UMEÅ UNIVERSITY

Through the Coding-Lens

Community Detection and Beyond

Christopher Blöcker

Department of Physics
Umeå 2022

Dissertation for PhD

ISBN: 978-91-7855-827-8 (print)
ISBN: 978-91-7855-828-5 (pdf)

Cover design by Anja Sundberg, Inhousebyrån, Umeå University
Electronic version available at: http://umu.diva-portal.org/
Printed by: Cityprint i Norr AB
Umeå, Sweden, 2022

Contents

To my father who would have enjoyed reading this.

Abstract

We live in a highly-connected world and find networks wherever we look: social networks, public transport networks, telecommunication networks, financial networks, and more. These networks can be immensely complex, comprising potentially millions or even billions of inter-connected objects. Answering questions such as how to control disease spreading in contact networks, how to optimise public transport networks, or how to diversify investment portfolios requires understanding each network's function and working principles.

Network scientists analyse the structure of networks in search of communities: groups of objects that form clusters and are more connected to each other than the rest. Communities form the building blocks of networks, corresponding to their subsystems, and allow us to represent networks with coarse-grained models. Analysing communities and their interactions helps us unravel how networks function.

In this thesis, we use the so-called map equation framework, an information-theoretic community-detection approach. The map equation follows the minimum description length principle and assumes complete data in networks with one node type. We challenge these assumptions and adapt the map equation for community detection in networks with two node types and incomplete networks where some data is missing. We move beyond detecting communities and derive approaches for how, based on communities, we can identify influential objects in networks, and predict links that do not (yet) exist.

Sammanfattning

Vi lever i en värld som blir mer och mer sammanlänkad. Vart vi än tittar hittar vi nätverk: sociala nätverk, kollektivtrafiknätverk, telekommunikationsnätverk, finansiella nätverk och så vidare. Dessa nätverk kan vara oerhört komplexa och omfatta potentiellt miljoner eller till och med miljarder sammankopplade objekt. För att kunna besvara frågor som: hur kontrollerar vi sjukdomsspridning i kontaktnät, hur optimerar vi kollektivtrafiksnätverk eller hur diversifierar vi investeringsportföljer, krävs det att vi förstår varje nätverks funktion och principer.

Nätverksforskare analyserar strukturen i nätverk i jakt på kluster: grupper av objekt som är mer kopplade till varandra än till resten av nätverket. Kluster utgör byggstenarna, eller delsystemen, i nätverken och låter oss representera dessa med förenklade modeller. Att analysera kluster och deras interaktioner hjälper oss att ta reda på hur nätverk fungerar.

I denna avhandling vidareutvecklar vi den så kallade kartekvationen, en informationsteoretisk klusterdetekteringsmetod. Kartekvationen följer principen om minsta beskrivningslängd och förutsätter fullständiga data i nätverk som bara består av en typ av noder. Vi utmanar dessa antaganden och anpassar kartekvationen för klusterdetektering i nätverk som består av två typer av noder och ofullständiga nätverk där viss data saknas. Vi dyker också djupare in i kluster och härleder lösningar för hur vi, baserat på kluster, kan identifiera inflytelserika objekt i nätverk och förutsäga länkar som (ännu) inte existerar.

Zusammenfassung

Wir leben in einer hochgradig vernetzten Welt und finden Netzwerke wo auch immer wir hinschauen: soziale Netzwerke, öffentliche Verkehrsnetze, Telekommunikationsnetze, Finanznetzwerke und mehr. Diese Netzwerke können immens komplex sein und potenziell Millionen oder sogar Milliarden miteinander verbundener Objekten umfassen. Um beantworten zu können, wie wir die Ausbreitung von Krankheiten in Kontaktnetzwerken kontrollieren, öffentliche Verkehrsnetze optimieren oder Anlageportfolios diversifizieren können, müssen wir die Funktionsweise und Arbeitsprinzipien dieser Netzwerke verstehen.

Netzwerkwissenschaftler analysieren die Struktur von Netzwerken auf der Suche nach Communities: Gruppen von Objekten, die Cluster bilden und stärker miteinander verbunden sind als mit dem Rest. Communities repräsentieren die Bausteine von Netzwerken, entsprechen ihren Subsystemen und erlauben es uns, Netzwerke mit vereinfachten Modellen darzustellen. Communities und ihre Interaktionen untereinander zu verstehen hilft uns dabei, zu enträtseln, wie Netzwerke funktionieren.

In dieser Doktorarbeit verwenden wir die sogenannte Kartengleichung, ein informationstheoretischer Ansatz zur Community-Erkennung. Die Kartengleichung folgt dem Prinzip der minimalen Beschreibungslänge und nimmt an, dass Netzwerke einen Knotentypen haben und ihre zugrundeliegenden Daten vollständig sind. Wir stellen diese Annahmen infrage und passen die Kartengleichung zur Community-Erkennung in Netzwerken mit zwei Knotentypen und unvollständigen Daten an. Darüber hinaus leiten wir Ansätze ab, die, basierend auf Communities, einflussreiche Objekte in Netzwerken identifizieren und (noch) nicht existierende Verbindungen zwischen Objekten vorhersagen.

List of Papers

This thesis is based on the following papers:

I *Mapping Flows on Bipartite Networks*
Christopher Blöcker and Martin Rosvall
Physical Review E 102, 052305, November 2020

II *Mapping Flows on Weighted and Directed Networks with Incomplete Observations*
Jelena Smiljanić, **Christopher Blöcker**, Daniel Edler, Martin Rosvall
Journal of Complex Networks, Volume 9, Issue 6, December 2021

III *Map Equation Centrality: Community-aware Centrality Based on the Map Equation*
Christopher Blöcker, Juan Carlos Nieves, Martin Rosvall
Applied Network Science, Volume 7, Issue 1, August 2022

IV *Similarity-based Link Prediction from Modular Compression of Network Flows*
Christopher Blöcker, Jelena Smiljanić, Ingo Scholtes, Martin Rosvall
arXiv Preprint 2208.14220

Other work by the author that is not included in this thesis:

V *Sustainable International Experience: A Collaborative Teaching Project*
Thomas Mejtoft, Helen Cripps, Stefan Berglund, **Christopher Blöcker**
Proceedings of the 16th International CDIO Conference,
Gothenburg, Sweden, 9-11 June 2020

VI *Internationalization at home: An international interdisciplinary experience*
Thomas Mejtoft, Helen Cripps, **Christopher Blöcker**
8:e Utvecklingskonferensen för Sveriges ingenjörsutbildningar,
Karlstads Universitet, 24-25 November 2021

VII *International Professional Skills: Interdisciplinary Project Work*
Thomas Mejtoft, Helen Cripps, **Christopher Blöcker**
Proceedings of the 18th International CDIO Conference,
Reykjavík, Iceland, 13-15 June 2022.

Preface

> There is a way out of every box,
> a solution to every puzzle;
> it's just a matter of finding it.
>
> Jean-Luc Picard
> In: *Star Trek: The Next Generation*

During the past five years, I've had a lot of puzzles to solve. At the time, each of them seemed overwhelmingly unsolvable. All the projects I worked on had at least one moment when I thought I had reached a dead end. And then what, throw it all away because it doesn't work as it should? In most cases it was just a small mistake in an equation or a line of code that wasn't correct. But in other cases, the problem was more profound and I needed to look at it from another angle, adjust my assumptions, and accept that science doesn't always work the way I wanted it to. Eventually, I managed to find a solution to most puzzles, or at least those ones that mattered.

My thesis is yet another puzzle: what is its purpose, what do I want to say? I decided that I wanted to make my research easy to understand for readers who are generally interested in science. Therefore, I have provided a bit more background than the bare minimum to understand my research, used simple undirected example networks, and explained things in more detail than I usually would in a paper. I have also experimented with a way to visualise Huffman codes that I haven't used before, and hope that it helps to make clear what's going on. I hope you'll enjoy reading about my research.

And for the future, I'm sure there will be many more puzzles to solve because...

The trial never ends.

Q
In: *Star Trek: The Next Generation*

Introduction

Hit it.

Christopher Pike
In: *Star Trek: Strange New Worlds*

Today's world is highly connected and is getting more connected by the day. Wherever we look, we find networks. For example, there is our social network of friends, the traffic network we use for our daily commute, digital communication networks, the global financial network... Despite their vastly different domains, such networks share structural properties: their parts form clusters, tightly-knit communities of objects that belong together (Figure 1). In social networks, such communities are groups of people where most people are friends with each other. In traffic networks, roads connect crossings within neighbourhoods, cities, and regions, forming communities at different scales. In financial networks, communities are groups of actors who trade with each other. Such networks can be immensely complex, with potentially several millions of nodes and links, which makes it difficult to grasp them. Network scientists study networks to understand how they work, and, to simplify them, they search for communities. If we know a network's communities, we can, instead of looking at the network as a whole, examine each community separately and focus on the communities' interplay.

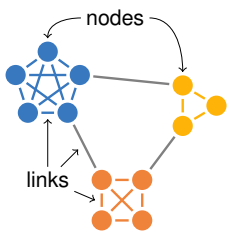


Figure 1: Three fully-connected communities in a network, shown by colours. Communities can be easy to spot in small networks.

In this thesis, we work with the map equation framework, an information-theoretic community detection approach that uses compression to find network communities. On a conceptual level, the map equation simulates a random walk through the

networks' nodes to reveal its structure. A so-called random walker begins at one of the network's nodes, picks one of the node's links, follows it to the next node, picks another link, follows it to the next node, and so on, generating a sequence of nodes whose regularities depend on the network's structure (Figure 2). Understanding those regularities enables us to devise an efficient way to encode, or compress, all possible node sequences that a random walker can generate on a network.

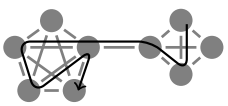


Figure 2: A random walk on a network.

The map equation was originally designed to detect communities in networks that have exactly one node type, and it assumes that the network data is complete, that is, that no links are missing. In other words, it simply describes the structure it finds when simulating random walks[1]. However, real-world networks can be more complicated, with nodes of different types or missing data, invalidating the map equation's assumptions. Moreover, we often want to do more than find a network's communities, for example, we want to know which nodes are influential, or what links are likely to form in the future.

[1] In practice, the map equation does not actually simulate random walks, but they are a good way to explain what is going on under the hood.

We look at the map equation through a coding-lens in this thesis, and study (i) community structure in bipartite networks[2], (ii) how we can avoid detecting spurious communities in networks with missing data, (iii) how we can identify influential network nodes, and (iv) how we can predict links that do not (yet) exist. In all four cases, we use the map equation's coding schemes for describing random walks to understand the challenges and to devise a solution. We explicitly distinguish between the node types in bipartite networks to compress bipartite random walks more efficiently. We use a Bayesian approach to account for missing data and to minimise the effect on what would be the optimal coding scheme with complete data. We derive a centrality score from the map equation's coding scheme to rank nodes by their importance. We interpret the coding scheme as an implicit embedding of the network's nodes, and use it to calculate similarities between node pairs to predict links.

[2] In bipartite networks, there are exactly two node types, and links connect only nodes of different types.

This thesis contains three parts: (I) a background sections that provides the necessary preliminaries about network science and the map equation, (II) the thesis' contributions with details about (i)-(iv), and (III) a summary of the contributions.

Part I

Background

Network Science

> If you want to understand
> function, study structure.
>
> Francis Crick

Network science is a discipline with contributions from computer science, mathematics, statistics, and other fields, and has applications in sociology and social network analysis [18, 26, 76], urban planning [15, 44, 57], logistics [11, 62], biology [16, 32, 45], ecology [24, 36, 80], finance [12, 75], and other areas [56]. The common theme amongst network science applications is that they use network models to represent phenomena and real-world systems, analyse those networks, and use the insights to explain observations and make predictions. That is, the aim is to understand how a real-world phenomenon works through modelling it as a network and analysing its structure. But what exactly is a network?

The Birth of Graph Theory

To answer the question what a network is, we take a short detour to Leonhard Euler who is often credited for inventing the field of graph theory in 1735 [53]. In what came to be known as the *Seven Bridges of Königsberg* problem, Euler was asked to find a round trip that crosses each of the seven bridges over the Pregel river in Königsberg exactly once (Figure 3).

Euler modelled the problem as a graph[3], a mathematical object consisting of nodes and links, representing land masses with nodes, and bridges with links (Figure 4). In graph theory terms,

[3] Some authors distinguish between *networks* and *graphs*, but we treat them as synonymous [56, p. 105].

Euler was looking for a round trip that starts and ends at the same node, and uses each link exactly once; today such a round trip is also known as a Eulerian cycle. He showed that a Eulerian cycle exist if and only if all nodes in the graph have even degree[4], and concluded that the round trip he was supposed to find cannot exist.

[4] The degree of a node is the number of links it has.

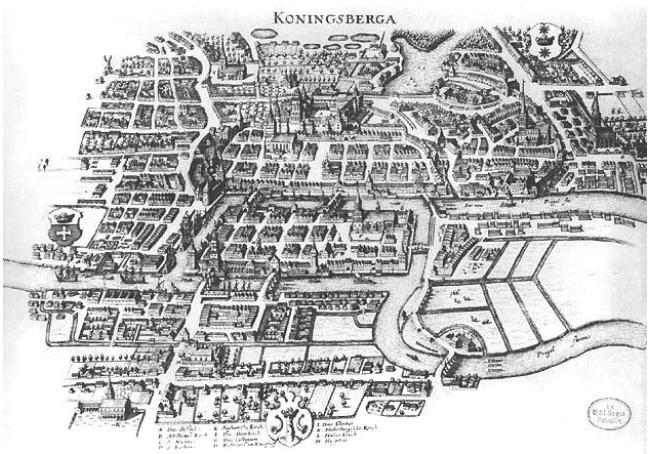


Figure 3: A map of the city Königsberg and its seven bridges over the Pregel river [52].

Figure 4: Graph model of the *Seven Bridges of Königsberg* problem.

Formally, a graph is a pair, $G = (V, E)$, where V is the set of nodes, and $E \subseteq V \times V$ is the set of links[5]. Links represent binary relationships between nodes, for example friendship between persons in social networks. We call nodes that are connected by a link *adjacent*, and say that they are neighbours. Graphs can be directed, then we draw the links as arrows that point from one node to another, and, similar to one-way streets, they can only be used in the direction they are pointing; or they can be undirected, then we draw links as lines between nodes, and they can be used in both directions. Often, there are values attached to links, so-called weights, that specify the strength of the relationship. A weighted graph is a triple, $G = (V, E, \delta)$, where $\delta \colon V \times V \to \mathbb{R}$ is a function that assigns weights to links[6] (Figure 5).

[5] In mathematics, nodes are usually called vertices, and links are called edges, explaining the symbols V and E.

[6] We will only consider positive link weights.

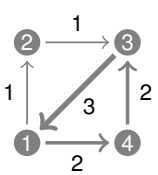


Figure 5: A directed and weighted graph where link weights are shown by labels and link thickness.

There are more graph model variants, for example multigraphs such as Euler's seven bridges graph where multiple links can connect the same node pairs, or hypergraphs where links can connect an arbitrary number of nodes. However, unless otherwise noted, we focus on simple weighted graphs.

Graphs as Matrices

We can define a graph by specifying its set of nodes and set of edges including weights, or by drawing it. But we can also represent it as a matrix and use linear algebra methods to analyse it. Converting a graphical representation of a graph into an equivalent matrix representation is quite straightforward. Consider again the directed and weighted graph in Figure 5, and its adjacency matrix A as well as its transition matrix T,

$$A = \begin{bmatrix} 0 & 0 & 3 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 2 \\ 2 & 0 & 0 & 0 \end{bmatrix}, \qquad T = \begin{bmatrix} 0 & 0 & 1 & 0 \\ \frac{1}{3} & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ \frac{2}{3} & 0 & 0 & 0 \end{bmatrix}.$$

The adjacency matrix A specifies which nodes are connected by a link and with what weight. For example, the first column shows that node 1 has two outgoing links: one of them points at node 2 and has weight 1, and the other one points at node 4 and has weight 2. The transitions matrix T is closely related with the adjacency matrix: it describes what fraction of a node's total outgoing link weight is assigned to each of its outgoing links, which corresponds to the rate at which a random walker uses each respective link. We obtain the transition matrix T by dividing the values in each column of A by the sum of the column's values.

Bipartite Graphs

Bipartite graphs are a special kind of graphs with two distinct sets of nodes, often called left and right nodes, where links are only allowed to run between different node types. Formally, a bipartite graph is a quadruple $G = (L, R, E, \delta)$, where L and R are the sets of left and right nodes, $E \subseteq L \times R$ is the set of links, and $\delta \colon L \times R \to \mathbb{R}$ is a function that assigns weights to the links. Figure 6 shows an example of a bipartite graph with four left nodes, four right nodes, and nine links. Typical examples for bipartite networks are user-movie rating networks where users are connected to movies if they have rated them, customer-product

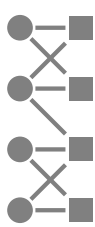


Figure 6: A bipartite network with four left nodes, four right nodes, and nine links.

purchase networks where customers are connected to the products they have bought, or ecological networks between species and the resources they use[7].

[7] Links that connect nodes with the same type, such as user-user or movie-movie connections, are not allowed in bipartite networks.

Modelling with Networks

Networks are simple, yet powerful tools for modelling real-world phenomena. But just as with any kind of model, using networks as models requires making assumptions and simplifications. Often, there are several options of how to *exactly* represent something as a network, each with different advantages and drawbacks. Depending on the research question, we need to decide, for example, what the nodes are, what relationships between the nodes are important and should be included, and how to define link weights. We also need to keep in mind that the results of our analyses can only be as good as our model is: we can only derive meaningful insights about the real world if the model reflects it accurately enough [70, 78].

Solving Graph Theory Problems

There are countless problems in graph theory, many of which we encounter in our everyday lives, perhaps even without thinking about it. We have already discussed the Eulerian cycle problem, that is, given a graph G, decide whether G has a Eulerian cycle. As we saw, it is easy to give an answer: we simply need to check whether all node degrees are even, and, if so, the graph has a Eulerian cycle[8].

[8] If we are interested to know whether a graph has a Eulerian cycle, chances are that we also want to find one. Luckily, there are efficient algorithms to do this [21].

Graph theory problems usually come in either of two forms: as decision problems, or as optimisation problems. In a decision problem, the task is to determine whether a graph has a certain property; in an optimisation problem, the task is to compute the best solution, given an optimisation criterion, such as finding the shortest path from a start node to a target node. However, conceptually there is no difference and decision problems can be converted to optimisation problems and vice versa [28, p. 19].

Complexity theory[9] quantifies how difficult a problem is by relating the problem's size to the minimum number of steps that

[9] An excellent introduction to complexity theory is the classic book by Garey and Johnson [28], as well as the introduction from a formal language point of view by Sipser [73].

any algorithm needs to execute on a computer to find a solution when confronted with the most difficult example of that problem — this is also called the problem's complexity. In graph theory, the size of a problem is typically the number of nodes, n, or number of links, m; the problem's complexity is expressed as a function of its size. For practical matters, we can group problems into easy and hard ones. Easy problems, also called *tractable*, are those whose complexity is bound by a polynomial, for example when a problem with size n needs quadratically many steps, we say that it is in complexity class $\mathcal{O}(n^2)$. Other examples for easy classes are $\mathcal{O}(\sqrt{n})$, $\mathcal{O}(n)$, $\mathcal{O}(n \log n)$, and $\mathcal{O}(n^k)$[10]. In contrast, hard problems, also called *intractable*, are those whose complexity is not bound by a polynomial, for example $\mathcal{O}(2^n)$. Essentially, computers can solve large examples of easy problems quickly, but may need hundreds of years to solve moderately sized hard problems. This is because the best known approach to solve hard problems is trying all possible solutions[11]. If it suffices to find a good enough solution instead of the best one, we can solve hard problems, albeit only approximately, within a reasonable amount of time using so-called heuristics or approximation algorithms. The trade-off is, however, that we may occasionally end up with a bad solution and no way of telling how bad it is.

[10] So long as k is a constant.

[11] This does not mean that no better algorithm exists, but so far, no one has found one [5].

Knowing how to solve a graph theory problem efficiently makes a difference in at least two ways. First, and on a more practically note, it enables a more sustainable approach using fewer resources, such as time or money, when we implement the solution in reality. Second, it means that we have gained a deeper understanding of the world around us.

Graph Theory in Everyday Life

A problem that many of us face every day is finding shortest paths. For example, when commuting to work, whether by car, bicycle, or public transport, we want to be efficient about it and take the fastest or shortest route[12]. But the best route is not always the same. During rush hour, the *shortest* route may require considerably more time than one that is *longer* but less busy. If

[12] Given a graph with n nodes and m links, Dijkstra's algorithm solves the shortest path problem in $\mathcal{O}(m + n \log n)$ time [27, 56].

we miss a connection because the bus was late, it may be faster to change our itinerary than sticking to our original plan. Clearly, the best route does not only depend on the network's structure, but changes with how others behave, that is, the dynamics on the network.

Another common problem is the travelling salesman problem, named after a hypothetical salesman who wishes to make a round trip through a number of cities while minimising his overall travel distance[13]. It comes up, for example, when a postman plans his delivery route, or when a worker plans a route through a warehouse to pick products from the shelves to compile customers' orders. In many cases, the network does not constrain the round trip and it is possible to visit the nodes in any order. But again, the current dynamics on the network, such as traffic patterns depending on the time of the day, determine which round trip is an efficient one.

[13] The travelling salesman problem is hard.

Community Detection

Community detection is our main focus in this thesis; it is the task of finding groups of nodes, called modules or communities[14], that are similar to each other in some sense. There is no single agreed upon definition of what a community is [25], so a group of pairwise disconnected nodes can be seen as one, called a dis-assortative community. However, we adopt the view of assortative communities, tightly-knit groups of nodes that are more connected to each other than the rest of the network (Figure 7) [55].

[14] We use the terms module and community interchangeably.

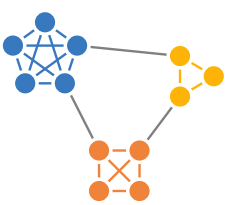


Figure 7: A network with three fully-connected communities, shown by colours.

The perhaps most intuitive community detection application is identifying groups of friends in a social network. But similarly, we may wish to find groups of proteins that interact with each other more than with others, sub-systems in an infrastructure network such as the road network or power grid, or categories of concepts in a knowledge graph. In all these cases, communities help us understand the networks' organisational structure and allow us to simplify them. Knowing their communities, we can reduce networks to their components and connections between those components[15], interpreting them as systems of interacting

[15] Some networks have a hierarchical organisation so that we can repeat this simplification several times, resulting in communities of communities.

sub-systems.

Just as there are many different ways to define what a community is, there are different approaches to detect them. Stochastic blockmodels are a class of approaches that generate modular networks by placing links between nodes according to some probability distribution, depending only on the nodes' block memberships. Given an empirical network, stochastic blockmodels can be turned into a community detection approach by fitting their parameters using, for example, maximum likelihood estimation or Bayesian inference [37, 61]. Modularity is an objective function that measures, given a network and a partition of its nodes into communities, how many links fall within those communities compared to how many links we would expect if they were placed at random [54]. The so-called Louvain and Leiden algorithms are heuristics that search for communities through maximising modularity [9, 79]. Compression-based community detection methods follow the principle that objects which compress more efficiently together should be grouped together [71]. The map equation, an information-theoretic community detection approach based on coding theory follows this idea; it is our main focus of study and we review it in detail in the next chapter, and extend it in papers I and II.

Figure 8: A network with two communities, connected through one node. Which of the ten nodes is most influential? The answer to this question depends on how we measure influence.

Influential Nodes

Who are the most influential persons in a social network (Figure 8)? Intuitively, we might say that those with the most connections to others are most influential. But what about someone who has relatively few connections, but is only connected to the most connected persons? Or someone who, through their position, controls the exchange of information between different groups in the network? Identifying influential nodes has applications such as finding relevant web pages when searching the world wide web for information, selecting critical infrastructure components to secure them with backups, and discovering persons who drive the spread of a disease to plan vaccination strategies.

Node centrality measures quantify how central, or influential,

nodes are, for example based on their features or their position in the network [13]. One of the simplest centrality measures, degree centrality, considers nodes with a higher degree as more important. Betweenness centrality regards a node as more central the more shortest paths between other nodes pass through it. PageRank determines node centrality by setting up a recursive reputation system where a node's centrality derives from the centrality of its neighbours and their support [31]. These measures, as well as others, consider nodes in isolation or in a global context when determining their centrality. Conversely, community-aware centrality measures use communities to determine node centrality. In paper III, we use the map equation to develop a community-aware centrality score.

Link Prediction

When we observe real-world phenomena and collect data to model them as networks, we may only be able to see a parts of the full picture. There are different reasons why this can happen, for example, nodes that interact only infrequently with others can remain inactive during the observation period. In some networks, links may be difficult to observe, such as protein-protein interactions in biological experiments. Certain links may not have formed yet, for example customers who have not bought a product yet. With a longer observation period, better measurement tools, or if we re-examine the network at a later point in time, we would potentially get a more complete picture.

Link prediction approaches take a network, analyse its link patters, and try to predict which of the links that are not present in the network should exist. This typically assumes some sort of process according to which the network has formed, and only really makes sense when we have reason to believe that the network is incomplete. In some technological networks, for example, where we know that the data is complete, we can technically apply link prediction approaches, but we have to ask ourselves whether it makes sense to do so. In paper IV, we develop a link prediction approach based on the map equation that uses communities to predict links.

The Map Equation

> Trees sprout up just about
> everywhere in computer science...
>
> *Donald Knuth*

The map equation is a community detection approach that combines ideas from information theory and coding theory with random walks [66]; and it is the basis for the works that we discuss in this thesis. Roughly speaking, given a network and a partition of its nodes into modules, the map equation measures how well the partition describes the network's structure[16]. Given two different partitions for the same network, we can use the map equation to calculate which one is better (Figure 9). And to find the best partition, we minimise the map equation with a search-based approach. But how exactly does the map equation work? To answer this, we need to cover some background on random walks and Huffman codes.

[16] Informally, a partition describes the network structure well when the density of links within modules is high, and low between.

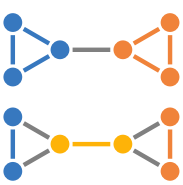


Figure 9: Two different partitions for the same network where colours show modules. Which partition describes the network's structure better?

Network Flow and Random Walks

Imagine a social network where people are connected if they are friends, and let us assume that one of the persons in the network has a ball. The person with the ball randomly chooses one of their friends, and passes the ball to that friend. We repeat this over and over. If we looked at how the ball moves through the network, we would see that, once it gets passed into a tightly-knit group of friends, it tends to stay there for some time. This is simply because there are many links within such a community, but only few links through which the ball can leave the com-

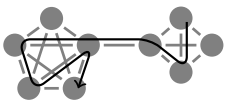


Figure 10: A ball passing game.

munity. We would also see the ball return to people with more links more often because they have more friends who can pass the ball to them (Figure 10).

