Mapping higher-order network flows in memory and multilayer networks with Infomap

Daniel Edler, Ludvig Bohlin, and Martin Rosvall*

Integrated Science Lab, Department of Physics, Umeå University, SE-901 87 Umeå, Sweden
* Correspondence: martin.rosvall@umu.se

Abstract: Comprehending complex systems by simplifying and highlighting important dynamical patterns often require modeling and mapping of higher-order network flows. However, complex systems come in many forms and demand a range of representations, including memory and multilayer networks, which in turn call for versatile community-detection algorithms to reveal important modular regularities in the flows. We show that various higher-order network flow representations can be represented with sparse memory networks, in which the multilevel map equation can identify overlapping and nested modules that capture network flows. We derive the map equation applied to sparse memory networks and describe its search algorithm Infomap, which can exploit the flexibility of sparse memory networks. Together they provide a general solution to reveal overlapping modular patterns in higher-order flows through complex systems.

Keywords: community detection; Infomap; higher-order network flows; overlapping communities; multilayer networks; memory networks

1. Introduction

To connect structure and dynamics in complex systems, researchers model flows on networks with random walkers. Take a social network as an example. In a standard network approach, nodes represent people, links represent contacts, and random walkers moving on the links between the nodes represent messages. This dynamical process corresponds to a first-order Markov model of network flows: an individual receiving a message will randomly forward the message to any contact. That means that an individual who receives two messages, one from a friend and one from a colleague, forwards both messages with the same probability to a friend. In reality, however, people are more likely to reply messages or forward messages from friends to other friends and from colleagues to other colleagues. Accordingly, describing network flows with a first-order Markov model suffers from memory loss and washes out significant dynamical patterns [1–3]. Similarly, aggregating flow pathways from multiple sources, such as Facebook conversation threads among friends and email threads among colleagues in the previous example, into a single network can distort both the topology of the network and the dynamics on the network (Fig. 1a) [4–7]. As a consequence, the actual patterns, such as pervasively overlapping modules in social networks, cannot be identified with community-detection algorithms that operate on first-order network flows [8]. Researchers have therefore developed models and community-detection algorithms that can take advantage of multistep pathways in memory networks [8,9] as well as temporal or multi-source data in multilayer networks (Fig. 1b) [4,7]. But the many useful higher-order network representations to best describe diverse complex systems seem to require a plethora of different community-detection algorithms.

Here we show that describing higher-order network flows with sparse memory networks and identifying modules with long flow persistence times, such as friends who share a hobby or colleagues who work in the same project, provides a general solution to reveal modular patterns in higher-order flows through complex systems. Higher-order network representations discriminate physical nodes, which represent a complex system's objects, from state nodes, which describe a complex system's internal flows. In sparse memory networks, state nodes are not bound to represent, for example, previous steps in memory networks or layers in multilayer networks, but are free to represent abstract states. We show that various higher-order network flow representations, including memory and multilayer networks, can be represented with sparse memory networks. We also provide a detailed derivation of the information-theoretic map equation for identifying hierarchically nested modules with long

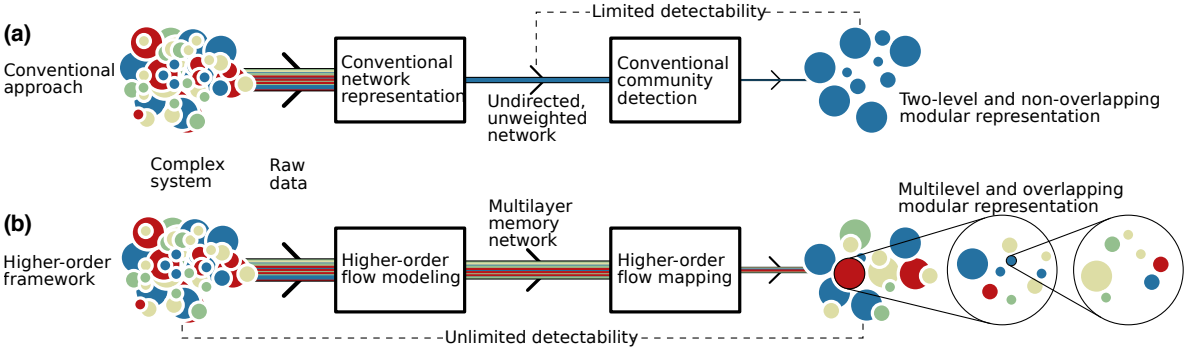

Figure 1. Going beyond standard methods makes it possible to take advantage of richer interaction data. (a) Standard methods shoehorn interaction data about a complex system into an often unweighted and undirected network, which limits what regularities can be detected. (b) Modeling and mapping higher-order network flows can break this detectability limit.

flow persistence times in sparse memory networks, and introduce a new version of the map equation's search algorithm Infomap that exploits the flexibility of sparse memory networks.

2. Modeling network flows

The dynamics and function of complex systems emerge from interdependences between their components [10–12]. Depending on the system under study, the components can be, for example, people, airports, hospital wards, banks, genes, and phenotypes. The interdependence, in turn, often comes from flows of some entity between the components, such as ideas circulating among colleagues, passengers traveling through airports, patients moving between hospital wards, or money transferred between banks. To efficiently capture such flows through complex systems, researchers model them with random walks on networks.

2.1. First-order network flows

In a first-order network representation, the flow direction only depends on the previously visited physical node. That is, for a random walker that moves between physical nodes $i \in \{1,2,\ldots,N\}$, which represent objects in a complex system, and in t steps generates a sequence of random variables $X_1, X_2, \ldots, X_t$, the transition probabilities

$$P(X_t|X_{t-1},X_{t-2},\ldots) = P(X_t|X_{t-1}) \tag{1}$$

only depend on the previously visited node's outlinks. With link weights w_{ij} between physical nodes i and j, and total outlink weights $w_i = \sum_j w_{ij}$, the first-order transition probabilities are

$$P(i \to j) = P_{ij} = \frac{w_{ij}}{w_i}, \tag{2}$$

which give the stationary visit rates

$$\pi_i = \sum_j \pi_j P_{ji}. \tag{3}$$

To ensure ergodic stationary visit rates, from each node we can let the random walker teleport with probability τ, or with probability 1 if the node has no outlinks, to a random target node proportional to the target node's inlink weight [13].

While a first-order model is sufficient to capture flow dynamics in some complex systems, recent studies have shown that higher-order flows are required to capture important dynamics in many complex systems [1,2,14]. The standard approach of modeling dynamical processes on networks with first-order flows oversimplifies the

real dynamics and sets a limit of what can actually be detected in the system (Fig. 1). Overcoming this problem to capture critical phenomena in the dynamics and function of complex systems therefore requires models of higher-order network flows [1–3,14–19].

2.2. Higher-order network flows

In higher-order network representations, the previously visited physical node's outlinks are not sufficient to determine where flows go. Examples of higher-order network representations are memory, multilayer, and temporal networks.

