# Generative benchmark models for mesoscale structure in multilayer networks

Marya Bazzi,[1,*] Lucas G. S. Jeub,[1,2,*] Alex Arenas,[3] Sam D. Howison,[1,4] and Mason A. Porter[1,5,6]

[1] *Oxford Centre for Industrial and Applied Mathematics, Mathematical Institute, University of Oxford, Oxford OX2 6GG, United Kingdom*
[2] *Center for Complex Networks and Systems Research, School of Informatics and Computing, Indiana University, Bloomington, Indiana 47408, USA*
[3] *Departament d'Enginyeria Informàtica i Matemàtiques, Universitat Rovira i Virgili, 43007 Tarragona, Spain*
[4] *Oxford-Man Institute of Quantitative Finance, University of Oxford, Oxford OX2 6ED, United Kingdom*
[5] *CABDyN Complexity Centre, University of Oxford, Oxford OX1 1HP, United Kingdom*
[6] *Department of Mathematics, University of California, Los Angeles, Los Angeles, California 90095, USA*



Multilayer networks allow one to represent diverse and interdependent connectivity patterns — e.g., time-dependence, multiple subsystems, or both — that arise in many applications and which are difficult or awkward to incorporate into standard network representations. In the study of multilayer networks, it is important to investigate "mesoscale" (i.e., intermediate-scale) structures, such as dense sets of nodes known as "communities" that are connected sparsely to each other, to discover network features that are not apparent at the microscale or the macroscale. A variety of methods and algorithms are available to identify communities in multilayer networks, but they differ in their definitions and/or assumptions of what constitutes a community, and many scalable algorithms provide approximate solutions with little or no theoretical guarantee on the quality of their approximations. Consequently, it is crucial to develop generative models of networks to use as a common test of community-detection tools. In the present paper, we develop a family of benchmarks for detecting mesoscale structures in multilayer networks by introducing a generative model that can explicitly incorporate dependency structure between layers. Our benchmark provides a standardized set of null models, together with an associated set of principles from which they are derived, for studies of mesoscale structures in multilayer networks. We discuss the parameters and properties of our generative model, and we illustrate its use by comparing a variety of community-detection methods.

## I. INTRODUCTION

Many physical, technological, biological, financial, and social systems can be modeled as networks, which in their simplest form are represented as graphs [1]. A graph consists of a set of nodes that represent entities and a set of edges between pairs of nodes that represent interactions between those entities. One can consider either unweighted graphs or weighted graphs, in which each edge has a weight that quantifies the strength of the associated interaction. Edges can also have a direction to represent asymmetric interactions or a sign to differentiate between positive and negative interactions.

Given a network representation of a system, it is often useful to apply a coarse-graining technique to investigate features that lie between those at the "microscale" (e.g., nodes, pairwise interactions between nodes, or local properties of nodes) and those at the "macroscale" (e.g., total edge weight, degree distribution, and mean clustering coefficient) [2, 3]. One thereby studies "mesoscale" features such as community structure [2], core–periphery structure [4], role structure [5], or others. To make our discussions concrete, we focus primarily on community structure in this paper. We then briefly explain how our results also apply to other types of mesoscale structures in Section VI.

Loosely speaking, a community in a network is a set of nodes that are "more densely" connected to each other than they are to nodes in the rest of the network [2, 3, 6, 7]. Communities thus represent an "assortative structure" — i.e., intracommunity edges are more likely than intercommunity edges — between sets of nodes in a network. Most scholars agree that a "good community" should be a set of nodes that are 'surprisingly well-connected' in some sense, but what one actually means by 'surprising' and 'well-connected' can be application-dependent (and sometimes appears to be somewhat subjective). In many cases, a precise definition of "community" ultimately depends on the method and algorithm that one uses to detect them. Myriad methods have been developed to algorithmically detect communities, and efforts at community detection have led to insights in applications such as granular force networks [8], committee and voting networks in political science [9, 10], friendship networks at universities and schools [11, 12], protein interaction networks in biology [13, 14], brain and behavioral networks in neuroscience [15, 16], human communication networks [17, 18], and more.

A popular approach to community detection is the optimization of a quality function, such as modularity [10, 19, 20], stability [21–23], Infomap and its variants [24, 25], and likelihood functions derived from

* Both authors contributed equally.

"stochastic block models" (SBMs) [26–30], which are models for partitioning a network into blocks of nodes with statistically homogeneous connectivity patterns. Different quality functions are motivated by different interpretations of "community" (e.g., as bottlenecks of dynamical processes [22, 23, 25]), and different SBMs make different assumptions on the underlying network structure [27–29]. Further complications arise from the fact that community-assignment problems for many notions of community structure — including the most prominent methods — cannot be solved exactly in polynomial time (unless P = NP) [3, 31, 32]. Together with the need for methods to scale to be usable for very large networks, this necessitates the use of computational heuristics that only find approximate solutions, and there is often little theoretical understanding of how closely such approximate solutions resemble an optimal solution.

Benchmark networks with known structural properties are important for (1) analyzing and comparing the performance of different community-detection methods and algorithms and (2) determining which one(s) are most appropriate in a given situation. The ill-defined nature of community detection makes it particularly crucial to develop good benchmark networks. We propose to do this using generative models. In many benchmark networks, one plants a partition (i.e., an assignment of nodes to communities) of a network into well-separated communities, and one thereby imposes a so-called "ground truth" (should one wish to use such a notion) that a properly deployed community-detection method ought to be able to find [33]. However, another complication arises: there is a "detectability limit" for communities, as one can plant partitions that, under suitable conditions, cannot subsequently be detected algorithmically even though they exist by construction [34–36].

For a single-layer network (i.e., a "monolayer network"), which is the standard type of network and is represented mathematically as a graph, many different types of benchmark networks have been developed to capture different aspects of community structure. A benchmark that was employed in early studies of community structure is the "planted-$l$-partition model" [37], which was used in [19]. The most popular family of benchmark networks are the Lancichinetti–Fortunato–Radicchi (LFR) networks [38], which were generalized subsequently in [39] to be able to generate weighted and directed networks with either disjoint or overlapping communities. To attempt to capture the heterogeneity observed in many real networks [1], LFR networks have power-law degree distributions and community-size distributions. A similar benchmark, based on a degree-corrected SBM, was suggested in [27]. In Section IV, we point out some advantages of this benchmark over LFR networks when used to generate multilayer networks. In our numerical experiments in Section V, we use a slight variant of the degree-corrected SBM benchmark in which we impose the additional constraint that there can be neither self-edges nor multi-edges (see Algorithm 2). There have also been some efforts towards highlighting unrealistic features of LFR networks [40] and towards developing more realistic benchmarks [41]. Other benchmarks have also been developed to capture particular aspects of some networks, such as being embedded in or influenced by space [42].

In many applications, monolayer networks are insufficient for capturing the intricacies of connectivity patterns between entities. For example, this arises in both temporal [20, 43–45] and multiplex networks [46–50], which we represent as multilayer networks [51, 52]. Our motivation for considering a single multilayer network instead of several networks independently is that connectivity patterns in different layers often depend on each other in some way. For example, the connectivity patterns in somebody's Facebook friendship network today are not independent either of the connectivity patterns in that person's Facebook friendship network last year (temporal) or of the connectivity patterns in that person's Twitter follower/following network today (multiplex). Data sets that have multilayer structures are becoming increasingly available, and several methods to detect communities in multilayer networks have now been developed [10, 24, 28, 29, 51, 53, 54].

Existing benchmark networks for multilayer community detection assume either a temporal structure [35, 42, 55–58] or a simplified multiplex structure with independent blocks of layers in which layers in the same block have identical community structures [24]. Existing multilayer SBMs make different assumptions on the underlying network structure. For example, [59–61] used the same monolayer SBM for each layer, and [28] used independent monolayer SBMs (one for each set of "similar" layers). We also note Ref. [62], which developed several multilayer random-graph models (including an SBM) to use as null models. Much of the theory and examples in [29] entail fitting a monolayer SBM with the same partition (but in which other model parameters may differ) to each layer, although the author pointed out that his framework allows more flexibility if one allows overlapping communities in the model [29, 63]. Importantly, none of the aforementioned SBMs include an explicit parameter to capture dependency structure between layers. Therefore, although one may be able to use those SBMs to recover mesoscale structure that fits the employed model, one cannot use them to generate multilayer networks with a prescribed interlayer dependency structure other than the specific one(s) that are built into the model.

In this paper, we propose a method for generating random multilayer partitions with arbitrary interlayer dependency structures. Given a user-specified interlayer dependency structure, we define a Markov chain on the space of multilayer partitions and sample a multilayer partition from its stationary distribution. (See Section III for our generative model of a multilayer partition.) Our sampling procedure includes the temporal structure in [35, 42] and simplified multiplex structure

in [24] as special cases. Broadly speaking, our generation process is threefold: (1) a user specifies the interlayer dependency structure; (2) we then sample a multilayer partition of the set of nodes that satisfies the specified dependency structure; and (3) finally, we sample edges between nodes in a way that is consistent with the planted multilayer partition. We explain steps (1) and (2) in Section III, and we explain step (3) in Section IV. We treat the process of generating a multilayer partition separately from the process of generating edges for a given multilayer partition. That is, we carry out steps (2) and (3) successively (and not in parallel). This yields a modular approach for generating benchmark multilayer networks because one can modify steps 2 and 3 independently.

In our numerical experiments, we generate intralayer edges (i.e., edges within layers) using degree-corrected SBMs for each layer. Our construction produces a multilayer network with no edges between layers but in which the connectivity patterns in different layers depend on each other (i.e., they are "interdependent"). This is the most commonly studied type of multilayer network (see Table 2 of [51]). One can also use our multilayer formulation of the degree-corrected SBM in Section IV to generate interlayer edges (i.e., edges between layers).

One can combine our approach for generating multilayer partitions with different network generating models that capture various important features. For example, one can use an SBM to generate intralayer edges and/or interlayer edges, or one can replace the degree-corrected SBM in Section IV with any other monolayer network model with a planted partition (e.g., other variants of SBMs [26, 32] and models for spatially embedded networks [42, 64]). In all of these examples, dependencies between connectivities in different layers result only from the planted multilayer partition. (See [56] for a different approach, in which dependencies between different layers are introduced via the network generation process.) It is also possible to modify our network-generation process to introduce additional interdependencies between connectivity patterns in different layers beyond that induced by planted mesoscale structure (see Section VI). Our generative model is a very general one (and, as discussed above, we have constructed it to easily admit additional salient features of empirical multilayer networks), and we illustrate its use with a few important special cases. Note, however, that it is not the goal of our paper to cover as many special cases as possible.

Along with this paper, we include publicly available code [65] that users can modify to readily incorporate different types of interlayer dependency structures (see Section III A), null distributions (see Section III C), and monolayer network models with a planted partition (see Section IV).

The rest of the paper is organized as follows. We start in Section II with an overview of the definitions and notation that we use. In Section III, we explain in detail how we generate a multilayer partition with a specified dependency structure between layers. We give some properties of a sampled partition and discuss the effect of some parameter choices on the resulting partition. In Section IV, we describe how we generate edges that are consistent with the planted partition. The modular nature of our approach enables one to generate edges using any monolayer network model with a planted partition. In Section V, we describe numerical experiments to compare the performance of various community-detection methods. In Section VI, we provide a summary of our main results and we outline how one can incorporate more realistic features into the generative model (e.g., change points [29, 66], mesoscale structures other than community structure, and others) and discuss directions for future work.

## II. MULTILAYER NETWORKS: PRELIMINARIES

The simplest type of network is a graph $G = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{1, \ldots, n\}$ is a set of nodes and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is a set of edges. Using a graph, one can encode the presence or absence of connections (the edges) between entities (the nodes). However, in many situations, one would like to include more detailed information about connections between entities. A common extension is to allow edges to have a weight, which one can use to represent the strength of a connection. We represent edge weights using a *weight function* $w : \mathcal{E} \to \mathbb{R}$ which assigns a weight to each edge.

As we mentioned in Section I, one can further extend the network framework to represent different "aspects" of connections between entities, such as connections at different points in time or different types of connections that occur simultaneously. We adopt a "multilayer network" framework [51, 67] to represent such connections. In a multilayer network, a node is present in a variety of different "states", where each state is further characterized by a variety of different aspects. In this setting, edges join "state nodes", each of which is the representation of a given node in a particular state. One can think of the aspects as features that one needs to specify to identify the state of a node. In other words, a state is a collection of exactly one element from each aspect. For convenience, we introduce a mapping such that we assign an integer label to each element of an aspect. That is, the $l_a$ elements of the $a^{\text{th}}$ aspect are mapped to the elements of a set $\{1, \ldots, l_a\}$ of integers. Aspects can be unordered (e.g., social-media platform) or ordered (e.g., time). For an ordered aspect, we require that the mapping respects the order of the aspect (e.g., $t_i \to i$ for time, where $t_1 \leq \ldots \leq t_{l_a}$ is a set of discrete time points). A multilayer network can include an arbitrary number of ordered aspects and an arbitrary number of unordered aspects, and one can generalize these ideas further (e.g., by introducing a time horizon) [51].

To illustrate the above ideas, consider a hypothetical

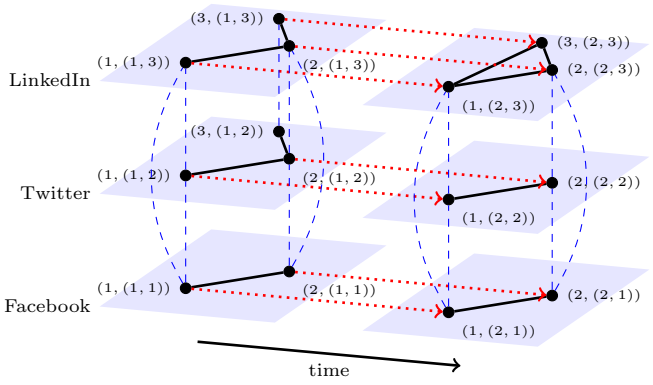



FIG. 1. Toy example of a multilayer network with three nodes and two aspects. (Although we use a hypothetical example with three aspects in Section II, we omit the third aspect from this figure for clarity.) We represent undirected intralayer edges using solid black lines, directed interlayer edges using dotted red lines, and undirected interlayer edges using dashed blue arcs. The first aspect is ordered and corresponds to time. It contains two time points (arising either from different instances or from aggregations over different time intervals), which we label by 1 and 2. That is, $\mathcal{L}_1 = \{1, 2\}$. The second aspect is unordered and represents social-media platform. It contains three elements: Facebook (which we label with the integer 1), Twitter (which we label with the integer 2), and LinkedIn (which we label with the integer 3). That is, $\mathcal{L}_2 = \{1, 2, 3\}$. A state node thus takes the form $(i, (\alpha_1, \alpha_2))$, with $i \in \{1, 2, 3\}$, $\alpha_1 \in \{1, 2\}$, and $\alpha_2 \in \{1, 2, 3\}$. The total number of states (i.e., layers) is $l = |\mathcal{L}_1||\mathcal{L}_2| = 6$. All interlayer edges are diagonal (i.e., between state nodes that correspond to the same physical node). Interlayer edges between layers at different times for a given social media platform are ordinal (i.e., between layers with successive time labels) and directed. Interlayer edges between layers at the same time and corresponding to different social media platforms are categorical (i.e., between all pairs of layers) and undirected. For example, information can flow from node 1 in Facebook at time 1 to node 1 in Facebook at time 2. Similarly, information can flow between node 1 in Facebook at time 1 and node 1 in Twitter at time 1. One can also define interlayer edges differently—for example, by introducing edges between state nodes at successive times and different social media platforms (e.g., $(1, (1, 1)) \to (1, (2, 2))$ similarly to the approach in [45].

social network in which we observe different types of connections ('friendship' and 'following') on different platforms ('Facebook', 'Twitter', and 'LinkedIn') between the same set of people at different points in time. In our example, we have three aspects: type of connection, social-media platform, and time. The first aspect is unordered and consists of two elements: 'friendship' (which we map to the integer 1) and 'following' (which we map to the integer 2). The second aspect is also unordered and consists of three elements: 'Facebook' (which we map to the integer 1), 'Twitter' (which we map to the integer 2), and 'LinkedIn' (which we map to the integer 3). The third aspect is ordered and consists of as many elements as there are time points or time intervals. If we assume that the time resolution is daily and spans the year 2010, an example of a state is the triple ('following', 'Facebook', '01-Jan-2010') or equivalently $(2, 1, 1)$.

We refer to the set of all state nodes that represent a given entity as a "physical node" and the set of all state nodes in a given state as a "layer". Note that there is a bijective mapping between nodes and physical nodes and a bijective mapping between states and layers. One can have connections between nodes in the same state (i.e., intralayer edges) and nodes in different states (i.e., interlayer edges). An example of an intralayer edge is (1, ('following', 'Twitter', '01-Jan-2010')) $\to$ (2, ('following', 'Twitter', '01-Jan-2010')), indicating that entity 1 was following entity 2 on Twitter on 1 January 2010. An example of an interlayer edge is (1, ('following', 'Twitter', '01-Jan-2010')) $\to$ (1, ('following', 'Twitter', '02-Jan-2010')), indicating that information can flow from entity 1 on 1 January 2010 to entity 1 on 2 January 2010.

