Comparing Hierarchical Network Partitions Based on Relative Entropy

Christopher Blöcker and Ingo Scholtes
Chair of Machine Learning for Complex Networks
Center for Artificial Intelligence and Data Science (CAIDAS)
Julius-Maximilians-Universität Würzburg
DE-97070 Würzburg, Germany
{christopher.bloecker,ingo.scholtes}@uni-wuerzburg.de

Abstract

Networks model the interconnected entities in systems and can be partitioned differently, prompting ways to compare partitions. Common partition similarity measures, such as the Jaccard index and variants of mutual information, are essentially based on measuring set overlaps. However, they ignore link patterns which are essential for the organisation of networks. We propose flow divergence, an information-theoretic divergence measure for comparing (hierarchical) network partitions, inspired by the ideas behind the Kullback-Leibler divergence and the description of random walks. Flow divergence adopts a coding perspective and compares network partitions A and B by considering the expected extra number of bits required to describe a random walk on a network using an estimate B of the network's assumed true partition A. We show that flow divergence distinguishes between partitions that traditional measures consider equally good when compared to a reference partition.

1 Introduction

Many real-world complex networks have communities: groups of nodes more linked to each other than to the rest. Communities capture link patterns and abstract from groups of individual nodes, revealing how networks are organised at the mesoscale. For example, tightly-knit groups of friends in social networks, groups of interacting proteins in biological networks, or traders who perform transactions in financial networks form communities. Motivated by various use cases and based on different assumptions, a plethora of ways to characterise what constitutes a community exist [1]. Naturally, based on their assumptions, different methods partition the same network differently; running a stochastic method on the same network several times can return different partitions. Consequently, we need measures to compare partitions and evaluate to what extent they agree and how they differ.

Researchers across scientific fields have proposed many partition similarity measures [2–7]. Arguably, the most commonly used measures are the Jaccard index [2] and information-theoretic measures based on mutual information [5, 7–9]. However, in the context of community detection, they have a crucial shortcoming: while communities are based on grouping nodes with similar link patterns, popular partition similarity measures ignore links altogether [3]. Instead, they merely consider how well the communities in different partitions coincide, essentially measured in terms of set overlaps [10, 11].

To address this shortcoming, we develop *flow divergence*, a partition dissimilarity measure based on the description of random walks. Combining the principles behind the Kullback-Leibler (KL) divergence [8] and the map equation for community detection [12], flow divergence quantifies the expected extra number of bits required to describe a random walk on a network when using an estimate B of the network's assumed true community structure A. Based on describing random walks, flow divergence naturally considers the network's link patterns and compares two-level and hierarchical partitions. Moreover, it quantifies contributions to the divergence on a per-node basis.

C. Blöcker and I. Scholtes, Comparing Hierarchical Network Partitions Based on Relative Entropy (Extended Abstract). Presented at the Third Learning on Graphs Conference (LoG 2024), Virtual Event, November 26–29, 2024.

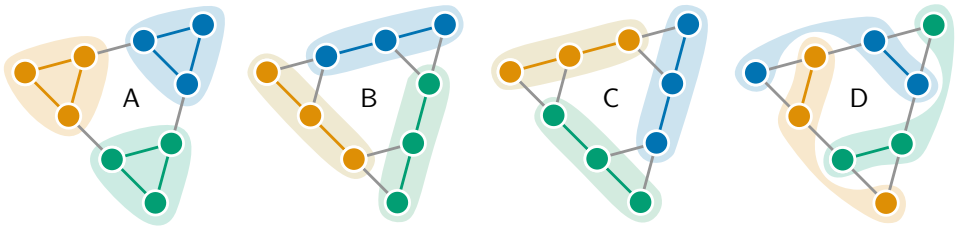

Figure 1: Four different partitions for the same network. Because common measures, such as the Jaccard index and mutual information, consider merely node labels but ignore link patterns, they consider partitions B, C, and D as equally good when compared against the reference partition A.

2 Related Work

Comparing partitions is a recurring problem in applications across scientific domains and has received much attention [2–6, 8, 13]. Given two partitions A and B of the same set X with $n = |X|$ objects, $\bigcup_{\mathsf{a}\in\mathsf{A}} \mathsf{a} = X = \bigcup_{\mathsf{b}\in\mathsf{B}} \mathsf{b}$, $\forall \mathsf{a}, \mathsf{a}' \in \mathsf{A} : \mathsf{a} \neq \mathsf{a}' \rightarrow \mathsf{a} \cap \mathsf{a}' = \emptyset$, and $\forall \mathsf{b}, \mathsf{b}' \in \mathsf{B} : \mathsf{b} \neq \mathsf{b}' \rightarrow \mathsf{b} \cap \mathsf{b}' = \emptyset$, the aim is to measure how similar those two partitions A and B are.

Measures for comparing partitions can be categorised into pair-counting, set-matching, and information-theoretic measures [14, 15]. A popular pair-counting measure is the so-called Rand index [16], which considers all possible pairs of objects $x, x' \in X$, counts t as the number of pairs where A and B agree whether x and x' belong to the same group, and defines the similarity between A and B as $R(\mathsf{A}, \mathsf{B}) = t/\binom{n}{2}$. The rand index can be computed in $\mathcal{O}(n)$. Two of the arguably most common partition similarity measures in network science are the set-matching approach known as the Jaccard index [2] and information-theoretic scores based on mutual information [5, 7–9, 17]. The Jaccard index computes the agreement between groups $\mathsf{a} \in \mathsf{A}$ and $b \in \mathsf{B}$ as $J(\mathsf{a}, \mathsf{b}) = |\mathsf{a} \cap \mathsf{b}| \,/\, |\mathsf{a} \cup \mathsf{b}|$. Computing the similarity between partitions A and B requires finding the best match $\mathsf{b} \in \mathsf{B}$ for each $\mathsf{a} \in \mathsf{A}$ and weighing according to a's size: $J(\mathsf{A}, \mathsf{B}) = \sum_{\mathsf{a}\in\mathsf{A}} \frac{|\mathsf{a}|}{n} \max_{\mathsf{b}\in\mathsf{B}} J(\mathsf{a}, \mathsf{b})$. The Jaccard index can be computed in $\mathcal{O}(n+|\mathsf{A}|\times|\mathsf{B}|)$. Mutual information, which is the basis for several information-theoretic measures, considers how much information the objects' assignments under A provide about their assignment under B: $I(\mathsf{A}, \mathsf{B}) = \sum_{\mathsf{a}\in\mathsf{A}} \sum_{\mathsf{b}\in\mathsf{B}} P(\mathsf{a}, \mathsf{b}) \log_2 \frac{P(\mathsf{a},\mathsf{b})}{P(\mathsf{a})P(\mathsf{b})}$, where $P(\mathsf{a}) = |\mathsf{a}| \,/n$ and $P(\mathsf{b}) = |\mathsf{b}| \,/n$ are the probabilities of selecting a and b, respectively, when choosing an object $x \in X$ at random; $P(\mathsf{a}, \mathsf{b}) = |\mathsf{a} \cap \mathsf{b}| \,/n$ is the joint probability of a and b. Mutual information can be computed in $\mathcal{O}(n+|\mathsf{A}|\times|\mathsf{B}|)$. Normalised mutual information scales the mutual information score to the interval $[0, 1]$, adjusted mutual information (AMI) additionally adjusts it for chance [7, 15].

The Rand index, Jaccard index, and mutual information ignore link patterns because they only consider group memberships. To see why this is an issue when comparing communities that summarise link patterns, consider the example network shown in Figure 1: a network with nine nodes, partitioned in four different ways. We assume that partition A represents the network's true community structure and compare partitions B, C, and D to A. The Rand index and Jaccard index are oblivious to the alternative partitions and judge them to match the reference partition equally well, $R(\mathsf{A}, \mathsf{B}) = R(\mathsf{A}, \mathsf{C}) = R(\mathsf{A}, \mathsf{D}) = \frac{2}{3}$, and $J(\mathsf{A}, \mathsf{B}) = J(\mathsf{A}, \mathsf{C}) = J(\mathsf{A}, \mathsf{D}) = \frac{1}{2}$. Mutual information suffers from the same issue, $I(\mathsf{A}, \mathsf{B}) = I(\mathsf{A}, \mathsf{C}) = I(\mathsf{A}, \mathsf{D}) \approx 0.46$. However, when evaluated with community detection in mind, it seems plausible that B and C agree with A to the same extent because of symmetry, but partition D should be distinguished from them because it captures a different pattern.

