathan T Wang, Daniel Ramage, Nada Amin, Benno Schwikowski, and Trey Ideker. Cytoscape: a software environment for integrated models of biomolecular interaction networks. *Genome research*, 13(11):2498–2504, 2003

²³ <https://cytoscape.org/>

²⁴ Michael Bostock and Jeffrey Heer. Protovis: A graphical toolkit for visualization. *IEEE transactions on visualization and computer graphics*, 15(6):1121–1128, 2009

²⁵ Michael Bostock, Vadim Ogievetsky, and Jeffrey Heer. D³ data-driven documents. *IEEE transactions on visualization and computer graphics*, 17(12):2301–2309, 2011

²⁶ <https://d3js.org/>

²⁷ Ulrik Brandes, Markus Eiglsperger, Jürgen Lerner, and Christian Pich. *Graph markup language (GraphML)*. 2013

²⁸ John Punin and Mukkai Krishnamoorthy. Xgmml (extensible graph markup and modeling language), 2001

sions for other major OS platforms. In any case, things have greatly improved over the years, so I like the directions in which it's going. Cytoscape is, in other words, Peter Jackson: he might not be perfect, he might have defects, but boy are his movies nice to look at!

The main alternative to Cytoscape is Gephi^{29,30}. In fact, calling it an "alternative" might even be unfair: my sense is that Gephi is actually *more* popular than Cytoscape among network scientists. However, that is not what I started using during my PhD and so I never ended up installing it. So I don't have any specific way to compare their relative strengths and weaknesses. Chances are that, for 99% of visualization tasks you will find yourself doing, the two programs can be considered equivalent. Gephi is like Guillermo Del Toro: I am unable to tell him apart from Peter Jackson. Regarding the topic of file formats, Networkx can read the GEXF file format, which is the one Gephi uses to save your network visualizations.

Muxviz^{31,32} is another great piece of software. Muxviz covers a slightly different angle from Cytoscape or Gephi. First, even if I classify it in the visualization subsection, it is much more analysis-oriented. In fact, it requires a lot of analytical power installed on your machine (Octave and R for instance). And you might find yourself using the command line interface more than the graphical interface. In this sense, it could have been listed as a library in the previous section.

More importantly, Muxviz is much more specialized. It has a specific focus on multilayer networks (Section 7.2). Which is a good thing, because Cytoscape is not very good for visualizing them. As far as I know, with Cytoscape the only choice you have is to either visualize them with a multigraph, or visualizing one layer at a time and then use a lot of elbow grease to piece the layers together. Muxviz, instead, supports them natively, and it is thus a very complementary choice if you want to be prepared for all the layers life will throw at you.

Finally, there's NetLogo^{33,34}. I frankly don't know where to classify it, because it is a weird mix of everything. First and foremost, NetLogo is a programming language. It is explicitly designed to facilitate the simulation of agent-based models. This includes all sorts of models, not necessarily the ones involving a network. Thus it is a more general tool, which allows you to do more than what this book focuses on.

Secondarily, you can use NetLogo for visualizing the effects of specific network processes. If you follow the link I provide, you can access NetLogo Web, a collection of simulations programmed in NetLogo that allows you to play with a bunch of different models. For instance, you can run a SIR model (Section 20.3), modifying its

²⁹ Mathieu Bastian, Sébastien Heymann, Mathieu Jacomy, et al. Gephi: an open source software for exploring and manipulating networks. *Icwsm*, 8(2009): 361–362, 2009

³⁰ <https://gephi.org/>

³¹ Manlio De Domenico, Mason A Porter, and Alex Arenas. Muxviz: a tool for multilayer analysis and visualization of networks. *Journal of Complex Networks*, 3(2):159–176, 2015c

³² <http://muxviz.net/>

³³ Seth Tisue and Uri Wilensky. NetLogo: A simple environment for modeling complexity. In *International conference on complex systems*, volume 21, pages 16–21. Boston, MA, 2004

³⁴ <https://ccl.northwestern.edu/netlogo/>

parameters and tracking the effects of your actions. NetLogo is a godsend for all visual thinkers who need to *see* things happening in front of them to really understand them.

For Analysis

There are many pieces of software out there that will allow you to perform network analysis and are commonly used by network professionals. They are far more than I can include here. So I will limit myself to those with which I had some personal experience.

The programs I talk about here are the ones that primarily provide analytic power. You can visualize networks with them, but you should not do that. Their visualization capabilities are not the main focus of the software, and are there mostly for you to get a quick sense of what sort of analyses you should ask the program to perform.

I think the program for network analysis I stumble the most upon in the literature is Pajek^{35,36}. Pajek allows you to perform a vast array of network analysis, ranging from classical social science ones, to more computer science-y ones – like community discovery. Pajek comes in different versions: Pajek, Pajek XXL, and Pajek 3XL. The main difference between the versions is the capability of handling larger and larger networks. The idea is that you would perform the memory-intense analyses on the XL versions of Pajek and then import the results for further investigation in the standard version of the program.

Pajek is such a popular program that its own specific file format is compatible with most of the software libraries I mentioned earlier. Both Networkx and iGraph have functions that will allow you to import networks saved in Pajek's file format. Pajek is Lars von Trier: perfect for geeking out every possible detail, but not the prettiest thing to look at.

A popular alternative from Pajek is UCINET^{37,38}. UCINET's strength is in its deep dive into the *social* branch of social network analysis. It is possibly the most comprehensive tool for social scientists to use.

As a result, its coverage of the more computer science and physics branches is less than optimal. UCINET works best with small networks, it is not particularly well optimized for large scale analysis, and will lack some of the typical algorithms you might expect to find after reading this book. However, my biggest gripe with it is probably the fact that – differently from almost everything I mentioned so far – UCINET is not a free program. If you're a full time student, it will cost \$40. UCINET is George Méliès: an immortal classic, but

³⁵ Wouter De Nooy, Andrej Mrvar, and Vladimir Batagelj. *Exploratory social network analysis with Pajek*. Cambridge University Press, 2018

³⁶ <http://mrvar.fdv.uni-lj.si/pajek/>

³⁷ Stephen P Borgatti, Martin G Everett, and Linton C Freeman. Ucinet for windows: Software for social network analysis. 2002

³⁸ <https://sites.google.com/site/ucinetsoftware/home>

probably not the style you want to adopt in 2024.

Finally, I should mention NodeXL^{39,40}. NodeXL is a weird animal, which lives on the border between being a software analysis tool with a graphical user interface like Pajek, and a library like NetworkX. NodeXL is a graphical front-end that integrates network analysis into Microsoft Excel. Excel is a phenomenal tool, easily the best and most used software Microsoft has ever written. Excel allows you to perform powerful and sophisticated analysis tasks. There are people whose entire careers could be summed up by a handful of painstakingly crafted Excel spreadsheets. So it is no wonder that there is demand for integrating network analysis in Excel. NodeXL fills that niche.

Community Discovery

I create this special subsection to focus exclusively of implementations of algorithms solving the community discovery problem. This is easily the largest subfield of network analysis. Thus this subsection satisfies two needs. First, it gives you an idea about the immense wealth of code that cannot find space in generic libraries/software. Second, it contains the necessary references to the algorithms I consider in my algorithm similarity network that was included in Section 35.6.

The way this subsection works is as follows. Now I will list a bunch of labels that are consistent with Figure 35.15. For each label, I tell you where to find the implementation I used to build that figure. The general disclaimer is that, of course, some of these links are bound to break in the future. I accessed them last time around November 2018, so the Internet Archive could help.

- edgebetween, fastgreedy, hrg, labelperc, leadeig, louvain, spinglass, walktrap: igraph implementation in R (<https://igraph.org/r/>).
- mcl: <https://www.micans.org/mcl/#source>.
- tabu, extr: options #5 and #6 in <http://deim.urv.cat/~sergio.gomez/radatools.php>.
- ganet, ganet+, moganet: <http://staff.icar.cnr.it/pizzuti/codes.html>.
- ganxis: <https://sites.google.com/site/communitydetectionslpa/>.
- conclude: <http://www.emilio.ferrara.name/code/conclude/>.
- conga, copra, cliquemod, peacock: <http://gregory.org/research/networks/>.

³⁹ Derek L Hansen, Ben Shneiderman, and Marc A Smith. *Analyzing social media networks with NodeXL: Insights from a connected world*. Morgan Kaufmann, 2010

⁴⁰ <http://nodexlgraphgallery.org/>

- mlrmcl: <https://sites.google.com/site/stochasticflowclustering/>.
- metis: <http://glaros.dtc.umn.edu/gkhome/metis/hmetis/download>.
- slpa, fluid, kerlin, kclique: networkx implementation in python (<https://networkx.github.io/documentation/stable/>).
- pmm: http://leitang.net/heterogeneous_network.html.
- crossass: <https://faculty.mccombs.utexas.edu/deepayan.chakrabarti/software.html>.
- demon: http://www.michelecoscia.com/?page_id=42.
- bigclam, agm: part of the SNAP library (<https://snap.stanford.edu/>).
- hlc: <http://barabasilab.neu.edu/projects/linkcommunities/>.
- tiles: <https://github.com/GiulioRossetti/TILES>.
- oslom: <https://sites.google.com/site/andrealancichinetti/software>.
- kmeans, dbscan, ward, agglomerative, spectral, meanshift, affinity, birch: <http://scikit-learn.org/stable/modules/classes.html#module-sklearn.cluster>.
- code-dense: <https://link.springer.com/article/10.1007/s10618-014-0373-y>.
- moses, collapsed-sbm: <https://sites.google.com/site/aaronmcdaid/downloads>.
- gce: <https://sites.google.com/site/greedycliqueexpansion/>.
- ilcd: <http://cazabetremy.fr/rRessources/iLCD.html>.
- svinet, mmsb: <https://github.com/premgopalan/svinet>.
- bnmtf: <http://www.cse.ust.hk/~dyyeung/code/BNMTF.zip>.
- rmcl: https://rdrr.io/github/DavidGilgien/ML.RMCL/man/ML_RMCL.html.
- OLC: <http://www-personal.umich.edu/~mejn/OverlappingLinkCommunities.zip>.
- cme-td, cme-bu: <https://github.com/linhongseba/ContentMapEquation>
- edgeclust: http://homes.sice.indiana.edu/filiradi/Data/radetal_algorithm.tgz.

⁴¹ Santo Fortunato, Vito Latora, and Massimo Marchiori. Method to find community structures based on information centrality. *Physical review E*, 70(5):056104, 2004

- infocentr: my own implementation of the algorithm described in the original paper⁴¹.
- msg, vm: <http://www.biochem-caflisch.uzh.ch/node/385>.
- ogc: <http://tagc.univ-mrs.fr/tagc/index.php/software/ogc>.
- savi: <http://dsec.pku.edu.cn/~tieli/>.
- mixnet: <http://www.math-evry.cnrs.fr/logiciels/mixnet/mixnet>.
- vbmod: https://github.com/jhofman/vbmod_python.
- bridgebound, bagrowLocal, clausetLocal, lwplocal: <https://github.com/kleinmind/bridge-bounding>.
- netcarto: <http://seeslab.info/downloads/network-cartography-netcarto/>.
- infomap, infomap-overlap: <http://www.mapequation.org/code.html#Download-and-compile>.
- graclus: <http://www.cs.utexas.edu/users/dml/Software/graclus.html>.
- graclus2stage: my own implementation of the algorithm described in the original paper⁴².
- fuzzyclust: <https://github.com/ntamas/fuzzyclust>.
- linecomms: <https://sites.google.com/site/linegraphs/> (to generate the line graph + igraph's implementation of Louvain).

It's now time to move to a section dedicated not to code, but to data. However, before doing so, I'll give you a small preview. In the paper building the algorithm similarity network, I test all these algorithms on 819 real world networks that I use as a benchmark. These networks are taken from data kindly shared by the authors of a bunch of papers^{43,44,45,46,47,48,49,50,51,52,53,54} and online web-pages^{55,56}.

53.3 Data

There's more to life than just the software running on your computer – or so I'm told. Many great resources that can make you a better network analyst – or even just a better person overall – can be found online. Specifically, here I focus on online network resources concerning the first ingredient of every network paper: network data. You need data to test your models, to run your algorithm, to make a compelling case of why your paper is important. Often, your study will start from a dataset you already have, or you will collect one specially tailored for your purposes. In many other cases, you simply need

⁴² Anand Narasimhamurthy, Derek Greene, Neil Hurley, and Pádraig Cunningham. Community finding in large social networks through problem decomposition. In *Proc. 19th Irish Conference on Artificial Intelligence and Cognitive Science, AICS*, volume 8, 2008

⁴³ E Gabasova. The star wars social network. *Evelina Gabasova's Blog*. Data available at: <https://github.com/evelinag/StarWars-social-network/tree/master/networks>, 2015

⁴⁴ Tom AB Snijders, Gerhard G Van de Bunt, and Christian EG Steglich. Introduction to stochastic actor-based models for network dynamics. *Social networks*, 32(1):44–60, 2010

⁴⁵ Gerhard G Van de Bunt, Marijtte AJ Van Duijn, and Tom AB Snijders. Friendship networks through time: An actor-oriented dynamic statistical network model. *Computational & Mathematical Organization Theory*, 5(2):167–192, 1999

⁴⁶ CJ Rhodes and P Jones. Inferring missing links in partially observed social networks. In *OR, Defence and Security*, pages 256–271. Springer, 2015

⁴⁷ Karine Descormiers and Carlo Morselli. Alliances, conflicts, and contradictions in montreal's street gang landscape. *International Criminal Justice Review*, 21(3):297–314, 2011

⁴⁸ Siva R Sundaresan, Ilya R Fischhoff, Jonathan Dushoff, and Daniel I Rubenstein. Network metrics reveal differences in social organization between two fission-fusion species, grevy's zebra and onager. *Oecologia*, 151(1):140–149, 2007

⁴⁹ Martin W Schein and Milton H Fohrman. Social dominance relationships in a herd of dairy cattle. *The British Journal of Animal Behaviour*, 3(2):45–55, 1955

⁵⁰ A Gimenez-Salinas Framis. Illegal networks or criminal organizations: Power, roles and facilitators in four cocaine trafficking structures. In *Third Annual Illicit Networks Workshop*, 2011

⁵¹ Andrew Beveridge and Jie Shan. Network of thrones. *Math Horizons*, 23(4):18–22, 2016

⁵² Wouter De Nooy. A literary playground: Literary criticism and balance theory. *Poetics*, 26(5-6):385–404, 1999

any network you can put your hands on that fulfills some specific constraints. This section should help you with this task.

There are many places where you can find networks directly available for download, but I start with an index: the Colorado Index of Complex Networks^{57,58} (ICON). This is quite possibly the most comprehensive index of network datasets from all domains of network science. Chances are that, if the network data is available somewhere, you can find it via ICON.

However, this is an index of network datasets, not a dataset repository, like the ones that will follow. This means that ICON is not hosting any network data itself. It rather contains the *links* to those datasets. This has advantages and disadvantages. The advantage is completeness: not all datasets can be moved from their original source and hosted somewhere else. ICON can include those datasets, while the other repositories cannot. The other side of the coin is the dynamism of the Internet. Resources get moved all the time, and not everybody does it properly via HTTP redirects – actually almost no one does it. Thus it is possible to find dead links in ICON, because the managers of the website cannot possibly constantly check that all links are working.

ICON will point you to tons of resources from which you can actually download your data, for instance Pajek’s and UCINET’s websites. There you can find a collection of network datasets you can download, which is a nice additional resource to the software. One issue you might have with this solution is that they distribute data in their own file formats, so you might need to convert them before you can use them with another software.

Also SNAP provides network data. While there is a large overlap between what you can find in Pajek and UCINET, SNAP’s focus goes decisively more towards computer science. You will find *very* large datasets there, sometimes larger than what you can handle – at the time of writing this chapter, I believe the largest network is from Friendster, which contains more than 1.8 billion edges. Just like with the implemented functions in SNAP, also the datasets are very much focused on the ones the Stanford research group used for their publications.

Another interesting resource is Konect^{59,60}. Konect is also a Matlab package for network analysis. Since I do not use Matlab unless someone is pointing a gun at me, I have no experience with it as an analysis tool. However, I used to browse Konect daily to find and download some interesting network data. The list of available datasets, as far as I can tell, is a superset of what you can find in the websites of Pajek and UCINET, and more. The web interface is also well done, and you will be able to tell what are the main char-

⁵³ Dale F Lott. Dominance relations and breeding rate in mature male american bison. *Zeitschrift für Tierpsychologie*, 49(4): 418–432, 1979

⁵⁴ Jermain Kaminski, Michael Schober, Raymond Albaladejo, Oleksandr Zastupailo, and Cesar Hidalgo. Moviegalaxies-social networks in movies. 2018

⁵⁵ <http://vlado.fmf.uni-lj.si/pub/networks/data/bio/foodweb/foodweb.htm>

⁵⁶ <http://wwwlovre.appspot.com/support.jsp>

⁵⁷ A Clauset, E Tucker, and M Sainz. The colorado index of complex networks, 2016

⁵⁸ <https://icon.colorado.edu/>

⁵⁹ Jérôme Kunegis. Konect: the koblenz network collection. In *Proceedings of the 22nd International Conference on World Wide Web*, pages 1343–1350, 2013

⁶⁰ <http://konect.cc/>

acteristics of a network before downloading it: if it's bipartite, what its degree distribution is, etc. I said "used to browse" because I believe the project might not be online any more, perhaps the Internet Archive could help.

Network Repository⁶¹ is probably the largest repository with direct network data download – thus excluding ICON. The interface isn't as good as Konect, but it's free and it includes more network data. Tiago Peixoto of graph-tool fame also launched his own network data resource: Netzschleuder⁶², which rivals Network Repository in size. It mostly takes all the network data indexed by ICON or Konect and provides a direct download in different formats. The networks can also be directly imported in graph-tool via a function, without worrying about having the network file saved on your hard disk.

If you're specifically interested in multilayer network data, one cool resource is the Comune Lab⁶³. Comune is a project owned by the same people behind Muxviz and the two can be considered as closely integrated.

53.4 Legendary Graphs

There are some graphs that are so widely used that you don't really need to look for them in an online repository. These are the pillars on which the entire cathedral of network science is founded. They are often directly included in software and libraries, and all online self-respecting network data repositories have one or multiple copies of them. I include a few here.

The first – and by far most popular – of these legendary graphs is the Zachary Karate Club⁶⁴. This is a network of members of a karate club, connecting two members if they sparred against each other. It is often used because the network focuses on two main nodes: the coach and the president of the club. The club eventually split due to a disagreement between the two, and one can reconstruct on which side each member went by analyzing with whom they sparred. It is a classical example of community discovery. Figure 53.4 shows this beauty in all of its glory.

Aaron Clauset told me a fun fact about this network. Zachary's original paper contains a figure showing the undirected adjacency matrix of the Karate Club network, except that it's not fully undirected! One edge appears in one direction, but not in the other. This means that there are technically two Karate Club graphs, depending on whether this edge is a typo or not, one with $|E| = 77$ edges and one with $|E| = 78$ edges. The latter is the most common you'll find around, because it is the one that Mark Newman and Michelle Gir-

⁶¹ <http://networkrepository.com/>

⁶² <https://networks.skewed.de/>

⁶³ <https://comunelab.fbk.eu/>

⁶⁴ Wayne W Zachary. An information flow model for conflict and fission in small groups. *Journal of anthropological research*, 33(4):452–473, 1977

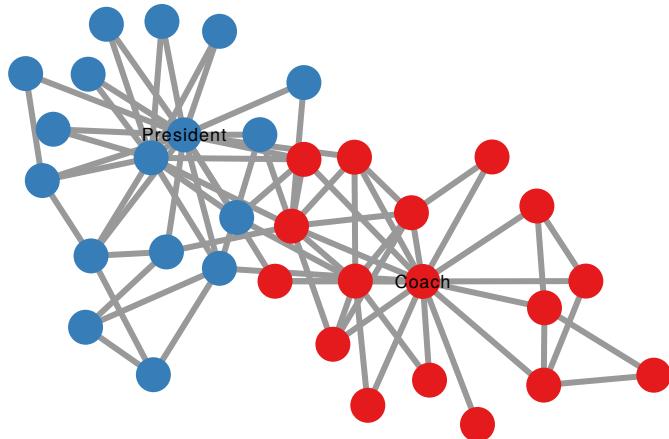


Figure 53.4: The Zachary Karate Club network. I label the nodes representing the coach and the president. The node color indicates whom the member followed after the club split up.

van used for their paper, which arguably launched the Karate Club network in the Olympus of network science.

Network scientists are obsessed with this network. It has its own t-shirt⁶⁵. They even created the Zachary Karate Club Club⁶⁶: the club of network scientists who are the first using the Zachary network as an example in their presentation at a network science conference. If you do so, you become the current holder of the Zachary Karate Club Trophy and you are responsible for handing it at the next conference you attend. This is fiercely competitive, and often you'll see this prize awarded at satellites events happening *before* the conference itself, because people will use the network as an example as soon as they can, to get their hands on the trophy.

The Network Science Society hands many prestigious awards: the Erdős-Rényi prize⁶⁷, to the career of the most outstanding network scientist under the age of forty; or the Euler award⁶⁸, to the authors of paradigm-changing publications in network science. But don't get fooled. The Zachary Karate Club Trophy is where it's at.

Another commonly used network is the one obtained from Victor Hugo's novel *Les Misérables*⁶⁹. In the network, each node is a character, and two characters are connected together if they appear in the same chapter. Also in this case the classical application is for community discovery, given that there are sets of characters closely interacting with each other that never appear in chapters with other groups of characters. Figure 53.5 shows an example. This is one of those graphs that even non-network scientist would use for examples related to other fields, for instance data visualization⁷⁰. The likely reason is the inclusion of this network in Knuth's popular book.

The college football network⁷¹ is another network commonly used for community discovery – I'm sensing a pattern here. Figure 53.6 shows it. The reason it works well is due to the way sports

⁶⁵ https://www.zazzle.co.uk/zachary_karate_club_with_label_t_shirt-235415254499870147, the label says: "If your method doesn't work on this network, then go home".

⁶⁶ <https://networkkarate.tumblr.com/>

⁶⁷ <https://netscisociety.net/award-prizes/er-prize>

⁶⁸ <https://netscisociety.net/award-prizes/euler-award>

⁶⁹ Donald Ervin Knuth. *The Stanford GraphBase: a platform for combinatorial computing*. AcM Press New York, 1993

⁷⁰ <https://bost.ocks.org/mike/miserables/>

⁷¹ https://figshare.com/articles/American_College_Football_Network_Files/93179

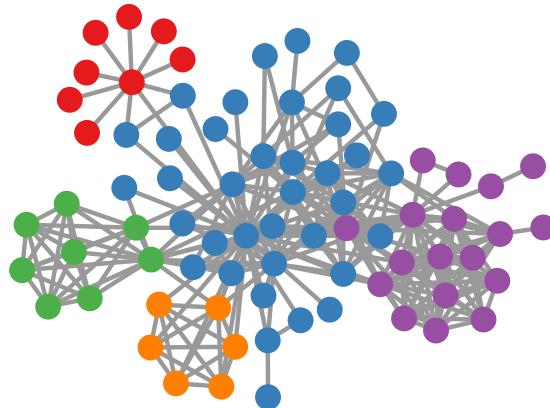


Figure 53.5: The Les Misérables network. The node color follows the community assignment from a label percolation community discovery

are organized in the United States. Usually, teams are divided in conferences and divisions. A team will play with all other teams in their division, but only with a selected number of teams in the same conference and almost no team from the other conference. This creates a nice hierarchical community structure. There is also an overlap, as the most successful teams will then access to the finals and thus play a significant number of matches with teams from the other conference.

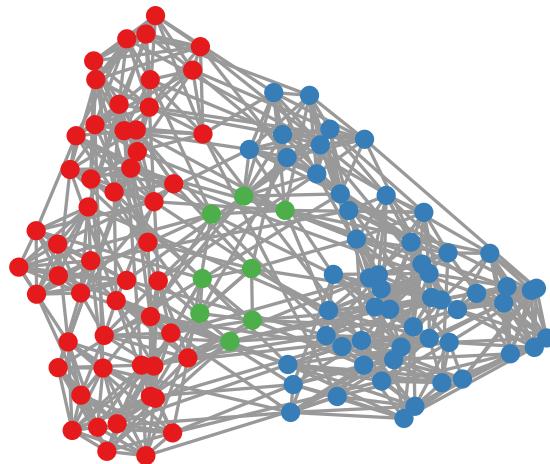


Figure 53.6: The Football network. The node color follows the community assignment from a label percolation community discovery

Other examples of networks I'm not going to discuss in details are:

- Florentine families⁷²: each node is a family from Renaissance Florence, and families are connected if there is a marriage tie between them (I used this network in Section 19.2 when talking about ERGMs);
- Davis Southern women social network⁷³: a bipartite network, connecting 18 women to 14 informal social events they attended;
- C. Elegans: this is not a single graph, it is actually multiple. We

⁷² Ronald L Breiger and Philippa E Pattison. Cumulated social roles: The duality of persons and their algebras. *Social networks*, 8(3):215–256, 1986

⁷³ Allison Davis, Burleigh Bradford Gardner, and Mary R Gardner. *Deep South: A social anthropological study of caste and class*. Univ of South Carolina Press, 1941

have extracted all possible ways to represent this poor little worm
in networks forms, from a neural network to protein-protein
interaction networks.

54

Glossary

A

Actor: The entity to which nodes in different layers in a multilayer network refer to. It can be considered as the connected component formed when using exclusively inter-layer couplings.

Acyclic Graph: See Tree.

Adjacency Matrix: A matrix where each row and column correspond to a node in the graph. The A_{uv} entry of the matrix is one if nodes u and v are connected, zero otherwise.

Adjusted Mutual Information: A uniform random variable will have some mutual information with a non-random variable. In AMI, by definition, if there is no relation between the two variables the result will be zero. Thus, AMI is mutual information adjusted for chance.

Arborescence: A directed tree in which all nodes have in-degree of one, except the root, which has in-degree of zero.

Arborescence Forest: A graph with multiple weakly connected components, each one of them being an arborescence.

Assortativity: The tendency of nodes to connect with other nodes carrying similar attributes. Synonym of homophily.

Arc: See Edge.

Average Path Length: The sum of the lengths of all shortest paths in a network over the total number of such paths.

B

Balanced Graph: A directed graph whose in- and out-degree sequences are the same.

Betweenness Centrality: Normalized number of shortest paths passing through the node.

Biclique: A clique in a bipartite network.

Bipartite Network: A network with two types of nodes and whose edges can only connect two nodes of different type.

Breadth First Search: The exploration of a graph by exploring all neighbors of a node before moving on the the neighbors of the next node.

C

Chain: A set of nodes that can be ordered, and each node is connected only to its predecessor – except the first node – and its successor – except the last node.

Clique: A set of nodes where all possible edges are present, i.e. each node in a clique is connected with each other node in the same clique.

Closeness Centrality: Normalized inverse of the average shortest path length from a node to all other nodes in the network (or connected component).

Commute Time: the expected length of a random walk starting from node u and coming back to u .

Complement Graph: given a graph G , its complement is a graph where we remove all edges from G and we connect all pairs of nodes that were not connected in G .

Complement of the Cumulative Distribution: A plot telling you the fraction of points with value equal to or greater than x .

Connected Component: The maximal (sub)set of nodes in a network that can all reach each other through walks.

Connected Network: A network composed by a single connected component.

Connection: see Edge.

Convex Network: A network whose all connected subgraphs are convex, i.e. a tree of cliques.

Convex Subgraph: A subgraphs that contains all shortest paths existing in the main network between its nodes.

Coupling Strategy: The way nodes belonging to the same actor connects to each other across layers in a multilayer network. Example: clique, chain, star.

Cumulative Distribution: A plot telling you the fraction of points with value lower than x .

Cycle: A path in which the starting and ending node is the same.

Cyclic Graph: A graph containing at least a cycle.

D

Degree: The number of edges a node has.

Degree Matrix: A matrix whose diagonal entries are the degrees of the corresponding nodes and the rest of the matrix is filled with zeros.

Depth First Search: The exploration of a graph by exploring as far as possible along a branch before backtracking.

Diameter: The length of the longest shortest path in a network.

Digraph: See Directed Graph.

Directed Acyclic Graph: A directed graph which does not contain a cycle.

Directed Cyclic Graph: A directed graph containing a cycle.

Directed Edge: A non-reciprocal edge, which implies a relationship that is not symmetric.

Directed Graph: A graph containing directed edges.

Directed Tree: A directed graph which would not contain a cycle even if we were to ignore edge directions.

Disassortativity: The tendency of nodes to connect with nodes with unlike attributes. Opposite of homophily.

Dynamic Network: A network whose edges can become active and/or inactive at different moments in time, usually represented as edge attributes.

E

Edge: The interaction between two nodes, usually represented as a pair of nodes.

Effective Resistance: the electric resistance between two points in an electric circuit (represented by nodes in G) assuming each edge in G is a 1 Ohm resistor.

Ego Network: A network focused on a node (ego). It contains the ego node, all his neighbors, and all the connections between these nodes.

Eigenvalue: Given a matrix A , an eigenvalue of A is the scaling factor of one of its eigenvectors, i.e. if $Av = \lambda v$ for some vector v , then λ is an eigenvalue of A .

Eigenvector: Given a matrix A , an eigenvector of A is a special vector that only changes its length – but not its direction – when multiplied to A , i.e. if $Av = \lambda v$ for some value λ , then v is an eigenvector of A .

F

Fiedler Vector: The second smallest eigenvector of the Laplacian.

Forest: A network composed by more than one connected component, each one of them being a tree.

G

Giant Connected Component: In real world networks, the largest component which holds the majority of the nodes of a network.

Graph: A set of nodes connected by a set of edges.

H

Hairball: Incoherent ball of nodes and edges, typical result of a naive visualization of a network too large and dense to be spread

out in a two dimensional plane. Also known as ridiculogram or spaghettiograph.

Heterogeneous Network: A network with multiple node and edge types.

Heterophily: The tendency of nodes to connect to nodes with unlike attributes. Synonym of disassortativity.

Hitting Time: the expected length of a random walk starting from node u and visiting v for the first time.

Homophily: The tendency of nodes to connect to other nodes with the same or similar attributes. Synonym of assortativity.

Hub: A central node with many connections.

Hyperedges: Edges that can connect more than two nodes at the same time.

Hypergraph: A graph containing hyperedges.

I

Identity Matrix: A matrix with ones on the diagonal and zeros everywhere else.

In-Component: A weakly connected component in a directed graph whose paths can reach a strongly connected component but will never reach back.

Incidence Matrix: A matrix with nodes on the rows, edges on the columns, and whose non-zero entries report to which edges a node is connected.

Induced Subgraph: a subgraph of an original graph formed from a subset of the vertices of the graph and all of the edges connecting pairs of vertices in that subset.

Interlayer Coupling: In a multilayer network, the special connections connecting the nodes belonging to the same actor.

Isolated Node: a node with zero degree.

K

k-Clique: A clique of k nodes.

k-core: Set of nodes that have a minimum degree of k , once you recursively remove from the network all nodes that have $k - 1$ connections or fewer.

L

Laplacian: The matrix obtained subtracting the adjacency matrix from the degree matrix.

Lattice: A simple graph in which nodes are uniformly distributed in a n-dimensional space and they connect with a given number of their nearest neighbors.

Leaf Node: A node with degree equal to one.

Left Eigenvector: An eigenvector obtained multiplying the matrix from the left. If $vA = v\lambda$, the v is a left eigenvector of A , in contrast with right eigenvectors.

Line Graph: The graph that represents the adjacencies between edges of an undirected graph: each edge of the original graph is a node in the line graph, and two nodes in the line graph connect if they have a node in common in the original graph.

Link: See Edge.

M

Maximal Clique: A clique in a network to which you cannot add any nodes and still obtain a clique.

Maximum Spanning Tree: A spanning tree of a weighted graph which has the highest edge weight sum of all spanning trees for that graph.

Metapath: A path in a heterogeneous network, including nodes of different types.

Minimum Spanning Tree: A spanning tree of a weighted graph which has the lowest edge weight sum of all spanning trees for that graph.

Multidimensional Network: A network with multiple edge types. A subtype of multilayer networks.

Multigraph: A graph in which there can be multiple parallel edges between the same two nodes.

Multilayer Network: A network in which nodes can connect to each other with different types edges, and can have multiple identities.

Multipartite Network: A network with two or more node types, and whose edges can only be established between nodes of unlike type.

Multiplex Network: A network with multiple edge types. A subtype of multilayer networks.

Mutual information: A relatedness measure between two random variables, namely the number of bits of information you obtain about a random variable if you know the other one.

N

n, m -Clique: A biclique with n nodes of type 1 and m nodes of type 2.

Neighbor: A node directly connected to your focus node by an edge.

Node: The fundamental interacting unit of a graph. In a social network, it will be a person. In the Internet network, it will be a router.

Normalized Mutual information: Equivalent to Mutual Information, normalized so that it takes values between zero and one.

O

Out-Component: A weakly connected component in a directed graph which can receive paths from a strongly connected component but cannot reach it back.

P

Parallel edges: Two (or more) edges established between the same pair of nodes.

Path: A walk with no repeating nodes.

Planar Graph: A graph you can draw on a 2D plane without intersecting any edges.

Probabilistic Network: a network whose edges have probabilities of existing.

R

Reverse Graph: the reverse graph of directed graph G is another directed graph where we flip all edge directions.

Ridiculogram: see Hairball.

Right Eigenvector: An eigenvector obtained multiplying the matrix from the right. If $Av = \lambda v$, the v is a right eigenvector of A , in contrast with left eigenvectors.