More formally, following the notation of [51], we consider a general *multilayer network* $M = (\mathcal{V}_M, \mathcal{E}_M, \mathcal{V}, \mathcal{L})$ with $n = |\mathcal{V}|$ *nodes* (or *physical nodes*) and $l = |\mathcal{L}|$ *states* (or *layers*). For ease of writing, we use the same notation for nodes and physical nodes, and we also use the same notation for states and layers. We use $d$ to denote the number of aspects and use $\mathcal{L}_a = \{1, \ldots, l_a\}$ to denote the labels of the $a^{\text{th}}$ *aspect* (where $a \in \{1, \ldots, d\}$). We use $\mathcal{O}$ to denote the set of *ordered aspects* and $\mathcal{U}$ to denote the set of *unordered aspects*. The set $\mathcal{L} = \mathcal{L}_1 \times \ldots \times \mathcal{L}_d$ of states is the Cartesian product of the aspects, where a state $\boldsymbol{\alpha} \in \mathcal{L}$ is an integer vector of length $d$ and each entry specifies an element of the corresponding aspect. Note that $l = |\mathcal{L}| = \prod_{a=1}^{d} l_a$.

We use $(i, \boldsymbol{\alpha}) \in \mathcal{V}_M \subseteq \mathcal{V} \times \mathcal{L}$ to denote the *state node* (i.e., "node-layer tuple" [51]) representing node $i \in \mathcal{V}$ in state $\boldsymbol{\alpha} \in \mathcal{L}$. We only include a state node in $\mathcal{V}_M$ if the corresponding node exists in that state. The edges $\mathcal{E}_M \subseteq \mathcal{V}_M \times \mathcal{V}_M$ in a multilayer network are between state nodes, where we use $((i, \boldsymbol{\alpha}), (j, \boldsymbol{\beta}))$ to denote a directed edge from $(i, \boldsymbol{\alpha})$ to $(j, \boldsymbol{\beta})$. For two state nodes, $(i, \boldsymbol{\alpha})$ and $(j, \boldsymbol{\beta})$, connected by a directed edge $((i, \boldsymbol{\alpha}), (j, \boldsymbol{\beta})) \in \mathcal{E}_M$, we say that $(i, \boldsymbol{\alpha})$ is an *in-neighbor* of $(j, \boldsymbol{\beta})$ and $(j, \boldsymbol{\beta})$ is an *out-neighbor* of $(i, \boldsymbol{\alpha})$. We categorize the edges into *intralayer edges* $\mathcal{E}_L$, which have the form $((i, \boldsymbol{\alpha}), (j, \boldsymbol{\alpha}))$ and link entities $i$ and $j$ in the same state $\boldsymbol{\alpha}$, and *interlayer* (or *coupling*) *edges* $\mathcal{E}_C$, which have the form $((i, \boldsymbol{\alpha}), (j, \boldsymbol{\beta}))$ for $\boldsymbol{\alpha} \neq \boldsymbol{\beta}$. We thereby decompose the edge set $\mathcal{E}_M = \mathcal{E}_L \cup \mathcal{E}_C$.

We define a weighted multilayer network by introducing a weight function $w : \mathcal{E}_M \to \mathbb{R}$ (analogous to the weight function for weighted monolayer networks), which encodes the edge weights within and between layers. For an unweighted multilayer network, we define $w(e) = 1$ for all $e \in \mathcal{E}_M$ to simplify our discussion. We encode the connectivity of a multilayer network using an *adjacency tensor* $\boldsymbol{A}$, analogous to the adjacency matrix for

monolayer networks, with entries

$$A_{i,\boldsymbol{\alpha}}^{j,\boldsymbol{\beta}} = \begin{cases} w(((i,\boldsymbol{\alpha}),(j,\boldsymbol{\beta}))) \,, & ((i,\boldsymbol{\alpha}),(j,\boldsymbol{\beta})) \in \mathcal{E}_M \\ 0 \,, & \text{otherwise} \,. \end{cases} \quad (1)$$

Note that $G_M = (\mathcal{V}_M, \mathcal{E}_M)$ is a graph on the state nodes of the multilayer network $M$. We refer to $G_M$ as the *flattened network* associated with $M$. The adjacency matrix of the flattened network is the "supra-adjacency matrix" [51, 67–69] of the multilayer network. One obtains the supra-adjacency matrix by flattening [70] the adjacency tensor (Eq. (1)) of the multilayer network. The multilayer network and the corresponding flattened network encode the same information [51, 71], provided one keeps track of the mapping between state nodes, nodes, and layers.

We denote a *multilayer partition* with $c$ elements by $\mathcal{S} = \{\mathcal{S}_1, \ldots, \mathcal{S}_c\}$, where $\bigcup_{s=1}^{c} \mathcal{S}_s = \mathcal{V}_M$ and $\mathcal{S}_s \cap \mathcal{S}_r = \varnothing$ for $s \neq r$. We represent a partition $\mathcal{S}$ using a *partition tensor* $\boldsymbol{S}$ with entries $S_{i,\boldsymbol{\alpha}}$, where $S_{i,\boldsymbol{\alpha}} = s$ if and only if the state node $(i,\boldsymbol{\alpha})$ is in set $\mathcal{S}_s$. A multilayer partition induces a partition $\mathcal{S}|_{\boldsymbol{\alpha}} = \{\mathcal{S}_1|_{\boldsymbol{\alpha}}, \ldots, \mathcal{S}_c|_{\boldsymbol{\alpha}}\}$ on each layer, where $\mathcal{S}_s|_{\boldsymbol{\alpha}} = \{i \in \mathcal{V} : (i,\boldsymbol{\alpha}) \in \mathcal{S}_s\}$. For convenience, we refer to an element $\mathcal{S}_s$ of a partition $\mathcal{S}$ as a *community* and to $\mathcal{S}_s|_{\boldsymbol{\alpha}}$ as the *induced community* on layer $\boldsymbol{\alpha}$. We call $s \in \{1, \ldots, c\}$ the *label* of community $\mathcal{S}_s$. We use the word "community" to make our discussions more concrete, but we emphasize that our model (see Section III) for generating a multilayer partition only assumes that its set of state nodes can be partitioned into subsets and makes no assumption on the multilayer network edge structure. In particular, one can use a planted multilayer partition to generate dependent mesoscale structures beyond community structure (e.g., including non-assortative structures) in different layers (see Section VI).

## III. GENERATING SAMPLED MULTILAYER PARTITIONS

We generate multilayer networks with planted mesoscale structure by proceeding in two steps. First, we sample a multilayer partition that has user-specified dependency properties between induced partitions in different layers. Second, we generate a random multilayer network that reflects this prescribed partition. Importantly, we generate a multilayer network after generating a multilayer partition rather than in parallel. Our approach for generating benchmark multilayer networks is thus modular, and each of these steps can be modified separately to incorporate features however one desires (see Section VI). In this section, we focus on how we generate a multilayer partition. In Section IV, we discuss how we generate multilayer networks.

We use an iterative process to introduce dependencies between induced partitions in different layers. Our community-assignment update process for generating a multilayer partition consists of two parts: (1) incorporating interlayer dependencies and (2) incorporating layer-specific random components. The first part is governed by a user-specified "interlayer dependency tensor" that determines the extent to which an induced partition in one layer can be inferred directly from the induced partition of another layer. The second part is governed by independent layer-specific "null distributions" that determine the community assignment of state nodes whenever these assignments are not updated by the interlayer dependency tensor. Our independence assumption on the null distributions allows us to keep all of the interlayer dependencies in a single object (the interlayer dependency tensor).

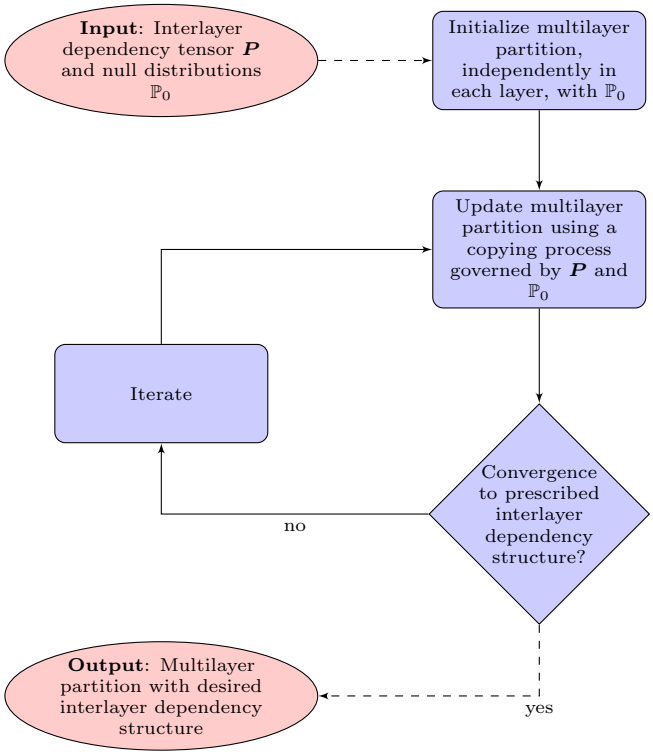



FIG. 2. Flow chart illustrating the main stages for generating a multilayer partition. We describe our iterative update process in detail in Section III A for an arbitrary choice of null distributions, and we discuss a specific choice of null distributions in Section III C.

Our procedure for generating a multilayer partition with a prescribed interlayer dependency structure has four main stages: (1) a user specifies the interlayer dependency structure via an interlayer dependency tensor; (2) one initializes the community assignments of state nodes using independent layer-specific null distributions; (3) one updates the community assignments of state nodes using a copying process that is governed by the interlayer dependency tensor and null distributions; (4) one repeats step (3) until the partition converges to a multilayer partition with the prescribed interlayer dependency structure. We illustrate these stages in Fig. 2.

In Section III A, we explain our model for generating a multilayer partition in its most general form. In Section III B, we restrict our discussions to the specific situation in which a physical node is present in all layers (i.e., the network is "fully interconnected" [51]), and in which interlayer dependencies occur only between state nodes that correspond to the same physical node (i.e., "diagonal" coupling [51]) and are uniform across state nodes for a given pair of layers (i.e., "layer-coupled" [51]). In Section III C, we describe possible choices for the layer-specific null distributions. In Section III D, we focus on the case of temporal networks with interlayer dependencies only between contiguous layers. We take advantage of the analytical tractability of this special case to discuss some of its properties and to illustrate some effects of the choice of null distribution on a multilayer partition.

### A. A general class of multilayer partitions

We begin by defining the interlayer dependency tensor. We then explain how, given a choice of null distributions, one can use the interlayer dependency tensor to generate multilayer partitions with a desired dependency structure. To do this, we use a copying process on the community assignment of state nodes to generate dependencies between induced partitions in different layers. The copying probabilities are user-specified and govern the dependencies between induced partitions.

We encode the copying probabilities in an *interlayer dependency tensor* $\boldsymbol{P}$, where $P_{i,\boldsymbol{\alpha}}^{j,\boldsymbol{\beta}}$ is the probability that state node $(j,\boldsymbol{\beta})$ copies its community assignment from state node $(i,\boldsymbol{\alpha})$. The total probability that state node $(j,\boldsymbol{\beta})$ copies its community assignment from another state node is $\hat{p}_{j,\boldsymbol{\beta}} = \sum_{(i,\boldsymbol{\alpha})\in\mathcal{V}_M} P_{i,\boldsymbol{\alpha}}^{j,\boldsymbol{\beta}}$, and we thus require that $\hat{p}_{j,\boldsymbol{\beta}} \leq 1$ for each state node $(j,\boldsymbol{\beta}) \in \mathcal{V}_M$. Furthermore, we assume that a state node can only copy its community assignment from a state node in a different layer (i.e., $P_{i,\boldsymbol{\alpha}}^{j,\boldsymbol{\beta}} = 0$ if $\boldsymbol{\alpha} = \boldsymbol{\beta}$). We call the network with adjacency tensor $\boldsymbol{P}$ the *interlayer dependency network*, and we note that it has only interlayer edges. In general, the interlayer dependency network is directed, with edges pointing in the direction of information flow between layers. The in-neighbors (see Section II) of a state node $(i,\boldsymbol{\alpha})$ in the interlayer dependency network thus correspond to the set of state nodes from which $(i,\boldsymbol{\alpha})$ can copy a community assignment.

As we mentioned in Section II, an aspect of a multilayer network can either be ordered or unordered, and we want to treat these two cases differently when generating multilayer partitions. To generate ordered aspects, the structure of the interlayer dependency tensor should reflect the causality implied by the order of the aspect's elements. For an ordered aspect, structure in a given layer therefore only depends directly on structure in previous layers. Formally, an aspect $a$ of a multilayer network is *causally ordered* if

$$\alpha_a \geq \beta_a \Rightarrow P_{i,\boldsymbol{\alpha}}^{j,\boldsymbol{\beta}} = 0\,, \tag{2}$$

where $\alpha_a$ denotes the element of state $\boldsymbol{\alpha}$ corresponding to aspect $a$ and where, as stated in Section II, we require that the labels of an aspect's elements reflect the ordering of those elements. We define a partial order for the layers based on the ordered aspects, where $\boldsymbol{\alpha} \preceq \boldsymbol{\beta}$ if and only if $\alpha_a \leq \beta_a$ for all $a \in \mathcal{O}$. This partial order of the layers allows us to combine the different notions of causality implied by the ordered aspects and to respect them through the order in which we update community assignments of state nodes.

A single community-assignment update step is independent of the partial order of the layers; it depends only on the choice of state node to update and the current multilayer partition. In this paragraph, we describe an update step for any given interlayer dependency tensor. Suppose that we are updating the community assignment of state node $(j,\boldsymbol{\beta})$ at step $\tau$ of the copying process and that the current multilayer partition is $\mathcal{S}(\tau)$ (with partition tensor $\boldsymbol{S}(\tau)$). The community assignment of state node $(j,\boldsymbol{\beta})$ is updated either by copying the community assignment in $\mathcal{S}(\tau)$ from one of its in-neighbors in the interlayer dependency network or by obtaining a new, random, community assignment from the specified *null distribution* $\mathbb{P}_0^{\boldsymbol{\beta}}$ for layer $\boldsymbol{\beta}$. In particular, with probability $\sum_{(i,\boldsymbol{\alpha})\in\mathcal{V}_M} P_{i,\boldsymbol{\alpha}}^{j,\boldsymbol{\beta}}$, a state node $(j,\boldsymbol{\beta})$ copies its community assignment from one of its neighbors in the interlayer dependency network; and with probability $1-\sum_{(i,\boldsymbol{\alpha})\in\mathcal{V}_M} P_{i,\boldsymbol{\alpha}}^{j,\boldsymbol{\beta}}$, it obtains its community assignment from the null distribution $\mathbb{P}_0^{\boldsymbol{\beta}}$. We use the notation $\mathbb{P}_0 = \{\mathbb{P}_0^{\boldsymbol{\alpha}}, \boldsymbol{\alpha} \in L\}$ for the set of null distributions [72]. This yields the following update equation at step $\tau$ of our copying process:

$$\begin{aligned}\mathbb{P}[S_{j,\boldsymbol{\beta}}(\tau+1) = s|\boldsymbol{S}(\tau)] &\\ = \left(\sum_{(i,\boldsymbol{\alpha})\in\mathcal{V}_M} P_{i,\boldsymbol{\alpha}}^{j,\boldsymbol{\beta}}\delta(S_{i,\boldsymbol{\alpha}}(\tau),s)\right) &\\ + \left(1-\sum_{(i,\boldsymbol{\alpha})\in\mathcal{V}_M} P_{i,\boldsymbol{\alpha}}^{j,\boldsymbol{\beta}}\right)\mathbb{P}_0^{\boldsymbol{\beta}}[S_{j,\boldsymbol{\beta}}=s]\,.&\end{aligned} \tag{3}$$

The update equation in Eq. (3) is at the heart of our generative model for mesoscale structure in multilayer networks. It is clear from Eq. (3) that the null distributions $\mathbb{P}_0$ are responsible for the specification of community assignments in the absence of interlayer dependencies (i.e., if $P_{i,\boldsymbol{\alpha}}^{j,\boldsymbol{\beta}} = 0$ for all $(i,\boldsymbol{\alpha})$, $(j,\boldsymbol{\beta})$). In particular, the support of the null distributions corresponds to the possible community assignments for a state node $(i,\boldsymbol{\alpha})$ whenever the state node does not copy its assignment from one of its neighbors. In Section III C, we discuss the choice of $\mathbb{P}_0$ and its effect on a sampled multilayer partition.

We want to sample multilayer partitions that are consistent with both the order of the layers and the conditional probability distributions defined by Eq. (3). We use Gibbs sampling [73], as it relies only on the conditional distributions and does not require the joint distribution for the community assignments. Our sampling algorithm proceeds as follows: (1) we sample an initial multilayer partition from the null distribution (i.e., $S_{i,\boldsymbol{\alpha}}(0) \sim \mathbb{P}_0^{\boldsymbol{\alpha}}$); and (2) we then repeatedly update the multilayer partition by sequentially updating the community assignment of each state node using Eq. (3). When updating community assignments, we respect the order of the layers: if state node $(i, \boldsymbol{\alpha})$ is updated before state node $(j, \boldsymbol{\beta})$, then $\boldsymbol{\alpha} \preceq \boldsymbol{\beta}$. This updating process defines a Markov chain on the space of multilayer partitions, and its stationary distribution is the desired joint distribution for the community assignments. We will sample from this joint distribution.