In a memory network, the flow direction depends on multiple previously visited nodes. Specifically, for a random walker that steps between physical nodes and generates a sequence of random variables $X_1,\ldots,X_t$, the transition probabilities for a higher-order flow model of order m,

$$P(X_t|X_{t-1},X_{t-2},\ldots) = P(X_t|X_{t-1},\ldots,X_{t-m}), \tag{4}$$

depend on the m previously visited physical nodes. Assuming stationarity and m visited nodes from x_{-m} m steps ago to previously visited i in sequence $x_{-m}x_{-m+1}\cdots x_{-2}i$, the mth-order transition probabilities between physical nodes i and j correspond to first-order transition probabilities $P(x_{-m}\cdots x_{-2}i \to x_{-m+1}\cdots x_{-2}ij)$ between state nodes $\alpha_i = x_{-m}\cdots x_{-2}i$ and $\beta_j = x_{-m+1}\cdots x_{-2}ij$. We use the optional subscript to highlight the state node's physical node. With link weights $w_{\alpha_i\beta_j}$ between state nodes α_i and β_j, and total outlink weights $w_{\alpha_i} = \sum_{\beta_j} w_{\alpha_i\beta_j}$, the transition probabilities for memory networks are

$$P(x_{-m}\cdots x_{-1}i \to x_{-m+1}\cdots x_{-1}ij) = P_{\alpha_i\beta_j} = \frac{w_{\alpha_i\beta_j}}{w_{\alpha_i}}. \tag{5}$$

As in a first-order network representation, teleportation can ensure ergodic stationary visit rates π_{α_i} [8].

In a multilayer network, the flow direction depends on both the previously visited physical node i and layer $\alpha \in \{1,2,\ldots,l\}$. Similar to the memory representation, the multilayer transition probabilities between physical nodes i in layer α and j in layer β correspond to first-order transition probabilities between states $\alpha_i = \overset{\alpha}{i}$ and $\beta_j = \overset{\beta}{j}$. As for memory networks, we use the optional subscript to highlight the corresponding physical node.

In many cases, available interaction data only contain links within layers. For each layer α, we denote such weighted intralayer links W_{ij}^{α}. Without data with links between layers, it is common to couple layers by allowing a random walker to move between layers at a relax rate r. With probability $1-r$, a random walker follows a link of the physical node in the currently visited layer, and with probability r the random walker relaxes the layer constraint and follows any link of the physical node in any layer. In both cases, the random walker follows a link proportional to its weight. With total intralayer outlink weight $W_i^{\beta} = \sum_j W_{ij}^{\beta}$ from physical node i in layer β and total outlink weight $W_i = \sum_{\beta,j} W_{ij}^{\beta}$ from physical node i across all layers, the transition probabilities for multilayer networks with modeled interlayer transitions are

$$P\left(\overset{\alpha}{i} \to \overset{\beta}{j}\right)(r) = P_{\alpha_i\beta_j}(r) = (1-r)\delta_{\alpha\beta}\frac{W_{ij}^{\beta}}{W_i^{\beta}} + r\frac{W_{ij}^{\beta}}{W_i}. \tag{6}$$

In this way, the transition probabilities capture completely separated layers for relax rate $r = 0$ and completely aggregated layers for relax rate $r = 1$.

With modeled or empirical interlayer links, the transition probabilities for multilayer networks can be written in their most general form,

$$P\left(\overset{\alpha}{i} \to \overset{\beta}{j}\right) = P_{\alpha_i\beta_j} = \frac{w_{\alpha_i\beta_j}}{w_{\alpha_i}}. \tag{7}$$

For directed networks, teleportation can ensure ergodic stationary visit rates π_{α_i}.

2.3. Sparse memory networks

While memory networks and multilayer networks operate on different higher-order interaction data, the resemblance between Eqs. 5 and 7 suggests that they are two examples of a more general network representation. In a memory network model, a random walker steps between physical nodes such that the next step depends on the previous steps (Eq. 5). In a multilayer network model, instead the next step depends on the visited layer (Eq. 7). But as Eqs. 5 and 7 show, both models correspond to first-order transitions between state nodes α_i and β_j associated with physical nodes i and j in the network,

$$P_{\alpha_i\beta_j} = \frac{w_{\alpha_i\beta_j}}{w_{\alpha_i}}. \tag{8}$$

Consequently, the state node visit rates are

$$\pi_{\alpha_i} = \sum_{\beta_j} \pi_{\beta_j} P_{\beta_j\alpha_i}, \tag{9}$$

where the sum is over all state nodes in all physical nodes. The physical node visit rates are

$$\pi_i = \sum_{\beta_j \in i} \pi_{\beta_j}, \tag{10}$$

where the sum is over all state nodes in physical node i. Both memory and multilayer networks can be represented with a network of physical nodes and state nodes that neither are bound to represent previous steps nor current layer. Because state nodes are free to represent abstract states, and redundant state nodes can be lumped together, we call this representation a sparse memory network.

2.4. Representing memory and multilayer networks with sparse memory network

To illustrate that sparse memory networks can represent both memory and multilayer networks, we use a schematic network with five physical nodes. The network represents five individuals, with the center node's two friends to the left and two colleagues to the right in Fig. 2a. We imagine that the multistep pathway data come from two sources, say a Facebook conversation thread among the friends illustrated with the top sequence above the network and the solid pathway on the network, and an email conversation thread among the colleagues illustrated with the bottom sequence and the dashed pathway. For simplicity, the pathway among the friends stays among the friends and steps between friends with equal probability. The pathway among the colleagues behaves in corresponding way. We first represent these pathway data with a memory and a multilayer network, and then show that both can be represented with a sparse memory network.

We can represent multistep pathways with links between state nodes in physical nodes. For a memory network representation of a second-order Markov model, each state node captures which physical node the flows come from. For example, the highlighted pathway step in Fig. 2a corresponds to a link between state node ε_i in physical node i – capturing flows coming to physical node i from physical node j – and state node γ'_k in physical node k – capturing flows coming to physical node k from physical node i (Fig. 2b). In this way, random walker movements between state nodes can capture higher-order network flows between observable physical nodes.

We can represent multistep pathways with links between state nodes in layers. First, we can map all state nodes that represent flows coming from a specific physical node onto the same layer. For example, we map red state nodes ε_i and ε'_k for flows coming from red physical node j to physical nodes i and k, respectively, in Fig. 2b onto the red layer ε at the bottom in Fig. 2c. This mapping gives one-to-one correspondence between the memory network and the multilayer network. Therefore we call it a multilayer memory network.

