# Flow Divergence: Comparing Hierarchical Network Partitions based on Relative Entropy

**Christopher Blöcker**
Chair of Machine Learning for Complex Networks
Center for Artificial Intelligence and Data Science
University of Würzburg, Germany
christopher.bloecker@uni-wuerzburg.de

**Ingo Scholtes**
Chair of Machine Learning for Complex Networks
Center for Artificial Intelligence and Data Science
University of Würzburg, Germany
ingo.scholtes@uni-wuerzburg.de

## Abstract

Networks model how the entities in complex systems are connected and can be partitioned into communities in different ways. Common approaches for comparing network partitions compute agreements between partitions in terms of 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 network partitions, inspired by the ideas behind the Kullback-Leibler (KL) divergence and describing random walks on networks. Like the KL divergence, flow divergence adopts a coding perspective and compares two network partitions A and B by considering the expected extra number of bits required to describe a random walk on a network using "estimate" B of the network's assumed "true" partition A. Because flow divergence is based on random walks, it can compare hierarchical and non-hierarchical partitions with arbitrary depths. Applied to synthetic and empirical networks, we show that flow divergence distinguishes partitions where traditional measures fail.

## 1 Introduction

Many real-world complex networks have communities: groups of nodes with more internal than external connections. 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, researchers have proposed a plethora of ways to characterise what constitutes a community, however, none of these characterisations is fundamentally more right or wrong than any other [1]. Naturally, based on their assumptions, different methods partition the same network into communities differently, and running stochastic community-detection methods with the same network as input several times can return different partitions. Consequently, we need measures to compare partitions and evaluate the extent to which they agree and how they differ.

Researchers across scientific fields have proposed many partition similarity measures [2–8]. Arguably, the most common measures in network science are the so-called 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 community detection in complex networks is about 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 propose *flow divergence*, a partition dissimilarity measure based on the description of random walks on networks. Combining the principles behind the Kullback-Leibler (KL) divergence [8] for measuring the dissimilarity between stochastic processes and the map equation for community detection [12], flow divergence quantifies the expected extra number of

C. Blöcker and I. Scholtes, Flow Divergence: Comparing Hierarchical Network Partitions based on Relative Entropy. *Proceedings of the Fourth Learning on Graphs Conference (LoG 2025)*, PMLR 338, Arizona State University, Phoenix, USA, December 10–12, 2025.

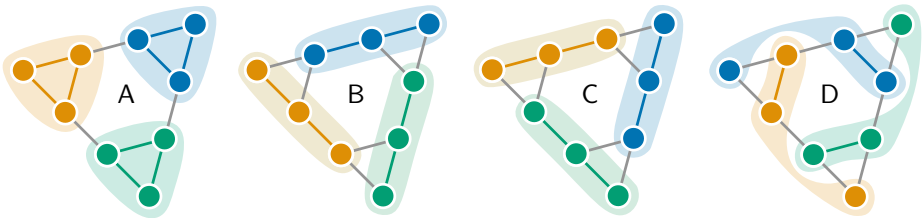


**Figure 1:** Four different partitions for the same network. Because common measures, such as the Jaccard index and mutual information, consider merely the nodes' community labels but ignore link patterns, they consider B, C, and D to match the reference partition A equally well.

bits required to describe a random walk on a network when using an "estimate" B of the network's assumed "true" community structure A. Flow divergence builds on map equation similarity [13], an information-theoretic measure that quantifies the number of bits for describing a random walker step, given a network partition into possibly hierarchical communities. Because flow divergence is based on describing random walks, it naturally takes link patterns into account and can compare hierarchical and non-hierarchical partitions. Via the Jensen-Shannon distance, we also define a symmetric and normalised version of flow divergence, which we call *flow distance*. Applied to synthetic and real networks, we verify that our measures distinguish between partitions where traditional measures fail.

## 2 Related Work

Comparing partitions is a recurring problem across scientific domains that has received much attention [14], and researchers have proposed a plethora of measures for comparing partitions [2–8]. Given two partitions A and B of the same set $X$ with $n = |X|$ objects into non-overlapping sets with $\bigcup_{\mathsf{a}\in\mathsf{A}} \mathsf{a} = X = \bigcup_{\mathsf{b}\in\mathsf{B}} \mathsf{b}$, $\forall \mathsf{a}, \mathsf{a}' \in \mathsf{A} : \mathsf{a} \neq \mathsf{a}' \rightarrow \mathsf{a} \cap \mathsf{a}' = \emptyset$, 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 the partitions A and B are.

Measures for comparing partitions can be categorised into pair-counting, set-matching, and information-theoretic measures [15, 16]. A popular pair-counting measure is the Rand index [17], which considers all possible pairs of objects $x, x' \in X$ and counts $t$ as the number of pairs for which A and B agree whether $x$ and $x'$ belong to the same group. The similarity $\mathrm{R}$ between A and B is the number of agreements $t$, divided by the total number of pairs, $\mathrm{R}(\mathsf{A}, \mathsf{B}) = t/\binom{n}{2}$. The adjusted Rand index [18] uses a null model to correct for assignments that agree due to chance. Two of the arguably most commonly used 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, 19]. The Jaccard index computes the agreement between groups $\mathsf{a} \in \mathsf{A}$ and $b \in \mathsf{B}$ as $\mathrm{J}(\mathsf{a}, \mathsf{b}) = \frac{|\mathsf{a} \cap \mathsf{b}|}{|\mathsf{a} \cup \mathsf{b}|}$. 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: $\mathrm{J}(\mathsf{A}, \mathsf{B}) = \sum_{\mathsf{a}\in\mathsf{A}} \frac{|\mathsf{a}|}{n} \max_{\mathsf{b}\in\mathsf{B}} \mathrm{J}(\mathsf{a}, \mathsf{b})$. Mutual information considers how much information the objects' assignments under A provide about their assignment under B: $\mathrm{MI}(\mathsf{A}, \mathsf{B}) = \sum_{\mathsf{a}\in\mathsf{A}} \sum_{\mathsf{b}\in\mathsf{B}} P(\mathsf{a}, \mathsf{b}) \log_2 \frac{P(\mathsf{a},\mathsf{b})}{P(\mathsf{a})P(\mathsf{b})}$, where $P(\mathsf{a}) = \sum_{x\in\mathsf{a}} p_x$ and $P(\mathsf{b}) = \sum_{x\in\mathsf{b}} p_x$ are the probabilities for selecting a and b, respectively, when choosing an object $x \in X$ according to its probability $p_x$; $P(\mathsf{a}, \mathsf{b}) = \sum_{x\in\mathsf{a}\cap\mathsf{b}} p_x$ is the joint probability of a and b. Normalised mutual information scales the mutual information score to the interval $[0, 1]$, adjusted mutual information (AMI) additionally adjusts it for chance [7, 16]. Reduced mutual information corrects for cases where unrelated clusterings produce greater-than-zero similarity [19].

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

Despite ignoring links, measures like the Jaccard index and mutual information are often used to compare network partitions whose groups summarise link patterns according to some community-detection objective function such as modularity [20], a stochastic block model [21], or the map equation [12]. But community detection is about identifying groups of nodes that summarise link patterns—links are essential for deciding what partition describes the structure of a network well. A possible way to incorporate links into traditional measures could be by considering link sets alongside node sets. However, while such an approach would consider nodes and links, it would ignore the relationships between them. Straulino et al. [3] recognised the need to incorporate link patterns and proposed a framework to compare network partitions based on any community-detection objective function $f$ that takes a graph $G$ and a partition of its nodes, and returns, possibly after scaling, a value in $[0, 1]$. They treat $f$ as a black box and define the non-symmetric distance between partitions $\mathsf{A}$ and $\mathsf{B}$ as $d(\mathsf{A}, \mathsf{B}) = 1 - \frac{f(G,\mathsf{A})}{f(G,\mathsf{B})}$, thus incorporating link patterns as captured by $f$. However, their approach cannot distinguish between partitions for which $f$ returns the same value.

## 3 Flow Divergence

To define our partition dissimilarity score, *flow divergence*, we combine the map equation for flow-based community detection with the Kullback-Leibler (KL) divergence [8], also known as relative entropy, which is 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 defined on the same sample space $X$, where $p_x$ and $q_x$, respectively, are the probabilities for drawing $x \in X$. 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, our goal is to define a partition dissimilarity measure that quantifies the expected additional number of bits required to describe a random walk on a network using an "estimate" $\mathsf{B}$ of its "true" structure $\mathsf{A}$.

### 3.1 Random-Walk Description Length

Let $G = (V, E)$ be a connected graph with nodes $V$, links $E$, and let $w_{uv} \in \mathbb{R}^+$ be the weight on the link from node $u$ to $v$. Further, let $P = \{p_v \mid v \in V\}$ be the set of ergodic node visit rates, which we also refer to as *flow*. According to Shannon's source coding theorem [22], describing the random walker's position on the graph requires

$$H(P) = -\sum_{v \in V} p_v \log_2 p_v \text{ bits} \tag{1}$$

per step, where $H$ is the Shannon entropy and $\log_2 p_v$ is the length of node $v$'s codeword in bits. We can design concrete codewords with a Huffman code [23] as shown in Figure 2a. Note, however, that we use concrete random walks and codewords only to illustrate how flow divergence works. In practice, we neither simulate random walks nor assign codewords. As is common in information theory, we are only interested in the codewords' theoretical lengths to calculate flow divergence. In undirected and strongly connected directed graphs, we calculate the nodes' visit rates using the power iteration method to solve the recursive set of equations,

$$p_v = \sum_{u \in V} p_u t_{uv}, \tag{2}$$

where $t_{uv} = {}^{w_{uv}}/_{\sum_{v \in V} w_{uv}}$ is the probability that a random walker at node $u$ steps to node $v$. In weakly connected graphs, we apply a power iteration to the process defined by PageRank [24] or so-called smart teleportation [25]. By combining the above equations and reordering, we explicitly relate describing the random walker's position to transitions along links,

$$H(P) = -\sum_{u \in V} p_u \sum_{v \in V} t_{uv} \log_2 p_v. \tag{3}$$

This formulation adopts a global perspective with globally unique codewords for nodes but does not account for the network's community structure. To make the coding depend on communities, also called *modules*, we introduce a parameter $\mathsf{M}$: the partition of the network's nodes into modules. We denote the module-dependent transition rate 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 $v$, and obtain the description length

$$L(\mathsf{M}) = -\sum_{u \in V} p_u \sum_{v \in V} t_{uv} \log_2 s(\mathsf{M}, u, v). \tag{4}$$

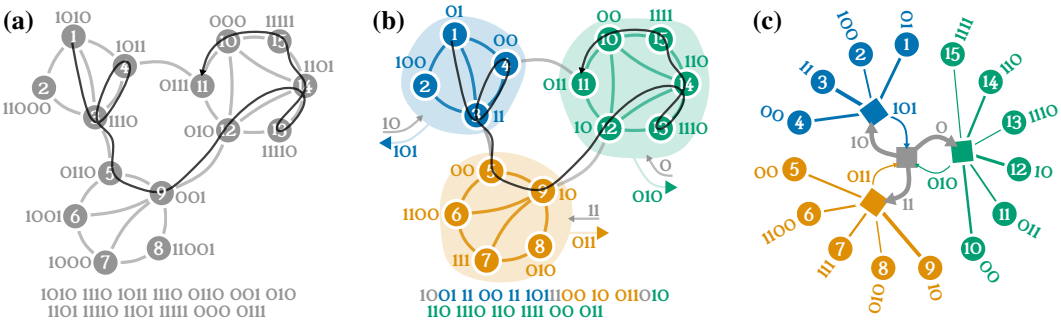


**Figure 2:** Illustration of the map equation's coding principles. **(a)** Without communities, we derive unique codewords from the nodes' visit rates and use them to describe the shown random walker steps with the codewords at the bottom. **(b)** With communities, we assign codewords that are unique within each community. Codewords for entering and exiting communities are shown next to arrows that point into and out of the communities. **(c)** The coding scheme from **(b)**, drawn as a radial tree. Link widths are proportional to module-normalised codeword usage rates.

Setting $s(\mathsf{M}, u, v) = p_v$ maintains a global perspective and globally unique codewords, while setting $s(\mathsf{M}, u, v) = t_{uv}$ adopts a node-local coding perspective where codewords depend on source and target node. Adopting an intermediate modular perspective means considering $u$'s and $v$'s modular context when computing coding costs and assigning codewords that are unique within modules.

### 3.2 The Map Equation for Modular Coding

The map equation [12, 26] is an information-theoretic objective function for flow-based community detection that provides a way to define a modular coding scheme. It identifies communities by minimising the modular description length for random walks on networks: Without communities, that is, when all nodes are assigned to the same module, the required number of bits per random walker step is the 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,

$$L(\mathsf{M}) = qH(Q) + \sum_{\mathsf{m}\in\mathsf{M}} p_{\mathsf{m}} H(P_{\mathsf{m}}). \tag{5}$$

Here, $q = \sum_{\mathsf{m}\in\mathsf{M}} q_{\mathsf{m}}$ is the index-level codebook usage rate, $q_{\mathsf{m}} = \sum_{u\notin\mathsf{m}} \sum_{v\in\mathsf{m}} p_u t_{uv}$ is the entry rate for module $\mathsf{m}$, $Q = \{q_{\mathsf{m}}/q \mid \mathsf{m}\in\mathsf{M}\}$ is the set of normalised module entry rates, $p_{\mathsf{m}} = \mathsf{m}_{\text{exit}} + \sum_{u\in\mathsf{m}} p_u$ is module $\mathsf{m}$'s codebook usage rate, $\mathsf{m}_{\text{exit}} = \sum_{u\in\mathsf{m}} \sum_{v\notin\mathsf{m}} p_u t_{uv}$ is module $\mathsf{m}$'s exit rate, and $P_{\mathsf{m}} = \{\mathsf{m}_{\text{exit}}/p_{\mathsf{m}}\} \cup \{p_u/p_{\mathsf{m}} \mid u\in\mathsf{m}\}$ is the set of normalised node visit rates in module $\mathsf{m}$, including its exit rate. Figure 2b shows an example of a two-level coding scheme where codewords are reused across modules for a shorter overall codelength. Through recursion, the map equation generalises to hierarchical partitions where modules can contain further submodules [26, 27].

Minimising the map equation creates a "map" of the network's organisational structure where nodes are grouped into modules such that a random walker tends to stay within modules, whereas switching modules occurs rarely. We can draw such 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 these two nodes; to describe the step, we use the codewords along the shortest path in the map.

To apply the ideas behind the KL divergence, we rewrite the map equation such that it more closely resembles a random walk, matching the form of Equation (4) (see Appendix A for derivation),

$$L(\mathsf{M}) = qH(Q) + \sum_{\mathsf{m}\in\mathsf{M}} p_{\mathsf{m}} H(P_{\mathsf{m}}) = -\sum_{u\in V} p_u \sum_{v\in V} t_{uv} \log_2 \operatorname{mapsim}(\mathsf{M}, u, v), \tag{6}$$

where $\log_2 \operatorname{mapsim}(\mathsf{M}, u, v)$ is the number of bits required to describe a random-walker step from node $u$ to $v$, given partition $\mathsf{M}$ [13]. mapsim, which is short for map equation similarity, is defined as

$$\operatorname{mapsim}(\mathsf{M}, u, v) = (1 - \delta_{\mathsf{m}_u,\mathsf{m}_v}) \left( \frac{q_{\mathsf{m}_u}}{p_{\mathsf{m}_u}} \cdot \frac{q_{\mathsf{m}_v}}{q} \cdot \frac{p_v}{p_{\mathsf{m}_v}} \right) + \delta_{\mathsf{m}_u,\mathsf{m}_v} \frac{p_v}{p_{\mathsf{m}_v}}, \tag{7}$$

where $\mathsf{m}_u$ and $\mathsf{m}_v$ are the modules to which nodes $u$ and $v$ belong, respectively, and $\delta$ is the Kronecker delta. Given a network partition, $\mathrm{mapsim}$ quantifies the rate at which a random walker transitions between pairs of nodes. Importantly, $\mathrm{mapsim}$ depends on the source node's module, but not on the specific source node itself; through recursion, $\mathrm{mapsim}$ generalises to hierarchical partitions [13, 28].

### 3.3 Relative Entropy Between Partitions

Partition can be seen as a summary of the random walker's movement patterns, representing them as ensembles of possibly nested random processes. Following the idea of the KL divergence, we assume that partition $\mathsf{A}$ captures the random walker's true movement patterns. And we ask: what is the expected extra number of bits required to describe a random walk using an "estimate" $\mathsf{B}$ of the true pattern $\mathsf{A}$? Using the partition-dependent transition rates $t^{\mathsf{A}}_{uv}$ (see Appendix B for how we derive them from partitions), we define our partition dissimilarity measure *flow divergence* as

$$D_F\left(\mathsf{A}\,||\,\mathsf{B}\right) = \sum_{u\in V} p_u \sum_{v\in V} t^{\mathsf{A}}_{uv} \log_2 \frac{\mathrm{mapsim}\left(\mathsf{A},u,v\right)}{\mathrm{mapsim}\left(\mathsf{B},u,v\right)}, \text{ where } t^{\mathsf{A}}_{uv} = \frac{\mathrm{mapsim}\left(\mathsf{A},u,v\right)}{\sum_v \mathrm{mapsim}\left(\mathsf{A},u,v\right)}. \quad (8)$$

For an example where we compare different partitions for the same network, one of which is hierarchical, see Appendix C. However, there is one issue. The $\mathrm{mapsim}$ values entering the logarithm do not form a probability distribution as they do not sum to 1, and, therefore, Equation (8) is not a true KL divergence. The reason is that the coding scheme induced by the map equation contains redundancies; in principle, a random walker could switch back and forth between modules, whereas describing the random walker's new location is always based on the shortest path in the coding tree. This leaves us with two options. We can either use flow divergence as in Equation (8) with the interpretation that it captures the expected extra number of bits for describing random walks when using an "estimate" of the network's assumed "true" structure. Or we can normalise the quantities entering the logarithms to ensure that they sum to 1, sacrificing flow divergence's clean interpretation to some extent. However, as we will see in our evaluation, the difference between these options is typically small and often negligible in practice.

### 3.4 Normalising Flow Divergence

Using the partition-dependent transition probabilities in Equation (24), we define a normalised version of flow divergence that is an expected KL divergence,

$$\tilde{D}_F\left(\mathsf{A}\,||\,\mathsf{B}\right) = \sum_{u\in V} p_u \sum_{v\in V} t^{\mathsf{A}}_{uv} \log_2 \frac{t^{\mathsf{A}}_{uv}}{t^{\mathsf{B}}_{uv}} = \sum_{u\in V} p_u \sum_{v\in V} t^{\mathsf{A}}_{uv} \left[\log_2 \frac{\mathrm{mapsim}\left(\mathsf{A},u,v\right)}{\mathrm{mapsim}\left(\mathsf{B},u,v\right)} - \lambda^{\mathsf{A}}_u + \lambda^{\mathsf{B}}_u\right], \quad (9)$$

where $\lambda^{\mathsf{A}}_u$ and $\lambda^{\mathsf{B}}_u$ are node-dependent normalisation terms with $\lambda^{\mathsf{A}}_u = \log_2\left(\sum_{v\in V} \mathrm{mapsim}\left(\mathsf{A},u,v\right)\right)$ and $\lambda^{\mathsf{B}}_u = \log_2\left(\sum_{v\in V} \mathrm{mapsim}\left(\mathsf{B},u,v\right)\right)$.