3 Flow Divergence

To define our partition dissimilarity score, *flow divergence*, we combine the map equation for flow-based community detection [12] with the Kullback-Leibler (KL) divergence [8], also known as relative entropy, defined as $D_{KL}(P \,||\, Q) = \sum_{x\in X} p_x \log_2 \frac{p_x}{q_x}$. Here P and Q are probability distributions that are defined on the same sample space X, where p_x and q_x, respectively, are the probabilities for drawing $x \in X$ when sampling from X. The KL-divergence quantifies the expected additional number of bits required to describe samples from X using an estimate Q of its true frequencies P. Following this idea, we define flow divergence to quantify the expected additional number of bits required to describe a random walk on a network using an estimate B of its "true" partition A.

Random-Walk Description Length. Let $G = (V, E, \delta)$ be a connected graph with nodes V, links E, and link weights $\delta\colon E \to \mathbb{R}^+$. Further, let $P = \{p_v \,|\, v \in V\}$ be the set of ergodic node visit rates, which can be computed by solving the recursive set of equations $p_v = \sum_{u \in V} p_u t_{uv}$, where $t_{uv} = \delta(u,v) / \sum_{v \in V} \delta(u,v)$ is the probability that a random walker at node u steps to node v [18]. In weakly connected graphs, we can use the PageRank algorithm [19] or so-called smart teleportation [20] to calculate the visit rates. Describing the random walker's position on the graph can be done with $H(P) = -\sum_{v \in V} p_v \log_2 p_v$ bits per step, where H is the Shannon entropy [12, 21].

Combining the above equations, we explicitly relate describing the random walker's position to transitions along links, using $H(P) = -\sum_{u \in V} p_u \sum_{v \in V} t_{uv} \log_2 p_v$ bits. To turn the coding dependent on communities, also called modules, we introduce a parameter M: the partition of the network's nodes into modules. We denote the module-dependent transition probability for stepping from u to v as $s(\mathsf{M}, u, v)$ such that $\log_2 s(\mathsf{M}, u, v)$ is the cost in bits for encoding a step from u to v,

$$H(\mathsf{M}) = -\sum_{u \in V} p_u \sum_{v \in V} t_{uv} \log_2 s(\mathsf{M}, u, v)\,. \tag{1}$$

The Map Equation for Modular Coding. The map equation [12, 18] is an information-theoretic objective function for flow-based community detection that provides a way to define a modular coding scheme. The map equation identifies communities by minimising the description of random walks on networks: Without communities, the codelength is the Shannon entropy over the nodes' visit rates. For a two-level partition, the map equation calculates the random walk's per-step description length L—also called *codelength*—as a weighted average of the modules' entropies and the entropy at the so-called index level for switching between modules (see Appendix A for an example),

$$L(\mathsf{M}) = qH(Q) + \sum_{\mathsf{m} \in \mathsf{M}} p_{\mathsf{m}} H(P_{\mathsf{m}})\,. \tag{2}$$

Here, $q = \sum_{\mathsf{m} \in \mathsf{M}} q_{\mathsf{m}}$ is the index-level codebook usage rate, q_{m} is the entry rate for module m, $Q = \{q_{\mathsf{m}}/q \,|\, \mathsf{m} \in \mathsf{M}\}$ is the set of module entry rates, $p_{\mathsf{m}} = \mathsf{m}_{\text{exit}} + \sum_{u \in \mathsf{m}} p_u$ is module m's codebook usage rate, m_{exit} is module m's exit rate, and $P_{\mathsf{m}} = \{\mathsf{m}_{\text{exit}}/p_{\mathsf{m}}\} \cup \{p_u/p_{\mathsf{m}} \,|\, u \in \mathsf{m}\}$ is the set of node visit rates in module m, including its exit rate. Through recursion, the map equation generalises to hierarchical partitions where modules can contain further submodules [18, 22].

To apply the ideas behind the KL divergence, we rewrite the map equation to match the form of Equation (1), see Appendix B for the details:

$$L(\mathsf{M}) = qH(Q) + \sum_{\mathsf{m} \in \mathsf{M}} p_{\mathsf{m}} H(P_{\mathsf{m}}) = -\sum_{u \in V} p_u \sum_{v \in V} t_{uv} \log_2 \operatorname{mapsim}(\mathsf{M}, u, v)\,, \tag{3}$$

where $\log_2 \operatorname{mapsim}(\mathsf{M}, u, v)$ is the number of bits required to describe a step from node u to v, given partition M [23]. For two-level partitions, mapsim, which is derived from the map equation and shorthand for *map equation similarity*, is defined as

$$\operatorname{mapsim}(\mathsf{M}, u, v) = \delta_{\mathsf{m}_u, \mathsf{m}_v} \frac{p_v}{p_{\mathsf{m}_v}} + (1 - \delta_{\mathsf{m}_u, \mathsf{m}_v}) \left(\frac{\mathsf{m}_{u,\text{exit}}}{p_{\mathsf{m}_u}} \cdot \frac{q_{\mathsf{m}_v}}{q} \cdot \frac{p_v}{p_{\mathsf{m}_v}}\right), \tag{4}$$

where δ is the Kronecker delta, and m_u and m_v are the modules to which nodes u and v belong, respectively. Based on a network's map M, mapsim quantifies the rate at which a random walker transitions between pairs of nodes. Importantly, mapsim depends on the source node's module, but not on the source node itself; mapsim has also been generalised to hierarchical partitions [23].

Relative Entropy Between Partitions. Different partitions, or maps, of the same network imply different random-walker dynamics with different codelengths. Following the idea of the KL divergence, we assume that A captures the random walker's "true" dynamics. And we ask: what is the expected extra number of bits required to describe a random walk using an estimate B of the true patterns A? With partition-dependent transition rates $t^{\mathsf{M}}_{uv} = \frac{\operatorname{mapsim}(\mathsf{M},u,v)}{\sum_{v(\neq u)} \operatorname{mapsim}(\mathsf{M},u,v)}$ (see Appendix C), we define our partition dissimilarity measure *flow divergence*, which naturally compares hierarhical and non-hierarchical partitions, as

$$D_F(\mathsf{A} \,||\, \mathsf{B}) = \sum_{u \in V} p_u \sum_{v \in V} t^{\mathsf{A}}_{uv} \log_2 \frac{\operatorname{mapsim}(\mathsf{A}, u, v)}{\operatorname{mapsim}(\mathsf{B}, u, v)} \tag{5}$$

Limitations and Complexity. Computing flow divergence requires considering mapsim scores between n^2 many node pairs, where $n = |V|$ is the number of nodes. However, it can be computed in $\mathcal{O}(m \cdot n)$, where m is the number of modules since mapsim depends on the source module but not on the specific source node. Therefore, we only need to consider $m \cdot n$ many pairs of source modules and target nodes instead of n^2 pairs of nodes (see Appendix D for details).

4 Evaluation

We apply flow divergence to compute partition dissimilarity scores for partitions in synthetic and real networks. In the synthetic network, we show that flow divergence can distinguish between partitions that popular measures consider equally good when compared to a reference partition. We also show that a partition with higher codelength can have lower flow divergence than partitions with lower codelengths, highlighting that flow divergence captures information that the map equation does not capture in the codelength. Our implementation of flow divergence is available on GitHub[1].

Synthetic Example. We return to our initial example and show in Table 1 that flow divergence can distinguish between partitions where the Jaccard index and mutual information cannot. Despite higher codelength, D is more similar to A than B and C. This is analogous to the fact that, for three probability distributions P, Q, and R defined on the same sample space X, R can be closer to P despite higher entropy than Q, that is, $H(Q) < H(R) \not\Rightarrow D_{KL}(P \,||\, Q) < D_{KL}(P \,||\, R)$. Better maps need not have lower flow divergence.