We can model phenomena like the ball passing game with so-called random walks [50]. Given a graph $G = (V, E, \delta)$, we randomly pick a starting node at which we place a random walker. The random walker chooses an outgoing link from that node proportional to the links' weights, and follows that link; that is, the random walker is more likely to pick a link with higher weight. When the random walker is at node u, the probability to visit node v next is

$$p_{u \to v} = \frac{\delta(u, v)}{\sum_{v' \in V} \delta(u, v')}. \tag{1}$$

[17] Node flow is a centrality measure that has been used in several applications [31], but it ignores communities.

We call p_u the visit rate of u, or the flow of u[17]; it is the probability that the random walker is at node u. In undirected networks, we can calculate p_u directly as

$$p_u = \frac{\sum_{v \in V} \delta(u, v)}{\sum_{w \in V} \sum_{v \in V} \delta(w, v)}, \tag{2}$$

that is, it is proportional to u's weighted degree. For directed networks, there is no closed-form expression to calculate the visit rates, but we can calculate them through a simulation. We begin with a uniform flow distribution, setting the visit rates at time $t = 0$ to $p_{u,0} = \frac{1}{|V|}$ for all nodes $u \in V$. Then, we update the visit rates until we find a stable flow distribution that does not change any more when we run more update steps — this approach is also known as a power iteration [31] — using the update rule

$$p_{u,t+1} \leftarrow \sum_{v \in V} p_{v,t} \cdot \frac{\delta(v, u)}{\sum_{v' \in V} \delta(v, v')}. \tag{3}$$

That is, at each time step t, each node takes its flow and sends it to its neighbours along its outgoing links, dividing it proportional to the links' weights. We stop the simulation when $p_{u,t+1} = p_{u,t}$ for all nodes $u \in V$. The resulting flow distribution corresponds to the leading eigenvector of the graph's transition matrix, and has only strictly positive entries [56, p. 160].

[18] A node v is reachable from u if there is a path leading from u to v.

However, there is a catch: this approach only works if the network is strongly connected, that is if every node v is reachable[18]

from every other node u. In this case, the Perron-Frobenius theorem guarantees that the flow distribution we are looking for exists [56, p. 161]. But if the network is only weakly connected, that is if we need to ignore link directions for nodes to be pairwise reachable, we are in trouble. The problem is, essentially, that in weakly connected networks, there are nodes where the random walker can get stuck because they have no outgoing links, or nodes that the random walker cannot reach because they have no incoming links (Figure 11). To fix this problem, we relax the rules for the random walk and allow the random walker to teleport, making it possible to escape from nodes with no outgoing links and to reach nodes with no incoming links. There are many options for how exactly we can implement teleportation. A common approach is through uniform node teleportation[19] where the random walker teleports at a small rate r, typically $r = 0.15$, to a node chosen uniformly at random [31]. With uniform node teleportation, the simulation update rule becomes

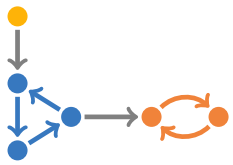


Figure 11: A weakly connected network. After leaving the yellow node, a random walker can never return. Eventually, every random walker will get stuck at the orange nodes.

[19] This approach is used in PageRank for ranking websites.

$$p_{u,t+1} \leftarrow r \cdot \frac{1}{|V|} + (1-r) \cdot \sum_{v \in V} p_{v,t} \cdot \frac{\delta(v,u)}{\sum_{v' \in V} \delta(v,v')}. \qquad (4)$$

Another option is so-called smart teleportation where the random walker teleports to links instead of nodes at a small rate r [42]. Uniform node teleportation and smart teleportation change the random walk's dynamics in different ways and have different advantages; in paper III, we investigate how they affect identifying influential nodes based on community structure [3]. In paper II, we develop a teleportation scheme where the teleportation rate depends on the random walker's current node, and the target node depends on the nodes' incoming degrees [2].

We can use random walks to model the flow of many processes, for example how people exchange ideas in social networks, how diseases spread through human contact networks, how users surf the world wide web by following hyperlinks, or how travellers use traffic networks. In some cases, random walkers have memory: users who surf the web are more likely to follow links that belong to a certain topic, and travellers are more likely to return to where they started their journey [10, 68, 81]. In general, with memory, the next random walker step

depends on some, or all, of the previous steps. But in this thesis, we focus on random walks without memory, and assume that the next step a random walker takes only depends on the current node.

Huffman Coding

Huffman coding, named after David A. Huffman, is a way to compress information in a lossless way by mapping symbols from a source alphabet to sequences of symbols from a target alphabet, called codewords. At a basic level, the idea is to assign shorter codewords to more common symbols. Take, for example, the Latin alphabet consisting of 26 letters, $\mathcal{A} = \{A, \ldots, Z\}$ as the source alphabet[20], and the binary alphabet $\mathcal{B} = \{0, 1\}$ as the target alphabet. Table 1 shows an example Huffman code that maps $\mathcal{A}$ to $\mathcal{B}^+$ based on letter frequencies in English, where $\mathcal{B}^+$ denotes sequences of symbols from $\mathcal{B}$. To encode a text, we take one symbol at a time and map it to its codeword, for example THE QUICK BROWN FOX would be encoded as `000 0110 100 001001001 01000 1011 01001 0010011 110000 0101 1101 00110 1010 00101 1101 001001011`, where the underlined sequences mark the original words.

	Freq.	Codeword
A	8.17	1110
B	1.49	110000
C	2.78	01001
D	4.25	11111
E	12.70	100
F	2.23	00101
G	2.02	110011
H	6.09	0110
I	6.97	1011
J	0.15	001001010
K	0.77	0010011
L	4.03	11110
M	2.41	00111
N	6.75	1010
O	7.51	1101
P	1.93	110001
Q	0.10	001001001
R	5.99	0101
S	6.33	0111
T	9.06	000
U	2.76	01000
V	0.98	001000
W	2.36	00110
X	0.15	001001011
Y	1.97	110010
Z	0.07	001001000

Table 1: Example Huffman code for the Latin alphabet with letters A-Z based on the letters' frequencies (in %) in English [88].

[20] For convenience, we ignore digits, punctuation marks, blanks, ...

How much did we compress the data in the example? This depends on how it was represented to start with, but let us assume it was stored in an ASCII-encoded text file with 7 bits per character. For 16 characters, this requires 112 bits. Using the Huffman code in Table 1, we only need 81 bits; on average, we expect to use 4.2 bits per symbol.

How much more can we compress the data without destroying information? Claude E. Shannon studied this question and laid the foundation for the field of information theory in his famous 1948 paper *A mathematical theory of communication* [72]. He showed that the theoretical lower bound for a lossless compression is the entropy of the source[21],

$$\mathcal{H}(\mathcal{A}) = -\sum_{x \in \mathcal{A}} p_x \log_2 p_x, \tag{5}$$

where x is a source symbol, and p_x the frequency at which it appears. However, Shannon only considered the theoretical lower

[21] By *source*, we mean some sort of random process that generates sequences of outcomes. Here, it is the English language that generates letters, but it could also be tossing a coin that generates heads or tails, rolling a die that generates the faces 1-6, or the ball passing game that generates a sequence of persons.

bound, but no specific compression algorithm, and, in fact, no compression algorithm that is guaranteed to reach the lower bound exists. The Huffman code in Table 1 with 4.2 bits per symbol gets pretty close to the lower bound of approximately 4.18 bits per symbol, derived from the symbols' frequencies using Equation (5). Huffman codes can be constructed using a simple and efficient algorithm, and they are optimal in the sense that they get as close to the entropy as it is possible for codes that encode one symbol at a time [35, 46].

	Freq.
A	0.10
B	0.25
C	0.15
D	0.15
E	0.35

Table 2: Five symbols and their frequencies.

Figure 12: Constructing a Huffman code. (a) We create a node for each symbol and label it with the symbol's frequency. (b) We choose the two nodes with the lowest frequencies and connect them into a tree, labelled with the sum of their frequencies. (c) We combine the next two lowest frequeny nodes into a tree. (d) When all nodes are part of a single tree, we label its links to obtain codewords.

Let us construct a binary Huffman code for an alphabet with five symbols, $\mathcal{A}_5 = \{A, B, C, D, E\}$, with frequencies as shown in Table 2, and entropy $\mathcal{H}(\mathcal{A}_5) \approx 2.18$ bits. We first create a node for each symbol and label it with its frequency (Figure 12(a)). Then, we select the two nodes with the lowest frequencies, connect them through a new node such that the new node forms the root of a tree, and label the root with the sum of the two nodes' frequencies (Figure 12(b)). If more than two nodes share the same lowest or second-lowest frequency, we can freely choose amongst them. This changes what symbol receives which exact codeword, but it does not change the average codelength. We repeat the same step, select the two lowest-frequency nodes or trees, and connect them into a new tree, again possibly choosing between different options (Figure 12(c)). Finally, when all nodes are part of a single tree (Figure 12(d)), we label its links: those links that point at a sub-tree to the left are labelled with `0`, and those ones that point at a sub-tree to the right are labelled with `1`. To find a symbol's codeword, we begin at the tree's root and follow the links leading to the corresponding node, reading the labels along the way. For example, the codeword for D is `001`.

While constructing Huffman codes, we can often choose between different lowest-frequency options, which affects how exactly the final tree looks like. That is, for the same symbols and symbol frequencies, we can construct potentially many Huffman codes, all of which are optimal. The simplest way to obtain another optimal Huffman code, given a Huffman code, is to flip all `0`s and `1`s.

However, when the source's statistics change, Huffman codes are no longer optimal. For example, Huffman codes based on the letter frequencies in German or Swedish would assign different codewords to the symbols because the letters have different frequencies in different languages. For the same reason, a Huffman code that is optimised for English would be suboptimal for compressing German texts. Relative entropy, also known as Kullback-Leibler divergence, quantifies how many extra bits we use per symbol if we encode a text with letter frequencies P but use a Huffman code that is optimised for letter frequencies Q,

$$D_{KL}(P\|Q) = -\sum_{x\in\mathcal{A}} p_x \log_2 \frac{q_x}{p_x}. \tag{6}$$

Random Walks + Huffman Coding ≈ Map Equation

To explain how the map equation works, let us consider a communication game where a sender observes how a random walker moves on a network and uses codewords to tell a receiver where the random walker is[22]. That is, we have a graph, $G = (V, E, \delta)$, and consider its nodes as symbols. For simplicity, we choose an undirected graph (Figure 13), and calculate the nodes' visit rates with Equation (2) to design a Huffman code that sender and receiver use for communication (Table 3). Every time the random walker moves to a new node, the sender transmits the corresponding codeword to the receiver. For example, to describe the node sequence 3 2 1 3 5 6 9 7 6, the sender communicates the codeword sequence `10 000 0110 10 001 010 1111 110 010` to the receiver, 27 bits in total, using exactly one codeword per random-walker step. On average, the coding scheme uses 3.125 bits per step of the random walker.

[22] In the ball passing game, for example, this means communicating who has the ball every time it is passed on.

u	p_u	$w(u)$
1	2/24	`0110`
2	3/24	`000`
3	4/24	`10`
4	2/24	`0111`
5	4/24	`001`
6	3/24	`010`
7	3/24	`110`
8	1/24	`1110`
9	2/24	`1111`

Table 3: One-level Huffman code for the network in Figure 13.

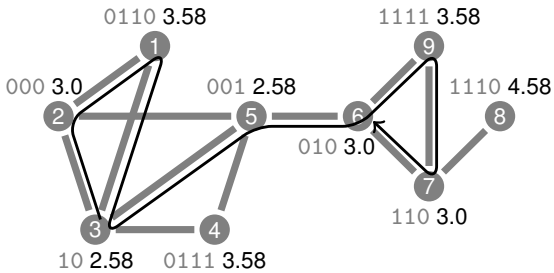


Figure 13: A network with eight nodes and twelve links. Codewords and their theoretical length in bits are shown next to the nodes. The black trace shows a possible random walker trajectory.

Again, the compression limit is determined by the entropy of the source, in this case the nodes' visit rates, that is, the the theoretically best possible compression uses on average

$$\mathcal{H}(P) = \sum_{p \in P} p \log_2 p \approx 3.07 \text{ bits}, \tag{7}$$

per random-walker step, where P is the set of node visit rates. We will refer to the entropy of the source, or the minimum average per-step number of bits, simply as *codelength*. Figure 14 visualises the Huffman code from Table 3 with blocks whose height is proportional to the codeword usage rates.

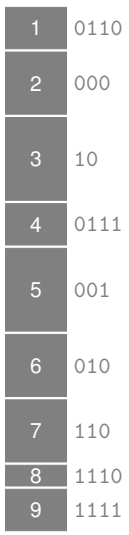


Figure 14: Visualisation of the Huffman code from Table 3, the block heights are proportional to codeword usage rates p_u for node u.

But this is not the end of the story; the sender can actually compress the random walk's description further. To do that, we change our interpretation of the network: instead of a single system, we consider the network as an ensemble of interconnected sub-systems, or modules, between which the random walker can switch. In terms of the ball passing game, modules correspond to tightly-knit groups of friends, and the ball can be passed from group to group. For each module, we design an individual Huffman code, including a designated exit codeword for describing when the random walker leaves the module. We also add an extra Huffman code at the index level with entry codewords for each module to describe when the random walker enters a module. The map equation formalises this idea and calculates the ensemble's overall entropy as the sum of the entropies at module and index level, each weighted by the rate at which the corresponding Huffman code is used [66]. If we can find a good split of the network into modules, one that leads to a lower overall entropy, we can compress the random walk's description further.

u	p_u	$w(u)$
A_{enter}	1/2	0
B_{enter}	1/2	1
1	2/16	110
2	3/16	10
3	4/16	00
4	2/16	1110
5	4/16	01
A_{exit}	1/16	1111
6	3/10	00
7	3/10	01
8	1/10	110
9	2/10	10
B_{exit}	1/10	111

Table 4: Two-level Huffman code for the network in Figure 15.

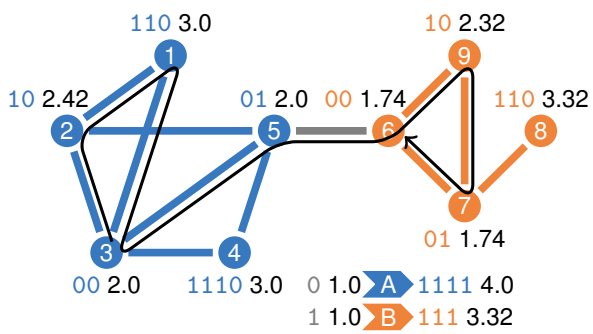
Figure 15: Splitting the network into two modules leads to a modular coding scheme. We can re-use the same codewords across different modules and reduce the overall codelength. Codewords and their theoretical length in bits are shown next to the nodes. The module entry and exit codewords are shown next to the coloured arrows.

[23] With a two-level coding scheme, the sender uses one codeword to describe intra-module transitions, and three codewords for inter-module transitions. Describing the starting node requires two codewords.

The map equation calculates the codelength for a given split of the network into modules, M,

$$\mathcal{L}(\mathsf{M}) = q \cdot \mathcal{H}(Q) + \sum_{\mathsf{m} \in \mathsf{M}} p_{\mathsf{m}} \cdot \mathcal{H}(P_{\mathsf{m}}), \tag{8}$$

where $Q = \{q_{m,\text{enter}} \mid \mathsf{m} \in \mathsf{M}\}$ is the set of module entry rates, and $P_{\mathsf{m}} = \{p_u \mid u \in \mathsf{m}\} \cup \{p_{\mathsf{m},\text{exit}}\}$ is the set of node visit rates for nodes in module m, including the module exit rate for m; $q = \sum_{\mathsf{m} \in \mathsf{M}} q_{\mathsf{m},\text{enter}}$ is the usage rate of the index-level codebook, and $p_{\mathsf{m}} = p_{\mathsf{m},exit} + \sum_{u \in \mathsf{m}} p_u$ is the usage rate of the module codebook for module m.



A possible way to split our network into two modules is shown in Figure 15, and gives us one codebook for the blue module, one codebook for the orange module, and one codebook at the index level. To design these codes, we interpret each module as a separate source and normalise the visit rates per module to sum to 1 (Table 4). With this new two-level coding scheme, the sender describes the node sequence 3 2 1 3 5 6 9 7 6 with the codewords 0 00 10 110 00 01 1111 1 00 10 01 00, 25 bits in total, where the underlined sequences corresponds to initially entering the blue module and the switch from the blue to the orange module, respectively[23]. The map equation (Equation (8)) tells us that the two-level coding scheme (Figure 16) has a codelength of

$$\mathcal{L}(\mathsf{M}) = q \cdot \mathcal{H}(Q) + \sum_{\mathsf{m} \in \mathsf{M}} p_{\mathsf{m}} \cdot \mathcal{H}(P_{\mathsf{m}}) \tag{9}$$

$$= \underbrace{\frac{2}{24} \cdot \mathcal{H}\left(\frac{1}{2}, \frac{1}{2}\right)}_{\text{index level}} + \underbrace{\frac{16}{24} \cdot \mathcal{H}\left(\frac{2}{16}, \frac{3}{16}, \frac{4}{16}, \frac{2}{16}, \frac{4}{16}, \frac{1}{16}\right)}_{\text{blue module}} + \underbrace{\frac{10}{24} \cdot \mathcal{H}\left(\frac{3}{10}, \frac{3}{10}, \frac{1}{10}, \frac{2}{10}, \frac{1}{10}\right)}_{\text{orange module}}$$

$$\approx 0.08 \text{ bits} + 1.64 \text{ bits} + 0.9 \text{ bits} \approx 2.62 \text{ bits},$$

an improvement over the one-level coding scheme that uses 3.07 bits per step on average. Partitioning the network into different modules and using a Huffman code that has codewords only for the nodes in the respective modules allows the sender to describe the random walk more efficiently.

Could we improve the compression further by splitting the network into even smaller parts? It depends. On the one hand, splitting the network into more modules leads to shorter codewords because modules become smaller and we can re-use the same, short codewords in different modules. On the other hand, more modules also mean that the index-level codebook grows because we need one entry codeword per module, making module switches more expensive. To achieve a short overall codelength, we need to find a partition that balances between (a) choosing a large number small modules to make intra-module transitions cheap, and (b) choosing a small number of large modules to make inter-module transitions cheap. Coming back to our initial example, we can now say that the two-module partition is a better description of the network's structure than the three-module partition (Figure 17).

Real-world networks are often more complex than our toy example and can have, for example, more than two modules, a hierarchical structure, or overlapping communities. We can address all these things with the map equation [10, 20, 81, 82]. Contrary to some other community detection approaches, we do not choose the number of modules we wish to detect. Instead, when minimising the map equation, we learn how many modules a network has. In networks with hierarchical structure, we can recursively apply the map equation and partition modules further into sub-modules [67], obtaining a coding scheme with more than two levels. And to address overlapping communi-

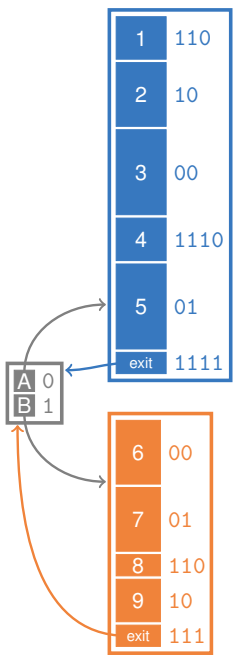


Figure 16: The modular coding scheme from Table 4. Block heights and widths are proportional to codeword usage rates and module entropy, respectively. Arrows represent module switches.

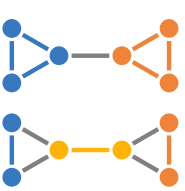


Figure 17: The two-module partition has codelength 2.32 bits, describing the network's structure better than the three-module partition which has a codelength of 3.34 bits.

ties where, for example, persons can be members in different groups of friends, we can use so-called higher-order networks [68, 82]. The main goal, in all cases, is to understand how networks are organised, exploiting the information-theoretic duality between compression and structure: by understanding the network's structure better, we can design a more efficient coding scheme to describe the dynamics on the network [67].

Finding Communities

In short, the map equation is an information-theoretic objective function that calculates the lower bound for the per-step description length of a Huffman code ensemble for describing random walks on a network, given a partition of the network's nodes into modules. While random walks and Huffman codes are useful tools to illustrate how the map equation works and what minimising the map equation means, in practice, we do not actually simulate random walks[24] or design Huffman codes during the minimisation. Instead, after calculating the nodes' visit and transition rates, we use the map equation directly to measure a partition's codelength. What remains is to turn this idea around: rather than measuring how good a partition is, we would like to use the map equation to find a good partition. Roughly, we do this by choosing some network partition as a starting point, and measureing its codelength. Next, we check whether we can reduce the partition's codelength by changing how the nodes are split into modules. If that is possible, we change the partition, and repeat until we can no longer reduce the codelength. The heuristic search algorithm Infomap implements a more sophisticated version of this basic strategy, reducing the chance to get stuck in local optimisation minima, and detecting communities in time $\mathcal{O}(m + n \log n)$, where n is the number of nodes, and m the number of links in the network [20]; Infomap is implemented in an open source software package [19].

[24] In principle, we could approximate the nodes' visit rates by simulating random walks instead of calculating them directly or with a power iteration, for example in case of a more complicated ball passing game [7]. However, we would do this before minimising the map equation and use the resulting node visit and transition rates as an input.

Open Questions

> Nothing that is is unimportant.
>
> Vulcan proverb
> In: *Strangers From The Sky*

Equipped with the map equation and Infomap, we can detect communities in large real-world networks, understand the networks' structure, and simplify them by representing them as networks of modules. Random walks help us explain why the modules are what they are: because a random walker tends to stay within modules for a relatively long amount of time. But in reality, things are not always as simple and clean as in the small networks we have considered so far.

Bipartite networks have two types of nodes that constrain link patterns, specifically, links can only connect nodes with different types. How can we use this constraint to find a more efficient coding scheme for describing random walks? When we observe a real-world network, we might end up with incomplete observations, such as missing links or incorrect link weights, meaning that we cannot calculate the nodes' visit rates accurately. How does missing data affect the communities we find, and how can we avoid detecting spurious communities? Moreover, community detection is often not the end of the story: based on a network's community structure, what can we say about how central a node is? And what links do we expect to see in the future?

Map Equation Assumptions

The map equation was originally designed for community detection in unipartite networks, that is, networks where all nodes have the same type and, in principle, any node can be connected to any other node. Huffman code ensembles correspond to network partitions, their codewords depend on the nodes' visit rates, which in turn are defined by the network's link patterns. As such, a partition that we infer using the map equation merely describes the observed link patterns, similar to how a descriptive statistic summarises data. From the perspective of the map equation, the available network data is all there is, whether it is in fact complete or not. Therefore, we must always interpret network partitions in the context of the data that we provided as input.

Through the Coding-Lens

We can detect communities with the map equation even if the assumptions are not satisfied, for example if our network is not unipartite, or has missing data. Doing that simply means that we pretend our network is unipartite and the data complete. We would still learn something about the network's structure, but nuances may be lost, and the detected communities can be imprecise. To understand what ignoring the assumptions means exactly, we will look through the coding-lens using Huffman codes. In paper I, we adapt the standard Huffman coding approach to capture the constraints that bipartite networks impose on link patterns, and, by extension, on random walks. In paper II, we assume incomplete network data and design a random walker teleportation scheme that reflects this assumption and allows us to estimate the true node visit and transition rates more accurately. We use the coding-lens to show how missing data affects the coding scheme, and how the new teleportation scheme corrects it.

To move beyond community detection, we use the coding-lens as well. While the original purpose of the map equation is to describe network in terms of modules by summarising flow patterns, we can use communities to figure out how important,

or central, specific nodes are. In paper III, we consider how not assigning a codeword to a single node affects the codewords assigned to other nodes. Assuming that leaving out more important nodes from the coding scheme affects the remaining nodes' codewords more, we derive a measure of node importance from the map equation. In paper IV, we use the fact that a Huffman code ensemble can describe random walker transitions along any link, even those that do not exist, to predict what links we expect to see in the future. Following the idea that random walker steps with a shorter description correspond to more likely links, we use the coding-lens to derive a measure for link prediction from the map equation.

In summary, the questions we discuss in the next part are:

I How do the link patterns in bipartite networks invalidate the assumptions made by the map equation? How can we reflect the constraints imposed by bipartite networks in the coding scheme, and what nuances does this let us see?

II What effect does missing data have on the community structure that the map equation identifies? How can we adjust for missing data and reduce the risk to detect spurious communities?

III How are the codewords for the rest of the nodes affected when we omit a node from the coding scheme, and how can we use the map equation to determine how important nodes are?

IV What does a network's community structure tell us about the likelihood that non-links[25] will form in the future, and how can we use the map equation to predict links?

[25] A non-link is a link that could, but does not exist.

Part II

Community Detection and Beyond

Bipartite Networks

> There are usually a large number of implied assumptions that are far from obvious if you think about them sufficiently carefully.
>
> Richard Feynman

How do the link patterns in bipartite networks[26] invalidate the assumptions made by the map equation? Let us remember that, in unipartite networks, links can in principle connect any two nodes. Huffman codes naturally reflect this property in the structure of their codebooks: within a codebook, any codeword can, in principle, be used at any time. Consider the bipartite network in Figure 18(a) with four left nodes, shown as circles, and four right nodes, shown as squares.

[26] We will only consider undirected bipartite networks.

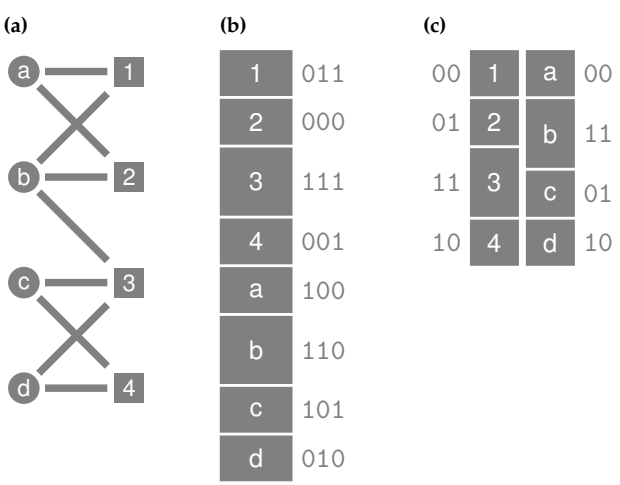


Figure 18: (a) A bipartite network with four left nodes, shown as circles, and four right nodes, shown as squares. The bipartite link pattern constrains the random walker to visit left and right nodes in turn, never the same type twice in a row. (b) A coding scheme that treats the network as unipartite. (c) A coding scheme that distinguishes between left and right nodes.