S

Self-loop: An edge connecting a node with itself.

Simple Path: See Path.

Simplicial Complex: A set of nodes whose connections to each other are part of a single high-order structure. Similar to hyperedges, with the constraint of being embedded in a geometric space.

Singleton: see Isolated Node.

Spaghettigraph: See Hairball.

Spanning Tree: A subgraph that is a tree and includes all of the nodes of its parent graph.

Square: A cycle of four nodes and four edges.

Star: A set of nodes with one acting as a center connected to all other nodes in the star. All other nodes have only one connection, to the star's center.

Stationary Distribution: The probability of ending in a node after a random walk of infinite length, equivalent to the degree for undirected networks.

Stochastic Adjacency: A normalized adjacency matrix, whose rows have been divided by their sum.

Strongly Connected Component: A component in a directed graph that contains paths from any node of the component to any other node of the component, respecting edge directions.

Subgraph: A graph whose sets of nodes and edges are completely included in the node and edge sets of another graph.

T

Temporal Network: See Dynamic Network.

Tree: A graph containing no cycles.

Triad: A connected graph with three nodes and two edges.

Triangle: A connected graph with three nodes and three edges.

Tripartite Network: A network with three node types and whose edges can only be established between nodes of unlike type.

U

Undirected Network: A network whose edges are all without direction, i.e. all connections are symmetric.

Uniform Hypergraph: A hypergraph containing hyperedges with the same cardinality – i.e. each hyperedge contains the same number of nodes.

Unipartite Network: A network with only one node type, without restrictions on how nodes connect – in direct contrast with a bipartite network.

Unweighted Network: A network whose edges have no weight – or where all weights are equal to one. In direct contrast with a weighted network.

V

Vertex: See Node.

W

Walk: A sequence of nodes. Two consecutive nodes in the sequence must be adjacent.

Weakly Connected Component: A component in a directed graph that contains paths from any node of the component to any other node of the component, but only if we ignore edge directions.

Weighted Adjacency: An adjacency matrix which is not binary. Each cell contains the weight of its corresponding edge.

Weighted Edge: An edge with quantitative information, determining its strength.

Weighted Network: A network containing weighted edges.

Most Common Abbreviations

A

A: Adjacency matrix.

AMI: Adjusted Mutual Information.

APL: Average Path Length.

APL_v : The Average Path Length of paths starting from node v .

AUC: Area Under the (ROC) Curve, a common way to estimate prediction performance.

B

BFS: Breadth First Search.

C

C: The commute time matrix, $C_{u,v} = H_{u,v} + H_{v,u}$, with H being the hitting time matrix.

CC: Global Clustering Coefficient.

CC_{avg} : Average Clustering Coefficient of the network, the average of CC_v for all v s in the network.

CC_v : Local Clustering Coefficient of node v .

CCDF: Complement of the Cumulative Distribution Function.

CDF: Cumulative Distribution Function.

D

D: Degree matrix.

DFS: Depth First Search.

E

E: Set of edges.

ERGM: Exponential Random Graph Model, a technique to generate synthetic graphs for statistical testing.

F

FN: False Negative.

FP: False Positive.

FPR: False Positive Rate, equal to $FP/(FP + TN)$.

G

GCC: Giant Connected Component.

GERM: Graph Evolution Rule Mining, a way to predict links in networks.

H

H: The hitting time matrix, telling you how long it'll take for a random walker to visit one node when starting from another.

I

I: The identity matrix.

K

\bar{k} : The average degree of the network.

k_v : The degree of node v .

L

L: Laplacian matrix.

M

MI: Mutual Information.

N

NMF: Non-negative Matrix Factorization.

NMI: Normalized Mutual Information.

N_u : The set of neighbors of node u .

$N_{u,l}$: In a multilayer network, the set of neighbors of node u in layer l .

P

P_{uv} : A path going from node u to node v .

PCA: Principal Component Analysis.

R

ROC: Receiver Operating Characteristic, a common way to estimate prediction performance.

S

SBM: Stochastic Block Model, a network generative model.

SCC: Strongly Connected Component.

SI: Susceptible-Infected, an compartmental epidemiology model with two states and one transition.

SIR: Susceptible-Infected-Removed, an compartmental epidemiology model with two states and two irreversible transitions.

SIS: Susceptible-Infected-Susceptible, an compartmental epidemiology model with two states and one reversible transition.

SVD: Singular Value Decomposition, an operation to decompose an arbitrary matrix into a diagonal one.

SVM: Support Vector Machine, a machine learning technique.

T

TN: True Negative.

TP: True Positive.

TPR: True Positive Rate, equal to $TP/(TP + FN)$.

V

V : Set of nodes.

W

W : Set of possible weights in a weighted network.

WCC: Weakly Connected Component.

Other Symbols

Ω : The effective resistance matrix of a graph.

Bibliography

Georges Aad, Tatevik Abajyan, Brad Abbott, Jalal Abdallah, S Abdel Khalek, Ahmed Ali Abdelalim, R Aben, B Abi, M Abolins, OS AbouZeid, et al. Observation of a new particle in the search for the standard model higgs boson with the atlas detector at the lhc. *Physics Letters B*, 716(1):1–29, 2012.

Scott Aaronson. P=?np. *Electronic Colloquium on Computational Complexity (ECCC)*, 24:4, 2017. URL <https://eccc.weizmann.ac.il/report/2017/004>.

Alan I Abramowitz and Kyle L Saunders. Is polarization a myth? *The Journal of Politics*, 70(2):542–555, 2008.

Sami Abu-El-Haija, Bryan Perozzi, Rami Al-Rfou, and Alexander A Alemi. Watch your step: Learning node embeddings via graph attention. *Advances in neural information processing systems*, 31, 2018.

Dimitris Achlioptas, Raissa M D’souza, and Joel Spencer. Explosive percolation in random networks. *Science*, 323(5920):1453–1455, 2009.

Lada A Adamic and Eytan Adar. Friends and neighbors on the web. *Social networks*, 25(3):211–230, 2003.

Lada A Adamic and Natalie Glance. The political blogosphere and the 2004 us election: divided they blog. In *Proceedings of the 3rd international workshop on Link discovery*, pages 36–43. ACM, 2005.

Lada A Adamic and Bernardo A Huberman. Power-law distribution of the world wide web. *science*, 287(5461):2115–2115, 2000.

Lada A Adamic, Rajan M Lukose, Amit R Puniyani, and Bernardo A Huberman. Search in power-law networks. *Physical review E*, 64(4):046135, 2001.

Lada A Adamic, Jun Zhang, Eytan Bakshy, and Mark S Ackerman. Knowledge sharing and yahoo answers: everyone knows something. In *Proceedings of the 17th international conference on World Wide Web*, pages 665–674, 2008.

Arun Advani and Banshi Malde. Empirical methods for networks data: Social effects, network formation and measurement error. Technical report, IFS Working Papers, 2014.

Nazanin Afsarmanesh and Matteo Magnani. Finding overlapping communities in multiplex networks. *arXiv preprint arXiv:1602.03746*, 2016.

Rakesh Agrawal, Ramakrishnan Srikant, et al. Fast algorithms for mining association rules. In *Proc. 20th int. conf. very large data bases, VLDB*, volume 1215, pages 487–499, 1994.

Nesreen Ahmed, Jennifer Neville, and Ramana Rao Kompella. Network sampling via edge-based node selection with graph induction. 2011.

Nesreen K Ahmed, Jennifer Neville, and Ramana Kompella. Space-efficient sampling from social activity streams. In *Proceedings of the 1st international workshop on big data, streams and heterogeneous source mining: algorithms, systems, programming models and applications*, pages 53–60. ACM, 2012.

Nesreen K Ahmed, Jennifer Neville, and Ramana Kompella. Network sampling: From static to streaming graphs. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 8(2):7, 2014.

Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Graph sketches: sparsification, spanners, and subgraphs. In *SIGMOD-SIGACT-SIGAI*, pages 5–14, 2012.

Yong-Yeol Ahn, James P Bagrow, and Sune Lehmann. Link communities reveal multiscale complexity in networks. *nature*, 466(7307):761, 2010.

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.

Sebastian E Ahnert. Power graph compression reveals dominant relationships in genetic transcription networks. *Molecular BioSystems*, 9(11):2681–2685, 2013.

Ravindra K Ahuja, Kurt Mehlhorn, James Orlin, and Robert E Tarjan. Faster algorithms for the shortest path problem. *Journal of the ACM (JACM)*, 37(2):213–223, 1990.

William Aiello, Fan Chung, and Linyuan Lu. A random graph model for massive graphs. In *Proceedings of the thirty-second annual ACM symposium on Theory of computing*, pages 171–180. Acm, 2000.

Edoardo M Airoldi, David M Blei, Stephen E Fienberg, and Eric P Xing. Mixed membership stochastic blockmodels. *Journal of Machine Learning Research*, 9(Sep):1981–2014, 2008.

Saeed Akhoondian Amiri, Lukasz Kaiser, Stephan Kreutzer, Roman Rabinovich, and Sebastian Siebertz. Graph searching games and width measures for directed graphs. In *LIPICS-Leibniz International Proceedings in Informatics*, volume 30. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2015.

Ricardo Alberich, Joe Miro-Julia, and Francesc Rosselló. Marvel universe looks almost like a real social network. *arXiv preprint cond-mat/0202174*, 2002.

Reka Albert. Scale-free networks in cell biology. *Journal of cell science*, 118(21):4947–4957, 2005.

Réka Albert, Hawoong Jeong, and Albert-László Barabási. Error and attack tolerance of complex networks. *nature*, 406(6794):378, 2000.

Lauren Alexander, Shan Jiang, Mikel Murga, and Marta C González. Origin–destination trips by purpose and time of day inferred from mobile phone data. *Transportation research part c*, 58:240–250, 2015.

Noga Alon, Erik D Demaine, Mohammad T Hajiaghayi, and Tom Leighton. Basic network creation games. *SIAM Journal on Discrete Mathematics*, 27(2):656–668, 2013.

Uri Alon. Network motifs: theory and experimental approaches. *Nature Reviews Genetics*, 8(6):450, 2007.

Ethem Alpaydin. *Introduction to machine learning*. 2020.

Brian Alspach. Searching and sweeping graphs: a brief survey. *Le matematiche*, 59(1, 2):5–37, 2006.

Jeff Alstott and Dietmar Plenz Bullmore. powerlaw: a python package for analysis of heavy-tailed distributions. *PloS one*, 9(1), 2014.

Taher Alzahrani and Kathy J Horadam. Community detection in bipartite networks: Algorithms and case studies. In *Complex systems and networks*, pages 25–50. Springer, 2016.

Taher Alzahrani, Kathy J Horadam, and Serdar Boztas. Community detection in bipartite networks using random walks. In *Complex Networks V*, pages 157–165. Springer, 2014.

Alessia Amelio and Clara Pizzuti. Overlapping community discovery methods: A survey. In *Social Networks: Analysis and Case Studies*, pages 105–125. Springer, 2014.

Kartik Anand and Ginestra Bianconi. Entropy measures for networks: Toward an information theory of complex topologies. *Physical Review E*, 80(4):045102, 2009.

Carolyn J Anderson, Stanley Wasserman, and Bradley Crouch. A p* primer: Logit models for social networks. *Social networks*, 21(1):37–66, 1999.

Philip W Anderson. More is different. *Science*, 177(4047):393–396, 1972.

Anton Andreychuk, Konstantin Yakovlev, Dor Atzmon, and Roni Sternr. Multi-agent pathfinding with continuous time. In *IJCAI*, volume 19, 2019.

Aamir Anis, Akshay Gadde, and Antonio Ortega. Towards a sampling theorem for signals on arbitrary graphs. In *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*, pages 3864–3868. IEEE, 2014.

Francis J Anscombe. Graphs in statistical analysis. *The american statistician*, 27(1):17–21, 1973.

Tibor Antal, Pavel L Krapivsky, and Sidney Redner. Dynamics of social balance on networks. *Physical Review E*, 72(3):036121, 2005.

Jac M Anthonisse. The rush in a directed graph. *Stichting Mathematisch Centrum. Mathematische Besliskunde*, (BN 9/71), 1971.

Naheed Anjum Arafat and Stéphane Bressan. Hypergraph drawing by force-directed placement. In *International Conference on Database and Expert Systems Applications*, pages 387–394. Springer, 2017.

Elsa Arcaute, Erez Hatna, Peter Ferguson, Hyejin Youn, Anders Johansson, and Michael Batty. Constructing cities, deconstructing scaling laws. *Journal of The Royal Society Interface*, 12(102):20140745, 2015.

Samin Aref and Mark C Wilson. Balance and frustration in signed networks. *Journal of Complex Networks*, 7(2):163–189, 2019.

Alex Arenas, Jordi Duch, Alberto Fernández, and Sergio Gómez. Size reduction of complex networks preserving modularity. *New Journal of Physics*, 9(6):176, 2007.

Alex Arenas, Alberto Fernandez, Santo Fortunato, and Sergio Gomez. Motif-based communities in complex networks. *Physics A*, 41(22):224001, 2008a.

Alex Arenas, Alberto Fernandez, and Sergio Gomez. Analysis of the structure of complex networks at different resolution levels. *New journal of physics*, 10(5):053039, 2008b.

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

Ira Assent, Andrea Wenning, and Thomas Seidl. Approximation techniques for indexing the earth mover's distance in multimedia databases. In *Data Engineering, 2006. ICDE'06. Proceedings of the 22nd International Conference on*, pages 11–11. IEEE, 2006.

Sitaram Asur, Srinivasan Parthasarathy, and Duygu Ucar. An event-based framework for characterizing the evolutionary behavior of interaction graphs. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 3(4):16, 2009.

Wirt Atmar and Bruce D Patterson. The measure of order and disorder in the distribution of species in fragmented habitat. *Oecologia*, 96(3):373–382, 1993.

James Atwood and Don Towsley. Diffusion-convolutional neural networks. In *NIPS*, pages 1993–2001, 2016.

Felix Auerbach. Das gesetz der bevölkerungskonzentration. *Petermanns Geographische Mitteilungen*, 59:74–76, 1913.

László Babai. Graph isomorphism in quasipolynomial time. In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, pages 684–697. ACM, 2016.

László Babai and Ludik Kucera. Canonical labelling of graphs in linear average time. In *20th annual symposium on foundations of computer science (sfcs 1979)*, pages 39–46. IEEE, 1979.

D Babić, Douglas J Klein, István Lukovits, Sonja Nikolić, and N Trinajstić. Resistance-distance matrix: a computational algorithm and its application. *International Journal of Quantum Chemistry*, 90(1):166–176, 2002.

Benjamin Bach, Nathalie Henry Riche, Christophe Hurter, Kim Marriott, and Tim Dwyer. Towards unambiguous edge bundling: Investigating confluent drawings for network visualization. *IEEE transactions on visualization and computer graphics*, 23(1):541–550, 2016.

Louis Bachelier. Théorie de la spéculation. In *Annales scientifiques de l'École normale supérieure*, volume 17, pages 21–86, 1900.

Giacomo Bachi, Michele Coscia, Anna Monreale, and Fosca Giannotti. Classifying trust/distrust relationships in online social networks. In *2012 International Conference on Privacy, Security, Risk and Trust and 2012 International Conference on Social Computing*, pages 552–557. IEEE, 2012.

Lars Backstrom, Cynthia Dwork, and Jon Kleinberg. Wherefore art thou r3579x?: anonymized social networks, hidden patterns, and structural steganography. In *Proceedings of the 16th international conference on World Wide Web*, pages 181–190. ACM, 2007.

Lars Backstrom, Paolo Boldi, Marco Rosa, Johan Ugander, and Sebastiano Vigna. Four degrees of separation. In *Proceedings of the 4th Annual ACM Web Science Conference*, pages 33–42. ACM, 2012.

James P Bagrow. Evaluating local community methods in networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(05):P05001, 2008.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

Bahman Bahmani, Ravi Kumar, and Sergei Vassilvitskii. Densest subgraph in streaming and mapreduce. *Proceedings of the VLDB Endowment*, 5(5), 2012.

Eytan Bakshy, Jake M Hofman, Winter A Mason, and Duncan J Watts. Everyone’s an influencer: quantifying influence on twitter. In *Proceedings of the fourth ACM international conference on Web search and data mining*, pages 65–74. ACM, 2011.

Eytan Bakshy, Solomon Messing, and Lada A Adamic. Exposure to ideologically diverse news and opinion on facebook. *Science*, 348(6239):1130–1132, 2015.

Bela Balassa. Trade liberalisation and “revealed” comparative advantage 1. *The manchester school*, 33(2):99–123, 1965.

Vladimír Baláž, Jaroslav Koča, Vladimír Kvasnička, and Milan Sekanina. A metric for graphs. *Časopis pro pěstování matematiky*, 111(4):431–433, 1986.

Duygu Balcan, Vittoria Colizza, Bruno Gonçalves, Hao Hu, José J Ramasco, and Alessandro Vespignani. Multiscale mobility networks and the spatial spreading of infectious diseases. *PNAS*, 106(51):21484–21489, 2009.

Pierre-Alexandre Balland, José Antonio Belso-Martínez, and Andrea Morrison. The dynamics of technical and business knowledge networks in industrial clusters: Embeddedness, status, or proximity? *Economic Geography*, 92(1):35–60, 2016.

Anirban Banerjee. Structural distance and evolutionary relationship of networks. *Biosystems*, 107(3):186–196, 2012.

Jørgen Bang-Jensen and Gregory Z Gutin. *Digraphs: theory, algorithms and applications*. Springer Science & Business Media, 2008.

Albert-László Barabási. *Linked: The new science of networks*, 2003.

Albert-Laszlo Barabasi. The origin of bursts and heavy tails in human dynamics. *Nature*, 435(7039):207, 2005.

Albert-László Barabási. Scale-free networks: a decade and beyond. *science*, 325(5939):412–413, 2009.

Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *science*, 286(5439):509–512, 1999.

Albert-László Barabási and Eric Bonabeau. Scale-free networks. *Scientific american*, 288(5):60–69, 2003.

Albert-Laszlo Barabási, Hawoong Jeong, Zoltan Néda, Erzsebet Ravasz, Andras Schubert, and Tamas Vicsek. Evolution of the social network of scientific collaborations. *Physica A: Statistical mechanics and its applications*, 311(3-4):590–614, 2002.

Albert-László Barabási, Chaoming Song, and Dashun Wang. Publishing: Handful of papers dominates citation. *Nature*, 491(7422):40, 2012.

Albert-László Barabási et al. *Network science*. Cambridge university press, 2016.

Paul Baran. Introduction to distributed communications networks. Technical report, Memorandum RM-3420-PR, Rand Corporation, 1964.

Michael J Barber. Modularity and community detection in bipartite networks. *Physical Review E*, 76(6):066102, 2007.

Michael J Barber and John W Clark. Detecting network communities by propagating labels under constraints. *Physical Review E*, 80(2):026129, 2009.

Pablo Barceló, Egor V Kostylev, Mikael Monet, Jorge Pérez, Juan Reutter, and Juan-Pablo Silva. The logical expressiveness of graph neural networks. In *8th International Conference on Learning Representations (ICLR 2020)*, 2020.

Mauro Barone and Michele Coscia. Birds of a feather scam together: Trustworthiness homophily in a business network. *Social Networks*, 54:228 – 237, 2018. ISSN 0378-8733.

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.

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

Baruch Barzel and Albert-László Barabási. Network link prediction by global silencing of indirect correlations. *Nature biotechnology*, 31(8):720–725, 2013a.

Baruch Barzel and Albert-László Barabási. Universality in network dynamics. *Nature physics*, 9(10):673, 2013b.

Jordi Bascompte, Pedro Jordano, Carlos J Melián, and Jens M Olesen. The nested assembly of plant-animal mutualistic networks. *Proceedings of the National Academy of Sciences*, 100(16):9383–9387, 2003.

Danielle S Bassett and Edward T Bullmore. Human brain networks in health and disease. *Current opinion in neurology*, 22(4):340, 2009.

Danielle S Bassett, Nicholas F Wymbs, Mason A Porter, Peter J Mucha, Jean M Carlson, and Scott T Grafton. Dynamic reconfiguration of human brain networks during learning. *Proceedings of the National Academy of Sciences*, 108(18):7641–7646, 2011.

Danielle S Bassett, Mason A Porter, Nicholas F Wymbs, Scott T Grafton, Jean M Carlson, and Peter J Mucha. Robust detection of dynamic community structure in networks. *Chaos Journal*, 23(1):013142, 2013.

Mathieu Bastian, Sébastien Heymann, Mathieu Jacomy, et al. Gephi: an open source software for exploring and manipulating networks. *Icwsrm*, 8(2009):361–362, 2009.

Ugo Bastolla, Miguel A Fortuna, Alberto Pascual-García, Antonio Ferrera, Bartolo Luque, and Jordi Bascompte. The architecture

of mutualistic networks minimizes competition and increases biodiversity. *Nature*, 458(7241):1018, 2009.

Vladimir Batagelj and Ulrik Brandes. Efficient generation of large random networks. *Physical Review E*, 71(3):036113, 2005.

Vladimir Batagelj and Matjaz Zaversnik. An $O(m)$ algorithm for cores decomposition of networks. *arXiv preprint cs/0310049*, 2003.

Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

Federico Battiston, Vincenzo Nicosia, and Vito Latora. Structural measures for multiplex networks. *Physical Review E*, 89(3):032804, 2014.

Federico Battiston, Vincenzo Nicosia, and Vito Latora. Efficient exploration of multiplex networks. *New Journal of Physics*, 18(4):043035, 2016.

Federico Battiston, Vincenzo Nicosia, Mario Chavez, and Vito Latora. Multilayer motif analysis of brain networks. *Chaos*, 27(4):047404, 2017.

Federico Battiston, Jeremy Guillon, Mario Chavez, Vito Latora, and Fabrizio De Vico Fallani. Multiplex core–periphery organization of the human connectome. *Journal of the Royal Society Interface*, 15(146):20180514, 2018.

Heiko Bauke. Parameter estimation for power-law distributions by maximum likelihood methods. *The European Physical Journal B*, 58(2):167–173, 2007.

Alex Bavelas. A mathematical model for group structures. *Human organization*, 7(3):16, 1948.

Marya Bazzi, Mason A Porter, Stacy Williams, Mark McDonald, Daniel J Fenn, and Sam D Howison. Community detection in temporal multilayer networks, with an application to correlation networks. *Multiscale Modeling & Simulation*, 14(1):1–41, 2016.

Stephen J Beckett. Improved community detection in weighted bipartite networks. *Royal Society open science*, 3(1):140536, 2016.

Michael Behrisch, Benjamin Bach, Nathalie Henry Riche, Tobias Schreck, and Jean-Daniel Fekete. Matrix reordering methods for table

and network visualization. In *Computer Graphics Forum*, volume 35, pages 693–716. Wiley Online Library, 2016.

Francesco Belardo. Balancedness and the least eigenvalue of laplacian of signed graphs. *Linear Algebra and its Applications*, 446:133–147, 2014.

Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *Advances in neural information processing systems*, pages 585–591, 2002.

David C Bell, Elizabeth B Erbaugh, Tabitha Serrano, Cheryl A Dayton-Shotts, and Isaac D Montoya. A comparison of network sampling designs for a hidden population of drug users: Random walk vs. respondent-driven sampling. *Social science research*, 62:350–361, 2017.

Austin R Benson, David F Gleich, and Jure Leskovec. Tensor spectral clustering for partitioning higher-order network structures. In *Proceedings of the 2015 SIAM International Conference on Data Mining*, pages 118–126. SIAM, 2015.

Austin R Benson, David F Gleich, and Jure Leskovec. Higher-order organization of complex networks. *Science*, 353(6295):163–166, 2016.

Jonah Berger and Katherine L Milkman. What makes online content viral? *Journal of marketing research*, 49(2):192–205, 2012.

Carl T Bergstrom and Jevin D West. *Calling bullshit: The art of skepticism in a data-driven world*. Random House Trade Paperbacks, 2021.

Michele Berlingerio, Francesco Bonchi, Björn Bringmann, and Aristides Gionis. Mining graph evolution rules. In *Joint European conference on machine learning and knowledge discovery in databases*, pages 115–130. Springer, 2009.

Michele Berlingerio, Michele Coscia, and Fosca Giannotti. Finding and characterizing communities in multidimensional networks. In *2011 International Conference on Advances in Social Networks Analysis and Mining*, pages 490–494. IEEE, 2011a.

Michele Berlingerio, Michele Coscia, and Fosca Giannotti. Finding redundant and complementary communities in multidimensional networks. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, pages 2181–2184, 2011b.

Michele Berlingerio, Danai Koutra, Tina Eliassi-Rad, and Christos Faloutsos. Netsimile: A scalable approach to size-independent network similarity. *arXiv preprint arXiv:1209.2684*, 2012.

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

Michele Berlingerio, Danai Koutra, Tina Eliassi-Rad, and Christos Faloutsos. Network similarity via multiple social theories. In *Proceedings of the 2013 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, pages 1439–1440, 2013b.

Michele Berlingerio, Fabio Pinelli, and Francesco Calabrese. Abacus: frequent pattern mining-based community discovery in multidimensional networks. *Data Mining and Knowledge Discovery*, 27(3):294–320, 2013c.

H Russell Bernard and Harvey Russell Bernard. *Social research methods: Qualitative and quantitative approaches*. Sage, 2013.

Timothy J Berners-Lee. Information management: A proposal. Technical report, 1989.

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.

Alessandro Bessi and Emilio Ferrara. Social bots distort the 2016 us presidential election online discussion. 2016.

Luís MA Bettencourt. The origins of scaling in cities. *science*, 340(6139):1438–1441, 2013.

Luís MA Bettencourt, José Lobo, Dirk Helbing, Christian Kühnert, and Geoffrey B West. Growth, innovation, scaling, and the pace of life in cities. *Proceedings of the national academy of sciences*, 104(17):7301–7306, 2007.

Richard F Betzel and Danielle S Bassett. Multi-scale brain networks. *Neuroimage*, 160:73–83, 2017.

Andrew Beveridge and Jie Shan. Network of thrones. *Math Horizons*, 23(4):18–22, 2016.

Kevin Beyer, Jonathan Goldstein, Raghu Ramakrishnan, and Uri Shaft. When is “nearest neighbor” meaningful? In *Database Theory—ICDT’99: 7th International Conference Jerusalem, Israel, January 10–12, 1999 Proceedings* 7, pages 217–235. Springer, 1999.

James C Bezdek. *Pattern recognition with fuzzy objective function algorithms*. Springer Science & Business Media, 2013.

Shankar Bhamidi, Guy Bresler, and Allan Sly. Mixing time of exponential random graphs. In *2008 49th Annual IEEE Symposium on Foundations of Computer Science*, pages 803–812. IEEE, 2008.

Filippo Maria Bianchi, Daniele Grattarola, and Cesare Alippi. Spectral clustering with graph neural networks for graph pooling. In *International conference on machine learning*, pages 874–883. PMLR, 2020.

Filippo Maria Bianchi, Daniele Grattarola, Lorenzo Livi, and Cesare Alippi. Graph neural networks with convolutional arma filters. *IEEE transactions on pattern analysis and machine intelligence*, 44(7):3496–3507, 2021.

Ginestra Bianconi. Statistical mechanics of multiplex networks: Entropy and overlap. *Physical Review E*, 87(6):062806, 2013.

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

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

Ginestra Bianconi and Christoph Rahmede. Network geometry with flavor: from complexity to quantum geometry. *Physical Review E*, 93(3):032315, 2016.

Ginestra Bianconi and Christoph Rahmede. Emergent hyperbolic network geometry. *Scientific reports*, 7:41974, 2017.

Ginestra Bianconi and Robert M Ziff. Topological percolation on hyperbolic simplicial complexes. *Physical Review E*, 98(5):052308, 2018.

Peter J Bickel and Kjell A Doksum. *Mathematical statistics: basic ideas and selected topics, volumes I-II package*. Chapman and Hall/CRC, 2015.

Patrick Biernacki and Dan Waldorf. Snowball sampling: Problems and techniques of chain referral sampling. *Sociological methods & research*, 10(2):141–163, 1981.

Christian Bird, David Pattison, Raissa D’Souza, Vladimir Filkov, and Premkumar Devanbu. Latent social structure in open source projects. In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, pages 24–35, 2008.

Arne Bjerhammar. Application of calculus of matrices to method of least squares: with special reference to geodetic calculations. (*No Title*), 1951.

Nils Bjorck, Carla P Gomes, Bart Selman, and Kilian Q Weinberger. Understanding batch normalization. *Advances in neural information processing systems*, 31, 2018.

Robert T Blackburn, Charles E Behymer, and David E Hall. Research note: Correlates of faculty publications. *Sociology of Education*, pages 132–141, 1978.

Neli Blagus, Lovro Šubelj, and Marko Bajec. Empirical comparison of network sampling techniques. *arXiv preprint arXiv:1506.02449*, 2015.

Vincent D Blondel, Anahí Gajardo, Maureen Heymans, Pierre Senellart, and Paul Van Dooren. A measure of similarity between graph vertices: Applications to synonym extraction and web searching. *SIAM review*, 46(4):647–666, 2004.

Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment*, 2008(10):P10008, 2008.

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.

Cristian Bodnar, Fabrizio Frasca, Yuguang Wang, Nina Otter, Guido F Montufar, Pietro Lio, and Michael Bronstein. Weisfeiler and lehman go topological: Message passing simplicial networks. In *International Conference on Machine Learning*, pages 1026–1037. PMLR, 2021.

Cristian Bodnar, Francesco Di Giovanni, Benjamin Chamberlain, Pietro Lio, and Michael Bronstein. Neural sheaf diffusion: A topological perspective on heterophily and oversmoothing in gnns. *Advances in Neural Information Processing Systems*, 35:18527–18541, 2022.

Marián Boguñá and Romualdo Pastor-Satorras. Class of correlated random networks with hidden variables. *Phys. Rev. E*, 68:036112, Sep 2003. DOI: [10.1103/PhysRevE.68.036112](https://doi.org/10.1103/PhysRevE.68.036112).

Marián Boguñá, Romualdo Pastor-Satorras, and Alessandro Vespignani. Absence of epidemic threshold in scale-free networks with degree correlations. *Phys. Rev. Lett.*, 90:028701, Jan 2003. DOI: [10.1103/PhysRevLett.90.028701](https://doi.org/10.1103/PhysRevLett.90.028701).

Marián Boguñá, Romualdo Pastor-Satorras, Albert Díaz-Guilera, and Alex Arenas. Models of social networks based on social distance attachment. *Physical review E*, 70(5):056122, 2004.

Marián Boguná, Romualdo Pastor-Satorras, and Alessandro Vespignani. Cut-offs and finite size effects in scale-free networks. *The European Physical Journal B*, 38(2):205–209, 2004.

Ludvig Bohlin, Daniel Edler, Andrea Lancichinetti, and Martin Rosvall. Community detection and visualization of networks with the map equation framework. In *Measuring Scholarly Impact*, pages 3–34. Springer, 2014.

Aleksandar Bojchevski, Oleksandr Shchur, Daniel Zügner, and Stephan Günnemann. Netgan: Generating graphs via random walks. In *International Conference on Machine Learning*, pages 609–618, 2018.

Paolo Boldi and Sebastiano Vigna. The webgraph framework i: compression techniques. In *Proceedings of the 13th international conference on World Wide Web*, pages 595–602, 2004.

Paolo Boldi and Sebastiano Vigna. Axioms for centrality. *Internet Mathematics*, 10(3-4):222–262, 2014.

Johan Bollen and Bruno Gonçalves. Network happiness: How online social interactions relate to our well being. In *Complex Spreading Phenomena in Social Systems*, pages 257–268. Springer, 2018.

Johan Bollen, Bruno Gonçalves, Guangchen Ruan, and Huina Mao. Happiness is assortative in online social networks. *Artificial life*, 17(3):237–251, 2011.

Johan Bollen, Bruno Gonçalves, Ingrid van de Leemput, and Guangchen Ruan. The happiness paradox: your friends are happier than you. *EPJ Data Science*, 6(1):4, 2017.

Béla Bollobás. Random graphs. In *Modern graph theory*, pages 215–252. Springer, 1998.

Béla Bollobás and Oliver Riordan. Robustness and vulnerability of scale-free random graphs. *Internet Mathematics*, 1(1):1–35, 2004.

Béla Bollobás, Oliver Riordan, Joel Spencer, and Gábor Tusnády. The degree sequence of a scale-free random graph process. *Random Structures & Algorithms*, 18(3):279–290, 2001.

Phillip Bonacich and Paulette Lloyd. Eigenvector-like measures of centrality for asymmetric relations. *Social networks*, 23(3):191–201, 2001.

Francesco Bonchi, Francesco Gullo, Andreas Kaltenbrunner, and Yana Volkovich. Core decomposition of uncertain graphs. In *SIGKDD*, pages 1316–1325, 2014.

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

Douglas G Bonett and Thomas A Wright. Sample size requirements for estimating pearson, kendall and spearman correlations. *Psychometrika*, 65(1):23–28, 2000.

Carlo Bonferroni. Teoria statistica delle classi e calcolo delle probabilita. *Pubblicazioni del R Istituto Superiore di Scienze Economiche e Commerciali di Firenze*, 8:3–62, 1936.

Paolo Bonifazi, Miri Goldin, Michel A Picardo, Isabel Jorquera, A Cattani, Gregory Bianconi, Alfonso Represa, Yechezkel Ben-Ari, and Rosa Cossart. Gabaergic hub neurons orchestrate synchrony in developing hippocampal networks. *Science*, 326(5958):1419–1424, 2009.

Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In *Advances in neural information processing systems*, pages 2787–2795, 2013.