Table 1: Flow divergence in bits, rounded to two decimal places, between the partitions shown in Figure 1. We also list the partitions' codelengths, in bits, at the bottom.

	Other			
Reference	A	B	C	D
A	0	1.92	1.92	1.5
B	1.8	0	1.17	1.99
C	1.8	1.17	0	1.48
D	1.14	1.78	1.25	0
Codelength	2.86	3.73	3.73	4.47

To explain why D is more similar to A, we consider the individual nodes' contribution to the divergences. Instead of computing the outer sum in Equation (5), we report flow divergence values per node in Table 2 (see Appendix E). We find that, the divergence per node is lower for the nodes with higher flow in D than it is for B and C. Conversely, the per-node divergence for the lower-flow nodes is higher. This can be explained by the fact that the modules in A and D each overlap in two of the higher-degree nodes, resulting in more similar partitions than B and C which overlap with A in a lower-degree node and a higher-degree node.

Hierarchical Example. We include an example with a hierarchical partition in Appendix E.

The Cost of Overfitting. Real-world data is often incomplete, causing community-detection methods to report spurious partitions. We explore applying flow divergence for quantifying the cost of overfitting in the regime of incomplete data, see Appendix F.

5 Conclusion

We have studied the problem of comparing network partitions and motivated the need for approaches that take link patterns into account: while community detection focuses on grouping nodes that share similar link patterns, measures for comparing partitions typically ignore links altogether.

Inspired by the Kullback-Leibler divergence for measuring "distances" between probability distributions, we developed a partition dissimilarity score based on random walks: *flow divergence*. Flow divergence uses the map equation and quantifies the expected additional number of bits for describing a random walk when using an estimate B of the network's "true" community structure A. Based on random walks, flow divergence naturally compares hierarchical and non-hierarchical partitions.

Applied to synthetic networks, we showed that flow divergence distinguishes between partitions where popular partition similarity measures fail. In real networks, we highlighted how flow divergence gives insights into the cost of overfitting when detecting communities in incomplete network data.

[1] https://github.com/mapequation/map-equation-similarity

References

[1] Santo Fortunato. Community detection in graphs. *Physics Reports*, 486(3):75–174, 2010. ISSN 0370-1573. doi: 10.1016/j.physrep.2009.11.002. 1

[2] Paul Jaccard. The distribution of the flora in the alpine zone. *New Phytologist*, 11(2):37–50, 1912. doi: https://doi.org/10.1111/j.1469-8137.1912.tb05611.x. 1, 2

[3] Daniel Straulino, Mattie Landman, and Neave O'Clery. A bi-directional approach to comparing the modular structure of networks. *EPJ Data Science*, 10(1):13, Mar 2021. ISSN 2193-1127. doi: 10.1140/epjds/s13688-021-00269-8. 1

[4] Milad Malekzadeh and Jed A. Long. A network community structure similarity index for weighted networks. *PLOS ONE*, 18(11):1–17, 11 2023. doi: 10.1371/journal.pone.0292018.

[5] Juan Ignacio Perotti, Claudio Juan Tessone, and Guido Caldarelli. Hierarchical mutual information for the comparison of hierarchical community structures in complex networks. *Phys. Rev. E*, 92:062825, Dec 2015. doi: 10.1103/PhysRevE.92.062825. 1, 2

[6] Marina Meil. Comparing clusteringsan information based distance. *Journal of Multivariate Analysis*, 98(5):873–895, 2007. ISSN 0047-259X. doi: https://doi.org/10.1016/j.jmva.2006.11.013. 2

[7] Simone Romano, James Bailey, Vinh Nguyen, and Karin Verspoor. Standardized mutual information for clustering comparisons: One step further in adjustment for chance. In *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 1143–1151, Bejing, China, 22–24 Jun 2014. PMLR. 1, 2

[8] Thomas M. Cover and Joy A. Thomas. *Elements of information theory, second edition*. John Wiley & Sons, 2006. ISBN 978-0-471-24195-9. 1, 2, 12

[9] Nguyen Xuan Vinh, Julien Epps, and James Bailey. Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance. *Journal of Machine Learning Research*, 11(95):2837–2854, 2010. 1, 2

[10] Alcides Viamontes Esquivel and Martin Rosvall. Comparing network covers using mutual information, 2012. 1

[11] Valérie Poulin and François Théberge. Comparing graph clusterings: Set partition measures vs. graph-aware measures. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43 (6):2127–2132, 2021. doi: 10.1109/TPAMI.2020.3009862. 1

[12] Martin Rosvall and Carl T. Bergstrom. Maps of random walks on complex networks reveal community structure. *Proceedings of the National Academy of Sciences*, 105(4):1118–1123, 2008. doi: 10.1073/pnas.0706851105. 1, 2, 3, 8

[13] Claire Donnat and Susan Holmes. Tracking network dynamics: a survey of distances and similarity metrics, 2018. 2

[14] Hanneke van der Hoef and Matthijs J. Warrens. Understanding information theoretic measures for comparing clusterings. *Behaviormetrika*, 46(2):353–370, Oct 2019. ISSN 1349-6964. doi: 10.1007/s41237-018-0075-7. 2

[15] 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, page 10731080, New York, NY, USA, 2009. Association for Computing Machinery. ISBN 9781605585161. doi: 10.1145/1553374.1553511. 2

[16] William M. Rand. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical Association*, 66(336):846–850, 1971. doi: 10.1080/01621459.1971.10482356. 2

[17] M. E. J. Newman, George T. Cantwell, and Jean-Gabriel Young. Improved mutual information measure for clustering, classification, and community detection. *Phys. Rev. E*, 101:042304, Apr 2020. doi: 10.1103/PhysRevE.101.042304. 2

[18] Jelena Smiljanić, Christopher Blöcker, Anton Holmgren, Daniel Edler, Magnus Neuman, and Martin Rosvall. Community detection with the map equation and infomap: Theory and applications. 2023. 3, 9

[19] David F. Gleich. Pagerank beyond the web. *SIAM Review*, 57(3):321–363, 2015. doi: 10.1137/140976649. 3

[20] R. Lambiotte and M. Rosvall. Ranking and clustering of nodes in networks with smart teleportation. *Phys. Rev. E*, 85:056107, May 2012. doi: 10.1103/PhysRevE.85.056107. 3

[21] C. E. Shannon. A mathematical theory of communication. *Bell Syst. Tech. J.*, 27:379–423, 1948. 3

[22] M. Rosvall and C. T. Bergstrom. Multilevel Compression of Random Walks on Networks Reveals Hierarchical Organization in Large Integrated Systems. *PLoS One*, 6:e18209, 2011. 3

[23] Christopher Blöcker, Jelena Smiljanić, Ingo Scholtes, and Martin Rosvall. Similarity-based link prediction from modular compression of network flows. In *Proceedings of the First Learning on Graphs Conference*, volume 198 of *Proceedings of Machine Learning Research*, pages 52:1–52:18. PMLR, 09–12 Dec 2022. 3, 9

[24] Amir Ghasemian, Homa Hosseinmardi, and Aaron Clauset. Evaluating overfit and underfit in models of network community structure. *IEEE Transactions on Knowledge and Data Engineering*, 32(9):1722–1735, 2020. doi: 10.1109/TKDE.2019.2911585. 10

[25] Jelena Smiljani, Christopher Blöcker, Daniel Edler, and Martin Rosvall. Mapping flows on weighted and directed networks with incomplete observations. *Journal of Complex Networks*, 9 (6), 2021. doi: 10.1093/comnet/cnab044. 12, 13

[26] Roger Guimerà, Marta Sales-Pardo, and Luís A. Nunes Amaral. Modularity from fluctuations in random graphs and complex networks. *Phys. Rev. E*, 70:025101, Aug 2004. doi: 10.1103/PhysRevE.70.025101.