A coding scheme that treats the network as unipartite[27] also treats the codewords for left and right nodes the same, requiring

$$\mathcal{L}_U(\mathrm{M}_1) = \mathcal{H}\Big(\underbrace{\tfrac{2}{18}, \tfrac{2}{18}, \tfrac{3}{18}, \tfrac{2}{18}}_{\text{right nodes}}, \underbrace{\tfrac{2}{18}, \tfrac{3}{18}, \tfrac{2}{18}, \tfrac{2}{18}}_{\text{left nodes}}\Big) \approx 2.97 \text{ bits} \quad (10)$$

per random-walker step (Figure 18(b)). However, a random walker must visit left and right nodes in turn, never the same type twice in a row, because of the bipartite link pattern. A coding scheme that distinguishes between left and right nodes by keeping track of the random walker's current node type enables re-using codewords across node types (Figure 18(c)), and reduces the bipartite codelength to

$$\mathcal{L}_B(\mathrm{M}_1) = \underbrace{\frac{1}{2} \cdot \mathcal{H}\left(\tfrac{2}{18}, \tfrac{2}{18}, \tfrac{3}{18}, \tfrac{2}{18}\right)}_{\text{left to right}} + \underbrace{\frac{1}{2} \cdot \mathcal{H}\left(\tfrac{2}{18}, \tfrac{3}{18}, \tfrac{2}{18}, \tfrac{2}{18}\right)}_{\text{right to left}} \approx 1.97 \text{ bits.} \quad (11)$$

Whenever the random walker is at a left node, we use a codeword from the left codebook which contains codewords for right nodes, and vice-versa, following the random walk's pattern of alternating between left and right[28].

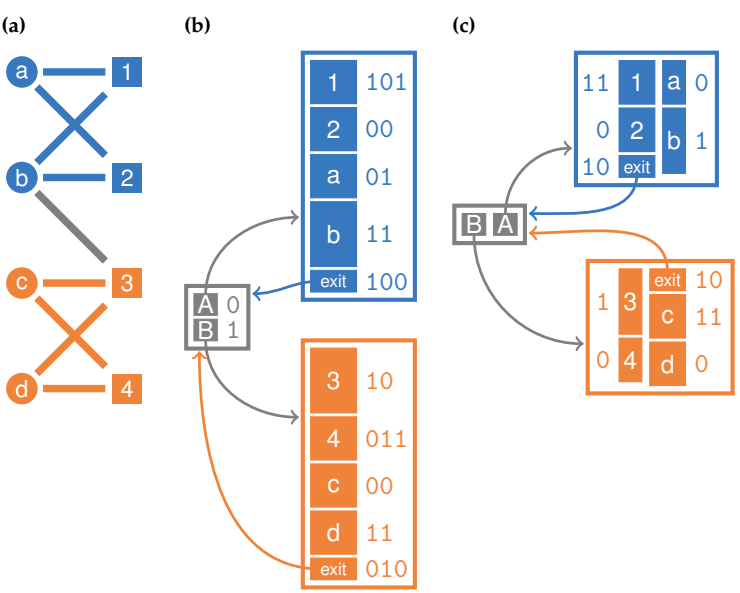


Figure 19: (a) Two-level partition of the bipartite example network. (b) A coding scheme that treats the network as unipartite, not distinguishing between left and right nodes. (c) A coding scheme that distinguishes between left and right nodes. In this specific case, there are no codewords at the index level because the random walker cannot choose between different options.

[27] In this chapter, we use the subscripts $_U$ and $_B$ to distinguish between unipartite and bipartite coding schemes.

[28] Since links in bipartite networks can only connect nodes with different types, the total degree of left nodes is equal to the total degree of right nodes, meaning that we encode left-to-right transitions half of the time, and right-to-left transitions the other half.

We can apply the same idea to two-level partitions. Splitting the network into its two fully connected compontents (Figure 19(a)) and treating it as if it was unipartite gives us a coding scheme where each codebook treats the left and right nodes the same (Figure 19(b)), with codelength

$$\mathcal{L}_U(\mathsf{M}_2) = \underbrace{\frac{2}{18}\cdot\mathcal{H}\left(\frac{1}{2},\frac{1}{2}\right)}_{\text{index level}} + \underbrace{\frac{10}{18}\cdot\mathcal{H}\left(\frac{2}{10},\frac{3}{10},\frac{2}{10},\frac{2}{10},\frac{1}{10}\right)}_{\text{blue module}} + \underbrace{\frac{10}{18}\cdot\mathcal{H}\left(\frac{2}{10},\frac{2}{10},\frac{3}{10},\frac{2}{10},\frac{1}{10}\right)}_{\text{orange module}}$$
$$\approx 0.11 \text{ bits} + 1.25 \text{ bits} + 1.25 \text{ bits} = 2.61 \text{ bits}. \tag{12}$$

By keeping track of the random walker's current node type, we separate left from right nodes in each module, reflecting how the random walk alternates between left and right (Figure 19(c)), and reducing the codelength to

$$\mathcal{L}_B(\mathsf{M}_2) = \underbrace{\underbrace{\frac{1}{18}\cdot\mathcal{H}(1)}_{\text{left to right}} + \underbrace{\frac{1}{18}\cdot\mathcal{H}(1)}_{\text{right to left}}}_{\text{index level}} \tag{13}$$
$$+ \underbrace{\underbrace{\frac{5}{18}\cdot\mathcal{H}\left(\frac{2}{5},\frac{2}{5},\frac{1}{5}\right)}_{\text{left to right}} + \underbrace{\frac{5}{18}\cdot\mathcal{H}\left(\frac{2}{5},\frac{3}{5}\right)}_{\text{right to left}}}_{\text{blue module}} + \underbrace{\underbrace{\frac{5}{18}\cdot\mathcal{H}\left(\frac{3}{5},\frac{2}{5}\right)}_{\text{left to right}} + \underbrace{\frac{5}{5}\cdot\mathcal{H}\left(\frac{2}{5},\frac{2}{5},\frac{1}{5}\right)}_{\text{right to left}}}_{\text{orange module}}$$
$$\approx 0 \text{ bits} + 0 \text{ bits} + 0.42 \text{ bits} + 0.27 \text{ bits} + 0.27 \text{ bits} + 0.42 \text{ bits} = 1.38 \text{ bits}.$$

Or, alternatively, by commutativity and re-ordering the terms, we may interpret the expression as two separate instances of the map equation, one each for encoding left-to-right and right-to-left transitions across the same partition[29],

$$\mathcal{L}_B(\mathsf{M}_2) = \underbrace{\underbrace{\frac{1}{18}\cdot\mathcal{H}(1)}_{\text{index level}} + \underbrace{\frac{5}{18}\cdot\mathcal{H}\left(\frac{2}{5},\frac{2}{5},\frac{1}{5}\right)}_{\text{blue module}} + \underbrace{\frac{5}{18}\cdot\mathcal{H}\left(\frac{3}{5},\frac{2}{5}\right)}_{\text{orange module}}}_{\text{left to right}} \tag{14}$$
$$+ \underbrace{\underbrace{\frac{1}{18}\cdot\mathcal{H}(1)}_{\text{index level}} + \underbrace{\frac{5}{18}\cdot\mathcal{H}\left(\frac{2}{5},\frac{3}{5}\right)}_{\text{blue module}} + \underbrace{\frac{5}{18}\cdot\mathcal{H}\left(\frac{2}{5},\frac{2}{5},\frac{1}{5}\right)}_{\text{orange module}}}_{\text{right to left}}.$$

[29] In thise case, the expression is quite symmetric because the network is quite symmetric. But this is not the case in general.

However, reducing the codelength comes at a cost. To use a bipartite coding scheme, sender and receiver need to keep track of the random walker's current node type and remember 1 bit of information. It turns out that this 1 bit is exactly the amount by which we can, and have reduced the codelength in the one-level case (Equation (10) vs. Equation (11)). Similarly, in the two-level case, the entropies at module level in all modules are reduced by 1 bit (Equation (12) vs. Equation (13)). However, in the two-level case, this is less obvious because the reduced entropies are multiplied with the codebook usage rates, resulting in an overall codelength reduction of more than 1 bit [1].

The Bipartite Map Equation

As we have seen, we can reflect the constraints imposed by bipartite networks in the coding scheme through keeping track of the random walker's current node type. We make use of more available information, and reduce the codelength.

For a more principled derivation of a bipartite map equation, we assume that $G = (L, R, E)$ is an unweighted and undirected bipartite graph[30] with left nodes L, right nodes R, and edges $E \subseteq L \times R$, and consider two random processes. Let us call the first process $\mathcal{X}$: *current node*, and assume that $\mathcal{X}$ yields a sequence of nodes from $L \cup R$ according to a random walk on G. For the one-level partition M_1 and treating the network as unipartite, the map equation describes $\mathcal{X}$'s entropy, that is $\mathcal{H}(\mathcal{X}) = \mathcal{L}_U(\mathsf{M}_1)$. The second process is $\mathcal{Y}$: *current node type*, and corresponds to the random walker's current node's type. We know that, in bipartite networks, random walks alternate between left and right nodes so that $\mathcal{Y}$ generates the sequence $l, r, l, r, l, r, \ldots$, and has an entropy of $\mathcal{H}(\mathcal{Y}) = 1$ bit[31].

We can define two more processes: $\mathcal{Y}|\mathcal{X}$: *current node type, given current node*, and $\mathcal{X}|\mathcal{Y}$: *current node, given current node type*. The third process, namely $\mathcal{Y}|\mathcal{X}$, is relatively boring because, if we know the current node, then we also know the current node type, that is $\mathcal{H}(\mathcal{Y}|\mathcal{X}) = 0$ bits. The fourth process, $\mathcal{X}|\mathcal{Y}$, is more interesting and corresponds to describing a random walk on a bipartite network with a bipartite coding scheme, and we define

[30] The derivation would also work with weights, but is clearer without.

[31] One may be tempted to argue that $\mathcal{H}(\mathcal{Y})$ is 0 bits because knowing $\mathcal{Y}$'s current state makes it possible to predict all of $\mathcal{Y}$'s future states correctly. But this only works if $\mathcal{Y}$'s current state was actually known, an information that is worth 1 bit.

$\mathcal{H}(\mathcal{X}|\mathcal{Y}) = \mathcal{L}_B(\mathsf{M}_1)$. Using Bayes' rule for conditional entropy, we relate the four processes to each other,

$$\mathcal{H}(\mathcal{X}|\mathcal{Y}) = \mathcal{H}(\mathcal{X}) - \mathcal{H}(Y) + \mathcal{H}(\mathcal{Y}|\mathcal{X}). \tag{15}$$

To construct a bipartite coding scheme, we consider the nodes' visit rates separately; let P^L and P^R be the sets of visit rates for left and right nodes, respectively[32]. Since the random walker moves from left to right nodes half of the time, and from right to left nodes the other half, we have

$$\mathcal{H}(\mathcal{X}|\mathcal{Y}) = \frac{1}{2}\cdot\mathcal{H}\left(P^R\right) + \frac{1}{2}\cdot\mathcal{H}\left(P^L\right). \tag{16}$$

We relate the one-level bipartite codelength to the one-level unipartite codelength by putting Equation (15) and Equation (16) together,

$$\mathcal{L}_U(\mathsf{M}_1) = \underbrace{\mathcal{H}(P)}_{\mathcal{H}(\mathcal{X})} = \underbrace{1\text{ bit}}_{\mathcal{H}(\mathcal{Y})} + \underbrace{\frac{1}{2}\cdot\mathcal{H}\left(P^R\right) + \frac{1}{2}\cdot\mathcal{H}\left(P^L\right)}_{\mathcal{H}(\mathcal{X}|\mathcal{Y})} = 1\text{ bit} + \mathcal{L}_B(\mathsf{M}_1). \tag{17}$$

And, finally, we plug Equation (17) into the two-level unipartite map equation, replacing the unipartite index and module codebooks with a bipartite version,

$$\begin{aligned}\mathcal{L}_U(\mathsf{M}) = q\left(1\text{ bit} + \frac{1}{2}\cdot\mathcal{H}\left(Q^R\right) + \frac{1}{2}\cdot\mathcal{H}\left(Q^L\right)\right)\\ + \sum_{\mathsf{m}\in\mathsf{M}} p_{\mathsf{m}}\left(1\text{ bit} + \frac{1}{2}\cdot\mathcal{H}\left(P^R_{\mathsf{m}}\right) + \frac{1}{2}\cdot\mathcal{H}\left(P^L_{\mathsf{m}}\right)\right),\end{aligned} \tag{18}$$

where M is a two-level partition; $Q^R = \left\{q^R_{\mathsf{m}} \,\middle|\, \mathsf{m}\in\mathsf{M}\right\}$ is the set of left-to-right, and $Q^L = \left\{q^L_{\mathsf{m}} \,\middle|\, \mathsf{m}\in\mathsf{M}\right\}$ the set of right-to-left module entry rates; and $P^R_{\mathsf{m}} = \left\{q^R_{\mathsf{m},\text{exit}}\right\} \cup \{p_u \,|\, u\in\mathsf{m}\cap R\}$ is the set of left-to-right, and $P^L_{\mathsf{m}} = \left\{q^L_{\mathsf{m},\text{exit}}\right\} \cup \{p_u \,|\, u\in\mathsf{m}\cap L\}$ the set of right-to-left node visit rates, both including module exits. Equation (18) makes it clear that a bipartite coding scheme saves more than 1 bit compared to a unipartite coding scheme[33].

Separating the left and right parts, and dropping $\mathcal{H}(\mathcal{Y})$, we define the bipartite map equation,

$$\begin{aligned}\mathcal{L}_B(\mathsf{M}) = q^R\cdot\mathcal{H}\left(Q^R\right) + \sum_{\mathsf{m}\in\mathsf{M}} p^R_{\mathsf{m}}\cdot\mathcal{H}\left(P^R_{\mathsf{m}}\right)\\ + q^L\cdot\mathcal{H}\left(Q^L\right) + \sum_{\mathsf{m}\in\mathsf{M}} p^L_{\mathsf{m}}\cdot\mathcal{H}\left(P^L_{\mathsf{m}}\right),\end{aligned} \tag{19}$$

[32] For ergodic visit rates in bipartite graphs, we consider averages over two random walker steps, that is, the visit rates we would obtain assuming a unipartite network [1].

[33] This is because the overall coding rate at which we save 1 bit is bigger than one in two-level partitions.

where $q^R = \sum_{\mathsf{m} \in \mathsf{M}} q^R_{\mathsf{m}}$ and $q^L = \sum_{\mathsf{m} \in \mathsf{M}} q^L_{\mathsf{m}}$ are the left-to-right and right-to-left index-level codebook usage rates, respectively; and $p^R_{\mathsf{m}} = \sum_{u \in \mathsf{m} \cap R} p_u$ and $p^L_{\mathsf{m}} = \sum_{u \in \mathsf{m} \cap L} p_u$ are the left-to-right and right-to-left module-level codebook usage rates, respectively.

Varying Node-Type Memory

With the bipartite map equation, we can encode random walks on bipartite networks more efficiently through remembering the random walker's current node type and alternating between left-to-right and right-to-left codebooks. But compression is not the ultimate goal. Instead, we want to learn something about the network's structure, and we consider those nodes that share codebooks and compress more efficiently together as belonging together [71]. However, in sparse regions of bipartite networks, remembering the node type comes close to remembering the actual node, preventing us from identifying modular structure [1]. The problem is that, from a compression perspective, it becomes attractive to create many small modules because they increase the coding rate, leading to more codelength savings (Equation (18)). To avoid this, we will use node-type information at intermediate rates.

We start by changing our interpretation of node flow and represent it as pairs in bipartite networks. Left nodes have all their flow on the left while right nodes have it on the right. A left node u with unipartite flow p_u has bipartite flow $p_{B,u} = (p_u, 0)$, and a right node v with unipartite flow p_v has bipartite flow $p_{B,v} = (0, p_v)$. Essentially, depending on the node's type, we push its flow to the left or right. This model also suggests another interpretation of flow in bipartite networks when we treat them as if they were unipartite: half of the flow is on the left, and the other half on the right, $p_{B,w} = \left(\frac{p_w}{2}, \frac{p_w}{2}\right)$, regardless of whether w is a left or right node, reflecting that we ignore node types. To move gradually from treating a network as unipartite to treating it as bipartite, we push only part of the flow towards the nodes' type. While nodes have a true type, left or right, we assume that we cannot remember them reliably: we remember

the correct type with probability $1-\alpha$, but make a mistake with probability α, on average leading to what we call mixed visit rates[34]. A left node u with unipartite visit rate p_u has a mixed visit rate $p_{\alpha,u} = ((1-\alpha)\,p_u, \alpha p_u)$, and a right node v with unipartite visit rate p_v has a mixed visit rate $p_{\alpha,v} = (\alpha p_v, (1-\alpha)\,p_v)$.

[34] We assume that we make those mistakes uniformly, independent from nodes and types.

With mixed visit rates, the entropy of $\mathcal{Y}|\mathcal{X}$: *current node type, given current node* changes because we expect to make a mistake with probability α, leading to $\mathcal{H}(\mathcal{Y}|\mathcal{X}) = \mathcal{H}_\alpha = \mathcal{H}(1-\alpha, \alpha)$. The entropy of $\mathcal{Y}$: *current node type* remains unchanged and is 1 bit because we still observe left and right nodes with probability $\frac{1}{2}$ each. We use Bayes' rule again to define a bipartite map equation with varying node-type memory α and relate it to the unipartite map equation. For a one-level partition M_1, we get

$$\mathcal{L}_U(\mathsf{M}_1) = \underbrace{\mathcal{H}(P)}_{\mathcal{H}(\mathcal{X})} = \underbrace{1 \text{ bit}}_{\mathcal{H}(\mathcal{Y})} - \underbrace{\mathcal{H}_\alpha}_{\mathcal{H}(\mathcal{Y}|\mathcal{X})} + \underbrace{\mathcal{H}(P_\alpha)}_{\mathcal{H}(\mathcal{X}|\mathcal{Y})} = 1 \text{ bit} - \mathcal{H}_\alpha + \mathcal{L}_\alpha(\mathsf{M}_1), \tag{20}$$

where P_α are the nodes' mixed visit rates, and $\mathcal{H}(P_\alpha)$ is shorthand for the total component-wise entropies of the mixed node visit rates. What Equation (20) says is that we reduce the codelength by 1 bit by knowing the nodes' types, but by making mistakes about the types, we increase it by $\mathcal{H}_\alpha$. Plugging Equation (20) into the two-level unipartite map equation, we get

$$\begin{aligned}\mathcal{L}_U(\mathsf{M}) = {} & q\,(1 \text{ bit} - \mathcal{H}_\alpha + \mathcal{H}(Q_\alpha)) \\ & + \sum_{\mathsf{m}\in\mathsf{M}} p_\mathsf{m}\,(1 \text{ bit} - \mathcal{H}_\alpha + \mathcal{H}(P_{\alpha,\mathsf{m}}))\,.\end{aligned} \tag{21}$$

Simplifying the notation and dropping the $1 \text{ bit} - \mathcal{H}_\alpha$ parts, we define the bipartite map equation with varying node type memory,

$$\mathcal{L}_\alpha(\mathsf{M}) = q_\alpha \cdot \mathcal{H}(Q_\alpha) + \sum_{\mathsf{m}\in\mathsf{M}} p_{\alpha,\mathsf{m}} \cdot \mathcal{H}(P_{\alpha,\mathsf{m}})\,, \tag{22}$$

where q_α and $p_{\alpha,\mathsf{m}}$ are the mixed index-level and module-level codebook usage rates, Q_α is the set of mixed module entry rates, and $P_{\alpha,\mathsf{m}}$ is the set of mixed node visit rates in module m, including the pair of module exit rates. For $\alpha = \frac{1}{2}$, we recover the unipartite map equation from Equation (22), and for $\alpha \in \{0, 1\}$, we recover the bipartite map equation[35].

[35] $\alpha = 0$ means we never make mistakes and always get the correct node types, and $\alpha = 1$ means we always make mistakes and never get the correct node types, but $\mathcal{L}_0(\mathsf{M}) = \mathcal{L}_1(\mathsf{M})$.

Setting $\alpha \in \left\{\frac{1}{2}, \frac{1}{4}, 0\right\}$ to encode the partition from Figure 20(a),

we gradually move from ignoring node types, $\mathcal{L}_U(M_2) = \mathcal{L}_{\frac{1}{2}}(M_2) \approx 2.61$ bits, through partially using node types,

$$\begin{aligned}
\mathcal{L}_{\frac{1}{4}}(M_2) = &\underbrace{\left(\frac{1}{18},\frac{1}{18}\right)\cdot\mathcal{H}\Big(\underbrace{\left(\frac{3}{4},\frac{1}{4}\right)}_{\text{blue}},\underbrace{\left(\frac{1}{4},\frac{3}{4}\right)}_{\text{orange}}\Big)}_{\text{index level}}\\
&+\underbrace{\left(\frac{5}{18},\frac{5}{18}\right)\cdot\mathcal{H}\Big(\underbrace{\left(\frac{6}{20},\frac{2}{20}\right)}_{a},\underbrace{\left(\frac{9}{20},\frac{3}{20}\right)}_{b},\underbrace{\left(\frac{2}{20},\frac{6}{20}\right)}_{1},\underbrace{\left(\frac{2}{20},\frac{6}{20}\right)}_{2},\underbrace{\left(\frac{1}{20},\frac{3}{20}\right)}_{\text{exit}}\Big)}_{\text{blue module}}\\
&+\underbrace{\left(\frac{5}{18},\frac{5}{18}\right)\cdot\mathcal{H}\Big(\underbrace{\left(\frac{6}{20},\frac{2}{20}\right)}_{c},\underbrace{\left(\frac{6}{20},\frac{2}{20}\right)}_{d},\underbrace{\left(\frac{3}{20},\frac{9}{20}\right)}_{3},\underbrace{\left(\frac{2}{20},\frac{6}{20}\right)}_{4},\underbrace{\left(\frac{3}{20},\frac{1}{20}\right)}_{\text{exit}}\Big)}_{\text{orange module}}\\
&\approx (0.05, 0.05)\text{ bits} + (0.53, 0.61)\text{ bits} + (0.61, 0.35)\text{ bits}\\
&= (1.19, 1.19)\text{ bits} = 2.38\text{ bits},
\end{aligned} \tag{23}$$

to fully using node types, $\mathcal{L}_B(M_2) = \mathcal{L}_0(M_2) = 1.38$ bits (Figure 20(b-d)). The struck-out codewords in Figure 20(c) are parts of the codes, but never actually used because they would describe transitions between same-type nodes that cannot happen.

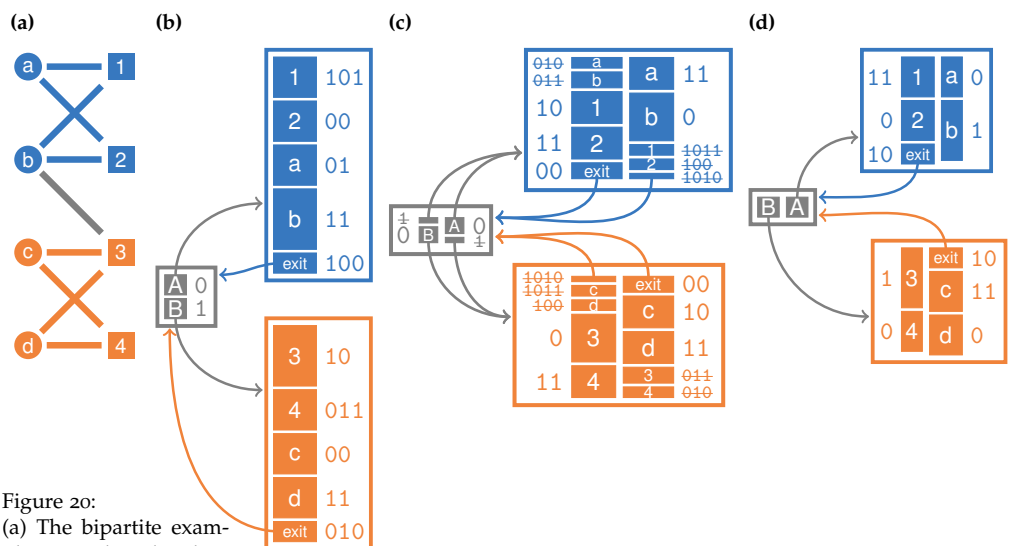


Figure 20:
(a) The bipartite example network and coding schemes for (b) $\alpha = \frac{1}{2}$, (c) $\alpha = \frac{1}{4}$, and (d) $\alpha = 0$.

Higher Resolution with more Node-Type Information

The bipartite map equation with varying node-type memory increases the resolution as we use more node-type information, and reveals communities at different scales [1]. For example, consider the weighted Fonseca-Ganade plant-ant web [23], a bipartite network where round nodes represent ant species, and square nodes represent plant species. Ant species are connected to those plant species which they use as a food source or for housing. With more certainty about node types, we detect communities at different scales, both coarser and finer (Figure 21).

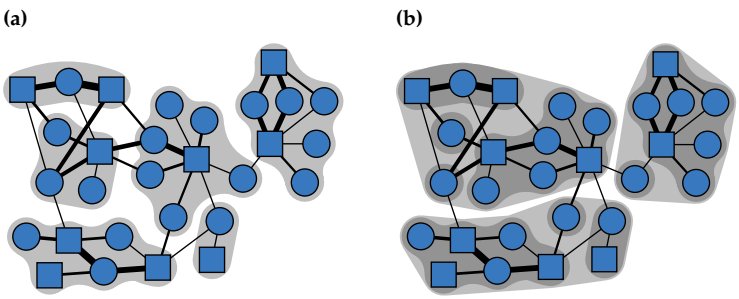


Figure 21: A food web where round nodes represent ant species, and square nodes represent plant species. A link between an ant and plant species means that the ant species uses the plant species as a food source or for housing. Link weights are shown by their thickness. We show the detected communities for (a) $\alpha = \frac{1}{2}$, and (b) $\alpha = \frac{1}{6}$.

To understand how the landscape of detected communities changes as we use more node-type information, we have implemented the bipartite map equation with varying node type memory in Infomap, and used it to detect communities in real-world bipartite networks, scanning from $\alpha = \frac{1}{2}$ to $\alpha = 0$. But because it is more intuitive to think about how much we know about node types rather than the probability of mis-remembering them, we use entropy to connect these two quantities and talk about the amount of node-type information that is available. Mis-remembering node types with probability α means that we have an uncertainty of $\mathcal{H}_\alpha$, and, consequently, the amount of available information about node types is $1 - \mathcal{H}_\alpha$. This connects well to the theory we have developed because, in one-level partitions, the available node-type information is exactly the amount by which we reduce the codelength as compared to the standard map equation.