Stephen P Borgatti. Structural holes: Unpacking burt’s redundancy measures. *Connections*, 20(1):35–38, 1997.

Stephen P Borgatti and Martin G Everett. Models of core/periphery structures. *Social Networks*, 21(4):375 – 395, 2000. ISSN 0378-8733. doi: [https://doi.org/10.1016/S0378-8733\(99\)00019-2](https://doi.org/10.1016/S0378-8733(99)00019-2).

Stephen P Borgatti and Martin G Everett. A graph-theoretic perspective on centrality. *Social networks*, 28(4):466–484, 2006.

Stephen P Borgatti, Candace Jones, and Martin G Everett. Network measures of social capital. *Connections*, 21(2):27–36, 1998.

Stephen P Borgatti, Martin G Everett, and Linton C Freeman. Ucinet for windows: Software for social network analysis. 2002.

Stephen P Borgatti, Ajay Mehra, Daniel J Brass, and Giuseppe Labianca. Network analysis in the social sciences. *science*, 323(5916):892–895, 2009.

Christian Borgelt and Michael R Berthold. Mining molecular fragments: Finding relevant substructures of molecules. In *2002 IEEE International Conference on Data Mining, 2002. Proceedings.*, pages 51–58. IEEE, 2002.

Christian Borgs, Jennifer Chayes, László Lovász, Vera T Sós, Balázs Szegedy, and Katalin Vesztergombi. Graph limits and parameter

testing. In *Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*, pages 261–270, 2006.

Otakar Borůvka. O jistém problému minimálním. 1926.

Edward Bortnikov, Maxim Gurevich, Idit Keidar, Gabriel Kliot, and Alexander Shraer. Brahms: Byzantine resilient random membership sampling. *Computer Networks*, 53(13):2340–2359, 2009.

Michael Bostock and Jeffrey Heer. Protovis: A graphical toolkit for visualization. *IEEE transactions on visualization and computer graphics*, 15(6):1121–1128, 2009.

Michael Bostock, Vadim Ogievetsky, and Jeffrey Heer. D³ data-driven documents. *IEEE transactions on visualization and computer graphics*, 17(12):2301–2309, 2011.

Giulio Bottazzi and Davide Pirino. Measuring industry relatedness and corporate coherence. 2010.

Oualid Boutemine and Mohamed Bouguessa. Mining community structures in multidimensional networks. *TKDD*, 11(4):51, 2017.

Levi Boxell, Matthew Gentzkow, and Jesse M Shapiro. Greater internet use is not associated with faster growth in political polarization among us demographic groups. *Proceedings of the National Academy of Sciences*, 114(40):10612–10617, 2017.

Ulrik Brandes and Daniel Fleischer. Centrality measures based on current flow. In *Annual symposium on theoretical aspects of computer science*, pages 533–544. Springer, 2005.

Ulrik Brandes, Markus Eiglsperger, Jürgen Lerner, and Christian Pich. *Graph markup language (GraphML)*. 2013.

Ronald L Breiger and Philippa E Pattison. Cumulated social roles: The duality of persons and their algebras. *Social networks*, 8(3):215–256, 1986.

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

Cynthia A Brewer. Color use guidelines for mapping. *Visualization in modern cartography*, 1994:123–148, 1994.

John S Bridle. Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. In *Neurocomputing: Algorithms, architectures and applications*, pages 227–236. Springer, 1990.

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

Björn Bringmann and Siegfried Nijssen. What is frequent in a single graph? In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 858–863. Springer, 2008.

Björn Bringmann, Michele Berlingerio, Francesco Bonchi, and Arisitdes Gionis. Learning and predicting the evolution of social networks. *IEEE Intelligent Systems*, 25(4):26–35, 2010.

Dirk Brockmann, Lars Hufnagel, and Theo Geisel. The scaling laws of human travel. *Nature*, 439(7075):462–465, 2006.

Shaked Brody, Uri Alon, and Eran Yahav. How attentive are graph attention networks? *arXiv preprint arXiv:2105.14491*, 2021.

Tom Broekel and Matté Hartog. Explaining the structure of inter-organizational networks using exponential random graph models. *Industry and Innovation*, 20(3):277–295, 2013.

Tom Broekel, Pierre-Alexandre Balland, Martijn Burger, and Frank van Oort. Modeling knowledge networks in economic geography: a discussion of four methods. *The annals of regional science*, 53(2):423–452, 2014.

Anna D Broido and Aaron Clauset. Scale-free networks are rare. *Nature communications*, 10(1):1017, 2019.

Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017.

Michael M Bronstein, Joan Bruna, Taco Cohen, and Petar Veličković. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. *arXiv preprint arXiv:2104.13478*, 2021.

Tom Brughmans. Connecting the dots: towards archaeological network analysis. *Oxford Journal of Archaeology*, 29(3):277–303, 2010.

Tom Brughmans. Thinking through networks: a review of formal network methods in archaeology. *Journal of Archaeological Method and Theory*, 20(4):623–662, 2013.

Charles D Brummitt, Raissa M D’Souza, and Elizabeth A Leicht. Suppressing cascades of load in interdependent networks. *Proceedings of the National Academy of Sciences*, 109(12):E680–E689, 2012.

- Joan Bruna and X Li. Community detection with graph neural networks. *stat*, 1050:27, 2017.
- Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann Lecun. Spectral networks and locally connected networks on graphs. In *ICLR*, 2014.
- Sergey V Buldyrev, Roni Parshani, Gerald Paul, H Eugene Stanley, and Shlomo Havlin. Catastrophic cascade of failures in interdependent networks. *Nature*, 464(7291):1025, 2010.
- Ed Bullmore and Olaf Sporns. Complex brain networks: graph theoretical analysis of structural and functional systems. *Nature reviews neuroscience*, 10(3):186–198, 2009.
- Horst Bunke. On a relation between graph edit distance and maximum common subgraph. *Pattern Recognition Letters*, 18(8):689–694, 1997.
- Rebekka Burkholz, Antonios Garas, and Frank Schweitzer. How damage diversification can reduce systemic risk. *Physical Review E*, 93(4):042313, 2016a.
- Rebekka Burkholz, Matt V Leduc, Antonios Garas, and Frank Schweitzer. Systemic risk in multiplex networks with asymmetric coupling and threshold feedback. *Physica D: Nonlinear Phenomena*, 323:64–72, 2016b.
- Ronald S Burt. Structural holes: The social structure of competition. *University of Illinois at Urbana-Champaign's Academy for Entrepreneurial Leadership Historical Research Reference in Entrepreneurship*, 1992.
- Roberto Busa. Index thomisticus sancti thomae aquinatis operum omnium indices et concordantiae in quibus verborum omnium et singulorum formae et lemmata cum suis frequentiis et contextibus variis modis referuntur. 1974.
- Vannevar Bush et al. As we may think. *The atlantic monthly*, 176(1):101–108, 1945.
- Sebastián Bustos, Charles Gomez, Ricardo Hausmann, and César A Hidalgo. The dynamics of nestedness predicts the evolution of industrial ecosystems. *PloS one*, 7(11):e49393, 2012.
- Carter T Butts. Network inference, error, and informant (in) accuracy: a bayesian approach. *social networks*, 25(2):103–140, 2003.
- Jin-Yi Cai, Martin Fürer, and Neil Immerman. An optimal lower bound on the number of variables for graph identification. *Combinatorica*, 12(4):389–410, 1992.

Qing Cai, Lijia Ma, Maoguo Gong, and Dayong Tian. A survey on network community detection based on evolutionary computation. *IJBIC*, 8(2):84–98, 2016.

Alberto Caimo and Isabella Gollini. A multilayer exponential random graph modelling approach for weighted networks. *Computational Statistics & Data Analysis*, 142:106825, 2020.

Alberto Cairo. *The Functional Art: An introduction to information graphics and visualization*. New Riders, 2012.

Guido Caldarelli. *Scale-free networks: complex webs in nature and technology*. Oxford University Press, 2007.

Guido Caldarelli and Michele Catanzaro. *Networks: A very short introduction*, volume 335. Oxford University Press, 2012.

Duncan S Callaway, Mark EJ Newman, Steven H Strogatz, and Duncan J Watts. Network robustness and fragility: Percolation on random graphs. *Physical review letters*, 85(25):5468, 2000.

Stijn Cambie and Matteo Mazzamurro. Resolution of yan’s conjecture on entropy of graphs. *arXiv preprint arXiv:2205.03357*, 2022.

Stijn Cambie, Yanni Dong, and Matteo Mazzamurro. Extremal values of degree-based entropies of bipartite graphs. *Information Sciences*, 676:120737, 2024.

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.

Julián Candia, Marta C González, Pu Wang, Timothy Schoenharl, Greg Madey, and Albert-László Barabási. Uncovering individual and collective human dynamics from mobile phone records. *Journal of physics A: mathematical and theoretical*, 41(22):224015, 2008.

Carlo Vittorio Cannistraci, Gregorio Alanis-Lobato, and Timothy Ravasi. From link-prediction in brain connectomes and protein interactomes to the local-community-paradigm in complex networks. *Scientific reports*, 3:1613, 2013.

Shaosheng Cao, Wei Lu, and Qiongkai Xu. Grarep: Learning graph representations with global structural information. In *Proceedings of the 24th ACM international on conference on information and knowledge management*, pages 891–900, 2015.

Shaosheng Cao, Wei Lu, and Qiongkai Xu. Deep neural networks for learning graph representations. In *AAAI*, 2016.

Amedeo Cappelli, Michele Coscia, Fosca Giannotti, Dino Pedreschi, and Salvo Rinzivillo. The social network of dante's inferno. *Leonardo*, 44(3):246–247, 2011.

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.

Vincenzo Carletti, Pasquale Foggia, Alessia Saggesse, and Mario Vento. Challenging the time complexity of exact subgraph isomorphism for huge and dense graphs with vf3. *IEEE transactions on pattern analysis and machine intelligence*, 40(4):804–818, 2017a.

Vincenzo Carletti, Pasquale Foggia, Alessia Saggesse, and Mario Vento. Introducing vf3: A new algorithm for subgraph isomorphism. In *Graph-Based Representations in Pattern Recognition: 11th IAPR-TC-15 International Workshop, GbRPR 2017, Anacapri, Italy, May 16–18, 2017, Proceedings* 11, pages 128–139. Springer, 2017b.

Shai Carmi, Shlomo Havlin, Scott Kirkpatrick, Yuval Shavitt, and Eran Shir. A model of internet topology using k-shell decomposition. *Proceedings of the National Academy of Sciences*, 104(27):11150–11154, 2007.

J Douglas Carroll and Jih-Jie Chang. Analysis of individual differences in multidimensional scaling via an n-way generalization of “eckart-young” decomposition. *Psychometrika*, 35(3):283–319, 1970.

Claudio Castellano and Romualdo Pastor-Satorras. Thresholds for epidemic spreading in networks. *Physical review letters*, 105(21):218701, 2010.

Claudio Castellano, Santo Fortunato, and Vittorio Loreto. Statistical physics of social dynamics. *Reviews of modern physics*, 81(2):591, 2009.

Ciro Cattuto, Wouter Van den Broeck, Alain Barrat, Vittoria Colizza, Jean-François Pinton, and Alessandro Vespignani. Dynamics of person-to-person interactions from distributed rfid sensor networks. *PloS one*, 5(7), 2010.

Meeyoung Cha, Haewoon Kwak, Pablo Rodriguez, Yong-Yeol Ahn, and Sue Moon. I tube, you tube, everybody tubes: analyzing the world’s largest user generated content video system. In *SIGCOMM*, pages 1–14, 2007.

Meeyoung Cha, Alan Mislove, Ben Adams, and Krishna P Gummadi. Characterizing social cascades in flickr. In *Proceedings of the first workshop on Online social networks*, pages 13–18, 2008.

Meeyoung Cha, Haewoon Kwak, Pablo Rodriguez, Yong-Yeol Ahn, and Sue Moon. Analyzing the video popularity characteristics of large-scale user generated content systems. *IEEE/ACM Transactions on networking*, 17(5):1357–1370, 2009a.

Meeyoung Cha, Alan Mislove, and Krishna P Gummadi. A measurement-driven analysis of information propagation in the flickr social network. In *WWW*, pages 721–730, 2009b.

Meeyoung Cha, Hamed Haddadi, Fabricio Benevenuto, and Krishna P Gummadi. Measuring user influence in twitter: The million follower fallacy. In *ICWSM*, 2010.

Deepayan Chakrabarti, Ravi Kumar, and Andrew Tomkins. Evolutionary clustering. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 554–560. ACM, 2006.

Ben Chamberlain, James Rowbottom, Maria I Gorinova, Michael Bronstein, Stefan Webb, and Emanuele Rossi. Grand: Graph neural diffusion. In *International Conference on Machine Learning*, pages 1407–1418. PMLR, 2021a.

Benjamin Chamberlain, James Rowbottom, Davide Eynard, Francesco Di Giovanni, Xiaowen Dong, and Michael Bronstein. Beltrami flow and neural diffusion on graphs. *Advances in Neural Information Processing Systems*, 34:1594–1609, 2021b.

Arun G Chandrasekhar and Matthew O Jackson. Tractable and consistent random graph models. Technical report, National Bureau of Economic Research, 2014.

Shiyu Chang, Wei Han, Jiliang Tang, Guo-Jun Qi, Charu C Aggarwal, and Thomas S Huang. Heterogeneous network embedding via deep architectures. In *SIGKDD*, pages 119–128, 2015.

O. Chapelle, B. Scholkopf, and A. Zien. *Semi-Supervised Learning*. Adaptive Computation and Machine Learning series. MIT Press, 2010. ISBN 9780262514125. URL <https://books.google.dk/books?id=TtWMEAAQBAJ>.

Gary Chartrand, Grzegorz Kubicki, and Michelle Schultz. Graph similarity and distance in graphs. *Aequationes Mathematicae*, 55(1-2):129–145, 1998.

Surajit Chaudhuri and Umeshwar Dayal. An overview of data warehousing and olap technology. *ACM Sigmod record*, 26(1):65–74, 1997.

Bernard Chazelle. A minimum spanning tree algorithm with inverse-ackermann type complexity. *Journal of the ACM (JACM)*, 47(6):1028–1047, 2000.

Chen Chen, Xifeng Yan, Feida Zhu, Jiawei Han, and S Yu Philip. Graph olap: Towards online analytical processing on graphs. In *ICDM*, pages 103–112. IEEE, 2008.

Chen Chen, Hanghang Tong, B Aditya Prakash, Charalampos E Tsourakakis, Tina Eliassi-Rad, Christos Faloutsos, and Duen Horng Chau. Node immunization on large graphs: Theory and algorithms. *IEEE Transactions on Knowledge and Data Engineering*, 28(1):113–126, 2015.

Haochen Chen, Bryan Perozzi, Yifan Hu, and Steven Skiena. Harp: Hierarchical representation learning for networks. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Zhengzhang Chen, Kevin A Wilson, Ye Jin, William Hendrix, and Nagiza F Samatova. Detecting and tracking community dynamics in evolutionary networks. In *ICDMW*, pages 318–327. IEEE, 2010.

Justin Cheng, Lada Adamic, P Alex Dow, Jon Michael Kleinberg, and Jure Leskovec. Can cascades be predicted? In *WWW*, pages 925–936. ACM, 2014.

Justin Cheng, Lada A Adamic, Jon M Kleinberg, and Jure Leskovec. Do cascades recur? In *WWW*, pages 671–681. ACM, 2016.

Michael Chertok and Yosi Keller. Efficient high order matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(12):2205–2215, 2010.

Kai-Yang Chiang, Nagarajan Natarajan, Ambuj Tewari, and Inderjit S Dhillon. Exploiting longer cycles for link prediction in signed networks. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, pages 1157–1162. ACM, 2011.

Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 257–266, 2019.

Eunjoon Cho, Seth A Myers, and Jure Leskovec. Friendship and mobility: user movement in location-based social networks. In *SIGKDD*, pages 1082–1090. ACM, 2011.

- Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *EMNLP*, pages 1724–1734, 2014.
- Sung-Bae Cho and Jin H Kim. Multiple network fusion using fuzzy logic. *IEEE Transactions on Neural Networks*, 6(2):497–501, 1995.
- Nicholas A Christakis and James H Fowler. The spread of obesity in a large social network over 32 years. *New England journal of medicine*, 357(4):370–379, 2007.
- Nicholas A Christakis and James H Fowler. The collective dynamics of smoking in a large social network. *New England journal of medicine*, 358(21):2249–2258, 2008.
- Walter Christaller. *Central places in southern Germany*. Prentice Hall, 1966.
- Fan Chung and Linyuan Lu. The average distances in random graphs with given expected degrees. *Proceedings of the National Academy of Sciences*, 99(25):15879–15882, 2002a.
- Fan Chung and Linyuan Lu. Connected components in random graphs with given expected degree sequences. *Annals of combinatorics*, 6(2):125–145, 2002b.
- Matteo Cinelli, Gianmarco De Francisci Morales, Alessandro Galeazzi, Walter Quattrociocchi, and Michele Starnini. The echo chamber effect on social media. *Proceedings of the National Academy of Sciences*, 118(9):e2023301118, 2021.
- Federico Cinus, Marco Minici, Corrado Monti, and Francesco Bonchi. The effect of people recommenders on echo chambers and polarization. In *Proceedings of the International AAAI Conference on Web and Social Media*, volume 16, pages 90–101, 2022.
- A Clauset, E Tucker, and M Sainz. The colorado index of complex networks, 2016.
- Aaron Clauset. Finding local community structure in networks. *Physical review E*, 72(2):026132, 2005.
- Aaron Clauset and Christopher Moore. Accuracy and scaling phenomena in internet mapping. *Physical Review Letters*, 94(1):018701, 2005.
- Aaron Clauset, Mark EJ Newman, and Christopher Moore. Finding community structure in very large networks. *Physical review E*, 70(6):066111, 2004.

Aaron Clauset, Cristopher Moore, and Mark EJ Newman. Hierarchical structure and the prediction of missing links in networks. *Nature*, 453(7191):98, 2008.

Aaron Clauset, Cosma Rohilla Shalizi, and Mark EJ Newman. Power-law distributions in empirical data. *SIAM review*, 51(4):661–703, 2009.

Aaron Clauset, Samuel Arbesman, and Daniel B Larremore. Systematic inequality and hierarchy in faculty hiring networks. *Science advances*, 1(1):e1400005, 2015.

David Clayton, Michael Hills, and A Pickles. *Statistical models in epidemiology*, volume 161. Oxford university press Oxford, 1993.

William S Cleveland and Robert McGill. Graphical perception: Theory, experimentation, and application to the development of graphical methods. *Journal of the American statistical association*, 79(387):531–554, 1984.

Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

Edith Cohen and Haim Kaplan. Spatially-decaying aggregation over a network. *Journal of Computer and System Sciences*, 73(3):265–288, 2007.

Reuven Cohen, Keren Erez, Daniel Ben-Avraham, and Shlomo Havlin. Resilience of the internet to random breakdowns. *Physical review letters*, 85(21):4626, 2000.

Reuven Cohen, Keren Erez, Daniel Ben-Avraham, and Shlomo Havlin. Breakdown of the internet under intentional attack. *Physical review letters*, 86(16):3682, 2001.

Reuven Cohen, Shlomo Havlin, and Daniel Ben-Avraham. Efficient immunization strategies for computer networks and populations. *Physical review letters*, 91(24):247901, 2003.

Jonathan R Cole. Fair science: Women in the scientific community. 1979.

Jonathan R Cole and Stephen Cole. Social stratification in science. 1974.

Vittoria Colizza, Alain Barrat, Marc Barthélémy, and Alessandro Vespignani. The role of the airline transportation network in the prediction and predictability of global epidemics. *Proceedings of the*

National Academy of Sciences of the United States of America, 103(7):2015–2020, 2006a.

Vittoria Colizza, Alessandro Flammini, M Angeles Serrano, and Alessandro Vespignani. Detecting rich-club ordering in complex networks. *Nature physics*, 2(2):110–115, 2006b.

Vittoria Colizza, Alain Barrat, Marc Barthelemy, Alain-Jacques Valleron, and Alessandro Vespignani. Modeling the worldwide spread of pandemic influenza: baseline case and containment interventions. *PLoS medicine*, 4(1), 2007.

Charles R Collins and Kenneth Stephenson. A circle packing algorithm. *Computational Geometry*, 25(3):233–256, 2003.

Linda M Collins and Clyde W Dent. Omega: A general formulation of the rand index of cluster recovery suitable for non-disjoint solutions. *Multivariate Behavioral Research*, 23(2):231–242, 1988.

Michael Conover, Jacob Ratkiewicz, Matthew Francisco, Bruno Gonçalves, Filippo Menczer, and Alessandro Flammini. Political polarization on twitter. In *Proceedings of the international aaai conference on web and social media*, volume 5, pages 89–96, 2011.

Kathryn Cooper and Mauricio Barahona. Role-based similarity in directed networks. *arXiv preprint arXiv:1012.2726*, 2010.

Gennaro Cordasco and Luisa Gargano. Community detection via semi-synchronous label propagation algorithms. In *2010 IEEE International Workshop on: Business Applications of Social Network Analysis (BASNA)*, pages 1–8. IEEE, 2010.

Luigi P Cordella, Pasquale Foggia, Carlo Sansone, and Mario Vento. A (sub) graph isomorphism algorithm for matching large graphs. *IEEE transactions on pattern analysis and machine intelligence*, 26(10):1367–1372, 2004.

Luigi Pietro Cordella, Pasquale Foggia, Carlo Sansone, and Mario Vento. An improved algorithm for matching large graphs. In *3rd IAPR-TC15 workshop on graph-based representations in pattern recognition*, pages 149–159, 2001.

Michele Coscia. Average is boring: How similarity kills a meme’s success. *Scientific reports*, 4:6477, 2014.

Michele Coscia. Popularity spikes hurt future chances for viral propagation of protomemes. *Communications of the ACM*, 61(1):70–77, 2017.

Michele Coscia. Using arborescences to estimate hierarchicalness in directed complex networks. *PloS one*, 13(1):e0190825, 2018.

Michele Coscia. Discovering communities of community discovery. In *Proceedings of the 2019 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, pages 1–8, 2019.

Michele Coscia. Generalized euclidean measure to estimate network distances. In *Proceedings of the International AAAI Conference on Web and Social Media*, volume 14, pages 119–129, 2020.

Michele Coscia. Pearson correlations on complex networks. *Journal of Complex Networks*, 9(6):cnab036, 2021.

Michele Coscia and Karel Devriendt. Pearson correlations on networks: Corrigendum. *arXiv preprint arXiv:2402.09489*, 2024.

Michele Coscia and Ricardo Hausmann. Evidence that calls-based and mobility networks are isomorphic. *PloS one*, 10(12):e0145091, 2015.

Michele Coscia and Frank MH Neffke. Network backboning with noisy data. In *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*, pages 425–436. IEEE, 2017.

Michele Coscia and Viridiana Rios. Knowing where and how criminal organizations operate using web content. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, pages 1412–1421, 2012.

Michele Coscia and Luca Rossi. Benchmarking api costs of network sampling strategies. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 663–672. IEEE, 2018.

Michele Coscia and Michael Szell. Multiplex graph association rules for link prediction. *arXiv preprint arXiv:2008.08351*, 2020.

Michele Coscia, Fosca Giannotti, and Dino Pedreschi. A classification for community discovery methods in complex networks. *SADM*, 4(5):512–546, 2011.

Michele Coscia, Giulio Rossetti, Fosca Giannotti, and Dino Pedreschi. Demon: a local-first discovery method for overlapping communities. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 615–623. ACM, 2012.

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.

Michele Coscia, Giulio Rossetti, Diego Pennacchioli, Damiano Ceccarelli, and Fosca Giannotti. You know because i know: a multidimensional network approach to human resources problem. In *Proceedings of the 2013 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, pages 434–441. ACM, 2013b.

Michele Coscia, Giulio Rossetti, Fosca Giannotti, and Dino Pedreschi. Uncovering hierarchical and overlapping communities with a local-first approach. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 9(1):6, 2014.

Michele Coscia, Andres Gomez-Lievano, James McNerney, and Frank Neffke. The node vector distance problem in complex networks. *ACM Computing Surveys*, 2020.

Wesley Cota, Silvio C Ferreira, Romualdo Pastor-Satorras, and Michele Starnini. Quantifying echo chamber effects in information spreading over political communication networks. *EPJ Data Science*, 8(1):35, 2019.

Owen T Courtney and Ginestra Bianconi. Generalized network structures: The configuration model and the canonical ensemble of simplicial complexes. *Physical Review E*, 93(6):062311, 2016.

Owen T Courtney and Ginestra Bianconi. Weighted growing simplicial complexes. *Physical Review E*, 95(6):062301, 2017.

Thomas M Cover. *Elements of information theory*. John Wiley & Sons, 1999.

Skyler J Cranmer and Bruce A Desmarais. Inferential network analysis with exponential random graph models. *Political analysis*, 19(1):66–86, 2011.

Antonia Creswell, Tom White, Vincent Dumoulin, Kai Arulkumaran, Biswa Sengupta, and Anil A Bharath. Generative adversarial networks: An overview. *IEEE signal processing magazine*, 35(1):53–65, 2018.

Matthieu Cristelli, Andrea Tacchella, and Luciano Pietronero. The heterogeneous dynamics of economic complexity. *PloS one*, 10(2):e0117174, 2015.

Nick Crossley, Elisa Bellotti, Gemma Edwards, Martin G Everett, Johan Koskinen, and Mark Tranmer. *Social network analysis for ego-nets: Social network analysis for actor-centred networks*. Sage, 2015.

Paolo Crucitti, Vito Latora, Massimo Marchiori, and Andrea Rapisarda. Error and attack tolerance of complex networks. *Physica A: Statistical mechanics and its applications*, 340(1-3):388–394, 2004.

Gabor Csardi and Tamas Nepusz. The igraph software package for complex network research. *InterJournal, Complex Systems*, 1695(5):1–9, 2006.

Peter Csermely, Tamás Korcsmáros, Huba JM Kiss, Gabor London, and Ruth Nussinov. Structure and dynamics of molecular networks: a novel paradigm of drug discovery: a comprehensive review. *Pharmacology & therapeutics*, 138(3):333–408, 2013a.

Peter Csermely, András London, Ling-Yun Wu, and Brian Uzzi. Structure and dynamics of core/periphery networks. *Journal of Complex Networks*, 1(2):93–123, 2013b.

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

Jesper Dall and Michael Christensen. Random geometric graphs. *Physical review E*, 66(1):016121, 2002.

Simone Daminelli, Josephine Maria Thomas, Claudio Durán, and Carlo Vittorio Cannistraci. Common neighbours and the local-community-paradigm for topological link prediction in bipartite networks. *New Journal of Physics*, 17(11):113037, 2015.

Leon Danon, Albert Diaz-Guilera, Jordi Duch, and Alex Arenas. Comparing community structure identification. *Journal of Statistical Mechanics: Theory and Experiment*, 2005(09):P09008, 2005.

Vinh-Loc Dao, Cécile Bothorel, and Philippe Lenca. Community structure: A comparative evaluation of community detection methods. *arXiv preprint arXiv:1812.06598*, 2018a.

Vinh-Loc Dao, Cécile Bothorel, and Philippe Lenca. Estimating the similarity of community detection methods based on cluster size distribution. In *International Workshop on Complex Networks and their Applications*, pages 183–194. Springer, 2018b.

Anirban Dasgupta, Ravi Kumar, and D Sivakumar. Social sampling. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 235–243. ACM, 2012.

Ron Davidson and David Harel. Drawing graphs nicely using simulated annealing. *TOG*, 15(4):301–331, 1996.

Allison Davis, Burleigh Bradford Gardner, and Mary R Gardner. *Deep South: A social anthropological study of caste and class*. Univ of South Carolina Press, 1941.

Darcy Davis, Ryan Lichtenwalter, and Nitesh V Chawla. Multi-relational link prediction in heterogeneous information networks. In *ASONAM*, pages 281–288. IEEE, 2011.

Henrique Ferraz de Arruda, Felipe Maciel Cardoso, Guilherme Ferraz de Arruda, Alexis R Hernández, Luciano da Fontoura Costa, and Yamir Moreno. Modelling how social network algorithms can influence opinion polarization. *Information Sciences*, 588:265–278, 2022.

Caterina De Bacco, Eleanor A Power, Daniel B Larremore, and Christopher Moore. Community detection, link prediction, and layer interdependence in multilayer networks. *Physical Review E*, 95(4):042317, 2017.

Caterina De Bacco, Daniel B Larremore, and Christopher Moore. A physical model for efficient ranking in networks. *Science advances*, 4(7):eaar8260, 2018.

Nicola De Cao and Thomas Kipf. Molgan: An implicit generative model for small molecular graphs. *arXiv preprint arXiv:1805.11973*, 2018.

Manlio De Domenico. Multilayer modeling and analysis of human brain networks. *Giga Science*, 6(5):gix004, 2017.

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.

Manlio De Domenico, Albert Solé-Ribalta, Sergio Gómez, and Alex Arenas. Navigability of interconnected networks under random failures. *Proceedings of the National Academy of Sciences*, 111(23):8351–8356, 2014.

Manlio De Domenico, Andrea Lancichinetti, Alex Arenas, and Martin Rosvall. Identifying modular flows on multilayer networks reveals highly overlapping organization in interconnected systems. *Physical Review X*, 5(1):011027, 2015a.

Manlio De Domenico, Vincenzo Nicosia, Alexandre Arenas, and Vito Latora. Structural reducibility of multilayer networks. *Nature communications*, 6:6864, 2015b.

Manlio De Domenico, Mason A Porter, and Alex Arenas. Muxviz: a tool for multilayer analysis and visualization of networks. *Journal of Complex Networks*, 3(2):159–176, 2015c.

Manlio De Domenico, Albert Solé-Ribalta, Elisa Omodei, Sergio Gómez, and Alex Arenas. Ranking in interconnected multilayer networks reveals versatile nodes. *Nature communications*, 6:6868, 2015d.

Lieven De Lathauwer, Bart De Moor, and Joos Vandewalle. A multilinear singular value decomposition. *SIAM journal on Matrix Analysis and Applications*, 21(4):1253–1278, 2000.

Pasquale De Meo, Emilio Ferrara, Giacomo Fiumara, and Alessandro Provetti. Mixing local and global information for community detection in large networks. *Journal of Computer and System Sciences*, 80(1):72–87, 2014.

Yves-Alexandre De Montjoye, César A Hidalgo, Michel Verleysen, and Vincent D Blondel. Unique in the crowd: The privacy bounds of human mobility. *Scientific reports*, 3:1376, 2013.

Wouter De Nooy. A literary playground: Literary criticism and balance theory. *Poetics*, 26(5-6):385–404, 1999.

Wouter De Nooy, Andrej Mrvar, and Vladimir Batagelj. *Exploratory social network analysis with Pajek*. Cambridge University Press, 2018.

Ithiel de Sola Pool and Manfred Kochen. Contacts and influence. *Social networks*, 1(1):5–51, 1978.

Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *NIPS*, pages 3844–3852, 2016.

Morris H DeGroot. Reaching a consensus. *Journal of the American Statistical Association*, 69(345):118–121, 1974.

Matthias Dehmer and Abbe Mowshowitz. A history of graph entropy measures. *Information Sciences*, 181(1):57–78, 2011.

Michela Del Vicario, Alessandro Bessi, Fabiana Zollo, Fabio Petroni, Antonio Scala, Guido Caldarelli, H Eugene Stanley, and Walter Quattrociocchi. The spreading of misinformation online. *PNAS*, 113(3):554–559, 2016a.

Michela Del Vicario, Gianna Vivaldo, Alessandro Bessi, Fabiana Zollo, Antonio Scala, Guido Caldarelli, and Walter Quattrociocchi. Echo chambers: Emotional contagion and group polarization on facebook. *Scientific reports*, 6:37825, 2016b.

Michela Del Vicario, Antonio Scala, Guido Caldarelli, H Eugene Stanley, and Walter Quattrociocchi. Modeling confirmation bias and polarization. *Scientific reports*, 7:40391, 2017.

Fabio Della Rossa, Fabio Dercole, and Carlo Piccardi. Profiling core-periphery network structure by random walkers. *Scientific reports*, 3:1467, 2013.

J-C Delvenne, Sophia N Yaliraki, and Mauricio Barahona. Stability of graph communities across time scales. *Proceedings of the national academy of sciences*, 107(29):12755–12760, 2010.

Jean-Charles Delvenne, Michael T Schaub, Sophia N Yaliraki, and Mauricio Barahona. The stability of a graph partition: A dynamics-based framework for community detection. In *Dynamics On and Of Complex Networks, Volume 2*, pages 221–242. Springer, 2013.

Arthur Dempster. Upper and lower probabilities induced by a multi-valued mapping. *Annals of Mathematical Statistics*, 38, 1967.

Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1):1–22, 1977.

Chenhui Deng, Zhiqiang Zhao, Yongyu Wang, Zhiru Zhang, and Zhuo Feng. Graphzoom: A multi-level spectral approach for accurate and scalable graph embedding. *arXiv preprint arXiv:1910.02370*, 2019.

Li Deng, Dong Yu, et al. Deep learning: methods and applications. *Foundations and Trends® in Signal Processing*, 7(3–4):197–387, 2014.

Xiaoheng Deng, Genghao Li, and Mianxiong Dong. Finding overlapping communities with random walks on line graph and attraction intensity. In *International Conference on Wireless Algorithms, Systems, and Applications*, pages 94–103. Springer, 2015.

Imre Derényi, Gergely Palla, and Tamás Vicsek. Clique percolation in random networks. *Physical review letters*, 94(16):160202, 2005.