By running the updating process for a sufficiently large number of iterations (the so-called "burn-in period"), the state of the Markov chain is sampled approximately from its stationary distribution. The stationary distribution of the Markov chain defined by Gibbs sampling does not depend on the order in which we update the state nodes. However, the order of the update steps can influence its convergence speed [74]. In particular, for fully ordered multilayer networks (i.e., for multilayer networks with $\mathcal{U} = \emptyset$), updating state nodes in an order that is consistent with the order of the layers ensures that the sampling algorithm converges after a single pass over all state nodes, as the results of subsequent passes are independent from each other in this case. We expect that respecting the partial order of the layers also helps for situations in which a multilayer network is only partially ordered, though to our knowledge this question remains open.

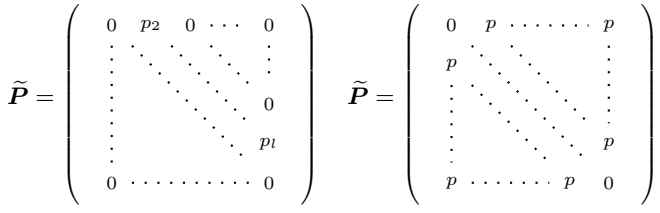
Sampling initial conditions in an 'unbiased' way can be important for ensuring that one successfully explores the space of possible partitions, as the updating process is not necessarily ergodic over the supports of the null distributions (which is a subspace of the space of all multilayer partitions) when $\sum_{(i,\boldsymbol{\alpha})\in\mathcal{V}_M} P_{i,\boldsymbol{\alpha}}^{j,\boldsymbol{\beta}} = 1$ for some state nodes. A non-ergodic updating process corresponds to the case in which the conditional probabilities given by Eq. (3) do not define a unique joint probability distribution for the community assignments. Instead, there are multiple joint distributions that are consistent with the conditional distributions. The updating process will converge to one of the possible joint distributions, and the joint distribution that is selected depends both on the initial condition and on the exact sequence of random steps taken by the updating process. For example, if $\sum_{(i,\boldsymbol{\alpha})\in\mathcal{V}_M} P_{i,\boldsymbol{\alpha}}^{j,\boldsymbol{\beta}} = 1$ for all $(j, \boldsymbol{\beta}) \in \mathcal{V}_M$, then any partition with a single community is an "absorbing state" of the Markov chain — i.e., a state that, once reached, cannot be changed by the updating process. In this example, the updating process eventually reaches an absorbing state and the distribution of final absorbing states depends on the initial partition. The absorbing states in this example correspond to partitions in which all state nodes in each component of the interlayer dependency network have the same community assignment.

Importantly, provided the updating process has converged, any partition that one generates still has the desired dependencies between induced partitions in different layers. Thus, to work around the problem of non-unique joint distributions when the updating process is not ergodic, we reinitialize the updating process by sampling from the null distribution for each partition that we want to generate. Reinitializing the initial partition from a fixed distribution for each sample always defines a unique joint distribution, which is a mixture of all possible joint distributions when the updating process is not ergodic. When the updating process is ergodic, sampling partitions by reinitializing and sampling multiple partitions from a single long chain is equivalent. The second approach is usually more efficient when many samples are needed and some dependence between samples is not an issue [75]. However, in our case, one usually only needs a few samples with the same parameters, and independence tends to be more important than ensuring perfect convergence (provided the generated partitions exhibit the desired interlayer dependency structure). Using multiple chains thus has clear advantages even when the updating process is ergodic, and it is necessary to use multiple chains when it is not ergodic.

Determining whether a Markov chain has converged to a stationary distribution is a difficult problem, mostly because it is difficult to distinguish the case of a slow-mixing chain becoming stuck in a particular part of the state space from the case in which the chain has converged to a stationary distribution. There has been much work on trying to define a convergence criterion for Markov chains [76], but none of the approaches are entirely successful. In practice, one usually runs a Markov chain (or chains) for a predetermined number of steps (e.g., 1000). One manually checks on a few examples that the resulting chains exhibit behavior that is consistent with convergence by examining autocorrelations between samples of the same chain and cross-correlations between samples of independent chains with the same initial state. When feasible, one can also check whether parts of different, independent chains or different parts of the same chain are consistent with being sampled from the same distribution.

### B. Fully-interconnected, diagonal, and layer-coupled multilayer partitions

In many situations, the complexity of allowing arbitrary interlayer dependencies is unnecessary (and even undesirable) when designing benchmark multilayer networks. A particularly useful restriction that still allows us to represent many situations of interest is to require the interlayer dependency network to be "fully intercon-



$\widetilde{P}_\alpha^\beta = \delta(\alpha+1,\beta)p_\beta\,,$ $\quad p_\beta \le 1$

(a) Temporal network

$\widetilde{P}_\alpha^\beta = (1-\delta(\alpha,\beta))p\,,$ $\quad \hat{p} = (l-1)\,p \le 1$

(b) Uniform multiplex network

FIG. 3. Layer-dependency tensors (which are matrices in this case) for different types of multilayer networks with a single aspect. (a) For a temporal network, an induced partition in a layer depends directly only on the induced partition in the previous layer. Therefore, the only nonzero elements of the layer-dependency tensor occur in the first superdiagonal. (b) For a multiplex network, an induced partition in a layer depends directly on the induced partitions in all other layers.

nected", "diagonal", and "layer-coupled" [51]. A multilayer network is *fully interconnected* if each node is represented in each layer, it is *diagonal* if interlayer edges only exist between state nodes that correspond to the same physical node, and it is *layer-coupled* if the weight of the interlayer edges depends only on the layers and not on the nodes. For a fully interconnected, diagonal, layer-coupled dependency network, we can represent the interlayer dependency tensor $\boldsymbol{P}$ using a *layer dependency tensor* $\widetilde{\boldsymbol{P}}$ such that its elements satisfy $P_{i,\boldsymbol{\alpha}}^{j,\boldsymbol{\beta}} = \delta(i,j)\widetilde{P}_{\boldsymbol{\alpha}}^{\boldsymbol{\beta}}$. The update equation (Eq. (3)) then simplifies to

$$\begin{aligned}\mathbb{P}[S_{j,\boldsymbol{\beta}}(\tau+1) = s|\boldsymbol{S}(\tau)] \\ = \left(\sum_{\boldsymbol{\alpha}\in\mathcal{L}} \widetilde{P}_{\boldsymbol{\alpha}}^{\boldsymbol{\beta}}\delta(S_{j,\boldsymbol{\alpha}}(\tau),s)\right) \\ + \left(1-\sum_{\boldsymbol{\alpha}\in\mathcal{L}}\widetilde{P}_{\boldsymbol{\alpha}}^{\boldsymbol{\beta}}\right)\mathbb{P}_0^{\boldsymbol{\beta}}[S_{j,\boldsymbol{\beta}}=s]\,,\end{aligned} \tag{4}$$

which depends only on the layer dependency tensor $\widetilde{\boldsymbol{P}}$ and the null distributions $\mathbb{P}_0$. Each term $\widetilde{P}_{\boldsymbol{\alpha}}^{\boldsymbol{\beta}}$ quantifies the extent to which an induced partition in layer $\boldsymbol{\beta}$ can be inferred directly from an induced partition in layer $\boldsymbol{\alpha}$. As before, we require that $\hat{p}_{\boldsymbol{\beta}} = \sum_{\boldsymbol{\alpha}\in\mathcal{L}}\widetilde{P}_{\boldsymbol{\alpha}}^{\boldsymbol{\beta}} \le 1$. By changing the structure of $\widetilde{\boldsymbol{P}}$, one can generate multilayer networks that correspond to several of the most important scenarios, including temporal networks, multiplex networks, and multilayer networks with more than one aspect (e.g., combinations of temporal and multiplex features). That is, the above restriction simplifies matters considerably while still allowing us to analyze several very important situations.

In Fig. 3, we show the dependency tensors for single-aspect multilayer networks. For a temporal network, one usually assumes that induced partitions in a layer depends directly only on induced partitions in the previous layer. There are thus $l-1$ copying probabilities (one for each pair of consecutive layers) that we are free to choose. Typical examples include choosing the same probability for each pair of consecutive layers to obtain a uniformly-evolving network [35, 42] or making some of the probabilities significantly smaller than the others to introduce change points that one may wish to detect [66].

For a multiplex network, an induced partition in any layer can in principle depend directly on induced partitions in all other layers. This yields $l(l-1)$ copying probabilities to choose. In Figure 3b, we illustrate the simplest case, in which each layer depends equally on every other layer. In Fig. 4 and Fig. 5, we show example multilayer partitions obtained with the interlayer dependency tensors of Fig. 3 .

We can also generate multilayer networks with more than one aspect and can thereby combine temporal and multiplex features. In Fig. 6, we illustrate how to construct an appropriate layer-dependency tensor to generate such a multilayer network on a simple example with two aspects, one of which is multiplex and the other of which is temporal.

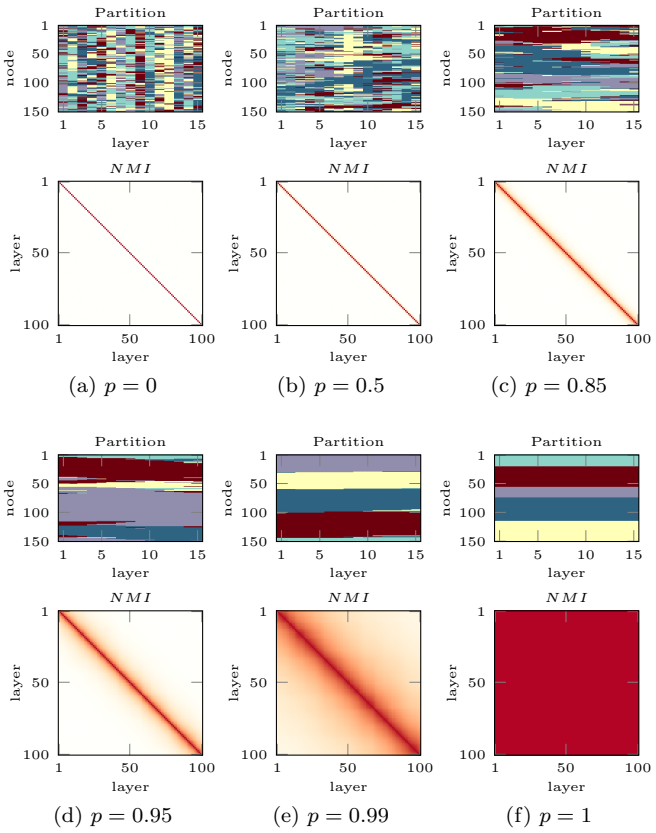
### C. Categorical null distribution

The null distributions in Eqs. (3) and (4) determine the multilayer partition in the absence of interlayer dependencies, thereby fixing the numbers and sizes of the communities in the sampled partitions. We seek to allow the possibility of having heterogeneous community-size distributions. Although the LFR benchmarks [38] incorporate such heterogeneity, the procedure used in [38] to sample community sizes and then assign nodes to communities in a way that exactly preserves these community sizes does not mesh well with the copying process that we use to induce dependencies between community structures in different layers. In particular, we consider a process that updates the community assignment of a single node at a time, which does not guarantee preserving community sizes. Instead, we only fix expected community sizes by defining null distributions $\mathbb{P}_0$ that we can use in Eqs. (3) and (4).

A simple choice for the null distributions is a categorical distribution, where for each layer $\boldsymbol{\alpha}$ and each community label $s$, we fix the probability $p_s^{\boldsymbol{\alpha}}$ for a random state node in layer $\boldsymbol{\alpha}$ to be assigned to a community $s$ in the absence of interlayer dependencies. That is,

$$\mathbb{P}_0^{\boldsymbol{\alpha}}[s] = \begin{cases} p_s^{\boldsymbol{\alpha}}\,, & s\in\{1,\dots,n_c\}\,, \\ 0\,, & \text{otherwise}\,, \end{cases} \tag{5}$$

where $n_c$ is the total number of communities in the multilayer partition and $\sum_{s=1}^{n_c} p_s^{\boldsymbol{\alpha}} = 1$ for all $\boldsymbol{\alpha}\in L$. The set $\{1,\dots,n_c\}$ corresponds to the set of community labels, and the support of a null distribution corresponds

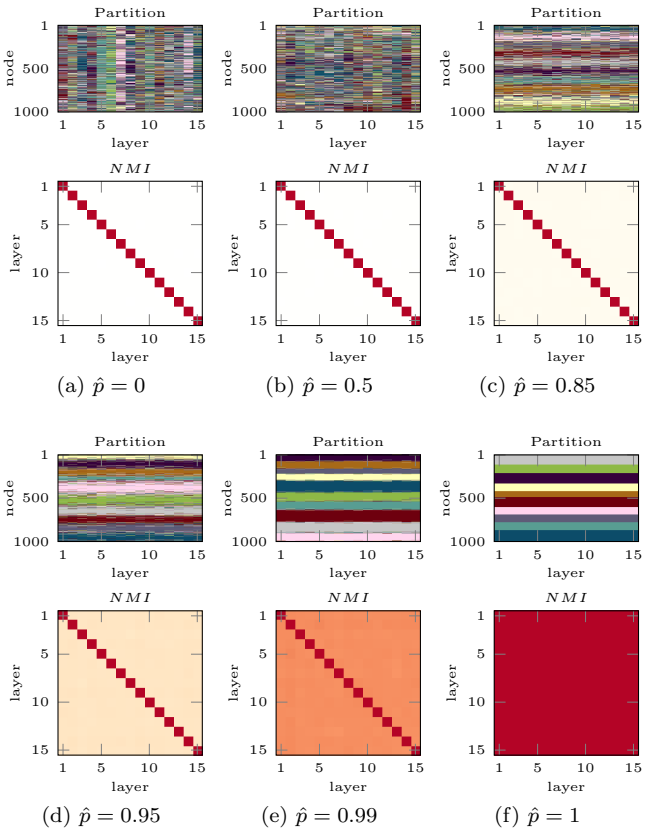



(a) $p = 0$ (b) $p = 0.5$ (c) $p = 0.85$

(d) $p = 0.95$ (e) $p = 0.99$ (f) $p = 1$

FIG. 4. Example temporal multilayer partitions for $(n, l) = (150, 100)$. For the update process, we use the interlayer dependency tensor in Fig. 3a with uniform probabilities $p_\beta = p$ for all $\beta \in \{1, \dots, l\}$ and a Dirichlet null distribution with $q = 1, \theta = 1$, and $n_c = 5$ (see Section III C). We perform a single iteration of the update process, as convergence is immediate for a fully ordered multilayer network (see Section III A). For (a) $p = 0$, (b) $p = 0.5$, (c) $p = 0.85$, (d) $p = 0.95$, (e) $p = 0.99$, and (f) $p = 1$, we show color-coded community assignments (, top) for a single example output partition and $NMI$[a] values (0 1, bottom) between induced partitions in different layers averaged over a sample of 10 output partitions. The parameter values that we use in this example match those that we use for our numerical examples in Section V (with the exception of $p = 0$, which we include for completeness). In our visualization of each multilayer partition, we choose a node ordering that emphasizes (whenever possible) "persistent" community structure [20] in that multilayer partition. We show only the first 15 layers of each multilayer partition, because (as one can see in the $NMI$ heatmaps) similarities between induced partitions for $p < 1$ decay "quickly" with the number of layers when dependencies exist only between contiguous layers.

[a] We use the variant of the normalized mutual information ($NMI$) with "joint entropy" as the normalization factor [77].