We consider four[36] networks: the Fonseca-Ganade ant-plant

[36] Paper I contains results for another 17 networks [1].

Table 5: Sizes of four real-world bipartite networks where $|L|$ is the number of left nodes, $|R|$ the number of right nodes, and $|E|$ the number of links.

Name	$\lvert L\rvert$	$\lvert R\rvert$	$\lvert E\rvert$
Fonseca-Ganade ant-plant	19	10	38
LVHK Meetup attendance	6,061	5,096	127,033
IMDb actor-movie	124,414	374,511	1,460,791
Last.fm user-song	992	1,084,620	4,413,834

web [23], the Las Vegas Hikers (LVHK) Meetup attendance network [74], an Internet Movie Database (IMDb) actor-movie network [41], and a Laft.fm user-song network [41] with sizes as listed in Table 5.

We use Infomap to detect two-level as well as hierarchical partitions for node-type information from 0 bits to 1 bit, and measure the extra compression as the codelength reduction compared to the one-level partition. The extra compression is always at least 0 bits because Infomap returns the one-level partition if it cannot find a modular partition with lower codelength. To quantify the achieved resolution, we calculate the effective module size as the perplexity over the relative sizes of the detected modules: let S be the set of leaf module[37] sizes, where size refers to the number of nodes in a module. The perplexity of S, that is $2^{\mathcal{H}(S)}$, with $\mathcal{H}(S) = -\sum_{s\in S} \frac{s}{N} \log_2 \frac{s}{N}$, describes the effective number of leaf modules, similar to how the perplexity of a (loaded) coin's probability distribution for heads and tails describes the coin's effective number of sides[38]. To calculate the effective number of modules, we divide the number of nodes by the perplexity, $\frac{N}{2^{\mathcal{H}(S)}}$.

Our results show that the extra compression that we achieve with the bipartite map equation with varying node-type memory goes beyond the amount of available node-type information. Similarly, with more node-type information, modules tend to get smaller, or, in other words, the community structure's resolution increases. When we get close to using full node-type information, the effective module size tends towards one, and we detect many small modules. Again, we can explain this with an incentive to increase the coding rate: we can reduce each module codelength by the amount of available node-type information, which is then multiplied with the module's codebook usage rate.

[37] Leaf modules are those modules that contain no sub-modules. For example in two-level partitions, the index-level module contains sub-modules and, therefore, is no leaf module.

[38] A fair coin that turns up heads or tails each with probability $\frac{1}{2}$ has effectively $2^{\mathcal{H}\left(\frac{1}{2},\frac{1}{2}\right)} = 2$ sides. But a coin that always turns up heads only has $2^{\mathcal{H}(1,0)} = 1$ side. In general, a coin that turns up heads with probability α has $2^{\mathcal{H}_\alpha}$ sides.

Increasing the overall coding rate increases the rate at which the compression benefits from the available node-type information. However, as the leaf modules become smaller, the community hierarchy becomes deeper such that higher levels still contain significant structures (Figure 22).

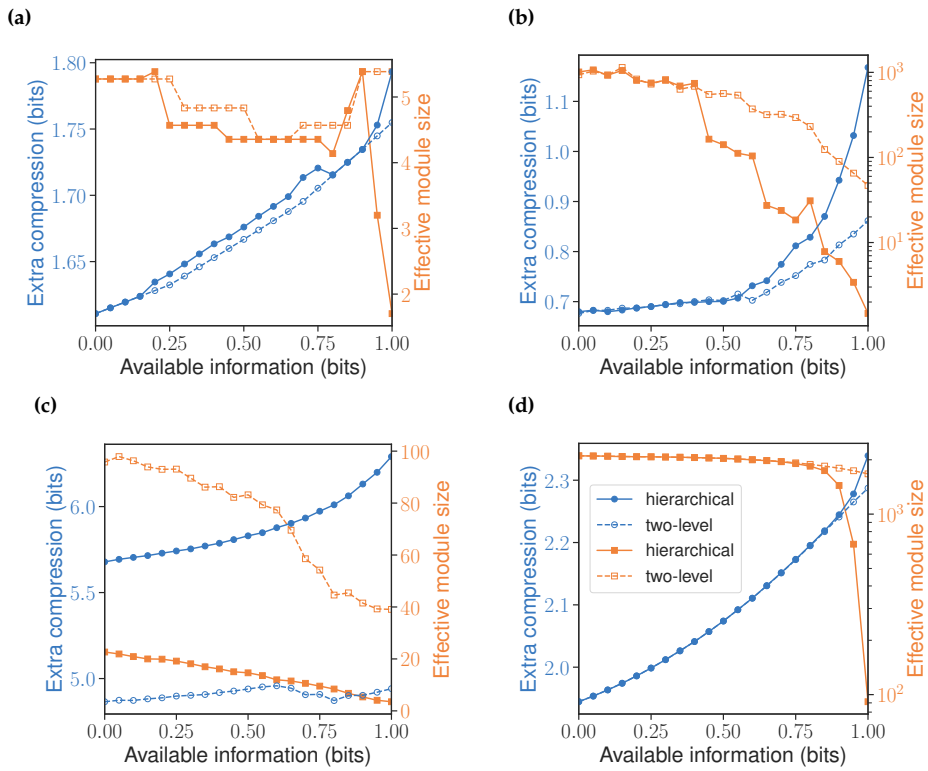


Figure 22: Properties of the community landscape in four bipartite networks. (a) Fonseca-Ganade ant-plant web. (b) Las Vegas Hikers Meetup attendance. (c) IMDb actor-movie network. (d) Last.fm user-song network.

Future Directions

The bipartite map equation with varying node-type memory is just one of many possible ways to generalise the standard map equation. With a similar approach, we could define a map equation for multipartite networks, that is, networks with more than two types of nodes where links follow some pattern in how they connect different-type nodes. For example, one could model bipartite customer-product purchase networks in a more fine-granular fashion by splitting products into categories such as movies, books, beverages, etc., and representing each category

with a different node type. Clearly, bipartite networks are a special case of multipartite networks with only two node types.

Another possible generalisation would be to consider partially bipartite networks where nodes could assume types on a continuous scale between left and right. Links could then in principle connect any two nodes, but may be more likely between more dissimilar nodes. For example, in network models of markets, the producers, traders, and consumers could be placed on a scale from left to right, or seller to buyer, depending on their respective volumes of business transactions where they assume roles as sellers or buyers. Bipartite networks are a special case of partially bipartite networks where each nodes assumes exactly one role, either left or right.

Since we have derived the bipartite map equation from the standard map equation, we ended up with a co-clustering of left an right nodes. However, one could de-couple the codebooks for left-to-right and right-to-left transitions to allow for different communities amongst left and right nodes.

Incomplete Data

> We have to remember that what we observe is not nature herself, but nature exposed to our method of questioning.
>
> Werner Heisenberg

What effect does missing data have on the community structure that the map equation identifies? Let us explore this question[39] using a weighted toy network with seven nodes and ten links (Figure 23). We assume that the network is complete as shown, that is, no data is missing, and denote it by $G_\bullet$. Infomap partitions the network into two modules, $\mathsf{M}_2 = \{\{1,2,3\},\{4,5,6,7\}\}$,

[39] We now leave the bipartite map equation behind and return to using unipartite coding schemes.

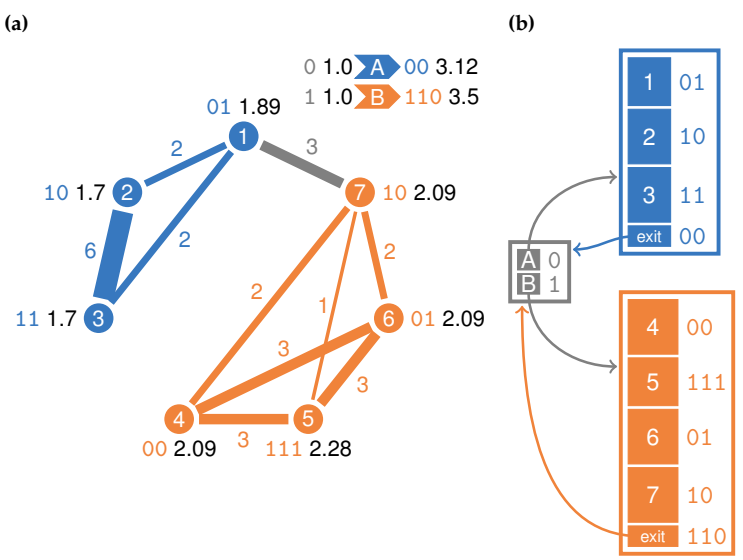


Figure 23: (a) A toy network with seven nodes and ten links where link weights are shown next to the links, and codewords and their theoretical length in bits are shown next to the nodes in coloured text and in black, respectively. We assume that the link data shown here is complete. (b) The coding scheme for the highlighted partition.

and the codelength for this partition is

$$\mathcal{L}(G_\bullet, \mathsf{M}_2) = \underbrace{\frac{6}{54} \cdot \mathcal{H}\left(\frac{3}{6}, \frac{3}{6}\right)}_{\text{index level}} + \underbrace{\frac{26}{54} \cdot \mathcal{H}\left(\frac{7}{26}, \frac{8}{26}, \frac{8}{26}, \frac{3}{26}\right)}_{\text{blue module}} + \underbrace{\frac{34}{54} \cdot \mathcal{H}\left(\frac{8}{34}, \frac{7}{34}, \frac{8}{34}, \frac{8}{34}, \frac{3}{34}\right)}_{\text{orange module}}$$

$$\approx 0.11 \text{ bits} + 0.92 \text{ bits} + 1.42 \text{ bits} = 2.45 \text{ bits}, \tag{24}$$

But what if some data was missing, for example because the network models a real-world system that is difficult to observe in its entirety? Let us assume that we have incomplete observations for four of the links (Figure 24), and denote the resulting network as $G_\circ$. In the new, incomplete network $G_\circ$, the codelength for M_2 is

$$\mathcal{L}(G_\circ, \mathsf{M}_2) = \underbrace{\frac{6}{46} \cdot \mathcal{H}\left(\frac{3}{6}, \frac{3}{6}\right)}_{\text{index level}} + \underbrace{\frac{24}{46} \cdot \mathcal{H}\left(\frac{6}{24}, \frac{7}{24}, \frac{8}{24}, \frac{3}{24}\right)}_{\text{blue module}} + \underbrace{\frac{28}{46} \cdot \mathcal{H}\left(\frac{7}{28}, \frac{6}{28}, \frac{7}{28}, \frac{5}{28}, \frac{3}{28}\right)}_{\text{orange module}}$$

$$\approx 0.13 \text{ bits} + 1 \text{ bit} + 1.38 \text{ bits} = 2.51 \text{ bits}. \tag{25}$$

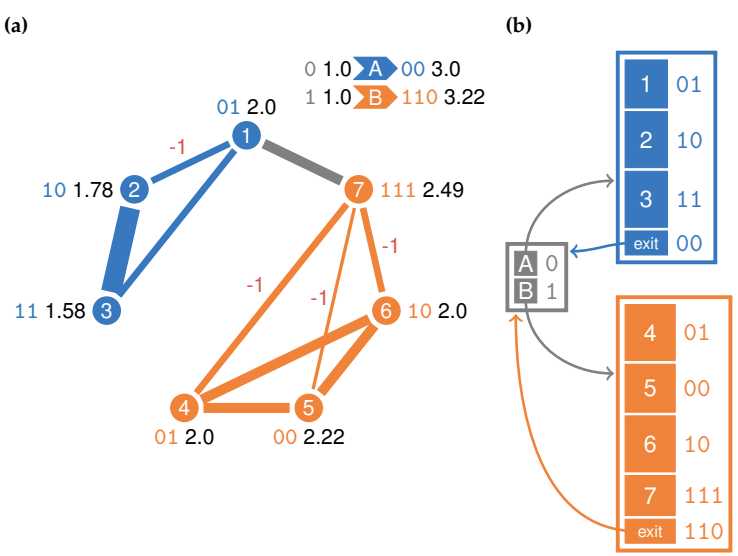


Figure 24: (a) The same network as in Figure 23, but now with incomplete observations: the link labels indicate the amount of missing observations; links without labels have the same weights as before. (b) A new coding scheme with new codewords for the old partition.

We begin to wonder whether M_2 describes $G_\circ$ best — it does not. Consider the partition $\mathsf{M}_3 = \{\{2,3\},\{4,5,6\},\{1,7\}\}$ with

three modules instead of two (Figure 25), and a lower codelength in $G_\circ$ than M_2,

$$\mathcal{L}(G_\circ, \mathrm{M}_3) = \underbrace{\frac{10}{46} \cdot \mathcal{H}\left(\frac{3}{10}, \frac{2}{10}, \frac{5}{10}\right)}_{\text{index level}} + \underbrace{\frac{18}{46} \cdot \mathcal{H}\left(\frac{7}{18}, \frac{8}{18}, \frac{3}{18}\right)}_{\text{blue module}} + \underbrace{\frac{22}{46} \cdot \mathcal{H}\left(\frac{7}{22}, \frac{6}{22}, \frac{7}{22}, \frac{2}{22}\right)}_{\text{orange module}} + \underbrace{\frac{16}{46} \cdot \mathcal{H}\left(\frac{6}{16}, \frac{5}{16}, \frac{5}{16}\right)}_{\text{yellow module}}$$

$$\approx 0.32 \text{ bits} + 0.58 \text{ bits} + 0.9 \text{ bits} + 0.55 \text{ bits} = 2.35 \text{ bits}, \tag{26}$$

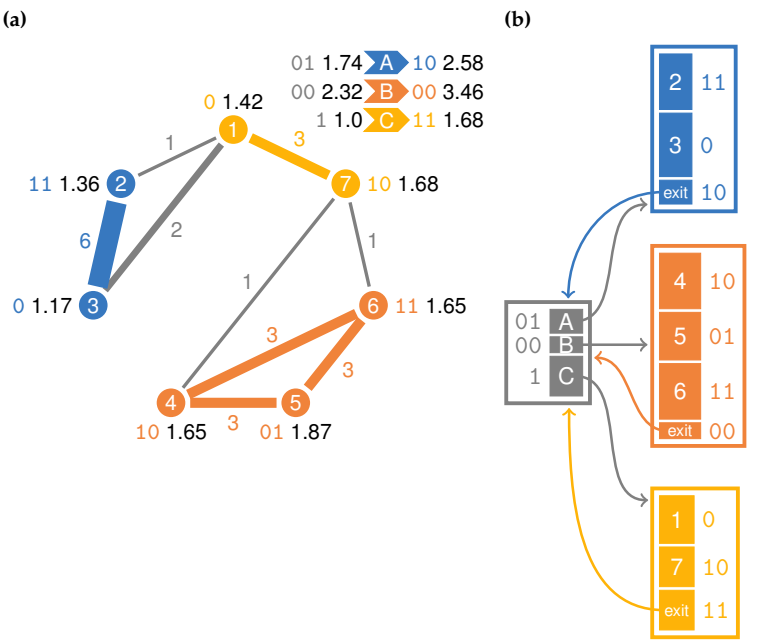


Figure 25: (a) The toy network with incomplete data, showing a three-module partition with lower codelength than the two-module partition which was optimal under complete data. The link weights correspond to the original weights (Figure 23) minus the missing observations (Figure 24). (b) The coding scheme for the three-module partition.

In $G_\bullet$, however, M_3 has a higher codelength than M_2,

$$\mathcal{L}(G_\bullet, \mathrm{M}_3) = \underbrace{\frac{18}{54} \cdot \mathcal{H}\left(\frac{4}{18}, \frac{5}{18}, \frac{9}{18}\right)}_{\text{index level}} + \underbrace{\frac{20}{54} \cdot \mathcal{H}\left(\frac{8}{20}, \frac{8}{20}, \frac{4}{20}\right)}_{\text{blue module}} + \underbrace{\frac{28}{54} \cdot \mathcal{H}\left(\frac{8}{28}, \frac{7}{28}, \frac{8}{28}, \frac{5}{28}\right)}_{\text{orange module}} + \underbrace{\frac{24}{54} \cdot \mathcal{H}\left(\frac{7}{24}, \frac{8}{24}, \frac{9}{24}\right)}_{\text{yellow module}}$$

$$\approx 0.5 \text{ bits} + 0.56 \text{ bits} + 1.02 \text{ bits} + 0.7 \text{ bits} = 2.78 \text{ bits}, \tag{27}$$

so that we would not detect M_3 in $G_\bullet$.

The toy example illustrates what happens when data is incomplete: the network gets fragmented into more and smaller communities because we cannot calculate the nodes' visit rates accurately, and module entry and exit rates get distorted [2]. Partitions that would be sub-optimal under complete data can achieve a lower codelength than the network's true community structure[40]. However, we would like to detect the true community structure even when data is missing, or at least a partition that is as close as possible to the network's true structure. The fundamental reason why this does not work out-of-the box is that the map equation describes the provided network as-is, implicitly assuming that the data is complete [2].

[40] Assuming that there is such a thing as a true community structure.

Updating our Assumptions

When modelling a real-world system, we should ask ourselves whether it is reasonable to assume complete data. Often, this is not the case. We rely on observations to derive link weights, but it may be prohibitively expensive or time-consuming to ensure complete data so that, instead, we collect a "good enough" sample[41]. Even networks that seem to be complete from one point of view can be incomplete from another. For example, one could argue that online social networks are complete because all friendship relations between their users can be retrieved from a database. However, not all users who are real-life friends are necessarily connected online; some links that should exist are missing. On the other hand, acquaintances who do not consider each other friends in real life may be friends online; we observe some links that should not be there. We can think about links that should exist, but do not, as false negatives, and links that should not exist, but do, as false positives, obtained through an imprecise measuring instrument.

[41] What "good enough" means depends on the research question and is impossible to quantify in general.

In this chapter, we interpret networks as directed[42] multigraphs where link weights represent the number of times we have observed each link. We assume that observed link weights can be smaller than or equal to their true values, that is, we can have false negatives, but no false positives. However, observing

[42] For simplicity, we use undirected networks in the examples.

a lower weight for a link (u, v) means that we underestimate the rate $t_{u,v}$ at which a random walker transitions from u to v, which can lead to overestimating the transition rates between other node pairs. To avoid this, we regularise the map equation with an empirical Bayes estimate of the transition rates. We adapt the so-called continuous configuration model [60] to calculate prior link weights based on the observed link weights, and combine those weights with a fully-connected network for teleportation, corresponding to an average over the ensemble of Erdős-Rényi random networks [22], representing our expectation so see uniformly distributed links in absence of data. The probability to teleport through the prior network depends both on the random walker's current node and its target node: teleporting from nodes with more observations is less likely, while choosing target nodes with more observations is more likely, reflecting the idea that more data gives us higher confidence in the observed link weights. When the community structure in the observed part of the data stands out clearly enough compared to the fully connected prior, we hope to detect the true community structure. If it does not, the prior should prevent the map equation from detecting modular structure and, instead, return the one-level partition.

Teleportation to the Rescue

In directed networks, we need to work with teleportation to guarantee ergodic transition rates between nodes, and one of the simplest approaches to do so is uniform node teleportation.

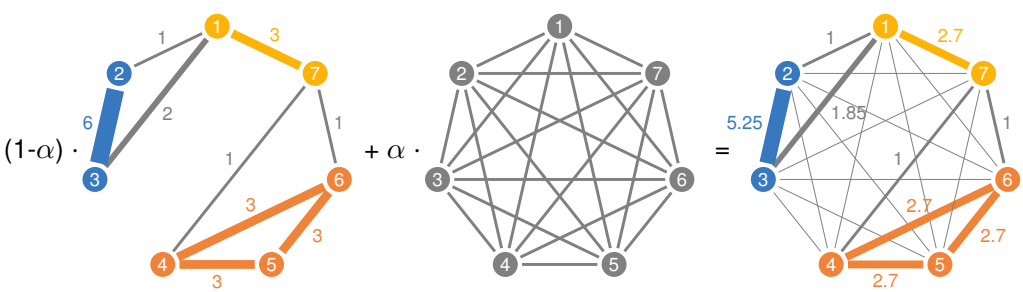


Figure 26: An attempt to address incomplete data with uniform node teleportation: the random walker follows observed links with probability $1 - \alpha$ (left), and teleports with probability α to a node chosen uniformly at random (middle). In the combined network (right), M_3 still has a lower codelength than M_2. Links in the middle have weight 1, and unlabelled links on the right weight $\alpha = 0.15$.

Using uniform node teleportation, the random walker teleports with fixed probability α to a node chosen uniformly at random, and follows observed links with probability $1-\alpha$ at every step [31]. We combine the example network with incomplete observations with a uniform node teleportation approach (Figure 26), and check the resulting codelengths for M_2 and M_3. For uniform node teleportation with teleportation rate α, we denote the codelength as $\mathcal{L}_*^{\alpha}$, use decimal numerals in the fractions for clarity, and do not explicitly normalise the inputs to the entropy function[43]. For $\alpha = 0.15$, the two-module partition M_2 has codelength

[43] In this chapter, we adopt the convention that the inputs to the entropy function are implicitly normalised by their sum, interpreting $\mathcal{H}(x,y)$ as $\mathcal{H}\left(\frac{x}{x+y}, \frac{y}{x+y}\right)$

$$\begin{aligned}\mathcal{L}_*^{0.15}(G_\circ, \mathsf{M}_2) = &\underbrace{\frac{8.7}{45.4} \cdot \mathcal{H}(4.35, 4.35)}_{\text{index level}} \\ &+ \underbrace{\frac{24.9}{45.4} \cdot \mathcal{H}(6, 6.85, 7.7, 4.35)}_{\text{blue module}} + \underbrace{\frac{29.2}{45.4} \cdot \mathcal{H}(6.85, 6, 6.85, 5.15, 4.35)}_{\text{orange module}} \\ &\approx 2.75 \text{ bits}, \end{aligned} \tag{28}$$

and the three-module partition M_3 has codelength

$$\begin{aligned}\mathcal{L}_*^{0.15}(G_\circ, \mathsf{M}_3) \approx &\underbrace{0.29 \cdot \mathcal{H}(3.9, 3.5, 5.75)}_{\text{index level}} + \underbrace{0.41 \cdot \mathcal{H}(6.85, 7.7, 3.9)}_{\text{blue module}} \\ &+ \underbrace{0.51 \cdot \mathcal{H}(6.85, 6, 6.85, 3.5)}_{\text{orange module}} + \underbrace{0.37 \cdot \mathcal{H}(6, 5.15, 5.75)}_{\text{yellow module}} \\ &\approx 2.65 \text{ bits}, \end{aligned} \tag{29}$$

so we prefer M_3 over M_2 as a description of the network's structure. In our example, uniform node teleportation could not compensate for the incomplete data. In general, for low teleportation rates, the observed network structure dominates over the teleportation such that the map equation can overfit to the noise. Conversely, for high teleportation rates, random teleportation jumps dominate over the observed network structure, and the map equation can underfit without detecting relevant community structure [2].

Therefore, instead of adopting a uniform teleportation scheme, we take a different approach and use the observed data to de-

rive a prior teleportation network and node-dependent teleportation rates α_u. Essentially, every piece of observed data provides more information and enables us to make a better choice when it comes to the prior network and teleportation rates. We calculate the link weights in the prior network as[44]

$$\gamma_{u,v} = \lambda_{u,v} \cdot \frac{\sum_{n \in V} k_n^{\text{in}} + k_n^{\text{out}}}{\sum_{n \in V} s_n^{\text{in}} + s_n^{\text{out}}} \cdot \frac{s_u^{\text{out}} s_v^{\text{in}}}{k_u^{\text{out}} k_v^{\text{in}}}, \tag{30}$$

where $\lambda_{u,v}$ is a connectivity parameter, k_u^{in} and k_u^{out} are node u's in and out degrees, and $s_u^{\text{in}} = \sum_{v \in V} w_{v,u}$ is node u's in-strength, and $s_u^{\text{out}} = \sum_{v \in V} w_{u,v}$ is node u's out-strength, respectively. We choose $\lambda_{u,v} = \frac{\ln|V|}{|V|}$, which corresponds to the connectivity threshold of random networks [22], that is, a network where each link exists independently with uniform probability $\frac{\ln|V|}{|V|}$ is almost surely connected. The node-dependent teleportation rates are

$$\alpha_u = \frac{\sum_{v \in V} \gamma_{u,v}}{\sum_{v \in V} w_{u,v} + \gamma_{u,v}}, \tag{31}$$

that is, the random walker follows links in the observed and prior network proportional to the current node's observed and prior out-strength, respectively. Based on Equation (30) and Equation (31), we construct a prior network for the example network and use node-dependent teleportation rates (Figure 27).

[44] For now, we simply state our solution and delay explaning where it comes from until the next section.

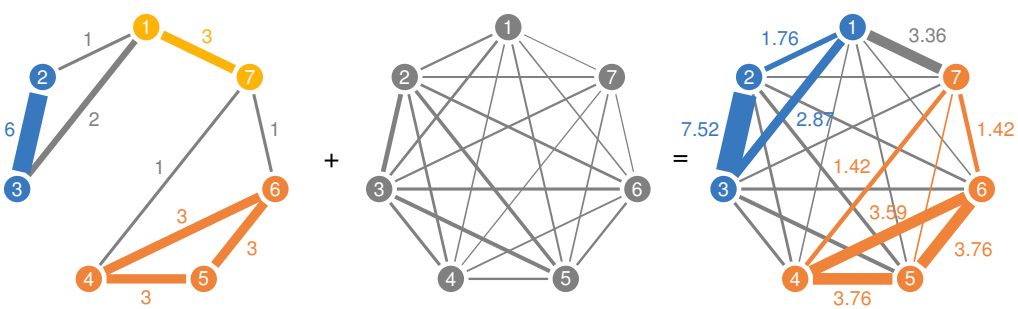


Figure 27: With a node-dependent teleportation scheme, we recover M_2 as a better description than M_3.

We denote the codelength under node-dependent teleportation as $\mathcal{L}_\gamma$. With node-dependent teleportation, the two-module partition M_2 has codelength

$$\mathcal{L}_\gamma(G_\circ, \mathsf{M}_2) \approx \underbrace{\frac{25.28}{78.61} \cdot \mathcal{H}(12.64, 12.64)}_{\text{index level}} + \underbrace{\frac{49.59}{78.61} \cdot \mathcal{H}(9.66, 12.84, 14.45, 12.64)}_{\text{blue module}} + \underbrace{\frac{54.3}{78.61} \cdot \mathcal{H}(11.19, 11.17, 11.19, 8.11, 12.64)}_{\text{orange module}} \approx 3.17 \text{ bits}, \tag{32}$$

and the three-module partition M_3 has codelength

$$\mathcal{L}_\gamma(G_\circ, \mathsf{M}_3) \approx \underbrace{\frac{34.61}{78.61} \cdot \mathcal{H}(12.25, 11.31, 11.05)}_{\text{index level}} + \underbrace{\frac{39.54}{78.61} \cdot \mathcal{H}(12.84, 14.45, 12.25)}_{\text{blue module}} + \underbrace{\frac{44.86}{78.61} \cdot \mathcal{H}(11.19, 11.17, 11.19, 11.31)}_{\text{orange module}} + \underbrace{\frac{28.82}{78.61} \cdot \mathcal{H}(9.66, 8.11, 11.05)}_{\text{yellow module}} \approx 3.2 \text{ bits}, \tag{33}$$

such that we recover M_2 as a better description of the network's structure than M_3.