Karine Descormiers and Carlo Morselli. Alliances, conflicts, and contradictions in montreal’s street gang landscape. *International Criminal Justice Review*, 21(3):297–314, 2011.

Bruce A Desmarais and Skyler J Cranmer. Statistical inference for valued-edge networks: The generalized exponential random graph model. *PloS one*, 7(1):e30136, 2012.

T Dettmers, P Minervini, P Stenetorp, and S Riedel. Convolutional 2d knowledge graph embeddings. In *32nd AAAI Conference on Artificial Intelligence, AAAI 2018*, volume 32, pages 1811–1818. AAI Publications, 2018.

Pierre Deville, Dashun Wang, Roberta Sinatra, Chaoming Song, Vincent D Blondel, and Albert-László Barabási. Career on the move: Geography, stratification, and scientific impact. *Scientific reports*, 4:4770, 2014.

Pierre Deville, Chaoming Song, Nathan Eagle, Vincent D Blondel, Albert-László Barabási, and Dashun Wang. Scaling identity connects human mobility and social interactions. *PNAS*, 113(26):7047–7052, 2016.

Karel Devriendt. Effective resistance is more than distance: Laplacians, simplices and the schur complement. *Linear Algebra and its Applications*, 639:24–49, 2022.

Karel Devriendt, Samuel Martin-Gutierrez, and Renaud Lambiotte. Variance and covariance of distributions on graphs. *SIAM Review*, 64(2):343–359, 2022.

Michel Marie Deza and Elena Deza. Encyclopedia of distances. In *Encyclopedia of distances*, pages 1–583. Springer, 2009.

Salma Ben Dhaou, Mouloud Kharoune, Arnaud Martin, and Boutheina Ben Yaghlane. A belief approach for detecting spammed links in social networks. In *International Conference on Agents and Artificial Intelligence*, 2019.

Inderjit S Dhillon. Co-clustering documents and words using bipartite spectral graph partitioning. In *SIGKDD*, pages 269–274, 2001.

Inderjit S Dhillon, Subramanyam Mallela, and Dharmendra S Modha. Information-theoretic co-clustering. In *SIGKDD*, pages 89–98, 2003.

Robert B Dial. Algorithm 360: Shortest-path forest with topological ordering [h]. *Communications of the ACM*, 12(11):632–633, 1969.

Navid Dianati. Unwinding the hairball graph: pruning algorithms for weighted complex networks. *Physical Review E*, 93(1):012304, 2016.

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

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

Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

Andrew Dobson, Kiril Solovey, Rahul Shome, Dan Halperin, and Kostas E Bekris. Scalable asymptotically-optimal multi-robot motion planning. In *2017 International Symposium on Multi-Robot and Multi-Agent Systems (MRS)*, pages 120–127. IEEE, 2017.

Ian Dobson, Benjamin A Carreras, and David E Newman. A loading-dependent model of probabilistic cascading failure. *Probability in the Engineering and Informational Sciences*, 19(1):15–32, 2005.

Ian Dobson, Benjamin A Carreras, Vickie E Lynch, and David E Newman. Complex systems analysis of series of blackouts: Cascading failure, critical points, and self-organization. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 17(2):026103, 2007.

Ugur Dogrusoz, Erhan Giral, Ahmet Cetintas, Ali Civril, and Emek Demir. A layout algorithm for undirected compound graphs. *Information Sciences*, 179(7):980–994, 2009.

Pedro Domingos and Matt Richardson. Mining the network value of customers. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 57–66. ACM, 2001.

Xiaowen Dong, Pascal Frossard, Pierre Vandergheynst, and Nikolai Nefedov. Clustering on multi-layer graphs via subspace analysis on grassmann manifolds. *IEEE Transactions on signal processing*, 62(4):905–918, 2013.

Yuxiao Dong, Jie Tang, Sen Wu, Jilei Tian, Nitesh V Chawla, Jinghai Rao, and Huanhuan Cao. Link prediction and recommendation across heterogeneous social networks. In *2012 IEEE 12th International conference on data mining*, pages 181–190. IEEE, 2012.

Yuxiao Dong, Nitesh V Chawla, and Ananthram Swami. metapath2vec: Scalable representation learning for heterogeneous networks. In *SIGKDD*, pages 135–144, 2017.

Stijn Dongen. A cluster algorithm for graphs. 2000.

Sergey N Dorogovtsev and Jose FF Mendes. Evolution of networks. *Advances in physics*, 51(4):1079–1187, 2002.

Sergey N Dorogovtsev, José Fernando F Mendes, and Alexander N Samukhin. Giant strongly connected component of directed networks. *Physical Review E*, 64(2):025101, 2001.

Constantinos Apostolos Doxiadis et al. Ekistics; an introduction to the science of human settlements. 1968.

- Xianzhi Du, Mostafa El-Khamy, Jungwon Lee, and Larry Davis. Fused dnn: A deep neural network fusion approach to fast and robust pedestrian detection. In *2017 IEEE winter conference on applications of computer vision (WACV)*, pages 953–961. IEEE, 2017.
- Jordi Duch and Alex Arenas. Community detection in complex networks using extremal optimization. *Physical review E*, 72(2):027104, 2005.
- Olivier Duchenne, Francis Bach, In-So Kweon, and Jean Ponce. A tensor-based algorithm for high-order graph matching. *IEEE transactions on pattern analysis and machine intelligence*, 33(12):2383–2395, 2011.
- Daniel M Dunlavy, Tamara G Kolda, and Evrim Acar. Temporal link prediction using matrix and tensor factorizations. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 5(2):10, 2011.
- Olive Jean Dunn. Multiple comparisons among means. *Journal of the American statistical association*, 56(293):52–64, 1961.
- Cody Dunne and Ben Shneiderman. Motif simplification: improving network visualization readability with fan, connector, and clique glyphs. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 3247–3256, 2013.
- Rick Durrett. *Probability: theory and examples*, volume 49. Duxbury Press, 1996.
- Rick Durrett. Some features of the spread of epidemics and information on a random graph. *Proceedings of the National Academy of Sciences*, 2010.
- David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. In *NIPS*, pages 2224–2232, 2015.
- Cynthia Dwork, Ravi Kumar, Moni Naor, and Dandapani Sivakumar. Rank aggregation methods for the web. In *Proceedings of the 10th international conference on World Wide Web*, pages 613–622. ACM, 2001.
- Tim Dwyer, Kim Marriott, and Michael Wybrow. Integrating edge routing into force-directed layout. In *International Symposium on Graph Drawing*, pages 8–19. Springer, 2006.
- Raissa M D’Souza and Michael Mitzenmacher. Local cluster aggregation models of explosive percolation. *Physical review letters*, 104(19):195702, 2010.

David Easley and Jon Kleinberg. *Networks, crowds, and markets: Reasoning about a highly connected world*. Cambridge University Press, 2010.

Holger Ebel, Lutz-Ingo Mielsch, and Stefan Bornholdt. Scale-free topology of e-mail networks. *Physical review E*, 66(3):035103, 2002.

Sergey Edunov, Carlos Diuk, Ismail Onur Filiz, Smriti Bhagat, and Moira Burke. Three and a half degrees of separation. *Research at Facebook*, 2016.

Anthony William Fairbank Edwards. Likelihood. In *Time Series and Statistics*, pages 126–129. Springer, 1972.

Victor M Eguiluz, Dante R Chialvo, Guillermo A Cecchi, Marwan Baliki, and A Vania Apkarian. Scale-free brain functional networks. *Physical review letters*, 94(1):018102, 2005.

Albert Einstein. Über die von der molekularkinetischen theorie der wärme geforderte bewegung von in ruhenden flüssigkeiten suspendierten teilchen. *Annalen der physik*, 4, 1905.

Matthias Erbar, Martin Rumpf, Bernhard Schmitzer, and Stefan Simon. Computation of optimal transport on discrete metric measure spaces. *arXiv preprint arXiv:1707.06859*, 2017.

P Erdős and A Rényi. On random graphs. *Publicationes Mathematicae Debrecen*, 6:290–297, 1959.

Paul Erdos and Alfréd Rényi. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci*, 5(1):17–60, 1960.

Paul Erdős and Alfréd Rényi. On the strength of connectedness of a random graph. *Acta Mathematica Hungarica*, 12(1-2):261–267, 1961.

Paul Erdos and Alfred Renyi. On random matrices. *Magyar Tud. Akad. Mat. Kutató Int. Közl*, 8(455-461):1964, 1964.

Paul Erdos and Alfréd Rényi. On the existence of a factor of degree one of a connected random graph. *Acta Math. Acad. Sci. Hungar*, 17(3-4):359–368, 1966.

Paul Erdős and Miklós Simonovits. Supersaturated graphs and hypergraphs. *Combinatorica*, 3(2):181–192, 1983.

Alcides Viamontes Esquivel and Martin Rosvall. Compression of flow can reveal overlapping-module organization in networks. *Physical Review X*, 1(2):021025, 2011.

Montacer Essid and Justin Solomon. Quadratically-regularized optimal transport on graphs. *arXiv preprint arXiv:1704.08200*, 2017.

Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231, 1996.

Jean-Baptiste Estoup. *Gammes sténographiques: méthode et exercices pour l'acquisition de la vitesse*. Institut sténographique, 1916.

Ernesto Estrada. *The structure of complex networks: theory and applications*. Oxford University Press, 2012.

Leonhard Euler. Solutio problematis ad geometriam situs pertinentis. *Commentarii academiae scientiarum Petropolitanae*, pages 128–140, 1741.

Lawrence C Evans. Partial differential equations and monge-kantorovich mass transfer. *Current developments in mathematics*, 1997(1):65–126, 1997.

Tim S Evans. Clique graphs and overlapping communities. *Journal of Statistical Mechanics: Theory and Experiment*, 2010(12):P12037, 2010.

TS Evans and Renaud Lambiotte. Line graphs, link partitions, and overlapping communities. *Physical Review E*, 80(1):016105, 2009.

Shimon Even. *Graph algorithms*. Cambridge University Press, 2011.

Martin G Everett and Stephen P Borgatti. The dual-projection approach for two-mode networks. *Social Networks*, 35(2):204–210, 2013.

Paul Expert, Tim S Evans, Vincent D Blondel, and Renaud Lambiotte. Uncovering space-independent communities in spatial networks. *Proceedings of the National Academy of Sciences*, 108(19):7663–7668, 2011.

Giorgio Fagiolo. Clustering in complex directed networks. *Physical Review E*, 76(2):026107, 2007.

Tom Fawcett. An introduction to roc analysis. *Pattern recognition letters*, 27(8):861–874, 2006.

Tomás Feder and Rajeev Motwani. Clique partitions, graph compression and speeding-up algorithms. *Journal of Computer and System Sciences*, 51(2):261–272, 1995.

Scott L Feld. Why your friends have more friends than you do. *American Journal of Sociology*, 96(6):1464–1477, 1991.

Willliam Feller. *An introduction to probability theory and its applications*, volume 2. John Wiley & Sons, 1968.

Emilio Ferrara, Onur Varol, Clayton Davis, Filippo Menczer, and Alessandro Flammini. The rise of social bots. *Communications of the ACM*, 59(7):96–104, 2016.

Richard Feynman. The theory of positrons. *Phys. Rev.*, 76:749–759, 1949.

Mathias Fiedler and Christian Borgelt. Support computation for mining frequent subgraphs in a single graph. In *MLG*, 2007.

Miroslav Fiedler. Laplacian of graphs and algebraic connectivity. *Banach Center Publications*, 25(1):57–70, 1989.

Alexander Fix, Aritanan Gruber, Endre Boros, and Ramin Zabih. A graph cut algorithm for higher-order markov random fields. In *2011 International Conference on Computer Vision*, pages 1020–1027. IEEE, 2011.

Jessica C Flack, Michelle Girvan, Frans BM De Waal, and David C Krakauer. Policing stabilizes construction of social niches in primates. *Nature*, 439(7075):426–429, 2006.

Gary William Flake, Steve Lawrence, C Lee Giles, et al. Efficient identification of web communities. In *KDD*, volume 2000, pages 150–160, 2000.

Philip J Fleming and John J Wallace. How not to lie with statistics: the correct way to summarize benchmark results. *Communications of the ACM*, 29(3):218–221, 1986.

Pablo Fleurquin, José J Ramasco, and Victor M Eguiluz. Systemic delay propagation in the us airport network. *Scientific reports*, 3:1159, 2013.

Robert W Floyd. Algorithm 97: shortest path. *Communications of the ACM*, 5(6):345, 1962.

Klaus-Tycho Foerster, Linus Groner, Torsten Hoefer, Michael Koenig, Sascha Schmid, and Roger Wattenhofer. Multi-agent pathfinding with n agents on graphs with n vertices: Combinatorial classification and tight algorithmic bounds. In *International Conference on Algorithms and Complexity*, pages 247–259. Springer, 2017.

Fedor V Fomin and Dimitrios M Thilikos. An annotated bibliography on guaranteed graph searching. *Theoretical computer science*, 399(3):236–245, 2008.

Santo Fortunato. Community detection in graphs. *Physics reports*, 486(3-5):75–174, 2010.

Santo Fortunato and Marc Barthelemy. Resolution limit in community detection. *Proceedings of the National Academy of Sciences*, 104(1):36–41, 2007.

Santo Fortunato and Darko Hric. Community detection in networks: A user guide. *Physics Reports*, 659:1–44, 2016.

Santo Fortunato, Vito Latora, and Massimo Marchiori. Method to find community structures based on information centrality. *Physical review E*, 70(5):056104, 2004.

Santo Fortunato, Marián Boguñá, Alessandro Flammini, and Filippo Menczer. Approximating pagerank from in-degree. In *International Workshop on Algorithms and Models for the Web-Graph*, pages 59–71. Springer, 2006.

Santo Fortunato, Carl T Bergstrom, Katy Börner, James A Evans, Dirk Helbing, Staša Milojević, Alexander M Petersen, Filippo Radicchi, Roberta Sinatra, Brian Uzzi, et al. Science of science. *Science*, 359(6379):eaao0185, 2018.

Bailey K Fosdick, Daniel B Larremore, Joel Nishimura, and Johan Ugander. Configuring random graph models with fixed degree sequences. *SIAM Review*, 60(2):315–355, 2018.

Jacob G Foster, David V Foster, Peter Grassberger, and Maya Paczuski. Edge direction and the structure of networks. *Proceedings of the National Academy of Sciences*, 107(24):10815–10820, 2010.

Samuel P Fraiberger, Roberta Sinatra, Magnus Resch, Christoph Riedl, and Albert-László Barabási. Quantifying reputation and success in art. *Science*, 362(6416):825–829, 2018.

B Francis. *A course in H 1 control theory. Lectures notes in control and information sciences*, volume 88. Springer Verlag Berlin, 1987.

Ove Frank and David Strauss. Markov graphs. *Journal of the american Statistical association*, 81(395):832–842, 1986.

Fabrizio Frasca, Emanuele Rossi, Davide Eynard, Ben Chamberlain, Michael Bronstein, and Federico Monti. Sign: Scalable inception graph neural networks. *arXiv preprint arXiv:2004.11198*, 2020.

Linton C Freeman. Centrality in social networks conceptual clarification. *Social networks*, 1(3):215–239, 1978.

Linton C Freeman, Douglas Roeder, and Robert R Mulholland. Centrality in social networks: II. experimental results. *Social networks*, 2(2):119–141, 1979.

Arne Frick, Andreas Ludwig, and Heiko Mehldau. A fast adaptive layout algorithm for undirected graphs (extended abstract and system demonstration). In *International Symposium on Graph Drawing*, pages 388–403. Springer, 1994.

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

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

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

Karl Friston. The free-energy principle: a unified brain theory? *Nature reviews neuroscience*, 11(2):127–138, 2010.

Thomas MJ Fruchterman and Edward M Reingold. Graph drawing by force-directed placement. *Software: Practice and experience*, 21(11):1129–1164, 1991.

Wenjie Fu, Le Song, and Eric P Xing. Dynamic mixed membership blockmodel for evolving networks. In *Proceedings of the 26th annual international conference on machine learning*, pages 329–336. ACM, 2009.

Kunihiro Fukushima. Visual feature extraction by a multilayered network of analog threshold elements. *IEEE Transactions on Systems Science and Cybernetics*, 5(4):322–333, 1969.

Daniel Funke, Sebastian Lamm, Ulrich Meyer, Manuel Penschuck, Peter Sanders, Christian Schulz, Darren Strash, and Moritz von Looz. Communication-free massively distributed graph generation. *Journal of Parallel and Distributed Computing*, 131:200–217, 2019.

Takayasu Fushimi, Kazumi Saito, Tetsuo Ikeda, and Kazuhiro Kazama. A new group centrality measure for maximizing the connectedness of network under uncertain connectivity. In *Complex Networks and Their Applications VII: Volume 1 Proceedings The 7th International Conference on Complex Networks and Their Applications COMPLEX NETWORKS 2018* 7, pages 3–14. Springer, 2019.

E Gabasova. The star wars social network. *Evelina Gabasova's Blog*. Data available at: <https://github.com/evelinag/StarWars-social-network/tree/master/networks>, 2015.

David J Galas, Gregory Dewey, James Kunert-Graf, and Nikita A Sakhanenko. Expansion of the kullback-leibler divergence, and a new class of information metrics. *Axioms*, 6(2):8, 2017.

Ryan J Gallagher, Jean-Gabriel Young, and Brooke Foucault Welles. A clarified typology of core-periphery structure in networks. *arXiv preprint arXiv:2005.10191*, 2020.

Francis Galton. Typical laws of heredity. Royal Institution of Great Britain, 1877.

Ayalvadi Ganesh, Laurent Massoulié, and Don Towsley. The effect of network topology on the spread of epidemics. In *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, volume 2, pages 1455–1466. IEEE, 2005.

Emden R Gansner and Yifan Hu. Efficient node overlap removal using a proximity stress model. In *International Symposium on Graph Drawing*, pages 206–217. Springer, 2008.

Emden R Gansner and Yehuda Koren. Improved circular layouts. In *International Symposium on Graph Drawing*, pages 386–398. Springer, 2006.

Emden R Gansner, Yifan Hu, Stephen North, and Carlos Scheidegger. Multilevel agglomerative edge bundling for visualizing large graphs. In *2011 IEEE Pacific Visualization Symposium*, pages 187–194. IEEE, 2011.

Hongyang Gao and Shuiwang Ji. Graph u-nets. In *international conference on machine learning*, pages 2083–2092. PMLR, 2019.

Hongyang Gao, Yi Liu, and Shuiwang Ji. Topology-aware graph pooling networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(12):4512–4518, 2021.

Jianxi Gao, Sergey V Buldyrev, Shlomo Havlin, and H Eugene Stanley. Robustness of a network of networks. *Phys. Rev. Lett.*, 107:195701, Nov 2011. doi: 10.1103/PhysRevLett.107.195701.

Jianxi Gao, Sergey V Buldyrev, H Eugene Stanley, and Shlomo Havlin. Networks formed from interdependent networks. *Nature physics*, 8(1):40, 2012.

Jianxi Gao, Yang-Yu Liu, Raissa M D’souza, and Albert-László Barabási. Target control of complex networks. *Nature communications*, 5(1):1–8, 2014.

Jianxi Gao, Baruch Barzel, and Albert-László Barabási. Universal resilience patterns in complex networks. *Nature*, 530(7590):307–312, 2016a.

Xiaofeng Gao, Zhiyin Chen, Fan Wu, and Guihai Chen. Energy efficient algorithms for k -sink minimum movement target coverage problem in mobile sensor network. *IEEE/ACM Transactions on Networking*, 25(6):3616–3627, 2017.

Xinbo Gao, Bing Xiao, Dacheng Tao, and Xuelong Li. A survey of graph edit distance. *Pattern Analysis and applications*, 13(1):113–129, 2010.

Yuan Gao, Lixing Yang, and Shukai Li. Uncertain models on railway transportation planning problem. *Applied Mathematical Modelling*, 40(7-8):4921–4934, 2016b.

David Garcia. Leaking privacy and shadow profiles in online social networks. *Science advances*, 3(8):e1701172, 2017.

Kiran Garimella, Gianmarco De Francisci Morales, Aristides Gionis, and Michael Mathioudakis. Quantifying controversy on social media. *ACM Transactions on Social Computing*, 1(1):1–27, 2018.

Thomas Gärtner, Peter Flach, and Stefan Wrobel. On graph kernels: Hardness results and efficient alternatives. In *Learning theory and kernel machines*, pages 129–143. Springer, 2003.

Johannes Gasteiger, Aleksandar Bojchevski, and Stephan Günnemann. Predict then propagate: Graph neural networks meet personalized pagerank. *arXiv preprint arXiv:1810.05997*, 2018.

Alexander J Gates and Yong-Yeol Ahn. The impact of random models on clustering similarity. *The Journal of Machine Learning Research*, 18(1):3049–3076, 2017.

Alexander J Gates, Ian B Wood, William P Hetrick, and Yong-Yeol Ahn. Element-centric clustering comparison unifies overlaps and hierarchy. *Scientific reports*, 9(1):8574, 2019.

Laetitia Gauvin, André Panisson, and Ciro Cattuto. Detecting the community structure and activity patterns of temporal networks: a non-negative tensor factorization approach. *PloS one*, 9(1):e86028, 2014.

Valerio Gemmetto, Alessio Cardillo, and Diego Garlaschelli. Irreducible network backbones: unbiased graph filtering via maximum entropy. *arXiv preprint arXiv:1706.00230*, 2017.

Neil A Gershenfeld. *The nature of mathematical modeling*. Cambridge university press, 1999.

Lise Getoor and Christopher P Diehl. Link mining: a survey. *Acm Sigkdd Explorations Newsletter*, 7(2):3–12, 2005.

Lise Getoor and Ashwin Machanavajjhala. Entity resolution: theory, practice & open challenges. *Proceedings of the VLDB Endowment*, 5(12):2018–2019, 2012.

Amir Ghasemian, Pan Zhang, Aaron Clauset, Christopher Moore, and Leto Peel. Detectability thresholds and optimal algorithms for community structure in dynamic networks. *Physical Review X*, 6(3):031005, 2016.

Amir Ghasemian, Homa HosseiniMardi, and Aaron Clauset. Evaluating overfit and underfit in models of network community structure. *TKDE*, 2019.

Amir Ghasemian, Homa HosseiniMardi, Aram Galstyan, Edoardo M Airola, and Aaron Clauset. Stacking models for nearly optimal link prediction in complex networks. *Proceedings of the National Academy of Sciences*, 117(38):23393–23400, 2020.

Mina Ghashami, Edo Liberty, and Jeff M Phillips. Efficient frequent directions algorithm for sparse matrices. In *SIGKDD*, pages 845–854, 2016.

Reza Ghorbanchian, Juan G Restrepo, Joaquín J Torres, and Ginestra Bianconi. Higher-order simplicial synchronization of coupled topological signals. *Communications Physics*, 4(1):120, 2021.

Gourab Ghoshal and Albert-László Barabási. Ranking stability and super-stable nodes in complex networks. *Nature communications*, 2:394, 2011.

Christos Giatsidis, Dimitrios M Thilikos, and Michalis Vazirgiannis. D-cores: Measuring collaboration of directed graphs based on degeneracy. In *Data Mining (ICDM), 2011 IEEE 11th International Conference on*, pages 201–210. IEEE, 2011.

Edgar N Gilbert. Random graphs. *The Annals of Mathematical Statistics*, 30(4):1141–1144, 1959.

Walter R Gilks, Sylvia Richardson, and David Spiegelhalter. *Markov chain Monte Carlo in practice*. Chapman and Hall/CRC, 1995.

Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *ICML*, pages 1263–1272. JMLR.org, 2017.

- A Gimenez-Salinas Framis. Illegal networks or criminal organizations: Power, roles and facilitators in four cocaine trafficking structures. In *Third Annual Illicit Networks Workshop*, 2011.
- Donna K Ginther, Walter T Schaffer, Joshua Schnell, Beth Masimore, Faye Liu, Laurel L Haak, and Raynard Kington. Race, ethnicity, and nih research awards. *Science*, 333(6045):1015–1019, 2011.
- Herbert Gintis. *The bounds of reason: Game theory and the unification of the behavioral sciences*. Princeton University Press, 2014.
- Michelle Girvan and Mark EJ Newman. Community structure in social and biological networks. *Proceedings of the national academy of sciences*, 99(12):7821–7826, 2002.
- Minas Gjoka, Maciej Kurant, Carter T Butts, and Athina Markopoulou. Walking in facebook: A case study of unbiased sampling of osns. In *2010 Proceedings IEEE Infocom*, pages 1–9. Ieee, 2010.
- Minas Gjoka, Maciej Kurant, Carter T Butts, and Athina Markopoulou. Practical recommendations on crawling online social networks. *IEEE Journal on Selected Areas in Communications*, 29(9):1872–1892, 2011.
- James P Gleeson, Jonathan A Ward, Kevin P O’sullivan, and William T Lee. Competition-induced criticality in a model of meme popularity. *Physical review letters*, 112(4):048701, 2014.
- Pablo M Gleiser and Leon Danon. Community structure in jazz. *Advances in complex systems*, 6(04):565–573, 2003.
- Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323. JMLR Workshop and Conference Proceedings, 2011.
- Kurt Gödel. Über formal unentscheidbare sätze der principia mathematica und verwandter systeme i. *Monatshefte für mathematik und physik*, 38(1):173–198, 1931.
- Julio E Godoy, Ioannis Karamouzas, Stephen J Guy, and Maria Gini. Adaptive learning for multi-agent navigation. In *Int Conf on Autonomous Agents and Multiagent Systems*, pages 1577–1585. International Foundation for Autonomous Agents and Multiagent Systems, 2015.
- Chris Godsil and Gordon F Royle. *Algebraic graph theory*, volume 207. Springer Science & Business Media, 2013.

Andrew V Goldberg. Finding a maximum density subgraph. 1984.

Mark Goldberg, Malik Magdon-Ismail, Srinivas Nambirajan, and James Thompson. Tracking and predicting evolution of social communities. In *SocialCom*, pages 780–783. IEEE, 2011.

Yoav Goldberg and Omer Levy. word2vec explained: deriving mikolov et al.’s negative-sampling word-embedding method. *arXiv preprint arXiv:1402.3722*, 2014.

Jacob Goldenberg, Barak Libai, and Eitan Muller. Using complex systems analysis to advance marketing theory development: Modeling heterogeneity effects on new product growth through stochastic cellular automata. *Academy of Marketing Science Review*, 9(3):1–18, 2001.

Oded Goldreich. Finding the shortest move-sequence in the graph-generalized 15-puzzle is np-hard., 2011.

Alexander V Goltsev, Sergey N Dorogovtsev, and Jose Ferreira F Mendes. k-core (bootstrap) percolation on complex networks: Critical phenomena and nonlocal effects. *Physical Review E*, 73(5):056101, 2006.

Andres Gomez-Lievano, HyeJin Youn, and Luis MA Bettencourt. The statistics of urban scaling and their connection to zipf’s law. *PloS one*, 7(7), 2012.

Andres Gomez-Lievano, Oscar Patterson-Lomba, and Ricardo Hausmann. Explaining the prevalence, scaling and variance of urban phenomena. *Nature Energy*, pages 1–9, 2018.

Marta C González, Hans J Herrmann, J Kertész, and Tamás Vicsek. Community structure and ethnic preferences in school friendship networks. *Physica A*, 379(1):307–316, 2007.

Marta C Gonzalez, Cesar A Hidalgo, and Albert-Laszlo Barabasi. Understanding individual human mobility patterns. *nature*, 453(7196):779–782, 2008.

Benjamin H Good, Yves-Alexandre De Montjoye, and Aaron Clauset. Performance of modularity maximization in practical contexts. *Physical Review E*, 81(4):046106, 2010.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

Leo A Goodman. Snowball sampling. *The annals of mathematical statistics*, pages 148–170, 1961.

Marco Gori, Gabriele Monfardini, and Franco Scarselli. A new model for learning in graph domains. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 2, pages 729–734. IEEE, 2005.

Amit Goyal, Francesco Bonchi, and Laks VS Lakshmanan. Learning influence probabilities in social networks. In *Proceedings of the third ACM international conference on Web search and data mining*, pages 241–250. ACM, 2010.

Palash Goyal and Emilio Ferrara. Graph embedding techniques, applications, and performance: A survey. *Knowledge-Based Systems*, 151:78–94, 2018.

Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

Daniel Grady, Christian Thiemann, and Dirk Brockmann. Robust classification of salient links in complex networks. *Nature communications*, 3:864, 2012.

Ronald L Graham and Pavol Hell. On the history of the minimum spanning tree problem. *Annals of the History of Computing*, 7(1):43–57, 1985.

Mark Granovetter. Threshold models of collective behavior. *American journal of sociology*, 83(6):1420–1443, 1978.

Mark Granovetter. The strength of weak ties: A network theory revisited. 1983.

Mark Granovetter and Roland Soong. Threshold models of diffusion and collective behavior. *Journal of Mathematical sociology*, 9(3):165–179, 1983.

Mark Granovetter and Roland Soong. Threshold models of interpersonal effects in consumer demand. *Journal of Economic Behavior & Organization*, 7(1):83–99, 1986.

Mark Granovetter and Roland Soong. Threshold models of diversity: Chinese restaurants, residential segregation, and the spiral of silence. *Sociological methodology*, pages 69–104, 1988.

Mark S Granovetter. The strength of weak ties. In *Social networks*, pages 347–367. Elsevier, 1977.

Daniele Grattarola, Daniele Zambon, Filippo Maria Bianchi, and Cesare Alippi. Understanding pooling in graph neural networks. *IEEE transactions on neural networks and learning systems*, 2022.

Steve Gregory. An algorithm to find overlapping community structure in networks. In *European Conference on Principles of Data Mining and Knowledge Discovery*, pages 91–102. Springer, 2007.

Steve Gregory. Finding overlapping communities using disjoint community detection algorithms. In *Complex networks*, pages 47–61. Springer, 2009.

Steve Gregory. Finding overlapping communities in networks by label propagation. *New Journal of Physics*, 12(10):103018, 2010.

Jacopo Grilli, György Barabás, Matthew J Michalska-Smith, and Stefano Allesina. Higher-order interactions stabilize dynamics in competitive network models. *Nature*, 548(7666):210, 2017.

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

Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864. ACM, 2016.

Aditya Grover, Aaron Zweig, and Stefano Ermon. Graphite: Iterative generative modeling of graphs. In *ICML*, pages 2434–2444, 2019.

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

Jeremy Guillon, Mario Chavez, Federico Battiston, Yohan Attal, Valentina La Corte, Michel Thiebaut de Schotten, Bruno Dubois, Denis Schwartz, Olivier Colliot, and Fabrizio De Vico Fallani. Disrupted core-periphery structure of multimodal brain networks in alzheimer’s disease. *Network Neuroscience*, 3(2):635–652, 2019.

Paulo R Guimaraes Jr and Paulo Guimaraes. Improving the analyses of nestedness for large sets of matrices. *Environmental Modelling & Software*, 21(10):1512–1513, 2006.

Roger Guimera and Luis A Nunes Amaral. Functional cartography of complex metabolic networks. *nature*, 433(7028):895, 2005.

- Roger Guimerà and Marta Sales-Pardo. Missing and spurious interactions and the reconstruction of complex networks. *Proceedings of the National Academy of Sciences*, 106(52):22073–22078, 2009.
- Roger Guimera, Marta Sales-Pardo, and Luís A Nunes Amaral. Modularity from fluctuations in random graphs and complex networks. *Physical Review E*, 70(2):025101, 2004.
- Roger Guimerà, Marta Sales-Pardo, and Luís A Nunes Amaral. Module identification in bipartite and directed networks. *Physical Review E*, 76(3):036102, 2007.
- Natali Gulbahce and Sune Lehmann. The art of community detection. *BioEssays*, 30(10):934–938, 2008.
- Mangesh Gupte, Pravin Shankar, Jing Li, Shanmuganayagam Muthukrishnan, and Liviu Iftode. Finding hierarchy in directed online social networks. In *Proceedings of the 20th international conference on World wide web*, pages 557–566. ACM, 2011.
- Thomas R Hagadone. Molecular substructure similarity searching: efficient retrieval in two-dimensional structure databases. *Journal of chemical information and computer sciences*, 32(5):515–521, 1992.
- Aric Hagberg, Pieter Swart, and Daniel S Chult. Exploring network structure, dynamics, and function using networkx. Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008.
- Patric Hagmann, Leila Cammoun, Xavier Gigandet, Reto Meuli, Christopher J Honey, Van J Wedeen, and Olaf Sporns. Mapping the structural core of human cerebral cortex. *PLoS biology*, 6(7):e159, 2008.
- Petr Hájek. *Metamathematics of fuzzy logic*, volume 4. Springer Science & Business Media, 2013.
- Arda Halu, Raúl J Mondragón, Pietro Panzarasa, and Ginestra Bianconi. Multiplex pagerank. *PloS one*, 8(10):e78293, 2013.
- Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *NIPS*, pages 1024–1034, 2017.
- William L Hamilton. *Graph representation learning*. Morgan & Claypool Publishers, 2020.
- David K Hammond, Pierre Vandergheynst, and Rémi Gribonval. Wavelets on graphs via spectral graph theory. *Applied and Computational Harmonic Analysis*, 30(2):129–150, 2011.

Jiawei Han, Jian Pei, and Yiwen Yin. Mining frequent patterns without candidate generation. In *ACM sigmod record*, volume 29, pages 1–12. ACM, 2000.

Jiawei Han, Hong Cheng, Dong Xin, and Xifeng Yan. Frequent pattern mining: current status and future directions. *Data mining and knowledge discovery*, 15(1):55–86, 2007.