### 3.5 Flow Distance: Symmetric and Normalised

Like the KL divergence, the normalised version of flow divergence is not bounded and can return arbitrarily large values. The exact behaviour depends on the network's size, topology, and compared community structures, but larger networks generally lead to higher flow divergence values. This can make it difficult for practitioners to compare partition similarities between different networks, especially when they have different sizes. Moreover, flow divergence is an asymmetric measure, and sometimes it is more convenient to work with symmetric measures. Flow divergence can be turned into a symmetric and normalised measure with values in the interval $[0,1]$ via the Jensen-Shannon (JS) distance. The JS distance takes two probability distributions $P, Q$ defined on the same sample space, and computes their "distance" as the square root of the average KL divergence between $P$ and $Q$ and their mixture $M$ [29]. It is defined as $d_{JS}\left(P,Q\right) = \sqrt{\frac{1}{2}D_{KL}\left(P\,||\,M\right) + \frac{1}{2}D_{KL}\left(Q\,||\,M\right)}$, where $M = \frac{1}{2}\left(P+Q\right)$. When using the logarithm with base 2, the JS divergence returns values in the interval $[0,1]$. We apply the same idea to flow divergence and define *flow distance* as

$$d_F\left(\mathsf{A},\mathsf{B}\right) = \sqrt{\frac{1}{2}\tilde{D}_F\left(\mathsf{A}\,||\,\mathsf{M}_{\mathsf{AB}}\right) + \frac{1}{2}\tilde{D}_F\left(\mathsf{B}\,||\,\mathsf{M}_{\mathsf{AB}}\right)}. \quad (10)$$

Here, $\mathsf{M}_{\mathsf{AB}} = \frac{1}{2}\left(\mathsf{A}+\mathsf{B}\right)$ is the "mixture partition" of $\mathsf{A}$ and $\mathsf{B}$, defined implicitly by $\mathrm{mapsim}$ values

$$\mathrm{mapsim}\left(\mathsf{M}_{\mathsf{AB}},u,v\right) = \frac{1}{2}\left(\mathrm{mapsim}\left(\mathsf{A},u,v\right) + \mathrm{mapsim}\left(\mathsf{B},u,v\right)\right). \quad (11)$$

### 3.6 Computational Complexity and Limitations

Computing flow divergence between two partitions $\mathsf{A}, \mathsf{B}$ involves $\mathrm{mapsim}$ values between $n^2$ many node pairs, where $n = |V|$ is the number of nodes. By exploiting the partitions' modular structure, we can compute flow divergence in time $\mathcal{O}(d_m \cdot m^2)$, where $m$ is the number of non-empty intersections between the modules $\mathsf{a} \in \mathsf{A}$ and $\mathsf{b} \in \mathsf{B}$, and $d_m$ is the average depth at which the modules are nested. Because $\mathrm{mapsim}$ depends on the source module but not the specific source node, and because we can aggregate the target nodes per module, we only need to consider $m$ source modules and $m$ target modules, that is, $m^2$ pairs of modules instead of $n^2$ pairs of nodes. With time $\mathcal{O}(d_m)$ to compute the $\mathrm{mapsim}$ value between two modules, we have an overall complexity of $\mathcal{O}(d_m \cdot m^2)$, where $d_m$ is typically small in empirical networks [27]. In the worst case, when $m = n$, we have quadratic complexity, however, the number of modules in real networks is typically much smaller than the number of nodes, and theoretical and empirical evidence suggest that it scales as $m = \mathcal{O}(\sqrt{n})$ [30]. In Appendix D, we show how we regroup the terms of flow divergence for an efficient implementation.

Computing flow distance is more expensive and requires time $\mathcal{O}\left(d_m \cdot m \cdot n\right)$. This is because we defined the mixture partition $\mathsf{M}_{\mathsf{AB}}$ between partitions $\mathsf{A}, \mathsf{B}$ (Equation (11)) without concretising its structure. Therefore, we cannot use the same trick we used for flow divergence. However, regrouping the terms of flow distance allows for some optimisation, which we show in Appendix E.

Since flow divergence and flow distance are based on the KL divergence, similar limitations apply: For probability distributions $P, Q$ defined on the same sample space $X$, the KL divergence is only defined if $\forall x \in X$, $q_x = 0 \rightarrow p_x = 0$. Otherwise, $D\left(P \,||\, Q\right) = \infty$ [8]. This means we require undirected networks to be connected and directed networks to be strongly connected because $\mathrm{mapsim}$ values between nodes in disconnected parts of the network are $0$, resulting in transitions with probability $0$, yielding an infinitely high flow divergence. To handle disconnected networks, we could add a small constant to each $\mathrm{mapsim}$ 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 designed for the map equation to reduce overfitting in sparse and incomplete networks [31]. However, both approaches change the random walker's dynamics and increase the codelength because they essentially add further links to the network. A possible ad-hoc solution for computing flow divergence on disconnected and weakly-connected networks would be to ignore disconnected node pairs.

## 4 Evaluation

We apply flow divergence and flow distance to compute partition dissimilarity scores for partitions in toy, synthetic, and real networks. In a toy network, we demonstrate that flow divergence, due to its awareness of link patterns, distinguishes partitions where popular measures fail. With synthetic networks, where the ground-truth community structure is known, and real networks, where we use detected communities as a drop-in for the unknown ground truth, we study how flow divergence and flow distance behave in practical settings. In our plots, we compare flow distance to one minus the Jaccard index and AMI because this allows us to interpret them all as distances instead of similarities.

To simulate missing data, we remove different $r$-fractions of the networks' links, and use Infomap [32, 33] to detect communities in the reduced networks. For the map equation and Infomap, less data generally reduces the codelength: fewer links make networks sparser, leading to smaller modules with lower entropy [31]. As we increase $r$, we expect the detected communities to become less similar to the reference partition. To ensure that the reduced networks remain connected, we first construct a random spanning tree, then we add a random subset of the original network's links such that the reduced network has by an $r$-fraction fewer links than the original. For robust results, we repeat each experiment 10 times with different seeds and show averages and standard deviations in our plots.

For completeness, we compare the normalised and non-normalised versions of flow divergence in Appendix F; we found that their difference is generally small, and often negligible. In Appendix G, we include plots showing synthetic and real networks with their partitions and flow distance values for different fractions of removed links for a more intuitive illustration of how our method behaves.

Our Python implementation of flow divergence and flow distance[1], and the Jupyter notebooks to reproduce our results[2] are available on GitHub.

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

[2] https://github.com/chrisbloecker/flow-divergence-reproducibility

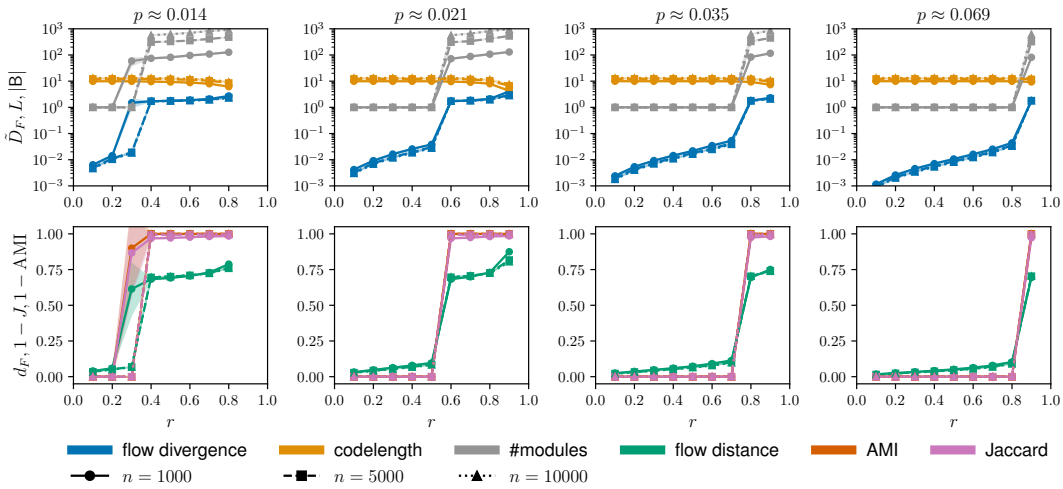


**Figure 3:** Results on ER random graphs with link probability $p \in \left\{\frac{2\ln n}{n}, \frac{3\ln n}{n}, \frac{5\ln n}{n}, \frac{10\ln n}{n}\right\}$. The top panel shows normalised flow divergence $\tilde{D}_F$ and codelength $L$ in bits, and the number of communities $|\mathsf{B}|$ detected by Infomap for different $r$-fractions of removed links on a logarithmic scale. The bottom panel shows the corresponding flow distance $d_F$, Jaccard distance $1 - \mathrm{J}$, and AMI distance $1 - \mathrm{AMI}$.

### 4.1 Toy Example

Returning to our initial example (Figure 1), we show that flow divergence can distinguish between partitions where traditional measures fail. Table 1 shows the normalised flow divergence scores according to Equation (9). Despite its higher codelength, D is more similar to A than B and C are. This is analogous to the fact that, for three probability distributions $P, Q, R$ defined on the same sample space, $R$ can be more similar to $P$ than $Q$ despite having higher entropy, that is, $H(Q) < H(R) \not\Rightarrow D_{KL}(P \,||\, Q) < D_{KL}(P \,||\, R)$. To explain why D is more similar to A, we consider the individual nodes' contribution to the divergences (see Appendix H), and find that A and D overlap in the higher-degree nodes, which have more flow and, therefore, carry higher importance.

**Table 1:** Rounded normalised flow divergence scores $\tilde{D}_F$ and codelength $L$ in bits for the partitions from Figure 1.

| | | Other | | | |
|---|---|---|---|---|---|
| Reference | $L$ | A | B | C | D |
| A | 2.86 | 0 | 1.25 | 1.25 | 0.92 |
| B | 3.73 | 1.36 | 0 | 0.86 | 1.35 |
| C | 3.73 | 1.36 | 0.86 | 0 | 0.96 |
| D | 4.47 | 1.04 | 1.39 | 1.00 | 0 |

### 4.2 Synthetic Networks

First, we use `networkx` to generate Erdős-Rényi (ER) random graphs with $n \in \{1000, 5000, 10000\}$ nodes. In ER graphs, links exist independently with probability $p$, and ER graphs are almost surely connected at $p = \frac{\ln n}{n}$. To ensure connected ER graphs with ample links for removal, we use $p \in \left\{\frac{2\ln n}{n}, \frac{3\ln n}{n}, \frac{5\ln n}{n}, \frac{10\ln n}{n}\right\}$. Because ER graphs do not have community structure [34, 35], we use the reference partition A that assigns all nodes to the same module. Our results in Figure 3 show that flow divergence and flow distance capture differences that the Jaccard index and AMI miss. For small $r$, Infomap does not detect communities and assigns all nodes to the same module. Consequently, the Jaccard index and AMI consider the detected partitions to align perfectly with the reference partition. However, removing links changes the random walker's node visit and transition rates, affecting how random walks would be encoded. Because flow divergence is designed to account for precisely these differences in link patterns via random walk dynamics, it can distinguish between such partitions. When we remove more links, and Infomap begins to detect communities, all the tested measures make a sharp jump. The exact point where this jump occurs depends on the network's density, controlled by $p$. Beyond this point, AMI returns a distance of one because the one-module partition does not provide any information about a partition with communities. The Jaccard distance becomes larger, but remains smaller than one. Flow divergence and flow distance make a similar jump, but increase more gradually after that, indicating a clearer distinction between the partitions.

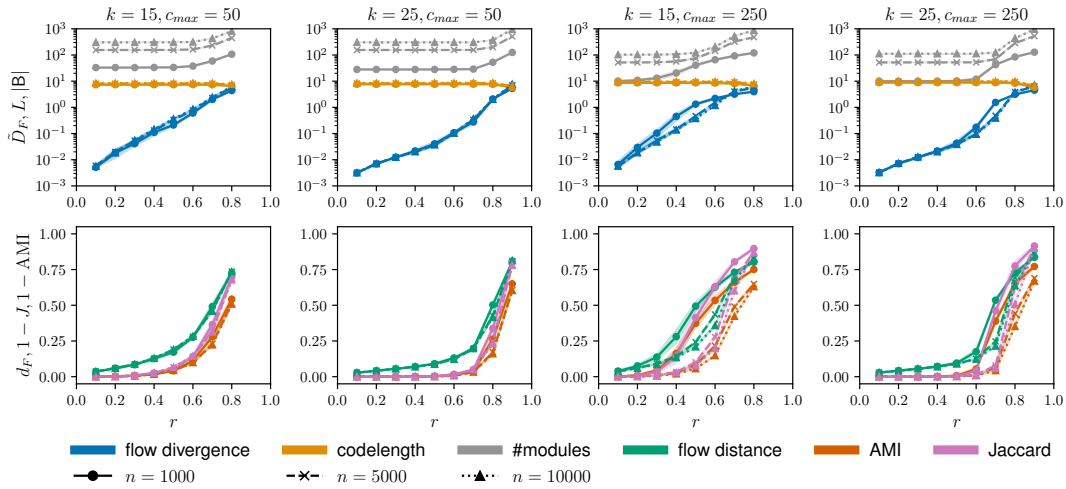


**Figure 4:** Results on undirected LFR graphs with non-hierarchical planted communities. The top panel shows normalised flow divergence $\tilde{D}_F$ and codelength $L$ in bits, and the number of communities $|\mathsf{B}|$ detected by Infomap for different $r$-fractions of removed links on a log scale. The bottom panel shows the corresponding flow distance $d_F$, Jaccard distance $1 - \mathrm{J}$, and AMI distance $1 - \mathrm{AMI}$.

Second, we generate Lancichinetti–Fortunato–Radicchi (LFR) networks with non-hierarchical planted communities [36] and $n \in \{1000, 5000, 10000\}$ nodes, using the original authors' implementation[3]. We use maximum node degree $k_{\text{max}} = 50$, minimum community size $c_{\text{min}} = 20$, mixing $\mu = 0.3$, power-law exponents $\tau_1 = 2$ and $\tau_2 = 1$ for the degree sequence and community size distribution, respectively, average node degree $k \in \{15, 25\}$, and maximum community size $c_{\text{max}} \in \{50, 250\}$. The results in Figure 4 show that, for small $r$, Infomap detects the ground-truth communities, leading to Jaccard and AMI scores of 1, and therefore a Jaccard and AMI "distance" of 0. However, because removing links affects the random walker's dynamics, flow divergence and flow distance detect these changes. While keeping the generated communities' properties fixed, the network size has only a small effect on how flow divergence and flow distance behave, suggesting that the networks' structural patterns play a more important role than their size. In Appendix I, we repeat these experiments for directed and hierarchical LFR graphs, and in Appendix K, we show the recorded wall-clock runtimes.

### 4.3 Real Networks

Real-world networks are often sparse or incomplete, which can lead to detecing spurious communities, similar to finding structure in random networks [31, 42, 43]. We use eight undirected real-world networks (Table 2) and detect hierarchical and non-hierarchical communities with Infomap, regarding those communities as the hierarchical and non-hierarchical reference partitions. Again, we remove different $r$-fractions of the links to simulate missing data and use Infomap to detect communities in the reduced networks. The reduced network must contain at least $|V| - 1$ links to remain connected, placing an upper bound on how many links we can remove.

**Table 2:** Properties of eight real networks. $\langle k \rangle$ is the average degree, $L$ and $L_h$ are the non-hierarchical and hierarchical codelengths, respectively, of the reference partitions Infomap detected.

| Network | Ref. | $\lvert V\rvert$ | $\lvert E\rvert$ | $\langle k \rangle$ | $L$ | $L_h$ |
|---|---|---|---|---|---|---|
| Football | [37] | 115 | 613 | 10.7 | 5.45 | – |
| Jazz | [38] | 198 | 2,742 | 27.7 | 6.86 | – |
| Copenhagen | [39] | 800 | 6,429 | 16.1 | 8.34 | – |
| FB Orgs. M1 | [40] | 1,429 | 19,357 | 27.1 | 8.74 | 8.71 |
| FB Orgs. L2 | [40] | 5,524 | 94,219 | 34.1 | 9.99 | 9.82 |
| Wiki Squirrel | [41] | 5,201 | 198,493 | 76.3 | 9.94 | 9.93 |
| Wiki Crocodile | [41] | 11,631 | 170,918 | 29.4 | 9.26 | 9.25 |
| GitHub Devs | [41] | 37,700 | 289,003 | 15.3 | 12.38 | – |

Figure 5 is in line with the observation from the synthetic networks. Fewer links lead to lower codelength, more communities, and larger distance from the reference partition for all measures.

[3] https://sites.google.com/site/andrealancichinetti/benchmarks

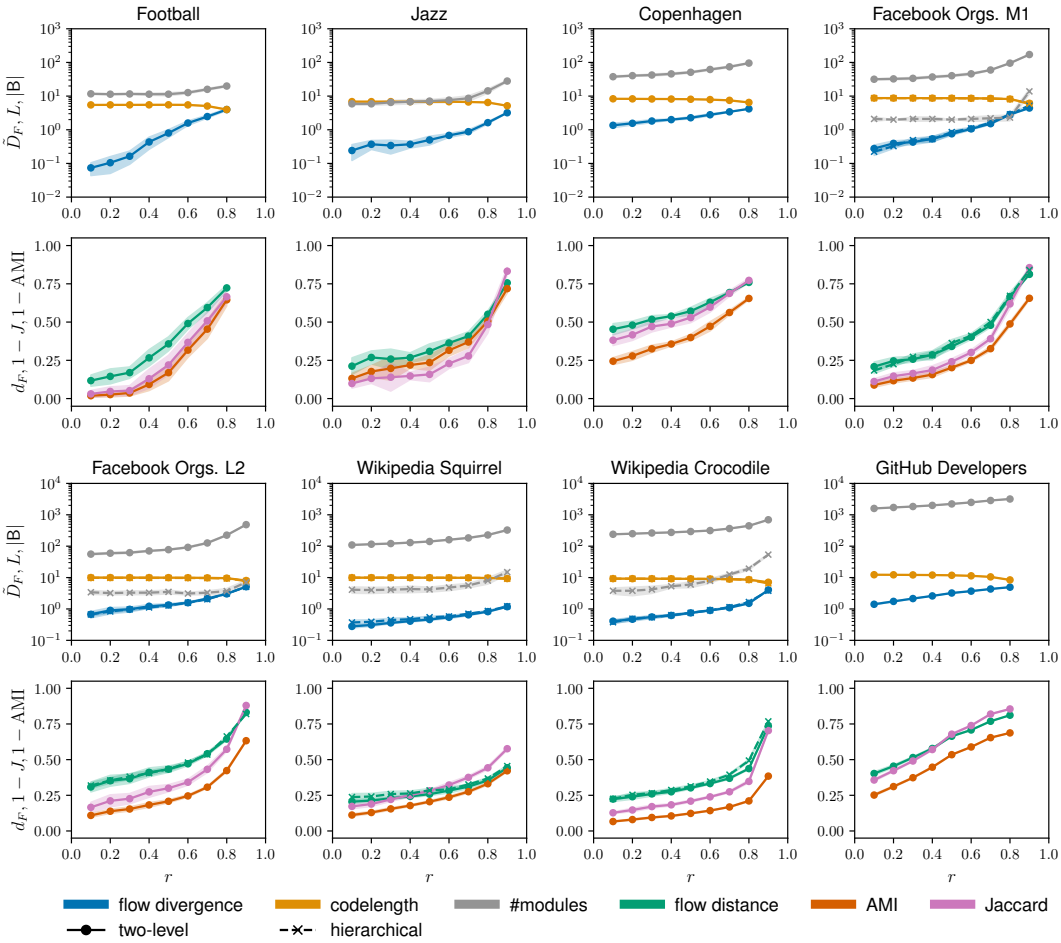


**Figure 5:** Results on real networks. We show normalised flow divergence $\tilde{D}_F$ and codelength $L$ in bits, and the number of communities $|\mathsf{B}|$ detected by Infomap for different $r$-fractions of removed links on a log scale, and the corresponding flow distance $d_F$, Jaccard distance $1-\mathrm{J}$, and AMI distance $1-\mathrm{AMI}$.