An alternative and more standard multilayer representation exploits that the pathway data come from two sources, which singly correspond to first-order network flows but combined to higher-order network flows, and uses one layer for each data source (Fig. 2d). The highlighted pathway step in Fig. 2a now corresponds to a step

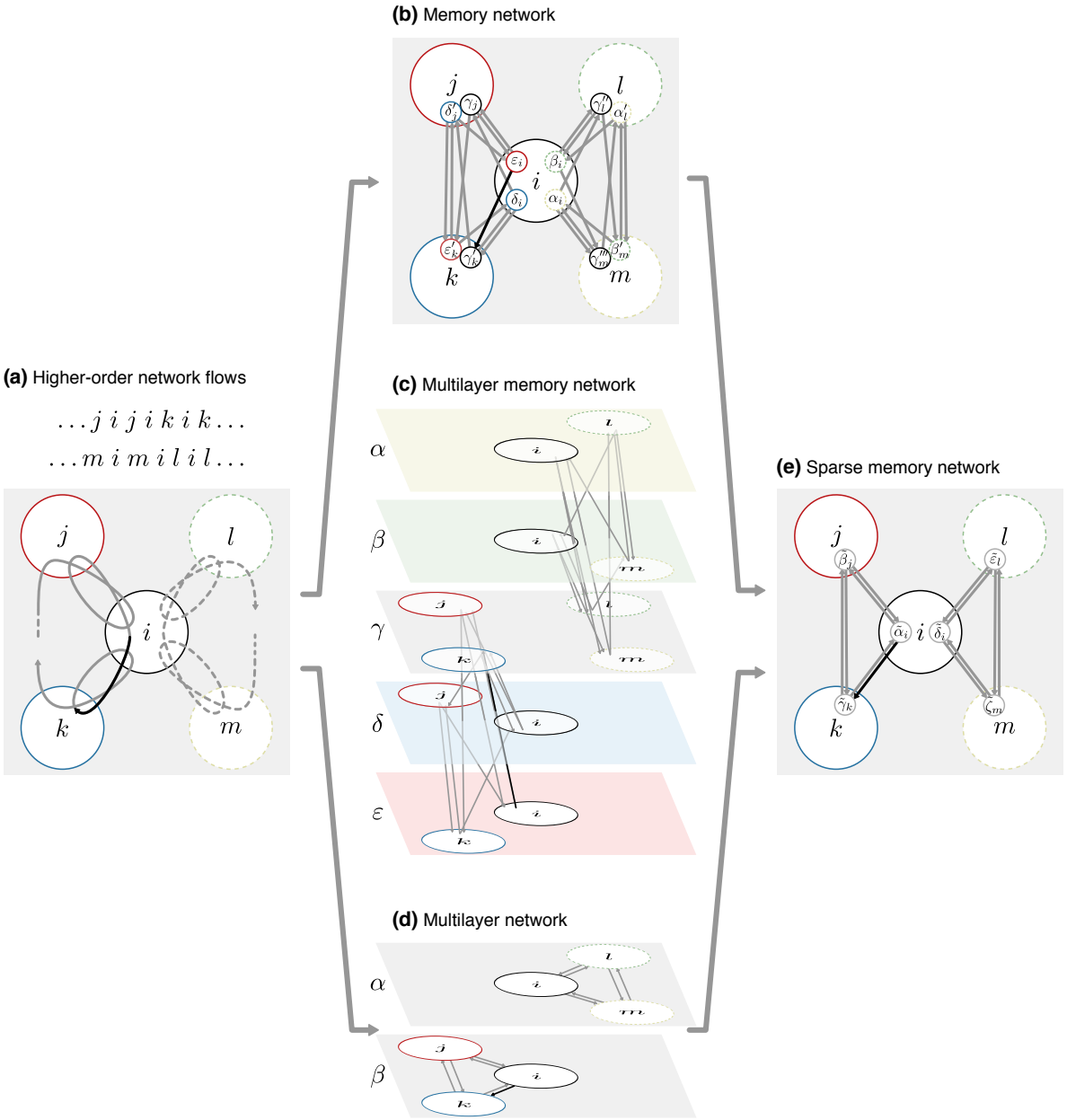

Figure 2. Modeling higher-order network flows with sparse memory networks. (a) Multistep pathways from two sources illustrated on a network with five physical nodes. (b) The pathway data modeled with a second-order Markov model on a memory network, where state nodes capture where flows come from. (c) The memory network represented as a multilayer network where layers, one for each physical node, capture where flows come from. (d) The pathway data modeled on a two-layer network, one layer for each data source. (e) Both memory and multilayer networks mapped on a sparse memory network with no redundant state nodes. The black link highlights the same step in all representations. See also the dynamic storyboard available on www.mapequation.org/apps/sparse-memory-network

in layer β between state node $\overset{\beta}{i}$ and state node $\overset{\beta}{k}$ – capturing flows remaining among friends. Again, random walker movements between state nodes can capture higher-order network flows between observable physical nodes.

In our multistep pathway example from two sources, a full second-order model in the memory network is not required and partly redundant to describe the network flows. We can model the same pathways with a more compact description. Specifically, we can lump together redundant state nodes with identical outlinks in the same physical node. For example, for describing where flows move from physical node i, state nodes ε_i and δ_i have identical outlinks – reflecting that it does not matter which friend who was previously active in the conversation – as well as state nodes β_i and α_i – reflecting that it does not matter which colleague who was previously active in the conversation. Lumping together all such redundant state nodes, such as $\varepsilon_i, \delta_i \to \tilde{\alpha}_i$ or $\beta_i, \alpha_i \to \tilde{\delta}_i$, gives the sparse memory network in Fig. 2e, where state nodes no longer are bound to capture the exact previous steps but still capture the same dynamics as the redundant full second-order model. These unbound state nodes are free to represent abstract states, such as lumped state nodes in a second-order memory network, but can also represent state nodes in a full second-order memory network or state nodes in a multilayer network, such as $\overset{\beta}{i} \to \tilde{\alpha}_i$ in Fig. 2. In fact, the map equation's search algorithm Infomap internally uses sparse memory networks for any higher-order network flow representation. Consequently, rather than an explosion of application and data dependent representations, a representation with physical nodes and state nodes that can represent abstract states in sparse memory networks provide a general and efficient solution for modeling higher-order network flows.

3. Mapping network flows

While networks and higher-order models make it possible to describe flows through complex systems, they remain highly complex even when abstracted to nodes and links. Thousands or millions of nodes and links can bury valuable information. To reveal this information, many times it is indispensable to comprehend the organization of large complex systems by assigning nodes into modules with community-detection algorithms.

3.1. The map equation for first-order network flows

When simplifying and highlighting network flows with possibly nested modules, the map equation measures how well a modular description compresses the flows [20]. Because compressing data is dual to finding regularities in the data [21], minimizing the modular description length of network flows is dual to finding modular regularities in the network flows. For describing movements within and between modules, the map equation uses code books that connects node visits, module exits, and module entries with code words. To estimate the shortest average description length of each code book, the map equation takes advantage of Shannon's source coding theorem and measures the Shannon entropy of the code word use rates [21]. Moreover, the map equation uses a hierarchically nested code structure designed such that the description can be compressed if the network has modules in which a random walker tends to stay for a long time. Therefore, with a random walker as a proxy for real flows, minimizing the map equation over all possible network clusterings reveals important modular regularities in network flows.