(a) $\hat{p} = 0$ (b) $\hat{p} = 0.5$ (c) $\hat{p} = 0.85$

(d) $\hat{p} = 0.95$ (e) $\hat{p} = 0.99$ (f) $\hat{p} = 1$

FIG. 5. Example multiplex multilayer partitions for $(n, l) = (1000, 15)$. For the update process, we use the interlayer dependency tensor in Fig. 3b and a Dirichlet null distribution with $q = 1, \theta = 1$, and $n_c = 10$ (see Section III C). We perform 200 iterations of our update process (see Section III A). For (a) $\hat{p} = 0$, (b) $\hat{p} = 0.5$, (c) $\hat{p} = 0.85$, (d) $\hat{p} = 0.95$, (e) $\hat{p} = 0.99$, and (f) $\hat{p} = 1$, where $\hat{p} = (l-1)p$ is the total probability that a state node copies its assignment from another state node, we show color-coded community assignments (, top) for a single example output partition and $NMI$ values (0 1, bottom) between induced partitions in different layers averaged over a sample of 10 output partitions. (For the temporal case in Fig. 4, note that $\hat{p} = p$.) The parameter values that we use match those that we use for our numerical examples in Section V (with the exception of $\hat{p} = 0$, which we include for completeness). In our visualization of each multilayer partition, we choose a node ordering that emphasizes (whenever possible) "persistent" community structure [20] in that multilayer partition.

to the set of labels that have nonzero probability. In other words, the support $\mathcal{G}^{\boldsymbol{\alpha}}$ of the null distribution $\mathbb{P}_0^{\boldsymbol{\alpha}}$ is given by $\mathcal{G}^{\boldsymbol{\alpha}} = \{s : \mathbb{P}_0^{\boldsymbol{\alpha}}[s] \neq 0\}$. We say that a label $s$ is *active* in a layer $\boldsymbol{\alpha}$ if it is in the support of the null distribution $\mathbb{P}_0^{\boldsymbol{\alpha}}$ (i.e., if $\mathbb{P}_0^{\boldsymbol{\alpha}}[s] \neq 0$), and we say that a label is *inactive* in layer $\boldsymbol{\alpha}$ if it is in the complement of the support of $\mathbb{P}_0^{\boldsymbol{\alpha}}$ (i.e., if $\mathbb{P}_0^{\boldsymbol{\alpha}}[s] = 0$). In the absence of interlayer dependencies, a categorical null distribution corresponds to fixing the expected size $np_s^{\boldsymbol{\alpha}}$ of each induced community in each layer. The actual community sizes follow a multinomial distribution. Therefore, by choosing the probabilities $p_s^{\boldsymbol{\alpha}}$, one has some control over the community-size distribution of the sampled multilayer partitions. A natural choice for $\boldsymbol{p}^{\boldsymbol{\alpha}}$ is to sample it

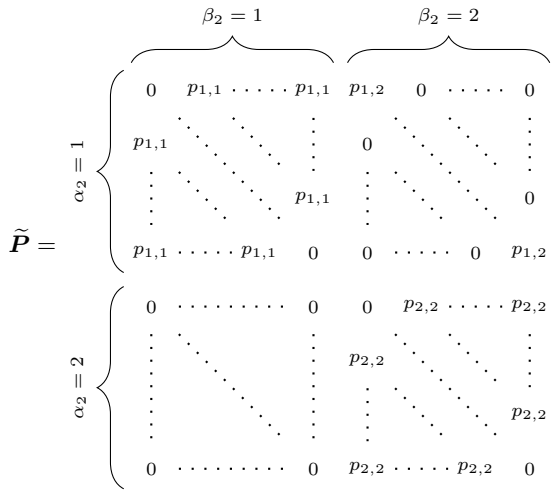



FIG. 6. Block-matrix representation of a layer-dependency tensor for a temporal multiplex network. This is an example of a multilayer network with more than one aspect and combines the features of the examples in Fig. 3. An induced partition in a layer depends directly on the induced partitions of all other layers in the same temporal layer and on the induced partition of the same layer in the previous temporal layer.

from a Dirichlet distribution, which is the conjugate prior for the categorical distribution [78, 79]. One can think of the Dirichlet distribution, which is the multivariate form of the beta distribution, as a probability distribution over the space of all possible categorical distributions with a given number of categories. Any other (probabilistic or deterministic) choice for $\boldsymbol{p}^{\boldsymbol{\alpha}}$ is also possible.

The Dirichlet distribution over $q$ variables takes $q$ parameters $\theta_1, \ldots, \theta_q$ (one for each variable). Its probability density function is

$$p(x_1, \ldots, x_q) = \frac{\Gamma\left(\sum_{i=1}^{q} \theta_i\right)}{\prod_{i=1}^{q} \Gamma(\theta_i)} \prod_{i=1}^{q} x_i^{\theta_i - 1} ,$$

where $x_i \in (0, 1)$ and $\theta_i > 0$ for all $i \in \{1, \ldots, q\}$. The case in which all $\theta_i$ are equal is called a symmetric Dirichlet distribution, which we parametrize instead by the common value $\theta$ (the so-called "concentration parameter") of the parameters and the number $q$ of variables.

The concentration parameter $\theta$ determines the kinds of discrete probability distributions that one is likely to obtain from the symmetric Dirichlet distribution. For $\theta = 1$, the symmetric Dirichlet distribution is the continuous uniform distribution over the space of all discrete probability distributions with $n_c$ states. As $\theta \to \infty$, the Dirichlet distribution becomes increasingly concentrated near the discrete uniform distribution, such that all entries $\boldsymbol{p}^{\boldsymbol{\alpha}}$ are approximately equal. As $\theta \to 0$, it becomes increasingly concentrated away from the uniform distribution, such that $\boldsymbol{p}^{\boldsymbol{\alpha}}$ tends to have 1 (or a few) large entries, and all other entries are close to 0. Consequently, to have very heterogeneous community sizes, one would choose $\theta \approx 1$. To have all communities to be of similar sizes, one would choose a large value of $\theta$. To have a few large communities and many small communities, one would choose $\theta$ sufficiently below 1. The value of $n_c$ also affects the amount of community label overlap across layers that is not a result of our copying process. For example, if $\boldsymbol{p}^{\boldsymbol{\alpha}}$ is the same for all layers, then larger values of $n_c$ incentivize less label overlap across layers (because there are more possible labels for each layer), and smaller values of $n_c$ incentivize more label overlap across layers (because there are fewer possible labels for each layer).

In some situations — e.g., when modeling the birth and death of communities in temporal networks — it is desirable to have communities that have a nonzero probability of obtaining nodes from the null distribution only in some layers. For example, if a label has 0 probability in a given layer, it may be desirable to ensure that it also has 0 probability in all subsequent layers. For these situations, we suggest sampling the support (i.e., the set of active communities in each layer) of the distributions before sampling the probabilities $\boldsymbol{p}^{\boldsymbol{\alpha}}$. As stated earlier in this section, the support $\mathcal{G}^{\boldsymbol{\alpha}}$ of the null distribution $\mathbb{P}_0^{\boldsymbol{\alpha}}$ is given by $\mathcal{G}^{\boldsymbol{\alpha}} = \{s : \mathbb{P}_0^{\boldsymbol{\alpha}}[s] \neq 0\}$. Given supports for each layer, the total number of communities is $n_c = \max_{\boldsymbol{\alpha} \in L} \max(\mathcal{G}^{\boldsymbol{\alpha}})$. We then write $\overline{\mathcal{G}}^{\boldsymbol{\alpha}} = \{s \in \{1, \ldots, n_c\} : \mathbb{P}_0^{\boldsymbol{\alpha}}[s] = 0\}$ for the complement. Given the supports for each layer, one samples the corresponding probabilities from a symmetric Dirichlet distribution. That is,

$$\boldsymbol{p}^{\boldsymbol{\alpha}}_{\mathcal{G}^{\boldsymbol{\alpha}}} \sim \mathrm{Dir}(\theta, |\mathcal{G}^{\boldsymbol{\alpha}}|) , \qquad \boldsymbol{p}^{\boldsymbol{\alpha}}_{\overline{G}^{\boldsymbol{\alpha}}} = 0 . \tag{6}$$

A simple example for a birth/death process for communities is the following. First, fix a number of communities and a support (i.e., active community labels) for the first layer. One then sequentially initializes the supports for the other layers by removing each community present in the support of the previous layer with probability $r_d \in [0, 1]$ and adding a number, sampled from a Poisson distribution with rate $r_b \in [0, \infty)$, of new communities (with new community labels that are not active in any previous layer). In temporal networks for example, this ensures that if a community label is not in the support of a given layer, then the label is also not in the support of any subsequent layers. For this process, the expected number $\langle |\mathcal{G}^{\boldsymbol{\alpha}}| \rangle$ of communities in a given layer approaches $r_b/r_d$ as the number of iterations increases. Hence, one should initialize the size of the support for the first layer close to this value to avoid transients in the number of communities. For this process, the lifetime of communities follows a geometric distribution. The nature of the copying process that we use to introduce dependencies between induced partitions in different layers tends to imply that communities that have been removed from the support of the null distribution do not lose all

of their members instantly but instead shrink at a speed that depends on the parameters (e.g., the values of the copying probabilities in the interlayer dependency tensor).

One can also allow labels to appear and disappear when examining multiplex multilayer partitions. For example, given a value for $n_c$, one can generate the support for each layer by allowing every label $s \in \{1, \dots, n_c\}$ to be present with some probability $\tilde{q}$ and absent with complementary probability $1 - \tilde{q}$. This yields a sets of active and inactive community labels for each layer. One can then sample the nonzero probabilities in $\boldsymbol{p}^{\boldsymbol{\alpha}}$ that correspond to active labels from a Dirichlet distribution and set $p_s^{\boldsymbol{\alpha}}$ to 0 for every inactive label $s$. Because multiplex partitions are unordered, there is no notion of one layer occurring after another one, so we do not need to ensure that an inactive label in a given layer is also inactive in "subsequent" layers.

In general, the choice of $\boldsymbol{p}^{\boldsymbol{\alpha}}$, which encodes both the expected community sizes and the support for the null distribution (by allowing some community labels to have 0 probability), can have a very large effect on the set of sampled multilayer partitions. We illustrate some effects of the choice of $\boldsymbol{p}^{\boldsymbol{\alpha}}$ for the case of fully ordered temporal multilayer networks in Section III D. For our numerical examples in Section V, we fix a value of $n_c \in \{1, \dots, nl\}$ and use a symmetric Dirichlet distribution with parameters $q = n_c$ and $\theta = 1$ to sample probability vectors $\boldsymbol{p}^{\boldsymbol{\alpha}}$ of length $n_c$. This produces multilayer partitions in which the active community labels are the same across layers (and given by $\{1, \dots, n_c\}$) and for which the expected induced community sizes (given by $np_s^{\boldsymbol{\alpha}}$) vary across layers.

### D. Temporal multilayer partitions

For the important special case of temporal networks that we illustrated in Fig. 3a (where each layer depends only on the previous one), our generative model for multilayer partitions simplifies significantly. We take advantage of the analytic tractability of this special case to point out some of its properties that hold for any choice of $\mathbb{P}_0$, and we use it to illustrate some features of the categorical null distribution in Section III C. In our discussion, we assume that dependencies between contiguous layers are uniform. That is, $p_\beta = p \in [0, 1]$ for all $\beta \in \{2, \dots, l\}$ in Fig 3a. (We note that one could generate change points [29, 66] by relaxing this assumption and allowing some probabilities to be significantly smaller than others. We show an example of this in Section V).

This example includes a single ordered aspect, so the layer index $\alpha \in \mathbb{N}$ is a scalar and the order of the layers corresponds to temporal ordering. Furthermore, as we mentioned in Section III A, for a fully ordered multilayer network, we require that the order of the community-assignment update process in Eq. (3) respects the order of the layers. The update order of state nodes $(1, \alpha) \dots (n, \alpha)$ in any given layer $\alpha$ can be arbitrary (i.e., these can be updated simultaneously), but each update is conditional on the community assignment of state nodes in layer $\alpha - 1$. The update process described in Section III A for generating a multilayer partition thus reduces to the procedure described in Algorithm 1. In accord with the discussion in Section III A, note that convergence is not an issue for this case as only one pass is needed. In Fig. 4, we show example multilayer partitions for this scenario using different values of $p$ .

The generative model for temporal multilayer partitions in Algorithm 1 was also suggested in [35]. The authors of that paper used the model to derive a detectability threshold for a planted partition for the case in which the null distributions are uniform across communities (i.e., $\theta \to \infty$ in Section III C) and intralayer edges are generated independently using the standard SBM (i.e., one replaces the DCSBM in Section IV by the non degree-corrected SBM in [26]). In the following paragraphs, we highlight properties of the generative model in Algorithm 1 that hold for any choice of null distributions, and we illustrate that the choice of null distributions can greatly influence resulting partitions. Our observations are independent of one's choice of monolayer network model with a planted partition. We also discuss how to generalize the temporal multilayer partition model in Algorithm 1 to include memory effects [80] and "burstiness" [81] in Section VI.

A first important feature of the generative model in Algorithm 1 is that it respects the arrow of time. In particular, community assignments in a given layer depend only on community assignments in the previous layer (e.g., the previous temporal snapshot) and on the null distributions $\mathbb{P}_0$. That is, for all $s \in \{1, \dots, nl\}$ and all $\alpha \in \{2, \dots, l\}$, the following condition is satisfied:

$$\begin{aligned} \mathbb{P}\left[S_{i,\alpha} = s \middle| \mathcal{S}|_{\alpha-1}\right] = p\delta(S_{i,\alpha-1}, s) \\ + (1-p)\mathbb{P}_0^{\alpha}\left[S_{i,\alpha} = s\right] , \end{aligned} \tag{7}$$

where $\mathcal{S}|_\alpha$ is the $n$-node partition induced on layer $\alpha$ by the multilayer partition $\mathcal{S}$. The relative importance of the previous layer versus the null distribution is determined by the value of $p$. When $p = 0$, community assignments in a given layer depend only on the null distribution of that layer [i.e., on the second term on the right-hand side of Eq. (7)]. When $p = 1$, community assignments in a given layer are identical to the community assignments of the previous layer (and, by recursion, to community assignments in all previous layers).

Using Eq. (7), one can easily compute the marginal probability that a given state node has a specific com-

**function** TEMPORALPARTITION($p$, $\mathbb{P}_0$)
  $\boldsymbol{S} = \mathbf{0}$ ▷ Initialize partition tensor of appropriate size
  **for** $i \in V$ **do** ▷ Loop over nodes in some order
    $S_{i,1} \sim \mathbb{P}_0^1$ ▷ Initialize induced partition on first layer using null distribution
  **end for**
  **for** $\alpha \in 2 \dots l$ **do** ▷ Loop over layers in sequential order
    **for** $i \in V$ **do** ▷ Loop over nodes in some order
      with probability $p$
      $S_{i,\alpha} = S_{i,\alpha-1}$; ▷ Copying step (Step C)
      with probability $1-p$
      $S_{i,\alpha} \sim \mathbb{P}_0^\alpha$ ▷ Reallocation step (Step R)
    **end for**
  **end for**
  **return** $\boldsymbol{S}$
**end function**

ALG. 1. Pseudocode for generating temporal multilayer partitions with uniform interlayer dependencies $p$ between successive layers (i.e., $p_\beta = p$ for all $\beta \in \{1, \dots, l\}$ in Fig. 3a).

munity assignment:

$$\begin{aligned}\mathbb{P}\left[S_{i,\alpha} = s\right] &= p\mathbb{P}\left[S_{i,\alpha-1} = s\right] \\ &\quad + (1-p)\mathbb{P}_0^\alpha\left[S_{i,\alpha} = s\right] , \\ &= \mathbb{P}_0^1\left[S_{i,1} = s\right] p^{\alpha-1} \\ &\quad + (1-p)\sum_{\beta=2}^{\alpha-1} \mathbb{P}_0^\beta\left[S_{i,\beta} = s\right] p^{\alpha-\beta} \\ &\quad + (1-p)\mathbb{P}_0^\alpha\left[S_{i,\alpha} = s\right] , \quad \alpha > 1 .\end{aligned}$$

Computing marginal probabilities can be useful for computing expected community sizes for a given choice of null distributions.

We now highlight how the copying step (i.e., Step C) and the reallocation step (i.e., Step R) in Algorithm 1 govern the evolution of community assignments between consecutive layers. Steps C and R deal with the movement of nodes by first removing some nodes ("subtraction") and then reallocating them ("addition"). In Step C, a community assignment $s$ in layer $\alpha$ can lose a number of nodes that ranges from 0 to all of them. It can keep all of its nodes in layer $\alpha+1$ (i.e., $\mathcal{S}_s|_{\alpha+1} = \mathcal{S}_s|_\alpha$), lose some of its nodes (i.e., $\mathcal{S}_s|_{\alpha+1} \subset \mathcal{S}_s|_\alpha$), or disappear entirely (i.e., $\mathcal{S}_s|_{\alpha+1} = \varnothing$ and $\mathcal{S}_s|_\alpha \neq \varnothing$). The null distribution in Step R is responsible for a community assignment $s$ gaining new nodes (i.e., $\mathcal{S}_s|_{\alpha+1} \not\subset \mathcal{S}_s|_\alpha$) or for a community label appearing (i.e., $\mathcal{S}_s|_{\alpha+1} \neq \varnothing$ and $\mathcal{S}_s|_\alpha = \varnothing$). One needs to bear the interplay between Step C and Step R in mind when defining the null distributions $\mathbb{P}_0$.

To illustrate how the community-assignment copying process and the null distribution in Algorithm 1 can interact with each other, we give the conditional probability that a label disappears in layer $\alpha$ and the conditional probability that a label appears in layer $\alpha$. For all $s \in \{1, \dots, nl\}$ and all $\alpha \in \{2, \dots, l\}$, the conditional probability that a label disappears in layer $\alpha$ is

$$\begin{aligned}&\mathbb{P}\left[\mathcal{S}_s|_\alpha = \varnothing \middle|\, \mathcal{S}|_{\alpha-1}\right] \\ &\qquad = \left[(1-p)\left(1 - \mathbb{P}_0^\alpha[S_{i,\alpha} = s]\right)\right]^{\left|\mathcal{S}_s|_{\alpha-1}\right|} \\ &\qquad \times \left[p + (1-p)\left(1 - \mathbb{P}_0^\alpha[S_{i,\alpha} = s]\right)\right]^{n - \left|\mathcal{S}_s|_{\alpha-1}\right|} .\end{aligned}$$

This expression depends only on our copying process and simplifies to $(1-p)^{\left|\mathcal{S}_s|_{\alpha-1}\right|}$ when $\mathbb{P}_0^\alpha[\mathcal{S}_{i,\alpha} = s] = 0$ (i.e., when the probability of being assigned to label $s$ is 0 using the null distribution of layer $\alpha$). Furthermore, as $\mathbb{P}_0^\alpha[\mathcal{S}_{i,\alpha} = s]$ increases, the probability that a label disappears decreases.

