LEARNING LOW-RANK LATENT MESOSCALE STRUCTURES IN NETWORKS

HANBAEK LYU, YACOUB H. KUREH, JOSHUA VENDROW, AND MASON A. PORTER

ABSTRACT. It is common to use networks to encode the architecture of interactions between entities in complex systems in the physical, biological, social, and information sciences. Moreover, to study the large-scale behavior of complex systems, it is important to study mesoscale structures in networks as building blocks that influence such behavior [17, 43]. In this paper, we present a new approach for describing low-rank mesoscale structure in networks, and we illustrate our approach using several synthetic network models and empirical friendship, collaboration, and protein–protein interaction (PPI) networks. We find that these networks possess a relatively small number of ‘latent motifs’ that together can successfully approximate most subnetworks at a fixed mesoscale. We use an algorithm that we call “network dictionary learning” (NDL) [30], which combines a network sampling method [29] and nonnegative matrix factorization [19, 30], to learn the latent motifs of a given network. The ability to encode a network using a set of latent motifs has a wide range of applications to network-analysis tasks, such as comparison, denoising, and edge inference. Additionally, using our new network denoising and reconstruction (NDR) algorithm, we demonstrate how to denoise a corrupted network by using only the latent motifs that one learns directly from the corrupted networks.

It is often insightful to examine structures in networks [40] at an intermediate scale (i.e., at a “mesoscale”) that lies between the microscale of nodes and edges but below macroscale distributions of local network properties. There are a large variety of network mesoscale structures of networks, including community structure [9, 47], core–periphery structure [49], and role structures [1]. We are interested in mesoscale network structures that are large enough that it is reasonable to discuss their collective properties, but that are also small enough so that we can also discuss their statistical properties. In this paper, we examine mesoscale network structures that we obtain using $k$-node induced subgraphs of networks. These networks have $k$ nodes that inherit their adjacency structures from the original networks from which we draw them. Because most real-world networks are sparse [40], independently choosing a set of $k$ nodes from a network may not return meaningful information. Instead, we use motif sampling [29]: we first uniformly randomly sample a set of $k$ nodes that form a path (this set is called a ‘$k$-chain motif’), and we then obtain the subgraph that is induced by that $k$-chain motif by including all of the edges between those $k$ nodes. This guarantees that we sample a connected subgraph of a network while assuming very little about the structure of the original network; by repeating this process, we obtain a data set of ‘mesoscale patches’ of a network. We then use ‘dictionary-learning’ algorithms [32] to learn mesoscale structures of networks that we call ‘latent motifs’. We then use latent motifs to infer subgraph structures of networks, compare different networks, and denoise corrupted networks.

Dictionary-learning algorithms are machine-learning techniques that learn interpretable latent structures of complex data sets and are applied regularly in the data analysis of text and images [7, 34, 46]. Such algorithms usually consist of two steps. First, one samples a large number of
structured subsets of a data set (e.g., square patches of an image or collections of a few sentences of a text); we refer to such a subset as a mesoscale patch of a data set. Second, one finds a set of basis elements such that taking a nonnegative linear combination of them can successfully approximate each of the sampled mesoscale patches. Such a set of basis elements is called a dictionary, and one can interpret each basis element as a latent structure of the data set. As an example, consider the image of the artwork Cycle by M. C. Escher in Figure 1a. We first sample 10,000 square patches of $21 \times 21$ pixels, and we then use a nonnegative matrix factorization (NMF) [19] algorithm to find a dictionary with $r = 25$ square patches (see Figure 1a). Each element of the learned dictionary describes a latent shape in the image.

Algorithms for network dictionary learning (NDL) [30] use a similar idea. As mesoscale patches of a network, we use the $k$-node subgraphs that are induced by motif sampling. We represent these subgraphs using their $k \times k$ adjacency matrices. After obtaining sufficiently many mesoscale patches of a network, we apply NMF to learn a dictionary for the latent adjacency matrices, which we call latent motifs of the network. We give a complete implementation of our approach in Algorithm 1 of the Supplementary Information (SI). See our SI for more details, including the theoretical guarantees for Algorithm 1 in Theorems G.2 and G.5.

**Figure 1.** Illustration of mesoscale structures that we learn from (a) images and (b,c) networks. In all experiments in this figure, we form a matrix $X$ of size $d \times n$ by sampling $n$ mesoscale patches of size $d = 21 \times 21$ from the corresponding object. For the image in panel (a), the columns of $X$ are square patches of $21 \times 21$ pixels. In panels (b) and (c), we show both heat maps and adjacency matrices. We take the columns of $X$ to be the $k \times k$ adjacency matrices of the connected subgraphs that are induced by a walk of $k = 21$ nodes, where a walk of $k$ nodes consists of $k$ nodes $x_1, \ldots, x_k$ such that $x_i$ and $x_{i+1}$ are adjacent for all $i \in \{1, \ldots, k\}$. Using nonnegative matrix factorization (NMF), we compute an approximate factorization $X \approx WH$ into nonnegative matrices, where $W$ has $r = 25$ columns. Because of this factorization, we can approximate any sampled mesoscale patches (i.e., the columns of $X$) of an object by a nonnegative linear combination of the columns of $W$, which we can interpret as latent shapes for images and latent motifs (i.e., subgraphs) for networks, respectively. The network dictionaries of latent motifs that we learn from the (b) UCLA and (c) Caltech Facebook networks reveal distinctive social structures. For example, if we uniformly sample a chain of 21 friends in one of these networks, we observe for Caltech that there are likely to be communities with six or more nodes and also some ‘hub’ users who know most of the others in the sample. However, for UCLA, it is unlikely that there are such communities or hubs. In the heat map of the UCLA network, we show only the first 3000 nodes according to the node labeling in the data set.
In Figure 1, we demonstrate the NDL method for networks of Facebook friendships (which were collected on one day in fall 2005) from UCLA (“UCLA”) and Caltech (“Caltech”) [55, 56]. Each node in one of these networks is a Facebook account of an individual, and each edge encodes a Facebook friendship between two individuals. Using NDL, we learn 25 latent motifs from each of these two social networks using a chain motif with \( k = 21 \) nodes. In each sample, the two diagonal lines in the latent motifs in Figure 1 correspond to the edges along with the \( k \)-chain motif. We refer to the remaining entries as ‘off-chain’ entries; one learns these from subgraphs that are induced by the chain motif. The latent motifs in UCLA’s dictionary (see Figure 1b) have sparse off-chain connections with a few clusters, whereas Caltech’s dictionary (see Figure 1c) has relatively dense off-chain connections. Such latent motifs reveal distinct social structures in the two networks. For example, if we uniformly sample a chain of 21 friends in one of these networks, we observe for Caltech that there are likely to be communities with six or more nodes and also some ‘hub’ users who know most of the others in the sample. However, for UCLA, it is unlikely that there are such communities or hubs. See Figure 3.

![Latent motifs](image)

**Figure 2.** Latent motifs that we learn from 14 networks (eight real-world networks and six synthetic networks, which include two distinct instantiations from each of three random-graph models) at five different scales (specifically, for \( k = 6, 11, 21, 51, 101 \)), which reveal distinct mesoscale structures in the networks. Using network dictionary learning (NDL), we learn network dictionaries of \( r = 25 \) latent motifs of \( k \) nodes for each of the 14 networks. For each network at each scale, we show only the second-most dominant latent motifs from each dictionary. These motifs include more information than the most dominant motifs for these sparse networks. Black squares represent 1 entries and white squares represent 0 entries. See Section D.2 in our SI for details of how we measure latent motif dominance, and see Figure 6 in our SI for the most dominant latent motifs of each network.

We examine mesoscale structures of eight real-world networks and six synthetic networks using NDL. The real-world networks are Facebook networks from Caltech, UCLA, Harvard, and MIT [55, 56], SNAP Facebook (which we also denote as SNAP FB as a shorthand) [11, 24], arXiv ASTRO-PH (with a shorthand of arXiv) [11, 23], Coronavirus PPI (with a shorthand of Coronavirus) [10, 44, 52], and Homo sapiens PPI (with a shorthand of H. sapiens) [11, 44]. The first four networks are 2005 Facebook networks from four universities from the FACEBOOK100 data set [56]. The fifth network is a 2012 Facebook network that was collected from survey participants [24]. The sixth network is a collaboration network based on coauthorship of preprints that were posted in the astrophysics category of the arXiv preprint server. The seventh network is a protein–protein interaction (PPI) network of proteins that are related to the coronaviruses that cause Coronavirus disease 2019.
(COVID-19), Severe Acute Respiratory Syndrome (SARS), and Middle Eastern Respiratory Syndrome (MERS) [52]. The eighth network is a PPI network of proteins that are related to *Homo sapiens* [44].

For the six synthetic networks, we generate two instances each of Erdős–Rényi (ER) $G(N,p)$ networks [8], Watts–Strogatz (WS) networks [57], and Barabási–Albert (BA) networks [2]. These are three types of well-studied random-graph models [40]. Each of the ER networks has 5000 nodes, and we independently connect each pair of nodes with probabilities of $p = 0.01$ (in the network that we call $ER_1$) and $p = 0.02$ (in $ER_2$). For the WS networks, we use rewiring probabilities of $p = 0.05$ (in $WS_1$) and $p = 0.1$ (in $WS_2$) starting from a 5000-node ring network in which each node is adjacent to its 50 nearest neighbors. For the BA networks, we use $m = 25$ (in $BA_1$) and $m = 50$ (in $BA_2$), where $m$ denotes the number of edges of each new node when it connects (via linear preferential attachment) to the existing network, which we grow from an initial network of $m$ individual nodes (i.e., none of them are adjacent to each other) until it has 5000 nodes. See Section F.1 in our SI for more details.

One can interpret the size (i.e., number of nodes) $k$ of a chain motif as a scale parameter. A $k \times k$ mesoscale patch of a network that one obtains by using the $k$-chain motif encodes connectivity between nodes that are at most $k - 1$ edges apart in the network. In Figure 2, we show the second-most dominant latent motif (see Section D.2 and Figure 6 in our SI) that we learn from each of 14 networks — eight real-world networks and six synthetic networks — at various scales (specifically, for $k = 6, 11, 21, 51, 101$) when we use a dictionary with $r = 25$ latent motifs. The latent motifs differ drastically both across the different networks and across different scales.

Suppose that we are given a network $G$ and a dictionary $W$ of latent motifs at scale $k$, where we may or may not learn $W$ from $G$. Consider the following two scenarios. In one scenario, we suppose that we know $G$ exactly, and we ask how to measure the ‘effectiveness’ of $W$ in approximating mesoscale patches of $G$ at scale $k$. In the other scenario, we suppose that $G$ is a noisy version of some true network $G_{\text{true}}$ and that $W$ is ‘faithful’ in the sense that it can well-approximate mesoscale patches of $G_{\text{true}}$ at scale $k$. (See (6) in the SI for the precise definition.) We then ask how we can infer the true network $G_{\text{true}}$ from $G$ and $W$.

To examine the above questions, we develop an algorithm that we call *network denoising and reconstruction* (NDR) (see Algorithm 2) that takes a network $G$ and network dictionary $W$ as input and outputs a weighted network $G_{\text{recons}}$ that has the same node set as $G$. The NDR algorithm repeatedly (until convergence) samples mesoscale patches of $G$ at scale $k$, finds a nonnegative linear approximation of them using the latent motifs in $W$. Because each edge $e$ of $G$ can appear in multiple mesoscale patches of $G$, there may be multiple reconstructed weights for $e$ in this procedure. We take their mean for the final weight of $e$ in $G_{\text{recons}}$ (see Algorithm 2). To measure the effectiveness of $W$ for an unweighted network $G$ (i.e., edges are either present or absent and there are no multi-edges), one can threshold the weighted edges of $G_{\text{recons}}$ at some fixed level $\theta \in [0, 1]$ to obtain an undirected reconstructed network $G_{\text{recons}}(\theta)$ with binary edge weights (of either 0 or 1), which one can then compare directly with the original unweighted network $G$. We regard $W$ as effective at describing $G$ at mesoscale $k$ if $G_{\text{recons}}(\theta)$ is close to $G$ for some $\theta$. (We will quantify our notion of ‘closeness’ in the next paragraph.) We interpret the edge weights in $G_{\text{recons}}$ as measures of confidence of the corresponding edges in $G$ with respect to $W$. For example, if an edge $e$ has the smallest weight in $G_{\text{recons}}$, we regard it as the most ‘outlying’ with respect to the latent motifs in $W$. See Theorems G.7 and G.10 in the SI for theoretical guarantees and error bounds for NDR.

In Figure 3, we show various reconstruction experiments using several real-world networks and synthetic networks. We perform these experiments for various values of the edge threshold $\theta$ and $r = 9, 16, 25, 36, 49, 81, 100$ latent motifs in a single dictionary. Each network dictionary in Figure 3 uses a chain motif with $k = 21$ nodes, for which the dimension of the space of all possible mesoscale patches (i.e., the adjacency matrices of the induced subgraphs) is $\binom{21}{2} = 200$. An important
observation is that one can reconstruct a given network using an arbitrary network dictionary, which one can even learn from a different network. Such a ‘cross-reconstruction’ allows us to quantitatively compare the learned mesoscale structures of different networks. We label each subplot of Figure 3 with \( Y \leftarrow X \) to indicate that we are reconstructing network \( Y \) using a network dictionary that we learn from network \( X \). We turn the weighted reconstructed networks into unweighted networks by thresholding their edges using some threshold \( \theta \). We measure reconstruction accuracy by calculating the Jaccard index of the original network’s edge set and the reconstructed network’s edge set. That is, to measure the similarity of two edge sets, we calculate the number of edges in the intersection of these sets divided by the number of edges in the union of these sets. We measure the reconstruction accuracy by calculating the Jaccard index between the original network’s edge set and the reconstructed network’s edge set. In panel (a), we plot accuracies versus \( \theta \) (keeping the number of latent motifs fixed at \( r = 25 \)), where \( X \) is one of five real-world networks (two PPI networks, two Facebook networks, and one collaboration network). In panels (b) and (c), we reconstruct each of the four Facebook networks using network dictionaries with \( r \in \{9, 16, 25, 36, 64, 81, 100\} \) latent motifs that we learn from one of ten networks (with the edge threshold value fixed at \( \theta = 0.5 \)).

In Figure 3a, we plot the accuracy for ‘self-reconstruction’ \( X \leftarrow X \) versus \( \theta \) (with \( r = 25 \)), where \( X \) is one of the real-world networks \texttt{CORONAVIRUS}, \texttt{H. SAPIENS}, \texttt{SNAP FB}, \texttt{CALTECH}, and \texttt{ARXIV}. The accuracies for \texttt{H. SAPIENS} and \texttt{CALTECH} peak above 95% when \( \theta \approx 0.4 \), the accuracies for \texttt{ARXIV} and \texttt{SNAP FB} peak above 82% for \( \theta \approx 0.6 \), and the accuracy for \texttt{CORONAVIRUS} peaks above 95% near \( \theta = 0.5 \). We choose \( \theta = 0.5 \) for the cross-reconstruction experiments for the Facebook networks.
**Caltech, Harvard, MIT, and UCLA** in Figures 3b,c. We observe that these four Facebook networks have self-reconstruction accuracies above 75% using \( r = 25 \) motifs with \( \theta = 0.5 \). The total number of dimensions when using mesoscale patches at scale \( k = 21 \) is 200, so this result suggests that all of these nine real-world networks have low-rank mesoscale structures at scale \( k = 21 \).

We consider accuracies for cross-reconstruction \( Y \leftarrow X \) in Figures 3b,c, where \( Y \) is one of the Facebook networks **Caltech, Harvard, MIT, and UCLA** and \( X \) (with \( X \neq Y \)) is one of these four network or one of the six synthetic networks ER\(_i\), WS\(_i\), or BA\(_i\) (with \( i \in \{1,2\} \)). From the cross-reconstruction accuracies and interpreting the latent motifs (see Section B.4 in the SI) in Figures 1 and 2 (see also Figures 8, 10, and 14 in the SI), we draw the following conclusions at scale \( k = 21 \). First, the mesoscale structure of **Caltech** is distinct from those of **Harvard, UCLA, and MIT**. This is consistent with prior studies of these networks (see, e.g., [15, 56]). Second, **Caltech**’s mesoscale structure at scale \( k = 21 \) has a higher dimension than those of the other three universities’ Facebook networks. Third, **Caltech** has a lot more communities of size at least 10 than the other three universities’ Facebook networks. Fourth, both BA networks capture the mesoscale structures of **MIT, Harvard, and UCLA** at scale \( k = 21 \) better than the synthetic networks that we generate from ER and WS models. For instance, the self-reconstruction accuracies in Figures 3b,c using \( r = 9 \) latent motifs are about 64% for **Caltech** and 80% or above for the other three universities’ Facebook networks. See Section F.5 in the SI for further discussion.

In Figure 4, we use our algorithms to perform two types of network denoising. We can think of them as distinct binary classification problems in network analysis: network denoising with **subtractive noise** (which is often called edge ‘prediction’ [12, 18, 26, 28, 37] and network denoising with **additive noise** [4]. In each scenario, we suppose that we are given an observed network \( G = (V,E) \) with node set \( V \) and edge set \( E \) and are asked to find an unknown network \( G' = (V,E') \) with the same node set \( V \) but a possibly different edge set \( E' \). We interpret \( G \) as a corrupted version of a ‘true network’ \( G'' \) that we observe with some uncertainty. One can interpret edges and non-edges in \( G \) as ‘false relations’ and ‘false non-relations’, respectively. In the subtractive-noise setting, we assume that \( G \) is a partially observed version of \( G'' \) (i.e., \( E \subseteq E' \)), and we seek to classify all non-edges in \( G \) into ‘positives’ (i.e., edges in \( G'' \)) and ‘negatives’ (i.e., non-edges in \( G'' \)). In the additive-noise setting, we suppose that \( G \) is a corrupted version of \( G'' \) to which some unknown edges have been added (i.e., \( E \supseteq E' \)), and we seek to classify all edges in \( G \) into ‘positives’ (i.e., edges in \( G'' \)) and ‘negatives’ (i.e., non-edges in \( G'' \)).

To experiment with these problems, we use the following four real-world networks: **Caltech, SNAP FB, arXiv Coronavirus, and H. sapiens**. Given a network \( G = (V,E) \), our experiments proceed as follows. In the subtractive-noise setting, we create two smaller networks by removing uniformly random subsets that consist of 10% (in one experiment) or 50% (in the other) of the edges from our network. In the additive-noise case, we create two corrupted networks by adding edges between node pairs that we choose independently with a fixed probability so that 10% or 50% of the edges in a corrupted network are new. We then apply NDL with \( r = 25 \) latent motifs at scale \( k = 21 \) to learn a network dictionary for each of these four networks, and we use each dictionary to reconstruct the network from which it was learned using NDR. The reconstruction algorithms output a weighted network \( G_{\text{recons}} \), where the weight of each edge is our confidence that the edge belongs to that network. For denoising subtractive (respectively, additive) noise, we classify each non-edge (respectively, edge) in a corrupted network as ‘positive’ if its weight in \( G_{\text{recons}} \) is strictly larger than some threshold \( \theta \) and ‘negative’ otherwise. By varying \( \theta \), we construct a receiver-operating characteristic (ROC) curve that consists of points whose horizontal and vertical coordinates are the false-positive rates and true-positive rates, respectively.

In Figure 4, we show the ROC curves and corresponding area-under-the-curve (AUC) scores for our network-denoising experiments with subtractive and additive noise for the four networks. For example, when we add 50% false edges (with one extra edge as a tie-breaker, given the odd
specifically, we compare to node2vec [45], and arXiv versus those of other methods. In our experiments with subtractive noise, we corrupt a network by uniformly randomly removing 10% or 50% of its edges, and we seek to classify the removed edges and the non-edges as true edges and false edges, respectively. In our experiments with additive noise, we corrupt a network by uniformly randomly adding 10% or 50% of the number of its edges, and we seek to classify the edges and non-edges in the resulting corrupted network as ‘negative’ (i.e., false edges) or ‘positive’ (i.e., true edges). To perform classification in a network, we first use NDL to learn latent motifs from a corrupted network and then reconstruct the networks using NDR to assign a confidence value to each potential edge. We then use these confidence values to infer membership of potential edges in the uncorrupted network. Importantly, we never use information from the original networks. For the Caltech Facebook network in panel (b), we also perform edge inference and denoising using a network dictionary that we learn from the MIT Facebook network. For each network, we indicate the receiver-operating characteristic (ROC) curves and corresponding area-under-the-curve (AUC) scores for network denoising with additive noise using the labels +10% and +50%, and we indicate the ROC curves and corresponding AUC scores for network denoising with subtractive noise using the labels −10% and −50%.

number of edges in the original network) to Coronavirus, such that 2,463 edges are true and 1,232 edges are false, we are able to detect over 90% of the false edges while misclassifying only 10% of the true edges. In Figure 4f, we compare the performance of our method to those of some popular supervised algorithms that are based on network embeddings. Specifically, we compare to node2vec [11], DeepWalk [45], and LINE [50] for the task of denoising 50% subtractive noise for SNAP FB, H. Sapiens, and arXiv. Our method achieves state-of-the-art results in all cases.