In detail, the map equation measures the minimum average description length for a multilevel map M of N physical nodes clustered into M modules, for which each module m has a submap M_m with M_m submodules, for which each submodule mn has a submap M_{mn} with M_{mn} submodules, and so on. In each submodule $mn\ldots o$ at the finest level, the code word use rate for exiting a module is

$$q_{mn\ldots o\curvearrowright} = \sum_{\substack{i\in\mathsf{M}_{mn\ldots o}\\ j\notin\mathsf{M}_{mn\ldots o}}} \pi_i P_{ij}, \tag{11}$$

and the total code word use rate for also visiting nodes in a module is

$$p^{\circlearrowleft}_{mn\ldots o} = q_{mn\ldots o\curvearrowright} + \sum_{i\in\mathsf{M}_{mn\ldots o}} \pi_i, \tag{12}$$

such that the average code word length is

$$H(\mathscr{P}_{mn\ldots o}) = -\frac{q_{mn\ldots o\curvearrowright}}{p^{\circlearrowleft}_{mn\ldots o}} \log \frac{q_{mn\ldots o\curvearrowright}}{p^{\circlearrowleft}_{mn\ldots o}} - \sum_{i\in \mathsf{M}_{mn\ldots o}} \frac{\pi_i}{p^{\circlearrowleft}_{mn\ldots o}} \log \frac{\pi_i}{p^{\circlearrowleft}_{mn\ldots o}}. \tag{13}$$

Weighting the average code word length of the code book for module $mn\ldots o$ at the finest level by its use rate gives the contribution to the description length,

$$L(\mathsf{M}_{mn\ldots o}) = p^{\circlearrowleft}_{mn\ldots o} H(\mathscr{P}_{mn\ldots o}). \tag{14}$$

In each submodule m at intermediate levels, the code word use rate for exiting to a coarser level is

$$q_{m\curvearrowright} = \sum_{\substack{i\in \mathsf{M}_m \\ j\notin \mathsf{M}_m}} \pi_i P_{ij}, \tag{15}$$

and for entering the M_m submodules M_{mn} at a finer level is

$$q_{mn\curvearrowleft} = \sum_{\substack{i\notin \mathsf{M}_{mn} \\ j\in \mathsf{M}_{mn}}} \pi_i P_{ij}, \tag{16}$$

such that the total code rate use in submodule m is

$$q^{\circlearrowleft}_m = q_{m\curvearrowright} + \sum_{n=1}^{M_m} q_{mn\curvearrowleft}, \tag{17}$$

which gives the average code word length

$$H(\mathscr{Q}_m) = -\frac{q_{m\curvearrowright}}{q^{\circlearrowleft}_m} \log \frac{q_{m\curvearrowright}}{q^{\circlearrowleft}_m} - \sum_{n=1}^{M_m} \frac{q_{mn\curvearrowleft}}{q^{\circlearrowleft}_m} \log \frac{q_{mn\curvearrowleft}}{q^{\circlearrowleft}_m}. \tag{18}$$

Weighting the average code word length of the code book for module m at intermediate levels by its use rate, and adding the description lengths of submodules at finer levels in a recursive fashion down to the finest level in Eq. 14, gives the contribution to the description length,

$$L(\mathsf{M}_m) = q^{\circlearrowleft}_m H(\mathscr{Q}_m) + \sum_{n=1}^{M_m} L(\mathsf{M}_{mn}). \tag{19}$$

At the coarsest level, there is no coarser level to exit to, and the code word use rate for entering the M submodules M_{mn} at a finer level is

$$q_{m\curvearrowleft} = \sum_{\substack{i\notin \mathsf{M}_m \\ j\in \mathsf{M}_m}} \pi_i P_{ij}, \tag{20}$$

such that the total code rate use at the coarsest level is

$$q_{\curvearrowleft} = \sum_{m=1}^{M} q_{m\curvearrowleft}, \tag{21}$$

which gives the average code word length

$$H(\mathscr{Q}) = -\sum_{m=1}^{M} \frac{q_{m\curvearrowleft}}{q_{\curvearrowleft}} \log \frac{q_{m\curvearrowleft}}{q_{\curvearrowleft}}. \tag{22}$$

Weighting the average code word length of the code book at the coarsest level by its use rate, and adding the description lengths of submodules at finer levels from Eq. 19 in a recursive fashion, gives the multilevel map equation [22]

$$L(\mathsf{M}) = q_{\curvearrowleft} H(\mathscr{Q}) + \sum_{m=1}^{M} L(\mathsf{M}_m). \tag{23}$$

To find the multilevel map that best represents flows in a network, we seek the multilevel clustering of the network that minimizes the multilevel map equation over all possible multilevel clusterings of the network (see Alg. 6 in Sec. 4).

While there are several advantages with the multilevel description, including potentially better compression and effectively eliminated resolution limit [23], for simplicity researchers often choose two-level descriptions. In this case, there are no intermediate submodules and the two-level map equation is

$$L(\mathsf{M}) = q_{\curvearrowleft} H(\mathscr{Q}) + \sum_{m=1}^{M} p_m^{\circlearrowleft} H(\mathscr{P}_m). \tag{24}$$

3.2. The map equation for higher-order network flows

The map equation for first-order network flows measures the description length of a random walker stepping between physical nodes within and between modules. This principle remains the same also for higher-order network flows, although higher-order models guide the random walker between physical nodes with the help of state nodes. Therefore, extending the map equation to higher-order network flows, including those described by memory, multilayer, and sparse memory networks, is straightforward. Equations 11 to 24 remain the same with $i \to \alpha_i$ and $j \to \beta_j$. The only difference is at the finest level. State nodes of the same physical node assigned to the same module should share code word, or they would not represent the same object. That is, if multiple state nodes β_j of the same physical node i are assigned to the same module $mn\ldots o$, we first sum their probabilities to obtain the visit rate of physical node i in module $mn\ldots o$,

$$\pi_{i\cap mn\ldots o} = \sum_{\beta_j \in i \cap \mathsf{M}_{mn\ldots o}} \pi_{\beta_j}. \tag{25}$$

In this way, the frequency weighted average code word length in submodule codebook $mn\ldots o$ in Eq. 13 becomes

$$H(\mathscr{P}_{mn\ldots o}) = -\frac{q_{mn\ldots o\curvearrowright}}{p^{\circlearrowleft}_{mn\ldots o}} \log \frac{q_{mn\ldots o\curvearrowright}}{p^{\circlearrowleft}_{mn\ldots o}} - \sum_{i \in \mathsf{M}_{mn\ldots o}} \frac{\pi_{i\cap mn\ldots o}}{p^{\circlearrowleft}_{mn\ldots o}} \log \frac{\pi_{i\cap mn\ldots o}}{p^{\circlearrowleft}_{mn\ldots o}}, \tag{26}$$

where the sum is over all physical nodes that have state nodes assigned to module $\mathsf{M}_{mn\ldots o}$. In this way, the map equation can measure the modular description length of state-node-guided higher-order flows between physical nodes.