Generally, all measures follow a similar pattern, however, in some cases, the Jaccard index makes sharper jumps than AMI and flow distance. While the description of random walks on the network can be compressed more when fewer links are present, flow divergence quantifies by how much Infomap diverges from the network's "true" partition. In practice, we would naturally choose a partition with a lower codelength because it appears to capture the network's structure better. But with missing data, this would be a mistake, and flow divergence quantifies the exact cost of making this mistake. Specifically, if $\mathsf{A}$ is the network's true partition but we select $\mathsf{B}$ instead because $L(\mathsf{B}) < L(\mathsf{A})$, we expect to pay $L(\mathsf{B})$ per random walker step, while the true cost is $L(\mathsf{B}) + D_F(\mathsf{A} \,||\, \mathsf{B})$. For additional plots showing unnormalised flow divergence values, see Appendix F.

### 4.4 Discussion about Comparing Partition Dissimilarity Scores

Because scores are comparisons, a lower value in one score than another does not necessarily mean anything. However, in our case, a lower score can tell us something. Consider the following points:

- The Jaccard index and AMI return a similarity of 1, or a "distance" of 0, when the partitions align perfectly in terms of node assignments. But links do not matter. We could randomly rewire or remove as many links as we wish without affecting Jaccard or AMI scores.
- Flow distance returns a distance of 0 when the coding schemes induced by the partitions are the same, that is, when the node visit rates and transition rates are the same. We can roughly

think about this as: the partitions need to align perfectly in terms of node assignments and links. Links matter, and random rewiring or removing links affects the flow distance score.

- When does a partition describe the network's structure well? Assuming assortative communities, good partitions define communities such that most links are within communities and few links between communities. Links matter, and random rewiring or removing links generally affects the network's community structure.

In our experiments with LFR networks, we used a mixing factor of $\mu = 0.3$, meaning that about 70% of the links are within communities. Therefore, removing links uniformly at random, but while keeping the graphs connected, we mostly delete links within communities and make the community structure less pronounced. For empirical networks, we also expect that they have community structure and more links within than between communities. Hence, when removing links from empirical networks uniformly at random, we also expect to make the community structure less pronounced.

In Figures 3 and 4, we saw that the Jaccard index and AMI return distances of 0 up to a certain fraction of removed links, meaning that the partitions did not change in terms of node assignments to communities. Flow distance, on the other hand, returns distances larger than 0 as soon as we remove links because it is sensitive to the changes in the network's transition matrix. In this situation, a flow distance score larger than 0, while the Jaccard index and AMI return a distance of 0, tells us that flow distance can distinguish between partitions where the other measures fail. This is possible because flow divergence considers links, and removing links changes how well the communities characterise the network's structure. In other words: even if the assignments of nodes to communities are the same, having fewer links within the communities means that those communities are less pronounced and, consequently, not as good a description of the network's structure as compared to the case when there are more links within the communities. The results for real networks in Figure 5 were less distinct, which is likely because the community structure in empirical networks tends to be less pronounced, or redundant, than in synthetic networks.

## 5 Conclusion

We have studied the problem of comparing network partitions and highlighted the need for approaches that consider link patterns. While community detection focuses on grouping nodes with similar link patterns, common measures for comparing partitions ignore links. Inspired by the Kullback-Leibler divergence for measuring dissimilarities between probability distributions, we developed a partition dissimilarity measure based on random walks: *flow divergence*. Flow divergence quantifies the expected additional number of bits per step for describing a random walk on a network when using an "estimate" $\mathsf{B}$ of the network's "true" partition $\mathsf{A}$. Via the Jensen-Shannon distance, we also derived a symmetric and normalised version of flow divergence, which we call *flow distance*. Applied to real and synthetic networks, we showed that incorporating link patterns allows flow divergence to distinguish between partitions where popular partition similarity measures fail.

We did not consider integrating node features, which play an important role in graph representation learning, directly into flow divergence. This allowed us to keep flow divergence conceptually clean and more widely applicable. However, in case node features are present, they can be incorporated at the community-detection stage, for example, by following the apporach by Bassolas et al. [44].

Our future research directions include developing an efficient way to compute flow distance and generalising flow divergence for soft communities, where nodes can be partially assigned to several modules. While flow divergence can be computed efficiently by exploiting the network partitions' modular regularities, the same approach cannot be used for flow distance because we defined the mixture $\mathsf{M}_{\mathsf{AB}}$ between partitions $\mathsf{A}$ and $\mathsf{B}$ implicitly. An explicit definition of mixture partitions would address this issue, allowing us to also compute flow distance more efficiently. Moreover, such an explicit notion of mixture partitions would imply a way to interpolate between network partitions.

## Acknowledgements

We thank Martin Rosvall and Daniel Edler for helpful discussions. Christopher Blöcker and Ingo Scholtes acknowledge funding from the Swiss National Science Foundation, grant 176938, and the German Federal Ministry of Education and Research, grant 00582863 (TissueNet).

## 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.

[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.

[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.

[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.

[6] Marina Meilă. Comparing clusterings—an 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.

[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.

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

[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.

[10] Alcides Viamontes Esquivel and Martin Rosvall. Comparing network covers using mutual information. *arXiv:1202.0425*, 2012.

[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.

[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.

[13] 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.

[14] Claire Donnat and Susan Holmes. Tracking network dynamics: A survey using graph distances. *The Annals of Applied Statistics*, 12(2):971 – 1012, 2018. doi: 10.1214/18-AOAS1176.

[15] 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.

[16] 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 1073–1080, New York, NY, USA, 2009. Association for Computing Machinery. ISBN 9781605585161. doi: 10.1145/1553374.1553511.

[17] 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.

[18] Lawrence Hubert and Phipps Arabie. Comparing partitions. *Journal of Classification*, 2(1):193–218, Dec 1985. ISSN 1432-1343. doi: 10.1007/BF01908075.

[19] Mark 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.

[20] Mark E. J. Newman. Modularity and community structure in networks. *Proceedings of the National Academy of Sciences*, 103(23):8577–8582, 2006. doi: 10.1073/pnas.0601602103.

[21] Brian Karrer and Mark E. J. Newman. Stochastic blockmodels and community structure in networks. *Phys. Rev. E*, 83:016107, Jan 2011. doi: 10.1103/PhysRevE.83.016107.

[22] Claude E. Shannon. A Mathematical Theory of Communication. *Bell Syst. Tech. J.*, 27:379–423, 1948.

[23] David A. Huffman. A Method for the Construction of Minimum-Redundancy Codes. *Proceedings of the IRE*, 40(9):1098–1101, 1952. doi: 10.1109/JRPROC.1952.273898.

[24] David F. Gleich. PageRank Beyond the Web. *SIAM Review*, 57(3):321–363, 2015. doi: 10.1137/140976649.

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

[26] Jelena Smiljanić, Christopher Blöcker, Anton Holmgren, Daniel Edler, Magnus Neuman, and Martin Rosvall. Community Detection with the Map Equation and Infomap: Theory and Applications. *arXiv:2311.04036*, 2023.

[27] Martin Rosvall and Carl T. Bergstrom. Multilevel compression of random walks on networks reveals hierarchical organization in large integrated systems. *PLOS ONE*, 6, 04 2011. doi: 10.1371/journal.pone.0018209.

[28] Christopher Blöcker. *Through the Coding-Lens: Community Detection and Beyond*. PhD thesis, Umeå University, 2022.

[29] Jianhua Lin. Divergence measures based on the Shannon entropy. *IEEE Transactions on Information Theory*, 37(1):145–151, 1991. doi: 10.1109/18.61115.

[30] 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.

[31] Jelena Smiljanić, 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.

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

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

[34] Paul Erdös and Alfréd Rényi. On Random Graphs I. *Publicationes Mathematicae Debrecen*, 6: 290, 1959.

[35] Paul Erdős and Alfréd Rényi. On the strength of connectedness of a random graph. *Acta Mathematica Academiae Scientiarum Hungarica*, 12(1):261–267, Mar 1964. ISSN 1588-2632. doi: 10.1007/BF02066689.

[36] Andrea Lancichinetti, Santo Fortunato, and Filippo Radicchi. Benchmark graphs for testing community detection algorithms. *Phys. Rev. E*, 78:046110, Oct 2008. doi: 10.1103/PhysRevE.78.046110.

[37] Michelle Girvan and Mark 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.

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

[39] 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.

[40] 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.

[41] Benedek Rozemberczki, Carl Allen, and Rik Sarkar. Multi-scale attributed node embedding. *Journal of Complex Networks*, 9(2):cnab014, 05 2021. ISSN 2051-1329. doi: 10.1093/comnet/cnab014.

[42] 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.

[43] 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.

[44] Aleix Bassolas, Anton Holmgren, Antoine Marot, Martin Rosvall, and Vincenzo Nicosia. Mapping nonlocal relationships between metadata and network structure with metadata-dependent encoding of random walks. *Science Advances*, 8(43):eabn7558, 2022. doi: 10.1126/sciadv.abn7558.

[45] Tiago P. Peixoto. Revealing Consensus and Dissensus between Network Partitions. *Phys. Rev. X*, 11:021003, Apr 2021. doi: 10.1103/PhysRevX.11.021003.

[46] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10):P10008, oct 2008. doi: 10.1088/1742-5468/2008/10/P10008.

# A Rewriting the Map Equation

Let $G = (V, E)$ be a network with nodes $V$, links $E$, and let $w_{uv} \in \mathbb{R}^+$ be the weight on the link from node $u$ to $v$. If $G$ is undirected, we can calculate the stationary visit rate for node $u$ as $p_u = \frac{\sum_{v \in V} w_{uv}}{\sum_{u \in V} \sum_{v \in V} w_{uv}}$. 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 $t_{uv} = \frac{w_{uv}}{\sum_{v \in V} w_{uv}}$ is the probability that a random walker at $u$ steps to $v$.

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

$$L(\mathsf{M}) = qH(Q) + \sum_{\mathsf{m} \in \mathsf{M}} p_{\mathsf{m}} H(P_{\mathsf{m}}), \tag{12}$$

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

Expanding the map equation, we obtain

$$L(\mathsf{M}) = -q \sum_{\mathsf{m} \in \mathsf{M}} \frac{q_{\mathsf{m}}}{q} \log_2 \frac{q_{\mathsf{m}}}{q} - \sum_{\mathsf{m} \in \mathsf{M}} p_{\mathsf{m}} \left( \frac{\mathsf{m}_{\text{exit}}}{p_{\mathsf{m}}} \log_2 \frac{\mathsf{m}_{\text{exit}}}{p_{\mathsf{m}}} + \sum_{u \in \mathsf{m}} \frac{p_u}{p_{\mathsf{m}}} \log_2 \frac{p_u}{p_{\mathsf{m}}} \right), \tag{13}$$

and after cancelling common factors

$$= - \sum_{\mathsf{m} \in \mathsf{M}} q_{\mathsf{m}} \log_2 \frac{q_{\mathsf{m}}}{q} - \sum_{\mathsf{m} \in \mathsf{M}} \left( \mathsf{m}_{\text{exit}} \log_2 \frac{\mathsf{m}_{\text{exit}}}{p_{\mathsf{m}}} + \sum_{u \in \mathsf{m}} p_u \log_2 \frac{p_u}{p_{\mathsf{m}}} \right). \tag{14}$$

Pulling out the summation and annotating the parts, we have

$$= - \sum_{\mathsf{m} \in \mathsf{M}} \overbrace{q_{\mathsf{m}} \log_2 \frac{q_{\mathsf{m}}}{q}}^{\text{entering module } \mathsf{m}} + \overbrace{\mathsf{m}_{\text{exit}} \log_2 \frac{\mathsf{m}_{\text{exit}}}{p_{\mathsf{m}}}}^{\text{exiting module } \mathsf{m}} + \overbrace{\sum_{u \in \mathsf{m}} p_u \log_2 \frac{p_u}{p_{\mathsf{m}}}}^{\text{visiting nodes in module } \mathsf{m}.} \tag{15}$$

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,

$$= - \sum_{\mathsf{m} \in \mathsf{M}} \left( \sum_{u \notin \mathsf{m}} p_u \sum_{v \in \mathsf{m}} t_{uv} \log_2 \frac{q_{\mathsf{m}}}{q} \right) + \left( \sum_{u \in \mathsf{m}} p_u \sum_{v \notin \mathsf{m}} t_{uv} \log_2 \frac{\mathsf{m}_{\text{exit}}}{p_{\mathsf{m}}} \right) \tag{16}$$

$$+ \left( \sum_{u \in \mathsf{m}} p_u \sum_{v \in \mathsf{m}} t_{uv} \log_2 \frac{p_v}{p_{\mathsf{m}}} \right) + \left( \sum_{u \notin \mathsf{m}} p_u \sum_{v \in \mathsf{m}} t_{uv} \log_2 \frac{p_v}{p_{\mathsf{m}}} \right). \tag{17}$$

Then, we merge the second and fourth terms into the first term. To merge the second term, we "reverse" the module exits, considering those steps that leave other modules to enter module $\mathsf{m}$ instead of steps that leave module $\mathsf{m}$. We denote node $u$'s and $v$'s module by $\mathsf{m}_u$ and $\mathsf{m}_v$, respectively,

$$= - \sum_{\mathsf{m} \in \mathsf{M}} \left( \sum_{u \notin \mathsf{m}} p_u \sum_{v \in \mathsf{m}} t_{uv} \log_2 \left( \overbrace{\frac{q_{\mathsf{m}_u}}{p_{\mathsf{m}_u}}}^{\text{exit } \mathsf{m}_u} \cdot \overbrace{\frac{q_{\mathsf{m}_v}}{q}}^{\text{enter } \mathsf{m}_v} \cdot \overbrace{\frac{p_v}{p_{\mathsf{m}_v}}}^{\text{visit node } v} \right) \right) + \left( \sum_{u \in \mathsf{m}} p_u \sum_{v \in \mathsf{m}} t_{uv} \log_2 \frac{p_v}{p_{\mathsf{m}_v}} \right). \tag{18}$$

We realise that, for each module $\mathsf{m}$, 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 $\mathsf{m}$, calculate the cost for transitioning to $v \in \mathsf{m}$ differently. We rewrite these two cases using the Kronecker delta $\delta$, summing over all nodes $u$,

$$= - \sum_{\mathsf{m} \in \mathsf{M}} \sum_{u \in V} p_u \sum_{v \in \mathsf{m}} t_{uv} \left[ (1 - \delta_{\mathsf{m}_u, \mathsf{m}_v}) \log_2 \left( \frac{q_{\mathsf{m}_u}}{p_{\mathsf{m}_u}} \cdot \frac{q_{\mathsf{m}_v}}{q} \cdot \frac{p_v}{p_{\mathsf{m}_v}} \right) + \delta_{\mathsf{m}_u, \mathsf{m}_v} \log_2 \frac{p_v}{p_{\mathsf{m}_v}} \right]. \tag{19}$$

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 $\mathsf{M}$ as

$$L(\mathsf{M}) = -\sum_{u\in V} p_u \sum_{v\in V} t_{uv} \left[ (1-\delta_{\mathsf{m}_u,\mathsf{m}_v}) \log_2 \left( \frac{q_{\mathsf{m}_u}}{p_{\mathsf{m}_u}} \cdot \frac{q_{\mathsf{m}_v}}{q} \cdot \frac{p_v}{p_{\mathsf{m}_v}} \right) + \delta_{\mathsf{m}_u,\mathsf{m}_v} \log_2 \frac{p_v}{p_{\mathsf{m}_v}} \right] \tag{20}$$

$$= -\sum_{u\in V} p_u \sum_{v\in V} t_{uv} \log_2 \left[ (1-\delta_{\mathsf{m}_u,\mathsf{m}_v}) \left( \frac{q_{\mathsf{m}_u}}{p_{\mathsf{m}_u}} \cdot \frac{q_{\mathsf{m}_v}}{q} \cdot \frac{p_v}{p_{\mathsf{m}_v}} \right) + \delta_{\mathsf{m}_u,\mathsf{m}_v} \frac{p_v}{p_{\mathsf{m}_v}} \right]. \tag{21}$$

The part inside the square brackets is known as map equation similarity, or *mapsim* for short, an information-theoretic measure for node similarity [13]. That is, we can calculate the codelength for a partition $\mathsf{M}$ using $\mathrm{mapsim}$, 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 $\mathsf{M}$. Thus, we can rewrite the map equation as

$$L(\mathsf{M}) = -\sum_{u\in V} p_u \sum_{v\in V} t_{uv} \log_2 \mathrm{mapsim}(\mathsf{M}, u, v). \tag{22}$$

## B Deriving Transition Probabilities from Partitions

Our goal is to define a partition dissimilarity measure that quantifies the expected additional number of bits for describing a random walker step using an "estimate" $\mathsf{B}$ of the network's "true" partition $\mathsf{A}$. We follow the same approach as the KL divergence, and consider two network partitions, $\mathsf{A}, \mathsf{B}$, as statistical processes that induce transition rates for the random walker. To see why we do this, consider a naïve attempt to define a dissimilarity measure using the observed transition rates $t_{uv}$,

$$D\left(\mathsf{A} \,||\, \mathsf{B}\right) = \sum_{u \in V} p_u \sum_{v \in V} t_{uv} \log_2 \frac{\operatorname{mapsim}\left(\mathsf{A}, u, v\right)}{\operatorname{mapsim}\left(\mathsf{B}, u, v\right)}. \tag{23}$$

The issue with this measure is that, because it uses the transition rates $t_{uv}$, it essentially assumes the same statistical process for both $\mathsf{A}$ and $\mathsf{B}$ and only computes the codelength difference between the two partitions. Via logarithm rules and by substituting the definition from Equation (6), we can simplify Equation (23) to $D\left(\mathsf{A} \,||\, \mathsf{B}\right) = L\left(\mathsf{A}\right) - L\left(\mathsf{B}\right)$. That is, we would simply learn by how much the codelength would increase or decrease if we use the other partition. But this is not what we intend to measure. To fix this, we need to consider the transition rates induced by the partition.

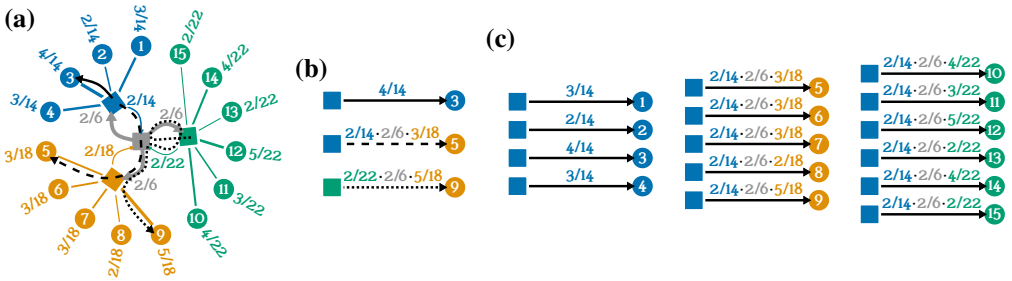


**Figure 6:** Partitions induce transition rates. **(a)** The same partition as in Figure 2c, annotated with transition and visit rates. The solid, dashed, and dotted arrows are examples of random-walker steps. **(b)** We derive transition rates between nodes according to mapsim: Transition rates depend on the source node's module, not on the source node itself (Equation (7)). Therefore, shortest paths 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 partition shown in Figure 6a 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 most recently visited node, the paths begin at the square-shaped nodes that represent the modules.