[27] Jelena Smiljanić, Daniel Edler, and Martin Rosvall. Mapping flows on sparse networks with missing links. *Phys. Rev. E*, 102:012302, Jul 2020. doi: 10.1103/PhysRevE.102.012302. 13

[28] M. Girvan and M. E. J. Newman. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences*, 99(12):7821–7826, 2002. doi: 10.1073/pnas.122653799. 14

[29] Pablo M. Gleiser and Leon Danon. Community structure in jazz. *Advances in Complex Systems*, 06(04):565–573, 2003. doi: 10.1142/S0219525903001067. 14

[30] Piotr Sapiezynski, Arkadiusz Stopczynski, David Dreyer Lassen, and Sune Lehmann. Interaction data from the copenhagen networks study. *Scientific Data*, 6(1):315, Dec 2019. ISSN 2052-4463. doi: 10.1038/s41597-019-0325-x. 14

[31] Michael Fire and Rami Puzis. Organization mining using online social networks. *Networks and Spatial Economics*, 16(2):545–578, Jun 2016. ISSN 1572-9427. doi: 10.1007/s11067-015-9288-4. 14

[32] Daniel Edler, Ludvig Bohlin, and Martin Rosvall. Mapping Higher-Order Network Flows in Memory and Multilayer Networks with Infomap. *Algorithms*, 10:112, 2017. 14

[33] Daniel Edler, Anton Holmgren, and Martin Rosvall. The MapEquation software package. `https://mapequation.org`. 14

A Map Equation Coding Example

Conceptually, the map equation is based on minimising the description of random walks. Consider a communication game where the sender updates the receiver about the location of the random walker on the network after each step. This can be done by assigning unique codewords to the nodes and communicating one codeword per random-walker step as shown in Figure 2(a). With this approach, the required number of bits per step is given by the entropy over the nodes' visit rates.

However, networks often have communities, which enable designing a more efficient approach: Assume that we have a set of modules that capture the network's structure, corresponding to the random walker's movement patterns. Then we assign unique codewords within modules but can reuse the same codewords for different nodes in different modules. For a uniquely decodable code, we need to introduce codewords to encode for module exits as well as a designated codebook, the so-called index-level codebook, for encoding module entries as shown in Figure 2(b).

We can draw network partitions, or "maps", as trees as shown in Figure 2c. Each random-walker step along a link in the network corresponds to traversing the map along the shortest path between two nodes; to describe the step, we use the codewords along the shortest path in the map.

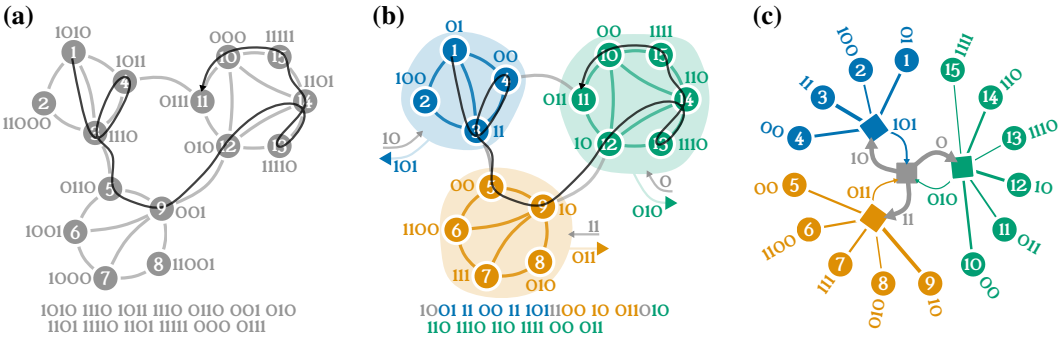

Figure 2: Illustration of encoding random walks and the principles behind the map equation. **(a)** Nodes are not partitioned into communities. We derive unique codewords from the nodes' visit rates and use them to describe the shown random-walk sequence with the codewords at the bottom. **(b)** Nodes are partitioned into three communities and receive codewords that are unique within each community. Codewords for entering and exiting communities are shown next to arrows that point into and out of the communities. **(c)** The map corresponding to the community structure and coding scheme from **(b)**, drawn as a radial tree. Link widths are proportional to module-normalised codeword usage rates. Good maps have small module exit rates.

B Rewriting the Map Equation

Let $G = (V, E, \delta)$ be a network with nodes V, links E, and links weights $\delta\colon E \to \mathbb{R}^+$. If G is undirected, we can calculate the stationary visit rate for node u as $p_u = \frac{\sum_{v \in V} \delta(u,v)}{\sum_{u \in V} \sum_{v \in V} \delta(u,v)}$. If G is directed, we can use a power iteration to solve the recursive set of equations $p_v = \sum_{u \in V} p_u t_{uv}$, where t_{uv} is the probability that a random walker at u steps to v.

We begin with the two-level map equation [12],

$$L(\mathsf{M}) = qH(Q) + \sum_{\mathsf{m} \in \mathsf{M}} p_{\mathsf{m}} H(P_{\mathsf{m}}),$$

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

Expanding the map equation, we obtain

$$-q \sum_{\mathsf{m} \in \mathsf{M}} \frac{q_{\mathsf{m}}}{q} \log_2 \frac{q_{\mathsf{m}}}{q} - \sum_{\mathsf{m} \in \mathsf{M}} p_{\mathsf{m}} \left(\frac{\mathsf{m}_{\text{exit}}}{p_{\mathsf{m}}} \log_2 \frac{\mathsf{m}_{\text{exit}}}{p_{\mathsf{m}}} + \sum_{u \in \mathsf{m}} \frac{p_u}{p_{\mathsf{m}}} \log_2 \frac{p_u}{p_{\mathsf{m}}} \right),$$

and after cancelling common factors

$$-\sum_{\mathsf{m} \in \mathsf{M}} q_{\mathsf{m}} \log_2 \frac{q_{\mathsf{m}}}{q} - \sum_{\mathsf{m} \in \mathsf{M}} \left(\mathsf{m}_{\text{exit}} \log_2 \frac{\mathsf{m}_{\text{exit}}}{p_{\mathsf{m}}} + \sum_{u \in \mathsf{m}} p_u \log_2 \frac{p_u}{p_{\mathsf{m}}} \right).$$

Pulling out the summation and annotating the parts, we have

$$-\sum_{\mathsf{m} \in \mathsf{M}} \overbrace{q_{\mathsf{m}} \log_2 \frac{q_{\mathsf{m}}}{q}}^{\text{entering module } \mathsf{m}} + \overbrace{\mathsf{m}_{\text{exit}} \log_2 \frac{\mathsf{m}_{\text{exit}}}{p_{\mathsf{m}}}}^{\text{exiting module } \mathsf{m}} + \overbrace{\sum_{u \in \mathsf{m}} p_u \log_2 \frac{p_u}{p_{\mathsf{m}}}}^{\text{visiting nodes in module } \mathsf{m}}.$$

Next, we use $q_{\mathsf{m}} = \sum_{u \notin \mathsf{m}} p_u \sum_{v \in \mathsf{m}} t_{uv}$, $\mathsf{m}_{\text{exit}} = \sum_{u \in \mathsf{m}} p_u \sum_{v \notin \mathsf{m}} t_{uv}$, and $p_v = \sum_u p_u t_{uv}$, and split up the last part,

$$-\sum_{\mathsf{m} \in \mathsf{M}} \left(\sum_{u \notin \mathsf{m}} p_u \sum_{v \in \mathsf{m}} t_{uv} \log_2 \frac{q_{\mathsf{m}}}{q} \right) + \left(\sum_{u \in \mathsf{m}} p_u \sum_{v \notin \mathsf{m}} t_{uv} \log_2 \frac{\mathsf{m}_{\text{exit}}}{p_{\mathsf{m}}} \right) + \left(\sum_{u \in \mathsf{m}} p_u \sum_{v \in \mathsf{m}} t_{uv} \log_2 \frac{p_v}{p_{\mathsf{m}}} \right) + \left(\sum_{u \notin \mathsf{m}} p_u \sum_{v \in \mathsf{m}} t_{uv} \log_2 \frac{p_v}{p_{\mathsf{m}}} \right).$$