For all $s \in \{1, \dots, nl\}$ and all $\alpha \in \{2, \dots, l\}$, the conditional probability that a label appears in layer $\alpha$ is

$$\begin{aligned}&\mathbb{P}\left[\, \mathcal{S}_s|_\alpha \neq \varnothing \middle|\, \mathcal{S}_s|_{\alpha-1} = \varnothing\right] \\ &\qquad = 1 - \left[p + (1-p)\left(\sum_{\mathcal{S}_r|_{\alpha-1} \in \mathcal{S}|_{\alpha-1}} \mathbb{P}_0^\alpha[S_{i,\alpha} = r]\right)\right]^n .\end{aligned}$$

This expression gives the probability that at least one node in layer $\alpha$ has the label $s$, given that no node in layer $\alpha-1$ has the label $s$. When $\sum_{\mathcal{S}_r|_{\alpha-1} \in \mathcal{S}|_{\alpha-1}} \mathbb{P}_0^\alpha[S_{i,\alpha} = r] = 0$, the probability that a node disappears depends only on our community-assignment copying process and is given by $1-p^n$. Furthermore, larger values of $\sum_{\mathcal{S}_r|_{\alpha-1} \in \mathcal{S}|_{\alpha-1}} \mathbb{P}_0^\alpha[S_{i,\alpha} = r]$ decrease the probability that a label appears in layer $\alpha$.

With the exception of Section III C, our discussions thus far hold for any choice of $\mathbb{P}_0$. In the next two paragraphs, we give two examples to illustrate some features of the categorical null distribution in Section III C. In particular, we focus on the effect of the support of a categorical null distribution on a sampled multilayer partition. As we stated in Section III C, the support $\mathcal{G}^{\boldsymbol{\alpha}}$ of a categorical null distribution $\mathbb{P}_0^{\boldsymbol{\alpha}}$ is given by $\mathcal{G}^{\boldsymbol{\alpha}} = \{s : p_s^{\boldsymbol{\alpha}} \neq 0\}$, where $s \in \{1, \dots, n_c\}$. An important feature of the support for a multilayer partition generated with the interlayer dependency matrix in Fig. 3a is that overlap between $\mathcal{G}^\alpha$ and $\mathcal{G}^{\alpha+1}$ (i.e., $\mathcal{G}^{\alpha+1} \cap \mathcal{G}^\alpha \neq \varnothing$) is a necessary condition for communities in layer $\alpha$ to gain new members in layer $\alpha+1$.

Let $\boldsymbol{c}^\alpha$ denote the vector of expected induced community sizes in layer $\alpha$ (i.e., $c_s^\alpha = np_s^\alpha$), and suppose that the probabilities $\boldsymbol{p}^\alpha$ are the same in each layer (i.e., $\boldsymbol{p}^\alpha = \boldsymbol{p}$ for all $\alpha$). The expected number of community labels is then the same for each layer, and the expected number of nodes with community label $s$ is also the same in each layer and is given by $c_s^\alpha$. This choice produces a temporal network in which nodes change community labels across layers in a way that preserves both the expected number of induced communities in a layer and the expected size of induced communities in a layer.

Now suppose that one chooses the $\boldsymbol{p}^\alpha$ values such that their supports are nonoverlapping (i.e., $\mathcal{G}^\alpha \cap \mathcal{G}^\beta = \varnothing$ for all $\alpha \neq \beta$). At each iteration of Step C in Algorithm 1,

an existing community label can then only lose members; and with probability $1-p^n$, at least one new label will appear in every subsequent layer. For this case, one expects that $pc_s^\alpha$ members of community $s$ in layer $\alpha$ to remain in community $s$ in layer $\alpha+1$ and that $(1-p)c_s^\alpha$ members of community $s$ in layer $\alpha$ are assigned to new communities (because labels are nonoverlapping) in layer $\alpha+1$. This choice thus produces multilayer partitions in which the expected number of new community labels per layer is nonzero (unless $p=1$) and the expected size of a given induced community decreases in time.

## IV. SAMPLING NETWORK EDGES

Having generated a multilayer partition $\mathcal{S}$, we want to sample multilayer networks in a way that reflects the desired mesoscale structure. We assume that all interdependencies between the different layers of the multilayer network are a result of dependencies between the partitions induced on the different layers. Therefore, we can generate edges independently for each layer. In Section VI, we point out how one can generalize our network generation process to include dependencies between layers beyond those induced by planted mesoscale structures. As we mentioned in Section I, for the purpose of the numerical examples in Section V, we generate multilayer networks without interlayer edges and with intralayer edges that reflect planted community structure in each layer. We consider multilayer networks without interlayer edges and with planted community structure (rather than another type of mesoscale structure) as an illustrative example, because it is the most commonly studied case. However, the multilayer SBM that we discuss in this section can be used to generate not only intralayer edges but also interlayer ones, and it can also be used to generate mesoscale structures other than the assortative structures of "communities".

The generative network model that we discuss in this section is a generalization to multilayer networks of the degree-corrected SBM (DCSBM) [27]. In other words, it is a multilayer DCSBM (M-DCSBM). The parameters of a general, directed M-DCSBM are a multilayer partition $\mathcal{S}$ (which determines the assignment of state nodes to communities), a block tensor $\boldsymbol{W}$ (which determines the expected number of edges between communities in different layers), and a set $\boldsymbol{\sigma}$ of state-node parameters (which determine the allocation of edges to state nodes within communities).

The probability of observing an edge (or the expected number of edges if we allow multi-edges) from state node $(i,\boldsymbol{\alpha})$ to state node $(j,\boldsymbol{\beta})$ with community assignments $r=S_{i,\boldsymbol{\alpha}}$ and $s=S_{j,\boldsymbol{\beta}}$ in a M-DCSBM is

$$\mathbb{P}\left[A_{i,\boldsymbol{\alpha}}^{j,\boldsymbol{\beta}}=1\right]=\sigma_{i,\boldsymbol{\alpha}}^{\boldsymbol{\beta}}\,W_{r,\boldsymbol{\alpha}}^{s,\boldsymbol{\beta}}\,\sigma_{\boldsymbol{\alpha}}^{j,\boldsymbol{\beta}}\,, \tag{8}$$

where $W_{r,\boldsymbol{\alpha}}^{s,\boldsymbol{\beta}}$ is the expected number of edges from state nodes in layer $\boldsymbol{\alpha}$ and community $\mathcal{S}_r$ to state nodes in layer $\boldsymbol{\beta}$ and community $\mathcal{S}_s$, the quantity $\sigma_{i,\boldsymbol{\alpha}}^{\boldsymbol{\beta}}$ is the probability for a random edge starting in community $\mathcal{S}_r$ in layer $\boldsymbol{\alpha}$ and ending in layer $\boldsymbol{\beta}$ to be attached to state node $(i,\boldsymbol{\alpha})$ (note that the dependence on $\mathcal{S}_r$ is implicit in $\sigma_{i,\boldsymbol{\alpha}}^{\boldsymbol{\beta}}$), and $\sigma_{\boldsymbol{\alpha}}^{j,\boldsymbol{\beta}}$ is the probability for an edge starting in layer $\boldsymbol{\alpha}$ and ending in community $\mathcal{S}_s$ in layer $\boldsymbol{\beta}$ to be attached to state node $(j,\boldsymbol{\beta})$ (note that the dependence on $\mathcal{S}_s$ is implicit in $\sigma_{\boldsymbol{\alpha}}^{j,\boldsymbol{\beta}}$). For an undirected M-DCSBM, both the block tensor $\boldsymbol{W}$ and the state-node parameters $\boldsymbol{\sigma}$ are symmetric. That is, $W_{s,\boldsymbol{\beta}}^{r,\boldsymbol{\alpha}}=W_{r,\boldsymbol{\alpha}}^{s,\boldsymbol{\beta}}$ and $\sigma_{\boldsymbol{\beta}}^{i,\boldsymbol{\alpha}}=\sigma_{i,\boldsymbol{\alpha}}^{\boldsymbol{\beta}}$.

The above M-DCSBM can generate multilayer networks with arbitrary expected layer-specific in-degrees and out-degrees for each state node. (Note that the DCSBM [27] can generate monolayer networks with arbitrary expected degrees.) Given a multilayer network with adjacency tensor $\boldsymbol{A}$, the *layer-$\boldsymbol{\alpha}$-specific in-degree* of state node $(j,\boldsymbol{\beta})$ is

$$k_{\boldsymbol{\alpha}}^{j,\boldsymbol{\beta}}=\sum_{i\in\mathcal{V}}A_{i,\boldsymbol{\alpha}}^{j,\boldsymbol{\beta}}\,,$$

and the *layer-$\boldsymbol{\beta}$-specific out-degree* of state node $(i,\boldsymbol{\alpha})$ is

$$k_{i,\boldsymbol{\alpha}}^{\boldsymbol{\beta}}=\sum_{j\in\mathcal{V}}A_{i,\boldsymbol{\alpha}}^{j,\boldsymbol{\beta}}\,.$$

(Note that the layer-$\boldsymbol{\alpha}$-specific in-degree and out-degree of state node $(i,\boldsymbol{\alpha})$ are the "intralayer in-degree" and "intralayer out-degree" of $(i,\boldsymbol{\alpha})$.) For an undirected multilayer network, layer-$\boldsymbol{\beta}$-specific in-degrees and out-degrees

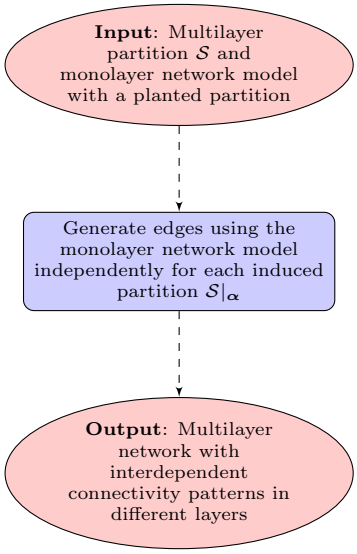



FIG. 7. Flow chart illustrating the main steps for generating a multilayer network with a planted mesoscale structure. In this paper, we use the DCSBM as a monolayer network model with a planted partition (see Section IV). We generate multilayer networks without interlayer edges and interdependent connectivity patterns in different layers for our numerical experiments, but the M-DCSBM that we introduce in Section IV can be used to generate both interlayer and intralayer edges.

are equal (i.e., $k_{\boldsymbol{\beta}}^{i,\boldsymbol{\alpha}} = k_{i,\boldsymbol{\alpha}}^{\boldsymbol{\beta}}$). We refer to their common value as the "layer-$\boldsymbol{\beta}$-specific degree" of a state node. For an ensemble of networks generated from an M-DCSBM, the associated means are

$$\langle k_{\boldsymbol{\alpha}}^{j,\boldsymbol{\beta}} \rangle = \sigma_{\boldsymbol{\alpha}}^{j,\boldsymbol{\beta}} \sum_{r=1}^{|\mathcal{S}|} W_{r,\boldsymbol{\alpha}}^{s,\boldsymbol{\beta}}\,, \qquad s = S_{j,\boldsymbol{\beta}} \tag{9}$$

and

$$\langle k_{i,\boldsymbol{\alpha}}^{\boldsymbol{\beta}} \rangle = \sigma_{i,\boldsymbol{\alpha}}^{\boldsymbol{\beta}} \sum_{s=1}^{|\mathcal{S}|} W_{r,\boldsymbol{\alpha}}^{s,\boldsymbol{\beta}}\,, \qquad r = S_{i,\boldsymbol{\alpha}}\,. \tag{10}$$

For the experiments in Section V, we use a model that is a slight variant (avoiding the creation of self-loops and multi-edges) of the DCSBM benchmark suggested in [27]. As we mentioned earlier, we only consider undirected multilayer networks with only intralayer edges for our experiments. The block tensor $\boldsymbol{W}$ thus does not have any interlayer contributions (i.e., $W_{r,\boldsymbol{\alpha}}^{s,\boldsymbol{\beta}} = 0$ if $\boldsymbol{\alpha} \neq \boldsymbol{\beta}$). Furthermore, we can reduce the number of node parameters that we need to specify the M-DCSBM to a single parameter $\sigma_{i,\boldsymbol{\alpha}} = \sigma_{i,\boldsymbol{\alpha}}^{\boldsymbol{\alpha}} = \sigma_{\boldsymbol{\alpha}}^{i,\boldsymbol{\alpha}}$ for each state node $(i,\boldsymbol{\alpha})$. For this case, the M-DCSBM reduces to using independent monolayer DCSBMs for each layer. In Fig. 7, we describe the main stages for generating edges for a given multilayer partition with an arbitrary choice of monolayer network model with a planted partition.

We parametrize the DCSBM benchmark in terms of its distribution of expected degrees and a community-mixing parameter $\mu \in [0,1]$ that controls the strength of the community structure in the benchmark. For $\mu = 0$, all edges lie within communities; for $\mu = 1$, edges are distributed independently of the communities, where the probability of observing an edge between two state nodes in the same layer depends only on the expected degrees of those two state nodes. We choose a truncated power law as the distribution for expected degrees.

We sample the expected intralayer degrees $e_{i,\boldsymbol{\alpha}} = \langle k_{i,\boldsymbol{\alpha}}^{\boldsymbol{\alpha}} \rangle$, where $k_{i,\boldsymbol{\alpha}}^{\boldsymbol{\alpha}} = \sum_{j \in \mathcal{V}_M} A_{i,\boldsymbol{\alpha}}^{j,\boldsymbol{\alpha}}$, for the state nodes from a truncated power law [82] with exponent $\tau_k$, minimum cutoff $k_{\min}$, and maximum cut-off $k_{\max}$. We then construct the block tensor $\boldsymbol{W}$ and state node parameters $\boldsymbol{\sigma}$ for the M-DCSBM from the sampled expected degrees $\boldsymbol{e}$ and the community assignments $\mathcal{S}$. Let

$$\kappa_{s,\boldsymbol{\alpha}} = \sum_{i \in \mathcal{S}_s|_{\boldsymbol{\alpha}}} e_{i,\boldsymbol{\alpha}}\,, \qquad \mathcal{S}_s \in \mathcal{S}$$

be the expected degree of community $s$ in layer $\boldsymbol{\alpha}$, and let

$$w_{\boldsymbol{\alpha}} = \frac{1}{2} \sum_{i \in \mathcal{V}} e_{i,\boldsymbol{\alpha}}$$

be the expected number of edges in layer $\boldsymbol{\alpha}$. Consequently,

$$\sigma_{i,\boldsymbol{\alpha}} = \frac{e_{i,\boldsymbol{\alpha}}}{\kappa_{s,\boldsymbol{\alpha}}}\,, \qquad s = S_{i,\boldsymbol{\alpha}}$$

is the probability for an intralayer edge in layer $\boldsymbol{\alpha}$ to be attached to the particular state node $(i,\boldsymbol{\alpha})$, given that the edge is attached to a state node in layer $\boldsymbol{\alpha}$ and is attached to community $S_{i,\boldsymbol{\alpha}}$.

The elements

$$W_{r,\boldsymbol{\alpha}}^{s,\boldsymbol{\beta}} = \delta(\boldsymbol{\alpha},\boldsymbol{\beta}) \left( (1-\mu)\delta(r,s)\kappa_{s,\boldsymbol{\alpha}} + \mu \frac{\kappa_{r,\boldsymbol{\alpha}}\kappa_{s,\boldsymbol{\alpha}}}{2w_{\boldsymbol{\alpha}}} \right)$$

of the block tensor give, for $r \neq s$, the expected number of edges between state nodes in community $s$ in layer $\boldsymbol{\beta}$ and state nodes in community $r$ in layer $\boldsymbol{\alpha}$. For $s = r$ and $\boldsymbol{\beta} = \boldsymbol{\alpha}$, the block-tensor element $W_{r,\boldsymbol{\alpha}}^{s,\boldsymbol{\beta}}$ instead gives twice the expected number of edges. One way to think of the DCSBM benchmark is that we categorize each edge that we want to sample as an intracommunity edge with probability $1-\mu$ or as a "random edge" (i.e., an edge that can be either an intracommunity edge or an intercommunity edge) with probability $\mu$. To sample an edge, we sample two state nodes (which we then join by an edge). We call these two state nodes the "end points" of the edge. The two end points of an intracommunity edge are sampled with a frequency that is proportional to the expected degree of their associated state nodes, conditional on the end points being in the same community. By contrast, the two end points of a random edge are sampled with a frequency proportional to expected degree of their associated nodes (without conditioning on anything). We assume that the total number of edges in a layer $\boldsymbol{\alpha}$ is sampled from a Poisson distribution [83] with mean $w_{\boldsymbol{\alpha}}$. One can easily extend this model to generate interlayer edges [84].

Although our procedure for sampling edges at the end of the previous paragraph describes a potential algorithm for sampling networks from the DCSBM benchmark, it is usually more efficient to sample edges separately for each pair of communities. We describe this process in Algorithm 2. The only difference between Algorithm 2 and the sampling algorithm of [27] is that we use rejection sampling to avoid creating self-loops and multi-edges (i.e., if we sample an edge that has already been sampled or that is a self-loop, then we do not include it in the multilayer network and resample). Rejection sampling is efficient provided all blocks of the network are sufficiently sparse, such that the probability of generating multi-edges remains small. For dense blocks of the network, we instead sample edges from independent Bernoulli distributions with success probability given by Eq. (8). This algorithm for sampling networks from a DCSBM is very efficient, as it scales linearly with the number of edges in a network.