Figure 6b details at what rates a random walker follows the solid, dashed, and dotted paths. A random walker in the blue module visits node 3 at rate $\frac{4}{14}$. A random walker in the blue module exits at rate $\frac{2}{14}$, enters the orange module at rate $\frac{2}{6}$, and 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 shortest paths. Therefore, a random walker in the green module exits at rate $\frac{2}{22}$, 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 transition.

Figure 6c shows all shortest paths starting in the blue module and their usage rates. Since we only consider shortest paths, their usage rates do not sum to 1; here, their sum is approximately $0.94$. Because paths with loops are ways to make detours from shortest paths, their usage rate is proportional to the shortest path they contain. To obtain the transition probability from $u$ to $v$ according to partition $\mathsf{M}$, we normalise with the sum of transition rates for the shortest paths from $u$ to all nodes $v$,

$$t^{\mathsf{M}}_{uv} = \frac{\operatorname{mapsim}\left(\mathsf{M}, u, v\right)}{\sum_{v \in V} \operatorname{mapsim}\left(\mathsf{M}, u, v\right)}. \tag{24}$$

## C Comparing Maps

Network partitions represent the organisational structure of networks as possibly nested random processes. Figure 7 shows two different partitions for the same network, one of which is hierarchical, that is, there is at least one module that contains further sub-modules. Each of these two partitions corresponds to different assumed random-walker dynamics, inducing different transition rates. Using our definition of flow divergence,

$$D_F\left(\mathsf{A}\,||\,\mathsf{B}\right) = \sum_{u\in V} p_u \sum_{v\in V} t^{\mathsf{A}}_{uv} \log_2 \frac{s\left(\mathsf{A},u,v\right)}{s\left(\mathsf{B},u,v\right)}. \tag{25}$$

we obtain $D_F\left(\mathsf{M}_2\,||\,\mathsf{M}_3\right) \approx 0.09$ bits and $D_F\left(\mathsf{M}_3\,||\,\mathsf{M}_2\right) \approx 0.01$ bits. Assuming that $\mathsf{M}_2$ captures the random walker's true dynamics, the expected additional cost per step for using $\mathsf{M}_3$ is $0.09$ bits. Conversely, assuming that $\mathsf{M}_3$ describes the random walker's true dynamics, the expected additional cost per step for using $\mathsf{M}_2$ is $0.01$ bits. That is, besides the higher cost of $\mathsf{M}_2$ in terms of codelength, choosing $\mathsf{M}_2$ while $\mathsf{M}_3$ represents the network's true structure would lead to a higher excess in coding cost than the other way around.

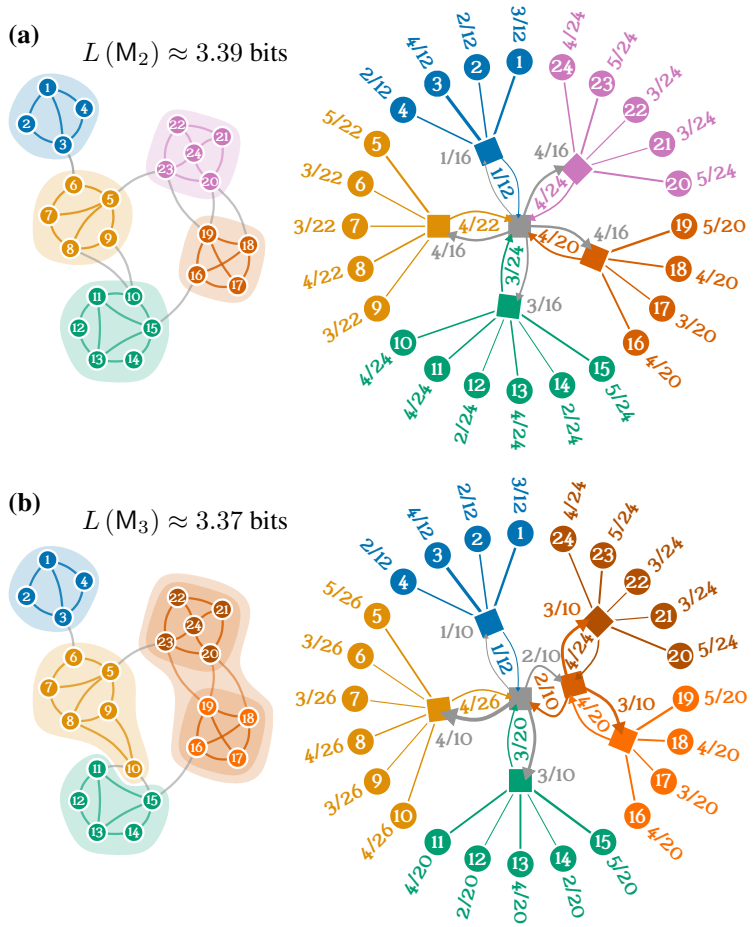


**Figure 7:** Different partitions for the same network, drawn on the network in the left column and as a tree in the right column. **(a)** A two-level map, $\mathsf{M}_2$, of the network into five modules. **(b)** A three-level map, $\mathsf{M}_3$, of the network into four modules, one of which has 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 $\mathsf{M}_2$ and $\frac{3}{20}$ in map $\mathsf{M}_3$, respectively.

# D Computational Complexity of Flow Divergence

Computing flow divergence between two partitions $\mathsf{A}$ and $\mathsf{B}$ requires considering $n^2$ $\mathrm{mapsim}$ scores per partition, where $n = |V|$. Therefore, it can be naïvely computed in time $\mathcal{O}\left(d_m \cdot n^2\right)$ with a double loop over the nodes to compute all pairwise $\mathrm{mapsim}$ values, where each $\mathrm{mapsim}$ value can be computed in time $\mathcal{O}\left(d_m\right)$ based on a tree data structure. Here, $d_m$ is the average depth at which the modules are located in the network's community structure, which is typically small [27, 45]. However, the regularities in the networks' community structures and the details of $\mathrm{mapsim}$ make it possible to compute flow divergence in time $\mathcal{O}\left(d_m \cdot m^2\right)$, where $m = |\mathsf{A} \times \mathsf{B}|$, and $\mathsf{A} \times \mathsf{B} = \{\mathsf{m_a} \cap \mathsf{m_b} \mid \mathsf{m_a} \in \mathsf{A}, \mathsf{m_b} \in \mathsf{B}, \mathsf{m_a} \cap \mathsf{m_b} \neq \emptyset\}$ is the set of non-empty intersections between the modules in $\mathsf{A}$ and $\mathsf{B}$, also called the meet between $\mathsf{A}$ and $\mathsf{B}$ [6].

Since $\mathrm{mapsim}\left(\mathsf{M}, u, v\right)$ depends on the source node's module, $\mathsf{m}_u$, but not the source node $u$ itself, we can drop the dependence on the source node and consider the source module instead, that is, $\mathrm{mapsim}\left(\mathsf{M}, u, v\right) = \mathrm{mapsim}\left(\mathsf{M}, \mathsf{m}_u, v\right)$. Consequently, the number of $\mathrm{mapsim}$ values we need to compute reduces to $m \cdot n$, with typically $m \ll n$. Theoretical and empirical evidence suggest that the number of modules typically scales as $m = \mathcal{O}\left(\sqrt{n}\right)$ [30]. Furthermore, based on its definition, we can decompose $\mathrm{mapsim}\left(\mathsf{M}, \mathsf{m}_u, v\right)$ as $\mathrm{mapsim}\left(\mathsf{M}, \mathsf{m}_u, \mathsf{m}_v\right) \cdot \frac{p_v}{p_{\mathsf{m}_v}}$, meaning that we can precompute and reuse the $\mathrm{mapsim}$ values between all pairs of source and target modules and aggregate the node visits per module. Therefore, we only need to consider the $\mathrm{mapsim}$ values between $m^2$ pairs of modules. In the worst case, that is, when each node is assigned to its own singleton module, the complexity becomes $\mathcal{O}\left(n^2\right)$ because the depth $d_m$ in this case would be 2 for each of the singleton modules, which is a constant. However, with $m = \mathcal{O}\left(\sqrt{n}\right)$ many modules, there are $\mathcal{O}\left(n\right)$ many pairs of modules. For each pair of modules, we need to find the source module $\mathsf{m}_u$ and target module $\mathsf{m}_v$ in the partition tree and multiply the random walker's transition rates along the shortest path from $\mathsf{m}_u$ to $\mathsf{m}_v$. Based on a tree data structure, this can be done in time $\mathcal{O}\left(d_m\right)$ for each pair of modules by caching the modules' locations in the tree in a hash table with constant-time lookup. Overall, this results in time $\mathcal{O}\left(d_m \cdot m^2\right)$ for precomputing the $\mathrm{mapsim}$ values between all pairs of modules.

In the following two subsections, we provide details on how flow divergence can be decomposed and how its terms can be regrouped for efficient computation.

**Notation.** For layout reasons, we use the shorthand $s\left(\mathsf{M}, u, v\right)$ to denote $\mathrm{mapsim}\left(\mathsf{M}, u, v\right)$.

## D.1 Without Normalisation

We have defined flow divergence as

$$D_F\left(\mathsf{A} \,||\, \mathsf{B}\right) = \sum_{u \in V} p_u \sum_{v \in V} t_{uv}^{\mathsf{A}} \log_2 \frac{s\left(\mathsf{A}, u, v\right)}{s\left(\mathsf{B}, u, v\right)}. \tag{26}$$

Naïvely, we can compute flow divergence in time $\mathcal{O}\left(d_m \cdot n^2\right)$ with a double loop for all pairs of nodes, where computing each $\mathrm{mapsim}$ value requires time $\mathcal{O}\left(d_m\right)$ using a tree data structure, and $d_m$ is the average depth at which the modules are located in the network's community structure, or equivalently, in the partition tree. However, by expanding and simplifying the terms in Equation (26), we can compute flow divergence efficiently in time $\mathcal{O}\left(d_m \cdot m^2\right)$. We begin by substituting the definition of the partition-dependent transition rates,

$$t_{uv}^{\mathsf{M}} = \frac{s\left(\mathsf{M}, u, v\right)}{\sum_{v \in V} s\left(\mathsf{M}, u, v\right)}. \tag{27}$$

into Equation (26),

$$D_F\left(\mathsf{A} \,||\, \mathsf{B}\right) = \sum_{u \in V} p_u \sum_{v \in V} \frac{s\left(\mathsf{A}, u, v\right)}{\sum_{v \in V} s\left(\mathsf{A}, u, v\right)} \log_2 \frac{s\left(\mathsf{A}, u, v\right)}{s\left(\mathsf{B}, u, v\right)}. \tag{28}$$

We pull out the normalisation factor for $u$ from the second sum and define the weight factor $\phi_u^{\mathsf{A}} = \frac{p_u}{\sum_{v \in V} s(\mathsf{A}, u, v)}$,

$$D_F\left(\mathsf{A} \,||\, \mathsf{B}\right) = \sum_{u \in V} \underbrace{\frac{p_u}{\sum_{v \in V} s\left(\mathsf{A}, u, v\right)}}_{=\phi_u^{\mathsf{A}}} \sum_{v \in V} s\left(\mathsf{A}, u, v\right) \log_2 \frac{s\left(\mathsf{A}, u, v\right)}{s\left(\mathsf{B}, u, v\right)}. \tag{29}$$

The weight factor $\phi_u^{\mathsf{A}}$ can be computed efficiently by exploiting the regularities in the modular network structure. As per the definition of $\mathrm{mapsim}$ (Equation (7)), we make use of two useful facts. First, that $\mathrm{mapsim}$ only depends on the source node's module $\mathsf{m}_u$ but not the source node itself, $\mathrm{mapsim}\left(\mathsf{M}, u, v\right) = \mathrm{mapsim}\left(\mathsf{M}, \mathsf{m}_u, v\right)$. And second, that $\mathrm{mapsim}$ can be decomposed into different parts that correspond to transitions between modules and visiting the target node, $\mathrm{mapsim}\left(\mathsf{M}, \mathsf{m}_u, v\right) = \mathrm{mapsim}\left(\mathsf{M}, \mathsf{m}_u, \mathsf{m}_v\right) \cdot \frac{p_v}{p_{\mathsf{m}_v}}$.

$$\phi_u^{\mathsf{A}} = \frac{p_u}{\sum_{v \in V} \mathsf{A}\left(u, v\right)} \tag{30}$$

$$= \frac{p_u}{\sum_{\mathsf{m} \in \mathsf{A}} \sum_{v \in \mathsf{m}} \mathsf{A}\left(\mathsf{m}_u, \mathsf{m}\right) \frac{p_v}{p_{\mathsf{m}}}} \tag{31}$$

$$= \frac{p_u}{\sum_{\mathsf{m} \in \mathsf{A}} \mathsf{A}\left(\mathsf{m}_u, \mathsf{m}\right) \sum_{v \in \mathsf{m}} \frac{p_v}{p_{\mathsf{m}}}} \tag{32}$$

$$= \frac{p_u}{\sum_{\mathsf{m} \in \mathsf{A}} \left(1 - \frac{\mathsf{m}_{\text{exit}}}{p_{\mathsf{m}}}\right) \mathsf{A}\left(\mathsf{m}_u, \mathsf{m}\right)} \tag{33}$$

Here, the last step follows from $p_{\mathsf{m}} = \mathsf{m}_{\text{exit}} + \sum_{u \in \mathsf{m}} p_u$. Because $\mathrm{mapsim}$ depends on the source module $\mathsf{m}_\mathsf{u}$ but not the specific source node $u$, we only need to consider $m^2 = \mathcal{O}\left(n\right)$ many pairs of modules, assuming that there are $m = \mathcal{O}\left(\sqrt{n}\right)$ many modules. With weight factor $\phi_u^{\mathsf{A}}$, we have

$$D_F\left(\mathsf{A} \,||\, \mathsf{B}\right) = \sum_{u \in V} \phi_u^{\mathsf{A}} \sum_{v \in V} s\left(\mathsf{A}, u, v\right) \log_2 \frac{s\left(\mathsf{A}, u, v\right)}{s\left(\mathsf{B}, u, v\right)}, \tag{34}$$

Next, we apply logarithm rules to split up flow divergence into two terms per source node,

$$D_F\left(\mathsf{A} \,||\, \mathsf{B}\right) = \sum_{u \in V} \phi_u^{\mathsf{A}} \left[\sum_{v \in V} s\left(\mathsf{A}, u, v\right) \log_2 s\left(\mathsf{A}, u, v\right)\right] \tag{35}$$

$$- \sum_{u \in V} \phi_u^{\mathsf{A}} \left[\sum_{v \in V} s\left(\mathsf{A}, u, v\right) \log_2 s\left(\mathsf{B}, u, v\right)\right], \tag{36}$$

where Equation (35) is the codelength when using $\mathsf{A}$, and Equation (36) is the codelength when using $\mathsf{B}$, both in the case that a random walker moves according to $\mathsf{A}$. Because $\mathrm{mapsim}$ only depends on the source module $\mathsf{m}_u$ but not the specific source node $u$, we can drop the dependence on the source node. This also allows us to aggregate the weights $\phi$ per source module, and we define $\Phi_{\mathsf{m}_u}^{\mathsf{A}} = \sum_{u \in \mathsf{m}_u} \phi_u^{\mathsf{A}}$.

$$D_F\left(\mathsf{A} \,||\, \mathsf{B}\right) = \sum_{\mathsf{m}_u \in \mathsf{A}} \Phi_{\mathsf{m}_u}^{\mathsf{A}} \underbrace{\left[\sum_{v \in V} s\left(\mathsf{A}, \mathsf{m}_u, v\right) \log_2 s\left(\mathsf{A}, \mathsf{m}_u, v\right)\right]}_{I} \tag{37}$$

$$- \sum_{\mathsf{m}_u \in \mathsf{A} \times \mathsf{B}} \Phi_{\mathsf{m}_u}^{\mathsf{A}} \underbrace{\left[\sum_{v \in V} s\left(\mathsf{A}, \mathsf{m}_u, v\right) \log_2 s\left(\mathsf{B}, \mathsf{m}_u, v\right)\right]}_{II}. \tag{38}$$

In Equation (37), we can simply sum over the source modules $\mathsf{m}_u$ in $\mathsf{A}$. However, in Equation (38), we need to sum over the intersections between the modules $\mathsf{m}_u$ in $\mathsf{A}$ and $\mathsf{B}$, where we define $\mathsf{A} \times \mathsf{B} = \left\{\mathsf{m}_\mathsf{a} \cap \mathsf{m}_\mathsf{b} \mid \mathsf{m}_\mathsf{a} \in \mathsf{A}, \mathsf{m}_\mathsf{b} \in \mathsf{B}\right\}$. These intersections can be computed efficiently with a single pass over the nodes and tabulating their memberships. We note that the expressions $\mathrm{mapsim}\left(\mathsf{A}, \mathsf{m}_u, v\right)$ and $\mathrm{mapsim}\left(\mathsf{B}, \mathsf{m}_u, v\right)$ are a slight abuse of notation because the modules $\mathsf{m}_u \in \mathsf{A} \times \mathsf{B}$ technically do not exist in $\mathsf{A}$ and $\mathsf{B}$, but any node in $\mathsf{m}_u$ can be used as a representative to compute these $\mathrm{mapsim}$ values. Now, we consider parts I and II in turn, starting with part I. As before, instead of directly summing over all target nodes $v$, we sum over the target modules $\mathsf{m}$ and then over their nodes,

$$\left(I\right) = \sum_{v \in V} s\left(\mathsf{A}, \mathsf{m}_u, v\right) \log_2 s\left(\mathsf{A}, \mathsf{m}_u, v\right) = \sum_{\mathsf{m} \in \mathsf{A}} \sum_{v \in \mathsf{m}} s\left(\mathsf{A}, \mathsf{m}_u, v\right) \log_2 s\left(\mathsf{A}, \mathsf{m}_u, v\right), \tag{39}$$

pull out the last factor from the $\mathrm{mapsim}$ operators

$$= \sum_{\mathsf{m} \in \mathsf{A}} \sum_{v \in \mathsf{m}} s\left(\mathsf{A}, \mathsf{m}_u, \mathsf{m}\right) \frac{p_v}{p_{\mathsf{m}}} \log_2 \left(s\left(\mathsf{A}, \mathsf{m}_u, \mathsf{m}\right) \frac{p_v}{p_{\mathsf{m}}}\right), \tag{40}$$

pull out common factors

$$= \sum_{\mathsf{m}\in\mathsf{A}} s\left(\mathsf{A}, \mathsf{m}_u, \mathsf{m}\right) \sum_{v\in\mathsf{m}} \frac{p_v}{p_\mathsf{m}} \log_2 \left( s\left(\mathsf{A}, \mathsf{m}_u, \mathsf{m}\right) \frac{p_v}{p_\mathsf{m}} \right), \tag{41}$$

and apply logarithm rules and simplify

$$= \sum_{\mathsf{m}\in\mathsf{A}} s\left(\mathsf{A}, \mathsf{m}_u, \mathsf{m}\right) \left[ \sum_{v\in\mathsf{m}} \frac{p_v}{p_\mathsf{m}} \log_2 s\left(\mathsf{A}, \mathsf{m}_u, \mathsf{m}\right) + \sum_{v\in\mathsf{m}} \frac{p_v}{p_\mathsf{m}} \log_2 \frac{p_v}{p_\mathsf{m}} \right] \tag{42}$$

$$= \sum_{\mathsf{m}\in\mathsf{A}} s\left(\mathsf{A}, \mathsf{m}_u, \mathsf{m}\right) \left[ \left(1 - \frac{\mathsf{m}_{\text{exit}}}{p_\mathsf{m}}\right) \log_2 s\left(\mathsf{A}, \mathsf{m}_u, \mathsf{m}\right) + \sum_{v\in\mathsf{m}} \frac{p_v}{p_\mathsf{m}} \log_2 \frac{p_v}{p_\mathsf{m}} \right]. \tag{43}$$