Then, we merge the second and fourth term into the first term. To merge the second term, we turn the module exits around, considering those steps that leave other modules to enter module m instead of steps that leave module m. We denote node u's and v's module by m_u and m_v, respectively,

$$-\sum_{\mathsf{m} \in \mathsf{M}} \left(\sum_{u \notin \mathsf{m}} p_u \sum_{v \in \mathsf{m}} t_{uv} \log_2 \Big(\overbrace{\frac{q_{\mathsf{m}_u}}{p_{\mathsf{m}_u}}}^{\text{exiting } \mathsf{m}_u} \cdot \overbrace{\frac{q_{\mathsf{m}_v}}{q}}^{\text{entering } \mathsf{m}_v} \cdot \overbrace{\frac{p_v}{p_{\mathsf{m}_v}}}^{\text{visiting node } v} \Big) \right) + \left(\sum_{u \in \mathsf{m}} p_u \sum_{v \in \mathsf{m}} t_{uv} \log_2 \frac{p_v}{p_{\mathsf{m}_v}} \right).$$

We realise that, for each module m, we sum over all nodes $u \notin \mathsf{m}$ and all nodes $u \in \mathsf{m}$, and, depending on whether they are a member of m, calculate the cost for transitioning to $v \in \mathsf{m}$ differently. We rewrite these two cases using the Kronecker delta δ, summing over all nodes u,

$$-\sum_{\mathsf{m} \in \mathsf{M}} \sum_u p_u \sum_{v \in \mathsf{m}} t_{uv} \left[(1 - \delta_{\mathsf{m}_u, \mathsf{m}_v}) \log_2 \left(\frac{q_{\mathsf{m}_u}}{p_{\mathsf{m}_u}} \cdot \frac{q_{\mathsf{m}_v}}{q} \cdot \frac{p_v}{p_{\mathsf{m}_v}} \right) + \delta_{\mathsf{m}_u, \mathsf{m}_v} \log_2 \frac{p_v}{p_{\mathsf{m}_v}} \right].$$

Finally, instead of summing over all modules and all nodes in each module, we sum over all nodes directly and can calculate the codelength for partition M as

$$\begin{aligned} L(\mathsf{M}) &= -\sum_u p_u \sum_v t_{uv} \left[(1 - \delta_{\mathsf{m}_u, \mathsf{m}_v}) \log_2 \left(\frac{q_{\mathsf{m}_u}}{p_{\mathsf{m}_u}} \cdot \frac{q_{\mathsf{m}_v}}{q} \cdot \frac{p_v}{p_{\mathsf{m}_v}} \right) + \delta_{\mathsf{m}_u, \mathsf{m}_v} \log_2 \frac{p_v}{p_{\mathsf{m}_v}} \right] \\ &= -\sum_u p_u \sum_v t_{uv} \log_2 \left[(1 - \delta_{\mathsf{m}_u, \mathsf{m}_v}) \left(\frac{q_{\mathsf{m}_u}}{p_{\mathsf{m}_u}} \cdot \frac{q_{\mathsf{m}_v}}{q} \cdot \frac{p_v}{p_{\mathsf{m}_v}} \right) + \delta_{\mathsf{m}_u, \mathsf{m}_v} \frac{p_v}{p_{\mathsf{m}_v}} \right]. \end{aligned}$$

The part inside the square brackets is known as map equation similarity, or *mapsim* for short, an information-theoretic measure for node similarity [23]. That is, we can calculate the codelength for a partition M using mapsim, where $\log_2 \mathrm{mapsim}\,(\mathsf{M}, u, v)$ quantifies how many bits are required to describe a random-walker-transition from node u to v, given partition M,

$$L\,(\mathsf{M}) = -\sum_u p_u \sum_v t_{uv} \log_2 \mathrm{mapsim}\,(\mathsf{M}, u, v)\,.$$

C Partition-Dependent Transition Rates

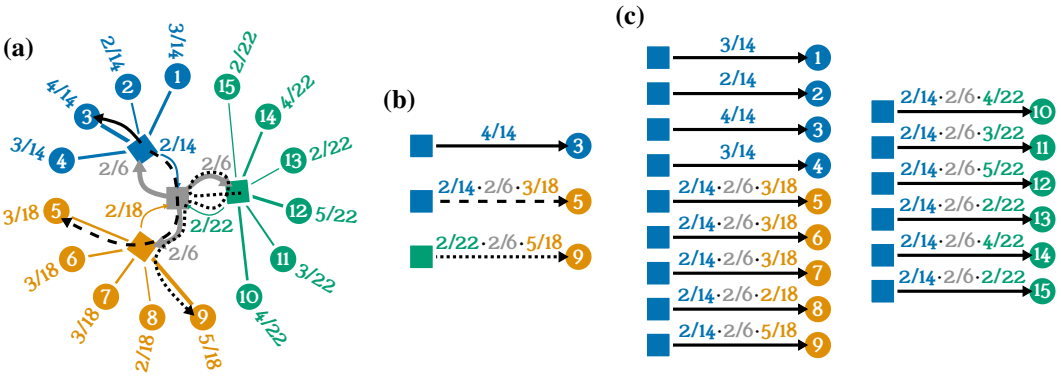

Figure 3: Walking on maps. **(a)** The same map as in Figure 2c, but now annotated with module-normalised node visit rates instead of codewords. The solid, dashed, and dotted arrows show examples of three random-walker paths on the map. **(b)** We derive transition rates between pairs of nodes according to mapsim: Transition rates depend on the source node's module, not on the source node itself [18, 23]. Therefore, the shortest paths on the map start at module nodes and we obtain the rate at which each shortest path is used by multiplying the transition rates along that path. The dotted arrow is not a shortest path because it contains a loop which we discard for efficient coding. **(c)** All shortest paths that start in the blue module and their usage rates.

Consider the map shown in Figure 3a where nodes are annotated with their module-normalised visit rates and arrows between modules show the modules' entry and exit rates. Three black arrows are drawn on the map: a solid arrow for a random walker who is in the blue module and steps to node 3, a dashed arrow for a random walker who is in the blue module and steps to node 5 in the orange module, and a dotted arrow for a random walker who is in the green module and visits node 9 in the orange module. Because codewords depend on the random walker's current module, but not on the current node, shortest paths in the map begin at the square-shaped module nodes.

Figure 3b details at what rates a random walker uses the solid, dashed, and dotted paths in the map. A random walker who is in the blue module visits node 3 at rate $\frac{4}{14}$. A random walker who is in the blue module exits at rate $\frac{2}{14}$, then enters the orange module at rate $\frac{2}{6}$, and then visits node 5 at rate $\frac{3}{18}$, resulting in a rate of $\frac{2}{14} \cdot \frac{2}{6} \cdot \frac{3}{18}$ for the dashed arrow. The dotted arrow contains a loop, which we discard because we describe transitions along the shortest paths in the map. Therefore, a random walker who is in the green module exits at rate $\frac{2}{22}$, then enters the orange module at rate $\frac{2}{6}$, and visits node 9 at rate $\frac{5}{18}$, resulting in a rate of $\frac{2}{22} \cdot \frac{2}{6} \cdot \frac{5}{18}$ for the dotted arrow. Taking the $\log_2$ of the arrows' usage rates returns the required number of bits for describing the corresponding step.

Figure 3c shows all shortest paths that start in the blue module together with their usage rates. However, since we only consider shortest paths in the map but not paths that contain loops, their usage rates do not sum to 1; here the rates for shortest paths starting in the blue module sum to approximately 0.94. Because paths with loops are ways to make detours from shortest paths and return to them, their usage rate is proportional to that of their contained shortest path. To obtain the transition probability from u to v according to partition M, we normalise with the sum of transition rates for shortest paths from u to all nodes v, except for u itself because we assume no self-links,

$$t^{\mathsf{M}}_{uv} = \frac{\mathrm{mapsim}\,(\mathsf{M}, u, v)}{\sum_{v(\neq u)} \mathrm{mapsim}\,(\mathsf{M}, u, v)}. \tag{6}$$

If we want to consider self-links, we also include u itself.