We make two important remarks about applying the NDL and NDR algorithms to network denoising. First, NDL and NDR are able to perform the desired classification tasks in an unsupervised manner, in the sense that we do not require fully known examples to train our algorithm. This is particularly useful when it is difficult to obtain a large number of fully known examples, such as measuring PPI networks for a new organism (e.g., SARS-CoV-2). Part of the reason that NDL and NDR are successful at these network-denoising tasks is because of the low-rank nature of the examined mesoscale structures of the social and PPI networks (see Figure 3a). Specifically, because NDL learns a small number of latent motifs that are able to successfully give an approximate basis for all mesoscale patches, they should not be affected significantly by noise. Second, unlike the existing algorithms that we just mentioned, we are able to perform denoising on a network using...
information that we learn from an entirely different network. Consequently, we are able to successfully perform not only self-reconstruction but also cross-reconstruction. For instance, in Figure 4, we show the results for denoising CALTECH using a dictionary that we learn from the MIT network, which we expect to have a similar structure to CALTECH based on the results of the experiments that we highlighted in Figure 3 and also on prior research on these networks [14, 43, 56].

Our experiments in Figures 3 and 4 illustrate that various social, collaboration and PPI networks have low-rank [36] mesoscale structures, in the sense that a few (e.g., $r = 25$, but see Figure 3 for other choices of $r$) latent motifs that we learn using NDL are able to reconstruct, infer, and denoise the edges in the entire networks by employing the NDR algorithm. We hypothesize that such low-rank mesoscale structures are a general phenomenon for networks of interactions in various complex systems beyond the social, collaboration, and PPI networks that we have examined. As we have illustrated in this paper, one can leverage mesoscale structures to perform important tasks like network denoising, so it is important in future studies to explore the level of generality of our insights.

References

[1] Nesreen Ahmed, Ryan Anthony Rossi, John Lee, Theodore Willke, Rong Zhou, Xiangnan Kong, and Hoda Eldardiry. Role-based graph embeddings. IEEE Transactions on Knowledge and Data Engineering, 2020. Available at doi:10.1109/TKDE.2020.3006475.
[2] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. Science, 286(5439):509–512, 1999.
[3] Marco Bressan. Faster algorithms for sampling connected induced subgraphs. arXiv preprint arXiv:2007.12102, 2020.
[4] Fernanda B. Correia, Edgar D. Coelho, José L. Oliveira, and Joel P. Arrais. Handling noise in protein interaction networks. BioMed Research International, 2019:1–13, 2019.
[5] Rick Durrett. Probability: Theory and Examples. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, Cambridge, UK, fourth edition, 2010.
[6] Bradley Efron, Trevor Hastie, Iain Johnstone, and Robert Tibshirani. Least angle regression. The Annals of Statistics, 32(2):407–499, 2004.
[7] Michael Elad and Michal Aharon. Image denoising via sparse and redundant representations over learned dictionaries. IEEE Transactions on Image Processing, 15(12):3736–3745, 2006.
[8] Paul Erdős and Alfréd Rényi. On random graphs. I. Publicationes Mathematicae, 6(290–297):18, 1959.
[9] Santo Fortunato and Darko Hric. Community detection in networks: A user guide. Physics Reports, 659:1–44, 2016.
[10] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. pages 855–864, 2016.
[11] Roger Guimerà. One model to rule them all in network science? Proceedings of the National Academy of Sciences of the United States of America, 117(41):25195–25197, 2020.
[12] Roger A Horn and Charles R Johnson. Matrix Analysis. Cambridge University Press, Cambridge, UK, second edition, 2012.
[13] Abigail Z. Jacobs, Samuel F. Way, Johan Ugander, and Aaron Clauset. Assembling the facebook: Using heterogeneity to understand online social network assembly. In Proceedings of the ACM Web Science Conference, WebSci ’15, New York City, NY, USA, 2015. Association for Computing Machinery.
[14] Lucas G. S. Jeub, Prakash Balachandran, Mason A. Porter, Peter J. Mucha, and Michael W. Mahoney. Think locally, act locally: Detection of small, medium-sized, and large communities in large networks. Physical Review E, 91:012821, 2015.
[15] Nadav Kashtan, Shalev Itzkovitz, Ron Milo, and Uri Alon. Efficient sampling algorithm for estimating subgraph concentrations and detecting network motifs. Bioinformatics, 20(11):1746–1758, 2004.
Ankit N. Khambhati, Ann E. Sizemore, Richard F. Betzel, and Danielle S. Bassett. Modeling and interpreting mesoscale network dynamics. *NeuroImage*, 180:337–349, 2018.

Katja Kovács, István A. and Luck, Kerstin Spirohn, Yang Wang, Carl Pollis, Sadie Schlabach, Wenting Bian, Dae-Kyum Kim, Nishka Kishore, and Tong Hao. Network-based prediction of protein interactions. *Nature Communications*, 10(1):1240, 2019.

Daniel D. Lee and H. Sebastian Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401(6755):788, 1999.

Daniel D. Lee and H. Sebastian Seung. Algorithms for non-negative matrix factorization. In *Advances in Neural Information Processing Systems*, pages 556–562, 2001.

Honglak Lee, Alexis Battle, Rajat Raina, and Andrew Y Ng. Efficient sparse coding algorithms. In *Advances in Neural Information Processing Systems*, pages 801–808, 2007.

Jure Leskovec and Christos Faloutsos. Sampling from large graphs. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 631–636, 2006.

Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford Large Network Dataset Collection. Available at [http://snap.stanford.edu/data](http://snap.stanford.edu/data), 2020.

Jure Leskovec and Julian J. McAuley. Learning to discover social circles in ego networks. In *Advances in Neural Information Processing Systems*, pages 539–547, 2012.

David A Levin and Yuval Peres. *Markov Chains and Mixing Times*. American Mathematical Society, Providence, RI, USA, 2017.

David Liben-Nowell and Jon Kleinberg. The link-prediction problem for social networks. *Journal of the American Society for Information Science and Technology*, 58(7):1019–1031, 2007.

László Lovász. *Large Networks and Graph Limits*, volume 60 of *Colloquium Publications*. American Mathematical Society, Providence, RI, USA, 2012.

Linyuan Lü and Tao Zhou. Link prediction in complex networks: A survey. *Physica A*, 390(6):1150–1170, 2011.

Hanbaek Lyu, Facundo Memoli, and David Sivakoff. Sampling random graph homomorphisms and applications to network data analysis. *arXiv:1910.09483*, 2019.

Hanbaek Lyu, Deanna Needell, and Laura Balzano. Online matrix factorization for Markovian data and applications to network dictionary learning. *Journal of Machine Learning Research*, 21:1–49, 2020.

Julien Mairal. Stochastic majorization-minimization algorithms for large-scale optimization. In *Advances in Neural Information Processing Systems*, pages 2283–2291, 2013.

Julien Mairal, Francis Bach, Jean Ponce, and Guillermo Sapiro. Online learning for matrix factorization and sparse coding. *Journal of Machine Learning Research*, 11:19–60, 2010.

Julien Mairal, Francis Bach, Jean Ponce, Guillermo Sapiro, and Andrew Zisserman. Non-local sparse models for image restoration. *IEEE 12th International Conference on Computer Vision*, pages 2272–2279, 2009.

Julien Mairal, Michael Elad, and Guillermo Sapiro. Sparse representation for color image restoration. *IEEE Transactions on Image Processing*, 17(1):53–69, 2007.

Julien Mairal, Michael Elad, and Guillermo Sapiro. Sparse learned representations for image restoration. *Proceedings of the 4th World Conference of the International Association for Statistical Computing*, page 118, 2008.

Ivan Markovsky and Konstantin Usevich. *Low Rank Approximation*. Springer-Verlag, Heidelberg, Germany, 2012.

Aditya Krishna Menon and Charles Elkan. Link prediction via matrix factorization. In Dimitrios Gunopulos, Thomas Hofmann, Donato Malerba, and Michalis Vazirgiannis, editors, *Machine Learning and Knowledge Discovery in Databases*, pages 437–452, Heidelberg, Germany, 2011. Springer-Verlag.

Sean P Meyn and Richard L Tweedie. *Markov Chains and Stochastic Stability*. Springer-Verlag, Heidelberg, Germany, 2012.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, pages 3111–3119, 2013.

Mark E. J. Newman. *Networks*. Oxford University Press, Oxford, UK, second edition, 2018.
[41] Andrew Y. Ng, Michael I. Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. *Advances in neural information processing systems*, pages 849–856, 2002.

[42] William S. Noble. What is a support vector machine? *Nature biotechnology*, 24(12):1565–1567, 2006.

[43] Jukka-Pekka Onnela, Daniel J. Fenn, Stephen Reid, Mason A. Porter, Peter J. Mucha, Mark D. Fricker, and Nick S. Jones. Taxonomies of networks from community structure. *Physical Review E*, 86(3):036104, 2012.

[44] Rose Oughtred, Chris Stark, Bobby-Joe Breitkreutz, Jennifer Rust, Lorrie Boucher, Christie Chang, Nadine Kolas, Lara O’Donnell, Genie Leung, and Rochelle McAdam. The BioGRID interaction database: 2019 update. *Nucleic Acids Research*, 47(D1):D529–D541, 2019.

[45] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. DeepWalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 701–710, 2014.

[46] Gabriel Peyré. Sparse modeling of textures. *Journal of Mathematical Imaging and Vision*, 34(1):17–31, 2009.

[47] Mason A. Porter, Jukka-Pekka Onnela, and Peter J. Mucha. Communities in networks. *Notices of the American Mathematical Society*, 56(9):1082–1097, 1164–1166, 2009.

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

[49] Puck Rombach, Mason A. Porter, James H. Fowler, and Peter J. Mucha. Core-periphery structure in networks (revisited). *SIAM Review*, 59(3):619–646, 2017.

[50] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. LINE: Large-scale information network embedding. In *Proceedings of the 24th International Conference on World Wide Web*, pages 1067–1077, 2015.

[51] Lei Tang and Huan Liu. Leveraging social media networks for classification. *Data Mining and Knowledge Discovery*, 23(3):447–478, 2011.

[52] theBiogrid.org. Coronavirus PPI network. 2020. Retrieved from https://wiki.thebiogrid.org/doku.php/covid (downloaded 24 July 2020, Ver. 3.5.187.tab3).

[53] theBiogrid.org. Homo sapiens PPI network. 2020. Retrieved from https://wiki.thebiogrid.org/doku.php/covid (downloaded 24 July 2020, Ver. 3.5.180.tab2).

[54] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996.

[55] Amanda L. Traud, Eric D. Kelsic, Peter J. Mucha, and Mason A. Porter. Comparing community structure to characteristics in online collegiate social networks. *SIAM Review*, 53:526–543, 2011.

[56] Amanda L. Traud, Peter J. Mucha, and Mason A. Porter. Social structure of Facebook networks. *Physica A*, 391(16):4165–4180, 2012.

[57] Duncan J. Watts and Steven H. Strogatz. Collective dynamics of ‘small-world’ networks. *Nature*, 393(6684):440–442, 1998.

[58] Sebastian Wernicke. Efficient detection of network motifs. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 3(4):347–359, 2006.
Supplementary Information:

LEARNING LOW-RANK LATENT MESOSCALE STRUCTURES IN NETWORKS

APPENDIX A. OVERVIEW

In this supplementary material, we present our algorithms for network dictionary learning (NDL) and network denoising and reconstruction (NDR), and we prove theoretical results about their convergence and error bounds. We give the full NDL algorithm (see Algorithm 1) in Section D, and we give the full NDR algorithm (see Algorithm 2) in Section E. We introduce the notion of ‘latent motif dominance’ in Section D.2 to measure the significance of each latent motif that we learn from a network. In Section G, we give a rigorous analysis of the NDL and NDR algorithms.

APPENDIX B. PROBLEM FORMULATION FOR NETWORK DICTIONARY LEARNING (NDL)

B.1. Definitions and notation. To facilitate our discussions, we use terminology and notation from [27, Ch. 3]. In the main text, we described a network as graph $G = (V, E)$ with node set $V$ and edge set $E$ without directed or multi-edges, but possibly with self-edges. One can characterize the edge set $E$ of a graph $G$ as an adjacency matrix $A_G : V^2 \rightarrow \{0, 1\}$, where $A(x, y) = 1$ if $(x, y) \in E$. The set $E$ determined a unique network $G = (V, A_G)$, where $A_G$ is the adjacency matrix of $G$. The set $V(G)$ is the node set of the network $G$, which has size $|V(G)|$, where $|S|$ is the number of elements in the set $S$. A pair $(x, y)$ of nodes in $G$ is called a directed edge if $A(x, y) > 0$. We say that a network $G = (V, A)$ is symmetric if its weight matrix is symmetric (i.e., $A(x, y) = A(y, x)$ for all $x, y \in V$) and we say that it is binary if $A(x, y) \in \{0, 1\}$ for all $x, y \in V$. A network $G = (V, A)$ is connected if for any two distinct nodes $x, y \in V$, there exists a sequence of nodes $x_0, x_1, \ldots, x_m$ for some $m \geq 1$ such that $A(x_i, x_{i+1}) > 0$ for all $i \in \{0, \ldots, m - 1\}$ and $\{x_0, x_m\} = \{x, y\}$.

Homomorphisms between networks and motif sampling. A network $G = (V, A)$ is bipartite if it admits a "bipartition", which is a partition $V = V_1 \cup V_2$ of the node set $V$ such that $V = V_1 \cup V_2$ and $A(x, y) = 0$ if $x \in V_1$ or $y \in V_2$ for each $x, y \in V$. Two networks $G = (V, A)$ and $G' = (V', A')$ satisfy $V' \subseteq V$ and $A'(x, y) \leq A(x, y)$ for all $x, y \in V'$, then we say that $G'$ is a subnetwork of $G$ and write $G' \subseteq G$.

B.2. Homomorphisms between networks and motif sampling. Being able to sample from a complex data set according to a known probability distribution (e.g., a uniform one) is a crucial ingredient in dictionary-learning problems. This is often the case for image-processing applications [7, 33, 35], as it is straightforward to sample a $k \times k$ patch uniformly at random from an image. However, the similar problem of uniformly randomly sampling a $k$-node connected subnetwork from a network is not straightforward [3, 16, 22, 58]. For our purpose of developing dictionary learning for networks, we use motif sampling, which
was introduced recently in [29]. In motif sampling, instead of directly sampling a connected subnetwork, one samples a random node map from a smaller network (i.e., a motif) into a target network that preserves adjacency relations, and one then uses the subnetwork that is induced on the nodes in the image of the node map. As we discuss below, such a node map between networks is a homomorphism.

Fix an integer \( k \geq 1 \) and a weight matrix \( A_F : [k]^2 \rightarrow [0, \infty) \), where we use the shorthand notation \([k] = \{1, \ldots, k\}\). We use the term motif for the corresponding network \( F = ([k], A_F) \). A motif is a network, and we use such motifs to sample from a given network. The type of motif in which we are particularly interested is a \( k \)-chain, for which \( A_F = 1 \{(1, 2), (2, 3), \ldots, (k - 1, k)\}\). A \( k \)-chain is a directed path with node set \([k]\). For a general \( k \)-node motif \( F = ([k], A_F) \) and a network \( G = (V, A) \), we define the probability distribution \( \pi_{F \rightarrow G} \) on the set \( V^{[k]} \) of all node maps \( \mathbf{x} : [k] \rightarrow V \) by

\[
\pi_{F \rightarrow G}(\mathbf{x}) = \frac{1}{Z} \prod_{i,j \in \{1, \ldots, k\}} A(\mathbf{x}(i), \mathbf{x}(j))^{A_F(i,j)},
\]

where \( Z \) is a normalizing constant that is called the homomorphism density of \( F \) in \( G \) [27]. A node map \( \mathbf{x} : [k] \rightarrow V \) a homomorphism \( F \rightarrow G \) if \( \pi_{F \rightarrow G}(\mathbf{x}) > 0 \), which is the case if and only if \( A(\mathbf{x}(a), \mathbf{x}(b)) > 0 \) for all \( a, b \in [k] \) with \( A_F(a, b) > 0 \) (with the convention that \( a^0 = 1 \) for all \( a \in \mathbb{R} \)). When both \( A \) and \( A_F \) are binary matrices, the probability distribution \( \pi_{F \rightarrow G} \) is the uniform distribution on the set of all homomorphisms \( F \rightarrow G \). This is the case for all examples that we discuss in the main manuscript. Motif sampling refers the problem of sampling a random homomorphism \( \mathbf{x} : F \rightarrow G \) according to the distribution in (1). In Section C, we discuss three Markov Chain Monte Carlo (MCMC) algorithms for motif sampling.

### B.3. Mesoscale patches of networks.

A homomorphism \( F \rightarrow G \) is a node map \( V(F) \rightarrow V(G) \) that maps the edges of \( F \) to some edges of \( G \), so it maps \( F \) onto a subgraph of \( G \). It thereby maps \( F \) ‘into’ \( G \). For each homomorphism \( \mathbf{x} : F \rightarrow G \) from a motif \( F = ([k], A_F) \) into a network \( G = (V, A) \), we define a \( k \times k \) matrix

\[
A_\mathbf{x}(a, b) := A(\mathbf{x}(a), \mathbf{x}(b)) \Phi_{F, \mathbf{x}}(a, b) \quad \text{for all} \quad a, b \in \{1, \ldots, k\},
\]

where \( \Phi_{F, \mathbf{x}} : [k]^2 \rightarrow \{0, 1\} \) is a \( k \times k \) binary matrix that we call a mask. Two particular choices of masks are

- **identity:** \( \Phi_{F, \mathbf{x}}(a, b) = 1 \) for all \( a, b \in \{1, \ldots, k\} \),
- **no-folding:** \( \Phi_{F, \mathbf{x}}(a, b) = 1 \left( A_F(a, b) > 0 \text{ or } \not\exists a', b' \in \{1, \ldots, k\} \text{ such that } [A_F(a', b') > 0 \text{ and } (\mathbf{x}(a), \mathbf{x}(b)) = (\mathbf{x}(a'), \mathbf{x}(b'))] \right) \) for all \( a, b \in \{1, \ldots, k\} \).

We call \( A_\mathbf{x} \) in (2) the mesoscale patch of \( G \) that is induced by the homomorphism \( \mathbf{x} : F \rightarrow G \), which is specified uniquely by the homomorphism \( \mathbf{x} : F \rightarrow G \), the weight matrix \( A \), and the mask \( \Phi \). For a \( k \times k \) matrix \( B \) and a homomorphism \( F \rightarrow G \), we say that the \( (a, b) \) entries of \( B \) are on-chain if \( A_F(a, b) > 0 \) and off-chain otherwise. The condition \( A_F(a, b) > 0 \) implies that \( A(\mathbf{x}(a), \mathbf{x}(b)) \) by the definition of the homomorphism \( \mathbf{x} \), so the on-chain entries of \( A_\mathbf{x} \) are always positive (and are always 1 if \( G \) is unweighted). However, the off-chain entries of \( A_\mathbf{x} \) are not necessarily positive, so they encode meaningful information about a network that is ‘detected’ by the homomorphism \( \mathbf{x} : F \rightarrow G \). For example, see the (1, 4) entries in the \( 4 \times 4 \) mesoscale patches in Figure 5b.

Suppose that the homomorphism \( \mathbf{x} \) uses \( k \) distinct nodes of \( G \) in its image \( \text{Im}(\mathbf{x}) := \{\mathbf{x}(a) \mid a \in \{1, \ldots, k\}\} \). It then follows that the network \( G_\mathbf{x} := (\text{Im}(\mathbf{x}), A_\mathbf{x}) \) that is induced by \( \mathbf{x} \) is a \( k \)-node subnetwork of \( G \). In this case, the no-folding mask (4) is the same as the identity mask (3). However, when \( \text{Im}(\mathbf{x}) \) has fewer than \( k \) nodes, \( G_\mathbf{x} \) is not necessarily a subnetwork of \( G \) and distinct positive off-chain entries of \( A_\mathbf{x} \) with the identity mask may represent the same edge in \( G \). We introduced the no-folding mask in (4) so that the positive off-chain entries of \( A_\mathbf{x} \) always have information that is not included in its on-chain entries.
As an illustration, consider the case in which $F$ is the 6-chain motif and $\mathcal{G} = (V, A)$ is an undirected and binary graph. Suppose that we have the following two learned factors

where the $x$ homomorphism where each entry distinct nodes, in this case, the mesoscale patch entries of $A_x$ equals $Q$, which has four off-chain entries of 1 in its upper triangle. However, all of the 1 entries in $A_x$ in this case correspond to the single edge between $p$ and $q$ in $\mathcal{G}$, so the off-chain entries of $A_x$ do not give any new information about the network $\mathcal{G}$ that is not already given by its on-chain entries. The indicator function in the definition of the no-folding mask $(4)$ prevents this situation. Indeed, in this case, the mesoscale patch $A_x$ with the no-folding mask is the matrix $P$ with all * entries equal to 0.