Jiawei Han, Jian Pei, and Hanghang Tong. *Data mining: concepts and techniques*. Morgan kaufmann, 2022.

Lin Han, Francisco Escolano, Edwin R Hancock, and Richard C Wilson. Graph characterizations from von neumann entropy. *Pattern Recognition Letters*, 33(15):1958–1967, 2012.

Yi Han, Shanika Karunasekera, and Christopher Leckie. Graph neural networks with continual learning for fake news detection from social media. *arXiv preprint arXiv:2007.03316*, 2020.

James A Hanley and Barbara J McNeil. The meaning and use of the area under a receiver operating characteristic (roc) curve. *Radiology*, 143(1):29–36, 1982.

Steve Hanneke, Wenjie Fu, Eric P Xing, et al. Discrete temporal models of social networks. *Electronic Journal of Statistics*, 4:585–605, 2010.

Robert A Hanneman and Mark Riddle. Introduction to social network methods. 2005.

Derek L Hansen, Ben Shneiderman, and Marc A Smith. *Analyzing social media networks with NodeXL: Insights from a connected world*. Morgan Kaufmann, 2010.

Lars Kai Hansen and Peter Salamon. Neural network ensembles. *IEEE transactions on pattern analysis and machine intelligence*, 12(10):993–1001, 1990.

Obaida Hanteer, Roberto Interdonato, Matteo Magnani, Andrea Tagarelli, and Luca Rossi. Community detection in multiplex networks, 2019.

Daniel Damir Harabor, Alban Grastien, et al. Online graph pruning for pathfinding on grid maps. In *AAAI*, pages 1114–1119, 2011.

Frank Harary. On the measurement of structural balance. *Behavioral Science*, 4(4):316–323, 1959.

Frank Harary. Graphs and matrices. *SIAM Review*, 9(1):83–90, 1967.

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

Frank Harary, Juhani Nieminen, et al. Convexity in graphs. *Journal of Differential Geometry*, 16(2):185–190, 1981.

Steve Harenberg, Gonzalo Bello, L Gjeltema, Stephen Ranshous, Jitendra Harlalka, Ramona Seay, Kanchana Padmanabhan, and Nagiza Samatova. Community detection in large-scale networks: a survey and empirical evaluation. *Wiley Interdisciplinary Reviews: Computational Statistics*, 6(6):426–439, 2014.

Mark Harrower and Cynthia A Brewer. Colorbrewer.org: an online tool for selecting colour schemes for maps. *The Cartographic Journal*, 40(1):27–37, 2003.

Richard A Harshman et al. Foundations of the parafac procedure: Models and conditions for an "explanatory" multimodal factor analysis. 1970.

Ki-ichiro Hashimoto. Zeta functions of finite graphs and representations of p-adic groups. In *Automorphic forms and geometry of arithmetic varieties*, pages 211–280. Elsevier, 1989.

Ricardo Hausmann, César A Hidalgo, Sebastián Bustos, Michele Coscia, Alexander Simoes, and Muhammed A Yildirim. *The atlas of economic complexity: Mapping paths to prosperity*. Mit Press, 2014.

Taher H Haveliwala. Topic-sensitive pagerank. In *Proceedings of the 11th international conference on World Wide Web*, pages 517–526. ACM, 2002.

Friedrich August Hayek. The use of knowledge in society. *The American economic review*, 35(4):519–530, 1945.

Megan L Head, Luke Holman, Rob Lanfear, Andrew T Kahn, and Michael D Jennions. The extent and consequences of p-hacking in science. *PLoS biology*, 13(3):e1002106, 2015.

Douglas D Heckathorn and Christopher J Cameron. Network sampling: From snowball and multiplicity to respondent-driven sampling. *Annual review of sociology*, 43:101–119, 2017.

David Heckerman, Chris Meek, and Daphne Koller. Probabilistic entity-relationship models, prms, and plate models. *Introduction to statistical relational learning*, pages 201–238, 2007.

Jeffrey Heer, Stuart K Card, and James A Landay. Prefuse: a toolkit for interactive information visualization. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 421–430. ACM, 2005.

Fritz Heider. *The psychology of interpersonal relations*. Psychology Press, 2013.

Mark Heimann, Haoming Shen, Tara Safavi, and Danai Koutra. Regal: Representation learning-based graph alignment. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, pages 117–126, 2018.

Keith Henderson, Brian Gallagher, Tina Eliassi-Rad, Hanghang Tong, Sugato Basu, Leman Akoglu, Danai Koutra, Christos Faloutsos, and Lei Li. Rolx: structural role extraction & mining in large graphs. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1231–1239. ACM, 2012.

Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

Herbert W Hethcote. Three basic epidemiological models. In *Applied mathematical ecology*, pages 119–144. Springer, 1989.

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

César A Hidalgo, Bailey Klinger, A-L Barabási, and Ricardo Hausmann. The product space conditions the development of nations. *Science*, 317(5837):482–487, 2007.

Frank L Hitchcock. The expression of a tensor or a polyadic as a sum of products. *Journal of Mathematics and Physics*, 6(1-4):164–189, 1927.

Frank L Hitchcock. The distribution of a product from several sources to numerous localities. *Studies in Applied Mathematics*, 20(1-4):224–230, 1941.

Manel Hmimida and Rushed Kanawati. Community detection in multiplex networks: A seed-centric approach. *NHM*, 10(1):71–85, 2015.

Qirong Ho, Le Song, and Eric Xing. Evolving cluster mixed-membership blockmodel for time-evolving networks. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 342–350, 2011.

Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02):107–116, 1998.

Nathan Oken Hodas and Kristina Lerman. How visibility and divided attention constrain social contagion. In *SocialCom*, pages 249–257. IEEE, 2012.

Douglas R Hofstadter. *Gödel, Escher, Bach*. Harvester press Hassocks, Sussex, 1979.

Marilena Hohmann, Karel Devriendt, and Michele Coscia. Quantifying ideological polarization on a network using generalized euclidean distance. *Science Advances*, 9(9):eabq2044, 2023.

Daniel A Hojman and Adam Szeidl. Core and periphery in networks. *Journal of Economic Theory*, 139(1):295–309, 2008.

Paul W Holland and Samuel Leinhardt. Transitivity in structural models of small groups. *Comparative group studies*, 2(2):107–124, 1971.

Paul W Holland and Samuel Leinhardt. An exponential family of probability distributions for directed graphs. *Journal of the american Statistical association*, 76(373):33–50, 1981.

Paul W Holland, Kathryn Blackmond Laskey, and Samuel Leinhardt. Stochastic blockmodels: First steps. *Social networks*, 5(2):109–137, 1983.

Sture Holm. A simple sequentially rejective multiple test procedure. *Scandinavian journal of statistics*, pages 65–70, 1979.

Petter Holme. Core-periphery organization of complex networks. *Phys. Rev. E*, 72:046111, Oct 2005. doi: 10.1103/PhysRevE.72.046111.

Petter Holme and Beom Jun Kim. Growing scale-free networks with tunable clustering. *Physical review E*, 65(2):026107, 2002.

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

Petter Holme, Mikael Huss, and Sang Hoon Lee. Atmospheric reaction systems as null-models to identify structural traces of evolution in metabolism. *PLoS One*, 6(5):e19759, 2011.

Danny Holten. Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data. *IEEE Transactions on visualization and computer graphics*, 12(5):741–748, 2006.

Danny Holten and Jarke J Van Wijk. Force-directed edge bundling for graph visualization. In *Computer graphics forum*, volume 28, pages 983–990. Wiley Online Library, 2009.

Sungpack Hong, Nicole C Rodia, and Kunle Olukotun. On fast parallel detection of strongly connected components (scc) in small-world graphs. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, pages 1–11, 2013.

John Hopcroft, Omar Khan, Brian Kulis, and Bart Selman. Tracking evolving communities in large linked networks. *Proceedings of the National Academy of Sciences*, 101(suppl 1):5249–5253, 2004.

Harold Hotelling. Stability in competition. *The Economic Journal*, 39(153):41–57, 1929.

Darko Hric, Richard K Darst, and Santo Fortunato. Community detection in networks: Structural communities versus ground truth. *Physical Review E*, 90(6):062805, 2014.

Desislava Hristova, Mirco Musolesi, and Cecilia Mascolo. Keep your friends close and your facebook friends closer: A multiplex network approach to the analysis of offline and online social ties. In *Eighth International AAAI Conference on Weblogs and Social Media*, 2014.

Desislava Hristova, Anastasios Noulas, Chloë Brown, Mirco Musolesi, and Cecilia Mascolo. A multilayer approach to multiplexity and link prediction in online geo-social networks. *EPJ Data Science*, 5(1):24, 2016.

Ren-Jie Hu, Qing Li, Guang-Yu Zhang, and Wen-Cong Ma. Centrality measures in directed fuzzy social networks. *Fuzzy Information and Engineering*, 7(1):115–128, 2015.

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

Jun Huan, Wei Wang, and Jan Prins. Efficient mining of frequent subgraphs in the presence of isomorphism. In *Third IEEE International Conference on Data Mining*, pages 549–552. IEEE, 2003.

Jianbin Huang, Heli Sun, Jiawei Han, Hongbo Deng, Yizhou Sun, and Yaguang Liu. Shrink: a structural clustering algorithm for

detecting hierarchical communities in networks. In *Proceedings of the 19th ACM international conference on Information and knowledge management*, pages 219–228. ACM, 2010.

Jianbin Huang, Heli Sun, Yaguang Liu, Qinbao Song, and Tim Weninger. Towards online multiresolution community detection in large-scale networks. *PLoS one*, 6(8):e23829, 2011a.

Junjie Huang, Huawei Shen, Liang Hou, and Xueqi Cheng. Signed graph attention networks. In *ICANN 2019: Workshop and Special Sessions*, pages 566–577. Springer, 2019.

Xuqing Huang, Jianxi Gao, Sergey V Buldyrev, Shlomo Havlin, and H Eugene Stanley. Robustness of interdependent networks under targeted attack. *Physical Review E*, 83(6):065101, 2011b.

Raymond Hubbard and R Murray Lindsay. Why p values are not a useful measure of evidence in statistical significance testing. *Theory & Psychology*, 18(1):69–88, 2008.

Lawrence Hubert and Phipps Arabie. Comparing partitions. *Journal of classification*, 2(1):193–218, 1985.

Darrell Huff. *How to lie with statistics*. Penguin UK, 1954.

Iacopo Iacopini, Giovanni Petri, Alain Barrat, and Vito Latora. Simplicial models of social contagion. *Nature communications*, 10(1):2485, 2019.

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.

Vijay Ingallali, Dino Ienco, and Pascal Poncelet. Sumgra: Querying multigraphs via efficient indexing. In *International Conference on Database and Expert Systems Applications*, pages 387–401. Springer, 2016.

Roberto Interdonato, Andrea Tagarelli, Dino Ienco, Arnaud Salaberry, and Pascal Poncelet. Local community detection in multilayer networks. *DMKD*, 31(5):1444–1479, 2017.

John PA Ioannidis. Why most published research findings are false. *PLoS medicine*, 2(8):e124, 2005.

Md Shahadat Iqbal, Charisma F Choudhury, Pu Wang, and Marta C González. Development of origin–destination matrices using mobile phone call data. *Transportation Research Part C*, 40:63–74, 2014.

Hiroshi Ishikawa. Higher-order clique reduction in binary graph cut. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2993–3000. IEEE, 2009.

Takeshi D Itoh, Takatomi Kubo, and Kazushi Ikeda. Multi-level attention pooling for graph neural networks: Unifying graph representations with multiple localities. *Neural Networks*, 145:356–373, 2022.

Shruti Jadon. A survey of loss functions for semantic segmentation. In *2020 IEEE conference on computational intelligence in bioinformatics and computational biology (CIBCB)*, pages 1–7. IEEE, 2020.

Ashesh Jain, Amir R Zamir, Silvio Savarese, and Ashutosh Saxena. Structural-rnn: Deep learning on spatio-temporal graphs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5308–5317, 2016.

Mahdi Jalili, Yasin Orouskhani, Milad Asgari, Nazanin Alipourfard, and Matjaž Perc. Link prediction in multiplex online social networks. *Royal Society open science*, 4(2):160863, 2017.

Svante Janson, Donald E Knuth, Tomasz Łuczak, and Boris Pittel. The birth of the giant component. *Random Structures & Algorithms*, 4(3):233–358, 1993.

Edwin T Jaynes. *Probability theory: The logic of science*. Cambridge university press, 2003.

Glen Jeh and Jennifer Widom. Simrank: a measure of structural-context similarity. KDD ’02, New York, NY, USA, 2002. ACM. ISBN 158113567X. doi: 10.1145/775047.775126. URL <https://doi.org/10.1145/775047.775126>.

Glen Jeh and Jennifer Widom. Scaling personalized web search. In *Proceedings of the 12th international conference on World Wide Web*, pages 271–279. Acm, 2003.

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

Tommy R Jensen and Bjarne Toft. *Graph coloring problems*, volume 39. John Wiley & Sons, 2011.

Lucas GS Jeub, Prakash Balachandran, Mason A Porter, Peter J Mucha, and Michael W Mahoney. Think locally, act locally: Detection of small, medium-sized, and large communities in large networks. *Physical Review E*, 91(1):012821, 2015.

- Lucas GS Jeub, Michael W Mahoney, Peter J Mucha, and Mason A Porter. A local perspective on community structure in multilayer networks. *Network Science*, 5(2):144–163, 2017.
- Guoliang Ji, Shizhu He, Liheng Xu, Kang Liu, and Jun Zhao. Knowledge graph embedding via dynamic mapping matrix. In *IJCNLP*, pages 687–696, 2015.
- Yuxin Jiang, Daniel I Bolnick, and Mark Kirkpatrick. Assortative mating in animals. *The American Naturalist*, 181(6):E125–E138, 2013.
- Ruoming Jin, Lin Liu, and Charu C Aggarwal. Discovering highly reliable subgraphs in uncertain graphs. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 992–1000, 2011.
- Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Junction tree variational autoencoder for molecular graph generation. In *International conference on machine learning*, pages 2323–2332. PMLR, 2018.
- Gudlaugur Jóhannesson, Gunnlaugur Björnsson, and Einar H Gudmundsson. Afterglow light curves and broken power laws: a statistical study. *The Astrophysical Journal Letters*, 640(1):L5, 2006.
- Samuel Johnson and Nick S Jones. Looplessness in networks is linked to trophic coherence. *Proceedings of the National Academy of Sciences*, 114(22):5618–5623, 2017.
- Samuel Jonhson, Virginia Domínguez-García, and Miguel A Muñoz. Factors determining nestedness in complex networks. *PloS one*, 8(9):e74025, 2013.
- Jakob Jonsson. *Simplicial complexes of graphs*, volume 3. Springer, 2008.
- Kara Joyner and Grace Kao. School racial composition and adolescent racial homophily. *Social science quarterly*, pages 810–825, 2000.
- Inderjit S Jutla, Lucas GS Jeub, and Peter J Mucha. A generalized louvain method for community detection implemented in matlab. URL <http://netwiki. amath. unc. edu/GenLouvain>, 2011.
- Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.
- Tomihisa Kamada and Satoru Kawai. A simple method for computing general position in displaying three-dimensional objects. *Computer Vision, Graphics, and Image Processing*, 41(1):43–56, 1988.

Jermain Kaminski, Michael Schober, Raymond Albaladejo, Oleksandr Zastupailo, and Cesar Hidalgo. Moviegalaxies-social networks in movies. 2018.

U Kang, Hanghang Tong, and Jimeng Sun. Fast random walk graph kernel. In *Proceedings of the 2012 SIAM international conference on data mining*, pages 828–838. SIAM, 2012.

Ravi Kannan, Santosh Vempala, and Adrian Vetta. On clusterings: Good, bad and spectral. *Journal of the ACM (JACM)*, 51(3):497–515, 2004.

George Karakostas. Faster approximation schemes for fractional multicommodity flow problems. *ACM Transactions on Algorithms (TALG)*, 4(1):13, 2008.

Brian Karrer and Mark EJ Newman. Stochastic blockmodels and community structure in networks. *Physical review E*, 83(1):016107, 2011.

Brian Karrer, Elizaveta Levina, and Mark EJ Newman. Robustness of community structure in networks. *Physical review E*, 77(4):046119, 2008.

Brian Karrer, Mark EJ Newman, and Lenka Zdeborová. Percolation on sparse networks. *Physical review letters*, 113(20):208702, 2014.

Leo Katz. A new status index derived from sociometric analysis. *Psychometrika*, 18(1):39–43, 1953.

Amin Kaveh, Matteo Magnani, and Christian Rohner. Comparing node degrees in probabilistic networks. *Journal of Complex Networks*, 7(5):749–763, 2019.

Amin Kaveh, Matteo Magnani, and Christian Rohner. Defining and measuring probabilistic ego networks. *Social Network Analysis and Mining*, 11:1–12, 2021a.

Amin Kaveh, Matteo Magnani, and Christian Rohner. Probabilistic network sparsification with ego betweenness. *Applied Network Science*, 6:1–21, 2021b.

Seyed Mehran Kazemi, Rishab Goel, Kshitij Jain, Ivan Kobyzev, Akshay Sethi, Peter Forsyth, and Pascal Poupart. Representation learning for dynamic graphs: A survey. *Journal of Machine Learning Research*, 21(70):1–73, 2020.

Brian P Kelley, Roded Sharan, Richard M Karp, Taylor Sittler, David E Root, Brent R Stockwell, and Trey Ideker. Conserved

pathways within bacteria and yeast as revealed by global protein network alignment. *Proceedings of the National Academy of Sciences*, 100(20):11394–11399, 2003.

David Kempe, Jon Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 137–146. ACM, 2003.

David Kempe, Jon Kleinberg, and Éva Tardos. Influential nodes in a diffusion model for social networks. In *International Colloquium on Automata, Languages, and Programming*, pages 1127–1138. Springer, 2005.

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

William Ogilvy Kermack and Anderson G McKendrick. A contribution to the mathematical theory of epidemics. *Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character*, 115(772):700–721, 1927.

Elias Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. In *Advances in Neural Information Processing Systems*, pages 6348–6358, 2017.

Arijit Khan, Francesco Bonchi, Aristides Gionis, and Francesco Gullo. Fast reliability search in uncertain graphs. In *EDBT*, pages 535–546, 2014.

Masoumeh Kheirkhahzadeh, Andrea Lancichinetti, and Martin Rosvall. Efficient community detection of network flows for varying markov times and bipartite networks. *Physical Review E*, 93(3):032309, 2016.

Jin Seop Kim, Kwang-Il Goh, Byungnam Kahng, and Doochul Kim. Fractality and self-similarity in scale-free networks. *New Journal of Physics*, 9(6):177, 2007.

Jungeun Kim and Jae-Gil Lee. Community detection in multi-layer graphs: A survey. *ACM SIGMOD Record*, 44(3):37–48, 2015.

Jungeun Kim, Jae-Gil Lee, and Sungsu Lim. Differential flattening: A novel framework for community detection in multi-layer graphs. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 8(2):27, 2017.

Youngdo Kim, Seung-Woo Son, and Hawoong Jeong. Finding communities in directed networks. *Physical Review E*, 81(1):016103, 2010.

Gary King. *Unifying political methodology: The likelihood theory of statistical inference*. University of Michigan Press, 1998.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Thomas Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017.

Thomas N Kipf and Max Welling. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*, 2016.

Gustav Kirchhoff. Ueber die auflösung der gleichungen, auf welche man bei der untersuchung der linearen vertheilung galvanischer ströme geführt wird. *Annalen der Physik*, 148(12):497–508, 1847.

Maksim Kitsak, Ivan Voitalov, and Dmitri Krioukov. Link prediction with hyperbolic geometry. *Physical Review Research*, 2(4):043113, 2020.

Mikko Kivelä and Mason A Porter. Isomorphisms in multilayer networks. *IEEE Transactions on Network Science and Engineering*, 5(3):198–211, 2017.

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.

Gunnar W Klau. A new graph-based method for pairwise global network alignment. *BMC bioinformatics*, 10(1):S59, 2009.

Cerry M Klein. Fuzzy shortest paths. *Fuzzy sets and systems*, 39(1):27–41, 1991.

Jon M Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM (JACM)*, 46(5):604–632, 1999.

Jon M Kleinberg, Ravi Kumar, Prabhakar Raghavan, Sridhar Rajagopalan, and Andrew S Tomkins. The web as a graph: measurements, models, and methods. In *International Computing and Combinatorics Conference*, pages 1–17. Springer, 1999.

Kaj-Kolja Kleineberg, Marián Boguná, M Ángeles Serrano, and Fragkiskos Papadopoulos. Hidden geometric correlations in real multiplex networks. *Nature Physics*, 12(11):1076, 2016.

- Yuval Kluger, Ronen Basri, Joseph T Chang, and Mark Gerstein. Spectral biclustering of microarray data: coclustering genes and conditions. *Genome research*, 13(4):703–716, 2003.
- Donald Ervin Knuth. *The Stanford GraphBase: a platform for combinatorial computing*. AcM Press New York, 1993.
- Jihoon Ko, Yunbum Kook, and Kijung Shin. Incremental lossless graph summarization. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 317–327, 2020.
- Ron Kohavi et al. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Ijcai*, volume 14, pages 1137–1145. Montreal, Canada, 1995.
- Sadamori Kojaku and Naoki Masuda. Core-periphery structure requires something else in the network. *New Journal of Physics*, 20(4):043012, 2018.
- Tamara Kolda and Brett Bader. The tophits model for higher-order web link analysis. In *Workshop on link analysis, counterterrorism and security*, volume 7, pages 26–29, 2006.
- Tamara G Kolda and Brett W Bader. Tensor decompositions and applications. *SIAM review*, 51(3):455–500, 2009.
- Giorgos Kollias, Shahin Mohammadi, and Ananth Grama. Network similarity decomposition (nsd): A fast and scalable approach to network alignment. *IEEE Transactions on Knowledge and Data Engineering*, 24(12):2232–2243, 2011.
- George Kollios, Michalis Potamias, and Eviatar Terzi. Clustering large probabilistic graphs. *IEEE Transactions on Knowledge and Data Engineering*, 25(2):325–336, 2011.
- Andrei N Kolmogorov. Three approaches to the quantitative definition of information'. *Problems of information transmission*, 1(1):1–7, 1965.
- Andrey Kolmogorov. Sulla determinazione empirica di una legge di distribuzione. *Inst. Ital. Attuari, Giorn.*, 4:83–91, 1933.
- Xiangnan Kong, Jiawei Zhang, and Philip S Yu. Inferring anchor links across multiple heterogeneous social networks. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*, pages 179–188. ACM, 2013.

Stephen Kosack, Michele Coscia, Evann Smith, Kim Albrecht, Albert-László Barabási, and Ricardo Hausmann. Functional structures of us state governments. *Proceedings of the National Academy of Sciences*, 115(46):11748–11753, 2018.

Danai Koutra, Joshua T Vogelstein, and Christos Faloutsos. Deltacon: A principled massive-graph similarity function. In *Proceedings of the 2013 SIAM International Conference on Data Mining*, pages 162–170. SIAM, 2013.

Danai Koutra, U Kang, Jilles Vreeken, and Christos Faloutsos. Vog: Summarizing and understanding large graphs. In *Proceedings of the 2014 SIAM international conference on data mining*, pages 91–99. SIAM, 2014.

Lauri Kovanen, Márton Karsai, Kimmo Kaski, János Kertész, and Jari Saramäki. Temporal motifs in time-dependent networks. *Journal of Statistical Mechanics: Theory and Experiment*, (11):P11005, 2011.

David Krackhardt, N Nohria, and B Eccles. The strength of strong ties. *Networks in the knowledge economy*, 82, 2003.

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

Devin Kreuzer, Dominique Beaini, Will Hamilton, Vincent Lé-tourneau, and Prudencio Tossou. Rethinking graph transformers with spectral attention. *Advances in Neural Information Processing Systems*, 34:21618–21629, 2021.

Dmitri Krioukov, Fragkiskos Papadopoulos, Maksim Kitsak, Amin Vahdat, and Marián Boguná. Hyperbolic geometry of complex networks. *Physical Review E*, 82(3):036106, 2010.

Balachander Krishnamurthy, Phillipa Gill, and Martin Arlitt. A few chirps about twitter. In *Proceedings of the first workshop on Online social networks*, pages 19–24. ACM, 2008.

Pavel N Krivitsky. Exponential-family random graph models for valued networks. *Electronic journal of statistics*, 6:1100, 2012.

Pavel N Krivitsky and Mark S Handcock. A separable model for dynamic networks. *Journal of the Royal Statistical Society. Series B, Statistical Methodology*, 76(1):29, 2014.

Joseph B Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical society*, 7(1):48–50, 1956.

Joseph B Kruskal. Three-way arrays: rank and uniqueness of trilinear decompositions, with application to arithmetic complexity and statistics. *Linear algebra and its applications*, 18(2):95–138, 1977.

Florent Krzakala, Cristopher Moore, Elchanan Mossel, Joe Neeman, Allan Sly, Lenka Zdeborová, and Pan Zhang. Spectral redemption in clustering sparse networks. *Proceedings of the National Academy of Sciences*, 110(52):20935–20940, 2013.

Martin Krzywinski, Inanc Birol, Steven JM Jones, and Marco A Marra. Hive plots – rational approach to visualizing networks. *Briefings in bioinformatics*, 13(5):627–644, 2011.

Emily Kubin and Christian von Sikorski. The role of (social) media in political polarization: a systematic review. *Annals of the International Communication Association*, 45(3):188–206, 2021.

Oleksii Kuchaiev and Nataša Pržulj. Integrative network alignment reveals large regions of global network similarity in yeast and human. *Bioinformatics*, 27(10):1390–1396, 2011.

Jussi M Kumpula, Mikko Kivelä, Kimmo Kaski, and Jari Saramäki. Sequential algorithm for fast clique percolation. *Physical Review E*, 78(2):026109, 2008.

Zhana Kuncheva and Giovanni Montana. Community detection in multiplex networks using locally adaptive random walks. In *ASONAM*, pages 1308–1315. ACM, 2015.

Jérôme Kunegis. Konect: the koblenz network collection. In *Proceedings of the 22nd International Conference on World Wide Web*, pages 1343–1350, 2013.

Michihiko Kuramochi and George Karypis. Finding frequent patterns in a large sparse graph. *Data mining and knowledge discovery*, 11(3):243–271, 2005.

James Ladyman, James Lambert, and Karoline Wiesner. What is a complex system? *European Journal for Philosophy of Science*, 3:33–67, 2013.

Darong Lai, Christine Nardini, and Hongtao Lu. Partitioning networks into communities by message passing. *Physical review E*, 83(1):016115, 2011.

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

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

Renaud Lambiotte, J-C Delvenne, and Mauricio Barahona. Laplacian dynamics and multiscale modular structure in networks. *arXiv preprint arXiv:0812.1770*, 2008.

Godfrey N Lance and William T Williams. Computer programs for hierarchical polythetic classification (“similarity analyses”). *The Computer Journal*, 9(1):60–64, 1966.

Andrea Lancichinetti and Santo Fortunato. Community detection algorithms: a comparative analysis. *Physical review E*, 80(5):056117, 2009.

Andrea Lancichinetti and Santo Fortunato. Limits of modularity maximization in community detection. *Physical review E*, 84(6):066122, 2011.

Andrea Lancichinetti and Santo Fortunato. Consensus clustering in complex networks. *Scientific reports*, 2:336, 2012.

Andrea Lancichinetti, Santo Fortunato, and Filippo Radicchi. Benchmark graphs for testing community detection algorithms. *Physical review E*, 78(4):046110, 2008.

Andrea Lancichinetti, Santo Fortunato, and János Kertész. Detecting the overlapping and hierarchical community structure in complex networks. *New Journal of Physics*, 11(3):033015, 2009.

Andrea Lancichinetti, Filippo Radicchi, José J Ramasco, and Santo Fortunato. Finding statistically significant communities in networks. *PloS one*, 6(4):e18961, 2011.

Nicholas W Landry and Juan G Restrepo. The effect of heterogeneity on hypergraph contagion models. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 30(10), 2020.

Daniel B Larremore, Aaron Clauset, and Abigail Z Jacobs. Efficiently inferring community structure in bipartite networks. *Physical Review E*, 90(1):012805, 2014.

Vito Latora and Massimo Marchiori. Vulnerability and protection of infrastructure networks. *Physical Review E*, 71(1):015103, 2005.

Vito Latora, Vincenzo Nicosia, and Giovanni Russo. *Complex networks: principles, methods and applications*. Cambridge University Press, 2017.

Eugene L Lawler, Jan Karel Lenstra, AHG Rinnooy Kan, David Bernard Shmoys, et al. *The traveling salesman problem: a guided tour of combinatorial optimization*, volume 3. Wiley New York, 1985.

Anna Lázár, Dániel Ábel, and Tamás Vicsek. Modularity measure of networks with overlapping communities. *EPL*, 90(1):18001, 2010.

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

Emmanuel Lazega and Marijte Van Duijn. Position in formal structure, personal characteristics and choices of advisors in a law firm: A logistic regression model for dyadic network data. *Social networks*, 19(4):375–397, 1997.

David Lazer, Alex Sandy Pentland, Lada Adamic, Sinan Aral, Albert Laszlo Barabasi, Devon Brewer, Nicholas Christakis, Noshir Contractor, James Fowler, Myron Gutmann, et al. Life in the network: the coming age of computational social science. *Science (New York, NY)*, 323(5915):721, 2009.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

Ernest Bruce Lee and Lawrence Markus. Foundations of optimal control theory. Technical report, Minnesota Univ Minneapolis Center For Control Sciences, 1967.

Sang Hoon Lee, Pan-Jun Kim, and Hawoong Jeong. Statistical properties of sampled networks. *Physical Review E*, 73(1):016102, 2006.

Sang Hoon Lee et al. Network nestedness as generalized core-periphery structures. *Physical Review E*, 93(2):022306, 2016.

Kristen LeFevre and Evimaria Terzi. Grass: Graph structure summarization. In *Proceedings of the 2010 SIAM International Conference on Data Mining*, pages 454–465. SIAM, 2010.

Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick Van Kleef, Sören Auer, et al. Dbpedia—a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic Web*, 6(2):167–195, 2015.

Elizabeth A Leicht and Mark EJ Newman. Community structure in directed networks. *Physical review letters*, 100(11):118703, 2008.

Elizabeth A Leicht, Petter Holme, and Mark EJ Newman. Vertex similarity in networks. *Physical Review E*, 73(2):026120, 2006.

Claire Lemercier. Formal network methods in history: why and how? In *Social networks, political institutions, and rural societies*, pages 281–310. 2015.

Ronny Lempel and Shlomo Moran. Salsa: the stochastic approach for link-structure analysis. *ACM Transactions on Information Systems (TOIS)*, 19(2):131–160, 2001.

Kristina Lerman and Rumi Ghosh. Information contagion: An empirical study of the spread of news on digg and twitter social networks. In *ICWSM*, 2010.

Kristina Lerman, Xiaoran Yan, and Xin-Zeng Wu. The "majority illusion" in social networks. *PloS one*, 11(2):e0147617, 2016.

Jure Leskovec and Christos Faloutsos. Sampling from large graphs. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 631–636. ACM, 2006.

Jure Leskovec and Christos Faloutsos. Scalable modeling of real graphs using kronecker multiplication. In *Proceedings of the 24th international conference on Machine learning*, pages 497–504. ACM, 2007.

Jure Leskovec and Julian J Mcauley. Learning to discover social circles in ego networks. In *Advances in neural information processing systems*, pages 539–547, 2012.

Jure Leskovec and Rok Sosič. Snap: A general-purpose network analysis and graph-mining library. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 8(1):1–20, 2016.

Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graphs over time: densification laws, shrinking diameters and possible explanations. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 177–187. ACM, 2005a.

Jure Leskovec, Lada A Adamic, and Bernardo A Huberman. The dynamics of viral marketing. *ACM Transactions on the Web (TWEB)*, 1(1):5, 2007a.

Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graph evolution: Densification and shrinking diameters. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 1(1):2, 2007b.

Jure Leskovec, Kevin J Lang, Anirban Dasgupta, and Michael W Mahoney. Statistical properties of community structure in large social and information networks. In *Proceedings of the 17th international conference on World Wide Web*, pages 695–704. ACM, 2008.

Jure Leskovec, Kevin J Lang, Anirban Dasgupta, and Michael W Mahoney. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. *Internet Mathematics*, 6(1):29–123, 2009.

Jure Leskovec, Daniel Huttenlocher, and Jon Kleinberg. Signed networks in social media. In *Proceedings of the SIGCHI conference on human factors in computing systems*, pages 1361–1370. ACM, 2010a.

Jure Leskovec, Kevin J Lang, and Michael Mahoney. Empirical comparison of algorithms for network community detection. In *WWW*, pages 631–640. ACM, 2010b.

Jurij Leskovec, Deepayan Chakrabarti, Jon Kleinberg, and Christos Faloutsos. Realistic, mathematically tractable graph generation and evolution, using kronecker multiplication. In *European Conference on Principles of Data Mining and Knowledge Discovery*, pages 133–145. Springer, 2005b.

Vladimir I Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, volume 10, pages 707–710, 1966.

Ron Levie, Federico Monti, Xavier Bresson, and Michael M Bronstein. Cayleynets: Graph convolutional neural networks with complex rational spectral filters. *IEEE Transactions on Signal Processing*, 67(1):97–109, 2018.

Cheng-Te Li and Shou-De Lin. Egocentric information abstraction for heterogeneous social networks. In *2009 International Conference on Advances in Social Network Analysis and Mining*, pages 255–260. IEEE, 2009.