D Computational Complexity and Limitations

Complexity. Computing flow divergence requires considering mapsim scores between n^2 many node pairs, where $n = |V|$. The regularities in the networks' community structures and the details of mapsim make it possible to compute flow divergence in time $\mathcal{O}(m \cdot n)$, where m is the number of modules. Since $\mathrm{mapsim}(\mathsf{M}, u, v)$ depends on the source node's module m_u, but not the source node u itself, we obtain the same values for different source nodes u, u' in the same module when the target node v is the same. Therefore, the number of mapsim values we need to compute reduces to $m \cdot n$ pairs, where $m \ll n$ is the number of modules, which has been reported to scale as $m = \mathcal{O}(\sqrt{n})$ in real-world networks [24]. For each mapsim value, we need to find the source module m_u and target node v in the partition tree and multiply the random walker's transition rates along the shortest path from m_u to v. Since the network's community structure is organised in a tree, the paths from the root to all nodes, forwards and backwards, can be precomputed in time $\mathcal{O}(n)$.

Naïvely, flow divergence can be computed with worst-case time complexity $\mathcal{O}(n^2)$ because it involves a double sum over all nodes.

$$D_F(\mathsf{A} \,||\, \mathsf{B}) = \sum_{u \in V} p_u \sum_{v \in V} t^{\mathsf{A}}_{uv} \log_2 \frac{\mathrm{mapsim}(\mathsf{A}, u, v)}{\mathrm{mapsim}(\mathsf{B}, u, v)}. \tag{7}$$

However, by expanding and regrouping the terms to exploit redundancies in the coding structure, flow divergence can be computed in time $\mathcal{O}(m \cdot n)$. For notational clarity, we allow self-loops in the following derivations. We begin by substituting the definition of the partition-dependent transition rates,

$$t^{\mathsf{M}}_{uv} = \frac{\mathrm{mapsim}(\mathsf{M}, u, v)}{\sum_{v \in V} \mathrm{mapsim}(\mathsf{M}, u, v)}. \tag{8}$$

into Equation (7),

$$D_F(\mathsf{A} \,||\, \mathsf{B}) = \sum_{u \in V} p_u \sum_{v \in V} \frac{\mathrm{mapsim}(\mathsf{A}, u, v)}{\sum_{v \in V} \mathrm{mapsim}(\mathsf{A}, u, v)} \log_2 \frac{\mathrm{mapsim}(\mathsf{A}, u, v)}{\mathrm{mapsim}(\mathsf{B}, u, v)}. \tag{9}$$

We pull out the common normalisation factor for source node u from the second sum and call the entire factor ϕ_u,

$$D_F(\mathsf{A} \,||\, \mathsf{B}) = \sum_{u \in V} \underbrace{\frac{p_u}{\sum_v \mathrm{mapsim}(\mathsf{A}, u, v)}}_{=\phi_u} \sum_{v \in V} \mathrm{mapsim}(\mathsf{A}, u, v) \log_2 \frac{\mathrm{mapsim}(\mathsf{A}, u, v)}{\mathrm{mapsim}(\mathsf{B}, u, v)}. \tag{10}$$

The normalisation factor ϕ_u can be computed efficiently by exploiting the regularities in the modular network structure. As per the definition of mapsim (Equation (4)), we make use of two useful facts. First, that mapsim only depends on the source node's module m_u but not the source node itself, $\mathrm{mapsim}(\mathsf{M}, u, v) = \mathrm{mapsim}(\mathsf{M}, \mathsf{m}_u, v)$. And second, that mapsim can be decomposed into different parts that correspond to transitions between modules and visiting the target node, $\mathrm{mapsim}(\mathsf{M}, \mathsf{m}_u, v) = \mathrm{mapsim}(\mathsf{M}, \mathsf{m}_u, \mathsf{m}_v) \cdot \frac{p_v}{p_{\mathsf{m}_v}}$.

$$\begin{aligned}
\phi_u &= \frac{p_u}{\sum_{v \in V} \mathrm{mapsim}(\mathsf{A}, u, v)} && (11)\\
&= \frac{p_u}{\sum_{\mathsf{m} \in \mathsf{A}} \sum_{v \in \mathsf{m}} \mathrm{mapsim}(\mathsf{A}, \mathsf{m}_u, \mathsf{m}) \frac{p_v}{p_{\mathsf{m}}}} && (12)\\
&= \frac{p_u}{\sum_{\mathsf{m} \in \mathsf{A}} \mathrm{mapsim}(\mathsf{A}, \mathsf{m}_u, \mathsf{m}) \sum_{v \in \mathsf{m}} \frac{p_v}{p_{\mathsf{m}}}} && (13)\\
&= \frac{p_u}{\sum_{\mathsf{m} \in \mathsf{A}} \left(1 - \frac{\mathsf{m}_{\mathsf{exit}}}{p_{\mathsf{m}}}\right) \mathrm{mapsim}(\mathsf{A}, \mathsf{m}_u, \mathsf{m})} && (14)
\end{aligned}$$

Here, the last step follows from $p_{\mathsf{m}} = \mathsf{m}_{\mathsf{exit}} + \sum_{u \in \mathsf{m}} p_u$. Because mapsim depends on the source module $\mathsf{m_u}$ but not the individual source node u, we expect that we only need to consider $m^2 = \mathsf{O}(\mathsf{n})$ many pairs of modules. With normalisation factor ϕ_u, we have

$$D_F(\mathsf{A} \,||\, \mathsf{B}) = \sum_{u \in V} \phi_u \sum_{v \in V} \mathrm{mapsim}(\mathsf{A}, u, v) \log_2 \frac{\mathrm{mapsim}(\mathsf{A}, u, v)}{\mathrm{mapsim}(\mathsf{B}, u, v)}, \tag{15}$$

which shows that flow divergence is an expected KL divergence, one per node, weighted by the corresponding normalisation factor. Next, we apply logarithm rules to split up flow divergence into two terms per source node, and call them I and II,

$$D_F\left(\mathsf{A}\,||\,\mathsf{B}\right) = \sum_{u\in V}\phi_u\Bigg[\underbrace{\sum_{v\in V}\operatorname{mapsim}\left(\mathsf{A},u,v\right)\log_2\operatorname{mapsim}\left(\mathsf{A},u,v\right)}_{I} \tag{16}$$

$$\underbrace{-\sum_{v\in V}\operatorname{mapsim}\left(\mathsf{A},u,v\right)\log_2\operatorname{mapsim}\left(\mathsf{B},u,v\right)}_{II}\Bigg], \tag{17}$$

where, essentially, I is a term that considers the entropy of partition A, and II is a term that considers the cross-entropy between partitions A and B. However, I and II are not proper entropies because we have moved the normalisation factor into ϕ_u.

We consider parts I and II in turn, starting with part I. For simplicity, we drop the outer sum over all source nodes u, focusing on a single fixed source node u instead. Similar to before, instead of summing over all target nodes v, we sum over the target modules,

$$(I) = \sum_{v\in V}\operatorname{mapsim}\left(\mathsf{A},u,v\right)\log_2\operatorname{mapsim}\left(\mathsf{A},u,v\right) \tag{18}$$

$$= \sum_{m\in\mathsf{A}}\sum_{v\in\mathsf{m}}\operatorname{mapsim}\left(\mathsf{A},u,v\right)\log_2\operatorname{mapsim}\left(\mathsf{A},u,v\right), \tag{19}$$

pull out the last factor from the mapsim operators

$$= \sum_{\mathsf{m}\in\mathsf{A}}\sum_{v\in\mathsf{m}}\operatorname{mapsim}\left(\mathsf{A},\mathsf{m}_u,\mathsf{m}\right)\frac{p_v}{p_{\mathsf{m}}}\log_2\left(\operatorname{mapsim}\left(\mathsf{A},\mathsf{m}_u,\mathsf{m}\right)\frac{p_v}{p_{\mathsf{m}}}\right), \tag{20}$$

pull out common factors

$$= \sum_{\mathsf{m}\in\mathsf{A}}\operatorname{mapsim}\left(\mathsf{A},\mathsf{m}_u,\mathsf{m}\right)\sum_{v\in\mathsf{m}}\frac{p_v}{p_{\mathsf{m}}}\log_2\left(\operatorname{mapsim}\left(\mathsf{A},\mathsf{m}_u,\mathsf{m}\right)\frac{p_v}{p_{\mathsf{m}}}\right), \tag{21}$$

and apply logarithm rules and simplify

$$= \sum_{\mathsf{m}\in\mathsf{A}}\operatorname{mapsim}\left(\mathsf{A},\mathsf{m}_u,\mathsf{m}\right)\left[\sum_{v\in\mathsf{m}}\frac{p_v}{p_{\mathsf{m}}}\log_2\operatorname{mapsim}\left(\mathsf{A},\mathsf{m}_u,\mathsf{m}\right)+\sum_{v\in\mathsf{m}}\frac{p_v}{p_{\mathsf{m}}}\log_2\frac{p_v}{p_{\mathsf{m}}}\right] \tag{22}$$

$$= \sum_{\mathsf{m}\in\mathsf{A}}\operatorname{mapsim}\left(\mathsf{A},\mathsf{m}_u,\mathsf{m}\right)\left[\left(1-\frac{\mathsf{m}_{\text{exit}}}{p_{\mathsf{m}}}\right)\log_2\operatorname{mapsim}\left(\mathsf{A},\mathsf{m}_u,\mathsf{m}\right)+\sum_{v\in\mathsf{m}}\frac{p_v}{p_{\mathsf{m}}}\log_2\frac{p_v}{p_{\mathsf{m}}}\right]. \tag{23}$$