One of the key reasons for using a multilayer approach for identifying communities in a multilayer network (rather than identifying communities in each layer separately) is that one would expect to be able to identify weaker planted community structure (e.g., $\mu$ values closer to 1 in Algorithm 2) using information from multiple layers with similar structures. The DCSBM benchmark produces networks with truncated power-law degree distri-

```
function DCSBM(S, σ, W)
    A = 0                                  ▷ Initialize adjacency tensor
                                             of appropriate size
    for α ∈ L do                           ▷ Loop over layers
        for r ∈ 1 ... |S| do
            for s ∈ r ... |S| do
                                           ▷ Sample number of edges
                                             from a Poisson distribution
                if r = s then
                    m = Poisson(W^{r;α}_{r,α}/2)
                else
                    m = Poisson(W^{s;α}_{r,α})
                end if
                e = 0                      ▷ Count edges sampled
                while e < m do
                                           ▷ Sample nodes from
                                             communities; node i is
                                             sampled with probability
                                             σ_{i,α} if it is in the
                                             community
                    i = Sample(S_r|_α, σ_{S_r|_α,α})
                    j = Sample(S_s|_α, σ_{S_s|_α,α})
                    if i ≠ j  &  A^{j,α}_{i,α} = 0 then
                                           ▷ Reject self-loops or
                                             multi-edges
                        A^{j,α}_{i,α} = 1,  A^{i,α}_{j,α} = 1
                        e = e + 1
                    end if
                end while
            end for
        end for
    end for
    return A
end function
```

ALG. 2. Sampling multilayer networks from a DCSBM with community assignments $\mathcal{S}$, node parameters $\boldsymbol{\sigma}$, and block tensor $\boldsymbol{W}$.

butions that are similar to the degree distributions of networks generated by the LFR benchmark. However, the DCSBM benchmark differs from the network-generation process of the LFR benchmark in crucial ways that make it more suitable for our purposes and that we specify below.

The LFR benchmarks impose community structure by ensuring that each node in a sampled network has a specified fraction of intra-community edges. This can create a sharp transition between networks with clearly identifiable community structure (where information from a single sample is sufficient to identify the structure) and those without identifiable community structure (where even information from many samples does not allow one to identify the structure). This leaves only a narrow range of parameters for LFR models in which methods based on a multilayer approach have the potential to improve on the performance of monolayer-network methods.

The DCSBM benchmark that we discuss in this section only imposes community structure as an expected feature of an ensemble of networks that it generates. Additionally, the definition of the mixing parameter $\mu$ of the DCSBM benchmark ensures that the planted partition remains community-like (i.e., within-community edges are more likely to be observed and between-community edges are less likely to be observed than in a random network with the same expected degrees) for any value of $\mu < 1$. (This contrasts with the behavior of the LFR benchmark, for which a planted partition switches from being community-like to being multipartite-like at some intermediate, parameter-dependent value of the mixing parameter.) Consequently, given sufficiently many samples from the same DCSBM benchmark (i.e., all samples have the same planted partition and expected degrees), one should be able to identify the planted community structure for any value of $\mu < 1$ (where the necessary number of samples goes to infinity as $\mu \to 1$). This feature makes the DCSBM benchmark an interesting test for the ability of multilayer community-detection methods to aggregate information from multiple layers. Given a multilayer network with sufficiently many layers that have sufficiently similar structures, we expect multilayer methods to be able to detect the community structure even for values of $\mu$ for which it is impossible to detect using monolayer-network methods. We show examples of this in Section V, where we note that the ability of multilayer community detection methods to exploit interlayer dependencies is more pronounced in the temporal examples that we consider (see Section V B) than in our multiplex examples (see Section V A).

## V. NUMERICAL EXAMPLES

In this section, we use the benchmark networks to compare the behavior of different variants of the Louvain-like [85] computational heuristic [86] to optimize a multilayer modularity objective function [10, 20] (using the standard Newman–Girvan null model, which is a variant of a "configuration model" [87]). Modularity is an objective function that is often used to partition sets of nodes into communities that have a larger total internal edge weight than the expected total internal edge weight in the same sets in a "null network" [20], which is generated from some null model. Modularity maximization consists of finding a partition that maximizes this difference. For our numerical experiments, we use the generalization of modularity to multilayer networks in [10]. In a multilayer network with "uniform" interlayer coupling, the strength of interaction between different layers of the network is governed by a layer-independent and node-independent interlayer parameter $\omega \geq 0$. We use diagonal and categorical (i.e., between all pairs of layers) interlayer coupling $\omega$ for the multiplex examples in Section V A, and we use diagonal and ordinal (i.e., between contiguous layers) interlayer coupling $\omega$ for the temporal examples in Section V B.

The Louvain algorithm [85] for maximizing (monolayer or multilayer) modularity proceeds in two phases, which are repeated iteratively. Starting from an initial parti-

tion, one considers the state nodes one by one (in some order) and places each state node in a set that results in the largest increase of modularity. (If there is no move that improves modularity, then a state node keeps the same assignment.) One repeats this first phase of the algorithm until reaching a local maximum. In the second phase of the Louvain algorithm, one obtains a new reduced modularity matrix by aggregating the sets of state nodes that one obtains after the convergence of the first phase. One then applies the algorithm's first phase to the new modularity matrix and iterates both phases until convergence to a local maximum. The two Louvain-like algorithms that we use in this section differ in how they select which moves to make. The first is GenLouvain, which always chooses the move that maximally increases modularity; the second is GenLouvainRand, which chooses modularity-increasing moves at random, such that the probability of a particular move is proportional to the resulting increase in the quality function. (The latter is a variant of the algorithm "LouvainRand" in [20], which chooses modularity-increasing moves uniformly at random.)

We also compare multilayer modularity maximization with multilayer InfoMap [88] [24], which uses an objective function called the "map equation" (which is not an equation), based on a discrete-time random walk and ideas from coding theory, to coarse-grain sets of nodes into communities [89]. In multilayer InfoMap, one uses a probability $r \in [0, 1]$ called the "relaxation rate" to control the relative frequency with which a random walker remains in the same layer or moves to other layers. (A random walker cannot change layers when $r = 0$.) The relaxation rate thus controls the interactions between different layers of a multilayer network. We allow the random walker to move to all other layers when $r \neq 0$ for the multiplex examples in Section V A, and we allow the random walker to move only to adjacent layers for the temporal examples in Section V B.

In all experiments in this section, we generate a multilayer partition using our copying process in Section III and a multilayer network for a fixed planted partition using the network model in Section IV. Given a multilayer planted partition, we generate multilayer networks that have only intralayer edges for our numerical computations. This produces multilayer networks in which the connectivity patterns in different layers are interdependent. We use normalized mutual information (NMI) [90] (with "joint entropy" as a normalization factor [77]) to compare the performance of different community-detection algorithms.

For each partition that we identify with an algorithm, we compute NMI between (1) the partition induced on each layer by the output partition and (2) that induced by the planted partition, and we compute the mean NMI ($\langle \text{NMI} \rangle$) by averaging across layers and runs of the algorithm. In all of our numerical examples, we average the results over 10 runs of the community-detection algorithms on one instantiation of the benchmark for each value of the parameters. All variables (for each parameter choice and each algorithmic run) used to generate Fig. 8, Fig. 9, and Fig. 10 are available in the Supplemental Information (SI).

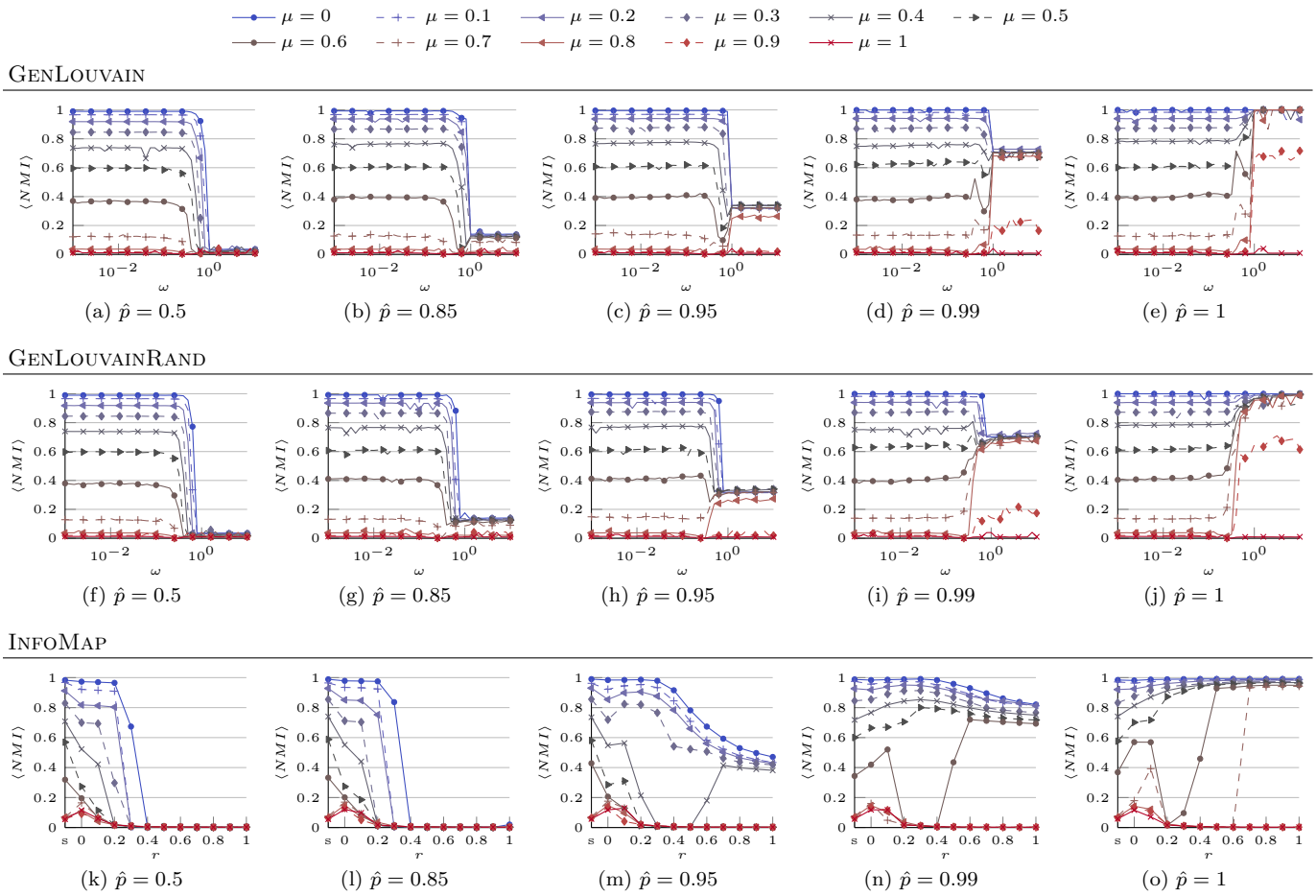
Our goal in this section is to illustrate that one can use our generative model to compare methods and algorithms on different types of multilayer networks. Accordingly, we do not investigate any given method in detail.

### A. Multiplex examples

In Fig. 8, we consider multiplex networks with uniform interlayer dependencies between each pair of layers (see Fig. 3b). In this kind of multilayer network, the probabilities in the interlayer dependency tensor are the same between all pairs of layers. We control the strength of interdependency between induced community structure in different layers using the parameter $\hat{p}$, which is the probability that a state node's community assignment on a given layer is the result of copying rather than being assigned from the null distribution. (Recall that $\hat{p} = (l-1)p$ in Fig. 3b).) We use networks with $n = 1\,000$ physical nodes and $l = 15$ layers. Each node is present on every layer, so there are a total of 15 000 state nodes. We use variants of the Louvain method to detect communities in panels (a)–(j), and we use multilayer InfoMap in panels (k)–(o).

The results in Fig. 8 suggest that none of the algorithms that we test can exploit the interdependencies between community structure in the different layers of the multilayer networks in this example unless the community structure is (almost) identical across layers. We show an instance with $\hat{p} < 1$ in Fig. 8i in which NMI increases as one increases the value of interlayer coupling with GenLouvainRand before collapsing onto other curves. In particular, for $\hat{p} = 0.99$ (see Fig. 8i), we observe that for multilayer networks with $\mu = 0.4$ and $\mu = 0.5$ the NMI increases for a narrow range of $\omega$ values before stabilizing at a value of approximately 0.7. Similarly, using multilayer InfoMap only results in a larger value of the NMI than that obtained with monolayer InfoMap (corresponding to the data point "s" on the horizontal axis) for large values of $\hat{p}$ (e.g., for $\hat{p} = 0.99$ and $\mu = 0.5$ in Fig. 8n). In our experiments, the methods based on multilayer modularity outperform multilayer InfoMap for networks with weak community structure and similar layers (i.e., when both $\mu$ and $\hat{p}$ are close to 1). In particular InfoMap yields an NMI value lower than 0.2 for $\mu > 0.7$ in all five numerical examples (whereas, for example, the NMI is 1 for $\mu = 0.8$ and $\hat{p} = 1$ with GenLouvain and GenLouvainRand for large values of $\omega$).

For GenLouvain, we observe some erratic behavior as the interlayer coupling $\omega$ approaches 1 from below. For values of $\omega$ near 1 but smaller than 1, the GenLouvain algorithm has a tendency to place all state nodes in a single community. By contrast, for $\omega \geq 1$, it identifies partitions that are identical across layers but does not




(a) $\hat{p} = 0.5$ (b) $\hat{p} = 0.85$ (c) $\hat{p} = 0.95$ (d) $\hat{p} = 0.99$ (e) $\hat{p} = 1$

(f) $\hat{p} = 0.5$ (g) $\hat{p} = 0.85$ (h) $\hat{p} = 0.95$ (i) $\hat{p} = 0.99$ (j) $\hat{p} = 1$

(k) $\hat{p} = 0.5$ (l) $\hat{p} = 0.85$ (m) $\hat{p} = 0.95$ (n) $\hat{p} = 0.99$ (o) $\hat{p} = 1$

FIG. 8. Effect of interlayer coupling strength $\omega$ and relaxation rate $r$ on the ability of different community-detection algorithms to recover planted partitions as a function of the mixing parameter $\mu$ in a uniform multiplex benchmark (see Fig. 3b). We parametrize the amount of interlayer dependency in a network by the probability $\hat{p}$ that a state node copies its community assignment from a neighbor in the interlayer dependency network. (Recall that $\hat{p} = (l-1)p$ in Fig. 3b.) For InfoMap, the first data point for each value of $\mu$ is the result of running monolayer InfoMap separately on each layer of the corresponding multilayer network. Each multilayer network has 1000 nodes and 15 layers, and each node is present in all layers. We perform 200 iterations of our update process (see Section III A). We use a Dirichlet null distribution to specify expected community sizes and set $n_c = 10, \theta = 1$, and $q = 1$ (see Section III C). We use the M-DCSBM benchmark (see Section IV) with $\tau_k = -2, k_{min} = 3$, and $k_{max} = 150$ to generate intralayer edges. All results are means over 10 runs of the algorithms on one instantiation of the benchmark network for each value of the parameters.