Teleporting with Empirical Bayes

For a principled derivation of the regularised node transition rates, we adopt a Bayesian approach [2, 86]. Let $T_u = [t_{u,v}]_{v \in V}$ be the hidden vector of true transition probabilities from node u to nodes $v \in V$. Our goal is to determine T_u to address the incomplete observations problem. The observed weights vector $W_u = [w_{u,v}]_{v \in V}$ can be understood as a sample from T_u.

We introduce a prior distribution over T_u to calculate an estimate of the true transition rates,

$$\hat{t}_{u,v} = \int t_{u,v} P(T_u | W_u)\, dT_u. \tag{34}$$

$P(T_u|W_u)$ is the posterior of the unknown distribution over T_u, given the observed weights W_u, defined by Bayes' rule as

$$P(T_u|W_u) = \frac{P(W_u|T_u)\, P(T_u)}{P(W_u)}. \tag{35}$$

We choose a Dirichlet prior for T_u,

$$P\left(T_u|\gamma_u\right) = \frac{\Gamma\left(\sum_{v\in V}\gamma_{u,v}\right)}{\prod_{v\in V}\Gamma\left(\gamma_{u,v}\right)}\prod_{v\in V} t_{u,v}^{\gamma_{u,v}-1}, \tag{36}$$

where Γ is the gamma function, and $\gamma_{u,v}$ are distribution parameters. Given T_u, the likelihood of the observed data W_u is

$$P\left(W_u|T_u\right) = \left(\sum_{v\in V} w_{u,v}\right)! \prod_{v\in V}\frac{t_{u,v}^{w_{u,v}}}{w_{u,v}!}, \tag{37}$$

and the total probability of the data is

$$P\left(W_u\right) = \int P\left(W_u|T_u\right) P\left(T_u\right) dT_u. \tag{38}$$

The posterior of T_u, given the observed data W_u and parameters $\gamma_{u,v}$ is

$$P\left(T_u|W_u,\gamma_u\right) \propto \prod_{v\in V} t_{u,v}^{w_{u,v}+\gamma_{u,v}-1}. \tag{39}$$

After plugging Equations (37) to (39) into Equation (35), we can integrate Equation (34), and get

$$\begin{aligned}\hat{t}_{u,v} &= \frac{w_{u,v}+\gamma_{u,v}}{\sum_{v\in V} w_{u,v}+\gamma_{u,v}} \\ &= \left(1-\alpha_u\right)\frac{w_{u,v}}{\sum_{v\in V} w_{u,v}} + \alpha_u\frac{\gamma_{u,v}}{\sum_{v\in V}\gamma_{u,v}},\end{aligned} \tag{40}$$

where we interpret

$$\alpha_u = \frac{\sum_{v\in V}\gamma_{u,v}}{\sum_{v\in V} w_{u,v}+\gamma_{u,v}} \tag{41}$$

as node-dependent teleportation probabilities. The random walker follows links in the observed network with probability $1-\alpha_u$, and links in the prior network with probability α; teleportation probabilities depend both on the source and target node.

We define the parameters $\gamma_{u,v}$ as

$$\gamma_{u,v} = \lambda_{u,v}\cdot c_{u,v}, \tag{42}$$

where $\lambda_{u,v}$ is a connectivity parameter, and $c_{u,v}$ reflects our prior belief about the links' weights. In unipartite networks, we choose a uniform connectivity parameter $\lambda_{u,v} = \lambda = \frac{\ln|V|}{|V|}$, which is the threshold at which Erdős-Rényi random networks are almost

surely connected [22]. To calculate expected link weights, we use the so-called continuous configuration model [60],

$$c_{u,v} = \frac{\sum_{n\in V} k_n^{\text{in}} + k_n^{\text{out}}}{\sum_{n\in V} s_n^{\text{in}} + s_n^{\text{out}}} \cdot \frac{s_u^{\text{out}} s_v^{\text{in}}}{k_u^{\text{out}} k_v^{\text{in}}}, \tag{43}$$

where k_u^{in} and k_u^{out} are node u's in and out degrees, and s_u^{in} and s_u^{out} are node u's in and out strengths, respectively. In case of undirected networks, we have $k_u^{\text{in}} = k_u^{\text{out}}$ and $s_u^{\text{in}} = s_u^{\text{out}}$ for all u. Equation (43) preserves the expected weights of in- and out-links incident to a node, and places higher weights between pairs of nodes with stronger connections.

Connectivity in Bipartite and Metadata Networks

[45] As a brief reminder, bipartite networks have two sets of nodes that we called L and R.

[46] Metadata for nodes that represent persons could be, for example, age, place of birth, citizenship, ...

Node types in bipartite networks[45] and metadata labels in node-annotated networks[46] provide information that we can use to adjust the connectivity parameter $\lambda_{u,v}$.

Following the constraint that only different-type nodes in bipartite networks can be connected, we choose the connectivity

$$\lambda_{LR} = \frac{\ln(|L| + |R|)}{\min(|L|, |R|)}, \tag{44}$$

which is the threshold at which random bipartite networks are almost surely connected [69], and set

$$\gamma_{u,v}^{\text{bi}} = (1 - \delta_{\tau_u,\tau_v})\, \lambda_{LR} c_{u,v}, \tag{45}$$

where δ is the Kronecker delta, and τ_u and τ_v are the types – left or right – of nodes u and v, respectively.

In unipartite networks with metadata, we assume that nodes follow the homophily principle [51] and prefer to connect with other nodes belonging to the same metadata class. As before, we connect each pair of nodes uniformly with connectivity $\lambda = \frac{\ln|V|}{|V|}$, but, in addition, reinforce the connectivity between nodes with the same label m by $\lambda_m = \frac{\ln n_m}{n_m}$, where n_m is the number of nodes labelled with m. With metadata labels m_u and m_v for nodes u and v, we adjust the prior link weight to

$$\gamma_{u,v}^{\text{meta}} = (\lambda + \delta_{m_u,m_v} \lambda_{m_u})\, c_{u,v}. \tag{46}$$

Modelling Incomplete Data Prevents Overfitting

We can compensate for small to moderate[47] amounts of missing data with the regularised map equation and still detect communities that describe the network's structure well. When large amounts of data are missing such that there is not enough evidence to support communities, the regularised map equation returns the one-level partition.

We compare how the standard and regularised map equation behave under incomplete observations by applying them to a synthetic and empirical networks where we randomly remove different r-fractions of the data. For each r, we measure how many modules each version of the map equation detects, and how well the partitions characterise the networks' community structure.

[47] How much missing data we can handle depends on the specifics of the network and how many link observations are left to work with.

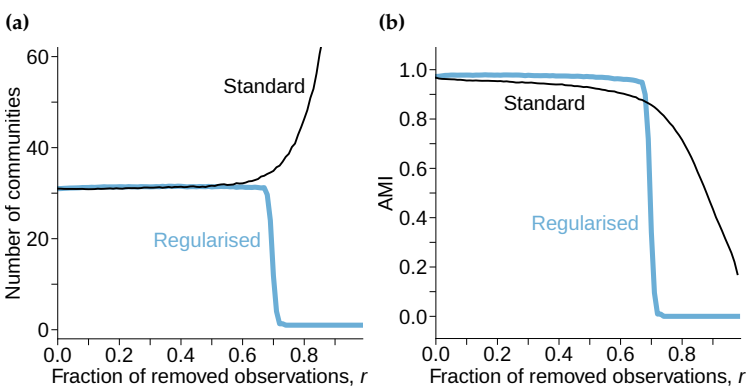


Figure 28: With the regularised map equation, we compensate for incomplete observations up to a point where 70% of the data in an LFR network is missing. The results are averages over 100 samples.

In a synthetic Lancichinetti-Fortunato-Radicchi (LFR) network with 1000 nodes, average node degree 7, mixing parameter[48] 0.4, and known community structure with 31 modules, the regularised map equation compensates up to a point where 70% of the data is missing (Figure 28). It detects the correct number of 31 communities up to approximately $r = 0.7$, and returns the one-level partitions beyond that. The standard map equation begins to detect more than 31 communities already around $r = 0.5$ (Figure 28(a)).

Since we know the planted community structure, we can use adjusted mutual information (AMI) [84] to measure how well

[48] Mixing quantifies the fraction of links that connect nodes in different modules. For example, a mixing of 0.1 means that 10% of the links connect nodes in different module, and 90% connect nodes in the same module.

the detected communities align with the ground truth: the AMI score for the standard map equation continues to decrease as r increases, whereas it remains stable close to 1 for the regularised map equation up to around $r = 0.7$. While the standard map equation may detect the correct number of modules, it assigns some nodes to the wrong module; the regularised map equation corrects this problem (Figure 28(b)).

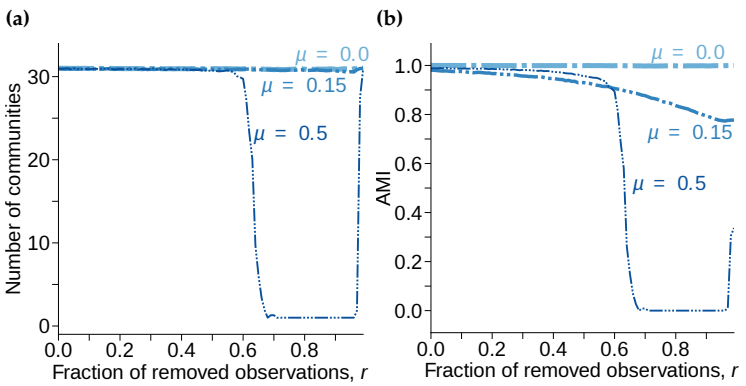


Figure 29: With correct metadata labels, the regularised map equation avoids overfitting even for large amounts of missing data. For large amounts of missing data and mislabelled nodes, it return the one-level partition.

We use the same synthetic network to test to what extent metadata helps to compensate for missing data, and assign metadata labels to the nodes based on their ground truth community membership. However, incorrect metadata labels can be a source of further uncertainty, which we simulate by assigning random labels to a μ-fraction of the nodes. The regularised map equation detects the correct number of modules as longs as a large fraction of the nodes is labelled correctly, but assigns some nodes to the wrong modules. For large amounts of missing data and mislabelled nodes, it returns the one-level partition (Figure 29).

We also test how the standard and regularised map equation perform in three[49] real-world networks with metadata labels: a dataset of citations between computer science research articles (CoRA) where we use research topics as metadata [47], a network of Pokémon where any two Pokémon who share the same ability are connected and we use their primary type as metadata [6], and a network of airports outside the United States of America that are connected by regular flights where we use the airports' country as metadata [58]. The networks' sizes and

[49] Paper II contains results for six real-world networks [2].

Name	$\lvert V\rvert$	$\lvert E\rvert$	m	Kind
CoRA	3,385	22,902	9	Directed
Pokémon	743	18,184	18	Undirected
Openflights	964	8,850	97	Directed

Table 6: Details of three real-world networks with metadata where $|V|$ is the number of nodes, $|E|$ the number of links, and m the number of metadata categories.

number of metadata categories are listed in Table 6.

In the real-world networks, too, the regularised map equation reduces overfitting such that the number of detected communities remains relatively stable for different amounts of missing data. The point at which it returns the one-level partition differs depending on the network. The results suggest that metadata labels can help in uncovering the networks community structure, such as in the Pokémon and Openflights datasets, while they may represent a too strong prior in, for example, the CoRA dataset (Figure 30).

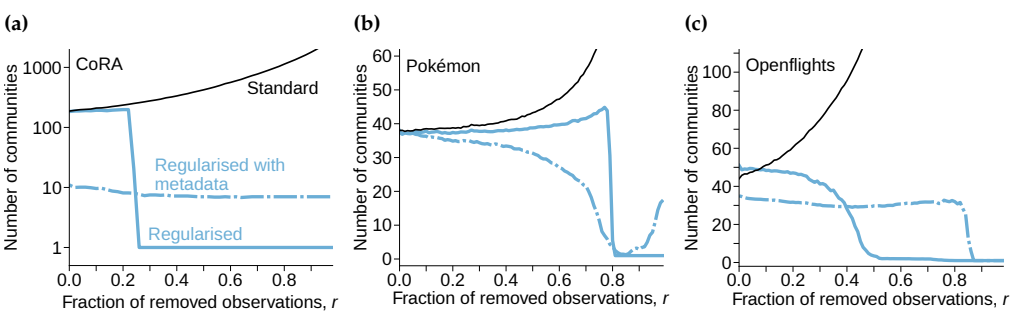


Figure 30: Number of communities detected by the regularised and standard map equation in three real-world networks. The regularised map equation reduces the impact of missing data, and detects no communities when we remove too much data.

Since no ground truth communities for the real-world networks are known, we take a different approach to quantify how well we can describe their structure: first, we remove an r-fraction of the links. Second, we split the remaining links into equally sized train and test networks, G_{train} and G_{test}, and detect communities in the train network. Third, we use the communities from the train network to partition the test network, and measure the codelength savings compared to the one-level partition[50]. Let M be the partition detected in the train network, and M_1 be the one-level partition; then we calculate codelength savings achieved by M as $1 - \frac{\mathcal{L}(\mathsf{M}, G_{\text{test}})}{\mathcal{L}(\mathsf{M}_1, G_{\text{test}})}$. Positive codelength savings indicate that the partition detected in the train network

[50] We need to keep in mind that this approach reduces the amount of available data so that we cannot compare the results with those from the previous analysis.

captures structure that is present in the test network, simply because it provides a better description of the data than the one-level partition. Conversely, partitions that lead to negative codelength savings capture patterns that are not present in the test network. Both the standard and regularised map equation de-

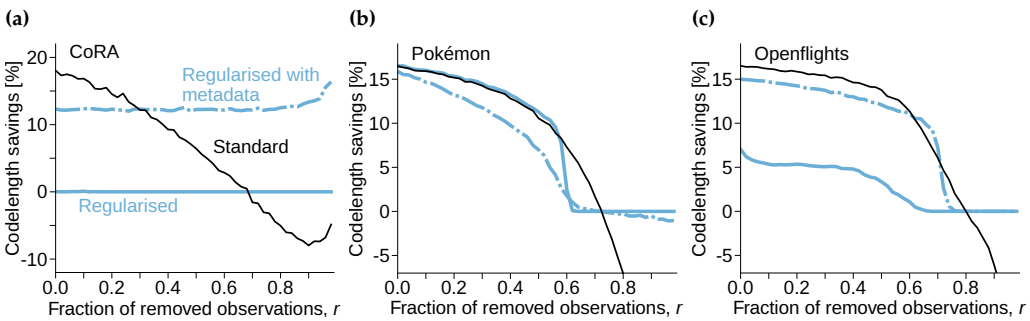


Figure 31: Codelength savings for the regularised and standard map equation in three real-world networks. The standard map equation detects useful structure when small to moderate amounts of data are missing, but overfits for large amounts of missing data. In contrast, the regularised map equation reduces overfitting and returns the one-level partition when too much data is missing.

tect useful communities when small to moderate amounts of the data are missing. However, when large amounts are missing, the standard map equation finds spurious communities, leading to negative codelength savings. In contrast, the regularised map equation returns the one-level partition when too much data is missing, and, thereby, reduces overfitting (Figure 31).

Future Directions

In this chapter, we have only considered incomplete observations, but in practice, networks may also involve spurious observations where link weights are higher than they should be, caused by, for example, a malfunctioning measuring instrument, or by human mistakes. We could generalise our approach to address this situation by making the right prior assumptions regarding how the data is distributed.

Another generalisation could be to assume specific patterns regarding how incomplete observations are distributed. Perhaps some nodes are more, or less prone to be involved in missing links and, again, by making the right prior assumptions, we could address this case.

Community-Aware Centrality

> There are more things in heaven
> and earth, Horatio, than are
> dreamt of in your philosophy.
>
> *Hamlet*
> by William Shakespeare

The map equation's original purpose is to detect communities in complex networks. It helps us understand how networks are organised, and allows us to represent them as networks of modules that interact with each other. But community detection is not the end of the story. In many real-life applications, we also want to compare nodes with each other and rank them according to their importance (Figure 32). For example, when we search the web for information, we want to select the most relevant website for our search query. In infrastructure networks, we want to identify what components have the most impact when they fail so we can secure them with backups. To devise effective vaccination strategies, we need to find those people in a contact network who would drive the spread of a disease.

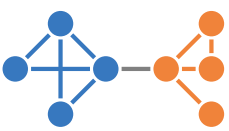


Figure 32: Example network with eight nodes and ten links. Which of the nodes are most important, and how does their community membership contribute to their importance?

How can we use the map equation to determine how important a node is? We look through the coding-lens again and consider how much a node's presence affects the codewords assigned to the remaining nodes compared to if it was not present. We assume that the larger the effect a node has, the more important it is. Combining the concept of network vitality, the Vickrey-Clarke-Groves (VCG) principle for setting prices in multi-item auctions, and the coding principles behind the map equation, we define node u's importance as the codelength difference be-

tween (i) the optimal coding scheme that assigns codewords to all nodes but never uses the codeword for u, and (ii) the optimal coding scheme that assigns codewords to all nodes but u [3].

Network Vitality

Our idea to relate a node's importance to how its presence affects the coding scheme is inspired by the concept of network vitality. With respect to some function f that operates on graphs and calculates a numerical value, the vitality $\mu(G, u)$ for node u is

$$\mu(G, u) = f(G) - f(G - \{u\}), \tag{47}$$

where $G - \{u\}$ denotes G with node u removed [40]. We can easily extend network vitality to include communities,

$$\mu(G, \mathsf{M}, u) = f(G, \mathsf{M}) - f(G - \{u\}, \mathsf{M} - \{u\}), \tag{48}$$

where $\mathsf{M} - \{u\}$ denotes the partition M with node u removed, and f is now a function that operates on pairs of graphs and partitions. Then, we could set $f = \mathcal{L}$ and use the map equation to calculate node importances[51].

[51] This idea was used to define modularity vitality [48], a community-aware centrality score based on the modularity function [54].

However, there is a problem with deleting a node from the network: the random walk's dynamics change. In general, this affects the network's community structure because node visit and transition rates change. Equation (48) addresses this issue by fixing the partition, but with the map equation, we can take a more subtle approach: we keep the network unchanged, and only omit the respective node's codeword when describing the network — we call this silencing a node.

Second-Price Auctions and the VCG Principle

Imagine you are bidding for a painting in an auction. How much should you bid? The answer depends on how much you think the painting is worth, how many other bidders are involved, and the format of the auction. But clearly, you should not bid more than you think the painting is worth and, ideally, you want to buy it for less than that. The painting's selling price depends

on how much you and others bid, and is determined by the auction's rules.

In so-called second-price sealed-bid auctions, the participants submit bids for a single item simultaneously and secretly, and the item is sold to the person with the highest bid for a price equal to the second-highest bid [83]. That is, a person's bid only determines whether they win the auction, but how much they have to pay depends on the other participants' bids[52].

The VCG principle generalises this idea to the multi-item case, such as AdWords auctions, and determines the price that bidder b pays for item i as the marginal harm caused to other bidders who, because of b's existence, receive an item $j \neq i$ that they value lower than i [43]. Figure 33 shows a scenario with three bidders, three items, the bidders' valuations for the items, and who receives which item. We assume that all bidders make bids according to their valuations, that they are willing to pay no more than their valuations, and are interested in purchasing at most one item. What should the prices be? Figure 34 shows which items B and C would receive if A did not exist: B and C would get items ★ and ■, which they value at 4 and 1, instead of ■ and ▲, which they value at 2 and 0, respectively. Together, they would be better off by a value of (4-2) + (1-0) = 3, which is A's price for ★. B's price for ■ is 3-1 = 2 because B only causes harm to C, who would get ■ instead of ▲ if B did not exist. C's price for ▲ is 0 because C causes no harm to anyone[53]. Importantly, these prices do not depend on the bidders own valuations or bids, but on the harm they cause to others.

[52] For second-price auctions, the best strategy is to bid what you think the item is worth.

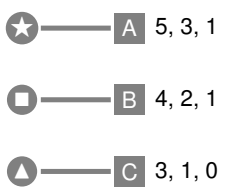


Figure 33: Three bidders A, B, and C are bidding on three items ★, ■, and ▲. The bidders' item valuations are shown to their right in the same order as the items, and links show who gets which item.

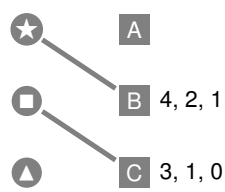


Figure 34: Without bidder A, B gets ★, and C gets ■.

[53] We can interpret a price of 0 as some arbitrary baseline and all other amounts as offsets from that baseline.

Applying the VCG Principle to Coding

We use the VCG principle to assign importances to nodes based on the total additional codeword length their presence causes for the remaining nodes. As an example, we take the network G and partition $\mathsf{M} = \{\{1,2,3,4\},\{5,6,7,8\}\}$ as shown in Figure 35(a), and calculate the importance for node 5. To begin with, we design a coding scheme where all nodes have codewords, and all codewords are used (Figure 35(b)). We describe the random walk sequence through nodes 2 1 3 4 5 6 5 8 with the codeword

sequence 0 10 00 110 01 111 1 0 100 0 110, which is 22 bits long. The codelength for M is

$$\mathcal{L}(G,\mathsf{M}) = \underbrace{\frac{2}{20}\cdot\mathcal{H}\left(\frac{1}{2},\frac{1}{2}\right)}_{\text{index level}} + \underbrace{\frac{12}{20}\cdot\mathcal{H}\left(\frac{3}{12},\frac{2}{12},\frac{2}{12},\frac{4}{12},\frac{1}{12}\right)}_{\text{blue module}} + \underbrace{\frac{10}{20}\cdot\mathcal{H}\left(\frac{4}{10},\frac{2}{10},\frac{2}{10},\frac{1}{10},\frac{1}{10}\right)}_{\text{orange module}}$$
$$\approx 2.47 \text{ bits}, \tag{49}$$

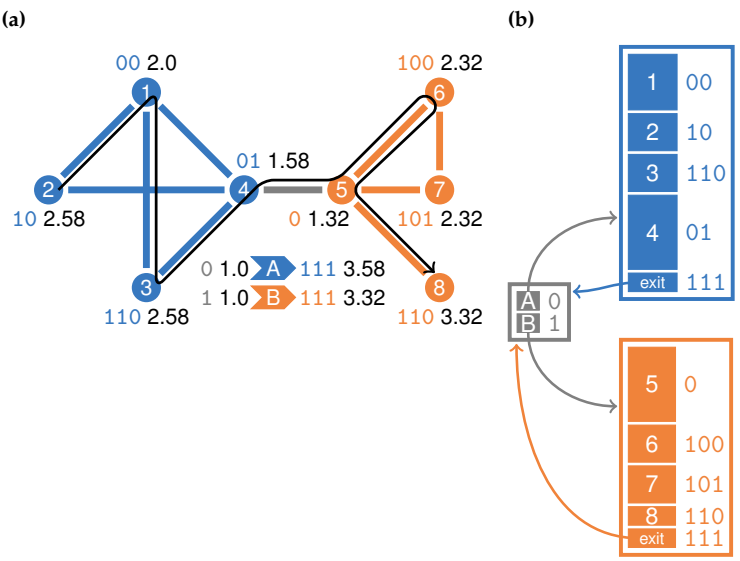


Figure 35: (a) A network with eight nodes, ten links, two communities, and (b) its coding scheme. All the nodes have codewords, and all codewords are used when the random walker visits the respective node.

[54] We only omit visits to node 5, but encode module entries and exits through 5.

[55] Note that the inputs to the entropy of the orange module are *not* normalised, and we do not assume normalisation by convention in this chapter.

If we silence node 5 by simply not using its codeword[54], we would describe the sequence as 0 10 00 110 01 111 1 100 110, and use only 20 bits (Figure 36). The codelength for using the same coding scheme but silencing node 5 is[55]

$$\mathcal{L}^{u}(G,\mathsf{M}) = \underbrace{\frac{2}{20}\cdot\mathcal{H}\left(\frac{1}{2},\frac{1}{2}\right)}_{\text{index level}} + \underbrace{\frac{12}{20}\cdot\mathcal{H}\left(\frac{3}{12},\frac{2}{12},\frac{2}{12},\frac{4}{12},\frac{1}{12}\right)}_{\text{blue module}} + \underbrace{\frac{10}{20}\cdot\mathcal{H}\left(\frac{2}{10},\frac{2}{10},\frac{1}{10},\frac{1}{10}\right)}_{\text{orange module}}$$
$$\approx 2.21 \text{ bits}. \tag{50}$$

However, assigning a codeword but never using it is inefficient. Node 5 has codeword 0, which makes it impossible for any other codeword in the same module to begin with 0. We should instead design a new coding scheme that does not assign a codeword to the silenced node in the first place. That way, we can

shorten the codewords for the remaining nodes (Figure 37). With

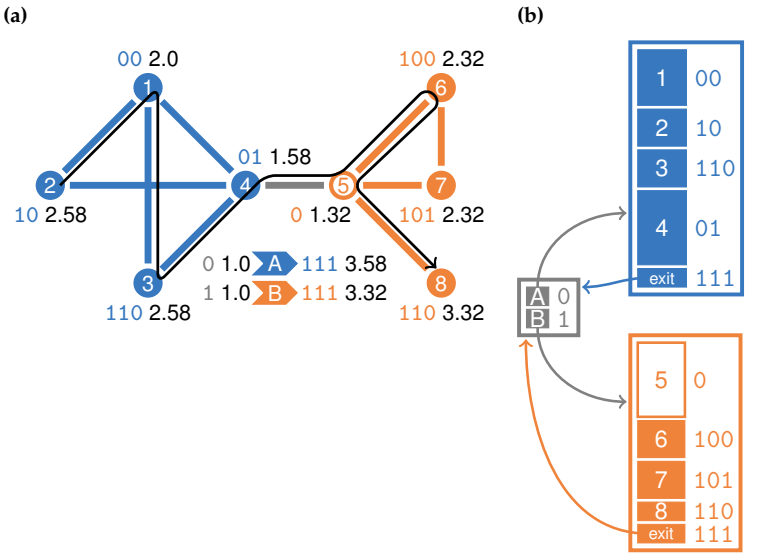


Figure 36: A coding scheme where node 5 is silenced by not using its codeword. In principle, we could encode random walker visits to node 5 because it has a codeword. Node 5 is drawn as a ring with labels to symbolise that it is silenced but has a codeword.