B.4. Problem formulation for network dictionary learning (NDL). The goal of the NDL problem is to learn, for a fixed integer $r \geq 1$, a set of $r$ nonnegative matrices $L_1, \ldots, L_r$ of size $k \times k$, Frobenius norms of at most 1, and

$$A_x \approx a_1(x)L_1 + \cdots + a_r(x)L_r$$

(5)

for each homomorphism $x : F \to \mathcal{G}$ for some coefficients $a_1(x), \ldots, a_r(x) \geq 0$. For each homomorphism $x : F \to \mathcal{G}$, this implies that one can approximate the mesoscale patch $A_x$ of $\mathcal{G}$ that is induced by $x$ as a suitable linear combination of the $r$ matrices $L_1, \ldots, L_r$. We call the the tuple $[L_1, \ldots, L_r]$ a network dictionary for $\mathcal{G}$, and we call each $L_i$ a latent motif of $\mathcal{G}$. We identify a network dictionary $[L_1, \ldots, L_r]$ with the nonnegative matrix $W \in \mathbb{R}_{\geq 0}^{k \times r}$ whose $j$th column is the vectorization of the $j$th latent motif $L_j$ for $j \in \{1, \ldots, r\}$. The choice of vectorization $\mathbb{R}^{k \times k} \to \mathbb{R}^{k^2}$ can be arbitrary, but we use a column-wise vectorization in Algorithm A4.

For the latent motifs to be interpretable, it is crucial that we require the entries of latent motifs $L_i$ and the coefficients $a_i(x)$ to be nonnegative. The nonnegativity constraint on each $L_i$ allows one to interpret each $L_i$ as the weight matrix of a $k$-node network. Additionally, because the coefficients $a_j(x)$ are also nonnegative, the approximate decomposition in (5) implies that $a_i(x)L_i \preceq A_x$. Therefore, if $a_i(x) > 0$, any network structure (e.g., nodes with large degree, communities, and so on) in the latent motif $L_i$ must exist in $A_x$. Therefore, one can consider the latent motifs as approximate $k$-node subnetworks $\mathcal{G}$ that exhibit typical network structure of $\mathcal{G}$ at scale $k$. In the spirit of Lee and Seung [19], one can regard the latent motifs as ‘parts’ of a network $\mathcal{G}$.

For a more precise formulation of (5), consider the following optimization problem:

$$\arg \min_{L_1, \ldots, L_r \in \mathbb{R}^{k \times k}} \mathbb{E}_{x \sim \pi_{F \to \mathcal{G}}} \left[ \inf_{a_1(x), \ldots, a_r(x) \geq 0} \left\| A_x - \sum_{i=1}^r a_i(x)L_i \right\|_F \right]$$

(6)

where $\pi_{F \to \mathcal{G}}$ is the probability distribution that we defined in (1) and $\| \cdot \|_F$ denotes the matrix Frobenius norm. The choice of the probability distribution $\pi_{F \to \mathcal{G}}$ for the homomorphisms $x : F \to \mathcal{G}$ is natural because it becomes the uniform distribution on the set of all homomorphisms $F \to \mathcal{G}$ when the adjacency matrices for $\mathcal{G}$ and $F$ are both binary. The NDL problem is computationally difficult because the objective function in (6) is non-convex and it is not obvious how to sample a homomorphism $F \to \mathcal{G}$ according to the distribution $\pi_{F \to \mathcal{G}}$ that we defined in (1). In Section D, we state an algorithm for NDL that approximately solves (6).

\footnote{Lee and Seung [19] discussed a similar nonnegative decomposition in which the $A_x$ are images of faces. The learned factors $L_i$ then capture parts of human faces (such as eyes, noses, and mouths).}
B.5. Overview of our algorithms and their theoretical guarantees. We overview our algorithms and their theoretical guarantees.

Algorithm 1: Given a network \( \mathcal{G} \), the NDL algorithm (see Algorithm 1) computes a sequence \((W_t)_{t \geq 0}\) of network dictionaries, which take the form of \( k^2 \times r \) matrices, of latent motifs.

Algorithm 2: Given a network \( \mathcal{G} \), a network dictionary \( W \), and a threshold parameter \( \theta > 0 \), the NDR algorithm (see Algorithm 2) computes a sequence of weighted networks \( \mathcal{G}_{\text{recons}} \) and binary (i.e., unweighted) networks \( \mathcal{G}_{\text{recons}}(\theta) \).

Theorem G.2: Given a non-bipartite network \( \mathcal{G} \) and a choice of the parameters in Algorithm 1, we prove that the sequence \((W_t)_{t \geq 0}\) of network dictionaries converges almost surely to the set of stationary points of the associated objective function in (6).

Theorem G.5: Given a bipartite network \( \mathcal{G} \) and a choice of the parameters in Algorithm 1, we prove a convergence result that is analogous to the one in Theorem G.2.

Theorem G.7: Given a non-bipartite target network \( \mathcal{G} \) and a network dictionary \( W \), we show that (i) the sequence of weighted reconstructed networks \( \mathcal{G}_{\text{recons}} \) that we compute using the NDR algorithm (see Algorithm 2) converges almost surely to some limiting network, and (ii) we obtain a closed-form expression of this limiting network. We also show that (iii) a suitable distance between an arbitrary network \( \mathcal{G}' \) and the limiting reconstructed network is bounded by the mean \( L_1 \) distance between the \( k \times k \) mesoscale patches of \( \mathcal{G}' \) and their nonnegative linear approximations from the latent motifs in \( W \). Finally, (iv) if \( \mathcal{G}' = \mathcal{G} \) in (iii), we show that upper bound of the distance between \( \mathcal{G} \) and \( \mathcal{G}_{\text{recons}} \) is approximately optimized if one learns the network dictionary \( W \) from the NDL algorithm 1.

Theorem G.10: We show a convergence result that is analogous to the one in Theorem G.10 for a bipartite target network \( \mathcal{G} \).

Appendix C. Markov Chain Monte Carlo (MCMC) Motif-Sampling Algorithms

We mentioned in Section B.4 that one of the main difficulties in solving the optimization problem (6) is to directly sample a homomorphism \( x : F \to \mathcal{G} \) from the distribution \( \pi_{F \to \mathcal{G}} \) (see (1)). To overcome this difficulty, we use the Markov Chain Monte Carlo (MCMC) algorithms that were introduced in [29]. Although the algorithms in [29] apply to networks with edge weights and/or node weights, we only use simplified forms of them that we give in Algorithms MP and MG. Additionally, Algorithm MP with the option \( \text{AcceptProb} = \text{Approximate} \) is a novel algorithm of the present paper. Using these MCMC sampling algorithms, we generate a sequence \((x_t)_{t \geq 0}\) of homomorphisms \( F \to \mathcal{G} \) such that the distribution of \( x_t \) converges to \( \pi_{F \to \mathcal{G}} \) under some mild conditions on \( \mathcal{G} \) and \( F \) [30, Thm. 5.7].

In the pivot chain (Algorithm MP with \( \text{AcceptProb} = \text{Exact} \)), for each update \( x_t \mapsto x_{t+1} \), the pivot \( x_t(1) \) first performs a random-walk move on \( \mathcal{G} \) (see (7)) to move to a new node \( x_{t+1}(1) \in V \). It accepts this move with a suitable acceptance probability (see (8)) according to the Metropolis–Hastings algorithm (see, e.g., [25, Sec. 3.2]). After the move of \( x_t(1) \mapsto x_{t+1}(1) \), we sample each \( x_{t+1}(i) \in V \) for \( i \in \{2,3,\ldots,k\} \) successively from the appropriate conditional distribution (see (9)). This ensures that the desired distribution \( \pi_{F \to \mathcal{G}} \) in (1) is a stationary distribution of the resulting Markov chain. In the Glauber chain, we pick one node \( i \in [k] \) of \( F \) uniformly at random, and we resample its location \( x_t(i) \in V(\mathcal{G}) \) at time \( t \) to \( \ell = x_{t+1}(i) \in V \) from the correct conditional distribution in (10) (see Figure 5a). See [25, Sec. 3.3] for background on the Metropolis–Hastings algorithm and Glauber-chain MCMC sampling.

Let \( \Delta \) denote the maximum degree (i.e., number of neighbors) of the nodes in the network \( \mathcal{G} = (V,A) \). We also say that the network \( \mathcal{G} \) itself has a maximum degree of \( \Delta \). The Glauber chain has an efficient local update (with a computational complexity of \( O(\Delta) \)), but it converges quickly to the stationary distribution \( \pi_{F \to \mathcal{G}} \) only for networks that are dense enough so that two homomorphisms that differ at one node have a probability of at least \( 1/(2\Delta) \) to coincide after a single Glauber chain update. (See [29, Thm. 6.1] for a precise statement.)

By contrast, the pivot chain (see Algorithm MP with \( \text{AcceptProb} = \text{Exact} \)) has more computationally expensive local updates with a computational complexity of \( O(\Delta^{k-1}) \) (as discussed in [29, Remark 5.6]), but it converges as fast as a ‘lazy’ random walk on a network. (In such a random walk, each move has a chance to be rejected according to a Metropolis–Hastings algorithm; see [29, Thm. 6.2].) Experimentally, we find
the Glauber chain is slow, especially for sparse networks (e.g., for COVID PPI, which has an edge density of 0.0010, and UCLA, which has an edge density of 0.0037) and that the pivot chain is too expensive to compute for chain motifs with $k \geq 21$. As a compromise with a low computational complexity and fast convergence (as fast as the standard random walk), we employ an approximate pivot chain, which is Algorithm MP with the option $\text{AcceptProb} = \text{Approximate}$. Specifically, we compute the acceptance probability $\alpha$ in (8) only approximately and thereby reduce the computational cost to $O(\Delta)$. The compromise, which we discuss in the next paragraph, is that the stationary distribution of the approximate pivot chain may be slightly different from our target distribution $\pi_{F \rightarrow \mathcal{G}}$.

According to Proposition G.1, the stationary distribution for the approximate pivot chain is

$$
\hat{\pi}_{F \rightarrow \mathcal{G}}(x) := \frac{\prod_{i=1}^{k} A(x(i - 1), x(i))}{|V| \sum_{y_2, \ldots, y_k \in V} A(x(1), y_2) \prod_{l=3}^{k} A(y_{l-1}, y_l)}. 
$$

\textbf{Algorithm MP}. Pivot-Chain Update

1: \textbf{Input}: Symmetric network $\mathcal{G} = (V, A)$, motif $F = ([k], A_F)$, and homomorphism $x : F \rightarrow \mathcal{G}$
2: \textbf{Parameters}: $\text{AcceptProb} \in \{\text{Exact}, \text{Approximate}\}$
3: \textbf{Do}: $x' \leftarrow x$
4: \hspace{1em} If $\sum_{c \in V} A(x(1), c) = 0$: \textbf{Terminate}
5: \hspace{1em} \textbf{Else}:
6: \hspace{2em} Sample $\ell \in V$ at random from the distribution
7: \hspace{3em} \hspace{1em} $p_1(w) = \frac{A(x(1), w)}{\sum_{c \in V} A(x(1), c)}$, \hspace{1em} $w \in V$
8: \hspace{2em} Compute the acceptance probability $\alpha \in [0, 1]$ by
9: \hspace{3em} $\alpha \left\{ \begin{array}{ll}
\min \left( \frac{\sum_{c \in [n]} A^{k-1}(\ell, c)}{\sum_{c \in [n]} A^{k-1}(x(1), c)}, 1 \right), & \text{if } \text{AcceptProb} = \text{Exact} \\
\min \left( \frac{\sum_{c \in V} A(c, x(1))}{\sum_{c \in V} A(x(1), c)}, 1 \right), & \text{if } \text{AcceptProb} = \text{Approximate}
\end{array} \right.$
10: \hspace{2em} Sample $U \in [0, 1]$ uniformly at random, independently of everything else
11: \hspace{2em} $\ell \leftarrow x(1)$ if $U > \lambda$ and $x'(1) \leftarrow \ell$
12: \hspace{2em} For $i = 2, 3, \ldots, k$:
13: \hspace{3em} Sample $x'(i) \in V$ from the distribution
14: \hspace{3em} $p_i(w) = \frac{A(x(i - 1), w)}{\sum_{c \in V} A(x(i - 1), c)}$, \hspace{1em} $w \in V$
15: \textbf{Output}: Homomorphism $x' : F \rightarrow \mathcal{G}$

\textbf{Algorithm MG}. Glauber-Chain Update

1: \textbf{Input}: Network $\mathcal{G} = (V, A)$, $k$-chain motif $F = ([k], A_F)$, and homomorphism $x : F \rightarrow \mathcal{G}$
2: \textbf{Do}: Sample $v \in [k]$ uniformly at random
3: \hspace{1em} Sample $z \in V$ at random from the distribution
4: \hspace{2em} $p(w) = \frac{1}{Z} \left( \prod_{u \in [k]} A(x(u), w)^{A_F(u, v)} \right) \left( \prod_{u \in [k]} A(w, x(u))^{A_F(v, u)} \right)$, \hspace{1em} $w \in V$
5: \hspace{2em} where $Z = \sum_{c \in V} \left( \prod_{u \in [k]} A(x(u), c)^{A_F(u, v)} \right) \left( \prod_{u \in [k]} A(c, x(u))^{A_F(v, u)} \right)$ is the normalization constant.
6: \hspace{2em} Define a new homomorphism $x' : F \rightarrow \mathcal{G}$ by $x'(w) = z$ if $w = v$ and $x'(w) = x(w)$ otherwise
7: \textbf{Output}: Homomorphism $x' : F \rightarrow \mathcal{G}$
In general, the distribution (11) is different from the desired target distribution \( \pi_{F \rightarrow G} \). Specifically, \( \pi_{F \rightarrow G}(x) \) is proportional only to the numerator in (11) and the sum in the denominator in (11) is a weighted count of homomorphisms \( y : F \rightarrow G \) for which \( y(1) = x(1) \). Therefore, under \( \hat{\pi}_{F \rightarrow G} \), we penalize the probability of each homomorphism \( x : F \rightarrow G \) according to the number of \( k \)-step walks in \( G \) that start from \( x(1) \in V \). (The exact acceptance probability in (8) neutralizes this penalty.) It follows that \( \hat{\pi}_{F \rightarrow G} \) is close to \( \pi_{F \rightarrow G} \) when the \( k \)-step-walk counts that start from each node in \( G \) do not differ too much for different nodes. For example, on degree-regular networks like lattices, such counts do not depend on the starting node, and it thus follows that \( \hat{\pi}_{F \rightarrow G} = \pi_{F \rightarrow G} \). Nevertheless, despite the potential discrepancy between \( \pi_{F \rightarrow G} \) and \( \hat{\pi}_{F \rightarrow G} \), the approximate pivot chain gives good results for the reconstruction and denoising experiments that we showed in Figures 3 and 4 in the main manuscript.

### Appendix D. Algorithm for Network Dictionary Learning (NDL)

**D.1. Algorithm overview and statement.** The essential idea behind our algorithm for NDL (see Algorithm 1) is as follows. We first sample a large number \( M \) of homomorphisms \( x_t : F \rightarrow G \) from \( \pi_{F \rightarrow G} \) and compute their corresponding mesoscale patches \( A_{x_t} \), for \( t \in \{1, \ldots, M\} \). These \( M \) mesoscale patches of \( G \) form the data set (a so-called ‘batch’) in which we apply a dictionary-learning algorithm. Specifically, we (column-wise) vectorize each of these \( k \times k \) matrices (using Algorithm A4) and obtain a \( k^2 \times M \) data matrix \( X \), and we then apply nonnegative matrix factorization (NMF) [19] to obtain a \( k^2 \times r \) nonnegative matrix \( W \) for some fixed integer \( r \geq 1 \) to yield an approximate factorization \( X \approx WH \) for some nonnegative matrix \( H \). From this procedure, we approximate each column of \( X \) by the nonnegative linear combination of the \( r \) columns of \( W \) with coefficients that are given by the \( r \)-th column of \( H \). Therefore, if we let \( L_i \) be the \( k \times k \) matrix that we obtain by reshaping the \( i \)-th column of \( W \) (using Algorithm A5), then \( \{L_1, \ldots, L_r\} \) is an approximate solution of (6). We will give the precise meaning of ‘approximate solution’ in Theorems G.2 and G.5.

The scheme in the paragraph above requires one to store all \( M \) mesoscale patches, entailing a memory requirement that is at least of order \( k^2M \). Because \( M \) should scale with the size of \( G \), this implies that we need unbounded memory to handle arbitrarily large networks. To address this issue, Algorithm 1 implements the above scheme in the setting of ‘online learning’, where subsets (so-called ‘minibatches’) of data arrive in a sequential manner and one does not store previous subsets of the data before processing new subsets. Specifically, at each iteration \( t = 1, 2, \ldots, T \), we only process a sample matrix \( X_t \) that is smaller than the full matrix \( X \) and includes only \( N \ll M \) mesoscale patches, where one can take \( N \) to be independent of the network size. Instead of the standard NMF algorithms for a fixed matrix [20], we use an ‘online’ NMF algorithm [30, 32] that applies to sequences of matrices, where the intermediate dictionary matrices \( W_t \) that we obtain by factorizing the sample matrix \( X_t \) typically improves as we iterate (see [30, 32]). In Algorithm 1, we give a full implementation of the NDL algorithm.

We now explain how the NDL algorithm works. It combines one of the three MCMC algorithms — a pivot chain (in which we use Algorithm MP with \texttt{AcceptProb} = \texttt{Exact}), an approximate pivot chain (in which we use Algorithm MP with \texttt{AcceptProb} = \texttt{Approximate}), and a Glauber chain (in which we use Algorithm MG) — for motif sampling that we presented in Section C with the online NMF algorithm of [30]. Suppose that we have an undirected and binary graph \( G = (V, A) \) and a \( k \)-chain motif \( F = ([k], A_F) \). We satisfy the requirement in Algorithm 1 that there exists at least one homomorphism \( F \rightarrow G \) (as long as \( G \) has at least one edge), so we can find an initial homomorphism \( x_0 : F \rightarrow G \) by rejection sampling (see Algorithm A3). At each iteration \( t = 1, 2, \ldots, T \), the motif-sampling algorithm generates a sequence \( x_t : F \rightarrow G \) of \( N \) homomorphisms and corresponding mesoscale patches \( A_{x_t} \) (see Figure 5a), which we summarize as the \( k^2 \times N \) data matrix \( X_t \). The online NMF algorithm in (12) learns a nonnegative factor matrix \( W_t \) of size \( k^2 \times r \) by improving the previous factor matrix \( W_{t-1} \) with respect to the new data matrix \( X_t \). It is an ‘online’ NMF algorithm because it factorizes a sequence \( (X_t)_{t \in \{1, \ldots, T\}} \) of data matrices, rather than a single matrix as in conventional NMF algorithms [20]. During this entire process, the algorithm only needs to store auxiliary matrices \( P_t \) and \( Q_t \) of fixed sizes \( r \times r \) and \( r \times k^2 \), respectively; it does not need the previous data matrices \( X_1, \ldots, X_{t-1} \). Therefore, NDL is efficient in memory and scales well with network size. Moreover, NDL is applicable to time-dependent networks because of its online nature, although we do not pursue this direction in the present paper.
Algorithm 1. Network Dictionary Learning (NDL)