Simplifying part II is done analogously and, again, requires considering the intersections between the modules from partitions $\mathsf{A}$ and $\mathsf{B}$.

$$(II) = \sum_{v\in V} s\left(\mathsf{A}, \mathsf{m}_u, v\right) \log_2 s\left(\mathsf{B}, \mathsf{m}_u, v\right) = \sum_{\mathsf{m_a}\in\mathsf{A}} \sum_{\mathsf{m_b}\in\mathsf{B}} \sum_{v\in\mathsf{m_a}\cap\mathsf{m_b}} s\left(\mathsf{A}, \mathsf{m}_u, v\right) \log_2 s\left(\mathsf{B}, \mathsf{m}_u, v\right) \tag{44}$$

Next, we pull out the last factors from the $\mathrm{mapsim}$ operator again,

$$= \sum_{\mathsf{m_a}\in\mathsf{A}} \sum_{\mathsf{m_b}\in\mathsf{B}} \sum_{v\in\mathsf{m_a}\cap\mathsf{m_b}} s\left(\mathsf{A}, \mathsf{m}_u, \mathsf{m_a}\right) \frac{p_v}{p_{\mathsf{m_a}}} \log_2 \left( s\left(\mathsf{B}, \mathsf{m}_u, \mathsf{m_b}\right) \frac{p_v}{p_{\mathsf{m_b}}} \right), \tag{45}$$

pull out common factors

$$= \sum_{\mathsf{m_a}\in\mathsf{A}} s\left(\mathsf{A}, \mathsf{m}_u, \mathsf{m_a}\right) \sum_{\mathsf{m_b}\in\mathsf{B}} \sum_{v\in\mathsf{m_a}\cap\mathsf{m_b}} \frac{p_v}{p_{\mathsf{m_a}}} \log_2 \left( s\left(\mathsf{B}, \mathsf{m}_u, \mathsf{m_b}\right) \frac{p_v}{p_{\mathsf{m_b}}} \right), \tag{46}$$

and apply logarithm rules

$$= \sum_{\mathsf{m_a}\in\mathsf{A}} s\left(\mathsf{A}, \mathsf{m}_u, \mathsf{m_a}\right) \sum_{\mathsf{m_b}\in\mathsf{B}} \left[ \sum_{v\in\mathsf{m_a}\cap\mathsf{m_b}} \frac{p_v}{p_{\mathsf{m_a}}} \log_2 s\left(\mathsf{B}, \mathsf{m}_u, \mathsf{m_b}\right) + \sum_{v\in\mathsf{m_a}\cap\mathsf{m_b}} \frac{p_v}{p_{\mathsf{m_a}}} \log_2 \frac{p_v}{p_{\mathsf{m_b}}} \right] \tag{47}$$

$$= \sum_{\mathsf{m_a}\in\mathsf{A}} s\left(\mathsf{A}, \mathsf{m}_u, \mathsf{m_a}\right) \sum_{\mathsf{m_b}\in\mathsf{B}} \left[ \frac{p_{\mathsf{m_a}\cap\mathsf{m_b}}}{p_{\mathsf{m_a}}} \log_2 s\left(\mathsf{B}, \mathsf{m}_u, \mathsf{m_b}\right) + \sum_{v\in\mathsf{m_a}\cap\mathsf{m_b}} \frac{p_v}{p_{\mathsf{m_a}}} \log_2 \frac{p_v}{p_{\mathsf{m_b}}} \right], \tag{48}$$

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

$$D_F\left(\mathsf{A} \,||\, \mathsf{B}\right)$$

$$= \sum_{\mathsf{m}_u\in\mathsf{A}} \Phi^{\mathsf{A}}_{\mathsf{m}_u} \sum_{\mathsf{m}\in\mathsf{A}} s\left(\mathsf{A}, \mathsf{m}_u, \mathsf{m}\right) \left[ \left(1 - \frac{\mathsf{m}_{\text{exit}}}{p_\mathsf{m}}\right) \log_2 s\left(\mathsf{A}, \mathsf{m}_u, \mathsf{m}\right) + \sum_{v\in\mathsf{m}} \frac{p_v}{p_\mathsf{m}} \log_2 \frac{p_v}{p_\mathsf{m}} \right] \tag{49}$$

$$- \sum_{\mathsf{m}_u\in\mathsf{A}\times\mathsf{B}} \Phi^{\mathsf{A}}_{\mathsf{m}_u} \sum_{\mathsf{m_a}\in\mathsf{A}} s\left(\mathsf{A}, \mathsf{m}_u, \mathsf{m_a}\right) \sum_{\mathsf{m_b}\in\mathsf{B}} \left[ \frac{p_{\mathsf{m_a}\cap\mathsf{m_b}}}{p_{\mathsf{m_a}}} \log_2 s\left(\mathsf{B}, \mathsf{m}_u, \mathsf{m_b}\right) + \sum_{v\in\mathsf{m_a}\cap\mathsf{m_b}} \frac{p_v}{p_{\mathsf{m_a}}} \log_2 \frac{p_v}{p_{\mathsf{m_b}}} \right]. \tag{50}$$

### D.2 With Normalisation

Because $\mathrm{mapsim}\left(\mathsf{M}, u, v\right)$ is based on the shortest path from node $u$ to node $v$ in the coding tree induced by partition $\mathsf{M}$, certain paths are never considered, specifically those that are not shortest paths and contain loops. Consequently, when modules are present, the sum $\sum_{v\in V} \mathrm{mapsim}\left(\mathsf{M}, u, v\right)$ is smaller than one for every node and, thus, the transition rates originating at $u$, that is $\{\mathrm{mapsim}\left(\mathsf{M}, u, v\right) \mid v \in V\}$, do not form a probability distribution. Therefore, flow divergence is not an expected KL divergence. We can easily fix this by defining a normalised version of flow divergence based on the partition-dependent transition rates from Equation (24).

$$\tilde{D}_F\left(\mathsf{A} \,||\, \mathsf{B}\right) = \sum_{u\in V} p_u \sum_{v\in V} t^{\mathsf{A}}_{uv} \log_2 \frac{t^{\mathsf{A}}_{uv}}{t^{\mathsf{B}}_{uv}}. \tag{51}$$

We expand Equation (51) by plugging in the partition-dependent transition rates from Equation (24),

$$\tilde{D}_F\left(\mathsf{A}\,||\,\mathsf{B}\right) = \sum_{u\in V} p_u \sum_{v\in V} \frac{s\left(\mathsf{A},u,v\right)}{\sum_{v\in V} s\left(\mathsf{A},u,v\right)} \log_2\left(\frac{s\left(\mathsf{A},u,v\right)}{s\left(\mathsf{B},u,v\right)} \cdot \frac{\sum_{v\in V} s\left(\mathsf{B},u,v\right)}{\sum_{v\in V} s\left(\mathsf{A},u,v\right)}\right). \tag{52}$$

Then, we reorder and apply logarithm rules,

$$= \sum_{u\in V} \underbrace{\frac{p_u}{\sum_{v\in V} s\left(\mathsf{A},u,v\right)}}_{=\phi_u^{\mathsf{A}} \text{ (see Equation (33))}} \sum_{v\in V} s\left(\mathsf{A},u,v\right)\left(\log_2 \frac{s\left(\mathsf{A},u,v\right)}{s\left(\mathsf{B},u,v\right)} + \log_2 \frac{\sum_{v\in V} s\left(\mathsf{B},u,v\right)}{\sum_{v\in V} s\left(\mathsf{A},u,v\right)}\right) \tag{53}$$

$$= \sum_{u\in V} \phi_u^{\mathsf{A}} \sum_{v\in V}\left[s\left(\mathsf{A},u,v\right)\log_2 \frac{s\left(\mathsf{A},u,v\right)}{s\left(\mathsf{B},u,v\right)} + s\left(\mathsf{A},u,v\right)\log_2 \frac{\sum_{v\in V} s\left(\mathsf{B},u,v\right)}{\sum_{v\in V} s\left(\mathsf{A},u,v\right)}\right]. \tag{54}$$

We split this up into two parts, one for the non-normalised version of flow divergence, and one for normalising it.

$$= \sum_{u\in V} \phi_u^{\mathsf{A}}\left[\sum_{v\in V} s\left(\mathsf{A},u,v\right)\log_2 s\left(\mathsf{A},u,v\right) - \sum_{v\in V} s\left(\mathsf{A},u,v\right)\log_2 s\left(\mathsf{B},u,v\right)\right] \tag{55}$$

$$- \sum_{u\in V} \phi_u^{\mathsf{A}}\left[\underbrace{\sum_{v\in V} s\left(\mathsf{A},u,v\right)\log_2\left(\sum_{v\in V} s\left(\mathsf{A},u,v\right)\right)}_{I} - \underbrace{\sum_{v\in V} s\left(\mathsf{A},u,v\right)\log_2\left(\sum_{v\in V} s\left(\mathsf{B},u,v\right)\right)}_{II}\right]. \tag{56}$$

Here, Equation (55) is exactly the non-normalised version of flow divergence. The terms in Equation (56) normalise flow divergence and turn it into an expected KL divergence. We can compute these normalisation terms efficiently in a similar way as Equation (34), that is, by exploiting the partitions' modular structure. We begin with part I,

$$(I) = \sum_{v\in V} s\left(\mathsf{A},u,v\right) \underbrace{\log_2\left(\sum_{v\in V} s\left(\mathsf{A},u,v\right)\right)}_{=\lambda_u^{\mathsf{A}}} \tag{57}$$

$$= \lambda_u^{\mathsf{A}} \cdot \sum_{\mathsf{m}\in\mathsf{A}} \sum_{v\in\mathsf{m}} s\left(\mathsf{A},\mathsf{m}_u,\mathsf{m}\right)\frac{p_v}{p_{\mathsf{m}}} \tag{58}$$

$$= \lambda_u^{\mathsf{A}} \cdot \sum_{\mathsf{m}\in\mathsf{A}} s\left(\mathsf{A},\mathsf{m}_u,\mathsf{m}\right)\left(1 - \frac{\mathsf{m}_{\text{exit}}}{p_{\mathsf{m}}}\right). \tag{59}$$

Similarly, for part II, we have

$$(II) = \sum_{v\in V} s\left(\mathsf{A},u,v\right)\log_2\left(\sum_{v\in V} s\left(\mathsf{B},u,v\right)\right) \tag{60}$$

$$= \lambda_u^{\mathsf{B}} \cdot \sum_{\mathsf{m}\in\mathsf{A}} s\left(\mathsf{A},\mathsf{m}_u,\mathsf{m}\right)\left(1 - \frac{\mathsf{m}_{\text{exit}}}{p_{\mathsf{m}}}\right). \tag{61}$$

Now, it only remains to compute $\lambda_u^{\mathsf{M}}$,

$$\lambda_u^{\mathsf{M}} = \log_2\left(\sum_{v\in V} s\left(\mathsf{M},u,v\right)\right) \tag{62}$$

$$= \log_2\left(\sum_{\mathsf{m}\in\mathsf{M}} \sum_{v\in\mathsf{m}} s\left(\mathsf{M},\mathsf{m}_u,\mathsf{m}\right)\frac{p_v}{p_{\mathsf{m}}}\right) \tag{63}$$

$$= \log_2\left(\sum_{\mathsf{m}\in\mathsf{M}} s\left(\mathsf{M},\mathsf{m}_u,\mathsf{m}\right)\left(1 - \frac{\mathsf{m}_{\text{exit}}}{p_{\mathsf{m}}}\right)\right) \tag{64}$$

Putting everything together and aggregating the normalisation terms per source module, we can rewrite the normalised version of flow divergence as

$$\tilde{D}_F\left(\mathsf{A}\,||\,\mathsf{B}\right)$$

$$= \sum_{\mathsf{m}_u \in \mathsf{A}} \Phi^{\mathsf{A}}_{\mathsf{m}_u} \sum_{\mathsf{m} \in \mathsf{A}} s\left(\mathsf{A}, \mathsf{m}_u, \mathsf{m}\right) \left[\left(1 - \frac{\mathsf{m}_{\text{exit}}}{p_{\mathsf{m}}}\right) \log_2 s\left(\mathsf{A}, \mathsf{m}_u, \mathsf{m}\right) + \sum_{v \in \mathsf{m}} \frac{p_v}{p_{\mathsf{m}}} \log_2 \frac{p_v}{p_{\mathsf{m}}}\right] \tag{65}$$

$$- \sum_{\mathsf{m}_u \in \mathsf{A}\times\mathsf{B}} \Phi^{\mathsf{A}}_{\mathsf{m}_u} \sum_{\mathsf{m}_\mathsf{a} \in \mathsf{A}} s\left(\mathsf{A}, \mathsf{m}_u, \mathsf{m}_\mathsf{a}\right) \sum_{\mathsf{m}_\mathsf{b} \in \mathsf{B}} \left[\frac{p_{\mathsf{m}_\mathsf{a} \cap \mathsf{m}_\mathsf{b}}}{p_{\mathsf{m}_\mathsf{a}}} \log_2 s\left(\mathsf{B}, \mathsf{m}_u, \mathsf{m}_\mathsf{b}\right) + \sum_{v \in \mathsf{m}_\mathsf{a} \cap \mathsf{m}_\mathsf{b}} \frac{p_v}{p_{\mathsf{m}_\mathsf{a}}} \log_2 \frac{p_v}{p_{\mathsf{m}_\mathsf{b}}}\right] \tag{66}$$

$$- \sum_{\mathsf{m}_u \in \mathsf{A}} \Phi^{\mathsf{A}}_{\mathsf{m}_u} \sum_{\mathsf{m} \in \mathsf{A}} s\left(\mathsf{A}, \mathsf{m}_u, \mathsf{m}\right) \left(1 - \frac{\mathsf{m}_{\text{exit}}}{p_{\mathsf{m}}}\right) \log_2 \left(\sum_{\mathsf{m} \in \mathsf{A}} s\left(\mathsf{A}, \mathsf{m}_u, \mathsf{m}\right) \left(1 - \frac{\mathsf{m}_{\text{exit}}}{p_{\mathsf{m}}}\right)\right) \tag{67}$$

$$+ \sum_{\mathsf{m}_u \in \mathsf{A}\times\mathsf{B}} \Phi^{\mathsf{A}}_{\mathsf{m}_u} \sum_{\mathsf{m} \in \mathsf{A}} s\left(\mathsf{A}, \mathsf{m}_u, \mathsf{m}\right) \left(1 - \frac{\mathsf{m}_{\text{exit}}}{p_{\mathsf{m}}}\right) \log_2 \left(\sum_{\mathsf{m} \in \mathsf{B}} s\left(\mathsf{B}, \mathsf{m}_u, \mathsf{m}\right) \left(1 - \frac{\mathsf{m}_{\text{exit}}}{p_{\mathsf{m}}}\right)\right), \tag{68}$$

where the first two parts are the non-normalised version of flow divergence, and the last two lines are for normalisation.

## E Computational Complexity of Flow Distance

We define flow distance, $d_F$, a normalised and symmetric version of flow divergence by following the ideas behind the Jensen-Shannon distance,

$$d_F\left(\mathsf{A},\mathsf{B}\right)=\sqrt{\frac{1}{2}\tilde{D}_F\left(\mathsf{A}\,||\,\mathsf{M}_{\mathsf{AB}}\right)+\frac{1}{2}\tilde{D}_F\left(\mathsf{B}\,||\,\mathsf{M}_{\mathsf{AB}}\right)}, \tag{69}$$

where $\mathsf{M}_{\mathsf{AB}}=\frac{1}{2}\left(\mathsf{A}+\mathsf{B}\right)$ is the "mixture partition" of $\mathsf{A}$ and $\mathsf{B}$, defined indirectly via its $\mathrm{mapsim}$ values,

$$\mathrm{mapsim}\left(\mathsf{M}_{\mathsf{AB}},u,v\right)=\frac{\mathrm{mapsim}\left(\mathsf{A},u,v\right)}{2}+\frac{\mathrm{mapsim}\left(\mathsf{B},u,v\right)}{2}. \tag{70}$$

**Notation.** Again, for layout reasons, we use the shorthand $s\left(\mathsf{M},u,v\right)$ to denote $\mathrm{mapsim}\left(\mathsf{M},u,v\right)$.