Chun-Hsien Li, Chiung-Chiou Tsai, and Suh-Yuh Yang. Analysis of epidemic spreading of an sirs model in complex heterogeneous networks. *Communications in Nonlinear Science and Numerical Simulation*, 19(4):1042–1054, 2014.

Geng Li, Murat Semerci, Bulent Yener, and Mohammed J Zaki. Graph classification via topological and label attributes. In *Proceedings of the 9th international workshop on mining and learning with graphs (MLG), San Diego, USA*, volume 2, 2011.

Jiaoyang Li, Pavel Surynek, Ariel Felner, Hang Ma, TK Satish Kumar, and Sven Koenig. Multi-agent path finding for large agents. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 7627–7634, 2019.

Juanhui Li, Yao Ma, Yiqi Wang, Charu Aggarwal, Chang-Dong Wang, and Jiliang Tang. Graph pooling with representativeness. In *2020 IEEE International Conference on Data Mining (ICDM)*, pages 302–311. IEEE, 2020.

Jundong Li, Harsh Dani, Xia Hu, Jiliang Tang, Yi Chang, and Huan Liu. Attributed network embedding for learning in a dynamic environment. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pages 387–396, 2017a.

Menghui Li, Ying Fan, Jiawei Chen, Liang Gao, Zengru Di, and Jinshan Wu. Weighted networks of scientific communication: the measurement and topological role of weight. *Physica A: Statistical Mechanics and its Applications*, 350(2-4):643–656, 2005.

Michael Y Li and James S Muldowney. Global stability for the seir model in epidemiology. *Mathematical biosciences*, 125(2):155–164, 1995.

Michael Y Li, John R Graef, Liancheng Wang, and János Karsai. Global dynamics of a seir model with varying total population size. *Mathematical biosciences*, 160(2):191–213, 1999.

Siming Li, Christopher M Armstrong, Nicolas Bertin, Hui Ge, Stuart Milstein, Mike Boxem, Pierre-Olivier Vidalain, Jing-Dong J Han, Alban Chesneau, Tong Hao, et al. A map of the interactome network of the metazoan *c. elegans*. *Science*, 303(5657):540–543, 2004.

Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu. Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. *arXiv preprint arXiv:1707.01926*, 2017b.

Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*, 2015.

Yujia Li, Oriol Vinyals, Chris Dyer, Razvan Pascanu, and Peter Battaglia. Learning deep generative models of graphs. *arXiv preprint arXiv:1803.03324*, 2018.

Lizi Liao, Xiangnan He, Hanwang Zhang, and Tat-Seng Chua. Attributed social network embedding. *IEEE Transactions on Knowledge and Data Engineering*, 30(12):2257–2270, 2018.

Renjie Liao, Yujia Li, Yang Song, Shenlong Wang, Will Hamilton, David K Duvenaud, Raquel Urtasun, and Richard Zemel. Efficient graph generation with graph recurrent attention networks. *Advances in neural information processing systems*, 32, 2019a.

Renjie Liao, Zhizhen Zhao, Raquel Urtasun, and Richard S Zemel. Lanczosnet: Multi-scale deep graph convolutional networks. *arXiv preprint arXiv:1901.01484*, 2019b.

David Liben-Nowell and Jon Kleinberg. The link-prediction problem for social networks. *Journal of the American society for information science and technology*, 58(7):1019–1031, 2007.

Edo Liberty. Simple and deterministic matrix sketching. In *SIGKDD*, pages 581–588, 2013.

Ryan N Lichtenwalter, Jake T Lussier, and Nitesh V Chawla. New perspectives and methods in link prediction. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 243–252, 2010.

Ryan Lichtenwalter and Nitesh V Chawla. Link prediction: fair and effective evaluation. In *2012 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, pages 376–383. IEEE, 2012.

Nan Lin. *Foundations of social research*. McGraw-Hill Companies, 1976.

Yankai Lin, Zhiyuan Liu, Maosong Sun, Yang Liu, and Xuan Zhu. Learning entity and relation embeddings for knowledge graph completion. In *Twenty-ninth AAAI conference on artificial intelligence*, 2015.

Zhouhan Lin, Minwei Feng, Cicero Nogueira dos Santos, Mo Yu, Bing Xiang, Bowen Zhou, and Yoshua Bengio. A structured self-attentive sentence embedding. *arXiv preprint arXiv:1703.03130*, 2017.

Seppo Linnainmaa. *The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors*. PhD thesis, Master’s Thesis (in Finnish), Univ. Helsinki, 1970.

Baoding Liu. *Uncertainty theory*. Springer, 2010.

Chuang Liu, Yibing Zhan, Jia Wu, Chang Li, Bo Du, Wenbin Hu, Tongliang Liu, and Dacheng Tao. Graph pooling for graph neural networks: Progress, challenges, and opportunities. *arXiv preprint arXiv:2204.07321*, 2022.

- Kun Liu and Evinaria Terzi. Towards identity anonymization on graphs. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 93–106, 2008.
- Lin Liu, Ruoming Jin, Charu Aggarwal, and Yelong Shen. Reliable clustering on uncertain graphs. In *2012 IEEE 12th international conference on data mining*, pages 459–468. IEEE, 2012.
- Lu Liu, Yang Wang, Roberta Sinatra, C Lee Giles, Chaoming Song, and Dashun Wang. Hot streaks in artistic, cultural, and scientific careers. *Nature*, 559(7714):396, 2018a.
- Meng Liu, Yue Liu, Ke Liang, Wenxuan Tu, Siwei Wang, Sihang Zhou, and Xinwang Liu. Deep temporal graph clustering. *ICLR*, 2024.
- Minghua Liu, Hang Ma, Jiaoyang Li, and Sven Koenig. Task and path planning for multi-agent pickup and delivery. In *Int Conf on Autonomous Agents and MultiAgent Systems*, pages 1152–1160. IFAAMAS, 2019.
- Weiping Liu and Linyuan Lü. Link prediction based on local random walk. *EPL (Europhysics Letters)*, 89(5):58007, 2010.
- Xin Liu and Tsuyoshi Murata. Community detection in large-scale bipartite networks. *Transactions of the Japanese Society for Artificial Intelligence*, 25(1):16–24, 2010.
- Yang Liu and Jeffrey Heer. Somewhere over the rainbow: An empirical assessment of quantitative colormaps. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, pages 1–12, 2018.
- Yang-Yu Liu, Jean-Jacques Slotine, and Albert-László Barabási. Controllability of complex networks. *nature*, 473(7346):167, 2011.
- Yike Liu, Tara Safavi, Abhilash Dighe, and Danai Koutra. Graph summarization methods and applications: A survey. *ACM Computing Surveys (CSUR)*, 51(3):1–34, 2018b.
- Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. *arXiv preprint arXiv:1810.05270*, 2018c.
- Stuart Lloyd. Least squares quantization in pcm. *IEEE transactions on information theory*, 28(2):129–137, 1982.
- Dale F Lott. Dominance relations and breeding rate in mature male american bison. *Zeitschrift für Tierpsychologie*, 49(4):418–432, 1979.

László Lovász et al. Random walks on graphs: A survey. *Combinatorics, Paul erdos is eighty*, 2(1):1–46, 1993.

Ada Lovelace. Notes on menabrea’s “sketch of the analytical engine invented by charles babbage”, 1842.

Can Lu, Jeffrey Xu Yu, Rong-Hua Li, and Hao Wei. Exploring hierarchies in online social networks. *IEEE Transactions on Knowledge and Data Engineering*, 28(8):2086–2100, 2016.

Linyuan Lü and Weiping Liu. Information filtering via preferential diffusion. *Physical Review E*, 83(6):066119, 2011.

Linyuan Lü and Tao Zhou. Link prediction in complex networks: A survey. *Physica A: statistical mechanics and its applications*, 390(6):1150–1170, 2011.

Xuesong Lu and Stéphane Bressan. Sampling connected induced subgraphs uniformly at random. In *International Conference on Scientific and Statistical Database Management*, pages 195–212. Springer, 2012.

Flaminia L Luccio. Intruder capture in sierpinski graphs. In *FUN*, pages 249–261. Springer, 2007.

Feng Luo, James Z Wang, and Eric Promislow. Exploring local community structures in large networks. *Web Intelligence and Agent Systems: An International Journal*, 6(4):387–400, 2008.

Jianxi Luo and Christopher L Magee. Detecting evolving patterns of self-organizing networks by flow hierarchy measurement. *Complexity*, 16(6):53–61, 2011.

Athen Ma and Raúl J Mondragón. Rich-cores in networks. *PloS one*, 10(3):e0119678, 2015.

Hang Ma, TK Satish Kumar, and Sven Koenig. Multi-agent path finding with delay probabilities. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

Hao Ma, Haixuan Yang, Michael R Lyu, and Irwin King. Mining social networks using heat diffusion processes for marketing candidates selection. In *Proceedings of the 17th ACM conference on Information and knowledge management*, pages 233–242. ACM, 2008.

Jan Maas. Gradient flows of the entropy for finite markov chains. *Journal of Functional Analysis*, 261(8):2250–2292, 2011.

Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

Robert H Mac Arthur and Edward Osborne Wilson. The theory of island biogeography. Technical report, 1967.

David JC MacKay. *Information theory, inference and learning algorithms*. Cambridge university press, 2003.

James MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Berkeley symp on math statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA, 1967.

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

Matteo Magnani, Luca Rossi, and Davide Vega. Analysis of multiplex social networks with r.

Arun S Maiya and Tanya Y Berger-Wolf. Benefits of bias: Towards better characterization of network sampling. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 105–113. ACM, 2011.

Erkki Mäkinen. How to draw a hypergraph. *International Journal of Computer Mathematics*, 34(3-4):177–185, 1990.

Sabrine Mallek, Imen Boukhris, Zied Elouedi, and Eric Lefèvre. Evidential link prediction in social networks based on structural and social information. *Journal of computational science*, 30:98–107, 2019.

Fragkiskos D Malliaros and Michalis Vazirgiannis. Clustering and community detection in directed networks: A survey. *Physics Reports*, 533(4):95–142, 2013.

Ebrahim H Mamdani. Application of fuzzy algorithms for control of simple dynamic plant. In *Proceedings of the institution of electrical engineers*, volume 121, pages 1585–1588. IET, 1974.

Anqi Mao, Mehryar Mohri, and Yutao Zhong. Cross-entropy loss functions: Theoretical analysis and applications. In *International Conference on Machine Learning*, pages 23803–23828. PMLR, 2023.

Riccardo Marcaccioli and Giacomo Livan. A pólya urn approach to information filtering in complex networks. *Nature Communications*, 10(1):745, 2019.

Diego Marcheggiani and Ivan Titov. Encoding sentences with graph convolutional networks for semantic role labeling. *arXiv preprint arXiv:1703.04826*, 2017.

Massimo Marchiori and Vito Latora. Harmony in the small-world. *Physica A: Statistical Mechanics and its Applications*, 285(3-4):539–546, 2000.

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.

Peter V Marsden. Network data and measurement. *Annual review of sociology*, 16(1):435–463, 1990.

Travis Martin, Xiao Zhang, and Mark EJ Newman. Localization and centrality in networks. *Physical review E*, 90(5):052808, 2014.

Víctor Martínez, Fernando Berzal, and Juan-Carlos Cubero. A survey of link prediction in complex networks. *ACM Computing Surveys (CSUR)*, 49(4):69, 2017.

Guido Previte Massara, Tiziana Di Matteo, and Tomaso Aste. Network filtering for big data: Triangulated maximally filtered graph. *Journal of complex Networks*, 5(2):161–178, 2016.

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

Prabhaker Mateti and Narsingh Deo. On algorithms for enumerating all circuits of a graph. *SIAM Journal on Computing*, 5(1):90–99, 1976.

Michael Mathioudakis, Francesco Bonchi, Carlos Castillo, Aristides Gionis, and Antti Ukkonen. Sparsification of influence networks. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 529–537, 2011.

Ryuta Matsuno and Tsuyoshi Murata. Mell: effective embedding method for multiplex networks. In *Companion Proceedings of the Web Conference 2018*, pages 1261–1268, 2018.

Christopher L McClendon, Lan Hua, Gabriela Barreiro, and Matthew P Jacobson. Comparing conformational ensembles using the kullback–leibler divergence expansion. *Journal of chemical theory and computation*, 8(6):2115–2126, 2012.

Elizabeth Aura McClintock. When does race matter? race, sex, and dating at an elite university. *Journal of Marriage and Family*, 72(1):45–72, 2010.

Aaron F McDaid, Derek Greene, and Neil Hurley. Normalized mutual information to evaluate overlapping community finding algorithms. *arXiv preprint arXiv:1110.2515*, 2011.

Richard McElreath. *Statistical rethinking: A Bayesian course with examples in R and Stan*. Chapman and Hall/CRC, 2018.

Fintan McGee, Mohammad Ghoniem, Guy Melançon, Benoît Ot-jacques, and Bruno Pinaud. The state of the art in multilayer network visualization. In *Computer Graphics Forum*, volume 38, pages 125–149. Wiley Online Library, 2019.

Andrew McGregor and Daniel Stubbs. Sketching earth-mover distance on graph metrics. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 274–286. Springer, 2013.

Leland McInnes, John Healy, and James Melville. Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*, 2018.

Brendan D McKay et al. *Practical graph isomorphism*. Department of Computer Science, Vanderbilt University Tennessee, USA, 1981.

Miller McPherson, Lynn Smith-Lovin, and James M Cook. Birds of a feather: Homophily in social networks. *Annual review of sociology*, 27(1):415–444, 2001.

Yasir Mehmood, Nicola Barbieri, Francesco Bonchi, and Antti Ukkonen. Csi: Community-level social influence analysis. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 48–63. Springer, 2013.

Marina Meilă. Comparing clusterings—an information based distance. *Journal of multivariate analysis*, 98(5):873–895, 2007.

Isabel Meirelles. *Design for information: an introduction to the histories, theories, and best practices behind effective information visualizations*. Rockport publishers, 2013.

Victor Gabriel Lopez Mejia, Frank L Lewis, Yan Wan, Edgar N Sanchez, and Lingling Fan. Solutions for multiagent pursuit-evasion games on communication graphs: Finite-time capture and asymptotic behaviors. *IEEE Transactions on Automatic Control*, 2019.

Yelena Mejova, Amy X Zhang, Nicholas Diakopoulos, and Carlos Castillo. Controversy and sentiment in online news. *arXiv preprint arXiv:1409.8152*, 2014.

David Melamed. Community structures in bipartite networks: A dual-projection approach. *PloS one*, 9(5):e97823, 2014.

Sergey Melnik, Hector Garcia-Molina, and Erhard Rahm. Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In *Proceedings 18th International Conference on Data Engineering*, pages 117–128. IEEE, 2002.

Filippo Menczer, Santo Fortunato, and Clayton A Davis. *A First Course in Network Science*. Cambridge University Press, 2020.

Christian Merkwirth and Thomas Lengauer. Automatic generation of complementary descriptors with molecular graph networks. *Journal of chemical information and modeling*, 45(5):1159–1168, 2005.

Robert K Merton. The matthew effect in science: The reward and communication systems of science are considered. *Science*, 159(3810):56–63, 1968.

Carl D Meyer. *Matrix analysis and applied linear algebra*, volume 71. Siam, 2000.

Giovanni Micale, Alfredo Pulvirenti, Alfredo Ferro, Rosalba Giugno, and Dennis Shasha. Fast methods for finding significant motifs on labelled multi-relational networks. *Journal of Complex Networks*, 2019.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013a.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, 26:3111–3119, 2013b.

Tijana Milenković, Weng Leong Ng, Wayne Hayes, and Nataša Pržulj. Optimal network alignment with graphlet degree vectors. *Cancer informatics*, 9:CIN-S4744, 2010.

Stanley Milgram. The small world problem. *Psychology today*, 2(1):60–67, 1967.

Ana P Millán, Joaquín J Torres, and Ginestra Bianconi. Explosive higher-order kuramoto dynamics on simplicial complexes. *Physical Review Letters*, 124(21):218301, 2020.

Joel C Miller. Percolation and epidemics in random clustered networks. *Physical Review E*, 80(2):020901, 2009.

K Miller and Tina Eliassi-Rad. Continuous time group discovery in dynamic graphs. Technical report, LLNL, 2010.

Barbara J Mills, Jeffery J Clark, Matthew A Peeples, W Randall Haas, John M Roberts, J Brett Hill, Deborah L Huntley, Lewis Borck, Ronald L Breiger, Aaron Clauset, et al. Transformation of social networks in the late pre-hispanic us southwest. *Proceedings of the National Academy of Sciences*, 110(15):5785–5790, 2013.

Ron Milo, Shai Shen-Orr, Shalev Itzkovitz, Nadav Kashtan, Dmitri Chklovskii, and Uri Alon. Network motifs: simple building blocks of complex networks. *Science*, 298(5594):824–827, 2002.

Staša Milojević. Power law distributions in information science: Making the case for logarithmic binning. *Journal of the American Society for Information Science and Technology*, 61(12):2417–2425, 2010.

Byungjoon Min, Su Do Yi, Kyu-Min Lee, and K-I Goh. Network robustness of multiplex networks with interlayer degree correlations. *Physical Review E*, 89(4):042811, 2014.

Erxue Min, Runfa Chen, Yatao Bian, Tingyang Xu, Kangfei Zhao, Wenbing Huang, Peilin Zhao, Junzhou Huang, Sophia Ananiadou, and Yu Rong. Transformer for graphs: An overview from architecture perspective. *arXiv preprint arXiv:2202.08455*, 2022.

Giorgia Minello, Luca Rossi, and Andrea Torsello. On the von neumann entropy of graphs. *Journal of Complex Networks*, 7(4):491–514, 2019.

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

Michael Molloy and Bruce Reed. A critical point for random graphs with a given degree sequence. *Random structures & algorithms*, 6(2-3):161–180, 1995.

Michael Molloy and Bruce Reed. The size of the giant component of a random graph with a given degree sequence. *Combinatorics, probability and computing*, 7(3):295–305, 1998.

Raul J Mondragon, Jacopo Iacovacci, and Ginestra Bianconi. Multi-link communities of multiplex networks. *PloS one*, 13(3):e0193821, 2018.

Enys Mones. Hierarchy in directed random networks. *Physical Review E*, 87(2):022817, 2013.

Enys Mones, Lilla Vicsek, and Tamás Vicsek. Hierarchy measure for complex networks. *PloS one*, 7(3):e33799, 2012.

Gaspard Monge. Mémoire sur la théorie des déblais et des remblais. *Histoire de l'Académie Royale des Sciences de Paris*, 1781.

Bjarke Mønsted and Sune Lehmann. Characterizing polarization in online vaccine discourse—a large-scale study. *PloS one*, 17(2): e0263746, 2022.

Bjarke Mønsted, Piotr Sapieżyński, Emilio Ferrara, and Sune Lehmann. Evidence of complex contagion of information in social media: An experiment using twitter bots. *PloS one*, 12(9), 2017.

Federico Monti, Fabrizio Frasca, Davide Eynard, Damon Mannion, and Michael M Bronstein. Fake news detection on social media using geometric deep learning. *arXiv preprint arXiv:1902.06673*, 2019.

Todd K Moon. The expectation-maximization algorithm. *IEEE Signal processing magazine*, 13(6):47–60, 1996.

Edward F Moore. The shortest path through a maze. In *Proc. Int. Symp. Switching Theory*, 1959, pages 285–292, 1959.

Eliakim H Moore. On the reciprocal of the general algebraic matrix. *Bulletin of the american mathematical society*, 26:294–295, 1920.

Behnaz Moradabadi and Mohammad Reza Meybodi. Link prediction in fuzzy social networks using distributed learning automata. *Applied Intelligence*, 47:837–849, 2017.

Jacob L Moreno and Helen H Jennings. Statistics of social configurations. *Sociometry*, pages 342–374, 1938.

Matteo Morini, Patrick Flandrin, Eric Fleury, Tommaso Venturini, and Pablo Jensen. Revealing evolutions in dynamical networks. *arXiv preprint arXiv:1707.02114*, 2017.

Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4602–4609, 2019.

Fred Morstatter, Jürgen Pfeffer, Huan Liu, and Kathleen M Carley. Is the sample good enough? comparing data from twitter’s streaming api with twitter’s firehose. In *Seventh international AAAI conference on weblogs and social media*, 2013.

Fred Morstatter, Jürgen Pfeffer, and Huan Liu. When is it biased?: assessing the representativeness of twitter’s streaming api. In *Proceedings of the 23rd international conference on world wide web*, pages 555–556. ACM, 2014.

Peter J Mucha, Thomas Richardson, Kevin Macon, Mason A Porter, and Jukka-Pekka Onnela. Community structure in time-dependent, multiscale, and multiplex networks. *science*, 328(5980):876–878, 2010.

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.

Abubakr Muhammad and Magnus Egerstedt. Control using higher order laplacians in network topologies. In *Proc. of 17th International Symposium on Mathematical Theory of Networks and Systems*, pages 1024–1038. Citeseer, 2006.

Tamara Munzner. *Visualization analysis and design*. AK Peters/CRC Press, 2014.

Ryan Murphy, Balasubramaniam Srinivasan, Vinayak Rao, and Bruno Ribeiro. Relational pooling for graph representations. In *International Conference on Machine Learning*, pages 4663–4673. PMLR, 2019.

Ryan L Murphy, Balasubramaniam Srinivasan, Vinayak Rao, and Bruno Ribeiro. Janossy pooling: Learning deep permutation-invariant functions for variable-size inputs. *arXiv preprint arXiv:1811.01900*, 2018.

Mirco Musolesi and Cecilia Mascolo. A community based mobility model for ad hoc network research. In *MOBIHOC*, pages 31–38, 2006.

Richard Myers, RC Wilson, and Edwin R Hancock. Bayesian graph edit distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(6):628–635, 2000.

Marc Najork, Sreenivas Gollapudi, and Rina Panigrahy. Less is more: sampling the neighborhood graph makes salsa better and faster. In *Proceedings of the Second ACM International Conference on Web Search and Data Mining*, pages 242–251, 2009.

Sunil K Narang, Akshay Gadde, Eduard Sanou, and Antonio Ortega. Localized iterative methods for interpolation in graph structured data. In *Global Conference on Signal and Information Processing (GlobalSIP), 2013 IEEE*, pages 491–494. IEEE, 2013.

Anand Narasimhamurthy, Derek Greene, Neil Hurley, and Pádraig Cunningham. Community finding in large social networks through problem decomposition. In *Proc. 19th Irish Conference on Artificial Intelligence and Cognitive Science, AICS*, volume 8, 2008.

Arvind Narayanan and Vitaly Shmatikov. De-anonymizing social networks. In *2009 30th IEEE symposium on security and privacy*, pages 173–187. IEEE, 2009.

Saket Navlakha, Rajeev Rastogi, and Nisheeth Shrivastava. Graph summarization with bounded error. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 419–432, 2008.

Maziar Nekovee, Yamir Moreno, Ginestra Bianconi, and Matteo Marsili. Theory of rumour spreading in complex social networks. *Physica A: Statistical Mechanics and its Applications*, 374(1):457–470, 2007.

Azadeh Nematzadeh, Emilio Ferrara, Alessandro Flammini, and Yong-Yeol Ahn. Optimal network modularity for information diffusion. *Physical review letters*, 113(8):088701, 2014.

Tamás Nepusz, Andrea Petróczi, László Négyessy, and Fülöp Bazsó. Fuzzy communities and the concept of bridgeness in complex networks. *Physical Review E*, 77(1):016107, 2008.

Erich Neuwirth and R Color Brewer. Colorbrewer palettes. *R package version*, pages 1–1, 2014.

Mark Newman. *Networks*. Oxford university press, 2018a.

Mark EJ Newman. Clustering and preferential attachment in growing networks. *Physical review E*, 64(2):025102, 2001a.

Mark EJ Newman. Scientific collaboration networks. i. network construction and fundamental results. *Physical review E*, 64(1):016131, 2001b.

Mark EJ Newman. Scientific collaboration networks. ii. shortest paths, weighted networks, and centrality. *Physical review E*, 64(1):016132, 2001c.

Mark EJ Newman. Assortative mixing in networks. *Physical review letters*, 89(20):208701, 2002.

Mark EJ Newman. Mixing patterns in networks. *Physical Review E*, 67(2):026126, 2003a.

Mark EJ Newman. The structure and function of complex networks. *SIAM review*, 45(2):167–256, 2003b.

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

- Mark EJ Newman. Detecting community structure in networks. *The European Physical Journal B*, 38(2):321–330, 2004b.
- Mark EJ Newman. Fast algorithm for detecting community structure in networks. *Physical review E*, 69(6):066133, 2004c.
- Mark EJ Newman. A measure of betweenness centrality based on random walks. *Social networks*, 27(1):39–54, 2005a.
- Mark EJ Newman. Power laws, pareto distributions and zipf’s law. *Contemporary physics*, 46(5):323–351, 2005b.
- Mark EJ Newman. Finding community structure in networks using the eigenvectors of matrices. *Physical review E*, 74(3):036104, 2006a.
- Mark EJ Newman. Modularity and community structure in networks. *Proceedings of the national academy of sciences*, 103(23):8577–8582, 2006b.
- Mark EJ Newman. Random graphs with clustering. *Physical review letters*, 103(5):058701, 2009.
- Mark EJ Newman. Equivalence between modularity optimization and maximum likelihood methods for community detection. *Physical Review E*, 94(5):052315, 2016a.
- Mark EJ Newman. Mathematics of networks. *The new Palgrave dictionary of economics*, pages 1–8, 2016b.
- Mark EJ Newman. Network structure from rich but noisy data. *Nature Physics*, 14(6):542–545, 2018b.
- Mark EJ Newman and Michelle Girvan. Finding and evaluating community structure in networks. *Physical review E*, 69(2):026113, 2004.
- Mark EJ Newman and Duncan J Watts. Renormalization group analysis of the small-world network model. *Physics Letters A*, 263(4-6):341–346, 1999.
- Mark EJ Newman, Steven H Strogatz, and Duncan J Watts. Random graphs with arbitrary degree distributions and their applications. *Physical review E*, 64(2):026118, 2001.
- Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. A three-way model for collective learning on multi-relational data. In *Icml*, volume 11, pages 809–816, 2011.
- Maximilian Nickel, Kevin Murphy, Volker Tresp, and Evgeniy Gabrilovich. A review of relational machine learning for knowledge graphs. *Proceedings of the IEEE*, 104(1):11–33, 2015.

Vincenzo Nicosia, Giuseppe Mangioni, Vincenza Carchiolo, and Michele Malgeri. Extending the definition of modularity to directed graphs with overlapping communities. *Journal of Statistical Mechanics: Theory and Experiment*, 2009(03):Po3024, 2009.

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.

Mathias Niepert, Mathias Ahmed, and Konstantin Kutzkov. Learning convolutional neural networks for graphs. In *ICML*, pages 2014–2023, 2016.

Siegfried Nijssen and Joost N Kok. The gaston tool for frequent subgraph mining. *Electronic Notes in Theoretical Computer Science*, 127(1):77–87, 2005.

Shirin Nilizadeh, Apu Kapadia, and Yong-Yeol Ahn. Community-enhanced de-anonymization of online social networks. In *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security*, pages 537–548, 2014.

Jae Dong Noh, Hyeong-Chai Jeong, Yong-Yeol Ahn, and Hawoong Jeong. Growing network model for community with group structure. *Physical Review E*, 71(3):036131, 2005.

Mary L Northway. A method for depicting social relationships obtained by sociometric testing. *Sociometry*, pages 144–150, 1940.

Anastasios Noulas, Salvatore Scellato, Renaud Lambiotte, Massimiliano Pontil, and Cecilia Mascolo. A tale of many cities: universal patterns in human urban mobility. *PloS one*, 7(5), 2012.

Esko Nuutila and Eljas Soisalon-Soininen. On finding the strongly connected components in a directed graph. *Inf. Process. Lett.*, 49(1):9–14, 1994.

Chigozie Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. Activation functions: Comparison of trends in practice and research for deep learning. *arXiv preprint arXiv:1811.03378*, 2018.

Abdel R Omran. The epidemiologic transition: a theory of the epidemiology of population change. *The Milbank Quarterly*, 83(4):731–757, 2005.

Jukka-Pekka Onnela, Jari Saramäki, János Kertész, and Kimmo Kaski. Intensity and coherence of motifs in weighted complex networks. *Physical Review E*, 71(6):065103, 2005.

Jukka-Pekka Onnela, Samuel Arbesman, Marta C González, Albert-László Barabási, and Nicholas A Christakis. Geographic constraints on social network groups. *PLoS one*, 6(4):e16939, 2011.

Tore Opsahl, Filip Agneessens, and John Skvoretz. Node centrality in weighted networks: Generalizing degree and shortest paths. *Social networks*, 32(3):245–251, 2010.

Günce Keziban Orman and Vincent Labatut. A comparison of community detection algorithms on artificial networks. In *DS*, pages 242–256. Springer, 2009.

Mingdong Ou, Peng Cui, Jian Pei, Ziwei Zhang, and Wenwu Zhu. Asymmetric transitivity preserving graph embedding. In *SIGKDD*, pages 1105–1114, 2016.

Norman H Packard. Adaptation toward the edge of chaos. *Dynamic patterns in complex systems*, 212:293–301, 1988.

John F Padgett and Christopher K Ansell. Robust action and the rise of the medici, 1400-1434. *American journal of sociology*, 98(6):1259–1319, 1993.

Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999.

Gergely Palla, Albert-László Barabási, and Tamás Vicsek. Quantifying social group evolution. *Nature*, 446(7136):664, 2007.

Raj Kumar Pan and Jari Saramäki. Path lengths, correlations, and centrality in temporal networks. *Physical Review E*, 84(1):016105, 2011.

Shirui Pan, Jia Wu, Xingquan Zhu, Chengqi Zhang, and Yang Wang. Tri-party deep network representation. *Network*, 11(9):12, 2016.

Panagiotis Papadimitriou, Ali Dasdan, and Hector Garcia-Molina. Web graph similarity for anomaly detection. *Journal of Internet Services and Applications*, 1(1):19–30, 2010.

Spiros Papadimitriou, Jimeng Sun, Christos Faloutsos, and S Yu Philip. Hierarchical, parameter-free community discovery. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 170–187. Springer, 2008.

Fragkiskos Papadopoulos, Maksim Kitsak, M Ángeles Serrano, Marián Boguná, and Dmitri Krioukov. Popularity versus similarity in growing networks. *Nature*, 489(7417):537–540, 2012.

Symeon Papadopoulos, Andre Skusa, Athena Vakali, Yiannis Komatsiaris, and Nadine Wagner. Bridge bounding: A local approach for efficient community discovery in complex networks. *arXiv preprint arXiv:0902.0871*, 2009.

Manos Papagelis, Gautam Das, and Nick Koudas. Sampling online social networks. *IEEE Transactions on knowledge and data engineering*, 25(3):662–676, 2013.

Odysseas Papapetrou, Ekaterini Ioannou, and Dimitrios Skoutas. Efficient discovery of frequent subgraph patterns in uncertain graph databases. In *EDBT*, pages 355–366, 2011.

Luca Pappalardo, Giulio Rossetti, and Dino Pedreschi. "how well do we know each other?" detecting tie strength in multidimensional social networks. In *2012 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, pages 1040–1045. IEEE, 2012.

Luca Pappalardo, Filippo Simini, Salvatore Rinzivillo, Dino Pedreschi, Fosca Giannotti, and Albert-László Barabási. Returners and explorers dichotomy in human mobility. *Nature communications*, 6:8166, 2015.

Ashwin Paranjape, Austin R Benson, and Jure Leskovec. Motifs in temporal networks. In *WSDM*, pages 601–610. ACM, 2017.

Panos Pachas, Nikolaos Papailiou, Dimitris Papadias, and Francesco Bonchi. Uncertain graph sparsification. *IEEE Transactions on Knowledge and Data Engineering*, 30(12):2435–2449, 2018.

Vilfredo Pareto. *Manuale di economia politica con una introduzione alla scienza sociale*, volume 13. Società editrice libraria, 1919.

Eli Pariser. *The filter bubble: What the Internet is hiding from you*. penguin UK, 2011.

Namyong Park, Andrey Kan, Xin Luna Dong, Tong Zhao, and Christos Faloutsos. Estimating node importance in knowledge graphs using graph neural networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 596–606, 2019.

Roni Parshani, Celine Rozenblat, Daniele Ietri, Cesar Ducruet, and Shlomo Havlin. Inter-similarity between coupled networks. *EPL (Europhysics Letters)*, 92(6):68002, 2011.

Torrence D Parsons. Pursuit-evasion in a graph. In *Theory and applications of graphs*, pages 426–441. Springer, 1978.

Srinivasan Parthasarathy, Yiye Ruan, and Venu Satuluri. Community discovery in social networks: Applications, methods and emerging trends. In *Social network data analytics*, pages 79–113. Springer, 2011.

Emanuel Parzen. On estimation of a probability density function and mode. *The annals of mathematical statistics*, 33(3):1065–1076, 1962.

Filippo Passerini and Simone Severini. The von neumann entropy of networks. *arXiv preprint arXiv:0812.2597*, 2008.

Romualdo Pastor-Satorras and Alessandro Vespignani. Epidemic dynamics and endemic states in complex networks. *Physical Review E*, 63(6):066117, 2001a.

Romualdo Pastor-Satorras and Alessandro Vespignani. Epidemic spreading in scale-free networks. *Physical review letters*, 86(14):3200, 2001b.

Romualdo Pastor-Satorras, Alexei Vázquez, and Alessandro Vespignani. Dynamical and correlation properties of the internet. *Physical review letters*, 87(25):258701, 2001.