To illustrate that the separation between physical nodes and state nodes matters when clustering higher-order network flows, we cluster the red state nodes and the dashed blue state nodes in Fig. 3 in two different modules overlapping in the center physical node. For a more illustrative example, we also allow transitions between red and blue nodes at rate $r/2$ in the center physical node. In the memory and sparse memory networks, the transitions correspond to links from state nodes in physical node i to state nodes in the other module with relative weight $r/2$ and to the same module with relative weight $1 - r/2$ (Figs. 3b and c). In the multilayer network, the transitions correspond to relax rate r according to Eq. 6, since relaxing to any layer in physical node i with equal

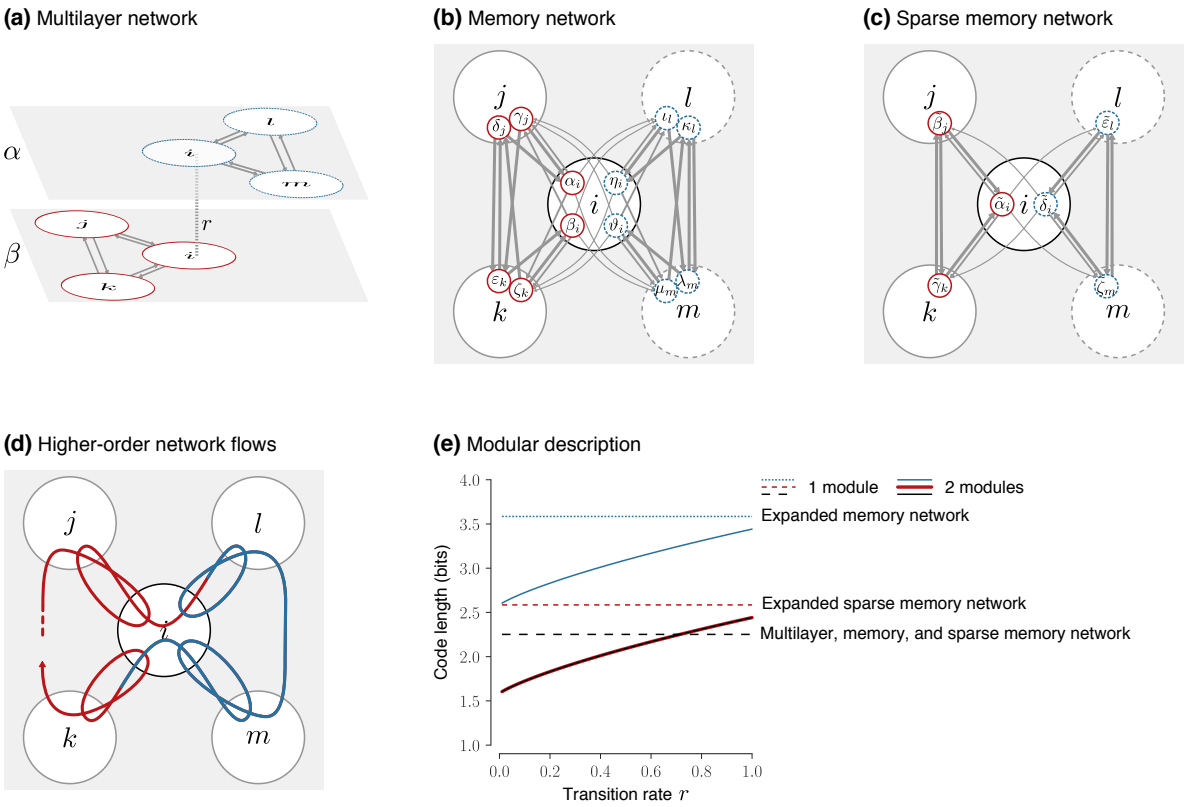

Figure 3. Mapping higher-order network flows. The same network flows represented with a multilayer network with relax rate r in (a), a second-order memory network in (b), and a sparse memory network in (c). (d) Sample of corresponding higher-order flow pathways. (e) Code length as a function of the transition rate for all representation as well as extended networks with virtual physical nodes.

link weights in both layers means that one half of relaxed flows switch layer (Fig. 3a). Independent of the relax rate, in these symmetric networks the node visit rates are uniformly distributed: $\pi_{\alpha_i} = 1/6$ for each of the six state nodes in the multilayer and sparse memory networks, and $\pi_{\alpha_i} = 1/12$ for each of the twelve state nodes in the memory network. For illustration, if we incorrectly treat state nodes as physical nodes in the map equation, Eqs. 11 to 24, the one-module clustering of the memory network with twelve virtual physical nodes, $\mathsf{M}_1^{12\mathrm{p}}$, has code length

$$L(\mathsf{M}_1^{12\mathrm{p}}) = \underbrace{\frac{12}{12}}_{p_1^{\circlearrowleft}} \underbrace{H\left(\frac{1}{12},\frac{1}{12},\frac{1}{12},\frac{1}{12},\frac{1}{12},\frac{1}{12},\frac{1}{12},\frac{1}{12},\frac{1}{12},\frac{1}{12},\frac{1}{12},\frac{1}{12}\right)}_{H(\mathscr{P}_1) \text{ with } H(p_1,p_2,\ldots)=-\sum_i \frac{p_i}{\sum_j p_j} \log \frac{p_i}{\sum_j p_j}} \approx 3.58 \text{ bits}. \tag{27}$$

The corresponding two-module clustering of the memory network with twelve virtual physical nodes, $\mathsf{M}_2^{12\mathrm{p}}$, has code length

$$L(\mathsf{M}_2^{12\mathrm{p}}) = \underbrace{\frac{r}{6}}_{q_{\curvearrowleft}} \underbrace{H\left(\frac{r}{12},\frac{r}{12}\right)}_{H(\mathscr{Q})} + 2\underbrace{\frac{6+r}{12}}_{p_m^{\circlearrowleft}} \underbrace{H\left(\frac{1}{12},\frac{1}{12},\frac{1}{12},\frac{1}{12},\frac{1}{12},\frac{1}{12},\frac{r}{12},\frac{r}{12}\right)}_{H(\mathscr{P}_m)} \approx [2.58, 3.44] \text{ for } r \in [0,1]. \tag{28}$$

Therefore, the two-module clustering gives best compression for all relax rates (Fig. 3e). For the sparse memory network with six virtual physical nodes, the one-module clustering, $\mathsf{M}_1^{6\mathrm{p}}$, has code length

$$L(\mathsf{M}_1^{6\mathrm{p}}) = \frac{6}{6} H\left(\frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}\right) \approx 2.58, \tag{29}$$

and the two-module clustering, $\mathsf{M}_2^{6\mathrm{p}}$, has code length

$$L(\mathsf{M}_2^{6\mathrm{p}}) = \frac{r}{6} H\left(\frac{r}{12}, \frac{r}{12}\right) + 2\frac{6+r}{12} H\left(\frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{r}{12}\right) \approx [1.58, 2.44] \text{ for } r \in [0,1]. \tag{30}$$

While the two-module clustering again gives best compression for all relax rates, the code lengths are shifted by 1 bit compared with the memory network with twelve virtual physical nodes (Fig. 3e). Same dynamics but different code length. These expanded solutions with virtual physical nodes do not capture the important and special role that physical nodes play as representatives of a system's objects.

Properly separating state nodes and physical nodes as in Eq. 26 instead gives equal code lengths for identical clusterings irrespective of representation. For example, the one-module clustering of the memory network with twelve state nodes, $\mathsf{M}_1^{5\mathrm{p}12\mathrm{s}}$, and the sparse memory network with six state nodes, $\mathsf{M}_1^{5\mathrm{p}6\mathrm{s}}$, have identical code length

$$L(\mathsf{M}_1^{5\mathrm{p}12\mathrm{s}}) = L(\mathsf{M}_1^{5\mathrm{p}6\mathrm{s}}) = H\left(\frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{2}{6}\right) \approx 2.25, \tag{31}$$

because visits to a physical node's state nodes in the same module are aggregated according to Eq. 25 such that those state nodes share code words and the encoding represents higher-order flows between a system's objects. Similarly, the two-module clustering of the memory network with twelve state nodes, $\mathsf{M}_2^{5\mathrm{p}12\mathrm{s}}$, and the sparse memory network with six state nodes, $\mathsf{M}_2^{5\mathrm{p}6\mathrm{s}}$, have identical code length

$$L(\mathsf{M}_2^{5\mathrm{p}12\mathrm{s}}) = L(\mathsf{M}_2^{5\mathrm{p}6\mathrm{s}}) = \frac{r}{6} H\left(\frac{r}{12}, \frac{r}{12}\right) + 2\frac{6+r}{12} H\left(\frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{r}{12}\right) \approx [1.58, 2.44] \text{ for } r \in [0,1], \tag{32}$$

That is, same dynamics give same code length for identical clusterings with proper separation of state nodes and physical nodes.