1: **Input:** Network $\mathcal{G} = (V, A)$
2: **Parameters:** $F = ([k], A_F)$ (a motif), $T \in \mathbb{N}$ (the number of iterations), $N \in \mathbb{N}$ (the number of homomorphisms per iteration), $r \in \mathbb{N}$ (the number of latent motifs), $\lambda \geq 0$ (the coefficient of an $L_1$-regularizer)
3: **Options:** mask $\in \{\text{Id}, \text{NF}\}$, MCMC $\in \{\text{Pivot, PivotApprox, Glauber}\}$
4: **Requirement:** There exists at least one homomorphism $F \rightarrow \mathcal{G}$
5: **Initialization:**
   6: Sample a homomorphism $x : F \rightarrow \mathcal{G}$ using the rejection sampling (see Algorithm A3)
   7: $W = (k^2 \times r)$ matrix of independent entries that we sample uniformly from $[0, 1]$
   8: $P_0 = $ matrix of size $r \times r$ whose entries are 0
   9: $Q_0 = $ matrix of size $r \times k^2$ whose entries are 0
10: For $t = 1, 2, \ldots, T$:
11: **MCMC update and sampling mesoscale patches:**
12: Successively generate $N$ homomorphisms $x_{N(t-1)+1}, x_{N(t-1)+2}, \ldots, x_{Nt}$ by applying
   - Algorithm MP with AcceptProb = Exact if MCMC = Pivot
   - Algorithm MP with AcceptProb = Approximate if MCMC = PivotApprox
   - Algorithm MG with AcceptProb = Glauber if MCMC = Glauber
13: For $s = N(t - 1) + 1, \ldots, Nt$:
14: $A_{x_s} \leftarrow k \times k$ mesoscale patch of $\mathcal{G}$ that is induced by $x_s$ (see (2)) with
   - $\Phi_{F,x_s} = \begin{cases} \text{identity mask in (3)}, & \text{if mask = Id} \\ \text{no-folding mask in (4)}, & \text{if mask = NF} \end{cases}$
15: $X_t \leftarrow k^2 \times N$ matrix whose $j^{th}$ column is $\text{vec}(A_{x_t})$ with $\ell = N(t - 1) + j$
   (where $\text{vec}(\cdot)$ denotes the vectorization operator defined in Algorithm A4)
16: **Single iteration of online nonnegative matrix factorization:**
   
   \[
   \begin{align*}
   H_t & \leftarrow \arg\min_{H \in \mathbb{R}^{k^2 \times N}} \|X_t - W_{t-1}H\|_F^2 + \lambda\|H\|_1 \\
   P_t & \leftarrow (1 - t^{-1})P_{t-1} + t^{-1}H_t H_t^T \\
   Q_t & \leftarrow (1 - t^{-1})Q_{t-1} + t^{-1}H_t X_t^T \\
   W_t & \leftarrow \arg\min_{W \in C_{\text{dict}}} \|W\|_{R_{\geq 0}^{k^2 \times r}} \text{tr}(WPW^T) - 2\text{tr}(WQ_t) \quad \text{(using Algorithm A2)}
   \end{align*}
   \]

   where $C_{\text{dict}} = \{W \in \mathbb{R}_{\geq 0}^{k^2 \times r} | \text{columns of W have Frobenius norm at most } 1\}$
17: **Output:** Network dictionary $W_T \in \mathbb{R}_{\geq 0}^{k^2 \times r}$

In (12), we solve convex optimization problems to find matrices $H_t \in \mathbb{R}^{r \times N}$ and $W_t \in \mathbb{R}^{k^2 \times r}$. The first subproblem in (12) is a coding problem. Given two matrices $X_t$ and $W_{t-1}$, we seek to find a factor matrix (i.e., a 'code matrix') $H_t$ such that $X_t \approx W_{t-1}H_t$. The parameter $\lambda \geq 0$ is an $L_1$-regularizer, which encourages $H_t$ to have a small $L_1$ norm. One can solve the coding problem efficiently by using Algorithm A1 or one of a variety of existing algorithms (e.g., LARS [6], LASSO [54], or feature-sign search [21]). The second and third lines in (12) update the 'aggregate matrices' $P_{t-1} \in \mathbb{R}^{r \times r}$ and $Q_{t-1} \in \mathbb{R}^{r \times k^2}$ by taking a weighted average of them with the new information $X_t H_t^T \in \mathbb{R}^{r \times r}$ and $H_t X_t^T$, respectively. We weight the old aggregate matrices by $1 - t^{-1}$ and the new information by $t^{-1}$. By induction, we obtain $P_{t} = t^{-1} \sum_{s=1}^{t} H_s H_s^T$ and $Q_{t} = t^{-1} \sum_{s=1}^{t} H_s X_s^T$. We use the updated aggregate matrices, $P_t$ and $Q_t$, in the last subproblem in (12). This problem is called the dictionary-update problem and yields $W_t$. This is a constrained quadratic problem, and we can solve it using projected gradient descent (see Algorithm A2). In all of our experiments,
we take the compact and convex constraint set \( R_{k^2 \times r}^{\geq 0} \) to be the set of \( W \in R_{k^2 \times r}^{\geq 0} \) whose columns have a Frobenius norm of at most 1 (as required in (6)).

D.2. Dominance scores of latent motifs. In this subsection, we introduce a quantitative measurement of the ‘prevalence’ of latent motifs in the network dictionary \( W_T \) that we compute using NDL (see Algorithm 1) for a network \( G \).

Recall that the output of the NDL algorithm for a network \( G \) using a \( k \)-chain motif is a network dictionary \( W_T \), which consists of \( r \) latent motifs \( \mathcal{L}_1, \ldots, \mathcal{L}_r \) of size \( k \times k \). Recall as well that the algorithm computes data matrices \( X_1, \ldots, X_T \) of size \( k^2 \times N \). Suppose that we have code matrices \( H_1^*, \ldots, H_T^* \) such that \( X_t \approx W_T H_t^* \) for all \( t \in \{1, \ldots, T\} \). More precisely, we let

\[
H_t^* = \arg \min_{H \geq 0} \left( \|X_t - W_T H\|^2 + \lambda \|H\|_1 \right),
\]

where we take the \( \arg \min \) over all \( H \in R_{k^2 \times N}^{\geq 0} \). The columns of \( H_t^* \) encode how to nonnegatively combine the latent motifs in \( W_T \) to approximate the mesoscale patches in \( X_t \in R_{k^2 \times N}^{\geq 0} \), so the rows of \( H_t^* \) encode the linear coefficients of each latent motif in \( W_T \) that we use for approximating the columns of \( X_t \). Consequently, the mean inner products of the rows of \( H_t \) encode the mean usage of the latent motifs in \( W_T \) in \( G \). This motivates us to consider the following mean Gramian matrix [13]:

\[
P_T^* := \frac{1}{T} \sum_{t=1}^{T} H_t^*(H_t^*)^T \in R^{r \times r}.
\]

We then can take the square root of the diagonal entries of \( P_T^* \) to give us the mean prevalences of the latent motifs in \( W_T \) in \( G \).
Computing $P^*_T$ requires one to store the previous data matrices $X_1, \ldots, X_T$ and compute $H^*_1, \ldots, H^*_T$ by solving (13) for $t \in \{1, \ldots, T\}$; this is a very expensive computation. To address this issue, we use the aggregate matrix $P_T$ that we compute as part of Algorithm 1. Therefore, it does not require any extra computation. Note that $P^*_T$ is an approximation of $P^*_T$ because the defining equation of $H^*_t$ is the same as that of $H^*_t$ in (13) with $W_T$ replaced by $W_{t-1}$. However, the approximation error between $P^*_T$ and $P_t$ vanishes as $T \to \infty$ under mild conditions. Specifically, under the hypothesis of Theorems G.2 and G.5, $W_t$ converges almost surely to some limiting dictionary. It follows that $\|P^*_T - P_T\|_F \to 0$ almost surely as $T \to \infty$.

In Figure 6 of the main manuscript, we showed the second-most dominant latent motifs for twelve networks at the scales $k = 6, 11, 21, 51, 101$. In Figure 6, we show the most dominant latent motifs for the same sets of networks and scales. See Figures 8, 9, 10, and 14 in Section I for all 25 latent motifs at all five scales. For many of these networks (e.g., Harvard, UCLA, arXiv, and ER1) and at all five scales, the most dominant latent motifs in Figure 6 are close to the adjacency matrix of the $k$-chain itself. In these examples, the entries in the first superdiagonal and subdiagonal overwhelm the rest of the entries. However, for all networks except ER1, the second-most dominant latent motifs in Figure 6 reveal more interesting mesoscale structures (e.g., communities in Harvard, MIT, UCLA, and arXiv and bipartition for edges that are not in the 11-chain in Coronavirus PPI) than the most dominant latent motifs in Figure 6.

D.3. Influence of masking and MCMC algorithms on latent motifs. Our NDL algorithm (see Algorithm 1) uses mesoscale patches $A_k$ with a mask that consists of either the identity mask (3) or the no-folding mask (4), but one can generalize it for any choice of mask. The original NDL algorithm [30, Algorithm 1] corresponds to Algorithm 1 with the option mask = Id, the one with identity mask for mesoscale patches. In Section B, we discussed that the no-folding mask improves the interpretability of the positive entries in
the mesoscale patches. Consequently, it also improves the interpretability of the latent motifs that we learn from them using Algorithm 1.

![Image of network motifs](image)

**Figure 7.** Comparison of $r = 25$ latent motifs that we learn from Coronavirus PPI using the NDL algorithm (see Algorithm 1) in four different settings, which arise from the choices $\text{mask} \in \{\text{Id}, \text{NF}\}$ and $\text{MCMC} \in \{\text{PivotApprox}, \text{Glauber}\}$. The choices $\text{mask} = \text{Id}$ and $\text{mask} = \text{NF}$ indicate that the NDL algorithm uses mesoscale patches (2) with the identity mask (3) and the no-folding mask (4), respectively. The choices $\text{MCMC} = \text{PivotApprox}$ and $\text{MCMC} = \text{Glauber}$ indicate that the NDL algorithm uses the Glauber chain (see Algorithm MG) and the approximate pivot chain (see Algorithm MP with $\text{AcceptProb} = \text{Approximate}$), respectively. The other parameter values are $\lambda = 1$, $N = 100$, and $T = 100$. We use black squares for 1 entries and white squares for 0 entries. The numbers underneath the latent motifs give their dominance scores.

In Figure 7, we compare $r = 25$ latent motifs that we learn from Coronavirus PPI four different combinations of masks and MCMC motif-sampling. In Figure 7c,d, we see the latent motifs that we learn with the identity mask have large off-chain entries that dominate the on-chain entries. (See Section B.3 for definitions of on-chain and off-chain entries.) The most dominant latent motifs appear as complete bipartite networks, in which each node in one set is adjacent to all nodes in the other set in the bipartition, with 21 nodes. This is counterintuitive, as Coronavirus PPI has a very low edge density of 0.0010. However, as we see in Figure 7a,b, the latent motifs that we learn with the no-folding mask (4) have comparatively sparser off-chain entries, which are dominated by the on-chain entries. However, by comparing Figure 7a with Figure 7b and Figure 7c with Figure 7d, we also see that the choice of the MCMC algorithm between the approximate pivot chain and the Glauber chain for Algorithm 1 may not affect the latent motifs as much as whether we use the identity mask or the no-folding mask.

### Appendix E. Algorithms for Network Denoising and Reconstruction (NDR)

**E.1. Algorithm overview and statement.** The standard pipeline for image denoising and reconstruction [7, 33, 35] is to uniformly randomly sample a large number of $k \times k$ overlapping patches of an image and then average their associated approximations at each pixel to obtain a reconstructed version of the original image. A reasonable network analog of this pipeline is as follows. Given a network $G = (V, A)$, a motif $F = ([k], A_F)$, and a network dictionary with latent motifs $[\mathcal{L}_1, \ldots, \mathcal{L}_r]$, we uniformly randomly sample a large number $T$ of homomorphisms $x_t : F \rightarrow G$. To simplify the present discussion, we assume that each $x_t$ uses $k$ distinct nodes of $\mathcal{G}$ in its image $V^t := \{x_t(a) \in V \mid a \in \{1, \ldots, k\}\}$. (We do not make this assumption elsewhere in the paper.) This yields $T$ $k$-node subnetworks $G^t = (V^t, A(t))$, where $A(t)$ is the mesoscale patch $A_{x_t}$ in (2). We then approximate each weight matrix $A(t)$ by a nonnegative linear combination $\hat{A}(t)$ of the latent motifs $\mathcal{L}_i$. We then define a network $G_{\text{recons}} = (V, A_{\text{recons}})$, where we set $A_{\text{recons}}(p, q)$ for each $p, q \in V$ to be the mean of the approximate weights $\hat{A}(t)(p, q)$ for all $t \in \{1, \ldots, M\}$ such that $p, q \in V^t$.

Our network denoising and reconstruction (NDR) algorithm (see Algorithm 2) builds on the idea in the preceding paragraph. Suppose that we have a network $G = (V, A)$, a motif $F = ([k], A_F)$, and a network
dictionary $W$ that consists of $r$ nonnegative $k \times k$ matrices $L_1, \ldots, L_r$. First, because uniformly randomly sampling a homomorphism $x_t : F \rightarrow G$ is not as straightforward as uniformly randomly sampling $k \times k$ patches of an image, we generate a sequence $(x_t)_{t \in \{0, \ldots, T\}}$ of homomorphisms using a MCMC motif-sampling algorithm (see Algorithms MP and MG). For each $t \geq 0$, we approximate the mesoscale patch $A_{x_t}$ (see (2)) by a nonnegative linear combination of latent motifs $L_i$, and we then take the mean of the values of each entry $A(a, b)$ up to time $t$.

**Algorithm 2.** Network Denoising and Reconstruction (NDR)

1: **Input:** Network $\mathcal{G} = (V, A)$, and network dictionary $W \in \mathbb{R}_{\geq 0}^{k^2 \times r}$
2: **Parameters:** $F = ([k], A_F)$ (a motif), $T \in \mathbb{N}$ (number of iterations), $\lambda \geq 0$ (the coefficient of an $L_1$-regularizer), $\theta \in [0, 1]$ (an edge threshold)
3: **Options:** denoising $\in \{T, F\}$, mask $\in \{\text{Id}, \text{NF}\}$, MCMC $\in \{\text{Pivot}, \text{PivotApprox}, \text{Glauber}\}$
4: **Requirement:** There exists at least one homomorphism $F \rightarrow \mathcal{G}$
5: **Initialization:**
6: $A_{\text{recons}}, A_{\text{count}} : V^2 \rightarrow \{0\}$ (matrices with 0 entries)
7: Sample a homomorphism $x_0 : F \rightarrow \mathcal{G}$ by the rejection sampling in Algorithm A3
8: **For** $t = 1, 2, \ldots, T$:
9: **MCMC update and mesoscale patch extraction:**
10: $x_t \leftarrow$ Updated homomorphism that we obtain by applying
   
   Algorithm MP with $\text{AcceptProb} = \text{Exact}$, if MCMC = Pivot
   Algorithm MP with $\text{AcceptProb} = \text{Approximate}$, if MCMC = PivotApprox
   Algorithm MG with $\text{AcceptProb} = \text{Glauber}$, if MCMC = Glauber
11: $A_{x_t} \leftarrow k \times k$ mesoscale patch of $\mathcal{G}$ that is induced by $x_t$ (see (2)) with
   
   $\Phi_{F,x_t} = \begin{cases} \text{identity mask in (3)}, & \text{if mask = Id} \\ \text{no-folding mask in (4)}, & \text{if mask = NF} \end{cases}$
12: $X_t \leftarrow k^2 \times 1$ matrix that we obtain by vectorizing $A_{x_t}$ (using Algorithm A4)
13: **Mesoscale reconstruction:**
14: 
   \[ \begin{cases} \hat{X}_t \leftarrow X_t \text{ and } \hat{W} \leftarrow W, & \text{if denoising = F} \\ \hat{X}_t \leftarrow (X_t)_{\text{off}} \text{ and } \hat{W} \leftarrow (W)_{\text{off}} \text{ using Algorithm 2a}, & \text{if denoising = T} \end{cases} \]
15: $H_t \leftarrow \arg\min_{H \in \mathbb{R}_{\geq 0}^{k \times 1}} (\|\hat{X}_t - \hat{W} H\|^2_2 + \lambda \|H\|_1)$ and $\hat{X}_t \leftarrow \hat{W} H_t$
16: $\hat{A}_{x_t,W} \leftarrow k \times k$ matrix that we obtain by reshaping the $k^2 \times 1$ matrix $\hat{X}_t$ using Algorithm A5
17: **Update global reconstruction:**
18: **For** $a, b \in \{1, \ldots, k\}:
19: \text{If (denoising = F or } A_F(a, b) = 0) \text{ and } \Phi_{F,x_t}(a, b) = 1:\n20: A_{\text{count}}(x_t(a), x_t(b)) \leftarrow A_{\text{count}}(x_t(a), x_t(b)) + 1
21: j \leftarrow A_{\text{count}}(x_t(a), x_t(b))
22: A_{\text{recons}}(x_t(a), x_t(b)) \leftarrow (1 - j^{-1}) A_{\text{recons}}(x_t(a), x_t(b)) + j^{-1} \hat{A}_{x_t,W}(x_t(a), x_t(b))$
23: **Output:** Reconstructed networks $\mathcal{G}_{\text{recons}} = (V, A_{\text{recons}})$ and $\mathcal{G}_{\text{recons}}(\theta) = (V, \mathbb{I}(A_{\text{recons}} > \theta))$

As in Algorithm 1, we require in Algorithm 2 that there exists at least one homomorphism $F \rightarrow \mathcal{G}$. This condition is satisfied when $F = ([k], A_F)$ is a chain motif and $\mathcal{G}$ has at least one edge; it holds for all of our experiments in the present paper. As in the first line of (12), the problem for finding $H_t$ in line 15 of Algorithm 2 is a standard convex problem, which one can solve by using Algorithm A1. There are two
Algorithm 2a. Off-Chain Projection

**Input:** Matrix $Y \in \mathbb{R}^{k \times m}$, motif $F = ([k], A_F)$

**Do:** Let $Y'$ be a $k \times k \times m$ tensor that we obtain by reshaping each column of $Y$ using Algorithm A5.
Let $Y''$ be a $k \times k \times m$ tensor that we obtain from $Y'$ by

$$Y''(a, b, c) = Y'(a, b, c)\mathbb{I}(A_F(a, b) = 0) \quad \text{for all} \quad a, b \in \{1, \ldots, k\} \text{ and } c \in \{1, \ldots, m\}$$

Let $Y_{\text{off}}$ be a $k \times m$ matrix that we obtain from $Y''$ by vectorizing each of its slices using Algorithm A4: $Y_{\text{off}}[:; c]$ for all $c \in \{1, \ldots, m\}$.

**Output:** Matrix $(Y)_{\text{off}} \in \mathbb{R}^{k \times m}$

variants of the NDR algorithm. The variant is specified by the Boolean variable $\text{denoising}$. The NDR algorithm with $\text{denoising} = F$ is identical to the network-reconstruction algorithm in [30, Algorithm 2], except for thresholding step. The NDR algorithm with $\text{denoising} = T$ is a new variant of NDR that we present in the present work for the purpose of network denoising.

E.2. Further discussion of the denoising variant of the NDR algorithm. We now give a detailed discussion of Algorithm 2 with $\text{denoising} = T$ for network-denoising applications. Recall that the network-denoising problem that we consider is to reconstruct a true network $G_{\text{true}} = (V, A)$ from an observed network $G_{\text{obs}} = (V, A')$. The scheme that we used to produce Figure 4 is the following:

D.1 Learn a network dictionary $W \in \mathbb{R}_{\geq 0}^{k \times r}$ from an observed network $G_{\text{obs}} = (V, A)$ using NDL (see Algorithm 1).

D.2 Compute a reconstructed network $G_{\text{recons}} = (V, A_{\text{recons}})$ using NDR (see Algorithm 2) with input $G_{\text{obs}} = (V, A)$ and $W$.

D.3 Fix an edge threshold $\theta \in [0, 1]$. If $G_{\text{obs}}$ is $G_{\text{true}}$ with additive (respectively, subtractive) noise, we classify each edge (respectively, non-edge) $(p, q)$ as ‘positive’ if and only if $A_{\text{recons}}(p, q) > \theta$.

The NDR algorithm was first introduced in [30]. The version of the NDR algorithm from [30] is Algorithm 2 with the options $\text{mask} = \text{Id}$ and $\text{denoising} = F$. It has a competitive performance for denoising $-50\%$ subtractive noise on the networks SNAP FB, H. sapiens, and arXiv in comparison to the methods Spectral Clustering [41], DeepWalk [45], LINE [50], and node2vec [11]. In all of these methods, one first obtains a 128-dimensional vector representation of the nodes in a network; this is called a ‘node embedding’ of the network. One then uses this node embedding to compute vector representations of the edges using binary operations such as the Hadamard product. (See [11] for details.) One can then use a binary-classification algorithm (e.g., a support vector machine [42]) to attempt to detect the false edges. Spectral Clustering uses the top 128 eigenvectors of the normalized Laplacian matrix of $G_{\text{obs}}$ to learn vector embeddings of the nodes. (See [51] for details.) The other three benchmark methods first generate sequences of nodes using random-walk sampling and then use a word-embedding technique (see, e.g., [39]) to learn a node embedding.

In Table 1, we compare the AUC scores of our NDR and NDL approach that we described at the beginning of Section E.2 for our network-denoising tasks to the AUC scores that one obtains using the above four existing methods. As was discussed in [30, Remark 4], a limitation of using NDR with $\text{mask} = \text{Id}$ and $\text{denoising} = F$ for network denoising is that one needs to invert what it means to classify successfully depending on whether noise is additive or subtractive. Specifically, [30] used network denoising with NDR for additive noise, but it is necessary to classify each non-edge $(p, q)$ as ‘positive’ if $A_{\text{recons}}(p, q) < \theta$. Therefore, the AUC scores for $-50\%$ noise in the fifth in Table 1 is ‘flipped’ and the same is true for the sixth row in the table for $\text{mask} = \text{NF}$ and $\text{denoising} = F$. Our NDR algorithm with $\text{denoising} = T$ addresses this directionality issue and allows us to use the unified classification scheme above for both additive and subtractive noise.