To compute $d_F$ efficiently, we need to be able to compute $\tilde{D}_F\left(\mathsf{A}\,||\,\mathsf{M}_{\mathsf{AB}}\right)$ and $\tilde{D}_F\left(\mathsf{B}\,||\,\mathsf{M}_{\mathsf{AB}}\right)$ efficiently. According to Equations (9) and (51), respectively, we have

$$\tilde{D}_F\left(\mathsf{A}\,||\,\mathsf{M}_{\mathsf{AB}}\right)=\sum_{u\in V}p_u\sum_{v\in V}t^{\mathsf{A}}_{uv}\log_2\frac{t^{\mathsf{A}}_{uv}}{t^{\mathsf{M}_{\mathsf{AB}}}_{uv}}, \tag{71}$$

which, following Equations (55) and (56), can be rewritten as

$$=\sum_{u\in V}\phi^{\mathsf{A}}_u\left[\sum_{v\in V}s\left(\mathsf{A},u,v\right)\log_2 s\left(\mathsf{A},u,v\right)-\sum_{v\in V}s\left(\mathsf{A},u,v\right)\log_2 s\left(\mathsf{M}_{\mathsf{AB}},u,v\right)\right] \tag{72}$$

$$-\sum_{u\in V}\phi^{\mathsf{A}}_u\left[\sum_{v\in V}s\left(\mathsf{A},u,v\right)\log_2\left(\sum_{v\in V}s\left(\mathsf{A},u,v\right)\right)-\sum_{v\in V}s\left(\mathsf{A},u,v\right)\log_2\left(\sum_{v\in V}s\left(\mathsf{M}_{\mathsf{AB}},u,v\right)\right)\right]. \tag{73}$$

We split this further up, substitute $s\left(\mathsf{M}_{\mathsf{AB}},u,v\right)=\frac{s(\mathsf{A},u,v)}{2}+\frac{s(\mathsf{B},u,v)}{2}$, and aggregate per source module,

$$=\sum_{\mathsf{m}_u\in\mathsf{A}}\Phi^{\mathsf{A}}_{\mathsf{m}_u}\sum_{\mathsf{m}\in\mathsf{A}}s\left(\mathsf{A},\mathsf{m}_u,\mathsf{m}\right)\left[\left(1-\frac{\mathsf{m}_{\text{exit}}}{p_{\mathsf{m}}}\right)\log_2 s\left(\mathsf{A},\mathsf{m}_u,\mathsf{m}\right)+\sum_{v\in\mathsf{m}}\frac{p_v}{p_{\mathsf{m}}}\log_2\frac{p_v}{p_{\mathsf{m}}}\right] \tag{74}$$

$$-\sum_{\mathsf{m}_u\in\mathsf{A}\times\mathsf{B}}\Phi^{\mathsf{A}}_{\mathsf{m}_u}\underbrace{\sum_{v\in V}s\left(\mathsf{A},\mathsf{m}_u,v\right)\log_2\left(\frac{s\left(\mathsf{A},\mathsf{m}_u,v\right)}{2}+\frac{s\left(\mathsf{B},\mathsf{m}_u,v\right)}{2}\right)}_{I} \tag{75}$$

$$-\sum_{\mathsf{m}_u\in\mathsf{A}}\Phi^{\mathsf{A}}_{\mathsf{m}_u}\sum_{\mathsf{m}\in\mathsf{A}}s\left(\mathsf{A},\mathsf{m}_u,\mathsf{m}\right)\left(1-\frac{\mathsf{m}_{\text{exit}}}{p_{\mathsf{m}}}\right)\log_2\left(\sum_{\mathsf{m}\in\mathsf{A}}s\left(\mathsf{A},\mathsf{m}_u,\mathsf{m}\right)\left(1-\frac{\mathsf{m}_{\text{exit}}}{p_{\mathsf{m}}}\right)\right) \tag{76}$$

$$+\sum_{\mathsf{m}_u\in\mathsf{A}\times\mathsf{B}}\Phi^{\mathsf{A}}_{\mathsf{m}_u}\underbrace{\sum_{v\in V}s\left(\mathsf{A},\mathsf{m}_u,v\right)\log_2\left(\sum_{v\in V}\frac{s\left(\mathsf{A},\mathsf{m}_u,v\right)}{2}+\frac{s\left(\mathsf{B},\mathsf{m}_u,v\right)}{2}\right)}_{II}. \tag{77}$$

Here, we consider Parts I and II, which involve the mixture partition $\mathsf{M}_{\mathsf{AB}}$; we have discussed the other parts in Appendix D. Unfortunately, the sum in the logarithm in Part I prevents us from simplifying this expression, and we need to consider a total of $n$ target nodes per source module, resulting in a time complexity of $\mathcal{O}\left(d_m\cdot m\cdot n\right)$. Part II captures the normalisation term for the mixture partition

$\mathsf{M}_{\mathsf{AB}}$, which, somewhat surprisingly, can be computed efficiently in time $\mathcal{O}\left(d_m \cdot m^2\right)$.

$$(II) = \sum_{v \in V} s\left(\mathsf{A}, u, v\right) \log_2 \left( \sum_{v \in V} \frac{s\left(\mathsf{A}, \mathsf{m}_u, v\right)}{2} + \frac{s\left(\mathsf{B}, \mathsf{m}_u, v\right)}{2} \right) \tag{78}$$

$$= \sum_{\mathsf{m} \in \mathsf{A}} \sum_{v \in \mathsf{m}} s\left(\mathsf{A}, \mathsf{m}_u, \mathsf{m}\right) \frac{p_v}{p_{\mathsf{m}}} \log_2 \left( \sum_{v \in V} \frac{s\left(\mathsf{A}, u, v\right)}{2} + \frac{s\left(\mathsf{B}, u, v\right)}{2} \right) \tag{79}$$

$$= \sum_{\mathsf{m} \in \mathsf{A}} s\left(\mathsf{A}, \mathsf{m}_u, \mathsf{m}\right) \left(1 - \frac{\mathsf{m}_{\text{exit}}}{p_{\mathsf{m}}}\right) \log_2 \left( \sum_{\mathsf{m} \in \mathsf{A}} \frac{s\left(\mathsf{A}, \mathsf{m}_u, \mathsf{m}\right) \left(1 - \frac{\mathsf{m}_{\text{exit}}}{p_{\mathsf{m}}}\right)}{2} + \sum_{\mathsf{m} \in \mathsf{B}} \frac{s\left(\mathsf{B}, \mathsf{m}_u, \mathsf{m}\right) \left(1 - \frac{\mathsf{m}_{\text{exit}}}{p_{\mathsf{m}}}\right)}{2} \right) \tag{80}$$

Putting everything together, we compute the normalised version of flow divergence between the reference partition $\mathsf{A}$ and the mixture partition $\mathsf{M}_{\mathsf{AB}}$ as follows,

$$\tilde{D}_F\left(\mathsf{A} \,||\, \mathsf{M}_{\mathsf{AB}}\right)$$

$$= \sum_{\mathsf{m}_u \in \mathsf{A}} \Phi^{\mathsf{A}}_{\mathsf{m}_u} \sum_{\mathsf{m} \in \mathsf{A}} s\left(\mathsf{A}, \mathsf{m}_u, \mathsf{m}\right) \left[ \left(1 - \frac{\mathsf{m}_{\text{exit}}}{p_{\mathsf{m}}}\right) \log_2 s\left(\mathsf{A}, \mathsf{m}_u, \mathsf{m}\right) + \sum_{v \in \mathsf{m}} \frac{p_v}{p_{\mathsf{m}}} \log_2 \frac{p_v}{p_{\mathsf{m}}} \right] \tag{81}$$

$$- \sum_{\mathsf{m}_u \in \mathsf{A} \times \mathsf{B}} \Phi^{\mathsf{A}}_{\mathsf{m}_u} \sum_{v \in V} s\left(\mathsf{A}, \mathsf{m}_u, v\right) \log_2 \left( \frac{s\left(\mathsf{A}, \mathsf{m}_u, v\right)}{2} + \frac{s\left(\mathsf{B}, \mathsf{m}_u, v\right)}{2} \right) \tag{82}$$

$$- \sum_{\mathsf{m}_u \in \mathsf{A}} \Phi^{\mathsf{A}}_{\mathsf{m}_u} \sum_{\mathsf{m} \in \mathsf{A}} s\left(\mathsf{A}, \mathsf{m}_u, \mathsf{m}\right) \left(1 - \frac{\mathsf{m}_{\text{exit}}}{p_{\mathsf{m}}}\right) \log_2 \left( \sum_{\mathsf{m} \in \mathsf{A}} \left(1 - \frac{\mathsf{m}_{\text{exit}}}{p_{\mathsf{m}}}\right) s\left(\mathsf{A}, \mathsf{m}_u, \mathsf{m}\right) \right) \tag{83}$$

$$+ \sum_{\mathsf{m}_u \in \mathsf{A} \times \mathsf{B}} \Phi^{\mathsf{A}}_{\mathsf{m}_u} \sum_{\mathsf{m} \in \mathsf{A}} s\left(\mathsf{A}, \mathsf{m}_u, \mathsf{m}\right) \left(1 - \frac{\mathsf{m}_{\text{exit}}}{p_{\mathsf{m}}}\right) \log_2 \left( \sum_{\mathsf{m} \in \mathsf{A}} \frac{s\left(\mathsf{A}, \mathsf{m}_u, \mathsf{m}\right) \left(1 - \frac{\mathsf{m}_{\text{exit}}}{p_{\mathsf{m}}}\right)}{2} + \sum_{\mathsf{m} \in \mathsf{B}} \frac{s\left(\mathsf{B}, \mathsf{m}_u, \mathsf{m}\right) \left(1 - \frac{\mathsf{m}_{\text{exit}}}{p_{\mathsf{m}}}\right)}{2} \right) \tag{84}$$

Altogether, flow distance can be computed in time $\mathcal{O}\left(d_m \cdot m \cdot n\right)$. Because we defined the mixture partition $\mathsf{M}_{\mathsf{AB}}$ indirectly via its $\mathrm{mapsim}$ values, we cannot access its structural regularities to compute flow distance more efficiently.

# F Comparison of normalised and non-normalised flow divergence values

This appendix includes additional results showing the normalised and non-normalised flow divergence values for the synthetic and empirical networks used in our evaluation in Sections 4.2 and 4.3. We find that, for synthetic networks with communities (LFR networks) and real networks, the normalised and non-normalised versions of flow divergence yield virtually identical results. For synthetic networks without communities (Erdős-Rényi random graphs), the normalised and non-normalised versions of flow divergence return different results, where the difference tends to be larger in larger networks.

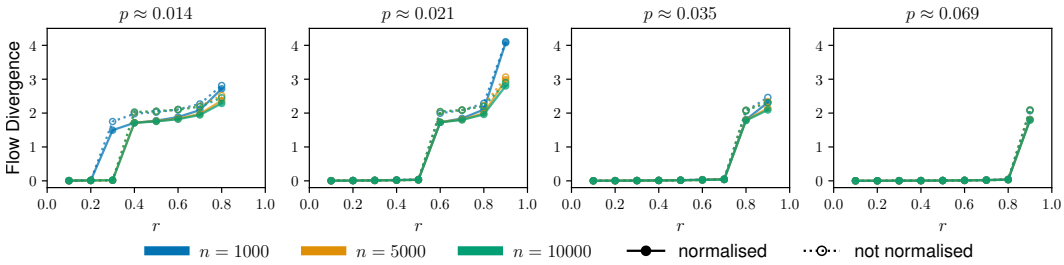


**Figure 8:** Normalised and non-normalised flow divergence values for Erdős-Rényi random graphs with the same setup as described in Section 4.2. Erdős-Rényi random graphs do not have communities, however, as we remove links and spurious communities are detected, the two versions of flow divergence return slightly different results.

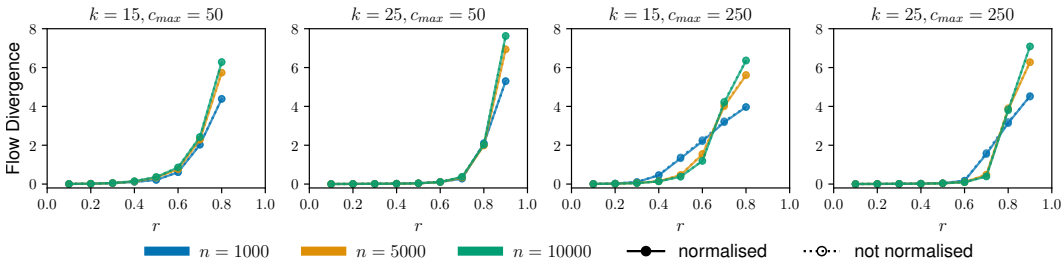


**Figure 9:** Normalised and non-normalised flow divergence values for LFR networks with the same setup as described in Section 4.2. Both versions of flow divergence return virtually the same results.

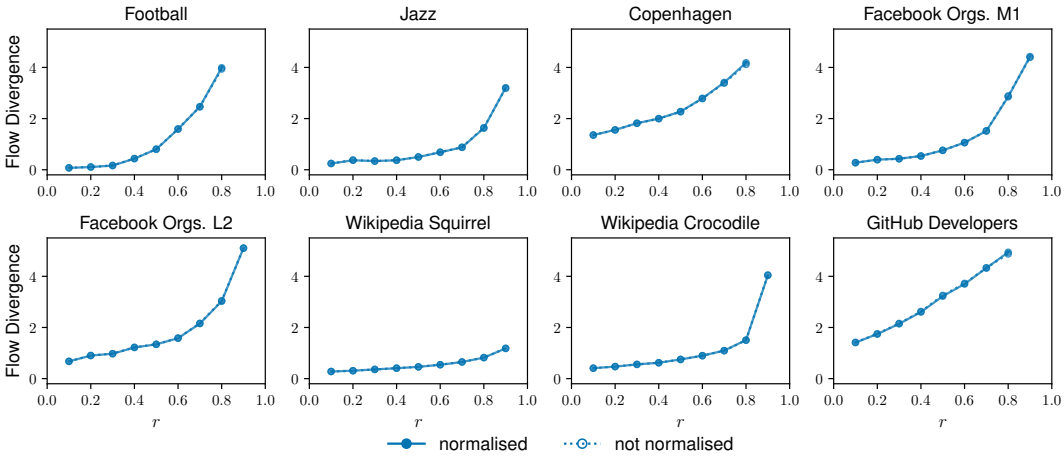


**Figure 10:** Normalised and non-normalised flow divergence values for real networks with the same setup as described in Section 4.3. Both versions of flow divergence return virtually the same results.

# G Networks with Partitions and Flow Divergence and Flow Distance Values

Here, we show some networks and their partitions as detected with Infomap for different fractions of removed links. We annotate each network with its flow divergence and flow distance compared to the reference partition. Different from the other experiments, we consecutively remove links from the network, that is, we obtain the network for $r = 0.2$ by removing further links from the network for $r = 0.1$, the network for $r = 0.3$ by removing further links from the network for $r = 0.2$, and so on.

## G.1 Erdős-Rényi Random Graph

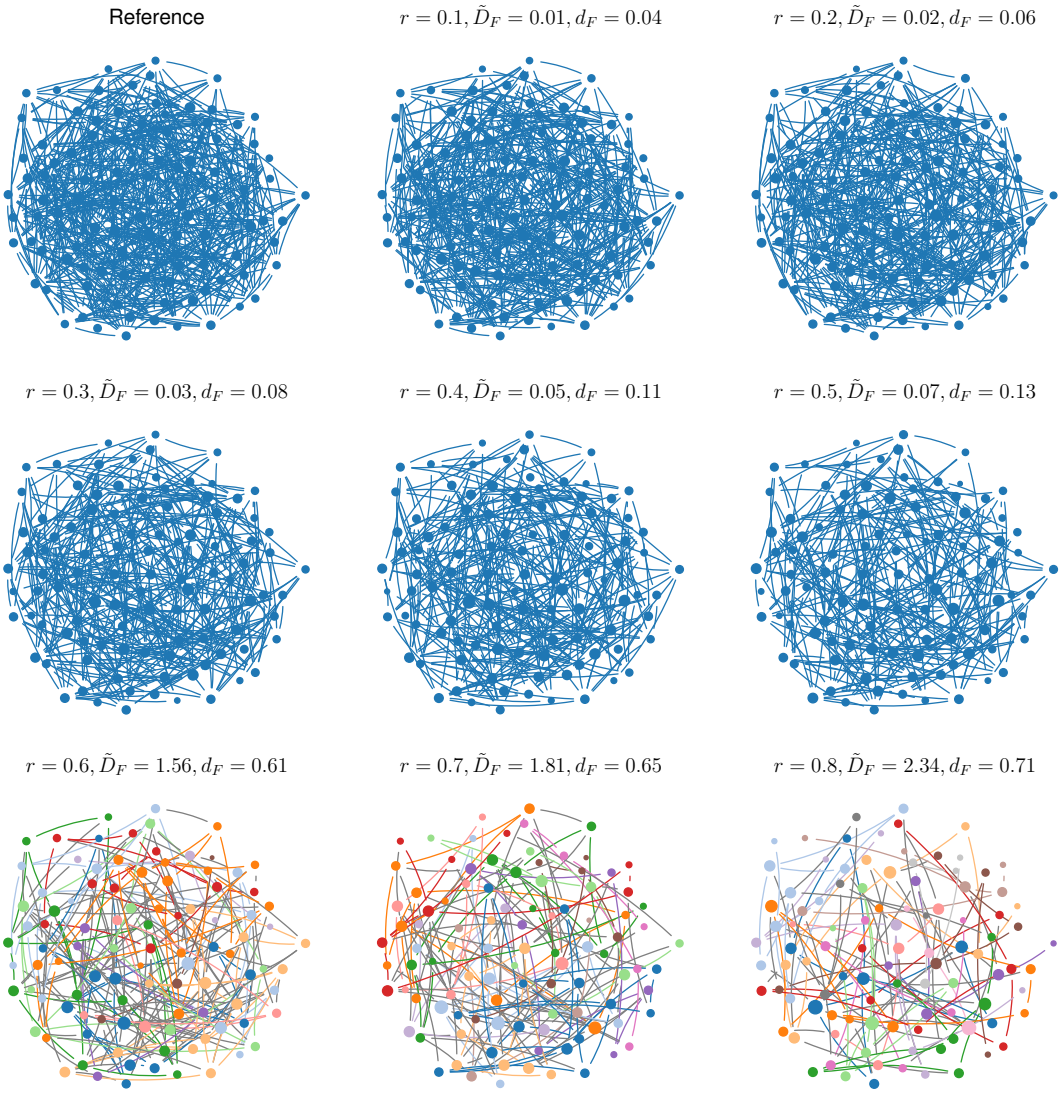


**Figure 11:** An Erdős-Rényi network with 100 nodes and $p = \frac{5 \ln n}{n}$. For different $r$-fractions of removed links, we show the resulting partitions with normalised flow divergence values $\tilde{D}_F$ and flow distance values $d_F$. The nodes' sizes correspond to their visit rates. Because the communities in the last three cases are spurious, they do not naturally match, making it difficult to align their colours.

### G.2 Lancichinetti–Fortunato–Radicchi Network

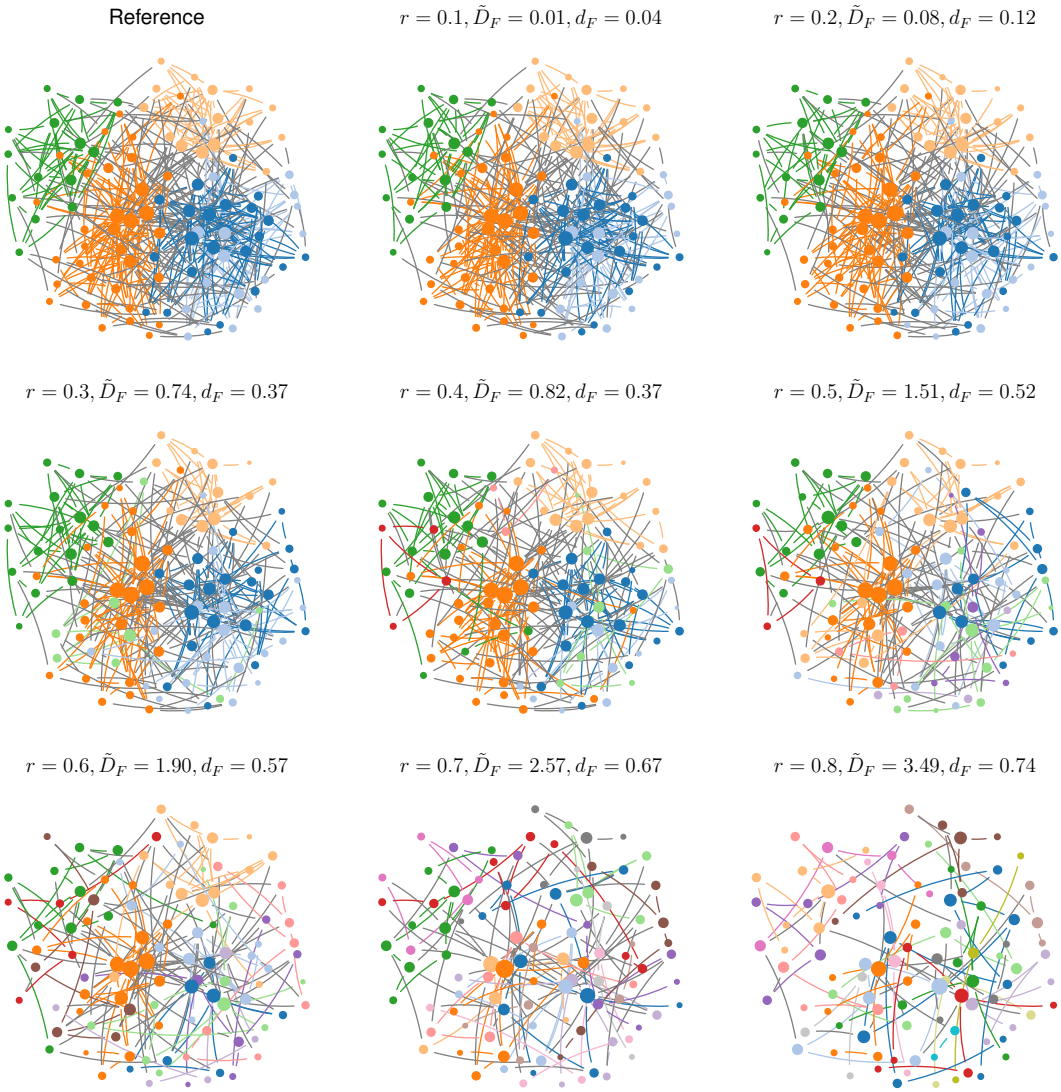


**Figure 12:** A Lancichinetti–Fortunato–Radicchi network with 100 nodes and 5 communities. We generated the network by setting the average node degree $k = 10$, maximum node degree $k_{\text{max}} = 30$, mixing $\mu = 0.3$, minimum community size $c_{\text{min}} = 10$, and maximum community size $c_{\text{max}} = 25$. For different $r$-fractions of removed links, we show the resulting partitions with normalised flow divergence values $\tilde{D}_F$ and flow distance values $d_F$. The flow divergence and flow distance values indicate that the communities found for up to 20% removed links are relatively stable. Once we remove 30% or more of the links, spurious communities start to appear, as confirmed visually by the additional colours.