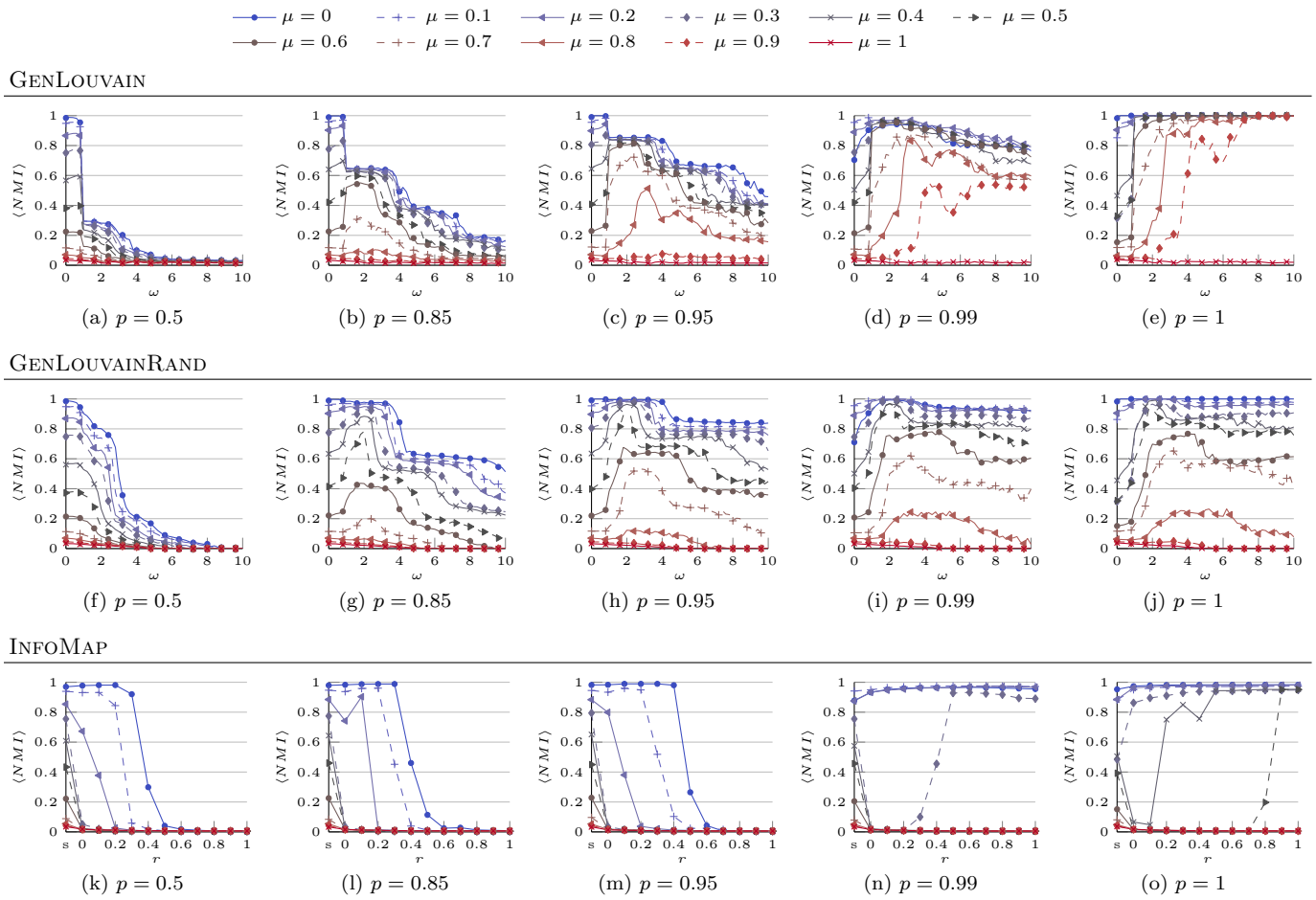
place all state nodes into a single community. This observation is related to the transition behavior described in [20].

### B. Temporal examples

In this section, we show two numerical examples with the interlayer adjacency tensor of Fig 3a: one in which interlayer dependencies between contiguous layers are uniform, and one in which interlayer dependencies between contiguous layers are nonuniform. Importantly, for both case, we show examples in which the employed algorithms can exploit interdependencies between community structure in the different layers of the multilayer network.

In Fig. 9, we examine temporal networks with uniform interlayer dependencies between contiguous layers. That is, $p_\beta = p \in [0, 1]$ for all $\beta \in \{2, \dots, l\}$ in Fig 3a. In all experiments of Fig. 9, we set $(n, l) = (150, 100)$. Additionally, each node is present in all layers, so there are a total of 15 000 state nodes. For this case, our generative model reduces to Algorithm 1. We use variants of the Louvain method to detect communities in panels (a)–(j), and we use multilayer InfoMap in panels (k)–(o).

We make a few remarks about Fig. 9. First, in panels (a) and (f), we observe that increasing the value of the coupling strength $\omega$ does not enhance the recovery of a multilayer planted partition with respect to the case $\omega = 0$ (i.e., when there is no interlayer coupling) for the case $p = 0.5$. Based on visual inspection, increasing the value of $\omega$ starts to help when $p \gtrapprox 0.6$. For example, when $p \in \{0.85, 0.95, 0.99, 1\}$, increasing the value of $\omega$




(a) $p = 0.5$ (b) $p = 0.85$ (c) $p = 0.95$ (d) $p = 0.99$ (e) $p = 1$

(f) $p = 0.5$ (g) $p = 0.85$ (h) $p = 0.95$ (i) $p = 0.99$ (j) $p = 1$

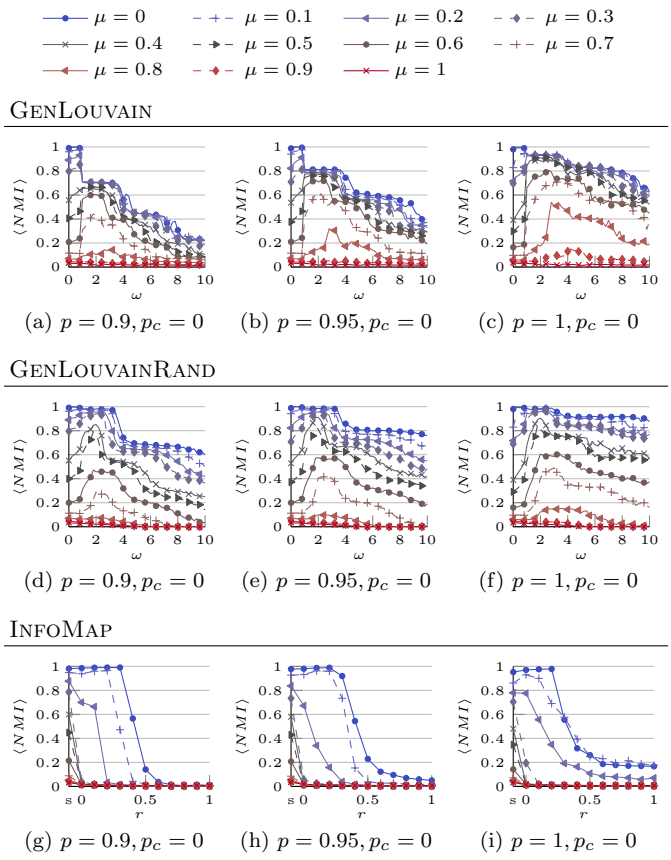
(k) $p = 0.5$ (l) $p = 0.85$ (m) $p = 0.95$ (n) $p = 0.99$ (o) $p = 1$

FIG. 9. Effect of interlayer coupling strength $\omega$ and relaxation rate $r$ on the ability of different community-detection algorithms to recover planted partitions as a function of the mixing parameter $\mu$ in a temporal benchmark with uniform interlayer dependencies (i.e., $p_\beta = p \in [0, 1]$ for all $\beta \in \{2, \ldots, l\}$ in Fig 3a). (Note that $\hat{p} = p$ in Fig 3a.) For InfoMap, the first data point for each value of $\mu$ is the result of running monolayer InfoMap separately on each layer of the corresponding multilayer network. Each multilayer network has 150 nodes and 100 layers, and each node is present in all layers. We use a Dirichlet null distribution to specify expected community sizes and set $n_c = 5, \theta = 1$, and $q = 1$ (see Section III C). We use the M-DCSBM benchmark (see Section IV) with $\tau_k = -2, k_{min} = 3$, and $k_{max} = 30$ to generate intralayer edges. All results are means over 10 runs of the algorithms on one instantiation of the benchmark network for each value of the parameters.

enhances the recovery of a planted partition for most values of $\mu < 1$. See Figs. 9(b)–(e) and Figs. 9(g)–(j). In many cases, the peak of NMI seems to occur for $2 \lessapprox \omega \lessapprox 4$. When $p \neq 1$, one expects NMI to decrease for sufficiently large values of $\omega$, as such values of $\omega$ favor more "persistence" of communities [20] than is the case for the multilayer planted partition. The abrupt change in behavior of GenLouvain near $\omega = 1$ in Figs. 9(a)–(e) is related to the transition behavior described in [20]. In Figs. 9(f)–(j), we show the same examples as those in Fig. 9(a)–(e) using GenLouvainRand.

In Fig. 9(k)–(o), we consider the same five examples using multilayer InfoMap. Observe that increasing the value of the relaxation rate only seems to have an effect when the value of $p$ is close to 1. In particular, for $p \in \{0.5, 0.85, 0.95\}$, the value of NMI in a multilayer setting does not exceed that obtained in a monolayer setting. (The monolayer NMI value is the data point that we label with "s" on the horizontal axis.) For $p = 1$ (i.e., induced partitions are the same across layers), increasing the value of the relaxation rate $r$ enhances the recovery of a planted partition when $\mu \leq 0.5$ but not beyond that value. For multilayer modularity, the recovery is enhanced beyond $\mu = 0.5$ for a few values of $p$ for both GenLouvain and GenLouvainRand (e.g., for $\mu \leq 0.8$ in Figs. 9(c)–(e) and in Figs. 9(h)–(j)).

In Fig. 10, we consider temporal networks with nonuniform interlayer dependencies between contiguous layers. In particular, every 25th layer (i.e., for layers 25, 50, and 75), we set the value of $p_\beta$ in Fig. 3a to $p_c = 0$ (thereby introducing an abrupt change in community structure) and we set all other values of $p_\beta$ in Fig. 3a to a fixed value $p$. As in Fig. 9, we set $(n, l) = (150, 100)$. Additionally, each node is present in all layers, so there are a




(a) $p = 0.9, p_c = 0$ (b) $p = 0.95, p_c = 0$ (c) $p = 1, p_c = 0$

(d) $p = 0.9, p_c = 0$ (e) $p = 0.95, p_c = 0$ (f) $p = 1, p_c = 0$

(g) $p = 0.9, p_c = 0$ (h) $p = 0.95, p_c = 0$ (i) $p = 1, p_c = 0$

FIG. 10. Effect of interlayer coupling strength $\omega$ and relaxation rate $r$ on the ability of different community-detection algorithms to recover planted partitions as a function of the mixing parameter $\mu$ in a temporal benchmark with nonuniform interlayer dependencies. Each multilayer network has 150 nodes and 100 layers, and each node is present in all layers. Every 25th layer (i.e., for layers 25, 50, and 75), we set the value of $p_\beta$ in Fig. 3a to $p_c = 0$ (thereby introducing an abrupt change in community structure) and set all other values of $p_\beta$ in Fig. 3a to $p$. For InfoMap, the first data point for each value of $\mu$ is the result of running monolayer InfoMap separately on each layer of the corresponding multilayer network. Each multilayer network has 150 nodes and 100 layers, and each node is present in all layers. We use a Dirichlet null distribution to specify expected community sizes and set $n_c = 5, \theta = 1$, and $q = 1$ (see Section III C). We use the M-DCSBM benchmark (see Section IV) with $\tau_k = -2, k_{min} = 3$, and $k_{max} = 30$ to generate intralayer edges. All results are means over 10 runs of the algorithms on one instantiation of the benchmark network for each value of the parameters.

total of 15 000 state nodes. We use variants of the Louvain method to detect communities in panels (a)–(j), and we use multilayer InfoMap in panels (k)–(o).

We make a few remarks about Fig. 10. First, we observe that the GenLouvain and GenLouvainRand algorithms significantly exploit interdependencies between community structure in the different layers of the multilayer network in this example for all three considered values of $p$, whereas InfoMap does not significantly exploit interdependencies in this example. In particular, for most $\mu$ values, NMI values in Figs. 10(a)–(f) exceed the NMI value obtained with monolayer modularity (i.e., $\omega = 0$) for some $\omega$ range. In contrast, the NMI values in Fig. 10(g–i) only slightly exceed that obtained with monolayer InfoMap for two values of $\mu$ ($\mu = 0$ and $\mu = 0.1$). Note for GenLouvain and GenLouvainRand that one expects NMI values to decrease for sufficiently large values of $\omega$, as such values of $\omega$ favor more "persistence" of communities [20] than is the case for the multilayer planted partition. As in Fig. 9, in many cases, the peak of NMI for GenLouvain and GenLouvainRand seems to occur for $2 \lessapprox \omega \lessapprox 4$. The abrupt change in behavior of GenLouvain near $\omega = 1$ in Figs. 10(a)–(c) is related to the transition behavior described in [20].

## VI. CONCLUSIONS

In this paper, we introduced a generative model for mesoscale structures in multilayer networks. The three most important features of our model are the following: (1) it includes an explicitly parametrizable tensor $\boldsymbol{P}$ that controls interlayer dependency structure; (2) our model can be used to generate benchmarks for a rather general class of multilayer networks (including, e.g., temporal, multiplex, and multi-aspect multilayer networks); and (3) our approach is modular, as its two main steps (generating a multilayer partition and generating a multilayer network) are carried out successively and can be modified separately. Along with our paper, we provide publicly available code [65] that users can modify to readily incorporate different types of interlayer dependency structures (see Section III A), null distributions (see Section III C), and monolayer network models with a planted partition (see Section IV).

The ability to explicitly specify interlayer dependency structure makes it possible for a user to control which layers depend directly on each other (by deciding which entries in the interlayer dependency tensor are nonzero) and the extent of such dependencies (by varying the magnitude of entries in the interlayer dependency tensor). One can thereby generate multilayer networks with either a single aspect or multiple aspects (e.g., temporal and/or multiplex networks) and vary dependencies between layers from the extreme case in which induced partitions in a planted multilayer partition are the same across layers to the opposite extreme, in which induced partitions in a planted multilayer partition are generated independently for each layer from a null distribution for that layer. To our knowledge, this level of generality is absent from existing generative models for mesoscale structures in multilayer networks, as those models tend to only consider networks with a single aspect (e.g., temporal or multiplex) or limited interlayer dependency structures (e.g., the induced partitions in a planted multilayer partition are the same across all layers). Moreover, to our knowledge, no current generative model for mesoscale structure

in multilayer networks includes an explicit parametrization of interlayer dependency structure.

As we have illustrated, our model is both general and flexible. By using it to generate multilayer networks with specified interdependency structure in different layers, one can (1) gain insight into whether, when, and how to build in such dependencies into methods for studying multilayer networks and (2) generate tunable benchmarks for community-detection (or more generally, "mesoscale-structure-detection") methods for multilayer networks. The explicit separation between our generative model for a multilayer partition and our generative model for a multilayer network with a given planted partition readily allows the incorporation of additional salient features of empirical multilayer networks.

We illustrated our generative model using several simple but important examples. Our goal was to illustrate the use of our model with a few special cases of interest rather than to explicitly discuss as many situations as possible. We focused on community structure because it is a commonly studied mesoscale structure, but one can also use our model to generate mesoscale structures other than community structure in each layer (e.g., core–periphery structure, bipartite structure, and so on) by taking advantage of the flexibility of the degree-corrected SBM. For our examples, we assumed that interlayer dependencies exist either between all contiguous layers (a special case of temporal networks) or between all layers (a special case of multiplex networks). For the case of temporal networks, we considered uniform and nonuniform dependencies, and for the multiplex case, we considered only uniform dependencies. However, our model's flexibility allows us to generate multilayer networks with more realistic features. For example, one can define a "block multiplex" interlayer dependency tensor by introducing dependencies between (say) layers from the set $\{1, \ldots, l/2\}$ and also between layers from the set $\{l/2+1, \ldots, l\}$ but not between a layer from one set and a layer from the other (so that Fig. 3b is a block-diagonal matrix with 0 entries on its two off-diagonal blocks). For temporal networks, on can introduce dependencies between a layer and all layers that follow it (so that Fig. 3a is an upper triangular matrix with nonzero entries above the diagonal) to incorporate memory effects [80]. One can also incorporate "burstiness" [81] in the inter-event-time distribution of edges by modifying our multilayer network model (as opposed to the partition model). In this scenario, the probability for an edge to exist in a given layer depends not only on the induced partition on that layer but also on the existence of the edge in previous layers. (For example, one could use a Hawkes process to specify the time points at which edges are active [91, 92].)

Our model for sampling edges in Section V produces the commonly studied case of a multilayer network with no edges between layers and interdependent connectivity patterns in the different layers. However, other types of multilayer networks are also important [51], and one can readily combine our approach for generating multilayer partitions with different network generating models that capture various important features. For example, one can use an SBM to generate interlayer edges, or one can replace the degree-corrected SBM in Section IV with any other monolayer network model with a planted partition or other interesting models (e.g., other variants of SBMs [26, 32] and models for networks whose structure is affected by space (perhaps spatially embedded) [8, 42] or arbitrary latent features [64]). In all of these examples, interdependencies between connectivity patterns in different layers result only from a planted multilayer partition. It is also possible to modify our network generation process (see Section IV) to introduce additional dependencies between layers beyond that induced by planted mesoscale structure (e.g., by introducing dependencies between a node's degree in different layers [93, 94]).

Our work has the potential for many useful and interesting extensions; we highlight three of these. First, although we have given some illustrative numerical examples in Section V, the area of benchmarking community-detection methods in multilayer networks is very far from being fully developed. Generative models are useful tools for understanding the behavior of community-detection methods in detail and accordingly for suggesting ways of improving heuristic algorithms without losing scalability. Our model can serve as a test bed for gaining insight into the advantages and shortcomings of community-detection methods, and importantly we expect it to be very informative for how they can be used in practice. Second, a well-understood generative model can be a powerful tool for statistical inference (i.e., inferring the structure of a multilayer network rather than generating a multilayer network with a planted structure) [6]. Finally, it is enormously important to model interdependent data streams (not just fixed data sets) and detect their structure in real time. It is critical to develop develop generative models that can be adapted readily to such situations, and our work in this paper is a step in this direction.

## ACKNOWLEDGEMENTS

MB acknowledges a CASE studentship award from the EPSRC (BK/10/41). LGSJ acknowledges a CASE studentship award from the EPSRC (BK/10/39). MB, LGSJ, AA, and MAP were supported by FET-Proactive project PLEXMATH (FP7-ICT-2011-8; grant #317614) funded by the European Commission; and MB, LGSJ, and MAP were also supported by the James S. McDonnell Foundation (#220020177). We thank Sang Hoon Lee and Peter Mucha (and his research group) for helpful comments.