Romualdo Pastor-Satorras, Claudio Castellano, Piet Van Mieghem, and Alessandro Vespignani. Epidemic processes in complex networks. *Reviews of modern physics*, 87(3):925, 2015.

Rob Patro and Carl Kingsford. Global network alignment using multiscale spectral signatures. *Bioinformatics*, 28(23):3105–3114, 2012.

Bruce D Patterson. The principle of nested subsets and its implications for biological conservation. *Conservation Biology*, 1(4):323–334, 1987.

Mateusz Pawlik and Nikolaus Augsten. Rted: a robust algorithm for the tree edit distance. *arXiv preprint arXiv:1201.0230*, 2011.

Claudia Payrató-Borras, Laura Hernández, and Yamir Moreno. Breaking the spell of nestedness: The entropic origin of nestedness in mutualistic systems. *Physical Review X*, 9(3):031024, 2019.

Judea Pearl. Reasoning with belief functions: An analysis of compatibility. *International Journal of Approximate Reasoning*, 4(5-6):363–389, 1990.

Judea Pearl and Dana Mackenzie. *The book of why: the new science of cause and effect*. Basic Books, 2018.

Leto Peel and Aaron Clauset. Detecting change points in the large-scale structure of evolving networks. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

Leto Peel, Daniel B Larremore, and Aaron Clauset. The ground truth about metadata and community detection in networks. *Science advances*, 3(5):e1602548, 2017.

Jian Pei, Jiawei Han, Runying Mao, et al. Closet: An efficient algorithm for mining frequent closed itemsets. In *ACM SIGMOD workshop on research issues in data mining and knowledge discovery*, volume 4, pages 21–30, 2000.

Tiago P Peixoto. Efficient monte carlo and greedy heuristic for the inference of stochastic block models. *Physical Review E*, 89(1):012804, 2014a.

Tiago P Peixoto. The graph-tool python library. *figshare*, 2014b.

Tiago P Peixoto. Hierarchical block structures and high-resolution model selection in large networks. *Physical Review X*, 4(1):011047, 2014c.

Tiago P Peixoto. Inferring the mesoscale structure of layered, edge-valued, and time-varying networks. *Physical Review E*, 92(4):042807, 2015.

Tiago P Peixoto. Nonparametric bayesian inference of the micro-canonical stochastic block model. *Physical Review E*, 95(1):012317, 2017.

Tiago P Peixoto. Reconstructing networks with unknown and heterogeneous errors. *Physical Review X*, 8(4):041011, 2018.

Tiago P Peixoto. Ordered community detection in directed networks. *Physical Review E*, 106(2):024305, 2022.

Tiago P Peixoto. *Descriptive vs. inferential community detection in networks: Pitfalls, myths and half-truths*. Cambridge University Press, 2023.

Ofir Pele and Michael Werman. A linear time histogram metric for improved sift matching. In *European conference on computer vision*, pages 495–508. Springer, 2008.

Ofir Pele and Michael Werman. Fast and robust earth mover’s distances. In *Computer vision, 2009 IEEE 12th international conference on*, pages 460–467. IEEE, 2009.

Shmuel Peleg, Michael Werman, and Hillel Rom. A unified approach to the change of resolution: Space and gray-level. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(7):739–742, 1989.

Diego Pennacchioli, Giulio Rossetti, Luca Pappalardo, Dino Pedreschi, Fosca Giannotti, and Michele Coscia. The three dimensions of social prominence. In *International Conference on Social Informatics*, pages 319–332. Springer, 2013.

Diego Pennacchioli, Michele Coscia, Salvatore Rinzivillo, Fosca Giannotti, and Dino Pedreschi. The retail market as a complex system. *EPJ Data Science*, 3(1):33, 2014.

Mathew Penrose et al. *Random geometric graphs*, volume 5. Oxford university press, 2003.

Roger Penrose. A generalized inverse for matrices. In *Mathematical proceedings of the Cambridge philosophical society*, volume 51, pages 406–413. Cambridge University Press, 1955.

Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710. ACM, 2014.

Trang Pham, Truyen Tran, Dinh Phung, and Svetha Venkatesh. Column networks for collective classification. In *Proceedings of the AAAI conference on artificial intelligence*, volume 31, 2017.

Thomas Piketty. Capital in the 21st century. 2014.

Ali Pinar, C Seshadhri, and Vaidyanathan Vishal. Escape: Efficiently counting all 5-vertex subgraphs. In *WWW*, pages 1431–1440, 2017.

Clara Pizzuti. Ga-net: A genetic algorithm for community detection in social networks. In *International conference on parallel problem solving from nature*, pages 1081–1090. Springer, 2008.

Clara Pizzuti. A multiobjective genetic algorithm to find communities in complex networks. *IEEE Transactions on Evolutionary Computation*, 16(3):418–430, 2012.

Henri Poincaré. *The foundations of science*. 1913.

Pascal Pons and Matthieu Latapy. Computing communities in large networks using random walks. *J. Graph Algorithms Appl.*, 10(2):191–218, 2006.

Mason A Porter. What is... a multilayer network. *Notices of the AMS*, 65(11):1419–1423, 2018.

Mason A Porter, Jukka-Pekka Onnela, and Peter J Mucha. Communities in networks. *Notices of the AMS*, 56(9):1082–1097, 2009.

Márton Pósfai, Yang-Yu Liu, Jean-Jacques Slotine, and Albert-László Barabási. Effect of correlations on network controllability. *Scientific reports*, 3:1067, 2013.

Michalis Potamias, Francesco Bonchi, Aristides Gionis, and George Kollios. K-nearest neighbors in uncertain graphs. *Proceedings of the VLDB Endowment*, 3(1-2):997–1008, 2010.

Eleanor A Power. Discerning devotion: Testing the signaling theory of religion. *Evolution and Human Behavior*, 38(1):82–91, 2017.

Jonathan D Power, Alexander L Cohen, Steven M Nelson, Gagan S Wig, Kelly Anne Barnes, Jessica A Church, Alecia C Vogel, Timothy O Laumann, Fran M Miezin, Bradley L Schlaggar, et al. Functional network organization of the human brain. *Neuron*, 72(4):665–678, 2011.

David Martin Powers. Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation. 2011.

William H Press, Saul A Teukolsky, William T Vetterling, and Brian P Flannery. *Numerical recipes 3rd edition: The art of scientific computing*. Cambridge university press, 2007.

Derek de Solla Price. A general theory of bibliometric and other cumulative advantage processes. *Journal of the American society for Information science*, 27(5):292–306, 1976.

Robert Clay Prim. Shortest connection networks and some generalizations. *Bell system technical journal*, 36(6):1389–1401, 1957.

Manisha Pujari and Rushed Kanawati. Link prediction in multiplex networks. *NHM*, 10(1):17–35, 2015.

John Punin and Mukkai Krishnamoorthy. Xgmml (extensible graph markup and modeling language), 2001.

Manish Purohit, B Aditya Prakash, Chanhyun Kang, Yao Zhang, and VS Subrahmanian. Fast influence-based coarsening for large networks. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1296–1305, 2014.

Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *CVPR*, pages 652–660, 2017.

Walter Quattrociocchi, Antonio Scala, and Cass R Sunstein. Echo chambers on facebook. Available at SSRN 2795110, 2016.

Filippo Radicchi, Claudio Castellano, Federico Cecconi, Vittorio Loreto, and Domenico Parisi. Defining and identifying communities in networks. *Proceedings of the National Academy of Sciences*, 101(9):2658–2663, 2004.

Filippo Radicchi, José J Ramasco, and Santo Fortunato. Information filtering in complex weighted networks. *Physical Review E*, 83(4):046101, 2011.

Usha Nandini Raghavan, Réka Albert, and Soundar Kumara. Near linear time algorithm to detect community structures in large-scale networks. *Physical review E*, 76(3):036106, 2007.

Kelly Raley, Megan Sweeney, and Danielle Wondra. The growing racial and ethnic divide in us marriage patterns. *Future of children*, 25(2):89, 2015.

Prajit Ramachandran, Barret Zoph, and Quoc V Le. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017.

Rajeev Raman. Recent results on the single-source shortest paths problem. *ACM SIGACT News*, 28(2):81–87, 1997.

William M Rand. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical association*, 66(336):846–850, 1971.

Amir H Rasti, Mojtaba Torkjazi, Reza Rejaie, D Stutzbach, N Duffield, and W Willinger. Evaluating sampling techniques for large dynamic graphs. *Univ. Oregon, Tech. Rep. CIS-TR-08*, 1, 2008.

Amir Hassan Rasti, Mojtaba Torkjazi, Reza Rejaie, Nick Duffield, Walter Willinger, and Daniel Stutzbach. Respondent-driven sampling for characterizing unstructured overlays. In *IEEE INFOCOM 2009*, pages 2701–2705. IEEE, 2009.

Erzsébet Ravasz and Albert-László Barabási. Hierarchical organization in complex networks. *Physical review E*, 67(2):026112, 2003.

Erzsébet Ravasz, Anna Lisa Somera, Dale A Mongru, Zoltán N Oltvai, and A-L Barabási. Hierarchical organization of modularity in metabolic networks. *science*, 297(5586):1551–1555, 2002.

John W Raymond, Eleanor J Gardiner, and Peter Willett. Rascal: Calculation of graph similarity using maximum common edge subgraphs. *The Computer Journal*, 45(6):631–644, 2002.

Jörg Reichardt and Stefan Bornholdt. Statistical mechanics of community detection. *Physical Review E*, 74(1):016110, 2006.

Fergal Reid, Aaron McDaid, and Neil Hurley. Percolation computation in complex networks. In *2012 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, pages 274–281. IEEE, 2012.

Davi de Castro Reis, Paulo Braz Golher, Altigran Soares Silva, and AlbertoF Laender. Automatic web news extraction using tree edit distance. In *Proceedings of the 13th international conference on World Wide Web*, pages 502–511, 2004.

Saulo DS Reis, Yanqing Hu, Andrés Babino, José S Andrade Jr, Santiago Canals, Mariano Sigman, and Hernán A Makse. Avoiding catastrophic failure in correlated networks of networks. *Nature Physics*, 10(10):762, 2014.

CJ Rhodes and P Jones. Inferring missing links in partially observed social networks. In *OR, Defence and Security*, pages 256–271. Springer, 2015.

Bruno Ribeiro and Don Towsley. Estimating and sampling graphs with multidimensional random walks. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, pages 390–403, 2010.

Leonardo FR Ribeiro, Pedro HP Saverese, and Daniel R Figueiredo. struc2vec: Learning node representations from structural identity. In *SIGKDD*, pages 385–394, 2017.

John A Rice. *Mathematical statistics and data analysis*. Cengage Learning, 2006.

Kaspar Riesen and Horst Bunke. Approximate graph edit distance computation by means of bipartite graph matching. *Image and Vision computing*, 27(7):950–959, 2009.

Matteo Riondato, David García-Soriano, and Francesco Bonchi. Graph summarization with quality guarantees. *Data mining and knowledge discovery*, 31(2):314–349, 2017.

Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

Elina Robeva and Anna Seigal. Singular vectors of orthogonally decomposable tensors. *Linear and Multilinear Algebra*, 65(12):2457–2471, 2017.

Garry Robins, Pip Pattison, Yuval Kalish, and Dean Lusher. An introduction to exponential random graph (p^*) models for social networks. *Social networks*, 29(2):173–191, 2007.

- Yannick Rochat. Closeness centrality extended to unconnected graphs: The harmonic centrality index. Technical report, 2009.
- M Puck Rombach, Mason A Porter, James H Fowler, and Peter J Mucha. Core-periphery structure in networks. *SIAM Journal on Applied mathematics*, 74(1):167–190, 2014.
- Murray Rosenblatt. Remarks on some nonparametric estimates of a density function. *The Annals of Mathematical Statistics*, pages 832–837, 1956.
- Giulio Rossetti and Rémy Cazabet. Community discovery in dynamic networks: a survey. *ACM Computing Surveys (CSUR)*, 51(2):35, 2018.
- Giulio Rossetti, Michele Berlingario, and Fosca Giannotti. Scalable link prediction on multidimensional networks. In *2011 IEEE 11th International Conference on Data Mining Workshops*, pages 979–986. IEEE, 2011.
- Giulio Rossetti, Luca Pappalardo, Dino Pedreschi, and Fosca Giannotti. Tiles: an online algorithm for community discovery in dynamic social networks. *Machine Learning*, 106(8):1213–1241, 2017.
- Giulio Rossetti, Letizia Milli, Salvatore Rinzivillo, Alina Sîrbu, Dino Pedreschi, and Fosca Giannotti. Ndlib: a python library to model and analyze diffusion processes over complex networks. *International Journal of Data Science and Analytics*, 5(1):61–79, 2018.
- Giulio Rossetti, Letizia Milli, and Rémy Cazabet. Cdlib: a python library to extract, compare and evaluate communities from complex networks. *Applied Network Science*, 4(1):52, 2019.
- Martin Rosvall and Carl T Bergstrom. Maps of random walks on complex networks reveal community structure. *PNAS*, 105(4):1118–1123, 2008.
- Martin Rosvall and Carl T Bergstrom. Multilevel compression of random walks on networks reveals hierarchical organization in large integrated systems. *PloS one*, 6(4):e18209, 2011.
- Martin Rosvall, Alcides V Esquivel, Andrea Lancichinetti, Jevin D West, and Renaud Lambiotte. Memory in network flows and its effects on spreading dynamics and community detection. *Nature communications*, 5:4630, 2014.
- Sam T Roweis and Lawrence K Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326, 2000.

A. Roxana Pamfil, Sam D. Howison, and Mason A. Porter. Edge correlations in multilayer networks. *arXiv preprint arXiv:1908.03875*, 2019.

Bernard Roy. Transitivité et connexité. *Comptes Rendus Hebdomadaires Des Séances De L Académie Des Sciences*, 249(2):216–218, 1959.

Yossi Rubner, Carlo Tomasi, and Leonidas J Guibas. The earth mover’s distance as a metric for image retrieval. *International journal of computer vision*, 40(2):99–121, 2000.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

Neha Runwal, Richard M Low, and Mark Stamp. Opcode graph similarity and metamorphic detection. *Journal in computer virology*, 8(1-2):37–52, 2012.

Kerstin Sailer and Ian McCulloh. Social networks and spatial configuration—how office layouts drive social interaction. *Social networks*, 34(1):47–58, 2012.

Marta Sales-Pardo, Roger Guimera, André A Moreira, and Luís A Nunes Amaral. Extracting the hierarchical organization of complex systems. *Proceedings of the National Academy of Sciences*, 104(39):15224–15229, 2007.

Matthew J Salganik and Douglas D Heckathorn. Sampling and estimation in hidden populations using respondent-driven sampling. *Sociological methodology*, 34(1):193–240, 2004.

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

Gerard Salton. Automatic text processing: The transformation, analysis, and retrieval of. *Reading: Addison-Wesley*, 169, 1989.

Aliaksei Sandryhaila and Jose MF Moura. Discrete signal processing on graphs: Frequency analysis. *IEEE Trans. Signal Processing*, 62(12):3042–3054, 2014.

Fabio Saracco, Mika J Straka, Riccardo Di Clemente, Andrea Gabrielli, Guido Caldarelli, and Tiziano Squartini. Inferring monopartite projections of bipartite networks: an entropy-based approach. *New Journal of Physics*, 19(5):053022, 2017.

Jari Saramäki, Mikko Kivelä, Jukka-Pekka Onnela, Kimmo Kaski, and Janos Kertesz. Generalizations of the clustering coefficient to weighted complex networks. *Physical Review E*, 75(2):027105, 2007.

Vikram Saraph and Tijana Milenković. Magna: maximizing accuracy in global network alignment. *Bioinformatics*, 30(20):2931–2940, 2014.

Venu Satuluri and Srinivasan Parthasarathy. Scalable graph clustering using stochastic flows: applications to community discovery. In *SIGKDD conference*, pages 737–746. ACM, 2009.

Venu Satuluri, Srinivasan Parthasarathy, and Yiye Ruan. Local graph sparsification for scalable clustering. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pages 721–732, 2011.

JW Scannell, GACP Burns, CC Hilgetag, MA O’Neil, and Malcolm P Young. The connectional organization of the cortico-thalamic system of the cat. *Cerebral Cortex*, 9(3):277–299, 1999.

Franco Scarselli, Sweah Liang Yong, Marco Gori, Markus Hagenbuchner, Ah Chung Tsoi, and Marco Maggini. Graph neural networks for ranking web pages. In *The 2005 IEEE/WIC/ACM International Conference on Web Intelligence (WI’05)*, pages 666–672. IEEE, 2005.

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2008.

Salvatore Scellato, Anastasios Noulas, Renaud Lambiotte, and Cecilia Mascolo. Socio-spatial properties of online location-based social networks. In *ICWSM*, 2011.

Satu Elisa Schaeffer. Graph clustering. *Computer science review*, 1(1):27–64, 2007.

Michael T Schaub, Jean-Charles Delvenne, Sophia N Yaliraki, and Mauricio Barahona. Markov dynamics as a zooming lens for multiscale community detection: non clique-like communities and the field-of-view limit. *PloS one*, 7(2):e32210, 2012a.

Michael T Schaub, Renaud Lambiotte, and Mauricio Barahona. Encoding dynamics for multiscale community detection: Markov time sweeping for the map equation. *Physical Review E*, 86(2):026112, 2012b.

Michael T Schaub, Jean-Charles Delvenne, Renaud Lambiotte, and Mauricio Barahona. Structured networks and coarse-grained

descriptions: A dynamical perspective. *Advances in Network Clustering and Blockmodeling*, pages 333–361, 2019.

Martin W Schein and Milton H Fohrman. Social dominance relationships in a herd of dairy cattle. *The British Journal of Animal Behaviour*, 3(2):45–55, 1955.

Thomas C Schelling. Hockey helmets, concealed weapons, and daylight saving: A study of binary choices with externalities. *Journal of Conflict resolution*, 17(3):381–428, 1973.

Maximilian Schich. Revealing matrices. 2010.

Maximilian Schich. Cultural analysis situs. 2019.

Maximilian Schich, Chaoming Song, Yong-Yeol Ahn, Alexander Mirsky, Mauro Martino, Albert-László Barabási, and Dirk Helbing. A network framework of cultural history. *science*, 345(6196):558–562, 2014.

Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *European Semantic Web Conference*, pages 593–607. Springer, 2018.

Philipp Schuetz and Amedeo Caflisch. Efficient modularity optimization by multistep greedy algorithm and vertex mover refinement. *Physical Review E*, 77(4):046112, 2008.

Christoph Schulz, Arlind Nocaj, Jochen Goertler, Oliver Deussen, Ulrik Brandes, and Daniel Weiskopf. Probabilistic graph layout for uncertain network visualization. *IEEE transactions on visualization and computer graphics*, 23(1):531–540, 2016.

John R Seeley. The net of reciprocal influence. a problem in treating sociometric data. *Canadian Journal of Experimental Psychology*, 3:234, 1949.

Stephen B Seidman. Network structure and minimum degree. *Social networks*, 5(3):269–287, 1983.

Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Gallagher, and Tina Eliassi-Rad. Collective classification in network data. *AI magazine*, 29(3):93–93, 2008.

Kari Sentz and Scott Ferson. Combination of evidence in dempster-shafer theory. 2002.

M Angeles Serrano, Dmitri Krioukov, and Marián Boguná. Self-similarity of complex networks and hidden metric spaces. *Physical review letters*, 100(7):078701, 2008.

M Ángeles Serrano, Marián Boguná, and Alessandro Vespignani. Extracting the multiscale backbone of complex weighted networks. *Proceedings of the national academy of sciences*, 106(16):6483–6488, 2009.

Glenn Shafer. *A mathematical theory of evidence*, volume 42. Princeton university press, 1976.

Neil Shah, Danai Koutra, Tianmin Zou, Brian Gallagher, and Christos Faloutsos. Timecrunch: Interpretable dynamic graph summarization. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1055–1064, 2015.

Haichuan Shang, Xuemin Lin, Ying Zhang, Jeffrey Xu Yu, and Wei Wang. Connected substructure similarity search. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 903–914, 2010.

Claude Elwood Shannon. A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423, 1948.

Paul Shannon, Andrew Markiel, Owen Ozier, Nitin S Baliga, Jonathan T Wang, Daniel Ramage, Nada Amin, Benno Schwikowski, and Trey Ideker. Cytoscape: a software environment for integrated models of biomolecular interaction networks. *Genome research*, 13(11):2498–2504, 2003.

Chengcheng Shao, Giovanni Luca Ciampaglia, Onur Varol, Alessandro Flammini, and Filippo Menczer. The spread of fake news by social bots. *arXiv*, pages 96–104, 2017.

Jia Shao, Sergey V Buldyrev, Shlomo Havlin, and H Eugene Stanley. Cascade of failures in coupled network systems with multiple support-dependence relations. *Phys. Rev. E*, 83:036116, Mar 2011. doi: 10.1103/PhysRevE.83.036116.

Hua-Wei Shen, Xue-Qi Cheng, and Jia-Feng Guo. Quantifying and identifying the overlapping community structure in networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2009(07):P07042, 2009.

Zeqian Shen, Kwan-Liu Ma, and Tina Eliassi-Rad. Visual analysis of large heterogeneous social networks by semantic and structural abstraction. *IEEE transactions on visualization and computer graphics*, 12(6):1427–1439, 2006.

Shai S Shen-Orr, Ron Milo, Shmoolik Mangan, and Uri Alon. Network motifs in the transcriptional regulation network of escherichia coli. *Nature genetics*, 31(1):64, 2002.

- Yuhong Sheng and Xuehui Mei. Uncertain random shortest path problem. *Soft Computing*, 24:2431–2440, 2020.
- Nino Shervashidze and Karsten Borgwardt. Fast subtree kernels on graphs. *Advances in neural information processing systems*, 22, 2009.
- David J Sheskin. *Handbook of parametric and nonparametric statistical procedures*. Chapman and hall/CRC, 2003.
- Guodong Shi, Claudio Altafini, and John S Baras. Dynamics over signed networks. *SIAM Review*, 61(2):229–257, 2019.
- Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *Departmental Papers (CIS)*, page 107, 2000.
- Yu Shi, Qi Zhu, Fang Guo, Chao Zhang, and Jiawei Han. Easing embedding learning by comprehensive transcription of heterogeneous information networks. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2190–2199, 2018.
- Ben Shneiderman. The eyes have it: A task by data type taxonomy for information visualizations. In *Proceedings 1996 IEEE symposium on visual languages*, pages 336–343. IEEE, 1996.
- David Shuman, Sunil Narang, Pascal Frossard, Antonio Ortega, and Pierre Vandergheynst. The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *IEEE Signal Processing Magazine*, 3(30):83–98, 2013.
- David I Shuman, Benjamin Ricaud, and Pierre Vandergheynst. Vertex-frequency analysis on graphs. *Applied and Computational Harmonic Analysis*, 40(2):260–291, 2016.
- Gerard Sierksma and Han Hoogeveen. Seven criteria for integer sequences being graphic. *Journal of Graph theory*, 15(2):223–231, 1991.
- Arlei Silva, Wagner Meira Jr, and Mohammed J Zaki. Mining attribute-structure correlated patterns in large attributed graphs. *Proceedings of the VLDB Endowment*, 5(5):466–477, 2012.
- Samuel Silva, Beatriz Sousa Santos, and Joaquim Madeira. Using color in visualization: A survey. *Computers & Graphics*, 35(2):320–333, 2011.
- Tiago Simas and Luis M Rocha. Distance closures on complex networks. *Network Science*, 3(2):227–268, 2015.

Filippo Simini, Marta C González, Amos Maritan, and Albert-László Barabási. A universal model for mobility and migration patterns. *Nature*, 484(7392):96–100, 2012.

David Simmons, Justin Coon, and Animesh Datta. The quantum theil index: characterizing graph centralization using von neumann entropy. *arXiv preprint arXiv:1707.07906*, 2017.

Herbert A Simon. Models of man; social and rational. 1957.

Martin Simonovsky and Nikos Komodakis. Dynamic edge-conditioned filters in convolutional neural networks on graphs. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3693–3702, 2017.

Martin Simonovsky and Nikos Komodakis. Graphvae: Towards generation of small graphs using variational autoencoders. In *ICANN*, pages 412–422. Springer, 2018.

Roberta Sinatra, Dashun Wang, Pierre Deville, Chaoming Song, and Albert-László Barabási. Quantifying the evolution of individual scientific impact. *Science*, 354(6312):aaf5239, 2016.

Koustuv Sinha, Shagun Sodhani, Jin Dong, Joelle Pineau, and William L Hamilton. Clutrr: A diagnostic benchmark for inductive reasoning from text. *arXiv preprint arXiv:1908.06177*, 2019.

Richard Sinkhorn. A relationship between arbitrary positive matrices and doubly stochastic matrices. *The annals of mathematical statistics*, 35(2):876–879, 1964.

Richard Sinkhorn and Paul Knopp. Concerning nonnegative matrices and doubly stochastic matrices. *Pacific Journal of Mathematics*, 21(2):343–348, 1967.

Per Sebastian Skardal and Alex Arenas. Abrupt desynchronization and extensive multistability in globally coupled oscillator simplexes. *Physical review letters*, 122(24):248301, 2019.

Paul B Slater. A two-stage algorithm for extracting the multiscale backbone of complex weighted networks. *Proceedings of the National Academy of Sciences*, 106(26):E66–E66, 2009.

Nickolay Smirnov. Table for estimating the goodness of fit of empirical distributions. *The annals of mathematical statistics*, 19(2):279–281, 1948.

Adam Smith. *The Wealth of Nations*. 1776.

Laura M Smith, Linhong Zhu, Kristina Lerman, and Allon G Percus. Partitioning networks with node attributes by compressing information flow. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 11(2):15, 2016.

Jamie Snape, Jur Van Den Berg, Stephen J Guy, and Dinesh Manocha. The hybrid reciprocal velocity obstacle. *IEEE Transactions on Robotics*, 27(4):696–706, 2011.

Tom AB Snijders. Markov chain monte carlo estimation of exponential random graph models. *Journal of Social Structure*, 3(2):1–40, 2002.

Tom AB Snijders, Philippa E Pattison, Garry L Robins, and Mark S Handcock. New specifications for exponential random graph models. *Sociological methodology*, 36(1):99–153, 2006.

Tom AB Snijders, Gerhard G Van de Bunt, and Christian EG Steglich. Introduction to stochastic actor-based models for network dynamics. *Social networks*, 32(1):44–60, 2010.

Justin Solomon, Raif Rustamov, Leonidas Guibas, and Adrian Butscher. Continuous-flow graph transportation distances. *arXiv preprint arXiv:1603.06927*, 2016.

Chaoming Song, Shlomo Havlin, and Hernan A Makse. Self-similarity of complex networks. *Nature*, 433(7024):392–395, 2005.

Chaoming Song, Zehui Qu, Nicholas Blumm, and Albert-László Barabási. Limits of predictability in human mobility. *Science*, 327 (5968):1018–1021, 2010.

Sébastien Sorlin and Christine Solnon. Reactive tabu search for measuring graph similarity. In *International Workshop on Graph-Based Representations in Pattern Recognition*, pages 172–182. Springer, 2005.

F Sorrentino, M Di Bernardo, G Huerta Cuellar, and S Boccaletti. Synchronization in weighted scale-free networks with degree-degree correlation. *Physica D: nonlinear phenomena*, 224(1-2):123–129, 2006.

Sucheta Soundarajan, Tina Eliassi-Rad, Brian Gallagher, and Ali Pinar. Maxreach: Reducing network incompleteness through node probes. In *2016 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, pages 152–157. IEEE, 2016.

Sucheta Soundarajan, Tina Eliassi-Rad, Brian Gallagher, and Ali Pinar. ε -wgx: Adaptive edge probing for enhancing incomplete

networks. In *Proceedings of the 2017 ACM on Web Science Conference*, pages 161–170. ACM, 2017.

Charles Spearman. The proof and measurement of association between two things. *Am J Psychol*, 15:72–101, 1904.

Daniel A Spielman and Shang-Hua Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 81–90, 2004.

Olaf Sporns, Giulio Tononi, and Rolf Kötter. The human connectome: a structural description of the human brain. *PLoS computational biology*, 1(4), 2005.

Shrikanth Srivastava and Hon Wai Leong. A survey of computational methods for protein complex prediction from protein interaction networks. *Journal of bioinformatics and computational biology*, 11(02):1230002, 2013.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

Natalie Stanley, Saray Shai, Dane Taylor, and Peter J Mucha. Clustering network layers with the strata multilayer stochastic block model. *IEEE transactions on network science and engineering*, 3(2):95–105, 2016.

Dietrich Stauffer and Amnon Aharony. *Introduction to percolation theory*. Taylor & Francis, 2014.

Juliette Stehlé, François Charbonnier, Tristan Picard, Ciro Cattuto, and Alain Barrat. Gender homophily from spatial behavior in a primary school: a sociometric study. *Social Networks*, 35(4):604–613, 2013.

Stephen V Stehman. Selecting and interpreting measures of thematic classification accuracy. *Remote sensing of Environment*, 62(1):77–89, 1997.

Hugo Steinhaus. Sur la division des corp matériels en parties. *Bull. Acad. Polon. Sci*, 1(804):801, 1956.

Douglas Steinley. Properties of the hubert-arable adjusted rand index. *Psychological methods*, 9(3):386, 2004.

Nicholas M Stiffler and Jason M O’Kane. Pursuit-evasion with fixed beams. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4251–4258. IEEE, 2016.

Jonathan M Stokes, Kevin Yang, Kyle Swanson, Wengong Jin, Andres Cubillos-Ruiz, Nina M Donghia, Craig R MacNair, Shawn French, Lindsey A Carfrae, Zohar Bloom-Ackermann, et al. A deep learning approach to antibiotic discovery. *Cell*, 180(4):688–702, 2020.

Gilbert Strang. *Introduction to linear algebra*. Wellesley-Cambridge Press Wellesley, MA, 1993.

Alexander Strehl and Joydeep Ghosh. Cluster ensembles—a knowledge reuse framework for combining multiple partitions. *Journal of machine learning research*, 3(Dec):583–617, 2002.

Michael PH Stumpf and Mason A Porter. Critical truths about power laws. *Science*, 335(6069):665–666, 2012.

Daniel Stutzbach, Reza Rejaie, Nick Duffield, Subhabrata Sen, and Walter Willinger. Sampling techniques for large, dynamic graphs. In *Proceedings IEEE INFOCOM 2006. 25TH IEEE International Conference on Computer Communications*, pages 1–6. IEEE, 2006.

Daniel Stutzbach, Reza Rejaie, Nick Duffield, Subhabrata Sen, and Walter Willinger. On unbiased sampling for unstructured peer-to-peer networks. *IEEE/ACM Transactions on Networking (TON)*, 17(2):377–390, 2009.

Lovro Šubelj. Convex skeletons of complex networks. *Journal of The Royal Society Interface*, 15(145):20180422, 2018.

Lovro Šubelj and Marko Bajec. Robust network community detection using balanced propagation. *The European Physical Journal B*, 81(3):353–362, 2011.

Hanlin Sun and Ginestra Bianconi. Higher-order percolation processes on multiplex hypergraphs. *Physical Review E*, 104(3):034306, 2021.

Jiankai Sun, Deepak Ajwani, Patrick K Nicholson, Alessandra Sala, and Srinivasan Parthasarathy. Breaking cycles in noisy hierarchies. In *Proceedings of the 2017 ACM on Web Science Conference*, pages 151–160. ACM, 2017.

Jimeng Sun, Christos Faloutsos, Spiros Papadimitriou, and Philip S Yu. Graphscope: parameter-free mining of large time-evolving graphs. In *SIGKDD*, pages 687–696. ACM, 2007a.

Jimeng Sun, Yinglian Xie, Hui Zhang, and Christos Faloutsos. Less is more: Compact matrix decomposition for large sparse graphs. In *Proceedings of the 2007 SIAM International Conference on Data Mining*, pages 366–377. SIAM, 2007b.

Yizhou Sun and Jiawei Han. Mining heterogeneous information networks: a structural analysis approach. *Acm Sigkdd Explorations Newsletter*, 14(2):20–28, 2013.

Yizhou Sun, Jie Tang, Jiawei Han, Manish Gupta, and Bo Zhao. Community evolution detection in dynamic heterogeneous information networks. In *MLGraphs*, pages 137–146. ACM, 2010.

Yizhou Sun, Rick Barber, Manish Gupta, Charu C Aggarwal, and Jiawei Han. Co-author relationship prediction in heterogeneous bibliographic networks. In *2011 International Conference on Advances in Social Networks Analysis and Mining*, pages 121–128. IEEE, 2011.

Yizhou Sun, Jiawei Han, Charu C Aggarwal, and Nitesh V Chawla. When will it happen?: relationship prediction in heterogeneous information networks. In *Proceedings of the fifth ACM international conference on Web search and data mining*, pages 663–672. ACM, 2012.

Siva R Sundaresan, Ilya R Fischhoff, Jonathan Dushoff, and Daniel I Rubenstein. Network metrics reveal differences in social organization between two fission–fusion species, grevy’s zebra and onager. *Oecologia*, 151(1):140–149, 2007.

Latanya Sweeney. k-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(05):557–570, 2002.

Tracy M Sweet, Andrew C Thomas, and Brian W Junker. Hierarchical mixed membership stochastic blockmodels for multiple networks and experimental interventions. *Handbook on mixed membership models and their applications*, pages 463–488, 2014.

Danielle Albers Szafir. Modeling color difference for visualization design. *IEEE transactions on visualization and computer graphics*, 24(1):392–401, 2017.

Gabor J Szekely and Maria L Rizzo. Hierarchical clustering via joint between-within distances: Extending ward’s minimum variance method. *Journal of classification*, 22(2):151–183, 2005.