### G.3 Real Networks

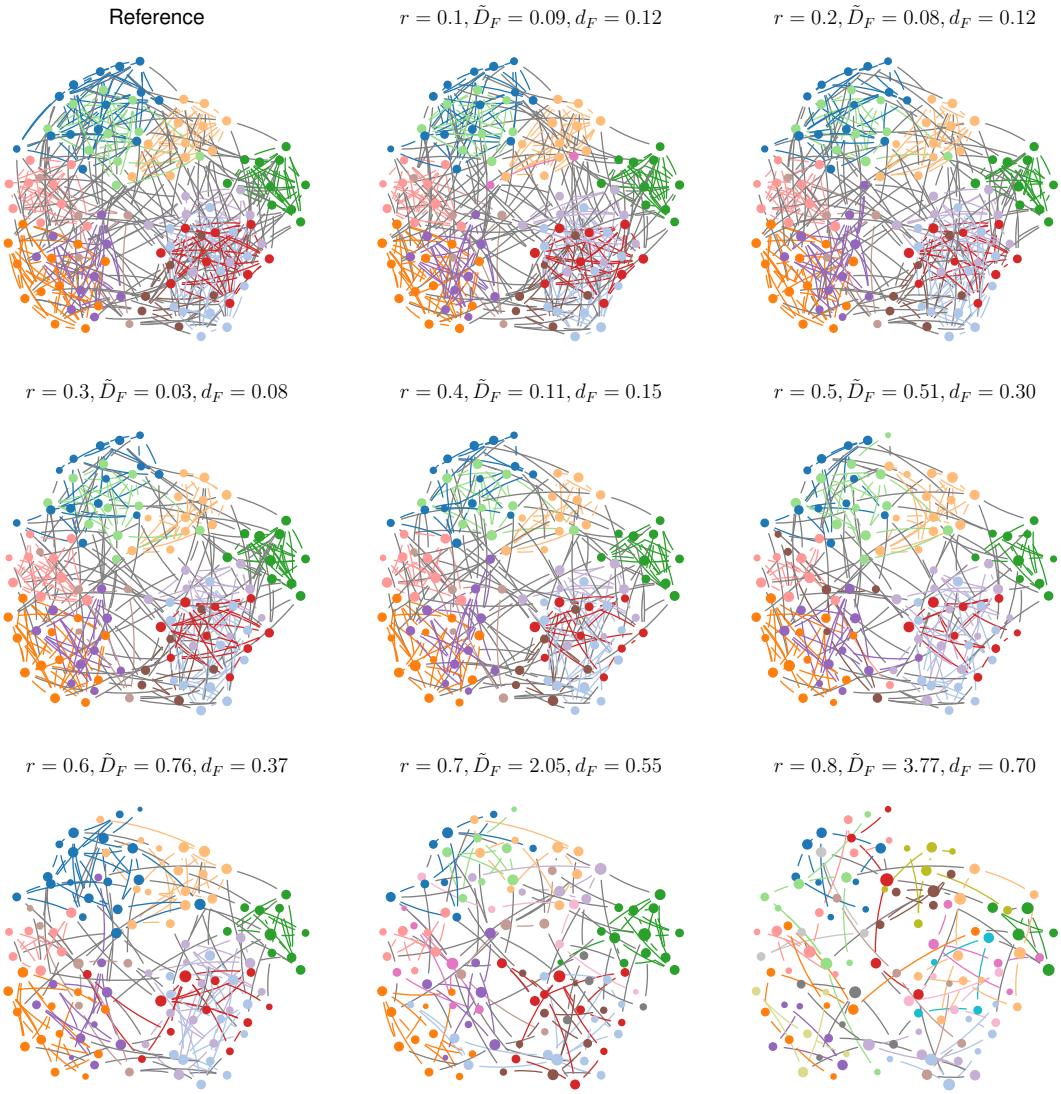


**Figure 13:** The football network with 115 nodes. For different $r$-fractions of removed links, we show the resulting partitions with normalised flow divergence values $\tilde{D}_F$ and flow distance values $d_F$.

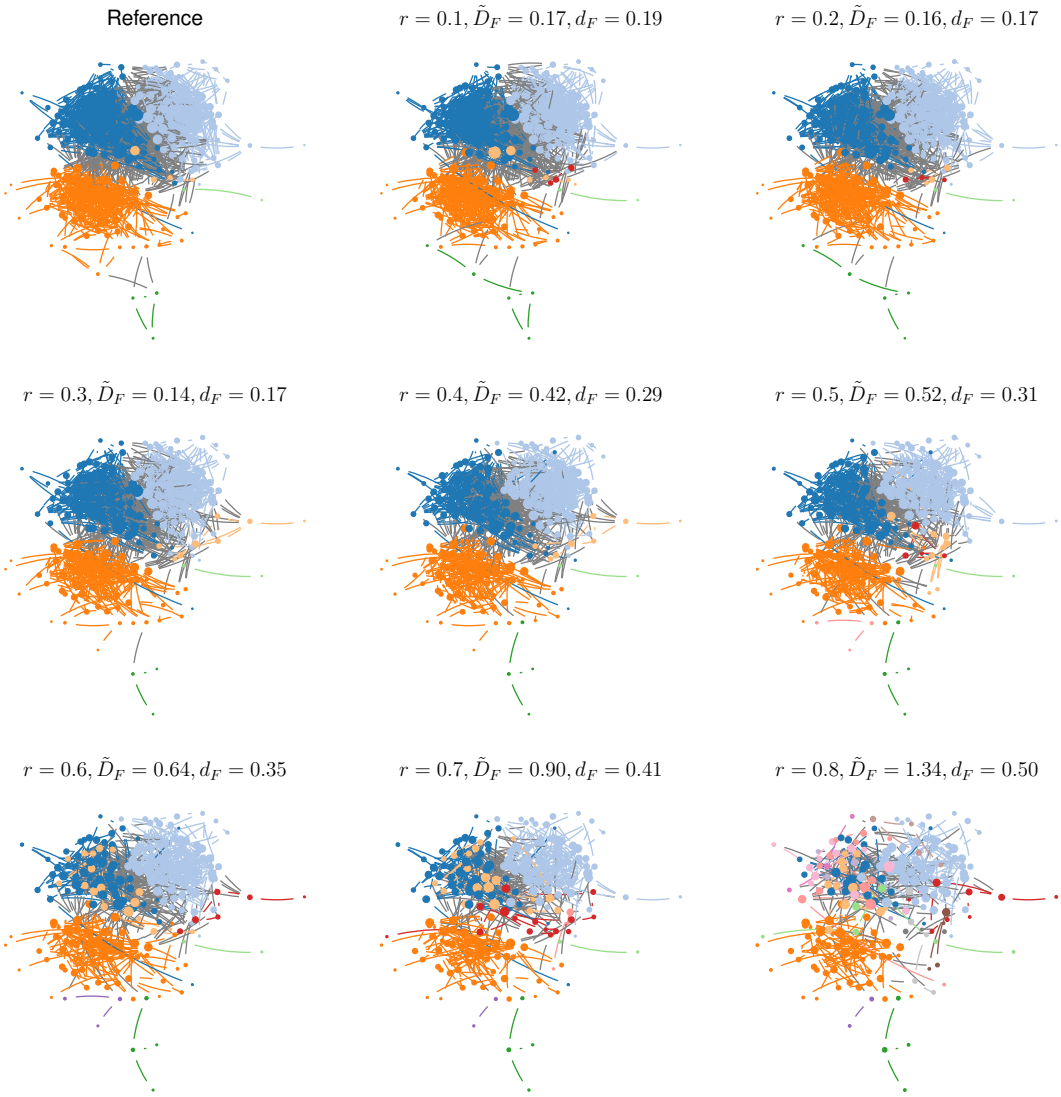


**Figure 14:** The jazz network with 198 nodes. For different $r$-fractions of removed links, we show the resulting partitions with normalised flow divergence values $\tilde{D}_F$ and flow distance values $d_F$.

## H Toy Network

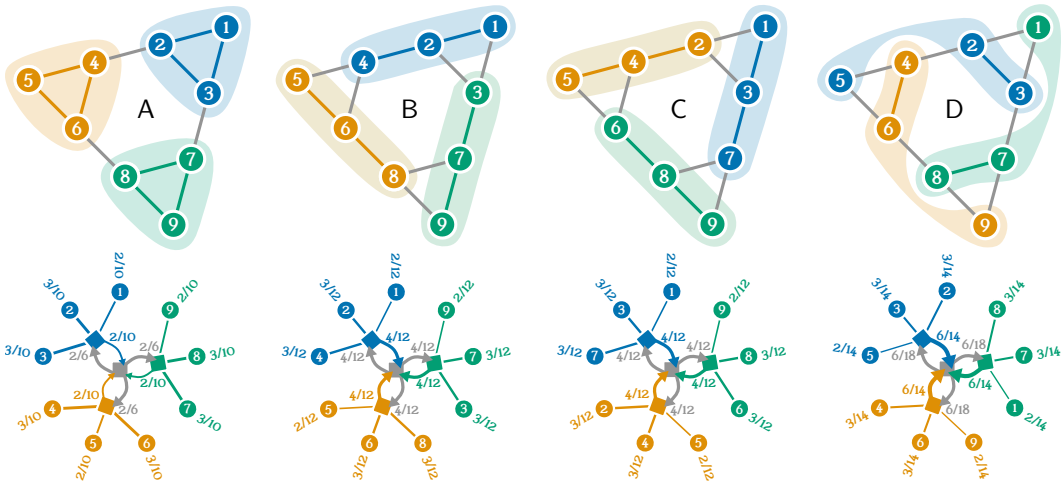


**Figure 15:** The same partitions as in Figure 1, together with their partition trees.

Here, we consider our motivational toy example in more detail and explain why flow divergence considers partition D to be more similar to A than B and C. Figure 15 shows four partitions of the same network, together with their maps drawn as radial trees and annotated with transition rates. We consider the individual nodes' contribution to flow divergences instead of computing the outer sum of

$$\tilde{D}_F\left(\mathsf{A}\,||\,\mathsf{B}\right) = \sum_{u\in V} p_u \sum_{v\in V} t^{\mathsf{A}}_{uv} \log_2 \frac{t^{\mathsf{A}}_{uv}}{t^{\mathsf{B}}_{uv}} = \sum_{u\in V} p_u \sum_{v\in V} t^{\mathsf{A}}_{uv} \left[\log_2 \frac{\mathrm{mapsim}\left(\mathsf{A}, u, v\right)}{\mathrm{mapsim}\left(\mathsf{B}, u, v\right)} - \lambda^{\mathsf{A}}_u + \lambda^{\mathsf{B}}_u\right], \tag{85}$$

where $\lambda^{\mathsf{A}}_u = \log_2\left(\sum_{v\in V} \mathrm{mapsim}\left(\mathsf{A}, u, v\right)\right)$ and $\lambda^{\mathsf{B}}_u = \log_2\left(\sum_{v\in V} \mathrm{mapsim}\left(\mathsf{B}, u, v\right)\right)$.

The values in Table 3 reveal that D is more similar to A because the modules in A and D overlap in the higher-degree nodes, which carry higher weight due to their higher flow. In contrast, the modules in B and C overlap in one higher-degree and one lower-degree node with the modules in A.

**Table 3:** Rounded normalised flow divergence scores on a per-node basis between the partitions shown in Figure 15. We fix A as the reference partition.

| | Node | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| B | 0.08 | 0.13 | 0.21 | 0.21 | 0.08 | 0.13 | 0.13 | 0.21 | 0.08 |
| C | 0.08 | 0.21 | 0.13 | 0.13 | 0.08 | 0.21 | 0.21 | 0.13 | 0.08 |
| D | 0.15 | 0.08 | 0.08 | 0.08 | 0.15 | 0.08 | 0.08 | 0.08 | 0.15 |

# I Experiments on directed Lancichinetti–Fortunato–Radicchi networks

Here, we show additional results for experiments on directed Lancichinetti–Fortunato–Radicchi networks, following the same setup as used in Section 4.2. We ensured that the generated LFR networks are strongly connected, meaning there is a path between every pair of nodes. However, removing links can create networks that are no longer strongly connected, meaning there is no path between some pairs of nodes anymore. For computing flow divergence and flow distance values, we ignore such node pairs, and show in the plot the fraction of pairs that we had to ignore.

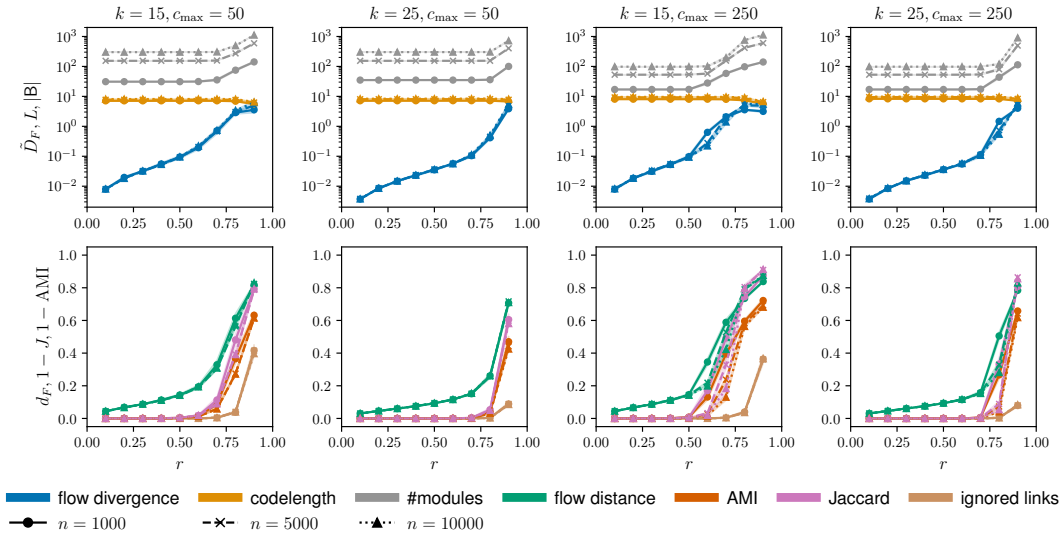


**Figure 16:** Results on directed Lancichinetti–Fortunato–Radicchi graphs with planted communities. The top panel shows normalised flow divergence values $\tilde{D}_F$ and codelength $L$ in bits, and the number of communities $|\mathsf{B}|$ detected by Infomap for different $r$-fractions of removed links on a log scale. The bottom panel shows the corresponding flow distance $d_F$, Jaccard index J, and AMI values on a linear scale. For more intuitive comparisons, we plot $1 - \mathrm{J}$ and $1 - \mathrm{AMI}$.

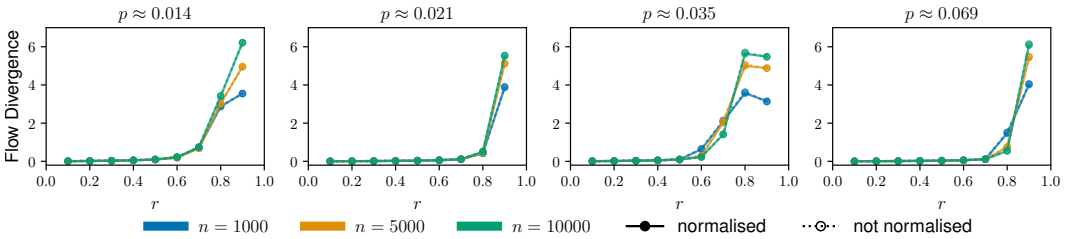


**Figure 17:** Normalised and non-normalised flow divergence values for directed LFR networks with the setup described in Section 4.2. Both versions of flow divergence return virtually the same results.

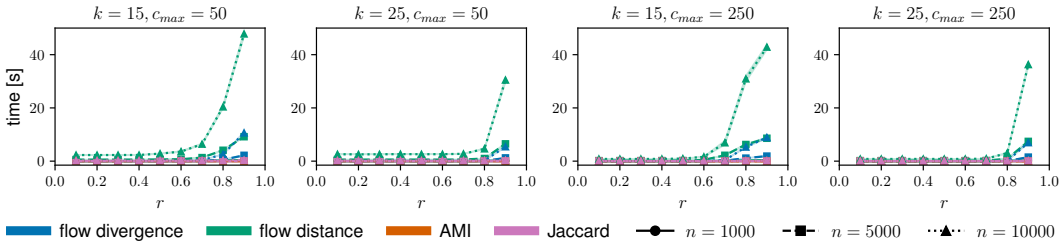


**Figure 18:** Empirical runtime for computing flow divergence, flow distance, AMI, and Jaccard index values for hierarchical partitions of LFR networks with the same setup as described in Section 4.2.

## J Experiments on hierarchical Lancichinetti–Fortunato–Radicchi networks

Here, we show additional results for experiments on hierarchical Lancichinetti–Fortunato–Radicchi networks with two levels of communities, following the same setup as used in Section 4.2.

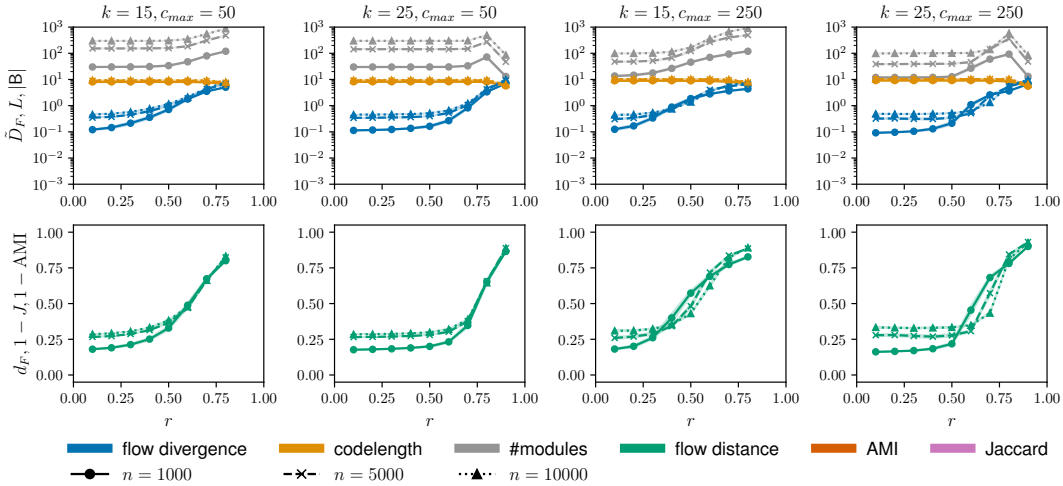


**Figure 19:** Results on hierarchical Lancichinetti–Fortunato–Radicchi graphs with planted communities. The top panel shows normalised flow divergence values $\tilde{D}_F$ and codelength $L$ in bits, and the number of communities $|\mathsf{B}|$ detected by Infomap for different $r$-fractions of removed links on a log scale. The bottom panel shows the corresponding flow distance $d_F$, Jaccard index J, and AMI values on a linear scale. For more intuitive comparisons, we plot $1 - \mathrm{J}$ and $1 - \mathrm{AMI}$.