The idea behind NDR with $\text{denoising} = T$ is to handle an issue when denoising additive noise for sparse real-world networks that does not arise in the image-denoising setting. Suppose that we obtain $G_{\text{obs}}$ by adding some false edges to a sparse binary network $G_{\text{true}}$. The on-chain entries of the mesoscale patches $A_x$ are always equal to 1. Therefore, the latent motifs that we learn from $G_{\text{obs}}$ have constant on-chain entries (see, e.g., Figures 2 and 6). Consequently, a linear approximation of mesoscale patches $A_x$ of $G_{\text{obs}}$ of the latent
motifs that we learn from $G_{\text{obs}}$ cannot distinguish between true and false on-chain entries. Furthermore, because $G_{\text{obs}}$ is sparse, there are many fewer positive off-chain entries in $A_x$ than the number of on-chain entries of $A_x$. Therefore, linear approximations of $A_x$ using the latent motifs are likely to assign larger weights to reconstruct on-chain entries of $A_x$ than off-chain entries. The resulting reconstruction of $G_{\text{obs}}$ is thus similar to $G_{\text{obs}}$, and it is very hard to detect any false edges in $G_{\text{obs}}$. Using the option denoising = T prevents this issue by ignoring all on-chain entries both for each sampled mesoscale patch $A_x$ and for each latent motif in the network dictionary $W$ that we use for denoising. For example, using denoising = T instead of denoising = F for SNAP FB and ARXIV (see Table 1) yields a performance gain of about 10% for +50% noise.

Another issue is using NDL with denoising = F for denoising negative noise in sparse real-world networks. In many of our experiments in this situation, our network-denoising scheme in (D.1)–(D.3) seems to give ‘flipped’ ROC curves that lie below the diagonal line $y = x$ that represents the baseline ROC curve and the AUC score is thus close to 0 instead of 1. Therefore, in the reconstructed network $G_{\text{recons}}$, false non-edges have larger weights than true non-edges. For the six instances in italics (for denoising = F and −50% noise) in Table 1, we give AUC scores after flipping the ROC curves, such that we classify a non-edge as ‘positive’ if $A_{\text{recons}}(p, q) < \theta$, which is the opposite of (D.3) that we used for additive noises in all cases. It is unfortunate to have to use the opposite classification scheme for different types of noise, especially when one may not know the type of noise in advance. We suspect that the sparsity of real-world networks may lead to such ‘opposite directionality’; this phenomenon requires further investigation. In contrast to this situation, the ROC curves for denoising = T are always above the diagonal in all experiments in Table 1 (as well as in Figure 4), regardless of whether the noise is additive or subtractive, and we always obtain the AUC scores in Table 1 using the scheme in (D.3).

### Appendix F. Experimental details

#### F.1. Data sets. We now describe the eight real-world networks that we examined in the main manuscript:

| Algorithm       | SNAP FB Noise +50% | SNAP FB Noise −50% | H. sapiens Noise +50% | H. sapiens Noise −50% | ARXIV Noise +50% | ARXIV Noise −50% |
|-----------------|--------------------|--------------------|-----------------------|------------------------|------------------|------------------|
| Spectral Clustering | - 0.619            | - 0.492            | -                     | - 0.574               |                 |
| DeepWalk        | - 0.968            | - 0.744            | -                     | - 0.934               |                 |
| LINE            | - 0.949            | - 0.725            | -                     | - 0.890               |                 |
| NODE2VEC        | - 0.968            | - 0.772            | -                     | - 0.934               |                 |
| NDL + NDR (mk = Id, dn = F) | 0.845 0.907         | 0.709 0.861         | 0.793 0.934           |                      |
| NDL + NDR (mk = NF, dn = F) | 0.898 0.871         | 0.837 0.841         | 0.934 0.923           |                      |
| NDL + NDR (mk = Id, dn = T) | 0.943 0.980         | 0.677 0.861         | 0.882 0.952           |                      |
| NDL + NDR (mk = NF, dn = T) | 0.979 0.981         | 0.814 0.859         | 0.950 0.954           |                      |

Table 1. Area-under-the-curve (AUC) scores of the ROC curve for our network-denoising experiments using NDL (see Algorithm 1) and NDR (see Algorithm 2). As in Figure 4, we first use NDL to learn latent motifs from a corrupted network and then reconstruct the networks using NDR to assign a confidence value to each potential edge. We use the networks SNAP FB, H. sapiens, and ARXIV with −50% and +50% noise on the edges. In the last four rows, mk stands for the choice of mask in NDR and dn stands for the denoising approach in NDR. We use mask = NF for NDL in each of the last four rows. For both NDL and NDR, we use MCMC = ApproxPivot. The last row is a duplicate from Figure 4, and we obtain the results in rows five–seven using the same parameter choices as in Figure 4. For the last four rows except the six instances in italics (i.e., for dn = F and −50% noise), we use the classification scheme (D.3). For these six instances, we compute the AUC of the ‘flipped’ ROC curve, as the algorithm technically takes on the opposite classification task. When dn = T, we do not flip the classification task. In the first four rows, we construct 128-dimensional vector representations of the networks using the indicated methods, and we then use them for edge classification.
(1) **Caltech**: This connected network, which is part of the Facebook100 data set [56] (and previously was studied as part of the Facebook5 data set [55]), has 762 nodes and 16,651 edges. Nodes represent users in the Facebook network of Caltech on one day in fall 2005, and edges represent Facebook ‘friendships’ between these users.

(2) **MIT**: This connected network, which is part of the Facebook100 data set [56], has 6,402 nodes and 251,230 edges. Nodes represent users in the Facebook network of MIT on one day in fall 2005, and edges represent Facebook ‘friendships’ between these users.

(3) **UCLA**: This connected network, which is part of the Facebook100 data set [56], has 20,453 nodes and 747,604 edges. Nodes represent users in the Facebook network of UCLA on one day in fall 2005, and edges represent Facebook ‘friendships’ between these users.

(4) **Harvard**: This connected network, which is part of the Facebook100 data set [56], has 15,086 nodes and 824,595 edges. Nodes represent users in the Facebook network of MIT on one day in fall 2005, and edges represent Facebook ‘friendships’ between these users.

(5) **SNAP Facebook (SNAP FB)** [24]: This connected network has 4,039 nodes and 88,234 edges. This network is a Facebook network that has been used as a benchmark example for edge inference [11].

(6) **arXiv ASTRO-PH (arXiv)** [11, 23]: This network has 18,722 nodes and 198,110 edges. Its largest connected component has 17,903 nodes and 197,031 edges. We use the full network in our experiments. It is a collaboration network between authors of papers in astrophysics that were posted to the arXiv preprint server. Nodes represent scientists and edges indicate coauthorship relationships.

(7) **Coronavirus PPI (Coronavirus)**: This connected network, which was curated by theBiogrid.org [10, 44, 52] from 142 publications and preprints, has 1,546 proteins that are related to coronaviruses and 2,481 protein–protein interactions (in the form of physical contacts) between them. We downloaded this data set on 24 July 2020. Among the 2,481 interactions, 1,546 are for SARS-CoV-2 and were reported by 44 publications and preprints; the rest are related to coronaviruses that cause Severe Acute Respiratory Syndrome (SARS) or Middle Eastern Respiratory Syndrome (MERS).

(8) **Homo sapiens PPI (H. sapiens)** [11, 44, 53]: This network has 24,407 nodes and 390,420 edges. Its largest connected component has 24,379 nodes and 390,397 edges. We use the full network in our experiments. The nodes represent proteins in the organism *Homo sapiens*, and the edges represent physical interactions between these proteins.

We now describe the six synthetic networks that we examined in the main manuscript:

(9) **ER1 and ER2**: An Erdős–Rényi (ER) network [8, 40], which we denote by ER(n, p), is a random-graph model. The parameter n is the number of nodes and the parameter p is the independent, homogeneous probability that each pair of distinct nodes has an edge between them. The network ER1 is an individual graph that we draw from ER(5000, 0.01), and ER2 is an individual graph that we draw from ER(5000, 0.02).

(10) **WS1 and WS2**: A Watts–Strogatz (WS) network, which we denote by WS(n, k, p), is a random-graph model to study the small-world phenomenon [40, 57]. In the version of WS networks that we use, we start with an n-node ring network in which each node is adjacent to its k nearest neighbors. With independent probability p, we then remove and rewire each edge to a pair of distinct nodes that we choose uniformly at random. The network WS1 is an individual graph that we draw from WS(5000, 50, 0.05), and WS2 is an individual graph that we draw from WS(5000, 50, 0.10).

(11) **BA1 and BA2**: A Barabási–Albert (BA) network, which we denote by BA(n, m), is a random-graph model with a linear preferential-attachment mechanism [2, 40]. In the version of BA networks that we use, we start with m isolated nodes and we introduce new nodes with m new edges each that attach preferentially (with a probability that is proportional to node degree) to existing nodes until we have a total of n nodes. The network BA1 is an individual graph that we draw from BA(5000, 25), and BA2 is an individual graph that we draw from WS(5000, 50).

F.2. **Figures 2, 6, 8, 9, 10, 11, 12, 13, and 14**: These figures give latent motifs of the networks that we described in Section F.1 using Algorithm 1 with various parameter choices. In all of these figures, we use a chain motif with the corresponding network $F = ([k], A_F)$ for $T = 100$ iterations, $N = 100$ homomorphisms.
per iteration, an $L_1$-regularizer with coefficient $\lambda = 1$, a mask of mask $= \text{Id}$, and an MCMC motif-sampling algorithm of MCMC $= \text{Pivot}$. We specify the number $r$ of latent motifs and the scale $k$ in the caption of each figure.

F.3. **Figure 1.** In this figure, we illustrate latent motifs that we learn from UCLA and Caltech and we compare them to an image dictionary. We use the following parameters in Algorithm 1 to generate the results in Figures 2 and 6. We use a chain motif with the corresponding network $F = ([k], A_F)$, a scale $k = 21$ for $T = 100$ iterations, $N = 100$ homomorphisms per iteration, $r = 25$ latent motifs, an $L_1$-regularizer with coefficient $\lambda = 1$, a mask of mask $= \text{Id}$, and an MCMC motif-sampling algorithm of MCMC $= \text{Pivot}$. The pivot chain in Algorithm MP uses AcceptProb $= \text{Approximate}$. The image dictionary for the artwork Cycle in Figure 1 uses an algorithm that is similar to Algorithm 1, except that we uniformly randomly sample $21 \times 21$ square patches of the image instead of $k \times k$ mesoscale patches of a network.

F.4. **Figure 7.** This figure compares $r = 25$ latent motifs at scale $k = 21$ that we learn from Coronavirus PPI using Algorithm 1 with MCMC motif-sampling algorithms MCMC $\in \{\text{Glauber}, \text{ApproxPivot}\}$ and masks mask $\in \{\text{Id}, \text{NF}\}$. We specify the parameters in Algorithm 1 that we use to generate the results in Figure 7 in the caption of that figure.

F.5. **Figure 3.** To generate Figure 3, we first apply the NDL algorithm (see Algorithm 1) to each network that we consider in the figure to learn $r = 25$ latent motifs for a chain motif with the corresponding network $F = ([k], A_F)$, a scale $k = 21$ for $T = 100$ iterations, $N = 100$ homomorphisms per iteration, an $L_1$-regularizer with coefficient $\lambda = 1$, a mask of mask $= \text{NF}$, and MCMC $= \text{PivotApprox}$. For each self-reconstruction $X \leftarrow X$ (see the caption of Figure 3), we apply the NDR algorithm (see Algorithm 2) to a chain motif with the corresponding network $F = ([k], A_F)$, a scale $k = 21$ for $T = n \ln n$ iterations (where $n$ is the number of nodes in the network), $N = 100$ homomorphisms per iteration, $r = 25$ square patches, an $L_1$-regularizer with coefficient $\lambda = 0$ (i.e., no regularization), a mask of mask $= \text{Id}$, a MCMC motif-sampling algorithm of MCMC $= \text{PivotApprox}$, and denoising $= F$. For each cross-reconstruction $Y \leftarrow X$ (see the caption of Figure 3), we apply the NDR algorithm (see Algorithm 2) to a chain motif with the corresponding network $F = ([k], A_F)$, a scale $k = 21$ for $T = n \ln n$ time steps (where $n$ is the number of nodes in the network), $N = 100$ homomorphisms, an edge-threshold value of $\theta = 0.5$, an $L_1$-regularizer with coefficient $\lambda = 0$ (i.e., no regularization), a mask of mask $= \text{NF}$, an MCMC motif-sampling algorithm of MCMC $= \text{PivotApprox}$, and denoising $= F$. We use multiple different choices of the number $r$ of latent motifs; we indicate them in the caption of Figure 3.

In the main manuscript, we mentioned that the following claims follow from the reconstruction accuracies that we reported in Figure 3 in conjunction with the latent motifs in Figures 1, 2, 8, and 10. We now provide their justifications.

1. The mesoscale structure of Caltech is rather different from those of Harvard, UCLA, and MIT at scale $k = 21$.
   - In Figure 3c, we observe that the accuracy of the cross-reconstruction $Y \leftarrow X$ is consistently higher for $X \in \{\text{UCLA, Harvard, MIT}\}$ than for $X = \text{Caltech}$ for all values of $r$. For instance, at $r = 9$, we can reconstruct UCLA with more than 90% accuracy and we can reconstruct Harvard and MIT with more than 80% accuracy. However, the latent motifs that we learn from Caltech for $r = 9$ gives only about 80% accuracy for reconstructing UCLA and only about 70% accuracy for reconstructing MIT and UCLA. This indicates that Caltech has significantly different mesoscale structure than the other three universities’ Facebook networks at scale $k = 21$. Indeed, from Figures 1, 2, 6, and 8, we see that the $r = 25$ latent motifs of Caltech at scale $k = 21$ have larger off-chain entries than those of UCLA, MIT, and Harvard.

2. The mesoscale structure of Caltech at scale $k = 21$ has a higher dimension than those of the other three universities’ Facebook networks.
   - Consider the cross-reconstructions Caltech $\leftarrow X$ for $X \in \{\text{UCLA, Harvard, MIT}\}$ in Figure 3b. The accuracy with $r = 9$ for the latent motifs that we learn from Caltech itself is as low as 64%. By contrast, it 80% or higher for the self-reconstructions $X \leftarrow X$ for the Facebook networks of the other universities. In other words, $r = 9$ latent motifs at scale $k = 21$ cannot approximate the mesoscale structures of Caltech as well as those of the other three universities’ Facebook networks.
networks. This indicates that the dimension of the mesoscale structures of CALTECH at scale $k = 21$ is larger than those of the other three universities’ Facebook networks.

(3) The networks $BA_1$ and $BA_2$ are better than $ER_1$, $ER_2$, $WS_1$, and $WS_2$ at capturing the mesoscale structures of MIT, HARVARD, and UCLA at scale $k = 21$.

- From the reconstruction accuracies for $Y \leftarrow X$ in Figure 3b,c, where $X$ is one of the six synthetic networks ($ER_i$, $WS_i$, and $BA_i$ for $i \in \{1, 2\}$), we observe that the two BA networks have higher accuracies than the networks from the ER and WS models for $Y \in \{UCLA, HARVARD, MIT\}$. This suggests that the mesoscale structures of UCLA, HARVARD, and MIT may be more similar in some respects to those of $BA_i$ than those of $ER_i$ and $WS_i$. The latent motifs of $BA_1$ in Figures 2 and 8 at $k = 21$ have characteristics that we also observe in UCLA, HARVARD, and MIT. (Specifically, they have hub nodes and off-chain entries that are much smaller (so they are lighter in color) than the on-chain entries.) By contrast, in Figure 14, we see that the latent motifs for $ER_1$ have sparse but seemingly randomly distributed off-chain connections and the ones for $WS_1$ have strongly interconnected communities of about 10 nodes (see the diagonal block of black entries). These patterns differ from the ones that we observe in the latent motifs of UCLA, MIT, and HARVARD (see Figure 8).

(4) If we uniformly sample a walk of $k = 21$ nodes, then it is more likely that there are communities with 10 or more nodes in the induced subnetwork for CALTECH than is the case for UCLA, HARVARD, and MIT.

- From the reconstruction accuracies for $Y \leftarrow X$ in Figure 3b,c, where $X$ is one of the six synthetic networks ($ER_i$, $WS_i$, and $BA_i$ for $i \in \{1, 2\}$), we observe that WS networks outperform both BA and ER networks in reconstructing CALTECH, but they are one of the lowest-performing networks for reconstructing the Facebook networks of the other three universities. In other words, nonnegative linear combinations of the latent motifs of $WS_i$ can better approximate the mesoscale patches of CALTECH than they can for those of UCLA, HARVARD, and MIT. Recall that most latent motifs of $WS_i$ at scale $k = 21$ have blocks of black entries of size $10 \times 10$ or larger (see Figure 10); these correspond to adjacency matrices of communities with 10 or more nodes. Therefore, such community structure should be more likely to occur in subnetworks that are induced from uniform samples of $k = 21$-node walks in CALTECH than from such samples in UCLA, HARVARD, or MIT.

F.6. Figure 4. To generate Figure 4, we first apply the NDL algorithm (see Algorithm 1) to each corrupted network that we consider in the figure to learn $r = 25$ latent motifs for a chain motif with the corresponding network $F = ([k], A_F)$, a scale $k = 21$ for $T = 400$ iterations, $N = 1000$ homomorphisms per iteration, an $L_1$-regularizer with coefficient $\lambda = 1$, a mask of $\text{mask} = \text{NF}$, and an MCMC motif-sampling algorithm of $\text{MCMC} = \text{PivotApprox}$. The NDR algorithm (see Algorithm 2) that we use to generate the results in Figure 4 uses $r = 25$ latent motifs for a chain motif with the corresponding network $F = ([k], A_F)$, a scale $k = 21$ for $T = 400,000$ iterations for $H. \text{SAPIENS}$ and $T = 200,000$ iterations for all other networks, an $L_1$-regularizer with coefficient $\lambda = 1$, a mask of $\text{mask} = \text{NF}$, $\text{MCMC} = \text{PivotApprox}$, and $\text{denoising} = T$. For Figure 4, we did not conduct the denoising experiment for CORONAVIRUS PPI with $\sim 50\%$ noise because the resulting network (with 1,536 nodes and 1,232 edges) cannot be connected. (To be connected, its spanning trees need to have 1,535 edges.)

We obtain the AUCs in Figure 4 for the methods NODE2VEC [11], DEEPWALK [45], and LINE [50] for the task of denoising 50% subtractive noise for SNAP FACEBOOK, H. SAPIENS, and ARXIV from [11]. In [11], the ROCs were computed from a ‘balanced test set’ that was chosen uniformly at random from all sets that include all of the $|E|/2$ false non-edges and $|E|/2$ true non-edges, where $|E|$ denotes the number of edges in the network. By contrast, we compute our ROCs in Figure 4 using all of the $|E|/2$ false non-edges and all of true non-edges, which is often very large. For instance, recall that ARXIV has $|V| = 18,752$ nodes and $|E| = 198,110$ edges. Therefore, after deleting $|E|/2$ edges to create a corrupted network, there are $|E|/2 = 99,055$ false non-edges and $|V|(|V| − 1)/2 − |E| = 175,049,171$ true non-edges. Such an imbalance in a data set does not affect the ROCs and hence does not affect the AUCs. Consequently, the true-positive rates and false-positive rates do not change even if we compute them after independently sampling $|E|/2$
true non-edges. Therefore, our results are directly comparable to the AUCs in [11] after uniformly sampling a balanced test set that consists of $|E|/2$ true non-edges and $|E|/2$ false non-edges.

**Appendix G. Convergence Analysis**

In this section, we give rigorous convergence guarantees for our main algorithms for NDL and NDR. In [30, Corollary 6.1], Lyu et al. obtained a convergence guarantee of the original NDL algorithm ([30, Algorithm 1]) for non-bipartite networks $G$ with the MCMC motif-sampling algorithms $\text{MCMC} \in \{\text{Pivot, Glauber}\}$. Theorem G.2(i) in the present paper establishes the same result for the NDL algorithm (see Algorithm 1) that uses mesoscale patches (2) with either the identity mask or the no-folding mask (4). Theorem G.2(ii) gives a similar convergence result for the NDL algorithm with the approximate pivot chain (i.e., for $\text{MCMC} = \text{PivotApprox}$). Theorem G.5 establishes a similar convergence result for NDL for bipartite networks $G$. Theorems G.7 and G.10 establish convergence and error bounds for the NDR algorithm (see Algorithm 2).

Except for Theorem G.2(i), all of these theoretical results are novel results of the present paper.