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.

Gabor Szucs and Gyula Sallai. Route planning with uncertain information using dempster-shafer theory. In *2009 International Conference on Management and Service Science*, pages 1–4. IEEE, 2009.

Andrea Tagarelli, Alessia Amelio, and Francesco Gullo. Ensemble-based community detection in multilayer networks. *Data Mining and Knowledge Discovery*, 31(5):1506–1543, 2017.

Shyam A Tailor, Felix L Opolka, Pietro Lio, and Nicholas D Lane. Adaptive filters and aggregator fusion for efficient graph convolutions. *arXiv preprint arXiv:2104.01481*, 2021.

Tomohiro Takagi and Michio Sugeno. Fuzzy identification of systems and its applications to modeling and control. *IEEE transactions on systems, man, and cybernetics*, (1):116–132, 1985.

Nassim Nicholas Taleb. *The black swan: The impact of the highly improbable*, volume 2. Random house, 2007.

Fei Tan, Yongxiang Xia, and Boyao Zhu. Link prediction in complex networks: a mutual information perspective. *PLoS one*, 9(9):e107056, 2014.

Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. Line: Large-scale information network embedding. In *Proceedings of the 24th international conference on world wide web*, pages 1067–1077. International World Wide Web Conferences Steering Committee, 2015a.

Jian Tang, Jingzhou Liu, Ming Zhang, and Qiaozhu Mei. Visualizing large-scale and high-dimensional data. In *WWW*, pages 287–297, 2016.

Jiliang Tang, Shiyu Chang, Charu Aggarwal, and Huan Liu. Negative link prediction in social media. In *Proceedings of the eighth ACM international conference on web search and data mining*, pages 87–96. ACM, 2015b.

Lei Tang, Huan Liu, Jianping Zhang, and Zohreh Nazeri. Community evolution in dynamic multi-mode networks. In *SIGKDD*, pages 677–685. ACM, 2008.

Lei Tang, Xufei Wang, and Huan Liu. Community detection via heterogeneous interaction analysis. *Data mining and knowledge discovery*, 25(1):1–33, 2012.

Jun Tao, Jian Xu, Chaoli Wang, and Nitesh V Chawla. Honvis: Visualizing and exploring higher-order networks. In *2017 IEEE Pacific Visualization Symposium (PacificVis)*, pages 1–10. IEEE, 2017.

Robert Tarjan. Depth-first search and linear graph algorithms. *SIAM journal on computing*, 1(2):146–160, 1972.

Marc Tarrés-Deulofeu, Antonia Godoy-Lorite, Roger Guimera, and Marta Sales-Pardo. Tensorial and bipartite block models for link prediction in layered networks and temporal networks. *Physical Review E*, 99(3):032307, 2019.

Nikolaj Tatti. Hierarchies in directed networks. In *2015 IEEE international conference on data mining*, pages 991–996. IEEE, 2015.

Dane Taylor, Saray Shai, Natalie Stanley, and Peter J Mucha. Enhanced detectability of community structure in multilayer networks through layer aggregation. *Physical review letters*, 116(22):228301, 2016.

Max Tegmark. The mathematical universe. *Foundations of physics*, 38(2):101–150, 2008.

Komal K Teru and William L Hamilton. Inductive relation prediction on knowledge graphs. *ICML, Virtual*, 2020.

Mikkel Thorup. On ram priority queues. *SIAM Journal on Computing*, 30(1):86–109, 2000.

Yu Tian and Renaud Lambiotte. Spreading and structural balance on signed networks. *SIAM Journal on Applied Dynamical Systems*, 23(1):50–80, 2024.

Yuanyuan Tian, Richard A Hankins, and Jignesh M Patel. Efficient aggregation for graph summarization. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 567–580, 2008.

Seth Tisue and Uri Wilensky. Netlogo: A simple environment for modeling complexity. In *International conference on complex systems*, volume 21, pages 16–21. Boston, MA, 2004.

Michele Tizzoni, Paolo Bajardi, Adeline Decuyper, Guillaume Kon Kam King, Christian M Schneider, Vincent Blondel, Zbigniew Smoreda, Marta C González, and Vittoria Colizza. On the use of human mobility proxies for modeling epidemics. *PLoS computational biology*, 10(7), 2014.

Hannu Toivonen, Fang Zhou, Aleksi Hartikainen, and Atte Hinkka. Compression of weighted graphs. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 965–973, 2011.

Hanghang Tong, B Aditya Prakash, Charalampos Tsourakakis, Tina Eliassi-Rad, Christos Faloutsos, and Duen Horng Chau. On the vulnerability of large graphs. In *2010 IEEE International Conference on Data Mining*, pages 1091–1096. IEEE, 2010.

- Yongxin Tong, Xiaofei Zhang, Caleb Chen Cao, and Lei Chen. Efficient probabilistic supergraph search over large uncertain graphs. In *CIKM*, pages 809–818, 2014.
- Jameson L Toole, Carlos Herrera-Yaqüe, Christian M Schneider, and Marta C González. Coupling human mobility and social ties. *Royal Society Interface*, 12(105):20141128, 2015.
- Jake Topping, Francesco Di Giovanni, Benjamin Paul Chamberlain, Xiaowen Dong, and Michael M Bronstein. Understanding over-squashing and bottlenecks on graphs via curvature. *arXiv preprint arXiv:2111.14522*, 2021.
- Leo Torres, Pablo Suárez-Serrato, and Tina Eliassi-Rad. Non-backtracking cycles: length spectrum theory and graph mining applications. *Applied Network Science*, 4(1):41, 2019.
- Vincent A Traag and Jeroen Bruggeman. Community detection in networks with positive and negative links. *Physical Review E*, 80(3):036115, 2009.
- Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. Complex embeddings for simple link prediction. International Conference on Machine Learning (ICML), 2016.
- Huynh Thanh Trung, Nguyen Thanh Toan, Tong Van Vinh, Hoang Thanh Dat, Duong Chi Thang, Nguyen Quoc Viet Hung, and Abdul Sattar. A comparative study on network alignment techniques. *Expert Systems with Applications*, 140:112883, 2020.
- Charalampos E Tsourakakis, Jakub Pachocki, and Michael Mitzenmacher. Scalable motif-aware graph clustering. In *Proceedings of the 26th International Conference on World Wide Web*, pages 1451–1460, 2017.
- Sho Tsugawa and Hiroyuki Ohsaki. Benefits of bias in crawl-based network sampling for identifying key node set. *IEEE Access*, 8:75370–75380, 2020.
- Ledyard R Tucker. Some mathematical notes on three-mode factor analysis. *Psychometrika*, 31(3):279–311, 1966.
- Edward Tufte and P Graves-Morris. The visual display of quantitative information.; 1983, 2014.
- Michele Tumminello, Tomaso Aste, Tiziana Di Matteo, and Rosario N Mantegna. A tool for filtering information in complex systems. *Proceedings of the National Academy of Sciences*, 102(30):10421–10426, 2005.

Michele Tumminello, Salvatore Micciche, Fabrizio Lillo, Jyrki Piilo, and Rosario N Mantegna. Statistically validated networks in bipartite complex systems. *PloS one*, 6(3):e17994, 2011.

Peter Uetz, Loic Giot, Gerard Cagney, Traci A Mansfield, Richard S Judson, James R Knight, Daniel Lockshon, Vaibhav Narayan, Maithreyan Srinivasan, Pascale Pochart, et al. A comprehensive analysis of protein–protein interactions in *saccharomyces cerevisiae*. *Nature*, 403(6770):623, 2000.

Eugenio Valdano, Luca Ferreri, Chiara Poletto, and Vittoria Colizza. Analytical computation of the epidemic threshold on temporal networks. *Physical Review X*, 5(2):021005, 2015.

Gerhard G Van de Bunt, Marijtje AJ Van Duijn, and Tom AB Snijders. Friendship networks through time: An actor-oriented dynamic statistical network model. *Computational & Mathematical Organization Theory*, 5(2):167–192, 1999.

Marijtje AJ Van Duijn, Tom AB Snijders, and Bonne JH Zijlstra. p2: a random effects model with covariates for directed graphs. *Statistica Neerlandica*, 58(2):234–254, 2004.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Jørn Vatn. Finding minimal cut sets in a fault tree. *Reliability Engineering & System Safety*, 36(1):59–62, 1992.

A Vazquez, R Dobrin, D Sergi, J-P Eckmann, ZN Oltvai, and A-L Barabási. The topological relationship between the large-scale attributes and local interaction patterns of complex networks. *Proceedings of the National Academy of Sciences*, 101(52):17940–17945, 2004.

Alexei Vázquez and Yamir Moreno. Resilience to damage of graphs with degree correlations. *Phys. Rev. E*, 67:015101, Jan 2003. DOI: [10.1103/PhysRevE.67.015101](https://doi.org/10.1103/PhysRevE.67.015101).

Andreas Veit, Michael Wilber, and Serge Belongie. Residual networks are exponential ensembles of relatively shallow networks. *arXiv preprint arXiv:1605.06431*, 1(2):3, 2016.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

Petar Veličković, William Fedus, William L Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. Deep graph infomax. *arXiv preprint arXiv:1809.10341*, 2018.

Petar Velickovic, William Fedus, William L Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. Deep graph infomax. *ICLR (Poster)*, 2(3):4, 2019.

Tiphaine Viard, Matthieu Latapy, and Clémence Magnien. Computing maximal cliques in link streams. *Theoretical Computer Science*, 609: 245–252, 2016.

Nguyen Xuan Vinh, Julien Epps, and James Bailey. Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance. *J. Mach. Learn. Res.*, 11(Oct): 2837–2854, 2010.

Pauli Virtanen, Ralf Gommers, Travis E Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, et al. Scipy 1.0: fundamental algorithms for scientific computing in python. *Nature methods*, pages 1–12, 2020.

SVN Vishwanathan, Karsten M Borgwardt, Nicol N Schraudolph, et al. Fast computation of graph kernels. In *NIPS*, volume 19, pages 131–138, 2006.

Ivan Voitalov, Pim van der Hoorn, Remco van der Hofstad, and Dmitri Krioukov. Scale-free networks well done. *arXiv preprint arXiv:1811.02071*, 2018.

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

Paul T Von Hippel. Mean, median, and skew: Correcting a textbook rule. *Journal of statistics Education*, 13(2), 2005.

Ulrike Von Luxburg. A tutorial on spectral clustering. *Statistics and computing*, 17(4):395–416, 2007.

John von Neumann. First draft of a report on the edvac. 1945.

Soroush Vosoughi, Deb Roy, and Sinan Aral. The spread of true and false news online. *Science*, 359(6380):1146–1151, 2018.

Claudia Wagner, David Garcia, Mohsen Jadidi, and Markus Strohmaier. It's a man's wikipedia? assessing gender inequality in an online encyclopedia. In *Ninth international AAAI conference on web and social media*, 2015.

Claudia Wagner, Eduardo Graells-Garrido, David Garcia, and Filippo Menczer. Women through the glass ceiling: gender asymmetries in wikipedia. *EPJ Data Science*, 5(1):5, 2016.

Glenn Wagner and Howie Choset. Subdimensional expansion for multirobot path planning. *Artificial Intelligence*, 219:1–24, 2015.

Thayne T Walker, David M Chan, and Nathan R Sturtevant. Using hierarchical constraints to avoid conflicts in multi-agent pathfinding. In *Int Conf on Automated Planning and Scheduling*, 2017.

Thayne T Walker, Nathan R Sturtevant, and Ariel Felner. Extended increasing cost tree search for non-unit cost domains. In *IJCAI*, pages 534–540, 2018.

Li Wan, Matthew Zeiler, Sixin Zhang, Yann Le Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In *International conference on machine learning*, pages 1058–1066. PMLR, 2013.

Bo Wang, Aziz M Mezlini, Feyyaz Demir, Marc Fiume, Zhuowen Tu, Michael Brudno, Benjamin Haibe-Kains, and Anna Goldenberg. Similarity network fusion for aggregating data types on a genomic scale. *Nature methods*, 11(3):333, 2014a.

Daixin Wang, Peng Cui, and Wenwu Zhu. Structural deep network embedding. In *SIGKDD*, pages 1225–1234, 2016a.

Dan J Wang, Xiaolin Shi, Daniel A McFarland, and Jure Leskovec. Measurement error in network data: A re-classification. *Social Networks*, 34(4):396–409, 2012a.

Dashun Wang, Dino Pedreschi, Chaoming Song, Fosca Giannotti, and Albert-Laszlo Barabasi. Human mobility, social ties, and link prediction. In *SIGKDD*, pages 1100–1108. Acm, 2011a.

Dashun Wang, Zhen Wen, Hanghang Tong, Ching-Yung Lin, Chaoming Song, and Albert-László Barabási. Information spreading in context. In *Proceedings of the 20th international conference on World wide web*, pages 735–744. ACM, 2011b.

Dashun Wang, Chaoming Song, and Albert-László Barabási. Quantifying long-term scientific impact. *Science*, 342(6154):127–132, 2013.

Liping Wang, Qing Li, Na Li, Guozhu Dong, and Yu Yang. Substructure similarity measurement in chinese recipes. In *Proceedings of the 17th international conference on World Wide Web*, pages 979–988, 2008a.

Peng Wang, BaoWen Xu, YuRong Wu, and XiaoYu Zhou. Link prediction in social networks: the state-of-the-art. *Science China Information Sciences*, 58(1):1–38, 2015a.

Pu Wang, Marta C González, César A Hidalgo, and Albert-László Barabási. Understanding the spreading patterns of mobile phone viruses. *Science*, 324(5930):1071–1076, 2009.

Quan Wang, Zhendong Mao, Bin Wang, and Li Guo. Knowledge graph embedding: A survey of approaches and applications. *IEEE Transactions on Knowledge and Data Engineering*, 29(12):2724–2743, 2017a.

Quanxin Wang, Olaf Sporns, and Andreas Burkhalter. Network analysis of corticocortical connections reveals ventral and dorsal processing streams in mouse visual cortex. *Journal of Neuroscience*, 32(13):4386–4399, 2012b.

Suhang Wang, Jiliang Tang, Charu Aggarwal, Yi Chang, and Huan Liu. Signed network embedding in social media. In *Proceedings of the 2017 SIAM international conference on data mining*, pages 327–335. SIAM, 2017b.

Xiao Wang, Houye Ji, Chuan Shi, Bai Wang, Yanfang Ye, Peng Cui, and Philip S Yu. Heterogeneous graph attention network. In *The world wide web conference*, pages 2022–2032, 2019.

Xiaohong Wang, Jun Huan, Aaron Smalter, and Gerald H Lushington. G-hash: towards fast kernel-based similarity search in large graph databases. In *Graph Data Management: Techniques and Applications*, pages 176–213. IGI Global, 2012c.

Yang Wang, Deepayan Chakrabarti, Chenxi Wang, and Christos Faloutsos. Epidemic spreading in real networks: An eigenvalue viewpoint. In *22nd International Symposium on Reliable Distributed Systems, 2003. Proceedings.*, pages 25–34. IEEE, 2003.

Yi Wang, Bin Wu, and Xin Pei. Commtracker: A core-based algorithm of tracking community evolution. In *ADMA*, pages 229–240. Springer, 2008b.

Yu-Xiang Wang, James Sharpnack, Alex Smola, and Ryan J Tibshirani. Trend filtering on graphs. *Journal of Machine Learning Research*, 17(105):1–41, 2016b.

Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. Knowledge graph embedding by translating on hyperplanes. In *Twenty-Eighth AAAI conference on artificial intelligence*, 2014b.

Zhen Wang, Lin Wang, Attila Szolnoki, and Matjaž Perc. Evolutionary games on multilayer networks: a colloquium. *The European physical journal B*, 88(5):124, 2015b.

Stephen Warshall. A theorem on boolean matrices. *Journal of the ACM (JACM)*, 9(1):11–12, 1962.

Stanley Wasserman and Katherine Faust. *Social network analysis: Methods and applications*, volume 8. Cambridge university press, 1994.

Ronald L Wasserstein and Nicole A Lazar. The asa statement on p-values: context, process, and purpose, 2016.

Duncan J Watts. Networks, dynamics, and the small-world phenomenon. *American Journal of sociology*, 105(2):493–527, 1999.

Duncan J Watts. A simple model of global cascades on random networks. *Proceedings of the National Academy of Sciences*, 99(9):5766–5771, 2002.

Duncan J Watts. *Six degrees: The science of a connected age*. WW Norton & Company, 2004.

Duncan J Watts and Steven H Strogatz. Collective dynamics of ‘small-world’networks. *nature*, 393(6684):440, 1998.

Bernard M Waxman. Routing of multipoint connections. *IEEE journal on selected areas in communications*, 6(9):1617–1622, 1988.

Samuel F Way, Allison C Morgan, Daniel B Larremore, and Aaron Clauset. Productivity, prominence, and the effects of academic environment. *Proceedings of the National Academy of Sciences*, 116(22):10729–10733, 2019.

Jan D Wegner, Javier A Montoya-Zegarra, and Konrad Schindler. A higher-order crf model for road network extraction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1698–1705, 2013.

E Weinan, Tiejun Li, and Eric Vanden-Eijnden. Optimal partition and effective dynamics of complex networks. *PNAS*, 105(23):7907–7912, 2008.

Boris Weisfeiler and Andrei Leman. The reduction of a graph to canonical form and the algebra which appears therein. *nti, Series*, 2(9):12–16, 1968.

Lilian Weng, Alessandro Flammini, Alessandro Vespignani, and Filippo Menczer. Competition among memes in a world with limited attention. *Scientific reports*, 2:335, 2012.

Lilian Weng, Filippo Menczer, and Yong-Yeol Ahn. Virality prediction and community structure in social networks. *Scientific reports*, 3:2522, 2013.

Lilian Weng, Filippo Menczer, and Yong-Yeol Ahn. Predicting successful memes using network and community structure. In *ICWSM*, 2014.

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

Charles Wheelan. *Naked statistics: Stripping the dread from the data*. WW Norton & Company, 2013.

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

Roland Wiese, Markus Eiglsperger, and Michael Kaufmann. yfiles—visualization and automatic layout of graphs. In *Graph Drawing Software*, pages 173–191. Springer, 2004.

Cort J Willmott and Kenji Matsuura. Advantages of the mean absolute error (mae) over the root mean square error (rmse) in assessing average model performance. *Climate research*, 30(1):79–82, 2005.

Michael Windzio. The network of global migration 1990–2013: Using ergms to test theories of migration between countries. *Social Networks*, 53:20–29, 2018.

Ludwig Wittgenstein. *Tractatus logico-philosophicus*, 1921.

Shoshana J Wodak, Shuye Pu, James Vlasblom, and Bertrand Seéraphin. Challenges and rewards of interaction proteomics. *Molecular & Cellular Proteomics*, 8(1):3–18, 2009.

Ling Heng Wong, Philippa Pattison, and Garry Robins. A spatial model for social networks. *Physica A*, 360(1):99–120, 2006.

Marc Wörlein, Thorsten Meinl, Ingrid Fischer, and Michael Philippsen. A quantitative comparison of the subgraph miners mofa, gspan, ffsm, and gaston. In *European Conference on Principles of Data Mining and Knowledge Discovery*, pages 392–403. Springer, 2005.

Chao-Yuan Wu, R Manmatha, Alexander J Smola, and Philipp Krahenbuhl. Sampling matters in deep embedding learning. In *Proceedings of the IEEE international conference on computer vision*, pages 2840–2848, 2017.

Zhihao Wu, Giulia Menichetti, Christoph Rahmede, and Ginestra Bianconi. Emergent complex network geometry. *Scientific reports*, 5(1):10073, 2015.

Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S Yu. A comprehensive survey on graph neural networks. *arXiv preprint arXiv:1901.00596*, 2019.

Wenjun Xiao and Ivan Gutman. Resistance distance and laplacian spectrum. *Theoretical chemistry accounts*, 110:284–289, 2003.

Jierui Xie, Boleslaw K Szymanski, and Xiaoming Liu. Slpa: Uncovering overlapping communities in social networks via a speaker-listener interaction dynamic process. In *2011 IEEE 11th International Conference on Data Mining Workshops*, pages 344–349. IEEE, 2011.

Jierui Xie, Stephen Kelley, and Boleslaw K Szymanski. Overlapping community detection in networks: The state-of-the-art and comparative study. *Acm computing surveys (csur)*, 45(4):43, 2013.

Eric P Xing, Wenjie Fu, Le Song, et al. A state-space mixed membership blockmodel for dynamic network tomography. *The Annals of Applied Statistics*, 4(2):535–566, 2010.

Jian Xu, Thanuka L Wickramarathne, and Nitesh V Chawla. Representing higher-order dependencies in networks. *Science advances*, 2(5):e1600028, 2016.

Kevin S Xu and Alfred O Hero. Dynamic stochastic blockmodels: Statistical models for time-evolving networks. In *International conference on social computing, behavioral-cultural modeling, and prediction*, pages 201–210. Springer, 2013.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018a.

Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. In *International conference on machine learning*, pages 5453–5462. PMLR, 2018b.

R. Xulvi-Brunet and I. M. Sokolov. Changing Correlations in Networks: Assortativity and Dissortativity. *Acta Physica Polonica B*, 36:1431, 5 2005.

Ramon Xulvi-Brunet and Igor M Sokolov. Reshuffling scale-free networks: From random to assortative. *Physical Review E*, 70(6):066102, 2004.

Konstantin Yakovlev and Anton Andreychuk. Any-angle pathfinding for multiple agents based on sipp algorithm. In *Twenty-Seventh International Conference on Automated Planning and Scheduling*, 2017.

- Bowen Yan and Steve Gregory. Detecting communities in networks by merging cliques. In *2009 IEEE International Conference on Intelligent Computing and Intelligent Systems*, volume 1, pages 832–836. IEEE, 2009.
- Gang Yan, Georgios Tsekenis, Baruch Barzel, Jean-Jacques Slotine, Yang-Yu Liu, and Albert-László Barabási. Spectrum of controlling and observing complex networks. *Nature Physics*, 11(9):779–786, 2015.
- Sijie Yan, Yuanjun Xiong, and Dahua Lin. Spatial temporal graph convolutional networks for skeleton-based action recognition. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.
- Xifeng Yan and Jiawei Han. gspan: Graph-based substructure pattern mining. In *2002 IEEE International Conference on Data Mining, 2002. Proceedings.*, pages 721–724. IEEE, 2002.
- Xifeng Yan and Jiawei Han. Closegraph: mining closed frequent graph patterns. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 286–295. ACM, 2003.
- Xifeng Yan, Philip S Yu, and Jiawei Han. Substructure similarity search in graph databases. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 766–777, 2005.
- Jaewon Yang and Jure Leskovec. Community-affiliation graph model for overlapping network community detection. In *2012 IEEE 12th international conference on data mining*, pages 1170–1175. IEEE, 2012.
- Jaewon Yang and Jure Leskovec. Overlapping community detection at scale: a nonnegative matrix factorization approach. In *Proceedings of the sixth ACM international conference on Web search and data mining*, pages 587–596. ACM, 2013.
- Jaewon Yang and Jure Leskovec. Overlapping communities explain core–periphery organization of networks. *Proceedings of the IEEE*, 102(12):1892–1902, 2014.
- Jaewon Yang and Jure Leskovec. Defining and evaluating network communities based on ground-truth. *Knowledge and Information Systems*, 42(1):181–213, 2015.
- Yang Yang, Ryan N Lichtenwalter, and Nitesh V Chawla. Evaluating link prediction methods. *Knowledge and Information Systems*, 45(3):751–782, 2015.
- Zhao Yang, René Algesheimer, and Claudio J Tessone. A comparative analysis of community detection algorithms on artificial networks. *Scientific Reports*, 6:30750, 2016a.

Zhen Yang, Ming Ding, Chang Zhou, Hongxia Yang, Jingren Zhou, and Jie Tang. Understanding negative sampling in graph representation learning. In *ACM SIGKDD*, pages 1666–1676, 2020.

Zhilin Yang, William W Cohen, and Ruslan Salakhutdinov. Revisiting semi-supervised learning with graph embeddings. *arXiv preprint arXiv:1603.08861*, 2016b.

Muhammed A Yildirim and Michele Coscia. Using random walks to generate associations between objects. *PLoS one*, 9(8):e104813, 2014.

Hao Yin, Austin R Benson, Jure Leskovec, and David F Gleich. Local higher-order graph clustering. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 555–564. ACM, 2017.

Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 974–983, 2018a.

Xue Ying. An overview of overfitting and its solutions. In *Journal of physics: Conference series*, volume 1168, page 022022. IOP Publishing, 2019.

Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. *Advances in neural information processing systems*, 31, 2018b.

Vahan Yoghoudjian, Tim Dwyer, Karsten Klein, Kim Marriott, and Michael Wybrow. Graph thumbnails: Identifying and comparing multiple graphs at a glance. *IEEE Transactions on Visualization and Computer Graphics*, 24(12):3081–3095, 2018.

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

Jiaxuan You, Rex Ying, Xiang Ren, William Hamilton, and Jure Leskovec. Graphrnn: Generating realistic graphs with deep autoregressive models. In *International Conference on Machine Learning*, pages 5694–5703, 2018.

Jiaxuan You, Zhitao Ying, and Jure Leskovec. Design space for graph neural networks. *Advances in Neural Information Processing Systems*, 33: 17009–17021, 2020.

- Hyejin Youn, Deborah Strumsky, Luis MA Bettencourt, and José Lobo. Invention as a combinatorial process: evidence from us patents. *Journal of The Royal Society Interface*, 12(106):20150272, 2015.
- Jean-Gabriel Young, Giovanni Petri, Francesco Vaccarino, and Alice Patania. Construction of and efficient sampling from the simplicial configuration model. *Physical Review E*, 96(3):032312, 2017.
- Amy Zhao Yu, Shahar Ronen, Kevin Hu, Tiffany Lu, and César A Hidalgo. Pantheon 1.0, a manually verified dataset of globally famous biographies. *Scientific data*, 3:150075, 2016.
- Bing Yu, Haoteng Yin, and Zhanxing Zhu. Spatio-temporal graph convolutional networks: a deep learning framework for traffic forecasting. In *IJCAI*, pages 3634–3640. AAAI Press, 2018.
- Haiyuan Yu and Mark Gerstein. Genomic analysis of the hierarchical structure of regulatory networks. *Proceedings of the National Academy of Sciences*, 103(40):14724–14731, 2006.
- Jingjin Yu and Daniela Rus. Pebble motion on graphs with rotations: Efficient feasibility tests and planning algorithms. In *Algorithmic Foundations of Robotics XI*, pages 729–746. Springer, 2015.
- Kai Yu, Wei Chu, Shipeng Yu, Volker Tresp, and Zhao Xu. Stochastic relational models for discriminative link prediction. In *Advances in neural information processing systems*, pages 1553–1560, 2007.
- Hao Yuan and Shuiwang Ji. Structpool: Structured graph pooling via conditional random fields. In *Proceedings of the 8th International Conference on Learning Representations*, 2020.
- Ye Yuan, Guoren Wang, Lei Chen, and Haixun Wang. Efficient subgraph similarity search on large probabilistic graph databases. *arXiv preprint arXiv:1205.6692*, 2012.
- Ye Yuan, Guoren Wang, Lei Chen, and Haixun Wang. Graph similarity search on large uncertain graph databases. *The VLDB Journal*, 24(2):271–296, 2015.
- Burcu Yucesoy and Albert-László Barabási. Untangling performance from success. *EPJ Data Science*, 5(1):17, 2016.
- Seongjun Yun, Minbyul Jeong, Raehyun Kim, Jaewoo Kang, and Hyunwoo J Kim. Graph transformer networks. *Advances in neural information processing systems*, 32, 2019.
- Wayne W Zachary. An information flow model for conflict and fission in small groups. *Journal of anthropological research*, 33(4):452–473, 1977.

Laura A Zager and George C Verghese. Graph similarity scoring and matching. *Applied mathematics letters*, 21(1):86–94, 2008.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep sets. *NeurIPS*, 30, 2017.

G Zehfuss. Über eine gewisse determinante. *Zeitschrift für Mathematik und Physik*, 3(1858):298–301, 1858.

Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. Graphsaint: Graph sampling based inductive learning method. *arXiv preprint arXiv:1907.04931*, 2019.

Zhiping Zeng, Jianyong Wang, Lizhu Zhou, and George Karypis. Coherent closed quasi-clique discovery from large dense graph databases. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 797–802. ACM, 2006.

Chenhan Zhang, JQ James, and Yi Liu. Spatial-temporal graph attention networks: A deep learning approach for traffic forecasting. *IEEE Access*, 7:166246–166256, 2019.

Hongming Zhang, Liwei Qiu, Lingling Yi, and Yangqiu Song. Scalable multiplex network embedding. In *IJCAI*, volume 18, pages 3082–3088, 2018a.

Jun Zhang and F Yu Kai. What's the relative risk?: A method of correcting the odds ratio in cohort studies of common outcomes. *Jama*, 280(19):1690–1691, 1998.

Jun Zhang, Mark S Ackerman, and Lada Adamic. Expertise networks in online communities: structure and algorithms. In *Proceedings of the 16th international conference on World Wide Web*, pages 221–230, 2007a.

Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks. In *Advances in Neural Information Processing Systems*, pages 5165–5175, 2018.

Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. An end-to-end deep learning architecture for graph classification. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018b.

Ning Zhang, Yuanyuan Tian, and Jignesh M Patel. Discovery-driven graph summarization. In *ICDE*, pages 880–891. IEEE, 2010.

Pan Zhang and Christopher Moore. Scalable detection of statistically significant communities and hierarchies, using message passing for modularity. *Proceedings of the National Academy of Sciences*, 111(51):18144–18149, 2014.

Peng Zhang, Jinliang Wang, Xiaoja Li, Menghui Li, Zengru Di, and Ying Fan. Clustering coefficient and community structure of bipartite networks. *Physica A: Statistical Mechanics and its Applications*, 387(27):6869–6875, 2008.

Shihua Zhang, Xuemei Ning, and Xiang-Sun Zhang. Identification of functional modules in a ppi network by clique percolation clustering. *Computational biology and chemistry*, 30(6):445–451, 2006.

Shihua Zhang, Rui-Sheng Wang, and Xiang-Sun Zhang. Identification of overlapping community structure in complex networks using fuzzy c-means clustering. *Physica A: Statistical Mechanics and its Applications*, 374(1):483–490, 2007b.

Xiao Zhang, Travis Martin, and Mark EJ Newman. Identification of core-periphery structure in networks. *Physical Review E*, 91(3):032803, 2015.

Xiaotong Zhang, Han Liu, Xiao-Ming Wu, Xianchao Zhang, and Xinyue Liu. Spectral embedding network for attributed graph clustering. *Neural Networks*, 142:388–396, 2021.

Zhilu Zhang and Mert Sabuncu. Generalized cross entropy loss for training deep neural networks with noisy labels. *Advances in neural information processing systems*, 31, 2018.

Zhong-Yuan Zhang and Yong-Yeol Ahn. Community detection in bipartite networks using weighted symmetric binary matrix factorization. *International Journal of Modern Physics C*, 26(09):1550096, 2015.

Peixiang Zhao, Xiaolei Li, Dong Xin, and Jiawei Han. Graph cube: on warehousing and olap multidimensional networks. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pages 853–864, 2011.

Elena Zheleva and Lise Getoor. Preserving the privacy of sensitive relationships in graph data. In *International Workshop on Privacy, Security, and Trust in KDD*, pages 153–171. Springer, 2007.

Elena Zheleva and Lise Getoor. To join or not to join: the illusion of privacy in social networks with mixed public and private user profiles. In *Proceedings of the 18th international conference on World wide web*, pages 531–540, 2009.

Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, and Maosong Sun. Graph neural networks: A review of methods and applications. *arXiv preprint arXiv:1812.08434*, 2018.

Shi Zhou and Raúl J Mondragón. The rich-club phenomenon in the internet topology. *IEEE Communications Letters*, 8(3):180–182, 2004.

Tao Zhou, Jie Ren, Matúš Medo, and Yi-Cheng Zhang. Bipartite network projection and personal recommendation. *Physical Review E*, 76(4):046115, 2007.

Tao Zhou, Linyuan Lü, and Yi-Cheng Zhang. Predicting missing links via local information. *The European Physical Journal B*, 71(4):623–630, 2009.

Tao Zhou, Zoltán Kuscsik, Jian-Guo Liu, Matúš Medo, Joseph Rushton Wakeling, and Yi-Cheng Zhang. Solving the apparent diversity-accuracy dilemma of recommender systems. *Proceedings of the National Academy of Sciences*, 107(10):4511–4515, 2010.

GK Zipf. The psycho-biology of language. 1935.

Marinka Zitnik, Monica Agrawal, and Jure Leskovec. Modeling polypharmacy side effects with graph convolutional networks. *Bioinformatics*, 34(13):i457–i466, 2018.

Vinko Zlatić, Andrea Gabrielli, and Guido Caldarelli. Topologically biased random walk and community finding in networks. *Physical Review E*, 82(6):066109, 2010.

Zhaonian Zou. Polynomial-time algorithm for finding densest subgraphs in uncertain graphs. In *Proceedings of MLG Workshop*, 2013.

Zhaonian Zou, Hong Gao, and Jianzhong Li. Discovering frequent subgraphs over uncertain graph databases under probabilistic semantics. In *SIGKDD*, pages 633–642, 2010.

Ezra W Zuckerman and John T Jost. What makes you think you're so popular? self-evaluation maintenance and the subjective side of the "friendship paradox". *Social Psychology Quarterly*, pages 207–223, 2001.

Konrad Zuse. *Der Plankalkül*. Number 63. Gesellschaft für Mathematik und Datenverarbeitung, 1972.