For these solutions with physical nodes and state nodes that properly capture higher-order network flows, the overlapping two-module clustering gives best compression with relax rate $r \lessapprox 0.71$ (Fig. 3e). In this example, the one-module clustering can for sufficiently high relax rate better compress the network flows than the two-module clustering. Compared with the expanded clusterings with virtual physical nodes where this cannot happen, the one-module clustering gives a relatively shorter code length thanks to code word sharing between state nodes in physical node i. In general, modeling higher-order network flows with physical nodes and state nodes, and accounting for them when mapping the network flows, gives overlapping modular clusterings that do not depend on the particular representation but only on the actual dynamical patterns.

4. Infomap

To find the best modular description of network flows according to the map equation, we have developed the stochastic and fast search algorithm called Infomap. Infomap can operate on standard, multilayer, memory, and sparse memory networks with unweighted or weighted and undirected or directed links, and identify two-level or multilevel and non-overlapping or overlapping clusterings. Infomap takes advantage of parallelization with OpenMP and there is also a distributed version implemented with GraphLab PowerGraph for standard networks [24]. In principle, the search algorithm can use other objective functions and find other types of community structures.

Recent versions of Infomap operate on physical nodes and state nodes. Each state node has a unique state id and is assigned to a physical id, which can be the same for many state nodes. Infomap only uses physical nodes and state nodes for higher-order network flow representations, such as multilayer, memory, and sparse memory networks, and physical nodes alone when they are sufficient to represent first-order network flows in standard networks.

To balance accuracy and speed, Infomap uses a number of repeatedly and recursively applied stochastic subroutine algorithms. For example, while the repeated node aggregation (Alg. 1) gives a good two-level clustering, additional fine-tuning (Alg. 4) and coarse-tuning (Alg. 5) together with complete restarts to avoid local minima can significantly improve the accuracy.

4.1. Repeated node aggregation

The repeated node aggregation in Alg. 1 follows closely the Louvain method [25]: Infomap aggregates neighboring nodes into modules, subsequently aggregates them into larger modules, and repeats. First, Infomap assigns each node to its own module. Then, in random sequential order, it moves each node to the neighboring or a new module that gives the optimal move with largest code length decrease (Alg. 2). If no move gives a code length decrease, the node stays in its original module. Infomap repeats this procedure, each time in a new random sequential order, until no move happens. Then Infomap rebuilds the network with the modules of the last level forming the nodes at the new level, and, exactly as in the previous level, aggregates the nodes into modules, which replace the existing ones. Infomap repeats this hierarchical network rebuilding until the map equation cannot be further reduced.

Algorithm 1 Repeated node aggregation

Require: a network of nodes
repeat
    Put each node in its own module
    repeat
        for node i in random order **do**
            Run Optimal move (Alg. 2) and make the move that decreases the code length the most
    until code length decrease is less than a small threshold **or** has reached a maximum number of iterations
    if code length decrease is less than a small threshold **then**
        Break
    aggregate nodes in modules to new network of nodes
until code length decrease is less than a small threshold
Ensure: a non-trivial two-level network clustering

Algorithm 2 Optimal move

Require: a node i with links to neighboring nodes
for neighboring node n **do**
    Calculate change in *enter* and *exit* flows for the current and neighboring node
    For state nodes, calculate change in physical node visit rates (Eq. 25)
Calculate change in *enter* and *exit* flow if moving to a new module
The change in code length for each possible move can be locally calculated only based on the change in *enter* and *exit* flows, the current flow in the moving node and affected modules, and the initial *enter* flow for the module. For state nodes, the physical node visit rates can be locally updated.

4.2. Two-level clustering

The complete two-level clustering algorithm improves the repeated node aggregation algorithm by breaking up modules of sub-optimally aggregated nodes. That is, once the algorithm assigns nodes to the same module, they are forced to move jointly when Infomap rebuilds the network, and what was an optimal move early in the algorithm might have the opposite effect later in the algorithm. Similarly, two or more modules that merge together and form a single module when the algorithm rebuilds the network can never be separated again in the repeated node aggregation algorithm. Therefore, the accuracy can be improved by breaking up modules of the final state of the repeated node aggregation algorithm and trying to move individual nodes or smaller submodules into new or neighboring modules. The two-level clustering algorithm (Alg. 3) performs this refinement by iterating fine-tuning (Alg. 4) and coarse-tuning (Alg. 5).

Algorithm 3 Two-level clustering

Require: a network of nodes
1 Run repeated node aggregation (Alg. 1)
2 **repeat**
3 Run fine-tuning (Alg. 4) and coarse-tuning (Alg. 5) alternately
4 **until** code length decrease is less than a small threshold **or** has reached a maximum number of iterations

4.3. Fine-tuning

In the fine-tuning step, Infomap tries to move single nodes out from existing modules into new or neighboring modules (Alg. 4). First, Infomap reassigns each node to be the sole member of its own module to allow for single-node movements. Then it moves all nodes back to their respective modules of the previous step. At this stage, with the same clustering as in the previous step but with each node free to move between the modules, Infomap reapplies the repeated node aggregation (Alg. 1).

Algorithm 4 Fine-tuning

1 Replace existing modules with each node in its separate module
2 Merge nodes into the former modules as an initial solution
3 Try to move the nodes to new or neighboring modules to improve the code length with repeated node aggregation (Alg. 1).

4.4. Coarse-tuning

In the course-tuning step, Infomap tries to move submodules out from existing modules into new or neighboring modules (Alg. 5). First, Infomap treats each module as a network on its own and applies repeated node aggregation (Alg. 1) on this network. This procedure generates one or more submodules for each module. Infomap then replaces the modules by the submodules and moves the submodules into their modules as an initial solution. At this stage, with the same clustering as in the previous step but with each submodule free to move between the modules, Infomap reapplies the repeated node aggregation (Alg. 1).

Algorithm 5 Coarse-tuning

1 Cluster each module to submodules
2 Replace the existing modules with the submodules
3 Merge submodules into the former modules as an initial solution
4 Try to move the submodules to new or neighboring modules to improve the code length with repeated node aggregation (Alg. 1).

4.5. Multilevel clustering

Infomap identifies multilevel clusterings by extending two-level clusterings both by iteratively finding supermodules of modules and by recursively finding submodules of modules. To find the optimal multilevel solution, Infomap first tries to iteratively find supermodules and then recursively tries to cluster those modules until the code length cannot be further improved (Alg. 6).