Let $F = ([k], A_F)$ be the $k$-chain motif, and let $G = (V, A)$ be a network. Let $\Omega \subseteq V^{[k]}$ denote the set of all homomorphisms $x : F \to G$. Algorithm 1 generates three stochastic sequences. The first one is the sequence $(x_t)_{t \geq 0}$ of homomorphisms $F \to G$ that are generated by the pivot chain (see Algorithm MP). The second one is the sequence $(X_t)_{t \geq 0}$ of $k^2 \times N$ data matrices whose columns encode $N$ mesoscale patches of $G$. More precisely, for each $y_1, \ldots, y_n \in \Omega$, we write $\Psi(y_1, \ldots, y_n) \in \mathbb{R}_{\geq 0}^{k^2 \times N}$ for the $k^2 \times N$ matrix whose $i$th column is the vectorization (using Algorithm A4) of the corresponding $k \times k$ mesoscale patch $A_{y_i}$ of $G$. For each $y_0 \in \Omega$, define

$$X^{(N)}(y_0) := \Psi(y_1, \ldots, y_N) \in \mathbb{R}_{\geq 0}^{k^2 \times N},$$

where we generate $y_1, \ldots, y_N$ using the pivot chain when it starts at $y_0$. It then follows that $X_t = X^{(N)}(X_{Nt})$ for each $t \geq 1$, where $x_{Nt}$ is the state of the pivot chain at time $Nt$. For the third (and final) sequence that we generate using Algorithm 1, let $(W_t)_{t \geq 0}$ denote the sequence of dictionary matrices, where we define each $W_t = W_t(y_0)$ via (12) with an initial homomorphism $x_0 : F \to G$ that we sample using Algorithm A3.

**G.1. Convergence of the approximate pivot chain.** We prove Proposition G.1, which states the convergence of the approximate pivot chain and gives an explicit formula for its unique stationary distribution.

**Proposition G.1.** Fix a network $G = (V, A)$ and the $k$-chain motif $F = ([k], A_F)$. Let $(x_t)_{t \geq 0}$ denote a sequence of homomorphisms $x_t : F \to G$ that we generate using the approximate pivot chain (in which we use Algorithm MP with $\text{AcceptProb} = \text{Approximate}$). Suppose that

(a) The weight matrix $A$ is ‘bidirectional’ (i.e., $A(a, b) > 0$ implies that $A(b, a) > 0$ for all $a, b \in V$) and the undirected and binary graph $(V, 1(A > 0))$ is connected and non-bipartite.

It then follows that $(x_t)_{t \geq 0}$ is an irreducible and aperiodic Markov chain with the unique stationary distribution $\pi_{F \to G}$ that we defined in (11).

**Proof.** We follow the proof of [29, Thm. 5.8]. Let $P : V^2 \to [0, 1]$ be a matrix with entries

$$P(a, b) := \frac{A(a, b)}{\sum_{c \in V} A(a, c)}, \quad a, b \in V.$$

This is the transition matrix of the standard random walk on the network $G$. By hypothesis (a), $P$ is irreducible and aperiodic. Additionally, it has the unique stationary distribution (see [25, Ch. 9])

$$\pi^{(1)}(u) := \frac{\sum_{c \in V} A(u, c)}{\sum_{c, c' \in V} A(c, c')}.$$

The approximate pivot chain generates a move $x_t(1) \mapsto x_{t+1}(1)$ of the pivot according to the distribution $P(x_t(1), \cdot)$. We accept this move of the pivot independently of everything else with the approximate acceptance probability $\alpha$ in (8). If we always accept each move of the pivot, then the pivot performs a random walk on $G$ with unique stationary distribution $\pi^{(1)}$. We compute the acceptance probability $\alpha$ using the Metropolis–Hastings algorithm (see [25, Sec. 3.3]), and we thereby modify the stationary distribution of the pivot from $\pi^{(1)}$ to the uniform distribution on $V$. (See the discussion in [29, Sec. 5].) Therefore, $(x_t(1))_{t \geq 0}$ is an irreducible and aperiodic Markov chain on $V$ that has the uniform distribution as its unique stationary
distribution. Because we sample the locations \( x_{t+1}(i) \in V \) of the subsequent nodes \( i = 2, 3, \ldots, k \) independently, conditional on the location \( x_{t+1}(1) \) of the pivot, it follows that the approximate pivot chain \( (x_t)_{t \geq 0} \) is also an irreducible and aperiodic Markov chain with a unique stationary distribution, which we denote by \( \hat{\pi}_{F \to G} \).

To compute the stationary distribution \( \hat{\pi}_{F \to G} \), we decompose \( x_t \) into return times of the pivot \( x_t(1) \) to a fixed node \( x_1 \in V \) in \( G \). Specifically, let \( \tau(j) \) be the \( j \)th return time of \( x_t(1) \) to \( x_1 \). By the independence of sampling \( x_t \) over \( \{2, \ldots, k\} \) for each \( t \), the strong law of large numbers yields

\[
\lim_{M \to \infty} \frac{1}{M} \sum_{j=1}^{M} \mathbb{1}(x_{\tau(j)}(2) = x_2, \ldots, x_{\tau(j)}(k) = x_k) = \frac{\prod_{i=2}^{k} A(x_{i-1}, x_i)}{\sum_{y_2, \ldots, y_k \in V} \prod_{i=2}^{k} A(x_{i}, y_{i}) A(y_{i+1}, y_{i+2}) \ldots A(y_{k-1}, y_{k})}.
\]

For each fixed homomorphism \( x : F \to G, i \mapsto x_i \), we use the Markov-chain ergodic theorem (see, e.g., [5, Theorem 6.2.1 and Example 6.2.4] or [38, Theorem 17.1.7]) to obtain

\[
\hat{\pi}_{F \to G}(x) = \lim_{N \to \infty} \frac{1}{N} \sum_{t=0}^{N} \mathbb{1}(x_t = x) = \lim_{N \to \infty} \frac{\sum_{t=0}^{N} \mathbb{1}(x_t = x) \cdot \sum_{t=0}^{N} \mathbb{1}(x_t(1) = x_1)}{N} = P\left(x_1(2) = x_2, \ldots, x_k(1) = x_k \mid x_1(1) = x_1\right) \pi^{(1)}(x_1) \frac{\prod_{i=1}^{k} A(x_{i-1}, x_i)}{\sum_{y_2, \ldots, y_k \in V} \prod_{i=2}^{k} A(x_{i}, y_{i}) A(y_{i+1}, y_{i+2}) \ldots A(y_{k-1}, y_{k})} \frac{1}{|V|}.
\]

This proves the assertion. \( \square \)

G.2. Convergence of the NDL algorithm. Recall the problem statement for NDL in (6). Informally, we seek to learn \( r \) latent motifs \( L_1, \ldots, L_r \in \mathbb{R}_{\geq 0}^{k \times k} \) to minimize the expectation of the error of approximating the mesoscale patch \( \mathcal{A}_x \) by a nonnegative combination of the motifs \( \mathcal{L}_i \), where \( x : F \to G \) is a random homomorphism that we sample from the distribution \( \pi_{F \to G} (1) \). We reformulate this problem as the following matrix-factorization problem, which generalizes (6). Let \( \mathcal{C}_{\text{dict}} \) denote the set of all matrices \( W \in \mathbb{R}_{\geq 0}^{k^2 \times r} \) whose columns have a Frobenius norm of at most 1. The matrix-factorization problem is then

\[
\arg \min_{W \in \mathcal{C}_{\text{dict}}} \mathbb{E}_{x \sim \pi_{F \to G}} \left[ \ell(X^{(N)}(x), W) \right],
\]

where we define the loss function

\[
\ell(X, W) := \inf_{H \in \mathbb{R}_{\geq 0}^{N \times N}} \|X - WH\|_F^2 + \lambda \|H\|_1, \quad X \in \mathbb{R}^{k^2 \times N}, W \in \mathbb{R}^{k^2 \times r}.
\]

The parameters \( N \in \mathbb{N} \) and \( \lambda \geq 0 \) appear in Algorithm 1. The former is the number of homomorphisms that we sample at each iteration of Algorithm 1, and the latter is the coefficient of an \( L_1 \)-regularizer that we use to find the code matrix \( H_t \) in (12). In the special case of \( N = 1 \) and \( \lambda = 0 \), the problem (14) is equivalent to the problem (6), because \( X^{(1)}(x) \) and the columns of \( W \) are vectorizations (using Algorithm A4) of the mesoscale patch \( \mathcal{A}_x \) and the latent motifs \( \mathcal{L}_1, \ldots, \mathcal{L}_r \), respectively.

Theorems G.2 and G.5 imply that our NDL algorithm (see Algorithm 1) finds a sequence \( (W_t)_{t \geq 0} \) of dictionary matrices such that almost surely \( W_t \) is asymptotically a stationary point of the objective function \( f \) in the optimization problem (14). The objective function \( f \) is non-convex, so it is generally difficult to find global optimum of \( f \). In practice, however, such stationary points have often been good enough for practical applications, such as image restoration [7, 35]. We find that this is also the case for our network-denoising problem (see Figure 4).

We are now ready to state our first convergence result for the NDL algorithm (see Algorithm 1).

**Theorem G.2** (Convergence of the NDL Algorithm for Non-Bipartite Networks). Let \( F = ([k], A_F) \) be the \( k \)-chain motif, and let \( G = (V, A) \) be a network that satisfies the following properties:
(a) The weight matrix $A$ is ‘bidirectional’ (i.e., $A(a,b) > 0$ implies that $A(b,a) > 0$ for all $a,b \in V$) and the undirected and binary graph $(V, 1(A > 0))$ is connected and non-bipartite.

(b) For all $t \geq 0$, there exists a unique solution $H_t$ in (12).

(c) For all $t \geq 0$, the eigenvalues of the positive semidefinite matrix $A_t$ that is defined in (12) are at least as large as some constant $\kappa_1 > 0$.

Let $(W_t)_{t \geq 0}$ denote the sequence of dictionary matrices that we generate using Algorithm 1. The following claims hold:

(i) For $\text{MCMC} \in \{\text{Pivot, Glauber}\}$, we have almost surely as $t \to \infty$ that $W_t$ converges to the set of stationary points of the objective function $f$ that we defined in (14). Furthermore, if $f$ has finitely many stationary points in $C^{\text{dict}}$, we then have that $W_t$ converges to a single stationary point of $f$ almost surely as $t \to \infty$.

(ii) For $\text{MCMC} = \text{PivotApprox}$, we have almost surely as $t \to \infty$ that $W_t$ converges to the set of stationary points of the objective function

$$\tilde{f}(W) := \mathbb{E}_{x \sim \tilde{\pi}_{F \to G}} \left[ \ell(X^{(N)}(x), W) \right],$$

where the distribution $\tilde{\pi}_{F \to G}$ is defined in (11). Furthermore, if $\tilde{f}$ has finitely many stationary points in $C^{\text{dict}}$, we then have that $W_t$ converges to a single stationary point of $\tilde{f}$ almost surely as $t \to \infty$.

Remark G.3. Assumptions (a)–(c) in Theorem G.2 are all reasonable and are easy to satisfy. Assumption (a) is satisfied if $G$ is undirected, binary, and connected, which is the case for all of our examples in the present paper. Assumptions (b) and (c) are standard assumptions in the study of online dictionary learning [30, 31, 32]. For instance, (b) is a common assumption in methods such as layer-wise adaptive-rate scaling (LARS) [6] that aim to find good solutions to problems of the form (15). Additionally, in practice, one can verify (c) experimentally after a few iterations of Algorithm 1 for a reasonable choice of the initial dictionary (e.g., $r$ samples of mesoscale patches). See [32, Sec. 4.1] and [30, Sec. 4.1] for more detailed discussions of these assumptions.

Remark G.4. It is also possible to slightly modify both the optimization problem (14) and our NDL algorithm so that Theorem G.2 holds for the modified problem and the algorithm without needing to assume (b) and (c). The modified problem is

$$\arg \min_{W \in C^{\text{dict}} \subseteq \mathbb{R}^{K \times N}} \mathbb{E}_{x \sim \pi} \left[ \inf_{H \in \mathbb{R}^{K \times N}} \|X - WH\|_F^2 + \lambda\|H\|_1 + \kappa_1\|H\|_F^2 + \lambda_0\|W\|_F^2 \right],$$

where $\pi = \tilde{\pi}_{F \to G}$ if $\text{MCMC} = \text{PivotApprox}$ and $\pi = \pi_{F \to G}$ otherwise. Note that (16) is the same as (14) with additional quadratic penalization terms for both $H$ and $W$ in the loss function $\ell$ that we defined in (15). By contrast, consider the modification of the NDL algorithm (see Algorithm 1) in which the objective function for $H_t$ in (12) has the additional term $\lambda\|H\|_F^2$ and we replace $P_t$ in (12) by $P_t + \kappa_1 I$. Assuming that $\lambda, \kappa_1 > 0$, the modified objective function for $H_t$ is strictly convex, so $H_t$ is uniquely defined. Therefore, it satisfies the uniqueness condition (b) in Theorem G.2. Additionally, the smallest eigenvalue of each matrix $P_t$ that we compute using the modified NDL algorithm has a lower bound of $\kappa_1$ for all $t$, so it satisfies condition (c) in Theorem G.2. One can then show that the same statement as Theorem G.2 for the modified problem (16) and the NDL algorithm hold without assumptions (b) and (c). The argument, which we omit, is almost identical to the one for Theorem G.2.

Proof of Theorem G.2. The proof of the first part of (i) is identical to the proof of [30, Corollary 6.1]. For the first part of (ii), we can use the same essential argument as in the proof of [30, Corollary 6.1]. However, because our assertion is for the approximate pivot chain that we propose in the present article (see Algorithm MP with $\text{AcceptProb} = \text{Approximate}$), we need to use Proposition G.1 (instead of [29, Prop. 5.8]) to establish irreducibility and convergence of our Markov chain. The proof of the second parts of both (i) and (ii) are identical.

We give a detailed proof of (ii). Let $\pi = \tilde{\pi}_{F \to G}$ if $\text{MCMC} = \text{PivotApprox}$ and $\pi = \pi_{F \to G}$ otherwise (see (1) and (11)). We define $(x_t)_{t \geq 0}$, $(X_t)_{t \geq 1}$, and $(W_t)_{t \geq 0}$ as before. We use a general convergence result for online NMF for Markovian data [30, Theorem 4.1].
We first observe that the matrices \( X_t \in \mathbb{R}^{k \times N} \) that we compute in line 15 of Algorithm 1 do not necessarily form a Markov chain, because the forward evolution of the Markov chain depends both on the induced mesoscale patches and on the actual homomorphisms \((x_s)_{N(1-\ell) < s < Nt}\). However, if one considers the sequence \( \overline{X}_t := \{X_t, X_{Nt}\} \), then \( \overline{X}_t \) forms a Markov chain. Specifically, the distribution of \( X_{t+1} \) given \( X_t \) depends only on \( X_{Nt} \) and \( A \). Indeed, \( X_{Nt} \) and \( A \) determine the distribution of the homomorphisms \((x_s)_{N1 < s < N(t+1)}\), which in turn determine the \( k^2 \times N \) matrix \( X_{t+1} \).

With assumption (a), [29, Theorems 5.7 and 5.8] and Proposition G.1 imply that the Markov chain \( (x_t)_{t \geq 0} \) of homomorphisms \( F \rightarrow \mathcal{G} \) is a finite-state Markov chain that is irreducible and aperiodic with a unique stationary distribution \( \pi \) (see (1)). This implies that the \( N \)-tuple of homomorphisms \((x_s)_{N(1-\ell) < s < Nt}\) also yields a finite-state, irreducible, and aperiodic chain with a unique stationary distribution. Consequently, the Markov chain \( (X_t)_{t \geq 0} \) that we defined above is also a finite-state, irreducible, and aperiodic chain with a unique stationary distribution. In this setting, one can regard Algorithm 1 as the online NMF algorithm in [30] for the input sequence \( X_t = \varphi(\overline{X}_t) \), where \( \varphi(X, x) = X \) is the projection onto its first coordinate. Because \( \overline{X}_t \) takes only finitely-many values, the range of \( \varphi \) is bounded. This verifies all hypotheses of [30, Theorem 4.1], so the first part of (ii) follows.

Now suppose the there are finitely many stationary points \( W_1, \ldots, W_m \) of \( f \) in \( C^{\text{dict}} \). Noting that \( \|W_{t-1} - W_t\|_F = O(1/t) \) (see [30, Prop. 7.5]), from the first part of (ii) (which we proved above), it follows that \( W_t \) converges to \( W_i \) for some unique \( i \in \{1, \ldots, m\} \) almost surely. \( \square \)

Our second convergence result for the NDL algorithm is similar to Theorem G.2, but it concerns the case in which the network \( \mathcal{G} \) is bipartite. Suppose that \( \mathcal{G} \) is bipartite and that \( F \) is the \( k \)-chain motif. Let \( V_1 \cup V_2 = V \) be a bipartition of \( \mathcal{G} \). We can define a subset \( \Omega_1 \subseteq \Omega \) of homomorphisms \( F \rightarrow \mathcal{G} \) by \( x \in \Omega_1 \) if and only if \( x(1) \in V_1 \). Let \( \Omega_2 = \Omega \setminus \Omega_1 \). In this case, neither the pivot chain nor the Glauber chain is irreducible on \( (x_t)_{t \geq 0} \) as Markov chains with state space \( \Omega \). However, they are irreducible when we restrict them to each \( \Omega_i \) with a unique stationary distribution \( \pi^{(i)}_{F \rightarrow \mathcal{G}} \). (See the proof of [29, Theorem 5.7].) More explicitly, we compute

\[
\pi^{(i)}_{F \rightarrow \mathcal{G}}(x) := \mathbb{P}_{y \sim \pi^{(i)}_{F \rightarrow \mathcal{G}}} (y = x \mid y \in \Omega_i) = \frac{1}{Z_i} \left( \prod_{j \in \{2, \ldots, k\}} A(x(j-1), x(j)) \right), \quad i \in \{1, 2\}
\]

where \( Z_i = \sum_{x \in \Omega_i} \prod_{j \in \{1, \ldots, k\}} A(x(j), x(j+1)) \) and we defined the homomorphism density \( Z \) just below (1).

We define two associated conditional expected loss functions:

\[
f^{(i)}(W) := \mathbb{E}_{x \sim \pi^{(i)}_{F \rightarrow \mathcal{G}}} \left[ \ell(X^{(N)}(x), W) \right] = \mathbb{E}_{x \sim \pi^{(i)}_{F \rightarrow \mathcal{G}}} \left[ \ell(X^{(N)}(x), W) \mid x \in \Omega_i \right], \quad i \in \{1, 2\}.
\]

where the equality follows from the first equality in (17). Because the Markov chain \( (x_t)_{t \geq 0} \) stays in \( \Omega_i \) if it is initialized in \( \Omega_i \), the above conditional expected loss functions \( f^{(i)} \) are the natural objective function to minimize (instead of the full expected loss function \( f \) that serves as the objective function in (14)). Similarly, for the approximate pivot chain, we define the distributions \( \hat{\pi}^{(i)}_{F \rightarrow \mathcal{G}} \) and the conditional expected loss functions \( \hat{f}^{(i)} \) as follows. For \( \hat{\pi}_{F \rightarrow \mathcal{G}} \) as in (11), let

\[
\hat{\pi}^{(i)}_{F \rightarrow \mathcal{G}}(x) := \mathbb{P}_{y \sim \hat{\pi}_{F \rightarrow \mathcal{G}}} (y = x \mid y \in \Omega_i) = \frac{1}{|V_i|} \left( \prod_{2 \leq j \leq k} A(x(j-1), x(j)) \right) \sum_{y_2, \ldots, y_k \in V} A(x(1), y_2) \prod_{i=3}^k A(y_{i-1}, y_i) = \frac{|V|}{|V_i|} \hat{\pi}^{(i)}_{F \rightarrow \mathcal{G}}(x),
\]

\[
f^{(i)}(W) := \mathbb{E}_{x \sim \hat{\pi}^{(i)}_{F \rightarrow \mathcal{G}}} \left[ \ell(X^{(N)}(x), W) \right] = \mathbb{E}_{x \sim \hat{\pi}^{(i)}_{F \rightarrow \mathcal{G}}} \left[ \ell(X^{(N)}(x), W) \mid x \in \Omega_i \right], \quad i \in \{1, 2\}.
\]

We now state our second convergence result for the NLD algorithm. This result is for bipartite networks. A convergence result that is analogous to the one in Theorem 1 holds for the associated conditional expected loss function. This implies that one can initialize homomorphisms in each \( \Omega_i \) and compute sequences \( W_t^{(i)} \) (with \( i \in \{1, 2\} \)) of dictionary matrices to learn stationary points of both associated conditional expected loss functions \( f^{(i)} \). However, when the chain motif \( F = ([k], A_F) \) has an even number of nodes, one only needs to compute one sequence of dictionary matrices, because one can obtain the other sequence by the
algebraic operation of taking a ‘mirror image’ of a given square matrix. More precisely, define a map \( \text{Flip} : \mathbb{R}^{k \times r} \rightarrow \mathbb{R}^{k \times r} \) that maps \( W \rightarrow W^\text{rev} \), such that the \( j \)-th column \( W(:,j) \) of \( W \) is defined by

\[
X(:,j) := \text{vec} \circ \text{rev} \circ \text{reshape}(W(:,j)), \quad j \in \{1, \ldots, r\},
\]

where \( W(:,j) \) denotes the \( j \)-th column of \( W \), \( \text{reshape} : \mathbb{R}^{k^2} \rightarrow \mathbb{R}^{k \times k} \) is the reshaping operator that we defined in Algorithm A5, \( \text{rev} \) maps a \( k \times k \) matrix \( K \) to the \( k \times k \) matrix \((K_{ab})_{1 \leq a, b \leq k}\) with entries \( K_{ab} = K(k - a + 1, k - b + 1) \), and \( \text{vec} \) denotes the vectorization operator in Algorithm A4. Applying \( \text{Flip} \) twice gives the identity map.