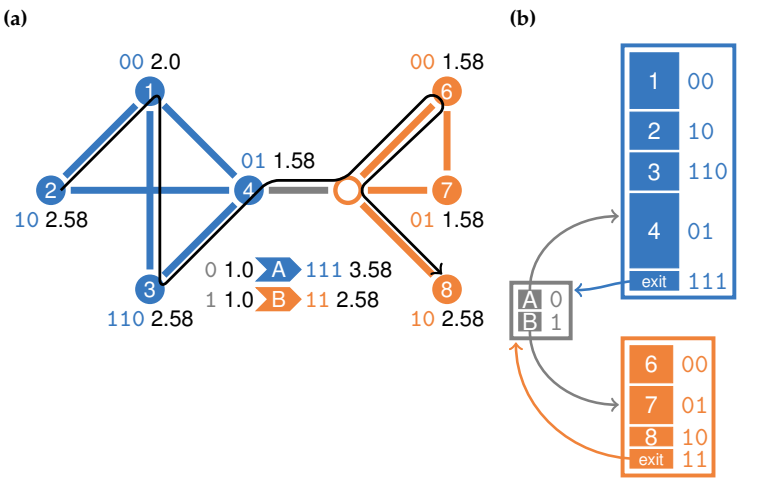


Figure 37: A coding scheme where node 5 is silenced by not assigning a codeword to it. With this coding scheme, the other nodes in the orange module are assigned shorter codewords, but it is impossible to encode random walker visits to node 5. Node 5 is drawn as a ring without labels to symbolise that it is silenced and does not have a codeword.

such a coding scheme, we describe the node sequence as `0 10 00 110 01 111 1 00 10`, using 18 bits. The codelength with the new coding scheme is

$$\mathcal{L}^{u*}(G,\mathsf{M}) = \underbrace{\frac{2}{20}\cdot\mathcal{H}\left(\frac{1}{2},\frac{1}{2}\right)}_{\text{index level}} + \underbrace{\frac{12}{20}\cdot\mathcal{H}\left(\frac{3}{12},\frac{2}{12},\frac{2}{12},\frac{4}{12},\frac{1}{12}\right)}_{\text{blue module}} + \underbrace{\frac{6}{20}\cdot\mathcal{H}\left(\frac{2}{6},\frac{2}{6},\frac{1}{6},\frac{1}{6}\right)}_{\text{orange module}}$$

$$\approx 1.99 \text{ bits.} \tag{51}$$

$\mathcal{L}^u$ (Equation (50)) corresponds to (i) the optimal coding scheme that assigns codewords to all nodes but never uses the codeword for node 5, and $\mathcal{L}^{u*}$ (Equation (51)) corresponds to (ii) the optimal coding scheme that assigns codewords to all nodes but 5 — the two ingredients to apply the VCG principle. The total harm that node 5 causes the other nodes is $\mathcal{L}^u - \mathcal{L}^{u*} = 2.21 \text{ bits} - 1.99 \text{ bits} = 0.22 \text{ bits}$. We call this measure map equation centrality.

Map Equation Centrality

To calculate map equation centrality for a node u directly, we begin by rewriting the map equation, and separate between the module that contains u and the rest of the modules,

$$\begin{aligned}\mathcal{L}(G,\mathsf{M}) &= q \cdot \mathcal{H}(Q) + \sum_{\mathsf{m}\in\mathsf{M}} p_{\mathsf{m}} \cdot \mathcal{H}(P_{\mathsf{m}}) \qquad (52)\\ &= \overbrace{q \cdot \mathcal{H}(Q)}^{\text{index level}} + \overbrace{\sum_{\substack{\mathsf{m}\in\mathsf{M}\\ \mathsf{m}\neq\mathsf{m}_u}} p_{\mathsf{m}} \cdot \mathcal{H}(P_{\mathsf{m}})}^{\text{modules without } u} - \overbrace{\sum_{p\in P_{\mathsf{m}_u}} p \log_2 \frac{p}{p_{\mathsf{m}_u}}}^{\text{module with } u}\end{aligned}$$

where m_u is the module that contains u.

We get the codelength for silencing a node u by not using its codeword by removing u from the summation in its module m_u,

$$\mathcal{L}^u(G,\mathsf{M}) = \overbrace{q \cdot \mathcal{H}(Q)}^{\text{index level}} + \overbrace{\sum_{\substack{\mathsf{m}\in\mathsf{M}\\ \mathsf{m}\neq\mathsf{m}_u}} p_{\mathsf{m}} \cdot \mathcal{H}(P_{\mathsf{m}})}^{\text{modules without } u} - \overbrace{\sum_{p\in P_{\mathsf{m}_u}\setminus\{p_u\}} p \log_2 \frac{p}{p_{\mathsf{m}_u}}}^{\text{module with } u}. \qquad (53)$$

For a coding scheme that does not assign codewords to silenced nodes, we re-normalise the codeword usage rates for the remaining nodes and module exits in m_u by $p_{\mathsf{m}_u} - p_u$,

$$\mathcal{L}^{u*}(G,\mathsf{M}) = \overbrace{qH(Q)}^{\text{index level}} + \overbrace{\sum_{\substack{\mathsf{m}\in\mathsf{M}\\ \mathsf{m}\neq\mathsf{m}_u}} p_{\mathsf{m}} H(P_{\mathsf{m}})}^{\text{modules without } u} - \overbrace{\sum_{p\in P_{\mathsf{m}_u}\setminus\{p_u\}} p \log_2 \frac{p}{p_{\mathsf{m}_u} - p_u}}^{\text{module with } u}. \qquad (54)$$

We define map equation centrality, λ, as the difference between Equation (53) and Equation (54), where the parts for the index level and the modules that do not contain u cancel out

such that node u's importance is defined by its module context m_u,

$$\begin{aligned}\lambda(G, \mathsf{M}, u) &= \mathcal{L}^u(G, \mathsf{M}) - \mathcal{L}^{u*}(G, \mathsf{M}) \\ &= -\sum_{p \in P_{\mathsf{m}_u} \setminus \{p_u\}} p \log_2 \frac{p_{\mathsf{m}_u} - p_u}{p_{\mathsf{m}_u}} \qquad (55) \\ &= -(p_{\mathsf{m}_u} - p_u) \log_2 \frac{p_{\mathsf{m}_u} - p_u}{p_{\mathsf{m}_u}}. \qquad (56)\end{aligned}$$

Equation (55) shows that all the nodes in m_u would have shorter codewords if u did not exist. We can calculate u's importance, that is the total number of bits by which the other nodes' codewords could be shortened, directly from u's and m_u's codeword usage rates (Equation (56)).

For the one-level partition M_1 where all nodes are assigned to the same module, Equations (53) and (54) simplify to

$$\mathcal{L}^u(G, \mathsf{M}_1) = -\sum_{\substack{v \in V \\ v \neq u}} p_v \log_2 p_v, \qquad (57)$$

$$\mathcal{L}^{u*}(G, \mathsf{M}_1) = -\sum_{\substack{v \in V \\ v \neq u}} p_v \log_2 \frac{p_v}{1 - p_u}, \qquad (58)$$

and the importance of node u becomes

$$\begin{aligned}\lambda(G, \mathsf{M}_1, u) &= \mathcal{L}^u(G, \mathsf{M}_1) - \mathcal{L}^{u*}(G, \mathsf{M}_1) \\ &= -(1 - p_u) \log_2 (1 - p_u). \qquad (59)\end{aligned}$$

Different Flows, Different Scores

The map equation is flexible in the sense that it can operate on top of different flow models, such as smart teleportation [42] or PageRank [31], to detect communities. In principle, we could design a specialised flow model for each network we analyse such that it represents the dynamics on the network well, and calculate the node visit and transition rates analytically or numerically [7]. Generally, different flow models lead to different node visit and transition rates and give rise to different partitions. The same applies to map equation centrality: different flow models lead to different centrality scores [3].

Table 7: Details of four real-world networks where $|V|$ is the number of nodes, $|E|$ the number of links, and m_{st} and m_{pr} the number of detected modules using the smart teleportation and PageRank flow models, respectively.

Name	$\lvert V\rvert$	$\lvert E\rvert$	p_{th}	m_{st}	m_{pr}
Facebook friends	329	1,954	0.048	21	22
Power	4,941	6,594	0.348	428	465
Physics collaborations	8,798	27,416	0.066	610	656
Google	15,763	171,206	0.001	597	600

The purpose of centrality scores is to decide how important a node is, and many different approaches exist, both community-aware and community-un-aware scores. Neither of those scores is a-priori more correct or incorrect than any other; different centrality scores simply provide different aspects on node importance, and it depends on the specific research question and scenario which centrality score is most applicable. Therefore, we evaluate map equation centrality by measuring how well it identifies the most important nodes according to two different spreading processes in different real-world networks, and compare it with a set of community-aware and community-un-aware scores.

For comparison, we use three community-aware centrality scores, modularity vitality [48], community hub-bridge [29], and community-based centrality [89], as well as two community-un-aware scores, degree centrality and betweenness centrality [40]. We apply the scores to four[56] real-world networks: a social network where nodes represent persons who are connected if they are friends on Facebook [49], a power grid network where nodes represent power stations that are connected if a power line runs between them [87], a co-authorship network between physicists who are connected if they have authored an arXiv preprint together [14], and a directed web network of internal pages at Google [59] (Table 7). We use Infomap to detect communities, both using smart teleportation and PageRank with teleportation rate 0.15, and select the partition with the shortest codelength from 1000 runs.

[56] Paper III contains results for eight more networks [3].

The first spreading process is the linear threshold model that can be used to model the adoption of ideas and behaviours, such as supporting a political party or using a certain technology [3, 64]. Initially, most nodes adopt opinion A while a small fraction

adopts opinion B. Then, we run a simulation: each node who believes A considers how many of their neighbours believe B, and if at least a t-fraction of them believe B, then the node switches to believe B as well; once nodes adopt opinion B, they are not allowed to switch to A. We repeat the simulation until no more nodes change their opinion, and measure the activation size as the fraction of nodes who have adopted opinion B. To decide which nodes start with B, we rank them by their importance according to each centrality score, and select different fractions of top-ranked nodes. The goal is to achieve a high activation size at the end of the simulation.

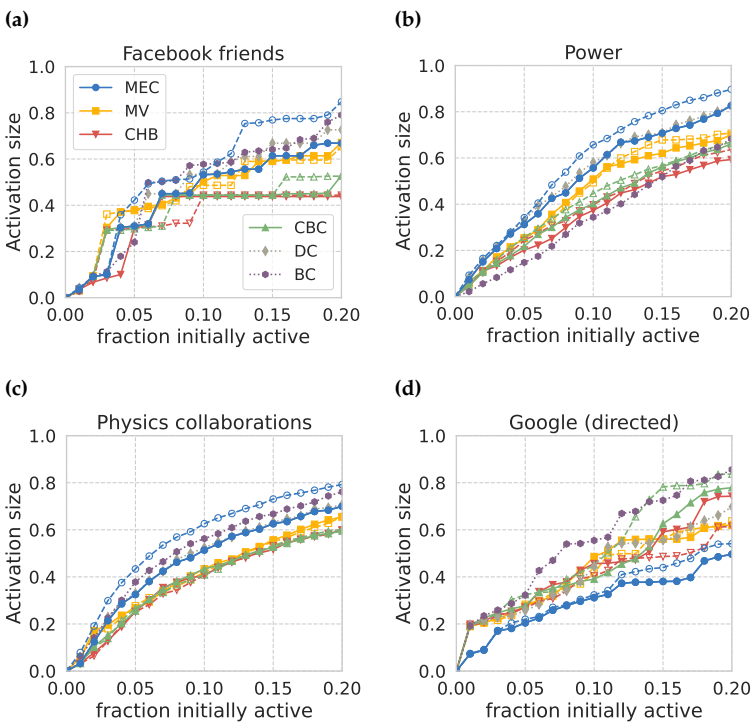


Figure 38: The activation size as the fraction of nodes that adopt opinion B at the end of a simulation following the linear threshold model. We compare the four community-aware scores map equation centrality (MEC), modularity vitality (MV), community hub-bridge (CHB), community-based centrality (CBC), and the community-un-aware scores degree centrality (DC) and betweenness centrality (BC). Solid lines and filled markers use the smart teleportation flow model, dashed lines and empty markers use standard PageRank.

For $t = 0.5$, that is, nodes adopt behaviour B if at least half of their neighbours have done so[57], we find that map equation centrality outperforms the remaining measures in three out of the four selected networks (Figure 38). The community-aware centrality scores' performance depends on which flow model we use, and is better with standard PageRank in this scenario. This

[57] Paper III contains results for $t \in \{0.4, 0.6\}$ as well.

suggests that PageRank captures the linear threshold model's dynamics better, leading to more suitable communities for identifying influential nodes in this application.

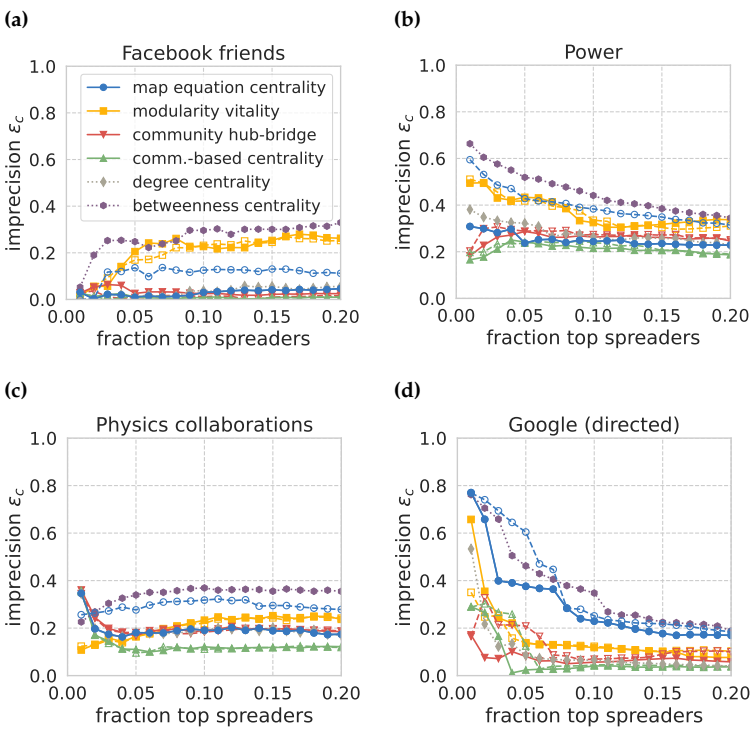


Figure 39: Imprecision for identifying the top spreaders, using SIR spreading power as ground truth. Solid lines and filled markers use the smart teleportation flow model, dashed lines and empty markers use standard PageRank.

As a second process, we consider the so-called Susceptible-Infected-Removed (SIR) disease spreading model: initially, all nodes but one are in the susceptible state, the remaining node u begins in the infected state. We run a simulation where, at each time step, infected nodes infect their susceptible neighbours uniformly and independently with probability p, then enter the removed state, and are no longer part of the simulation. We continue until no infected nodes are left, and define u's spreading power as the number of nodes in the removed state [3, 65]. Because of stochasticity, we repeat the simulation 1000 times per node, and take their average spreading power. We use centrality scores to approximate the nodes' spreading power and identify influential spreaders. We rank the nodes according to each centrality score c, select different x-fractions of top-ranked nodes,

and use the imprecision function $\epsilon_c(x) = 1 - \frac{M_c(x)}{M_{\text{SIR}}(x)}$ [39] to measure how well the selected nodes' and the SIR top spreaders' average spreading powers align. Here, $M_c(x)$ and $M_{\text{SIR}}(x)$ are the average spreading power of the top x-fraction according to centrality score c and the SIR model, respectively. As infection probability p, we use the so-called epidemic threshold $p_{\text{th}} = \frac{\langle k \rangle}{\langle k^2 \rangle - \langle k \rangle}$, where $\langle k \rangle = \frac{1}{|V|} \sum_{v \in V} k_v$ and $\langle k^2 \rangle = \frac{1}{|V|} \sum_{v \in V} k_v^2$ are the first and second moments of the network's degree distribution [85]. The goal is to achieve a low imprecision.

In this scenario, smart teleportation leads to better results than PageRank, especially for map equation centrality. The other three community-aware centrality scores are less affected (Figure 39). Map equation centrality performs well in the social, power, and co-authorship networks, but is outperformed in the web network where the other scores do better, except for betweenness centrality.

Node Distribution

Map equation centrality's performance depends on the choice of flow model because different flow models lead to different modules. The better the modules characterise the dynamics of the process we investigate — in our case the linear threshold model and the SIR disease spreading model — the more accurately we can identify important nodes. Our results (Figures 38 and 39) suggest that smart teleportation characterises the SIR disease spreading model better while standard PageRank is more aligned with the linear threshold model.

We calculate the perplexity for different fractions of top-ranked nodes to understand how the centrality scores distribute them across modules. Let M be a partition, $\mathsf{m} \in \mathsf{M}$ be a module, and let S be the set of selected nodes by some centrality measure. Then, $\frac{|\mathsf{m} \cap S|}{|S|}$ is the fraction of selected nodes in module m, and the perplexity of S is $2^{\mathcal{H}(S)}$ with $\mathcal{H}(S) = -\sum_{\mathsf{m} \in \mathsf{M}} \frac{|\mathsf{m} \cap S|}{|S|} \log_2 \frac{|\mathsf{m} \cap S|}{|S|}$. The perplexity is the effective number of same-sized modules across which the selected nodes are distributed uniformly, and a higher perplexity means that the nodes are distributed across more modules.

Our results show that the top-ranked nodes are more spread out for standard PageRank than for smart teleportation (Figure 40). We can explain this with how PageRank works: with teleportation rate r, each node receives a flow of at least $\frac{r}{|V|}$. With smart teleporation, smaller values are possible.

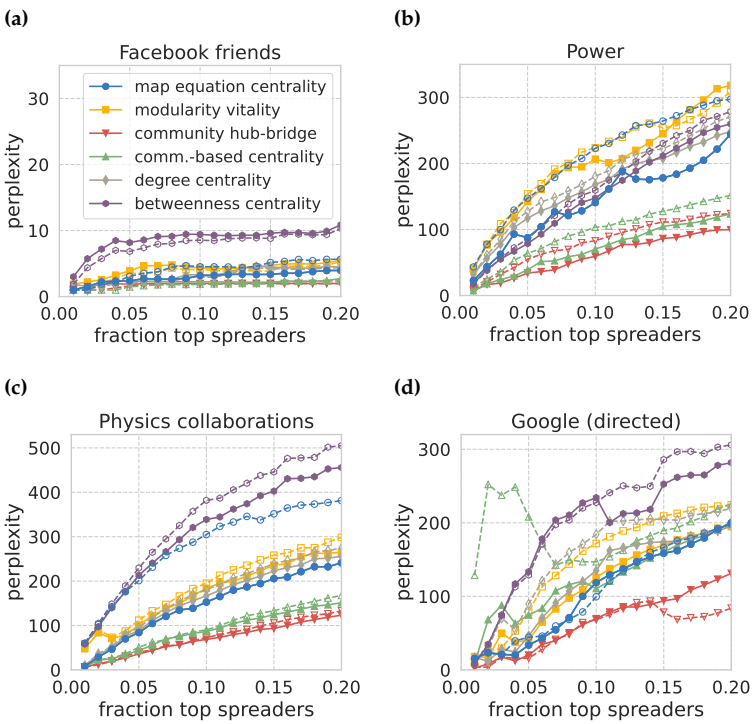


Figure 40: Node distribution perplexity for the six tested scores in four networks. A higher perplexity corresponds to a more uniform node distribution across modules.

To perform well in the SIR case, a centrality measure should select high-degree nodes because they have a higher opportunity to infect other nodes. Conversely, under the linear threshold model, it is more important to spread out the selected nodes across tightly-knit communities to reach a high activation size, or high-density communities will stop the activation of nodes [17, p. 507].

Future Directions

We could generalise map equation centrality for overlapping communities where nodes can be members in more than one

module, reflecting how people are part of multiple and overlapping social groups. Each module membership would then contribute to a node's centrality.

When selecting top-ranked nodes, we could adopt a dynamic view of node importance: instead of selecting the k top-ranked nodes all at once, we would select them one-by-one, considering how each node's importance changes once we have selected one node and silenced it. Each time we silence a node, we obtain a new coding scheme that defines the importance of the remaining nodes, potentially leading to a choice of k nodes that is different from what we would get from a static point of view.

Community-Based Link Prediction

> Mathematicians do not study objects, but the relations between objects; to them it is a matter of indifference if these objects are replaced by others, provided that the relations do not change.
>
> Henri Poincaré

Networks as models of real-world systems describe our observations, and communities summarise their structure. When networks change over time, we want to predict what links are most likely to form in the future. This would let us, for example, recommend new friends to users of online social networks, ensure that shops have the right products in stock by predicting their customers' purchase behaviour, and plan capacities in public transport networks through forecasting people's travel patterns.

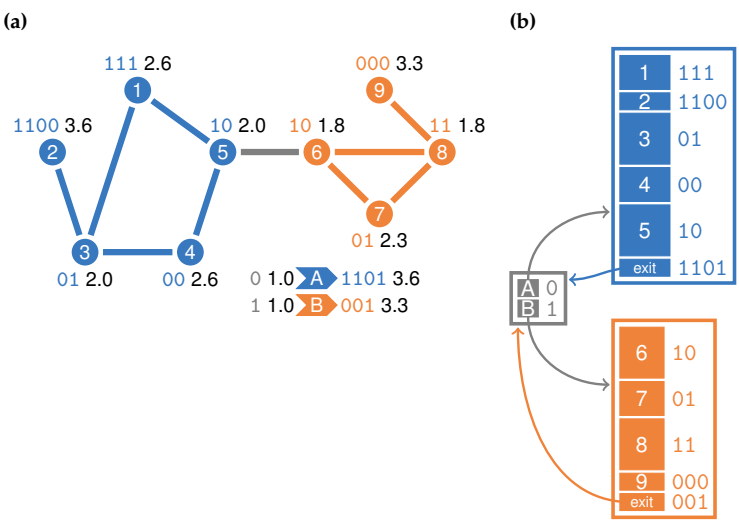


Figure 41: A network with two communities, both not fully connected, and a coding scheme. Which other, non-observed links, are most consistent with the observed links?

On a basic level, predicting links means assigning values to non-links so that we can select the best ones. Those values could be likelihoods or costs. With the map equation, a simple way to calculate the cost for a link $e = (u, v)$ is to consider how inserting e into a network $G = (V, E)$ with $u, v \in V$ and $e \notin E$ affects the codelength of G's partition M:

$$\sigma(G, \mathsf{M}, e) = \mathcal{L}(G + \{e\}, \mathsf{M}) - \mathcal{L}(G, \mathsf{M}), \tag{60}$$

where $G + \{e\}$ denotes G with link e inserted [30]. Between two links e_1 and e_2, we select the one with the lower cost, that is the one that increases M's codelength less. Inserting a link changes the degrees of its two endpoint nodes as well as all nodes' visit rates. Consider the example network in Figure 41 and partition $\mathsf{M} = \{\{1,2,3,4,5\}, \{6,7,8,9\}\}$ with codelength

$$\begin{aligned}\mathcal{L}(G, \mathsf{M}) = &\underbrace{\frac{2}{20} \cdot \mathcal{H}\left(\frac{1}{2}, \frac{1}{2}\right)}_{\text{index level}} \\ &+ \underbrace{\frac{12}{20} \cdot \mathcal{H}\left(\frac{2}{12}, \frac{1}{12}, \frac{3}{12}, \frac{2}{12}, \frac{3}{12}, \frac{1}{12}\right)}_{\text{blue module}} + \underbrace{\frac{10}{20} \cdot \mathcal{H}\left(\frac{3}{10}, \frac{2}{10}, \frac{3}{10}, \frac{1}{10}, \frac{1}{10}\right)}_{\text{orange module}} \\ &\approx 2.66 \text{ bits.}\end{aligned} \tag{61}$$

Inserting the link $e_1 = (1, 4)$ into G (Figure 42) leads to the new codelength

$$\begin{aligned}\mathcal{L}(G + \{(1,4)\}, \mathsf{M}) = &\underbrace{\frac{2}{22} \cdot \mathcal{H}\left(\frac{1}{2}, \frac{1}{2}\right)}_{\text{index level}} \\ &+ \underbrace{\frac{14}{22} \cdot \mathcal{H}\left(\frac{3}{14}, \frac{1}{14}, \frac{3}{14}, \frac{3}{14}, \frac{3}{14}, \frac{1}{14}\right)}_{\text{blue module}} + \underbrace{\frac{10}{22} \cdot \mathcal{H}\left(\frac{3}{10}, \frac{2}{10}, \frac{3}{10}, \frac{1}{10}, \frac{1}{10}\right)}_{\text{orange module}} \\ &\approx 2.64 \text{ bits,}\end{aligned} \tag{62}$$

and $\sigma(G, \mathsf{M}, e_1) = \mathcal{L}(G + \{(1,4)\}, \mathsf{M}) - \mathcal{L}(G, \mathsf{M}) \approx -0.02$ bits. e_1 improves the compression because it increases the blue module's flow persistence. While e_1 changes the blue nodes' theoretical codeword lengths, the concrete coding scheme remains optimal. Inserting $e_2 = (4, 7)$ into G (Figure 43) leads to the new codelength

$$\mathcal{L}(G + \{(4,7)\}, \mathsf{M}) = \underbrace{\frac{4}{22} \cdot \mathcal{H}\left(\frac{2}{4}, \frac{2}{4}\right)}_{\text{index level}} + \underbrace{\frac{14}{22} \cdot \mathcal{H}\left(\frac{2}{14}, \frac{1}{14}, \frac{3}{14}, \frac{3}{14}, \frac{3}{14}, \frac{2}{14}\right)}_{\text{blue module}} + \underbrace{\frac{12}{20} \cdot \mathcal{H}\left(\frac{3}{12}, \frac{3}{12}, \frac{3}{12}, \frac{1}{12}, \frac{2}{12}\right)}_{\text{orange module}}$$
$$\approx 2.99 \text{ bits}, \tag{63}$$

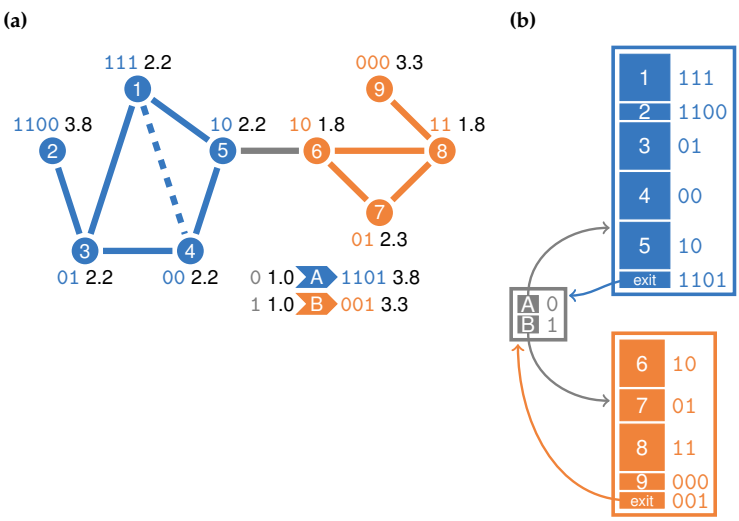


Figure 42: The network with link $(1,4)$ inserted and its new coding scheme for the same partition. The theoretical codelengths for all codewords in the blue module change while the index level and the orange module remain unaffected.