[1] M. E. J. Newman, *Networks: An Introduction* (Oxford University Press, Oxford, UK, 2010).
[2] M. A. Porter, J.-P. Onnela, and P. J. Mucha, Notices Amer. Math. Soc. **56**, 1082 (2009).
[3] S. Fortunato, Phys. Rep. **486**, 75 (2010).
[4] P. Csermely, A. London, L.-Y. Wu, and B. Uzzi, J. Complex Networks **1**, 93 (2013).
[5] R. A. Rossi and N. K. Ahmed, IEEE Trans. Knowl. Data Eng. **27**, 1112 (2015).
[6] S. Fortunato and D. Hric, arXiv:1608.00163 (2016).
[7] M. T. Schaub, J.-C. Delvenne, M. Rosvall, and R. Lambiotte, arXiv:1611.07769 (2016).
[8] D. S. Bassett, E. T. Owens, M. A. Porter, M. L. Manning, and K. E. Daniels, Soft Matter **11**, 2731 (2015).
[9] M. A. Porter, P. J. Mucha, M. E. J. Newman, and C. M. Warmbrand, Proc. Nat. Acad. Sci. U.S.A. **102**, 7057 (2005).
[10] P. J. Mucha, T. Richardson, K. Macon, M. A. Porter, and J.-P. Onnela, Science **328**, 876 (2010).
[11] A. L. Traud, P. J. Mucha, and M. A. Porter, Physica A **391**, 4165 (2012).
[12] M. C. González, H. J. Herrmann, J. Kertész, and T. Vicek, Physica A **379**, 307 (2007).
[13] A. C. F. Lewis, N. S. Jones, M. A. Porter, and C. M. Deane, BMC Syst. Biol. **4**, 100 (2010).
[14] R. Guimerà, M. Sales-Pardo, and L. A. N. Amaral, Nature **443**, 895 (2005).
[15] D. S. Bassett, N. F. Wymbs, M. A. Porter, P. J. Mucha, J. M. Carlson, and S. T. Grafton, Proc. Nat. Acad. Sci. U.S.A. **118**, 7641 (2011).
[16] N. F. Wymbs, D. S. Bassett, P. J. Mucha, M. A. Porter, and S. T. Grafton, Neuron **74**, 936 (2012).
[17] J.-P. Onnela, J. Saramäki, J. Hyvönen, G. Szabó, D. Lazer, K. Kaski, J. Kertész, and A.-L. Barabási, Proc. Nat. Acad. Sci. U.S.A. **104**, 7332 (2007).
[18] P. Expert, T. S. Evans, V. D. Blondel, and R. Lambiotte, Proc. Nat. Acad. Sci. U.S.A. **108**, 7663 (2011).
[19] M. Girvan and M. E. J. Newman, Proc. Nat. Acad. Sci. U.S.A. **99**, 7821 (2002).
[20] M. Bazzi, M. A. Porter, S. Williams, M. McDonald, D. J. Fenn, and S. D. Howison, Multiscale Model. Simul. **14**, 1 (2016).
[21] G. Petri and P. Expert, Phys. Rev. E **90**, 022813 (2014).
[22] R. Lambiotte, J.-C. Delvenne, and M. Barahona, IEEE Trans. Network Sci. Eng. **1**, 76 (2015), (see also the precursor paper at arXiv:0812.1770, 2008).
[23] J.-C. Delvenne, S. N. Yaliraki, and M. Barahona, Proc. Nat. Acad. Sci. U.S.A. **107**, 12755 (2010).
[24] M. De Domenico, A. Lancichinetti, A. Arenas, and M. Rosvall, Phys. Rev. X **5**, 011027 (2015).
[25] M. Rosvall and C. T. Bergstrom, Proc. Nat. Acad. Sci. U.S.A. **105**, 1118 (2008).
[26] P. W. Holland, K. B. Laskey, and S. Leinhardt, Soc. Networks **5**, 109 (1983).
[27] B. Karrer and M. E. J. Newman, Phys. Rev. E **83**, 016107 (2011).
[28] N. Stanley, S. Shai, D. Taylor, and P. J. Mucha, IEEE Trans. Network Sci. Eng. **3**, 95 (2016).
[29] T. P. Peixoto, Phys. Rev. E **92**, 042807 (2015).
[30] I. Kloumann, J. Ugander, and J. Kleinberg, arXiv:1607.03483 (2016).
[31] U. Brandes, D. Delling, M. Gaertler, R. Göke, M. Hoefer, Z. Nikoloski, and D. Wagner, IEEE Trans. Knowl. Data Eng. **20**, 172 (2008).
[32] A. Z. Jacobs and A. Clauset, in *NIPS Workshop on Networks: From Graphs to Rich Data* (2014) arXiv:1411.4070.
[33] L. Peel, D. B. Larremore, and A. Clauset, arXiv:1608.05878 (2016).
[34] A. Decelle, F. Krzakala, C. Moore, and L. Zdeborova, Phys. Rev. Lett. **107**, 065701 (2011).
[35] A. Ghasemian, P. Zhang, A. Clauset, C. Moore, and L. Peel, Phys. Rev. X **6**, 031005 (2016).
[36] D. Taylor, R. S. Caceres, and P. J. Mucha, arXiv:1609.04376 (2016).
[37] A. Condon and R. M. Karp, Random Struct. Algor. **18**, 116 (2000).
[38] A. Lancichinetti, S. Fortunato, and F. Radicchi, Phys. Rev. E **78**, 046110 (2008).
[39] A. Lancichinetti and S. Fortunato, Phys. Rev. E **80** (2009).
[40] L. G. S. Jeub, P. Balachandran, M. A. Porter, P. J. Mucha, and M. W. Mahoney, Phys. Rev. E **91**, 012821 (2015).
[41] G. K. Orman, V. Labatut, and H. Cherifi, Int. J. Web Based Communities **9**, 349 (2013).
[42] M. Sarzynska, E. A. Leicht, G. Chowell, and M. A. Porter, J. Complex Networks **4**, 363 (2015).
[43] P. Holme and J. Saramäki, Phys. Rep. **519**, 97 (2012).
[44] P. Holme, Eur. Phys. J. B **88**, 234 (2015).
[45] E. Valdano, L. Ferreri, C. Poletto, and V. Colizza, Phys. Rev. E **5**, 021005 (2015).
[46] S. J. Cranmer, E. J. Menninga, and P. J. Mucha, Proc. Nat. Acad. Sci. U.S.A. **112**, 11812 (2015), http://www.pnas.org/content/112/38/11812.full.pdf.
[47] R. Gallotti and M. Barthélemy, Sci. Rep. **4**, 6911 (2014).
[48] W. Fan and A. Yeung, Commun. Nonlinear Sci. Numer. Simul. **20**, 1015 (2015).
[49] L. Cantini, E. Medico, S. Fortunato, and M. Caselle, Sci. Rep. **5**, 17386 (2015).
[50] S. Kéfi, V. Miele, E. A. Wieters, S. A. Navarrete, and E. L. Berlow, PLoS Biol. **14**, 1 (2016).
[51] M. Kivelä, A. Arenas, M. Barthélemy, J. P. Gleeson, Y. Moreno, and M. A. Porter, J. Complex Networks **2**, 203 (2014).
[52] S. Boccaletti, G. Bianconi, R. Criado, C. D. Genio, J. Gómez-Gardenes, M. Romance, I. Sendina-Nadal, Z. Wang, and M. Zanin, Phys. Rep. **544**, 1 (2014).
[53] L. G. S. Jeub, M. W. Mahoney, P. J. Mucha, and M. A. Porter, arXiv:1510.05185 (2015), (to appear in Network Science).
[54] J. D. Wilson, J. Palowitch, S. Bhamidi, and A. B. Nobel, arXiv:1610.06511 (2016).
[55] C. Granell, R. K. Darst, A. Arenas, S. Fortunato, and S. Gómez, Phys. Rev. E **92**, 012805 (2015).
[56] X. Zhang, C. Moore, and M. E. J. Newman, arXiv:1607.07570 (2016).
[57] Y. Hulovatyy and T. Milenković, arXiv:1605.01491 (2016).
[58] M. Q. Pasta and F. Zaidi, arXiv:1606.01169 (2016).
[59] S. Paul and Y. Chen, arXiv:1411.1098 (2015).

[60] T. Vallès-Català, F. A. Massucci, R. Guimerà, and M. Sales-Pardo, Phys. Rev. X **6**, 011036 (2016).

[61] P. Barbillon, S. Donnet, E. Lazega, and A. Bar-Hen, J. Roy. Stat. Soc. A (2016), 10.1111/rssa.12193.

[62] S. Paul and Y. Chen, arXiv:1608.00623 (2016).

[63] T. P. Peixoto, Phys. Rev. X **5**, 011033 (2015).

[64] M. E. J. Newman and T. P. Peixoto, Phys. Rev. Lett. **115**, 088701 (2015).

[65] L. G. S. Jeub and M. Bazzi, "A generative model for mesoscale structure in multilayer networks implemented in MATLAB," `https://github.com/MultilayerBenchmark/MultilayerBenchmark/` (2016), version 1.0.

[66] L. Peel and A. Clauset, in *Proceedings of the 29th Conference on Artificial Intelligence* (AAAI Press, Palo Alto, California, USA, 2015) pp. 2914–2920.

[67] M. De Domenico, A. Solè-Ribalta, E. Cozzo, M. Kivelä, Y. Moreno, M. A. Porter, S. Gómez, and A. Arenas, Phys. Rev. X **3**, 041022 (2013).

[68] M. De Domenico, A. Solè-Ribalta, S. Gómez, and A. Arenas, Proc. Nat. Acad. Sci. U.S.A. **111**, 8351 (2014).

[69] S. Gómez, A. Díaz-Guilera, J. Gómez-Gardeñes, C. J. Pérez-Vicente, Y. Moreno, and A. Arenas, Phys. Rev. Lett. **110**, 028701 (2013).

[70] Flattening (or 'matricizing') [51] a tensor $\boldsymbol{T}$ corresponds to writing its entries in matrix form. Consequently, instead of having a tensor $\boldsymbol{T}$ with elements $T_{\boldsymbol{\alpha}}^{\boldsymbol{\beta}}$, one has a matrix $\widetilde{\boldsymbol{T}}$ with entries $\widetilde{T}_{\tilde{\alpha},\tilde{\beta}}$, where there are bijective mappings between the indexes.

[71] T. G. Kolda and B. W. Bader, SIAM Rev. **51**, 455 (2009).

[72] Note that we can use this framework to represent overlapping communities by identifying multiple state nodes in a single layer with the same physical node.

[73] A. E. Gelfand, J. Am. Stat. Assoc. **95**, 1300 (2000).

[74] G. O. Roberts and S. K. Sahu, J. Roy. Stat. Soc. B **59**, 291 (1997).

[75] L. Tierney, Ann. Statist. **22**, 1701 (1994).

[76] M. K. Cowles and B. P. Carlin, J. Am. Stat. Assoc. **91**, 883 (1996).

[77] M. Li, J. H. Badger, X. Chen, S. Kwong, P. Kearney, and H. Zhang, Bioinformatics **7**, 194 (2001).

[78] H. Raiffa and R. Schlaifer, *Applied statistical decision theory*, Wiley Classics Library (Wiley-Interscience, New York, 2000).

[79] A. Agarwal and H. Daumé, Mach. Learn. **81**, 99 (2010).

[80] M. Rosvall, A. V. Esquivel, A. Lancichinetti, J. D. Westand, and R. Lambiotte, Nat. Commun. **5**, 4630 (2014).

[81] R. Lambiotte, L. Tabourier, and J.-C. Delvenne, Eur. Phys. J. B **86**, 320 (2013).

[82] A power law with cutoffs $x_{\min}$ and $x_{\max}$ and exponent $\tau$ is a continuous probability distribution with probability density function

$$p(x) = \begin{cases} C\, x^{-\tau}\,, & x_{\min} \le x \le x_{\max}\,, \\ 0\,, & \text{otherwise}\,, \end{cases}$$

where

$$C = \frac{\tau - 1}{x_{\min}^{-(\tau-1)} - x_{\max}^{-(\tau-1)}}$$

is the normalization constant.

[83] The choice of a Poisson distribution for the number of edges ensures that this approach for sampling edges is approximately consistent with sampling edges independently from Bernoulli distributions with success probabilities given by Eq. (8). The exact distribution for the number of edges is a Poisson-Binomial distribution which is difficult to sample from and is well-approximated by a Poisson distribution if none of the individual edge probabilities are too large.

[84] In the most general case of a directed multilayer network with interlayer edges, one would sample expected layer-$\boldsymbol{\beta}$-specific in-degrees $e_{\boldsymbol{\beta}}^{i,\boldsymbol{\alpha}}$ and out-degrees $e_{i,\boldsymbol{\alpha}}^{\boldsymbol{\beta}}$ for each state node $(i,\boldsymbol{\alpha})$ and layer $\boldsymbol{\beta}$ from appropriate distributions. Given expected layer-specific degrees $\boldsymbol{e}$ and a multilayer partition $\mathcal{S}$, one then constructs the block tensor $\boldsymbol{W}$ and state node parameters $\boldsymbol{\sigma}$ for the M-DCSBM analogously to the special case described in the main text. Let

$$\kappa_{\boldsymbol{\beta}}^{s,\boldsymbol{\alpha}} = \sum_{i \in \mathcal{S}_s|_{\boldsymbol{\alpha}}} e_{\boldsymbol{\beta}}^{i,\boldsymbol{\alpha}}\,, \quad \kappa_{s,\boldsymbol{\alpha}}^{\boldsymbol{\beta}} = \sum_{i \in \mathcal{S}_s|_{\boldsymbol{\alpha}}} e_{i,\boldsymbol{\alpha}}^{\boldsymbol{\beta}}\,, \quad \mathcal{S}_s \in \mathcal{S}\,,$$

be the expected layer-$\boldsymbol{\beta}$-specific in-degree and out-degree of community $s$ in layer $\boldsymbol{\alpha}$., and let

$$w_{\boldsymbol{\alpha}}^{\boldsymbol{\beta}} = \sum_{i \in \mathcal{V}} e_{i,\boldsymbol{\alpha}}^{\boldsymbol{\beta}} = \sum_{i \in \mathcal{V}} e_{\boldsymbol{\alpha}}^{i,\boldsymbol{\beta}}$$

be the expected number of edges from layer $\boldsymbol{\alpha}$ to layer $\boldsymbol{\beta}$ (note the consistency constraint on expected in-degrees and out-degrees), then

$$\sigma_{\boldsymbol{\beta}}^{i,\boldsymbol{\alpha}} = \frac{e_{\boldsymbol{\beta}}^{i,\boldsymbol{\alpha}}}{\kappa_{\boldsymbol{\beta}}^{s,\boldsymbol{\alpha}}}\,, \quad \sigma_{i,\boldsymbol{\alpha}}^{\boldsymbol{\beta}} = \frac{e_{i,\boldsymbol{\alpha}}^{\boldsymbol{\beta}}}{\kappa_{s,\boldsymbol{\alpha}}^{\boldsymbol{\beta}}}\,, \quad s = S_{i,\boldsymbol{\alpha}}\,,$$

and

$$W_{r,\boldsymbol{\alpha}}^{s,\boldsymbol{\beta}} = (1-\mu)\delta(r,s)\frac{\kappa_{s,\boldsymbol{\alpha}}^{\boldsymbol{\beta}} + \kappa_{\boldsymbol{\alpha}}^{s,\boldsymbol{\beta}}}{2} + \mu\frac{\kappa_{r,\boldsymbol{\alpha}}^{\boldsymbol{\beta}}\kappa_{\boldsymbol{\alpha}}^{s,\boldsymbol{\beta}}}{w_{\boldsymbol{\alpha}}^{\boldsymbol{\beta}}}\,.$$

Note that in the directed case, the expected layer specific in-degrees and out-degrees generated by this model do not correspond to the values given by the input parameters $\boldsymbol{e}$ exactly, except when $\mu = 1$.

[85] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, J. Stat. Mech: Theory Exp. **2008**, P10008 (2008).

[86] L. G. S. Jeub, M. Bazzi, I. S. Jutla, and P. J. Mucha, "A generalized Louvain method for community detection implemented in MATLAB," `http://netwiki.amath.unc.edu/GenLouvain` (2011–2016), version 2.1.

[87] B. K. Fosdick, D. B. Larremore, J. Nishimura, and J. Ugander, arXiv:1608.00607 (2016).

[88] We use version 0.18.2 of the multilayer InfoMap code, which is available at `http://mapequation.org/code`.

[89] M. Rosvall and C. T. Bergstrom, Proc. Nat. Acad. Sci. U.S.A. **18**, 7327 (2007).

[90] A. Strehl, J. Ghosh, and C. Cardie, J. Mach. Learn. Res. **3**, 583 (2002).

[91] D. J. Daley and D. Vere-Jones, *An Introduction to the Theory of Point Processes*, 2nd ed., Vol. 1 (Springer, New York, 2003).

[92] A. G. Hawkes, Biometrika **58**, 83 (1971).

[93] V. Nicosia and V. Latora, Phys. Rev. E **92**, 032805 (2015).

[94] K.-M. Lee, J. Y. Kim, W.-K. Cho, K.-I. Goh, and I.-M. Kim, New J. Phys. **14**, 033027 (2012).