Algorithm 6 Multilevel clustering

```
1 Run two-level clustering (Alg. 3)
2 Run Supermodule clustering (Alg. 7)
3 Run Submodule clustering (Alg. 8)
```

To identify a hierarchy of supermodules from a two-level clustering, Infomap first tries to find a shorter description of flows between modules. It iteratively runs the two-level clustering algorithm (Alg. 3) on a network with one node for each module at the coarsest level, node-visit rates from the module-entry rates, and links that describe aggregated flows between the modules (Alg. 7). For each such step, a two-level code book replaces the coarsest code book. In this way, describing entries into, within, and out from supermodules at a new coarsest level replaces only describing entries into the previously coarsest modules.

Algorithm 7 Supermodule clustering

```
1 Create a new network with one node for each module at the coarsest level, node-visit rates from the
  module-entry rates, and links that describe aggregated flows between the modules
2 Run the two-level clustering (Alg. 3) on the network to find supermodules
3 if non-trivial solution found then
4     Add those supermodules to the existing solution to increase the hierarchical level by one
5     Repeat from 1
6 else
7     Quit
```

For a fast multilevel clustering, Infomap can keep this hierarchy of supermodules and try to recursively cluster the bottom modules into submodules with Alg. 8. But as this hierarchy of supermodules is constrained by the initial optimal two-level clustering, discarding everything but the top supermodules and starting the recursive submodule clustering from them often identifies a better multilevel clustering. If Infomap finds non-trivial submodules, it collects them in a queue and processes them in parallel for each hierarchical level to efficiently explore the hierarchy with breadth-first search. To find submodules, Infomap runs the multilevel clustering (Alg.6 on the network within each module, which iteratively and recursively generates new supermodule and submodule clustering calls at different levels.

Algorithm 8 Submodule clustering

```
1  Put the top modules in a queue Q and remove existing submodules if any
2  while Q not empty do
3      Create empty queue R to collect submodules
4      for each module in Q (in parallel) do
5          Generate network with links only within the module
6          Run multilevel clustering (Alg. 6) on network
7          if non-trivial solution found then
8              for each module do
9                  Add module to queue R
10     Q ← R
```

4.6. Download Infomap

Infomap can be downloaded from www.mapequation.org, which contains installation instructions, a complete list of running options, and examples for how to run Infomap stand-alone or with other software, including igraph, Python, Jupyter, and R. In the Appendix, we provide Infomap's basic syntax (Appendix A.1) and examples for how to cluster a multilayer network (Appendix A.2), a memory network (Appendix A.3), and a sparse memory network (Appendix A.4). The website also contains interactive storyboards to explain the mechanics of the map equation, and interactive visualizations to simplify large-scale networks and their changes over time.

5. Conclusions

The map equation applied to sparse memory networks provides a general solution to reveal modular patterns in higher-order flows through complex systems. Rather than multiple community-detections algorithms for a range of network representations, the map equation's search algorithm Infomap can be applied to higher-order network flows described by sparse memory networks. A sparse memory network uses abstract state nodes to describe higher-order dynamics and physical nodes to represent a system's objects. This distinction makes all the difference. The flexible sparse memory network can efficiently describe higher-order network flows of various types. Simplifying and highlighting important patterns in these flows with the map equation and Infomap open for more effective analysis of complex systems.

Acknowledgments: We are grateful to Christian Persson and Manlio De Domenico for helpful discussions. M.R. was supported by the Swedish Research Council grant 2016-00796.

Author Contributions: All authors conceived the project and wrote the paper; D.E. wrote the code.

Conflicts of Interest: The authors declare no conflict of interest.

Appendix. Running Infomap

Appendix A.1. Infomap syntax

To run Infomap on a network, use the command

```
./Infomap [options] network_data dest
```

The option `network_data` should point to a valid network file and `dest` to a directory where Infomap should write the output files. If no option is given, Infomap will assume an undirected network and try to cluster it hierarchically. Optional arguments, including directed or undirected links, two-level or multilevel solutions, or various input formats, can be put anywhere. Run `./Infomap --help` for a list and explanation of supported arguments.

Appendix A.2. Clustering a multilayer network

The multilayer network in Fig. 3a can be described with the multilayer network format in `fig3a.net` below and clustered for relax rate $r = 0.4$ with the command

```
./Infomap --input-format multilayer --multilayer-relax-rate 0.4 fig3a.net .
```

See Appendix A.5 for the overlapping clustering output, and Appendix A.4 for an alternative representation with a sparse memory network. In fact, Infomap internally represents the multilayer network in `fig3a.net` for $r = 0.4$ with the sparse memory network in Appendix A.4 with transition rates $r/2$ between state nodes in different layers, since $r/2$ stays among state nodes in the same layer in this symmetric two-layer network.

```
# fig3a.net - Multilayer network
# Lines starting with # are ignored
*Vertices 5
#physicalId name
1 "i"
2 "j"
3 "k"
4 "l"
5 "m"
*Intra
#layerId physicalId physicalId weight
1 1 4 1
1 4 1 1
1 1 5 1
1 5 1 1
1 4 5 1
```

```
1 5 4 1
2 1 2 1
2 2 1 1
2 1 3 1
2 3 1 1
2 2 3 1
2 3 2 1
```

The inter-layer links will be generated based on the multilayer relax rate, but could also be modeled explicitly with an `*Inter` section as below

```
*Inter
#layerId physicalId layerId weight
1 1 2 1
2 1 1 1
```

Appendix A.3. Clustering a memory network

The memory network in Fig. 3b with transition rates corresponding to relax rate $r = 0.4$ for the multilayer network in Fig. 3a, can be described with the memory network format in `fig3b.net` below and clustered with the command

```
./Infomap --input-format memory fig3b.net .
```

See Appendix A.5 for the overlapping clustering output, and Appendix A.4 for an alternative representation with a sparse memory network.

```
# fig3b.net - memory network
# Lines starting with # are ignored
*Vertices 5
#physicalId name
1 "i"
2 "j"
3 "k"
4 "l"
5 "m"
*3grams
#from through to weight
2 1 2 0.8
2 1 3 0.8
2 1 4 0.2
2 1 5 0.2
3 1 2 0.8
3 1 3 0.8
3 1 4 0.2
3 1 5 0.2
1 2 1 1
1 2 3 1
3 2 1 1
3 2 3 1
2 3 1 1
2 3 2 1
1 3 1 1
1 3 2 1
4 1 4 0.8
4 1 5 0.8
4 1 2 0.2
```

```
4 1 3 0.2
5 1 4 0.8
5 1 5 0.8
5 1 2 0.2
5 1 3 0.2
1 4 1 1
1 4 5 1
5 4 1 1
5 4 5 1
1 5 1 1
1 5 4 1
4 5 1 1
4 5 4 1
```

Appendix A.4. Clustering a sparse memory network

The sparse memory network in Fig. 3c with transition rates corresponding to relax rate $r = 0.4$ for the multilayer network in Fig. 3a, can be described with the sparse memory network format in `fig3c.net` below and clustered with the command

```
./Infomap --input-format sparse fig3c.net .
```