**Theorem G.5** (Convergence of the NDL Algorithm for Bipartite Networks). Let \( F = ([k], A_F) \) be the \( k \)-chain motif, and let \( G = (V, A) \) be a network that satisfies the the following properties:

1. \( A \) is symmetric and the undirected and binary graph \((V, \Xi(A > 0))\) is connected and bipartite.
2. For all \( t \geq 0 \), there exists a unique solution \( H_t \) in (12).
3. For all \( t \geq 0 \), the eigenvalues of the positive semidefinite matrix \( A_t \) in (12) are at least as large as some constant \( \kappa_1 > 0 \).

Let \( (W_t)_{t \geq 0} \) denote the sequence of dictionary matrices that we generate using Algorithm 1. We then have the following properties hold:

(i) Suppose that \( \text{MCMC} \in \{\text{Pivot, Glauber}\} \). For each \( i \in \{1,2\} \), conditional on \( x_0 \in \Omega_i \), the sequence \( W_t \) of dictionary matrices converges almost surely as \( t \to \infty \) to the set of stationary points of the associated conditional expected loss function \( f^{(i)} \) that we defined in (18). If \( \text{MCMC} = \text{PivotApprox} \), then the same statement holds with \( f^{(i)} \) replaced by the function \( \hat{f}^{(i)} \) that we defined in (19). We also assume that \( f^{(1)} \) (respectively, \( \hat{f}^{(1)} \)) have only finitely many stationary points in \( \mathbb{R}^\text{dict} \). It then follows that \( W_t \) converges to a single stationary point of \( f^{(1)} \) (respectively, \( \hat{f}^{(1)} \)) almost surely as \( t \to \infty \).

(ii) Suppose that \( \text{MCMC} = \text{Glauber} \) in Algorithm 1 and that \( k \) is even. Assume that \( x_0 \in \Omega_1 \). It then follows that, almost surely, the sequences of dictionary matrices \( W_t \) and \( \overline{W}_t \) converge simultaneously to the sets of stationary points of the expected loss functions \( f^{(1)} \) and \( f^{(2)} \), respectively. Moreover, \( f^{(1)}(W_t) = f^{(2)}(\overline{W}_t) \) for all \( t \geq 0 \). We also assume that \( f^{(1)} \) (with \( i \in \{1,2\} \)) has only finitely many stationary points in \( \mathbb{R}^\text{dict} \). It then follows that both \( W_t \) and \( \overline{W}_t \) converge to a single stationary points of \( f^{(1)} \) and \( f^{(2)} \) almost surely as \( t \to \infty \).

**Proof.** We first prove (i). Fix \( j \in \{1,2\} \), and recall the conditional stationary distribution \( \pi^{(j)}_{F \rightarrow G} \) from (17). Conditional on \( x_0 \in \Omega_j \), the Markov chain \( (x_t)_{t \geq 0} \) is irreducible and aperiodic with a unique stationary distribution \( \pi^{(j)}_{F \rightarrow G} \). Recall that the conclusion of Theorem G.2 holds as long as the underlying Markov chain is irreducible. Therefore, \( W_t \) converges almost surely to the set of stationary points of the associated conditional expected loss function \( f^{(i)} \) that we defined in (18). The same argument verifies the case in which \( \text{MCMC} = \text{PivotApprox} \).

We now verify (ii). Define the notation \( \mu_j := \pi^{(i)}_{F \rightarrow G} \) and suppose that \( k \) is even. For each homomorphism \( \mathbf{x} : F \rightarrow G \), we define a map \( \mathbf{x} : [k] \rightarrow V \) by

\[
\mathbf{x}(j) := \mathbf{x}(k - j + 1) \quad \text{for all} \quad j \in \{1, \ldots, k\}.
\]

For even \( k \), we have that \( \mathbf{x} \in \Omega_1 \) if and only if \( \overline{\mathbf{x}} \in \Omega_2 \). Because \( A \) is symmetric, it follows that

\[
\prod_{j=1}^{k-1} A(\mathbf{x}(j), \mathbf{x}(j+1)) = \prod_{j=1}^{k-1} A(\mathbf{x}(j+1), \mathbf{x}(j)) = \prod_{j=1}^{k-1} A(\overline{\mathbf{x}}(j), \overline{\mathbf{x}}(j+1)).
\]

Therefore, \( Z_1 = Z_2 = Z/2 \). Consequently, for each \( \mathbf{x} \in \Omega_1 \), (17) implies that

\[
\mu_1(\mathbf{x}) = \mu_2(\overline{\mathbf{x}}) = 2\pi_{F \rightarrow G}(\mathbf{x}).
\]

Consider two Glauber chains, \((x_t)_{t \geq 0}\) and \((x_t')_{t \geq 0}\), where \( x_0 = y \) and \( x_0' = \overline{y} \). We evolve these two Markov chains using a common source of randomness so that individually they have Glauber-chain trajectories but they also satisfy the relation \( x_t' = \overline{x_t} \) for all \( t \geq 0 \). (This is typically called a ‘coupling argument’ in probability literature; see [25, Sec. 4.2].) Specifically, suppose that \( x_t' = \overline{x_t} \). For each update \( x_t \mapsto x_{t+1} \) and
\( x'_t \mapsto x'_{t+1} \), we choose a node \( v \in [k] \) uniformly at random and sample \( z \in V \) according to the conditional distribution (10). We define
\[
x_{t+1}(v) = z \quad \text{and} \quad x_{t+1}(u) = x_t(u) \quad \text{for} \quad u \neq v, \\
x'_{t+1}(k - v + 1) = z \quad \text{and} \quad x'_{t+1}(u) = x_t(u) \quad \text{for} \quad u \neq k - v + 1.
\]

We then have that \( x_t \mapsto x_{t+1} \) follows the Glauber-chain update in Algorithm MG. We also have the desired relation \( x'_{t+1} = x_{t+1} \) because
\[
x'_{t+1}(k - v + 1) = x_{t+1}(k - v) = x_t(k - v + 1), \\
x'_{t+1}(u) = x'_{t+1}(u) = x_t(k - u + 1) = x_{t+1}(k - u + 1) = x_t(u) \quad \text{for} \quad u \neq k - v + 1.
\]

Finally, we need to verify that \( x'_t \mapsto x'_{t+1} \) also follows the Glauber-chain update in Algorithm MG. It suffices to check that \( z \in V \) has the same distribution as \( x'_{t+1}(k - v + 1) \). Because \( v \) is uniformly distributed on \([k]\), so is \( k - v + 1 \). The distribution of \( z \in V \) is determined by
\[
p(z) \propto \begin{cases} 
A(z, x_t(2)) = A(z, x_t(k - 1)), & \text{if} \quad v = 1 \\
A(x_t(v - 1), z)A(z, x_t(v + 1)) = A(x_t(k - v), z)A(z, x_t(k - v + 2)), & \text{if} \quad v \in \{2, \ldots, k - 1\} \\
A(x_t(k - 1), z) = A(x_t(2), z), & \text{if} \quad v = k.
\end{cases}
\]

Because \( x'_t = x_t \), it follows that \( z \) is distributed as the conditional distribution (10) of \( x'_{t+1}(k - v + 1) \), as desired.

For the two Glauber chains, \( x_t \) and \( x'_t \), we observe that
\[
X^{(N)}(y) = X^{(N)}(\overline{y})
\]
almost surely. This result follows from the facts that \( x'_t = x_t \) for all \( t \geq 0 \) and \( \text{rev}(A_x) = A_{\overline{x}} \) for all \( x \in \Omega \). (See (2) for the definition of \( A_x \).) From this, we note that
\[
f^{(1)}(W) = \mathbb{E}_{x \sim \pi} \left[ \ell(X^{(N)}(x), W) \mid x \in \Omega_1 \right]
\]
\[
= \mathbb{E}_{x \sim \pi} \left[ \ell(X^{(N)}(\overline{x}), W) \mid x \in \Omega_2 \right]
\]
\[
= \mathbb{E}_{x \sim \pi} \left[ \ell(X^{(N)}(\overline{x}), \overline{W}) \mid x \in \Omega_2 \right]
\]
\[
= \mathbb{E}_{x \sim \pi} \left[ \ell(X^{(N)}(x), \overline{W}) \mid x \in \Omega_2 \right] = f^{(2)}(\overline{W}).
\]

The first and the last equalities use the second equality in (17). The second equality uses the fact that \( \ell(X, W) = \ell(\overline{X}, \overline{W}) \). The third equality follows from (21). The fourth equality follows from the change of variables \( x \mapsto \overline{x} \) and the fact that \( x \sim \pi \) if and only if \( \overline{x} \sim \pi \) (see (20)).

We now prove (ii). Its first part follows immediately from (i) and the above construction of the Glauber chains \( x_t \) and \( x'_t \) that satisfy \( x'_t = x_t \) for all \( t \geq 0 \). Specifically, let \( W_t = W_t(x_0) \) and \( W'_t = W'_t(x'_0) \) denote the sequences of dictionary matrices that we compute using Algorithm 1 with initial homomorphisms \( x_0 \) and \( x'_0 \), respectively. Suppose that \( x_0 \in \Omega_1 \), from which we see that \( x'_0 = x_0 \in \Omega_2 \). By (i), \( W_t \) and \( W'_t \) converge almost surely to the set of stationary points of the associated conditional expected loss functions \( f^{(1)} \) and \( f^{(2)} \), respectively. We complete the proof of the first part of (ii) by observing that, almost surely,
\[
W'_t = \overline{W_t} \quad \text{for all} \quad t \geq 0.
\]

The second part of (ii) follows immediately from (22).

We still need to verify (23). Roughly, the argument is that all \( k \times k \) mesoscale patches \( A_{\overline{x}_t} = \text{ref}(A_{x_t}) \) have the reversed row and column ordering from the original ordering, so \( k \times k \) latent motifs that we train on such matrices also have the reversed ordering of rows and columns. More concretely, one can check this claim by induction on \( t \) together with (21) and the uniqueness assumption (b). We omit the details.

**Remark G.6.** Our proofs of Theorems G.2 and G.5 do not depend on the particular choice of mask \( \Phi_{F:x} \) that we use to define mesoscale patches \( A_x \) in (2).
G.3. Convergence of the NDR algorithm. We prove convergence results for the NDR algorithm (see Algorithm 2) in Theorem G.7. Specifically, we show that the reconstructed network that we obtain using Algorithm 2 at iteration $t$ converges almost surely to some limiting network as $t \to \infty$, and we give a closed-form expression of the limiting network. We also give a bound for the 'distance' between the original network and the limiting reconstructed network in terms of a ‘fitness’ of the network dictionary that we use in the reconstruction algorithm. We measure this fitness using the expected $1$-norm between the sampled $k \times k$ mesoscale patch and its best nonnegative linear approximation using our network dictionary.

Let $\text{denoising}$ denote the Boolean variable in Algorithm 2. Fix a network $\mathcal{G} = (V, A)$, the $k$-chain motif $F = ([k], A_F)$, and a homomorphism $x : F \to \mathcal{G}$. Let $\Phi_{F,x} \in \{0, 1\}^{k \times k}$ denote the no-folding mask that we defined in (4). For each matrix $B : V^2 \to [0, \infty)$ and a node map $x : [k] \to V$, define the $k \times k$ matrix $B_x$ by

$$B_x(a, b) := B(x(a), x(b)) \Phi_{F,x}(a, b) \quad \text{for all} \quad a, b \in \{1, \ldots, k\}.$$  

If $B = A$, then $B_x = A_x$ equals the mesoscale patch of $\mathcal{G}$ that is induced by $x$ (see (2)). Additionally, given a network $\mathcal{G} = (V, A)$, a motif $F = ([k], A_F)$, a homomorphism $x : F \to \mathcal{G}$, and a nonnegative matrix $W \in \mathbb{R}_{\geq 0}^{k \times r}$, let $A_{x,W}$ denote the $k \times k$ matrix that we defined in line 11 of Algorithm 2. This matrix depends on the Boolean variable $\text{denoising}$. Recall that $A_{x,W}$ is a nonnegative linear approximation of $A_x$ that uses $W$. We introduce the event $(p, q) \xrightarrow{x} (a, b)$ using the following indicator function:

$$\mathbb{1} \left( (p, q) \xrightarrow{x} (a, b) \right) := \mathbb{1}(x(a) = p, x(b) = q) \mathbb{1}(\text{denoising} = F \text{ or } A_F(a, b) = 0) \Phi_{F,x}(a, b),$$

where $\Phi_{F,x}$ is the no-folding mask that we defined in (4). For each homomorphism $x : F \to \mathcal{G}$ and $p, q \in V$, we say that the pair $(p, q)$ is visited by $(a, b)$ through $x$ whenever the indicator on the left-hand side of (24) is 1. Additionally,

$$N_{pq}(x) := \sum_{a, b \in \{1, \ldots, k\}} \mathbb{1} \left( (p, q) \xrightarrow{x} (a, b) \right)$$

is the total number of visits to $(p, q)$ through $x$. When $N_{pq}(x) > 0$, we say that the pair $(p, q)$ is visited by $x$. In Algorithm 2, observe that both $A_{\text{count}}(p, q)$ and $A_{\text{recons}}(p, q)$ change at iteration $t$ if and only if $N_{pq}(x_t) > 0$.

Finally,

$$\Omega_{pq} := \{ x : F \to \mathcal{G} \mid N_{pq}(x) > 0 \}$$

is the set of all homomorphisms $x : F \to \mathcal{G}$ that visit the pair $(p, q)$.

Theorem G.7 (Convergence of the NDR Algorithm (see Algorithm 2) for Non-Bipartite Networks). Let $F = ([k], A_F)$ be the $k$-chain motif and fix a network $\mathcal{G} = (V, A)$ and a network dictionary $W \in \mathbb{R}_{\geq 0}^{k \times r}$. We use Algorithm 2 with inputs $\mathcal{G}$, $F$, and $W$ and the parameter value $T = \infty$. Let $\mathcal{G}_t = (V, A_t)$ denote the network that we reconstruct at iteration $t$, and suppose that $\mathcal{G}$ satisfies assumption (a) of Theorem G.2. Let $\pi = \pi_{F \to \mathcal{G}}$ if MCMC = PivotApprox and $\pi = \pi_{F \to \mathcal{G}}$ otherwise. The following statements hold:

(i) The network $\mathcal{G}_t$ converges almost surely to some limiting network $\mathcal{G}_\infty = (V, A_\infty)$ in the sense that

$$\lim_{t \to \infty} \mathcal{A}_t(p, q) = A_\infty(p, q) \in [0, \infty) \quad \text{almost surely for all} \quad p, q \in V.$$  

(ii) Let $A_\infty$ denote the limiting matrix in (i). For each $p, q \in V$, we then have that

$$A_\infty(p, q) = \sum_{y \in \Omega_{pq}} \left[ \sum_{a, b \in \{1, \ldots, k\}} \mathcal{A}_{y,W}(a, b) \mathbb{1} \left( (p, q) \xrightarrow{y} (a, b) \right) \right] \frac{\pi(y)}{E_{x \sim \pi} \left[ N_{pq}(x) \right]}. \quad (27)$$

(iii) Let $\hat{A}_\infty$ be as in (ii). For any network $\mathcal{G}' = (V, B)$, we have

$$\sum_{p, q \in V} \left| B(p, q) - A_\infty(p, q) \right| E_{x \sim \pi} \left[ N_{pq}(x) \right] \leq \begin{cases} E_{x \sim \pi} \left[ \| B_x - \hat{A}_\infty \|_1 \right], & \text{if denoising} = F \\ E_{x \sim \pi} \left[ \| B_x - \hat{A}_\infty \|_{1,F} \right], & \text{if denoising} = T, \end{cases}$$

where $\| R \|_{1,F} := \sum_{1 \leq a, b \leq k} |R(a, b)| (A_F(a, b) = 0)$ for each $R \in \mathbb{R}^{k \times k}$. 

(iv) Let \( \hat{A}_\infty \) be as in (ii) and suppose that denoising = \( F \). It then follows that
\[
\sum_{p, q \in V} \left| A(p, q) - \hat{A}_\infty(p, q) \right| \mathbb{E}_{x \sim \pi} \left[ N_{pq}(x) \right] \leq \sqrt{k} \mathbb{E} \left[ \ell(\text{vec}(A_x), W) \right]
\]
\[
\leq 2\sqrt{k} \left( \mathbb{E}_{x \sim \pi} \left[ \ell(\text{vec}(A_x), W) \right] \sup_{x : F \rightarrow G} \sqrt{\ell(\text{vec}(A_x), W)} \right)^{1/3}.
\]

Proof. Let \( x \) denote a random homomorphism \( F \rightarrow G \) with distribution \( \pi \), and let \( P \) and \( \mathbb{E} \) denote the associated probability measure and expectation, respectively.

We first verify (i) and (ii) simultaneously. Let \( (x_t)_{t \geq 0} \) denote the Markov chain that we generate during the reconstruction process (see Algorithm 2). We fix \( p, q \in V \) and let
\[
M_t := \sum_{s=1}^{t} \sum_{a, b \in \{1, \ldots, k\}} \mathbb{1}_{(p, q) \xrightarrow{x_s} (a, b)} = \sum_{s=1}^{t} N_{pq}(x_s),
\]
where we defined the indicator \( \mathbb{1}_{(p, q) \xrightarrow{x_s} (a, b)} \) in (24). The key observation is that
\[
\hat{A}_t(p, q) = \frac{1}{M_t} \sum_{s=1}^{t} \sum_{a, b \in \{1, \ldots, k\}} \hat{A}_{x_s; W}(a, b) \mathbb{1}_{(p, q) \xrightarrow{x_s} (a, b)}
\]
\[
= \sum_{a, b \in \{1, \ldots, k\}} \frac{1}{M_t} \sum_{s=1}^{t} \mathbb{1}_{\Omega_{pq}}(s, y) \hat{A}_{y; W}(a, b) \mathbb{1}_{(p, q) \xrightarrow{x_s} (a, b)}
\]
\[
= \sum_{y \in \Omega_{pq}} \sum_{a, b \in \{1, \ldots, k\}} \hat{A}_{y; W}(a, b) \mathbb{1}_{(p, q) \xrightarrow{x_s} (a, b)} \frac{1}{M_t} \sum_{s=1}^{t} \mathbb{1}(x_s = y).
\]

With assumption (a), the Markov chain \( (x_t)_{t \geq 0} \) of homomorphisms \( F \rightarrow G \) is irreducible and aperiodic with \( \pi \) (see (1)) as its unique stationary distribution. By the Markov-chain ergodic theorem (see, e.g., [5, Theorem 6.2.1 and Example 6.2.4] or [38, Theorem 17.1.7]), it follows that
\[
\lim_{t \rightarrow \infty} \frac{1}{M_t} \sum_{s=1}^{t} \mathbb{1}(x_s = y) = \frac{P(x = y)}{\mathbb{E}[N_{pq}(x)]}.
\]

This proves both (i) and (ii).

We now verify (iii). For each \( a, b \in \{1, \ldots, k\} \) and \( p, q \in \{1, \ldots, n\} \), let \( \Omega_{ab \rightarrow pq} \) denote the set of all homomorphisms \( x : F \rightarrow G \) such that \( \mathbb{1}_{(p, q) \xrightarrow{x} (a, b)} = 1 \). By changing the order of the sums, we rewrite the formula in (27) as
\[
\hat{A}_\infty(p, q) = \sum_{a, b \in \{1, \ldots, k\}} \mathbb{1}_{(p, q) \xrightarrow{x_s} (a, b)} \sum_{y \in \Omega_{ab \rightarrow pq}} \hat{A}_{y; W}(a, b) \frac{P(x = y)}{\mathbb{E}[N_{pq}(x)]}.
\]

For each \( a, b \in \{1, \ldots, k\} \), define the indicator function
\[
\mathbb{1}_{ab} := \mathbb{1}_{\text{denoising} = F} \text{ or } A_F(a, b) = 0.
\]

Observe that
\[
\mathbb{E}[N_{pq}(x)] = \sum_{a, b \in \{1, \ldots, k\}} \mathbb{1}(x(a) = p, x(b) = q) \mathbb{1}_{ab} \Phi_{F, x}
\]
\[
= \sum_{a, b \in \{1, \ldots, k\}} \mathbb{1}_{ab} \sum_{y \in \Omega_{ab \rightarrow pq}} P(x = y).
\]

Indeed, the indicator \( \mathbb{1}_{ab} \) does not depend on the homomorphism \( y \) and \( \Phi_{F, y} = 1 \) if \( y \in \Omega_{ab \rightarrow pq} \). We then can calculate
\[
\sum_{p, q \in V} \left| B(p, q) - \hat{A}_\infty(p, q) \right| \mathbb{E}[N_{pq}(x)]
\]
We define the notation \( M \) as the last expression in (32) is minimized when \( \delta \) is small (i.e., when there are only finitely many homomorphisms \( \phi \) in the right-hand side of (28), theorems 2 and 5 guarantee that our NDL algorithm (see Algorithm 1) always finds a locally optimal network dictionary. Indeed, from these theorems, the NDL algorithm with.