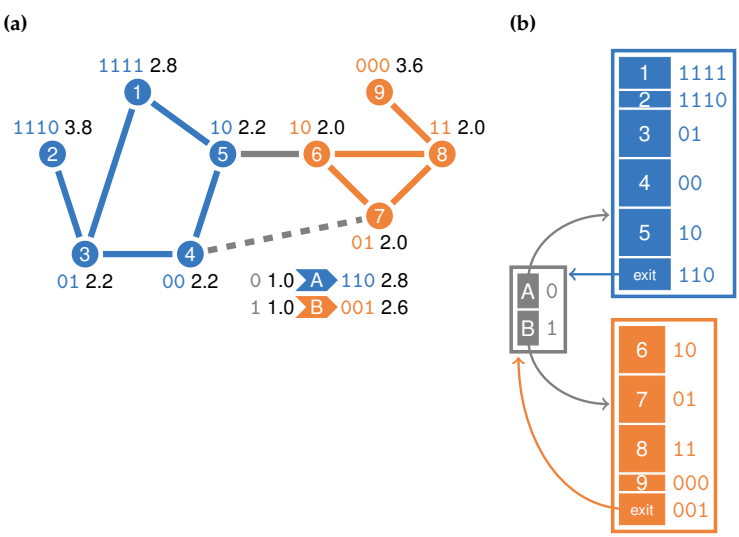


Figure 43: The network with link $(4,7)$ inserted and its new coding scheme for the same partition. The theoretical codelengths for all codewords in all modules change because the inserted link spans across both modules and also affects module entry rates.

and $\sigma(G, \mathsf{M}, e_2) = \mathcal{L}(G + \{(4,7)\}, \mathsf{M}) - \mathcal{L}(G, \mathsf{M}) \approx 0.33$ bits. e_2 has a higher cost because it decreases both modules' flow persistence and increases the index codebook's usage rate. In principle, adding a link to a network can change its community structure, which we would only notice if we search for communities again. We have avoided this by considering how an additional link changes a fixed partition's codelength, rather than asking what the new best partition and its codelength would be.

However, there is an efficiency problem with this: adding a link to a network changes all node visit and codebook usage rates, meaning that we need to evaluate the whole map equation again[58]. To find the link that, when inserted, would have the lowest cost, we need to calculate costs for $\mathcal{O}(n^2)$ many links in a network with n nodes, which quickly becomes impractical.

[58] We could remember and re-use the module-level entropies of unaffected modules.

Node Embeddings

Node-embedding methods map nodes to points in d-dimensional metric space, that is, nodes u and v would be located at $\vec{u}$ and $\vec{v}$, where typically $d \ll n$ [33]. Roughly speaking, a good embedding places more similar nodes closer together, that is when they share more common neighbours and are located in more similar network regions, meaning that we can approximate their similarity by measuring, for example, their distance in the embedding space as $\|\vec{u} - \vec{v}\|$, or the angle θ between their vectors with cosine similarity as $\cos(\theta) = \frac{\vec{u} \cdot \vec{v}}{\|\vec{u}\| \cdot \|\vec{v}\|}$, where $\|\cdot\|$ is the vector norm. A simple way to predict links is then to select those disconnected node pairs that are most similar, reflecting the idea that it is easier for nodes to connect if they are embedded closer together [90]. To avoid computing similarities for all $\mathcal{O}(n^2)$ node pairs when using their embedding vectors' distance for similarity, we could store them in a k-d tree, a data structure that enables finding a nearest neighbours in $\mathcal{O}(\log n)$ time [8]. Then, identifying the most likely link would involve finding each node's nearest neighbour, taking time $\mathcal{O}(n \log n)$, and selecting the most likely link from amongst n candidates.

Using the Codebooks

We take inspiration from node embedding-based approaches and interpret Huffman coding ensembles as implicit embeddings. But instead of mapping nodes to points in some metric space and then calculating their similarities, we calculate their similarities directly. The key insight is that, while random walks are constrained by the links in a network, a coding scheme is more flexible and we can use it to describe transitions between any pair of nodes, whether they are connected by a link or not.

Coming back to the idea that more similar objects compress more efficiently together [71], we use the coding principles behind the map equation to predict links, adopting a community-based point of view. Because we characterise modules as regions with high flow persistence, they naturally represent network neighbourhoods with high internal similarity. Consequently, we expect that compression-based link prediction favours links within modules.

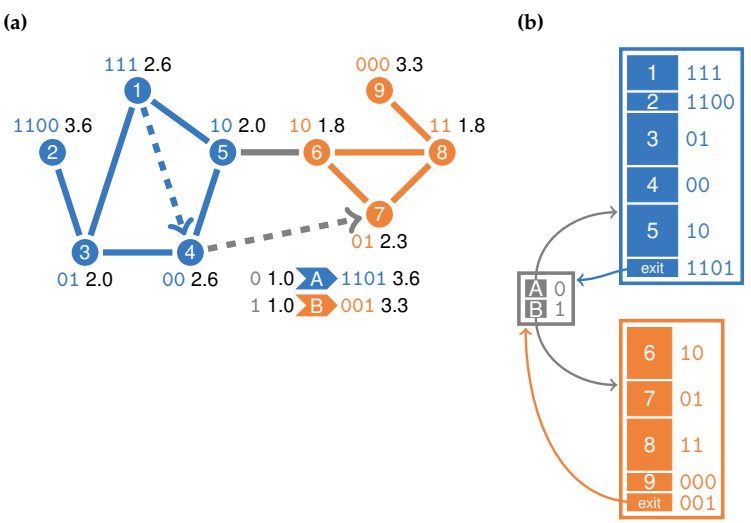


Figure 44: We fix the coding scheme and use it to describe random walker steps along the directed non-links $(1,4)$ and $(4,7)$.

Consider the example network and coding scheme from Figure 44 with the two directed non-links $(1,4)$ and $(4,7)$. How can we describe random-walker transitions along those links with the coding scheme we have? First, we recall that, when coding with the map equation, we keep track of the random walker's

current module, but not the current node, meaning that a node always has the same codeword, regardless of where the random walker comes from. To describe the directed intra-module link $(1,4)$, we use a single codeword, that is, the one assigned to the target node, `00`, using 2.6 bits in the information-theoretic limit. Describing the directed inter-module link $(4,7)$ requires three codewords, that is a module exit codeword, a module entry codeword, and a node visit codeword, `1101 1 01`, using 3.6 bits + 1.0 bit + 2.3 bits = 6.9 bits in the information-theoretic limit. Since we need fewer bits to describe $(1,4)$ than $(4,7)$, we consider the former more likely. In general, to describe a link, we begin in its source node's module and search the coding scheme for the shortest path to the link's target node, and use the codewords along this path.

Our approach has a few implications we should keep in mind. First, describing links from different source nodes in the same module to the same target node requires the same number of bits because both correspond to the same coding path. Turning link descriptions dependent on their source node requires equipping the random walker with memory, contradicting the map equation's coding principles. Second, we can only directly assign costs to directed links because describing a link corresponds to a random-walker transitions, which itself is directed. To assign a cost to an undirected link (u,v), we could consider the two directed links (u,v) and (v,u), calculate costs for both, and take the average. Third, we relate a link's likelihood to how expensive, in bits, it would be to describe, given a fixed coding scheme, but we do not say anything about its weight. In fact, regardless of a link's weight, with a fixed coding scheme, it would always have the same description.

Map Equation Similarity

Map equation similarity derives node similarity from modular coding schemes. Consider the coding scheme in Figure 44(b) again and recall that block heights show the codewords' usage rates. Random walkers at the index level use the entry codewords for the blue and orange modules at rate $\frac{1}{2}$ each. A ran-

dom walker in the blue module uses node 4's codeword at rate $\frac{2}{12}$, and the exit codeword at rate $\frac{1}{12}$; a random walker in the orange module uses node 7's codeword at rate $\frac{2}{10}$. By multiplying the usage rates along a path in the coding scheme, we get the rate r at which the random walker uses that path, and $-\log_2(r)$ tells us how many bits we need to describe that transition. For example, describing the transition from any node in the blue module to node 4 requires $-\log_2\left(\frac{2}{12}\right) \approx 2.6$ bits, and describing the transition from any node in the blue module to node 7 requires $-\log_2\left(\frac{1}{12} \cdot \frac{1}{2} \cdot \frac{2}{10}\right) \approx 6.9$ bits.

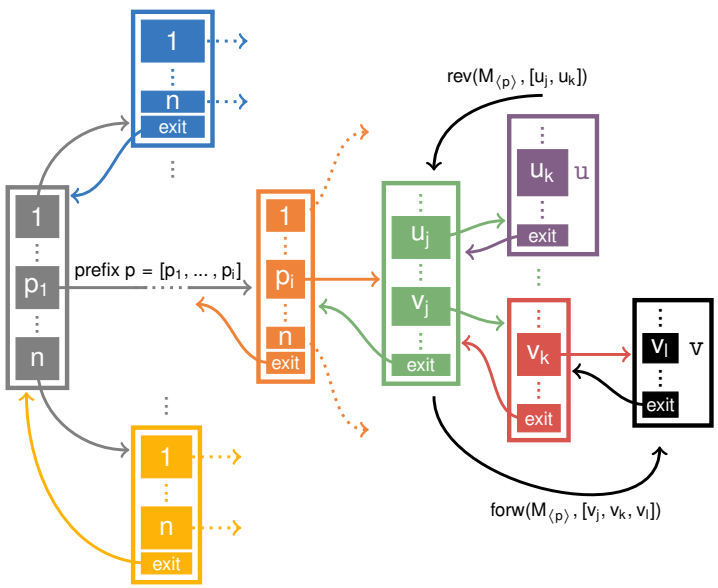


Figure 45: Illustration of map equation similarity between nodes u and v. We number each codebook's entries from 1 to n and use the numbers as addresses. Node u has address $[p_1, \ldots, p_i, u_j, u_k]$, and v has $[p_1, \ldots, p_i, v_j, v_k, v_l]$. Their longest common prefix is $p = [p_1, \ldots, p_i]$, and $\mathsf{M}_{\langle p \rangle}$ is the submodule at that address, that is the smallest module that contains u and v. We get the similarity of u to v by traversing backwards from u'v module to $\mathsf{M}_{\langle p \rangle}$, and then forwards to v's module, including visiting v, and multiplying the corresponding codeword usage rates along the way.

To derive map equation similarity for the general case, where M can be a hierarchical network partition, we number the submodules within each module m from 1 to n_{m}, and refer to these numbers as addresses, such that an ordered sequence of addresses uniquely identifies a path starting at the coding tree's root. Let $\text{addr}\colon \mathsf{M} \times V \to \mathit{List}(\mathbb{N})$ be a function that takes a network partition and a node as input, and returns the node's address in the partition. To calculate the similarity of node v to u, we identify the longest common prefix p of the nodes' addresses, $\text{addr}(\mathsf{M}, u)$ and $\text{addr}(\mathsf{M}, v)$, and select the coding tree's sub-tree $\mathsf{M}_{\langle p \rangle}$ that corresponds to the prefix p; $\mathsf{M}_{\langle p \rangle}$ is the small-

est sub-tree that contains u and v. We obtain the addresses for u and v within sub-tree $\mathsf{M}_{\langle p \rangle}$ by removing the prefix p from their addresses. That is, $\text{addr}\,(\mathsf{M}, u) = p + \!\!+ \,\text{addr}(\mathsf{M}_{\langle p \rangle}, u)$ and $\text{addr}\,(\mathsf{M}, v) = p + \!\!+ \,\text{addr}(\mathsf{M}_{\langle p \rangle}, v)$, where $+ \!\!+$ is list concatenation. With this notation, the rate at which a random walker transitions from u to v is the product of (i) the rate at which the random walker moves along the path $\text{addr}(\mathsf{M}_{\langle p \rangle}, u)$ in reverse direction, $\text{rev}(\mathsf{M}_{\langle p \rangle}, \text{addr}(\mathsf{M}_{\langle p \rangle}, u))$, that is from u to the root of $M_{\langle p \rangle}$, and (ii) the rate at which the random walker moves along the path $\text{addr}(\mathsf{M}_{\langle p \rangle}, v)$ in forward direction, $\text{forw}(\mathsf{M}_{\langle p \rangle}, \text{addr}(\mathsf{M}_{\langle p \rangle}, v))$, that is from the root of $\mathsf{M}_{\langle p \rangle}$ to v, where

$$\text{rev}\,(\mathsf{M}, a) = \begin{cases} 1 & \text{if } a = [x] \\ \mathsf{M}_{\langle [x] \rangle, \text{exit}} \cdot \text{rev}(\mathsf{M}_{\langle [x] \rangle}, a') & \text{if } a = [x] + \!\!+ \, a' \end{cases} \tag{64}$$

$$\text{forw}\,(\mathsf{M}, a) = \begin{cases} \frac{p_{\langle [x] \rangle}}{p_{\mathsf{M}}} & \text{if } a = [x] \\ \mathsf{M}_{\langle [x] \rangle, \text{enter}} \cdot \text{forw}(\mathsf{M}_{\langle [x] \rangle}, a') & \text{if } a = [x] + \!\!+ \, a' \end{cases} \tag{65}$$

and a' denotes non-empty sequences. Here p_{M} is the codebook usage rate for module M and $p_{\langle [x] \rangle}$ is the visit rate for the node identified by address x within the given module. The final addresses in Equations (64) and (65) are treated differently, reflecting that the random walker does not remember the source node but visits the target node. We define map equation similarity as

$$\text{MapSim}\,(M, u, v) = \text{rev}(\mathsf{M}_{\langle p \rangle}, \text{addr}(\mathsf{M}_{\langle p \rangle}, u)) \cdot \text{forw}(\mathsf{M}_{\langle p \rangle}, \text{addr}(\mathsf{M}_{\langle p \rangle}, v)), \tag{66}$$

where p is the longest common prefix shared by the addresses of u and v in the coding tree defined by M. To express map equation similarity in terms of description length, we take the $-\log_2$ of MapSim and consider pairs of nodes that yield a shorter description length more similar. Figure 45 illustrates these ideas with a generic coding scheme.

Efficient Predictions

With map equation similarity, we can predict links efficiently by traversing the coding tree, starting simultaneously from each node, enumerating links to other nodes in ascending cost order,

and merging the results into one list. In partitions that capture a network's structure well, random walkers change modules only rarely, and module exits are expensive because they involve long codewords. Therefore, traversing the coding scheme from each node yields intra-module links first before using the exit codeword and considering links to nodes in other modules.

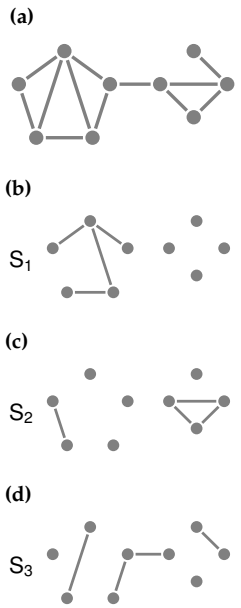


Figure 46: A network with 12 observed links (a) and a split of its links into three sets S_1 (b), S_2 (c), and S_3 (d).

Evaluating Link Predictions

There are many ways to set up the exact details for evaluating a method's link prediction performance, but often, cross-validation and negative sampling are used.

In so-called k-fold cross-validation, we randomly split the network's links into k equally-sized sub-sets, use k-1 of those subsets for training, and the remaining sub-set for validation. For example, with 3-fold cross validation, we split the links E into three sets, S_1, S_2, and S_3, where $E = S_1 \cup S_2 \cup S_3$ (Figure 46). Then, first, we use $S_1 \cup S_2$ to train a model, and S_3 to measure its prediction performance[59]. Second, we use $S_1 \cup S_3$ to train another model, and S_2 to measure its performance. Third, we use $S_2 \cup S_3$ to train yet another model, and S_1 to measure its performance. Finally, we take the average of the models' performances.

To evaluate a model's performance during cross-validation, we calculate scores for the non-links. But since networks tend to be sparse and there are of the order of $\mathcal{O}\left(n^2\right)$ many non-links, we only calculate scores for the links in the validation set S^+_{val} as well as for an equally large set of negative links S^-_{val}. A negative link is a randomly chosen non-link that is neither part of the train nor the validation set. We refer to the links in the original validation set S^+_{val} as positive links, those are links that do not exist in the network because we have removed them, and we hope to identify them through link prediction. Once we have calculated scores for all links $e \in S_{\text{val}}$, where $S_{\text{val}} = S^+_{\text{val}} \cup S^-_{\text{val}}$, we rank them.

To measure the performance, given the validation links' ranking, there are many options. A common way is to pick a threshold t that determines how many of the top-ranked non-links we

[59] In this case, we call $S_1 \cup S_2$ the training set, and S_3 the validation set.

select for evaluation. With the set T of t top-ranked non-links, we can use measures such as accuracy, recall, or precision to judge a method's performance, where accuracy is the fraction of correctly classified non-links, recall is the fraction of positive links we have selected, $\frac{T \cap S^+_{\text{val}}}{S^+_{\text{val}}}$, and precision is the fraction of positive links in our selection, $\frac{T \cap S^+_{\text{val}}}{T}$. However, choosing a different value for t affects a model's performance outcome by changing T. An alternative is to consider the performance across all possible thresholds, for example using the precision-recall (PR) or receiver-operating-characteristic (ROC) curve. The PR curve summarises a model's precision and recall performance for all thresholds while the ROC curve summarises the trade-off between true positives and false positives amongst the predicted links for each t. A model is considered better than another one if its area under the PR curve, AUPR, as well as its area under the ROC curve, AUC, is higher [90][60].

[60] While AUC is increasingly criticised as a measure of model performance, AUPR is becoming more popular. Nevertheless, a recommendation is to report both [90].

We evaluate map equation similarity on 47 real-world networks, 35 directed and 12 undirected. We use 5-fold cross-validation and negative sampling, and report the AUPR and AUC performance across the 47 networks (Figure 47)[61]. As baselines, we use the four random walk-based embedding methods DeepWalk [63], node2vec [34], LINE with first, second, and combined first-and-second order neighbourhoods [77], and NERD [38], using the implementations provided by the respective authors. Each of the baseline methods have a set of hyperparameters, such as the number of embedding dimensions or the number of random walks to be sampled, that we could optimise to tune each method for each specific network. However, we refrain from doing so because using different sets of hyperparameters essentially changes the method, but we want to compare how the same method performs across a large set of networks [38]. Moreover, in practical settings, constraints regarding time or other resources may make it infeasible to tune the parameters for each data set. Instead, we use defaults for all parameters as suggested by the respective authors [4]. Map equation similarity, on the other hand, is parameter-free, and Infomap determines the complexity of the embedding purely from the data using the map equation.

[61] Paper IV contains the full results on a per-network basis [4].

We find that, on average, map equation similarity performs best, both in terms of AUPR and AUC. Typically, one or more of the baseline methods outperform map equation similarity, however, across the set of 47 networks, they often perform worse than map equation similarity [4].

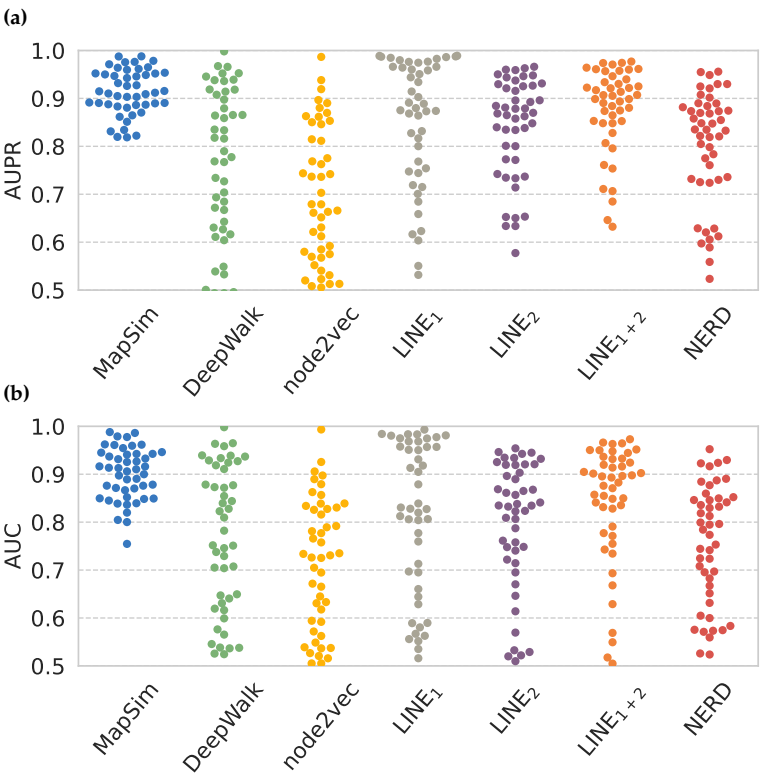


Figure 47: The area under (a) the precision-recall curve, AUPR, and under (b) the receiver-operating-characteristic curve, AUC, for map equation similarity, DeepWalk, node2Vec, LINE, and NERD on 47 real-world networks. Each dot represents the performance on one network.

Future Directions

We interpreted the coding structure for a network partition as an implicit network embedding that allows us to calculate similarities between nodes. A next step would be to make the embedding explicit, potentially in terms of a mapping to points in hyperbolic space. This could also help to address the drawback of map equation similarity that it must assign the same similarity to different node pairs (u, w) and (v, w) if u and v are in the same module because, when coding with the map equation, we

only remember the current module, but not the most recently visited node.

Map equation similarity could also be used to generate recommendations, which is similar to link prediction, but there is a difference. In link prediction, the goal is to predict what links are most likely to form across the whole network, whereas in the recommendation setting, the task is to predict the most likely links on a per-node basis.

Part III

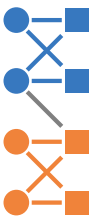
Conclusion

Summary

> Science is knowledge which we understand so well that we can teach it to a computer; and if we don't fully understand something, it is an art to deal with it.
>
> Donald Knuth

The world is full of networks, many of which are too large and complex for us to grasp as a whole. To understand those networks, we need to simplify them so that we can focus on analysing their components, one at a time, and how those components interact. The map equation framework provides a tool that does exactly this: it finds a network's modules, allowing us to represent it as a system of systems. Designed for community detection in unipartite networks with complete data, the map equation can in principle be applied when its assumptions are not satisfied, but then we must be prepared for a trade-off. For example, the map equation neither (i) takes the different node types in bipartite networks nor (ii) potentially missing data in real-world networks into account. Therefore, when analysing such data, we can miss important patterns or end up with spurious communities. Moreover, in practical applications, we often wish to do more than find communities, for example, (iii) decide how influential nodes are, or (iv) predict what links are likely to form in the future. However, both these tasks are outside the map equation's original scope of community detection.

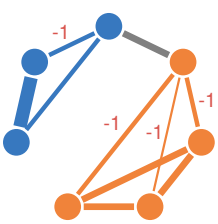
Looking through the coding-lens, we developed and explained solutions for all four problems, using different approaches for each of them.



Mapping Flows on Bipartite Networks

We took advantage of the link patterns in bipartite networks and equipped the map equation with an adapted coding scheme to reflect the existence of two node types. Our key insight was that random walks must alternate between left and right nodes, allowing us to narrow down which nodes the random walker can visit in the next step by remembering its current node's type. We used coding schemes with two separate codebooks in each module: one for left-to-right, and one for right-to-left transitions. By controlling the rate at which we use node-type information, we found communities at different scales, scanning the community landscape in bipartite networks from coarse to fine.

Many real-world networks are bipartite, for example user-song networks where users are connected to songs they like, user-event attendance networks, or director-company networks where persons are connected to companies if they are a member of its board of directors. The bipartite map equation helps us understand those networks better by paying more or less attention to node types and revealing the networks' structure at different scales.



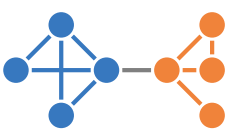
Mapping Flows on Networks with Incomplete Data

To avoid detecting spurious communities in networks where data is missing, we used an empirical Bayes estimate of the random walker's transition rate between nodes, assuming that a node's strength tells us to what extent we can trust its incident links' weights. Our transition rate estimate corresponds to node-dependent teleportation where both the random walker's teleportation rate and target depend on the current node. Effectively, we introduced a pre-processing step to address the missing data problem without changing the map equation itself, and were able to detect robust communities even when moderate to large amounts of data were missing.

In reality, when we collect data about a networked system by observing how its parts connect to each other, it can be difficult to obtain a complete picture of the network: some connections might be difficult to observe, perhaps because they happen so

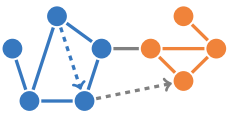
rarely and we only have a limited amount of resources available to spend on collecting data. This could be the case in social networks where some friends only interact every few months and we only have a few weeks of time to collect the data. However, we still want to analyse such a network's structure, and modelling missing data helps us do this with the map equation.

Map Equation Centrality



We derived a community-aware centrality score from the map equation by invoking the VGC principle. Map equation centrality relates a node's importance to the combined marginal harm that its existence causes to the remaining nodes, that is, by how many bits the codelength could be reduced if that node did not exist. Since a node's codeword only exists in the context of its own module, nodes outside of that module are unaffected by the node's existence. Therefore, using the map equation framework, we can determine node importance from a node's modular context alone, using only the node's visit rate and its module's codebook usage rate. Map equation centrality is flow-model agnostic, allowing us to choose the the model that best characterises the process at hand for computing node centrality scores.

Typically, we find community structure in social networks, that is, we can identify social groups and say something about who belongs to which of those groups. The influence that each person has depends on how well they are connected to others, but also on how influential their group is. As an example, consider membership in political parties: politicians have more influence if they have more connections, but this is not the whole story. As a member of a more influential party, they have more influence than a politician who is a member in a less influential party, even if they otherwise have the same connections. Map equation centrality takes such group memberships into account and gives us a better idea about how influential each node is in a network with communities.