Again, we need only consider mapsim values between $m^2=\mathcal{O}\left(n\right)$ many pairs of modules. The last term inside the square brackets, that is, the sum over all nodes in each module, can be precomputed once per module and then be reused, requiring overall time $\mathcal{O}\left(n\right)$.

Simplifying part II requires considering the intersections between the modules from partitions A and B. However, we can avoid computing these intersections explicitly. Instead, we can tabulate the values required to compute the following expressions in a single pass over the nodes.

$$(II) = \sum_{v\in V}\operatorname{mapsim}\left(\mathsf{A},u,v\right)\log_2\operatorname{mapsim}\left(\mathsf{B},u,v\right) \tag{24}$$

$$= \sum_{\mathsf{m_a}\in\mathsf{A}}\sum_{\mathsf{m_b}\in\mathsf{B}}\sum_{v\in\mathsf{m_a}\cap\mathsf{m_b}}\operatorname{mapsim}\left(\mathsf{A},u,v\right)\log_2\operatorname{mapsim}\left(\mathsf{B},u,v\right) \tag{25}$$

Next, we pull out the last factors from the mapsim operator again

$$= \sum_{\mathsf{m_a}\in\mathsf{A}}\sum_{\mathsf{m_b}\in\mathsf{B}}\sum_{v\in\mathsf{m_a}\cap\mathsf{m_b}}\operatorname{mapsim}\left(\mathsf{A},\mathsf{m}_{\mathsf{u,A}},\mathsf{m_a}\right)\frac{p_v}{p_{\mathsf{m_a}}}\log_2\left(\operatorname{mapsim}\left(\mathsf{B},\mathsf{m}_{\mathsf{u,B}},\mathsf{m_b}\right)\frac{p_v}{p_{\mathsf{m_b}}}\right), \tag{26}$$

pull out common factors

$$= \sum_{\mathsf{m_a} \in \mathsf{A}} \operatorname{mapsim}(\mathsf{A}, \mathsf{m_{u,A}}, \mathsf{m_a}) \sum_{\mathsf{m_b} \in \mathsf{B}} \sum_{v \in \mathsf{m_a} \cap \mathsf{m_b}} \frac{p_v}{p_{\mathsf{m_a}}} \log_2 \left(\operatorname{mapsim}(\mathsf{B}, \mathsf{m_{u,B}}, \mathsf{m_b}) \frac{p_v}{p_{\mathsf{m_b}}} \right), \tag{27}$$

and apply logarithm rules

$$= \sum_{\mathsf{m_a} \in \mathsf{A}} \operatorname{mapsim}(\mathsf{A}, \mathsf{m_{u,A}}, \mathsf{m_a}) \sum_{\mathsf{m_b} \in \mathsf{B}} \Bigg[\sum_{v \in \mathsf{m_a} \cap \mathsf{m_b}} \frac{p_v}{p_{\mathsf{m_a}}} \log_2 \operatorname{mapsim}(\mathsf{B}, \mathsf{m_{u,B}}, \mathsf{m_b}) \tag{28}$$

$$+ \sum_{v \in \mathsf{m_a} \cap \mathsf{m_b}} \frac{p_v}{p_{\mathsf{m_a}}} \log_2 \frac{p_v}{p_{\mathsf{m_b}}} \Bigg] \tag{29}$$

$$= \sum_{\mathsf{m_a} \in \mathsf{A}} \operatorname{mapsim}(\mathsf{A}, \mathsf{m_{u,A}}, \mathsf{m_a}) \sum_{\mathsf{m_b} \in \mathsf{B}} \Bigg[\frac{p_{\mathsf{m_a} \cap \mathsf{m_b}}}{p_{\mathsf{m_a}}} \log_2 \operatorname{mapsim}(\mathsf{B}, \mathsf{m_{u,B}}, \mathsf{m_b}) \tag{30}$$

$$+ \sum_{v \in \mathsf{m_a} \cap \mathsf{m_b}} \frac{p_v}{p_{\mathsf{m_a}}} \log_2 \frac{p_v}{p_{\mathsf{m_b}}} \Bigg], \tag{31}$$

where $p_{\mathsf{m_a} \cap \mathsf{m_b}} = \sum_{v \in \mathsf{m_a} \cap \mathsf{m_b}} p_v$.

Altogether, flow divergence can be rewritten as

$$D_F(\mathsf{A} \,||\, \mathsf{B}) = \sum_{u \in V} \phi_u \sum_{\mathsf{m} \in \mathsf{A}} \operatorname{mapsim}(\mathsf{A}, \mathsf{m}_u, \mathsf{m}) \Bigg[\left(1 - \frac{\mathsf{m}_{\text{exit}}}{p_{\mathsf{m}}}\right) \log_2 \operatorname{mapsim}(\mathsf{A}, \mathsf{m}_u, \mathsf{m}) \tag{32}$$

$$+ \sum_{v \in \mathsf{m}} \frac{p_v}{p_{\mathsf{m}}} \log_2 \frac{p_v}{p_{\mathsf{m}}} \Bigg] \tag{33}$$

$$- \sum_{u \in V} \phi_u \sum_{\mathsf{m_a} \in \mathsf{A}} \operatorname{mapsim}(\mathsf{A}, \mathsf{m_{u,A}}, \mathsf{m_a}) \sum_{\mathsf{m_b} \in \mathsf{B}} \Bigg[\frac{p_{\mathsf{m_a} \cap \mathsf{m_b}}}{p_{\mathsf{m_a}}} \log_2 \operatorname{mapsim}(\mathsf{B}, \mathsf{m_{u,B}}, \mathsf{m_b}) \tag{34}$$

$$+ \sum_{v \in \mathsf{m_a} \cap \mathsf{m_b}} \frac{p_v}{p_{\mathsf{m_a}}} \log_2 \frac{p_v}{p_{\mathsf{m_b}}} \Bigg] \tag{35}$$

Limitations. Because flow divergence is based on the KL divergence, similar limitations apply: To compare two probability distributions, P and Q that are defined on the same sample space X, the KL divergence is only defined if, for all $x \in X$, $q_x = 0$ implies $p_x = 0$. Otherwise, $D(P \,||\, Q) = \infty$ [8]. For flow divergence, this means that we require networks to be connected: mapsim values between nodes in disconnected parts of the network become 0, resulting in transitions with probability 0, which in turn would yield an infinitely high flow divergence.

To handle disconnected networks, we could add a small constant to each mapsim value, thus ensuring that all transition probabilities are larger than 0. Alternatively, we could regularise the random walker's transition rates with a Bayesian prior that was designed for the map equation to reduce overfitting in sparse and incomplete networks [25]. However, both approaches would increase the codelength because they add further links to the network.

E Comparing Maps

Figure 4 shows the maps for the partitions in our motivational example, annotated with module-normalised node visit rates and transition rates. While traditional partition similarity measures cannot distinguish between partitions B, C and D when comparing them against A, flow divergence distinguishes D from B and C (see Table 1). Examining nodes' individual contributions to the flow divergence also explains why partition D is considered more similar to A than B and C, see Table 2.