See Appendix A.5 for the overlapping clustering output.

```
# fig3c.net - Sparse memory network
# Lines starting with # are ignored
*Vertices 5
#physicalId name
1 "i"
2 "j"
3 "k"
4 "l"
5 "m"
*States
#stateId physicalId name
1 1 "alpha~_i"
2 2 "beta~_j"
3 3 "gamma~_k"
4 1 "delta~_i"
5 4 "epsilon~_l"
6 5 "zeta~_m"
*Links
#sourceStateId targetStateId weight
1 2 0.8
1 3 0.8
1 5 0.2
1 6 0.2
2 1 1
2 3 1
3 1 1
3 2 1
4 5 0.8
4 6 0.8
4 2 0.2
4 3 0.2
5 4 1
5 6 1
```

```
6 4 1
6 5 1
```

For reference, the memory network in Appendix A.3 can also be represented with the sparse memory network format in `fig3b_sparse.net` below and clustered with the command

```
./Infomap --input-format sparse fig3b_sparse.net .
```

See Appendix A.5 for the overlapping clustering output.

```
# fig3b_sparse.net - state network
# Lines starting with # are ignored
*Vertices 5
#physicalId name
1 "i"
2 "j"
3 "k"
4 "l"
5 "m"
*States
#stateId physicalId name
1 1 "alpha_i"
2 1 "beta_i"
3 2 "gamma_j"
4 2 "delta_j"
5 3 "epsilon_k"
6 3 "zeta_k"
7 1 "eta_i"
8 1 "theta_i"
9 4 "iota_l"
10 4 "kappa_l"
11 5 "lambda_m"
12 5 "mu_m"
*Links
#sourceStateId targetStateId weight
1 3 0.8
1 6 0.8
1 9 0.2
1 12 0.2
2 3 0.8
2 6 0.8
2 9 0.2
2 12 0.2
3 1 1
3 5 1
4 1 1
4 5 1
5 2 1
5 4 1
6 2 1
6 4 1
7 9 0.8
7 12 0.8
7 3 0.2
7 6 0.2
8 9 0.8
8 12 0.8
```

```
8 3 0.2
8 6 0.2
9 7 1
9 11 1
10 7 1
10 11 1
11 8 1
11 10 1
12 8 1
12 10 1
```

Appendix A.5. Clustering output

All examples in Appendices A.2–A.4 give the same overlapping physical node clustering below.

```
# A tree output from running Infomap on the example networks in Fig. 3
# path flow name physicalId:
1:1 0.166667 "i" 1
1:2 0.166667 "j" 2
1:3 0.166667 "k" 3
2:1 0.166667 "i" 1
2:2 0.166667 "l" 4
2:3 0.166667 "m" 5
```

With the optional argument `--expanded`, Infomap gives the clustering of each state node (not shown).

References

1. Belik, V.; Geisel, T.; Brockmann, D. Natural human mobility patterns and spatial spread of infectious diseases. *Phys. Rev. X* **2011**, *1*, 011001.
2. Pfitzner, R.; Scholtes, I.; Garas, A.; Tessone, C.J.; Schweitzer, F. Betweenness preference: quantifying correlations in the topological dynamics of temporal networks. *Phys. Rev. Lett.* **2013**, *110*, 198701.
3. Poletto, C.; Tizzoni, M.; Colizza, V. Human mobility and time spent at destination: Impact on spatial epidemic spreading. *J. Theor. Biol.* **2013**.
4. Mucha, P.J.; Richardson, T.; Macon, K.; Porter, M.A.; Onnela, J.P. Community structure in time-dependent, multiscale, and multiplex networks. *Science* **2010**, *328*, 876–878.
5. Kivelä, M.; Arenas, A.; Barthelemy, M.; Gleeson, J.P.; Moreno, Y.; Porter, M.A. Multilayer Networks. *J. Comp. Netw.* **2014**, *2*, 203–271.
6. Boccaletti, S.; Bianconi, G.; Criado, R.; Del Genio, C.; Gómez-Gardeñes, J.; Romance, M.; Sendiña-Nadal, I.; Wang, Z.; Zanin, M. The structure and dynamics of multilayer networks. *Phys. Rep.* **2014**, *544*, 1–122.
7. De Domenico, M.; Lancichinetti, A.; Arenas, A.; Rosvall, M. Identifying modular flows on multilayer networks reveals highly overlapping organization in interconnected systems. *Phys. Rev. X* **2015**, *5*.
8. Rosvall, M.; Esquivel, A.V.; Lancichinetti, A.; West, J.D.; Lambiotte, R. Memory in network flows and its effects on spreading dynamics and community detection. *Nat. Commun.* **2014**, *5*.
9. Peixoto, T.P.; Rosvall, M. Modeling sequences and temporal networks with dynamic community structures. *arXiv:1509.04740* **2017**.
10. Barabási, A.; Albert, R. Emergence of scaling in random networks. *Science* **1999**, *286*, 509–512.
11. Barrat, A.; Barthelemy, M.; Pastor-Satorras, R.; Vespignani, A. The architecture of complex weighted networks. *Proc. Natl. Acad. Sci. USA* **2004**, *101*, 3747.
12. Boccaletti, S.; Latora, V.; Moreno, Y.; Chavez, M.; Hwang, D.U. Complex networks: Structure and dynamics. *Physics reports* **2006**, *424*, 175–308.
13. Lambiotte, R.; Rosvall, M. Ranking and clustering of nodes in networks with smart teleportation. *Phys. Rev. E* **2012**, *85*, 056107.
14. Song, C.; Qu, Z.; Blumm, N.; Barabási, A. Limits of predictability in human mobility. *Science* **2010**, *327*, 1018–1021.

15. Meiss, M.R.; Menczer, F.; Fortunato, S.; Flammini, A.; Vespignani, A. Ranking web sites with real user traffic. Proceedings of the international conference on Web search and web data mining. ACM, 2008, pp. 65–76.
16. Chierichetti, F.; Kumar, R.; Raghavan, P.; Sarlós, T. Are web users really Markovian? Proceedings of the 21st international conference on World Wide Web. ACM, 2012, pp. 609–618.
17. Singer, P.; Helic, D.; Taraghi, B.; Strohmaier, M. Memory and Structure in Human Navigation Patterns. *arXiv:1402.0790* **2014**.
18. Takaguchi, T.; Nakamura, M.; Sato, N.; Yano, K.; Masuda, N. Predictability of conversation partners. *Phys. Rev. X* **2011**, *1*, 011008.
19. Holme, P.; Saramäki, J. Temporal networks. *Phys. Rep.* **2012**, *519*, 97–125.
20. Rosvall, M.; Bergstrom, C. Maps of random walks on complex networks reveal community structure. *Proc. Natl. Acad. Sci. USA* **2008**, *105*, 1118.
21. Shannon, C. A Mathematical Theory of Communication, 1948.
22. Rosvall, M.; Bergstrom, C. Multilevel compression of random walks on networks reveals hierarchical organization in large integrated systems. *PLoS ONE* **2011**, *6*, e18209.
23. Kawamoto, T.; Rosvall, M. Estimating the resolution limit of the map equation in community detection. *Physical Review E* **2015**, *91*, 012809.
24. Bae, S.H.; Howe, B. GossipMap: A distributed community detection algorithm for billion-edge directed graphs. Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. ACM, 2015, p. 27.
25. Blondel, V.D.; Guillaume, J.L.; Lambiotte, R.; Lefebvre, E. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment* **2008**, *2008*, P10008.