Link Prediction with Map Equation Similarity

To predict links with the map equation framework, we interpreted a network's community structure as an implicit node embedding and used its corresponding coding scheme to calculate similarities between pairs of nodes. We used the fact that, while a random walker is constrained by the link patterns in a network, coding schemes are more flexible and can describe transitions between any two nodes. We defined map equation similarity as the rate at which the random walker transitions from one node to another, and regard a node as more similar to another the higher that rate is. Map equation similarity is an asymmetric measure because random walker transitions rates between two nodes are, in general, asymmetric. Equivalently, we can describe the similarity between nodes as the number of bits required to encode a random-walker step between them; then, the more similar two nodes are, the fewer bits are needed to describe the random walker's transition, and the more likely we consider the corresponding non-link to form.

We often only get a snapshot of a network when we observe them in reality, even if the data is complete. For example, the data we can extract from online social networks is complete because it accurately captures the network's current state. But it may change over time: persons who are not yet connected may befriend each other. With map equation similarity, we can use a network's current community structure to predict who will become connected to whom, and make friend suggestions accordingly. Similarly, for customer-product networks, we can predict who will buy what product in the future, and recommend those products.

Conclusion

We looked at the map equation through the lens of coding and applied it to questions that were outside its original scope. We addressed each challenge with a different strategy, that is (i) we adapted the map equation's coding scheme to the specific link patterns in bipartite networks, (ii) we computed regularised estimates for the random walker's transition rates between nodes to

compensate for incomplete data, (iii) we derived a community-aware centrality score by considering how a node's existence affects other nodes' codewords, and (iv) we interpreted coding schemes as implicit node embeddings and used them to calculate similarities between pairs of nodes to make link predictions. In all cases, we demonstrated the map equation's flexibility and how it can be adapted, extended, or used in a novel way, and hope to provide inspiration to others who may wish to follow similar approaches to answer their own research questions.

Author Contributions

A short description of the authors' contributions to each paper

I *Mapping Flows on Bipartite Networks*
Christopher Blöcker and Martin Rosvall
Martin an I came up with the study idea. I was the main responsible for the project, implemented the bipartite map equation in Infomap, and ran the experiments. Martin and I interpreted the results together. I was the main responsible for writing the paper and Martin supported me.

II *Mapping Flows on Weighted and Directed Networks with Incomplete Observations*
Jelena Smiljanić, **Christopher Blöcker**, Daniel Edler, Martin Rosvall
Jelena and Martin came up with the study idea, and Jelena was the main responsible for the project and developed most of the theory. All four of us actively participated in discussions to bring the project forward. Jelena, Daniel, and I implemented different parts of the solution, and Jelena ran the experiments. Jelena and I mainly interpreted the results and wrote the paper with support from Daniel and Martin.

III *Map Equation Centrality: Community-aware Centrality Based on the Map Equation*
Christopher Blöcker, Juan Carlos Nieves, Martin Rosvall
Juan Carlos, Martin, and I came up with the study idea. I was the main responsible for the project, implemented map equation centrality, and ran the experiments. Juan Carlos, Martin, and I interpreted the results together. I was the main responsible for writing the paper and Juan Carlos and Martin supported me.

IV *Similarity-based Link Prediction from Modular Compression of Network Flows*
Christopher Blöcker, Jelena Smiljanić, Ingo Scholtes, Martin Rosvall
I came up with the initial study idea and developed it first with Martin, and later also with Jelena and Ingo. I was the main responsible for the project, implemented map equation similarity, and ran the experiments. Ingo and I mainly interpreted the results and wrote the paper and were supported by Jelena and Martin.

Acknowledgements

Many persons have helped me during my PhD studies, directly and indirectly, and I am grateful for their support. I am almost certain that I will forget to mention someone who deserves mentioning, and I am sorry about that, but I will do my best.

First and foremost, Nina, my love, you put up with me around the clock, when science worked as it should and I was happy, but also when it did not and I was upset. Sometimes I knew better than to follow your advice, but in the end it always turned out that I should have listened.

Martin, you introduced me to the world of network science, pushed me to do great research, showed me how to tell exciting stories in my papers, and helped me develop as a researcher and teacher — thank you!

Juan Carlos and Ingo, thank you for interesting discussions and the fun projects we did together, I hope there will be more.

Thomas, thank you for all your support, helping me get more teaching experience, and the papers we wrote together.

IceLab, old and new: small Ludvig, Markus, Peter, Jonas, Daniel, Alexis, Magnus, Joaquín, Lucas, Dolores, big Ludvig, Hugo, Moa, and all the rest. Special thanks to Jelena for excellent work together and feedback on an earlier version of this thesis. And Anton, thank you for great company throughout the years and especially on our trips to Salina and Tokyo.

My WASP buddies in Umeå: Tim and Tobias; and my batch, WASP AS2: Christian, Matthias, Martin L., Caroline, Hector, Georgia, Veronika, Martin I., Joakim, thank you for fun times together, in courses and on study trips. The WASP PhD student council: Matthias, Tim, Anoud, Alexandre, Georgia, Veronika, Joakim, Kristin, Shuangshuang, Anton, Lena, and Amandine,

thank you for great work together.

My dance family in Umeå, which provides a sometimes much needed counterbalance to life as a scientist: Elin, Sara, Anton, Jennie, Daniel, Frida, Sofia, Johanna J., Sandra, Simon, Philip, Alice, Ida, Kristoffer, Johannes, Magnus, Madeleine, Jens, Johanna N., Thomas, Fredrik, Lina, thank you for lots of fun on and off the dance floor. And beyond Umeå: Outi, Tero, Lauri, Therése, Joachim, Katja, Viktoria, Robin, Ibi, Chuck, Hanna O., Hanna T., Mari, Heli, Susanne, Leif, and countless others.

My friends back home whom I've known since school and university days: Julius, Gerrit, Jan-Philip, and Corvin, thank you for still catching up and being interested in what's going on.

Den Norgrenska klanen: Nina, Lena, Sven, Ewa, Thomas, Sofia, Ida, Stefan, Örjan och Carina, tack för att ni får mig att känna mig hemma i Sverige.

Und zu guter Letzt meine Familie: Mama, Julia, und Sabrina, es ist nicht immer einfach, in der Welt verstreut zu sein. Aber Mama, du bleibst trotzdem cool. Es ist schön, dass wir in einer Umgebung aufwachsen durften, in der Bildung wichtig war, ansonsten hätte ich (oder hätten wir?) womöglich nie studiert, geschweige denn, dass ich an eine Promotion gedacht hätte. Und Sabrina, danke für dein Interesse an meinen Papern, dein Feedback und dass du ein Pol der Logik bist.

Bibliography

[1] Christopher Blöcker and Martin Rosvall. "Mapping flows on bipartite networks". In: *Physical Review E* 102 (5 Nov. 2020), p. 052305. DOI: 10.1103/PhysRevE.102.052305.

[2] Jelena Smiljanić et al. "Mapping flows on weighted and directed networks with incomplete observations". In: *Journal of Complex Networks* 9.6 (Dec. 2021). DOI: 10.1093/comnet/cnab044.

[3] Christopher Blöcker, Juan Carlos Nieves, and Martin Rosvall. "Map equation centrality: community-aware centrality based on the map equation". In: *Applied Network Science* 7.1 (Aug. 2022), p. 56. DOI: 10.1007/s41109-022-00477-9.

[4] Christopher Blöcker et al. *Similarity-based Link Prediction from Modular Compression of Network Flows*. 2022. DOI: 10.48550/ARXIV.2208.14220.

[5] Eric Allender. "A status report on the P versus NP question". In: *Advances in Computers* 77 (2009), pp. 117–147. DOI: https://doi.org/10.1016/S0065-2458(09)01204-2.

[6] Rounak Banik. *The Complete Pokemon Dataset*. Accessed 2022-03-09. 2018. URL: https://www.kaggle.com/datasets/rounakbanik/pokemon.

[7] Aleix Bassolas et al. *Metadata-informed community detection with lazy encoding using absorbing random walks*. 2021. DOI: 10.48550/ARXIV.2111.05158.

[8] Jon Louis Bentley. "Multidimensional Binary Search Trees Used for Associative Searching". In: *Communications of the ACM* 18.9 (Sept. 1975), pp. 509–517. DOI: 10.1145/361002.361007.

[9] Vincent D Blondel et al. "Fast unfolding of communities in large networks". In: *Journal of Statistical Mechanics: Theory and Experiment* 2008.10 (Oct. 2008), P10008. DOI: 10.1088/1742-5468/2008/10/p10008.

[10] Ludvig Bohlin. "Toward higher-order network models". PhD thesis. Umeå University, 2018. ISBN: 978-91-7601-892-7.

[11] Alexandra Brintrup and Anna Ledwoch. "Supply network science: Emergence of a new perspective on a classical field". In: *Chaos: An Interdisciplinary Journal of Nonlinear Science* 28.3 (2018), p. 033120. DOI: 10.1063/1.5010766.

[12] Fabio Caccioli, Paolo Barucca, and Teruyoshi Kobayashi. "Network models of financial systemic risk: a review". In: *Journal of Computational Social Science* 1.1 (2018), pp. 81–114. DOI: 10.1007/s42001-017-0008-3.

[13] Kousik Das, Sovan Samanta, and Madhumangal Pal. "Study on centrality measures in social networks: a survey". In: *Social network analysis and mining* 8.1 (2018), pp. 1–11. DOI: 10.1007/s13278-018-0493-2.

[14] Manlio De Domenico et al. "Identifying Modular Flows on Multilayer Networks Reveals Highly Overlapping Organization in Interconnected Systems". In: *Physical Review X* 5 (1 Mar. 2015), p. 011027. DOI: 10.1103/PhysRevX.5.011027.

[15] Sybil Derrible and Christopher Kennedy. "Applications of Graph Theory and Network Science to Transit Network Design". In: *Transport Reviews* 31.4 (2011), pp. 495–519. DOI: 10.1080/01441647.2010.543709.

[16] Salva Duran-Nebreda and George W. Bassel. "Bridging Scales in Plant Biology Using Network Science". In: *Trends in Plant Science* 22.12 (2017), pp. 1001–1003. DOI: 10.1016/j.tplants.2017.09.017.

[17] David Easley and Jon Kleinberg. *Networks, crowds, and markets: Reasoning about a highly connected world*. Cambridge university press, 2010. ISBN: 978-0-521-19533-1.

[18] Achim Edelmann et al. "Computational social science and sociology". In: *Annual Review of Sociology* 46 (2020), pp. 61–81. DOI: 10.1146/annurev-soc-121919-054621.

[19] D. Edler, A. Eriksson, and M. Rosvall. *The Infomap Software Package*. Accessed 2022-04-01. 2020. URL: https://www.mapequation.org.

[20] Daniel Edler, Ludvig Bohlin, and Martin Rosvall. "Mapping Higher-Order Network Flows in Memory and Multilayer Networks with Infomap". In: *Algorithms* 10.4 (2017). DOI: 10.3390/a10040112.

[21] Jack Edmonds and Ellis L. Johnson. "Matching, Euler tours and the Chinese postman". In: *Mathematical Programming* 5.1 (Dec. 1973), pp. 88–124. DOI: 10.1007/BF01580113.

[22] Paul Erdős and Alfréd Rényi. "On the evolution of random graphs". In: *Publications of the Mathematical Institute of the Hungarian Academy of Sciences* 5.1 (1960), pp. 17–60.

[23] Carlos Roberto Fonseca and Gislene Ganade. "Asymmetries, Compartments and Null Interactions in an Amazonian Ant-Plant Community". In: *Journal of Animal Ecology* 65.3 (May 1996), pp. 339–347. DOI: 10.2307/5880.

[24] Marie-Josée Fortin, Mark RT Dale, and Chris Brimacombe. "Network ecology in dynamic landscapes". In: *Proceedings of the Royal Society B* 288.1949 (2021), p. 20201889. DOI: 10.1098/rspb.2020.1889.

[25] Santo Fortunato. "Community detection in graphs". In: *Physics Reports* 486.3 (2010), pp. 75–174. DOI: 10.1016/j.physrep.2009.11.002.

[26] Santo Fortunato et al. "Science of science". In: *Science* 359.6379 (2018), eaao0185. DOI: 10.1126/science.aao0185.

[27] M.L. Fredman and R.E. Tarjan. "Fibonacci Heaps And Their Uses In Improved Network Optimization Algorithms". In: *25th Annual Symposium on Foundations of Computer Science*. 1984, pp. 338–346. DOI: 10.1109/SFCS.1984.715934.

[28] Michael R Garey and David S Johnson. *Computers and Intractability*. Vol. 174. W.H. Freeman and Company, 1979. ISBN: 978-0-7167-1045-5.

[29] Zakariya Ghalmane, Mohammed El Hassouni, and Hocine Cherifi. "Immunization of networks with non-overlapping community structure". In: *Social Network Analysis and Mining* 9.1 (2019), pp. 1–22. DOI: 10.1007/s13278-019-0591-9.

[30] Amir Ghasemian, Homa Hosseinmardi, and Aaron Clauset. "Evaluating Overfit and Underfit in Models of Network Community Structure". In: *IEEE Transactions on Knowledge and Data Engineering* 32.9 (2020), pp. 1722–1735. DOI: 10.1109/TKDE.2019.2911585.

[31] David F. Gleich. "PageRank Beyond the Web". In: *SIAM Review* 57.3 (2015), pp. 321–363. DOI: 10.1137/140976649.

[32] Marko Gosak et al. "Network science of biological systems at different scales: A review". In: *Physics of Life Reviews* 24 (2018), pp. 118–135. DOI: 10.1016/j.plrev.2017.11.003.

[33] Palash Goyal and Emilio Ferrara. "Graph embedding techniques, applications, and performance: A survey". In: *Knowledge-Based Systems* 151 (2018), pp. 78–94. DOI: 10.1016/j.knosys.2018.03.022.

[34] 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*. ACM. 2016, pp. 855–864. DOI: 10.1145/2939672.2939754.

[35] David A Huffman. "A method for the construction of minimum-redundancy codes". In: *Proceedings of the IRE* 40.9 (1952), pp. 1098–1101. DOI: 10.1109/JRPROC.1952.273898.

[36] David M.P. Jacoby and Robin Freeman. "Emerging Network-Based Tools in Movement Ecology". In: *Trends in Ecology & Evolution* 31.4 (2016), pp. 301–314. DOI: 10.1016/j.tree.2016.01.011.

[37] Brian Karrer and M. E. J. Newman. "Stochastic blockmodels and community structure in networks". In: *Physical Review E* 83 (1 Jan. 2011), p. 016107. DOI: 10.1103/PhysRevE.83.016107.

[38] Megha Khosla et al. "Node Representation Learning for Directed Graphs". In: *Machine Learning and Knowledge Discovery in Databases*. Cham: Springer, 2020, pp. 395–411. DOI: 10.1007/978-3-030-46150-8_24.

[39] Maksim Kitsak et al. "Identification of influential spreaders in complex networks". In: *Nature physics* 6.11 (2010), pp. 888–893. DOI: 10.1038/nphys1746.

[40] Dirk Koschützki et al. "Centrality Indices". In: *Network Analysis: Methodological Foundations*. Ed. by Ulrik Brandes and Thomas Erlebach. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 16–61. DOI: 10.1007/978-3-540-31955-9_3.

[41] Jérôme Kunegis. "KONECT: The Koblenz Network Collection". In: *Proceedings of the 22nd International Conference on World Wide Web*. WWW '13 Companion. Rio de Janeiro, Brazil: Association for Computing Machinery, 2013, pp. 1343–1350. DOI: 10.1145/2487788.2488173.

[42] R. Lambiotte and M. Rosvall. "Ranking and clustering of nodes in networks with smart teleportation". In: *Physical Review E* 85 (5 May 2012), p. 056107. DOI: 10.1103/PhysRevE.85.056107.

[43] Herman B Leonard. "Elicitation of honest preferences for the assignment of individuals to positions". In: *Journal of Political Economy* 91.3 (1983), pp. 461–479. DOI: 10.1086/261158.

[44] Jingyi Lin and Yifang Ban. "Complex Network Topology of Transportation Systems". In: *Transport Reviews* 33.6 (2013), pp. 658–685. DOI: 10.1080/01441647.2013.848955.

[45] Chuang Liu et al. "Computational network biology: Data, models, and applications". In: *Physics Reports* 846 (2020). Computational network biology: Data, models, and applications, pp. 1–66. DOI: 10.1016/j.physrep.2019.12.004.

[46] David JC MacKay, David JC Mac Kay, et al. *Information theory, inference and learning algorithms*. Cambridge university press, 2003. ISBN: 978-0-521-64298-9.

[47] Sofus A Macskassy and Foster Provost. "Classification in networked data: A toolkit and a univariate case study." In: *Journal of Machine Learning Research* 8.5 (2007). URL: https://jmlr.csail.mit.edu/papers/v8/macskassy07a.html.

[48] Thomas Magelinski, Mihovil Bartulovic, and Kathleen M. Carley. "Measuring Node Contribution to Community Structure With Modularity Vitality". In: *IEEE Transactions on Network Science and Engineering* 8.1 (2021), pp. 707–723. DOI: 10.1109/TNSE.2020.3049068.

[49] Benjamin F. Maier and Dirk Brockmann. "Cover time for random walks on arbitrary complex networks". In: *Physical Review E* 96 (4 Oct. 2017), p. 042307. DOI: 10.1103/PhysRevE.96.042307.

[50] Naoki Masuda, Mason A. Porter, and Renaud Lambiotte. "Random walks and diffusion on networks". In: *Physics Reports* 716-717 (2017). Random walks and diffusion on networks, pp. 1–58. DOI: 10.1016/j.physrep.2017.07.007.

[51] Miller McPherson, Lynn Smith-Lovin, and James M Cook. "Birds of a feather: Homophily in social networks". In: *Annual Review of Sociology* 27.1 (2001), pp. 415–444. DOI: 10.1146/annurev.soc.27.1.415.

[52] Merian-Erben, Public domain, via Wikimedia Commons. *Königsberg 1651*. Accessed 2022-03-09. 2016. URL: https://commons.wikimedia.org/wiki/File:Image-Koenigsberg,_Map_by_Merian-Erben_1652.jpg.

[53] M. E. J. Newman. "The Structure and Function of Complex Networks". In: *SIAM Review* 45.2 (2003), pp. 167–256. DOI: 10.1137/S003614450342480.

[54] M. E. J. Newman and M. Girvan. "Finding and evaluating community structure in networks". In: *Physical Review E* 69 (2 Feb. 2004), p. 026113. DOI: 10.1103/PhysRevE.69.026113.

[55] M.E.J. Newman and M. Girvan. "Mixing Patterns and Community Structure in Networks". In: *Statistical Mechanics of Complex Networks*. Ed. by Romualdo Pastor-Satorras, Miguel Rubi, and Albert Diaz-Guilera. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 66–87. DOI: 10.1007/978-3-540-44943-0_5.

[56] Mark Newman. *Networks*. Oxford University Press, 2018. ISBN: 978-0-19-883997-2.

[57] Luis E. Olmos et al. "A data science framework for planning the growth of bicycle infrastructures". In: *Transportation Research Part C: Emerging Technologies* 115 (2020), p. 102640. DOI: 10.1016/j.trc.2020.102640.

[58] Tore Opsahl. *Why anchorage is not (that) important: Binary ties and sample selection*. Accessed 2022-05-16. 2011. URL: https://toreopsahl.com/2011/08/12/why-anchorage-is-not-that-important-binary-ties-and-sample-selection/.

[59] Gergely Palla et al. "Directed network modules". In: *New Journal of Physics* 9.6 (June 2007), pp. 186–186. DOI: 10.1088/1367-2630/9/6/186.

[60] John Palowitch, Shankar Bhamidi, and Andrew B. Nobel. "Significance-based community detection in weighted networks". In: *Journal of Machine Learning Research* 18.188 (2018), pp. 1–48. URL: http://jmlr.org/papers/v18/17-377.html.

[61] Tiago P. Peixoto. "Bayesian Stochastic Blockmodeling". In: *Advances in Network Clustering and Blockmodeling*. John Wiley & Sons, Ltd, 2019. Chap. 11, pp. 289–332. DOI: 10.1002/9781119483298.ch11.

[62] Supun Perera, Michael G.H. Bell, and Michiel C.J. Bliemer. "Network science approach to modelling the topology and robustness of supply chain networks: a review and perspective". In: *Applied Network Science* 2.1 (Oct. 2017), p. 33. ISSN: 2364-8228. DOI: 10.1007/s41109-017-0053-0.

[63] 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*. KDD '14. New York, New York, USA: Association for Computing Machinery, 2014, pp. 701–710. DOI: 10.1145/2623330.2623732.

[64] Stephany Rajeh et al. "Analyzing Community-Aware Centrality Measures Using the Linear Threshold Model". In: *Complex Networks & Their Applications X*. Ed. by Rosa Maria Benito et al. Cham: Springer International Publishing, 2022, pp. 342–353. DOI: 10.1007/978-3-030-93409-5_29.

[65] Stephany Rajeh et al. "Comparing Community-Aware Centrality Measures in Online Social Networks". In: *Computational Data and Social Networks*. Ed. by David Mohaisen and Ruoming Jin. Cham: Springer International Publishing, 2021, pp. 279–290. DOI: 10.1007/978-3-030-91434-9_25.

[66] Martin Rosvall and Carl T. Bergstrom. "Maps of random walks on complex networks reveal community structure". In: *Proc. of the National Academy of Sciences* 105.4 (2008), pp. 1118–1123. DOI: 10.1073/pnas.0706851105.

[67] Martin Rosvall and Carl T. Bergstrom. "Multilevel Compression of Random Walks on Networks Reveals Hierarchical Organization in Large Integrated Systems". In: *PLOS ONE* 6.4 (Apr. 2011), pp. 1–10. DOI: 10.1371/journal.pone.0018209.

[68] Martin Rosvall et al. "Memory in network flows and its effects on spreading dynamics and community detection". In: *Nature Communications* 5.1 (Aug. 2014), p. 4630. DOI: 10.1038/ncomms5630.

[69] A. I. Saltykov. "The number of components in a random bipartite graph". In: 5.6 (1995), pp. 515–524. DOI: 10.1515/dma.1995.5.6.515.

[70] Ingo Scholtes. "When is a Network a Network? Multi-Order Graphical Model Selection in Pathways and Temporal Networks". In: *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data*

Mining. KDD '17. Halifax, NS, Canada: Association for Computing Machinery, 2017, pp. 1037–1046. DOI: 10.1145/3097983.3098145.

[71] D. Sculley and C.E. Brodley. "Compression and machine learning: a new perspective on feature space vectors". In: *Data Compression Conference (DCC'06)*. 2006, pp. 332–341. DOI: 10.1109/DCC.2006.13.

[72] Claude Elwood Shannon. "A mathematical theory of communication". In: *The Bell System Technical Journal* 27.3 (1948), pp. 379–423. DOI: 10.1002/j.1538-7305.1948.tb01338.x.

[73] Michael Sipser. *Introduction to the Theory of Computation*. Vol. 27. 1. ACM New York, NY, USA, 1996, pp. 27–29. ISBN: 978-0-6192-1764-8.

[74] Jelena Smiljanić and Marija Mitrović Dankulov. "Associative nature of event participation dynamics: A network theory approach". In: *PLOS ONE* 12.2 (Feb. 2017), pp. 1–16. DOI: 10.1371/journal.pone.0171565.

[75] Martin Summer. "Financial Contagion and Network Analysis". In: *Annual Review of Financial Economics* 5.1 (2013), pp. 277–297. DOI: 10.1146/annurev-financial-110112-120948.

[76] Shazia Tabassum et al. "Social network analysis: An overview". In: *WIREs Data Mining and Knowledge Discovery* 8.5 (2018), e1256. DOI: https://doi.org/10.1002/widm.1256.

[77] Jian Tang et al. "LINE: Large-Scale Information Network Embedding". In: *Proceedings of the 24th International Conference on World Wide Web*. WWW '15. Florence, Italy, 2015, pp. 1067–1077. DOI: 10.1145/2736277.2741093.

[78] Leo Torres et al. "The Why, How, and When of Representations for Complex Systems". In: *SIAM Review* 63.3 (2021), pp. 435–485. DOI: 10.1137/20M1355896.

[79] V. A. Traag, L. Waltman, and N. J. van Eck. "From Louvain to Leiden: guaranteeing well-connected communities". In: *Scientific Reports* 9.1 (Mar. 2019), p. 5233. DOI: 10.1038/s41598-019-41695-z.

[80] Jason M. Tylianakis and Rebecca J. Morris. "Ecological Networks Across Environmental Gradients". In: *Annual Review of Ecology, Evolution, and Systematics* 48.1 (2017), pp. 25–48. DOI: 10.1146/annurev-ecolsys-110316-022821.

[81] Alcides Viamontes Esquivel. "Narrowing the gap between network models and real complex systems". PhD thesis. Umeå University, 2014. ISBN: 978-91-7601-085-3.

[82] Alcides Viamontes Esquivel and Martin Rosvall. "Compression of Flow Can Reveal Overlapping-Module Organization in Networks". In: *Physical Review X* 1 (2 Dec. 2011), p. 021025. DOI: 10.1103/PhysRevX.1.021025.

[83] William Vickrey. "Counterspeculation, auctions, and competitive sealed tenders". In: *The Journ. of finance* 16.1 (1961), pp. 8–37. DOI: 10.1111/j.1540-6261.1961.tb02789.x.

[84] Nguyen Xuan Vinh, Julien Epps, and James Bailey. "Information Theoretic Measures for Clusterings Comparison: Is a Correction for Chance Necessary?" In: *Proceedings of the 26th Annual International Conference on Machine Learning*. ICML '09. Montreal, Quebec, Canada: Association for Computing Machinery, 2009, pp. 1073–1080. DOI: 10.1145/1553374.1553511.

[85] Wei Wang et al. "Predicting the epidemic threshold of the susceptible-infected-recovered model". In: *Scientific reports* 6.1 (2016), pp. 1–12. DOI: 10.1038/srep24676.

[86] Xuanhui Wang et al. "DirichletRank: Solving the Zero-One Gap Problem of PageRank". In: *ACM Transactions on Information Systems* 26.2 (Apr. 2008). DOI: 10.1145/1344411.1344416.

[87] Duncan J. Watts and Steven H. Strogatz. "Collective dynamics of 'small-world' networks". In: *Nature* 393.6684 (June 1998), pp. 440–442. DOI: 10.1038/30918.

[88] Wikipedia. *English alphabet*. Accessed 2022-03-23. 2022. URL: https://en.wikipedia.org/wiki/English_alphabet.

[89] Zhiying Zhao et al. "A community-based approach to identifying influential spreaders". In: *Entropy* 17.4 (2015), pp. 2228–2252. DOI: 10.3390/e17042228.

[90] Tao Zhou. "Progresses and challenges in link prediction". In: *iScience* 24.11 (2021), p. 103217. DOI: 10.1016/j.isci.2021.103217.