To see how different maps for the same network correspond to assuming different random-walker dynamics with different transition rates, consider Figure 5. Comparing these maps with flow divergence, we obtain $D_F(\mathsf{M}_2 \,||\, \mathsf{M}_3) \approx 0.11$ bits and $D_F(\mathsf{M}_3 \,||\, \mathsf{M}_2) \approx 0.01$ bits. This means that assuming

(a) $L(\mathsf{A}) \approx 2.86$ bits **(b)** $L(\mathsf{B}) \approx 3.73$ bits **(c)** $L(\mathsf{C}) \approx 3.73$ bits **(d)** $L(\mathsf{D}) \approx 4.47$ bits

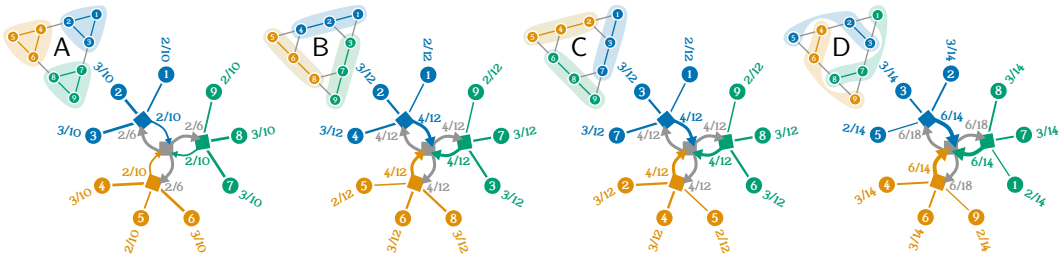

Figure 4: Comparing maps. The same partitions as shown in Figure 1, together with their maps. Flow divergence can distinguish between partitions where traditional measures fail. **(a)** The reference partition A. The partitions B in **(b)** and C in **(c)** have the same codelength and are symmetric: each module overlaps in two out of three nodes with the modules in the reference partition. **(d)** Partition D with disconnected communities but still a two-out-of-three overlap per module with the reference partition. However, the overlap between communities in A and D is in the higher-degree nodes while B's and C's communities overlap with A's communities in one higher and one lower-degree node.

Table 2: Flow divergence on a per-node basis in bits where we fix A as the reference partition.

Node	1	2	3	4	5	6	7	8	9
Flow	$\frac{2}{24}$	$\frac{3}{24}$	$\frac{3}{24}$	$\frac{3}{24}$	$\frac{2}{24}$	$\frac{3}{24}$	$\frac{3}{24}$	$\frac{3}{24}$	$\frac{2}{24}$
B	0.12	0.2	0.32	0.32	0.12	0.2	0.2	0.32	0.12
C	0.12	0.32	0.2	0.2	0.12	0.32	0.32	0.2	0.12
D	0.21	0.14	0.14	0.14	0.21	0.14	0.14	0.14	0.21

that M_2 captures the random walker's true dynamics, the expected additional cost per step for using M_3 is 0.11 bits. Conversely, assuming that M_3 describes the random walker's true dynamics, the expected additional cost per step for using M_2 is 0.01 bits.

(a) $L(\mathsf{M}_2) \approx 3.39$ bits **(b)** $L(\mathsf{M}_3) \approx 3.37$ bits

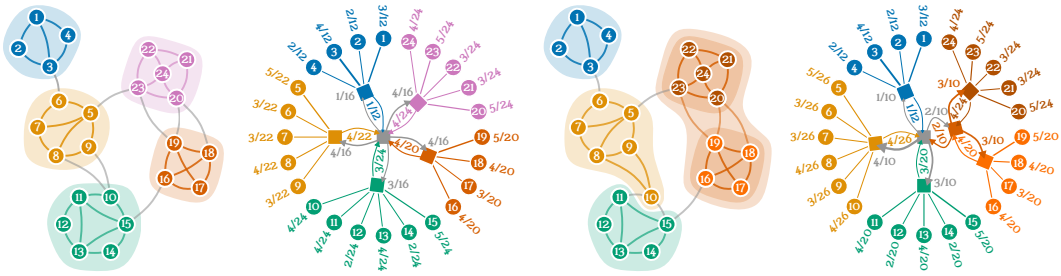

Figure 5: Different partitions for the same network. **(a)** A two-level map, M_2, of the network with five modules. **(b)** A three-level map, M_3, of the network with four modules, one of which has two submodules. Labels in the trees show the rate at which a random walker visits nodes and enters or exits modules. For example, a random walker who is in the blue module exits at rate $\frac{1}{12}$ in both maps. A random walker who is at the tree's root level enters the green module at rate $\frac{3}{24}$ in map M_2 and $\frac{3}{20}$ in map M_3, respectively.

F The Cost of Overfitting

Real-world data is often incomplete, resulting in spurious patterns in observed networks. Community-detection methods may pick up communities that exist purely due to chance because the data is sparse or incomplete, leading to finding structure even in random networks [25–27]. In the case of the map equation, less data generally reduces the codelength: fewer links make networks sparser, leading to smaller modules with lower entropy. However, the process that guides how links form does not

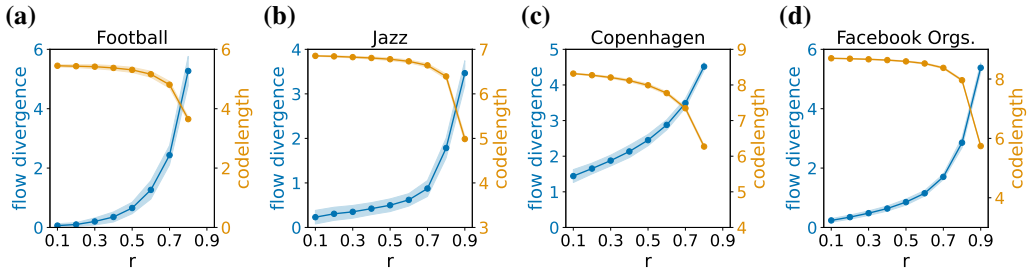

Figure 6: The cost of overfitting. We use four real-world networks, remove different r-fractions of their links, and partition the resulting networks with Infomap. We measure the detected partitions' divergence against the reference partition obtained from the "complete" network. For each r value, we repeat the sampling 100 times and report averages; the error bands show one standard deviation from the mean. As r increases, flow divergence increases faster than the codelength decreases. **(a)** A network of college football clubs that have played against each other. **(b)** A collaboration network between jazz musicians. **(c)** A social network between students. **(d)** A social network between employees of a company.

change and we have already seen in Section 4 that lower-codelength partitions do not necessarily capture the "true" dynamics best.

Table 3: Properties of four real networks that we use for measuring the cost of overfitting. We list the number of nodes, $|V|$, the number of links $|E|$, average degree $\langle k \rangle$, and the codelength L for the reference partition detected by Infomap.

Network	Ref.	$\lvert V\rvert$	$\lvert E\rvert$	$\langle k\rangle$	L
Football	[28]	115	613	10.7	5.45
Jazz	[29]	198	2,742	27.7	6.86
Copenhagen	[30]	800	6,429	16.1	8.34
Facebook Orgs.	[31]	1,429	19,357	27.1	8.71

Here, we consider the cost of overfitting due to incomplete data. We choose four real-world social networks (Table 3) and use Infomap [32, 33], the map equation's optimisation algorithm, to detect communities. We use those communities as the reference partition. Then, we remove an r-fraction of the links and use Infomap to detect communities in the reduced network. For removing links, we consider them in random order, removing one at a time until we have removed an r-fraction of the links. However, if removing a link would split the network into disconnected components, we keep the link and continue with the next one. The reduced network must contain at least $|V| - 1$ links to remain connected, placing an upper bound on the number of links we can remove. Finally, we use flow divergence to compute the expected additional cost in bits for overfitting, that is, relative to the reference partition based on "complete" data. For each r, we repeat this 100 times and report averages.

We find that, as r increases, flow divergence increases faster than the codelength decreases (Figure 6). While the description of random walks on the network can be compressed more when less data is available, flow divergence tells us about the actual cost of using what seems like a more efficient encoding. In other words: with incomplete data, Infomap finds modules that enable encoding random walks more efficiently while flow divergence tells us that we diverge more from what we assumed to be the "true" dynamics on the network.