This verifies (iii).

Finally, we prove (iv). First, by the Cauchy–Schwarz inequality,

\[
E \left[ \| A_{\mathcal{X}} - \hat{A}_{\mathcal{Y}W} \|_1 \right] \leq \sqrt{\delta} E \left[ \| A_{\mathcal{X}} - \hat{A}_{\mathcal{Y}W} \|_F \right] \leq \sqrt{\delta} E \left[ \sqrt{\ell(\text{vec}(A_{\mathcal{X}}), W)} \right],
\]

where \( \ell \) denotes the objective function that we defined in (15) and \( \text{vec} \) denotes the vectorization operator in Algorithm A4. By Markov’s inequality, we have for each \( \delta > 0 \) that

\[
P(\ell(\text{vec}(A_{\mathcal{X}}), W) \geq \delta) \leq \frac{E[\ell(\text{vec}(A_{\mathcal{X}}), W)]}{\delta}.\]

We define the notation \( M := \sup_{x:F \rightarrow G} \sqrt{\delta} P_{X \sim \pi}(\ell(\text{vec}(A_{\mathcal{X}}), W) < \delta) + \frac{E[\ell(\text{vec}(A_{\mathcal{X}}), W)]}{\delta} M \)

\[
\leq \sqrt{\delta} + \frac{E[\ell(\text{vec}(A_{\mathcal{X}}), W)]}{\delta} M.
\]

The last expression in (32) is minimized when \( \delta = (2M E[\ell(\text{vec}(A_{\mathcal{X}}), W)]^{2/3} \). This yields

\[
E \left[ \sqrt{\ell(\text{vec}(A_{\mathcal{X}}), W)} \right] \leq \left( 2^{1/3} + 2^{-2/3} \right) (E [\ell(\text{vec}(A_{\mathcal{X}}), W)] M)^{1/3}.
\]

Noting that \( 2^{1/3} + 2^{-2/3} < 2 \) and combining (31) and (32) with (iii) then verifies (iv).

**Remark G.8.** Suppose that \( \mathcal{G}' = \mathcal{G} + \text{denoising} = \mathcal{F} \) in Theorem G.7 (iv). The left-hand side of (28) is a measure of the difference between the original network \( \mathcal{G} = (V, A) \) and the limiting reconstructed network \( \mathcal{G}_\infty = (V, \hat{A}_\infty) \) that we compute using the NDR algorithm (see Algorithm 2) with network dictionary \( W \in \mathbb{R}^{Kxk} \). Recall that the columns of \( W \) encode \( r \) latent motifs \( L_1, \ldots, L_r \in \mathbb{R}^{kxk} \) (see Section B.4). According to (28), \( \mathcal{G} = \mathcal{G}_\infty \) if the right-hand side of (28) is 0. This is the case if \( \sup_{x:F \rightarrow \mathcal{G}} \ell(\text{vec}(A_{\mathcal{X}}), W) = 0 \), which means that \( W \) can perfectly approximate all mesoscale patches \( A_{\mathcal{X}} \) of \( \mathcal{G} \). However, the right-hand side of (28) can still be small if the worst-case approximation error \( \sup_{x:F \rightarrow \mathcal{G}} \ell(\text{vec}(A_{\mathcal{X}}), W) \) is large but the expected approximation error \( E_{x \sim \pi}[\ell(\text{vec}(A_{\mathcal{X}}), W)] \) is small (i.e., when \( W \) at effective in approximating most of the mesoscale patches).

How can we find a network dictionary \( W \) that minimizes the right-hand side of (28)? Although it is difficult to find a globally optimal network dictionary \( W \) that minimizes the non-convex objective function in the right-hand side of (28), Theorems G.2 and G.5 guarantee that our NDL algorithm (see Algorithm 1) always finds a locally optimal network dictionary. Indeed, from these theorems, the NDL algorithm with
N = 1 computes a network dictionary W that is approximately a local optimum of the following expected loss function:
\[ f(W) = \mathbb{E}_{x \sim \pi} [\ell(\text{vec}(A_x), W)] , \]
where \( \pi = \hat{\pi}_{F \rightarrow G} \) if MCMC = PivotApprox and \( \pi = \pi_{F \rightarrow G} \) otherwise. The function \( f \) in (33) appears in the upper bound in (29). In our experiments, we found that our NDL algorithm produces network dictionaries that are efficient in minimizing the reconstruction error. See the left-hand side of (28) and, e.g., Figure 3.

Another implication of Theorem G.7(iii) is also relevant to network denoising (see Figure 4). Suppose that we have an uncorrupted network \( G' = (V, B) \) and a corrupted network \( G = (V, A) \). Additionally, suppose that we have trained the network dictionary \( W \) for the uncorrupted network \( G' \), but that we use it to reconstruct the corrupted network \( G \). If \( A_{x,W} \) is a nonnegative linear approximation of the \( k \times k \) matrix \( A_x \) of a mesoscale patch of the corrupted network \( G \), it may be close to the corresponding mesoscale patch \( B_x \) of the uncorrupted network \( G' \), because we used the network dictionary \( W \) that we learned from the uncorrupted network \( G' \). Theorem G.7(iii) guarantees that the network \( \hat{G}_\infty \) that we reconstruct for the corrupted network \( G \) using the uncorrupted-network dictionary \( W \) is close to the uncorrupted network \( G' \).

**Remark G.9.** The update step (see line 17) for global reconstruction in Algorithm 2 indicates that we loop over all node pairs \( (a, b) \) in the \( k \)-chain motif and that we update the weight of edge \( (\mathbf{x}_t(a), \mathbf{x}_t(b)) \) in the reconstructed network using the homomorphism \( \mathbf{x} : F \rightarrow G \). There may be multiple node pairs \( (a, b) \) in \( F \) that contribute to the edge \( (p, q) \) in the reconstructed network, because \( \mathbf{x}_t(a) = p \) and \( \mathbf{x}_t(b) = q \) can occur for multiple choices of \( (a, b) \). The output of this update step does not depend on the ordering of \( a, b \in \{1, \ldots, k\} \), as one can see from the expressions in (30).

One can also consider the following alternative update step for global reconstruction. Specifically, we first choose two nodes \( p, q \) of the reconstructed network in the image \( \{\mathbf{x}_t(j) \mid j \in \{1, \ldots, k\}\} \) \( \ell \) changed to \( j \) of the homomorphism \( \mathbf{x}_t \) and average over all pairs \( (a, b) \in [k]^2 \) such that \( (p, q) \) is visited by \( (a, b) \) through \( \mathbf{x}_t \), and we then update the weight of \( (p, q) \) in the reconstructed network with this mean contribution from \( \mathbf{x}_t \).

Specifically, for each \( a, b \in \{1, \ldots, k\} \), let \( \mathbb{I}(p, q) \xleftarrow{\mathbf{x}_t} (a, b) \) denote the indicator that we defined in (24) and let \( N_{pq}(\mathbf{x}_t) \geq 0 \) be the number of visits of \( \mathbf{x}_t \) to \( (p, q) \) (see (25)). We can then replace line 17 in Algorithm 2 with the following line:

**Alternative update for global reconstruction:**

For \( p, q \in V \) such that \( N_{pq}(\mathbf{x}_t) > 0 \):
\[
\hat{A}_{x,W}(p, q) \leftarrow \frac{1}{\sum_{1 \leq a, b \leq k} \hat{A}_{x,W}(a, b)} \sum_{1 \leq a, b \leq k} \mathbb{I}(p, q) \xleftarrow{\hat{\mathbf{x}}_t}(a, b), \quad j \leftarrow A_{\text{count}}(p, q) + 1 \]

\[
A_{\text{recon}}(p, q) \leftarrow (1 - j^{-1})A_{\text{recon}}(p, q) + j^{-1}\hat{A}_{x,W}(p, q) .
\]

For the alternative NDR algorithm that we just described, we can establish a convergence result that is similar to Theorem G.5 using a similar argument as the one in our proof of Theorem G.7. Specifically, (i) holds for the alternative NDR algorithm, so there exists a limiting reconstructed network. In the proof of (ii), the formula for the limiting reconstructed network is now
\[
\hat{A}_\infty(p, q) = \sum_{y \in \Omega_{pq}} \hat{A}_{y,W}(p, q) \mathbb{P}_{x_\sim \pi}(x = y \mid x \in \Omega_{pq}) \quad \text{for all} \quad p, q \in V ,
\]
where \( \Omega_{pq} \) is the set of all homomorphisms that visit \( (p, q) \) (see (26)). In particular, if \( G \) is an undirected and binary graph, then
\[
\hat{A}_\infty(p, q) = \frac{1}{|\Omega_{pq}|} \sum_{y \in \Omega_{pq}} \hat{A}_{y,W}(p, q) \quad \text{for all} \quad p, q \in V .
\]
In the proofs of (iii) and (iv), the same error bounds hold with \( \mathbb{E}_{x_\sim \pi}[N_{pq}(x)] \) replaced by \( \mathbb{P}_{x_\sim \pi}(x \in \Omega_{pq}) \). We omit the details of the proofs of the above statements for this alternative NDR algorithm.

We now discuss convergence results of Algorithm 2 for a bipartite network \( G \). Recall the notation and discussions about bipartite networks above Theorem G.5. Additionally, recall for our bipartite networks that there exist disjoint subsets \( \Omega_1 \) and \( \Omega_2 \) of the set \( \Omega \) of all homomorphisms \( F \rightarrow G \) such that (1) \( \Omega = \Omega_1 \cup \Omega_2 \).
and (2) the Markov chain \((x)_{t \geq 0}\) restricted to each \(\Omega_i\) (with \(i \in \{1, 2\}\)) is irreducible but is not irreducible on the set \(\Omega\).

**Theorem G.10** (Convergence of the NDR Algorithm (see Algorithm 2) for Bipartite Networks). Let \(F = ([k], A_F)\) be the \(k\)-chain motif, and let \(G = (V, A)\) be a network that satisfies assumption \((a')\) in Theorem G.5. Let \(\hat{G}_t = (V, \hat{A}_t)\) denote the network that we reconstruct using Algorithm 2 at iteration \(t\) using a fixed network dictionary \(W \in \mathbb{R}^{k \times r}\). Fix \(i \in \{1, 2\}\) and an initial homomorphism \(x_0 \in \Omega_i\). Let \(\pi = \pi_{F \rightarrow G}\) if MCMC = PivotApprox and \(\pi = \pi_{F \rightarrow \hat{G}}\) otherwise. The following properties hold:

(i) The network \(\hat{G}_t\) converges almost surely to some limiting network \(\hat{G}_\infty = (V, \hat{A}_\infty)\) in the sense that
\[
\lim_{t \to \infty} \hat{A}(p, q) = \hat{A}_\infty(p, q) \quad \text{almost surely for all } p, q \in V.
\]

(ii)-(iv) The same statements as in Theorem G.7(ii)-(iv) hold with the expectation \(\mathbb{E}_{x \sim \pi}\) replaced by the conditional expectation \(\mathbb{E}_{x \sim \pi}[\cdot | x \in \Omega_i]\).

(v) The results in (ii)–(iv) do not depend on \(i \in \{1, 2\}\) if \(k\) is even.

**Proof.** The proofs of statements (i)–(iv) are identical to those for Theorem G.7. Statement (v) follows from a similar argument as in the proof of Theorem G.5(ii) by constructing coupled Markov chains \((x_t)_{t \geq 0}\) and \((x'_t)_{t \geq 0}\) such that \(x'_t = \Sigma_t\) for all \(t \geq 0\). □

**Remark G.11.** Our proofs of Theorems G.7 and G.10 do not depend on the particular choice of mask \(\Phi_{\pi, \infty}\) that we use to define the mesoscale patches \(A_\infty\) in (2).

**Remark G.12.** In Theorem G.10(ii), let \(\hat{G}_\infty^{i} = (V, \hat{A}_\infty^{i})\) denote the limiting reconstructed network for \(G\) conditional on the Markov chain being initialized in \(\Omega_i\) for \(i \in \{1, 2\}\). When \(k\) is even, Theorem G.10(iv) implies that \(\hat{G}_\infty^{1} = \hat{G}_\infty^{2}\). When \(k\) is odd, we run the NDR algorithm (see Algorithm 2) twice with the Markov chain initialized in both \(\Omega_1\) and \(\Omega_2\). We then define the network \(\hat{G}_\infty := (V, (\hat{A}_\infty^{1} + \hat{A}_\infty^{2})/2)\), whose weight matrix is the mean of those of the two limiting reconstructed networks \(\hat{G}_\infty^{i}\) for \(i \in \{1, 2\}\). We obtain a similar error bound as in Theorem G.10(iii) for this mean limiting reconstructed network. In practice, one can obtain a sequence of reconstructed networks that converges to the mean reconstructed network \(\hat{G}_\infty\) by reinitializing the Markov chain every \(\tau\) iterations of the reconstruction procedure for any fixed \(\tau\).

**Appendix H. Auxiliary Algorithms**

We now present auxiliary algorithms that we use to solve subproblems of Algorithms 1 and 2. Let \(\Pi_S\) denote the projection operator onto a subset \(S\) of ambient space. For each matrix \(A\), let \([A]_{i\bullet}\) (respectively, \([A]_{\bullet i}\)) denote the \(i^{th}\) column (respectively, \(i^{th}\) row) of \(A\).

**Algorithm A1. Coding**

1: **Input:** Data matrix \(X \in \mathbb{R}^{d \times b}\), dictionary matrix \(W \in \mathbb{R}^{d \times r}\)  
2: **Parameters:** \(T \in \mathbb{N}\) (the number of iterations) \(\lambda > 0\) (the coefficient of an \(L_1\)-regularizer)  
\(C_{\text{code}} \subseteq \mathbb{R}^{r \times b}\) (convex constraint set of codes)  
3: **For** \(t = 1, \ldots, T\):  
4: \hspace{1cm} **Do:**  
\hspace{2cm} \(H \leftarrow \Pi_{C_{\text{code}}} \left( H - \frac{1}{\operatorname{tr}(W^T W)} \left( W^T W H - W^T X + \lambda J \right) \right) \),  
\hspace{2cm} where \(J \subseteq \mathbb{R}^{d \times b}\) is the matrix with all 1 entries.  
5: **Output:** \(H \in C_{\text{code}} \subseteq \mathbb{R}^{r \times b}\)
Algorithm A2. Dictionary-Matrix Update

1: **Input:** Previous dictionary matrix $W_{t-1} \subseteq \mathbb{R}^{k_2 \times r}$, previous aggregate matrices $(P_t, Q_t) \in \mathbb{R}^{r \times r} \times \mathbb{R}^{r \times N}$
2: **Parameters:** $C_{\text{dict}} \subseteq \mathbb{R}^{k_2 \times r}$ (compactness and convexity constraint for dictionary matrices) $T \in \mathbb{N}$ (the number of iterations)
3: For $t = 1, \ldots, T$:
4: $W \leftarrow W_{t-1}$
5: For $j = 1, 2, \ldots, N$:
6: $W(:,j) \leftarrow \Pi_{C_{\text{dict}}} \left( W(:,j) - \frac{1}{A_t(j,j)} \left( WP_t(:,j) - Q_t^T(:,j) \right) \right)$
7: **Output:** $W_t = W \in C_{\text{dict}} \subseteq \mathbb{R}^{k_2 \times r}$

Algorithm A3. Rejection Sampling of Homomorphisms

1: **Input:** Network $\mathcal{G} = (V, A)$, motif $F = ([k], A_F)$
2: **Requirement:** There exists at least one homomorphism $F \to \mathcal{G}$
3: Repeat: Sample $x = [x(1), x(2), \ldots, x(k)] \in V^k$ so that the quantities $x(i)$ are independent and identically distributed
4: If $\prod_{i,j \in \{1,\ldots,k\}} A(x(i), x(j))^{A_F(i,j)} > 0$
5: **Return** $x : F \to \mathcal{G}$ and **Terminate**
6: **Output:** Homomorphism $x : F \to \mathcal{G}$

Algorithm A4. Vectorization

1: **Input:** Matrix $X \in \mathbb{R}^{k_1 \times k_2}$
2: **Output:** Matrix $Y \in \mathbb{R}^{k_1 k_2 \times 1}$, where $Y(k_2(j-1) + i, 1) = X(i,j)$ for all $i \in \{1, \ldots, k_1\}$ and $j \in \{1, \ldots, k_2\}$

Algorithm A5. Reshaping

1: **Input:** Matrix $X \in \mathbb{R}^{k_1 k_2 \times 1}$, a pair $(k_1, k_2)$ of integers
2: **Output:** Matrix $Y \in \mathbb{R}^{k_1 \times k_2}$, where $Y(i,j) = X(k_2(j-1) + i, 1)$ for all $i \in \{1, \ldots, k_1\}$ and $j \in \{1, \ldots, k_2\}$
APPENDIX I. ADDITIONAL FIGURES

We show additional figures of our network dictionaries.

![Network dictionaries for different scales](image)

**Figure 8.** The $r = 25$ latent motifs at scales $k = 6, 11, 21, 51, 101$ that we learn from the networks Caltech, MIT, UCLA, and Harvard. The numbers underneath the latent motifs give their dominance scores (see Section D.2). See Section F for the details of these experiments.
| scale = 6 | scale = 11 | scale = 21 | scale = 51 | scale = 101 |
|-----------|-----------|-----------|-----------|-----------|
| ![Coronavirus PPI](image1) | ![Coronavirus PPI](image2) | ![Coronavirus PPI](image3) | ![Coronavirus PPI](image4) | ![Coronavirus PPI](image5) |
| ![SNAP Facebook](image6) | ![SNAP Facebook](image7) | ![SNAP Facebook](image8) | ![SNAP Facebook](image9) | ![SNAP Facebook](image10) |
| ![arXiv ASTRO-PH](image11) | ![arXiv ASTRO-PH](image12) | ![arXiv ASTRO-PH](image13) | ![arXiv ASTRO-PH](image14) | ![arXiv ASTRO-PH](image15) |
| ![Homo sapiens PPI](image16) | ![Homo sapiens PPI](image17) | ![Homo sapiens PPI](image18) | ![Homo sapiens PPI](image19) | ![Homo sapiens PPI](image20) |

**Figure 9.** The $r = 25$ latent motifs at scales $k = 6, 11, 21, 51, 101$ that we learn from the networks *Coronavirus PPI, SNAP Facebook, arXiv ASTRO-PH, and Homo sapiens PPI*. The numbers underneath the latent motifs give their dominance scores (see Section D.2). See Section F for the details of these experiments.
Figure 10. The $r = 25$ latent motifs at scales $k = 6, 11, 21, 51, 101$ that we learn from the networks $WS_1, WS_2, BA_1,$ and $BA_2$. The numbers underneath the latent motifs give their dominance scores (see Section D.2). See Section F for details of the experiments.
The 25 latent motifs for \( r \in \{9, 16, 25, 36, 49\} \) at scale \( k = 21 \) that we learn from the networks Caltech, MIT, and UCLA. The \( r = 25 \) column is identical to the \( k = 21 \) column in Figure 8. The numbers underneath the latent motifs give their dominance scores (see Section D.2). See Section F for details of the experiments.
Figure 12. The latent motifs for $r \in \{9, 16, 25, 36, 49\}$ at scale $k = 21$ that we learn from the networks Coronavirus PPI, SNAP Facebook, arXiv ASTRO-PH, and Homo sapiens PPI. The $r = 25$ column is identical to the $k = 21$ column in Figure 9. The numbers underneath the latent motifs give their dominance scores (see Section D.2). See Section F for details of the experiments.
The latent motifs for $r \in \{9, 16, 25, 36, 49\}$ at scale $k = 21$ that we learn from the networks WS1, WS2, BA1, and BA2. The $r = 25$ column is identical to the $k = 21$ column in Figure 10. The numbers underneath the latent motifs give their dominance scores (see Section D.2). See Section F for details of our experiments.
Figure 14. (Top two rows) The $r = 25$ latent motifs at scales $k = 6, 11, 21, 51, 101$ that we learn from the networks $\text{ER}_1$ and $\text{ER}_2$. (Bottom two rows) The latent motifs for $r \in \{9, 16, 25, 36, 49\}$ at scale $k = 21$ that we learn from the networks $\text{ER}_1$ and $\text{ER}_2$. The $k = 21$ column of the first two rows are identical to the $r = 25$ column of the last two rows. The numbers underneath the latent motifs give their dominance scores (see Section D.2). See Section F for details of our experiments.