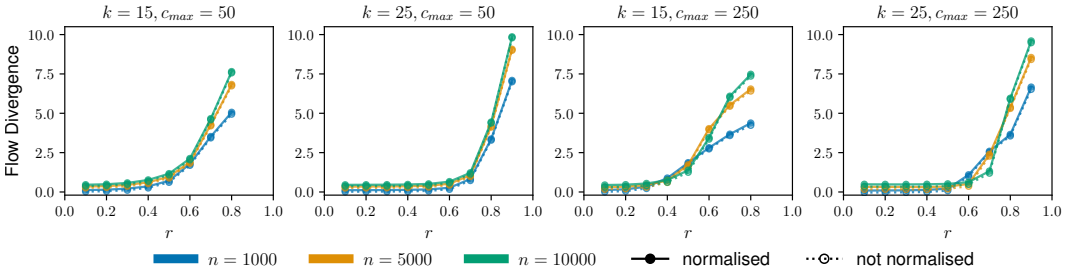


**Figure 20:** Normalised and non-normalised flow divergence for hierarchical LFR networks with the setup described in Section 4.2. Both versions of flow divergence return virtually the same results.

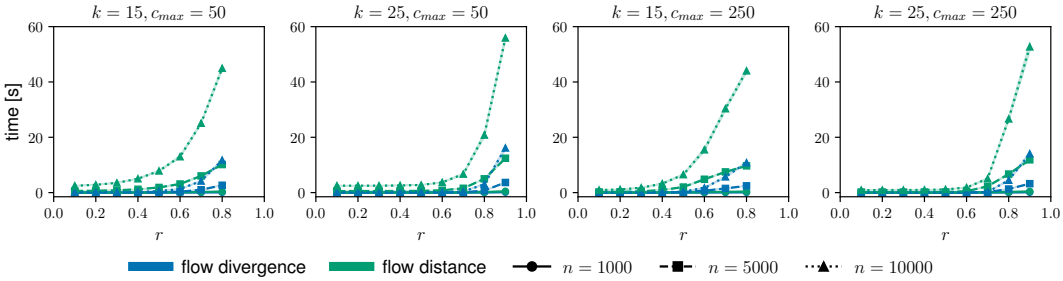


**Figure 21:** Empirical runtime for computing flow divergence and flow distance values for hierarchical partitions of LFR networks with the same setup as described in Section 4.2.

## K Empirical Runtime

Here, we report the empirical wall-clock runtimes for computing flow divergence, flow distance, AMI, and Jaccard index scores. We ran the experiments on an Intel Core Ultra 7 165H CPU. To compute flow distance, we used a single-threaded implementation, however, our Python library also contains a parallelised version. Our experimental results are in line with our theoretical results showing that computing flow divergence requires time $\mathcal{O}\left(m^2\right)$ while computing flow distance requires time $\mathcal{O}\left(d_m \cdot m \cdot n\right)$, where $n$ is the number of nodes, $m$ is the number of modules, and $d_m$ is the average depth at which the modules are located in the network's community structure.

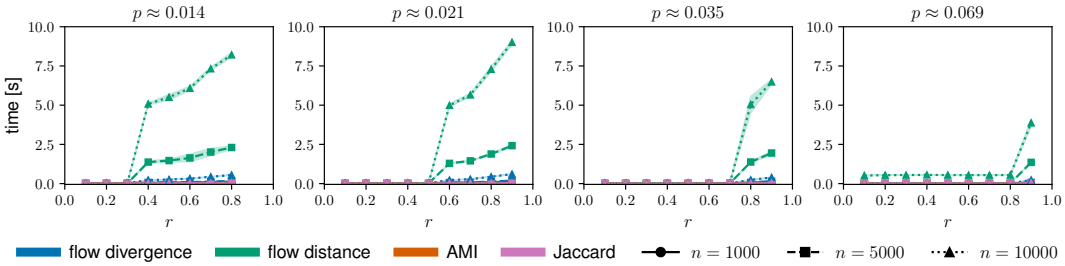


**Figure 22: ER Graphs.** Wall-clock runtimes for computing partition similarity values for partitions on ER random graphs with different sizes and link probabilities. The ER networks are the same ones used in the main text in Section 4.2.

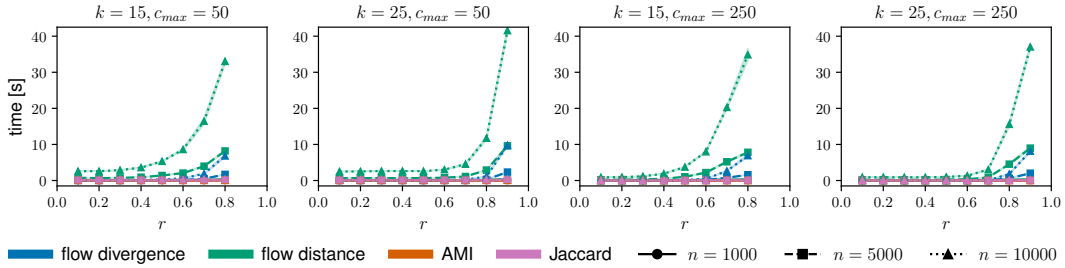


**Figure 23: LFR Graphs.** Wall-clock runtimes for computing partition similarity values for partitions on undirected LFR networks with planted communities and different sizes and average node degrees. The LFR networks are the same ones used in the main text in Section 4.2.

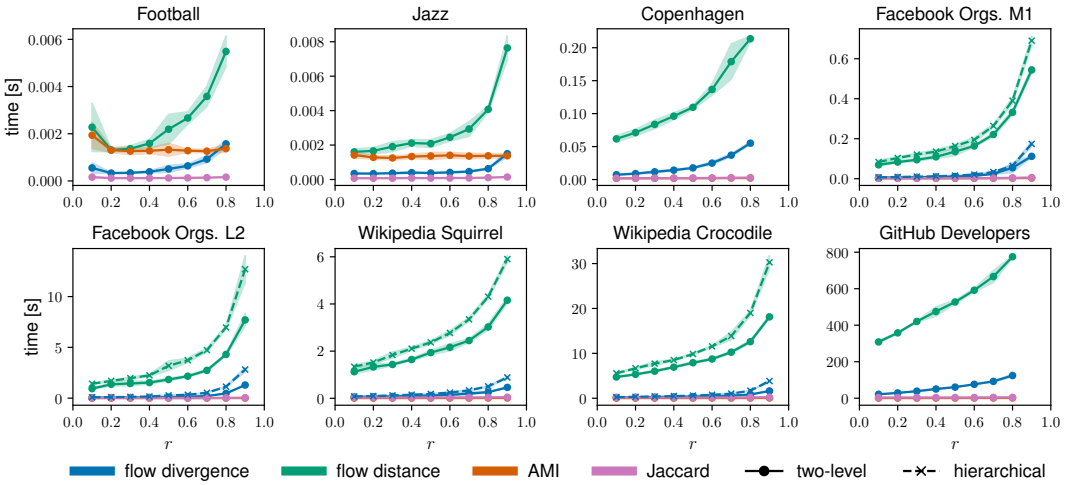


**Figure 24: Empirical Networks.** Wall-clock runtimes for computing partition similarity values for partitions on empirical networks. The networks are the same ones used in the main text in Section 4.3. Note the different scales on the y-axes due to different network sizes.

## L Experiments with Communities from Louvain Modularity Maximisation

Here, we repeat the experiments from the main text but use modularity maximisation via the Louvain algorithm [46], implemented in `networkx`, to detect communities. As reference partitions, we use again the partition that assigns all nodes to the same community for the ER networks, and the planted ground-truth partitions for the LFR networks. For the empirical networks, we use the partitions detected with Infomap before removing links to facilitate comparability with the results from the main text.

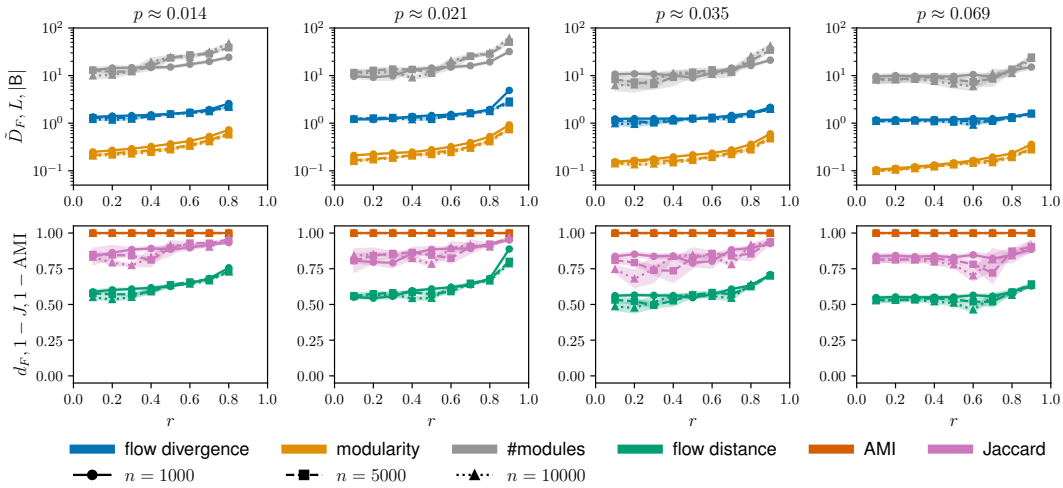


**Figure 25:** Results on ER random graphs with link probability $p \in \left\{\frac{2\ln n}{n}, \frac{3\ln n}{n}, \frac{5\ln n}{n}, \frac{10\ln n}{n}\right\}$. The top panel shows normalised flow divergence $\tilde{D}_F$ and modularity $Q$, and the number of communities $|\mathsf{B}|$ detected by Louvain for different $r$-fractions of removed links on a logarithmic scale. The bottom panel shows the corresponding flow distance $d_F$, Jaccard index J, and AMI values on a linear scale. For more intuitive comparisons, we plot $1 - \mathrm{J}$ and $1 - \mathrm{AMI}$.

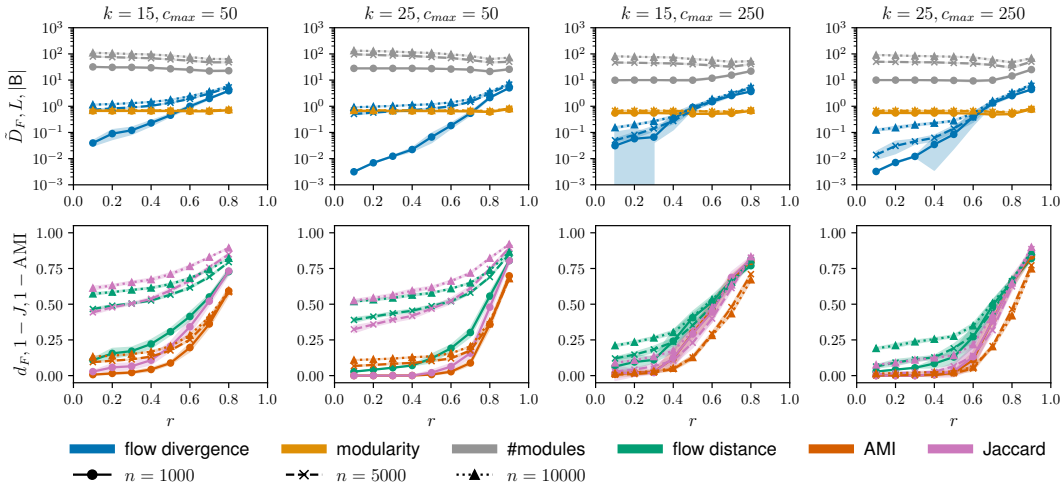


**Figure 26:** Results on undirected Lancichinetti–Fortunato–Radicchi graphs with non-hierarchical planted communities. The top panel shows normalised flow divergence $\tilde{D}_F$ and modularity $Q$, and the number of communities $|\mathsf{B}|$ detected by Louvain for different $r$-fractions of removed links on a log scale. The bottom panel shows the corresponding flow distance $d_F$, Jaccard index J, and AMI values on a linear scale. For more intuitive comparisons, we plot $1 - \mathrm{J}$ and $1 - \mathrm{AMI}$.

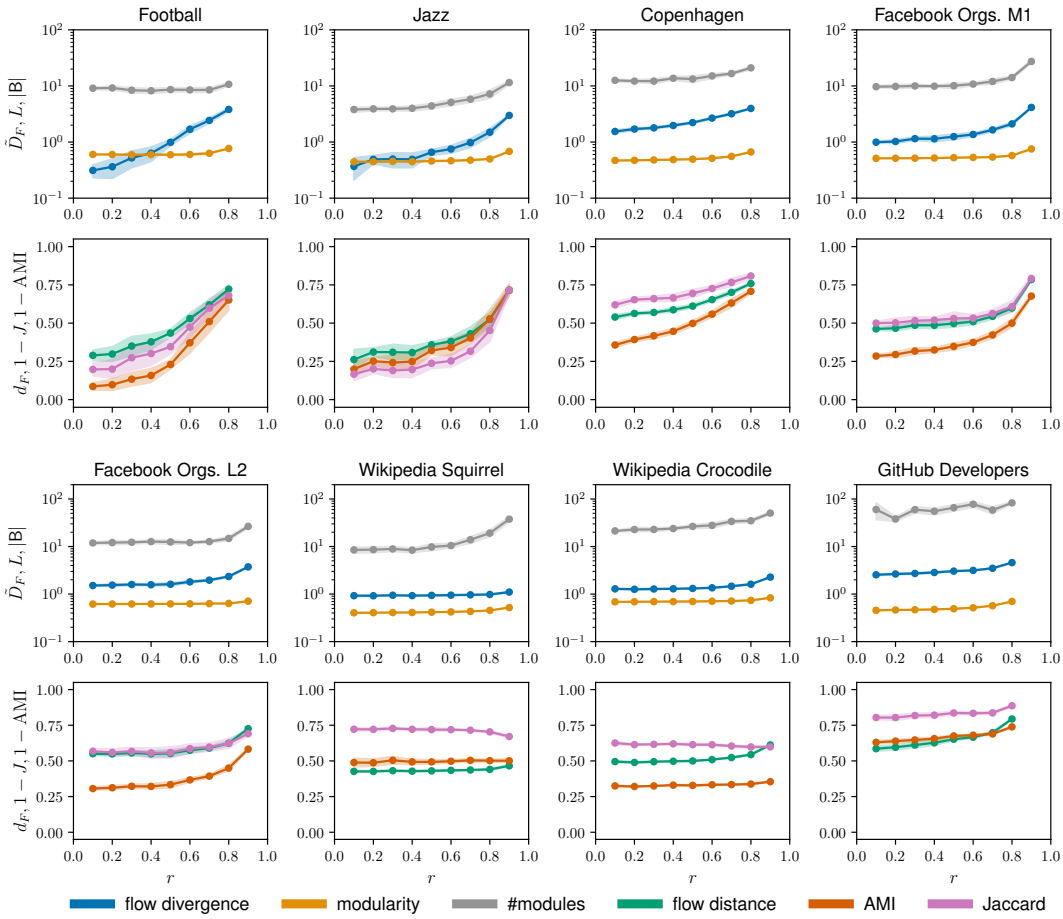


**Figure 27:** Results on real networks. We show normalised flow divergence $\tilde{D}_F$ and modularity $Q$, and the number of communities $|\mathsf{B}|$ detected by Louvain for different $r$-fractions of removed links on a log scale, and the corresponding flow distance $d_F$, Jaccard index J, and AMI values on a linear scale. For more intuitive comparisons, we plot $1 - \mathrm{J}$ and $1 - \mathrm{AMI}$.

## M Python-like Pseudo-Code for Flow Divergence and Flow Distance

```
def flow_divergence( A          : Partition # the reference partition
                   , B          : Partition # an alternative partition
                   , normalise  : bool
                   ) -> float:

  res : float = 0.0

  # for all source nodes
  for u in A.nodes:
    # node u's visit rate
    p_u : float = A.get_visit_rate(u)

    # sum of transition rates from u to all nodes in A
    sum_A : float = sum([A.transition_rate(u,v) for v in A.nodes])

    # set normalisation factors
    norm_A : float = 1.0
    norm_B : float = 1.0
    if normalise:
      norm_A = sum_A
      norm_B = sum([B.transition_rate(u,v) for v in B.nodes])

    # sum up node-wise divergence in coding costs for going
    # from node u to v
    for v in A.nodes:
      coding_rate_A : float = A.transition_rate(u,v) / norm_A
      coding_rate_B : float = B.transition_rate(u,v) / norm_B

      # probability that a random walker steps from u to v,
      # given partition A
      transition_prob : float = p_u * A.transition_rate(u,v)/sum_A

      res += transition_prob * log2(coding_rate_A / coding_rate_B)

  return res
```

**Listing 1:** Python-like pseudo-code for a naive approach to compute flow divergence with a double loop over all node pairs.

```python
def flow_distance( A         : Partition # the reference partition
                 , B         : Partition # an alternative partition
                 , normalise : bool
                 ) -> float:
  res : float = 0.0

  # for all source nodes
  for u in A.nodes:
    # node u's visit rate in both A and B, which is often
    # the same, but can be different when B is obtained
    # after removing links from the graph
    p_u_A : float = A.get_visit_rate(u)
    p_u_B : float = B.get_visit_rate(u)

    # sum of transition rates from u to all nodes in A and B
    sum_A : float = sum([A.transition_rate(u,v) for v in A.nodes])z
    sum_B : float = sum([B.transition_rate(u,v) for v in A.nodes])

    # set normalisation factors
    norm_A  : float = 1.0
    norm_B  : float = 1.0
    norm_AB : float = 1.0
    if normalise:
      norm_A = sum_A
      norm_B = sum_B

      # sum of the "average" transition rates from u to all nodes
      norm_AB = sum([ 0.5 * A.transition_rate(u,v)
                    + 0.5 * B.transition_rate(u,v)
                      for u in A.nodes
                    ])

    # sum up node-wise distance
    for v in A.nodes:
      coding_rate_A  : float = A.transition_rate(u,v) / norm_A
      coding_rate_B  : float = B.transition_rate(u,v) / norm_B
      coding_rate_AB : float = ( 0.5 * A.transition_rate(u,v)
                               + 0.5 * B.transition_rate(u,v)
                               ) / norm_AB

      transition_prob_A : float = p_u_A * A.transition_rate(u,v)/sum_A
      transition_prob_B : float = p_u_B * B.transition_rate(u,v)/sum_B

      # node-wise divergence between partitions A and B and their
      # mixture partition AB
      a = transition_prob_A * log2(coding_rate_A / coding_rate_AB)
      b = transition_prob_B * log2(coding_rate_B / coding_rate_AB)

      # following Jensen-Shannon distance
      res += 0.5 * (a + b)

  return sqrt(res)
```

**Listing 2:** Python-like pseudo-code for a naive approach to compute flow distance with a double loop over all node pairs.