

Bounded graph clustering with graph neural networks

Kibidi Neocosmos, Diego Baptista, Nicole Ludwig

Angaben zur Veröffentlichung / Publication details:

Neocosmos, Kibidi, Diego Baptista, and Nicole Ludwig. 2026. "Bounded graph clustering with graph neural networks." *Journal of Physics: Complexity* 7(2): 025005.
<https://doi.org/10.1088/2632-072x/ae6008>.

Nutzungsbedingungen / Terms of use:

CC BY 4.0

Dieses Dokument wird unter folgenden Bedingungen zur Verfügung gestellt: / This document is made available under these conditions:
CC-BY 4.0: Creative Commons: Namensnennung
Weitere Informationen finden Sie unter: / For more information see:
<https://creativecommons.org/licenses/by/4.0/deed.de>



PAPER • OPEN ACCESS

Bounded graph clustering with graph neural networks

To cite this article: Kibidi Neocosmos *et al* 2026 *J. Phys. Complex.* **7** 025005

View the [article online](#) for updates and enhancements.

You may also like

- [Unsupervised community detection in attributed networks based on mutual information maximization](#)
Junyou Zhu, Xianghua Li, Chao Gao et al.
- [Community structure detection in complex networks with partial background information](#)
Zhong-Yuan Zhang
- [Neural-prior stochastic block model](#)
O Duranthon and L Zdeborová



PAPER

OPEN ACCESS

RECEIVED

2 December 2025

REVISED

24 March 2026

ACCEPTED FOR PUBLICATION

14 April 2026

PUBLISHED

29 April 2026

Original content from this work may be used under the terms of the [Creative Commons Attribution 4.0 licence](#).

Any further distribution of this work must maintain attribution to the author(s) and the title of the work, journal citation and DOI.



Bounded graph clustering with graph neural networks

Kibidi Neocosmos^{1,*} , Diego Baptista² and Nicole Ludwig¹ ¹ University of Tübingen, Tübingen, Germany² Graz University of Technology, Graz, Austria

* Author to whom any correspondence should be addressed.

E-mail: kibidi.neocosmos@uni-tuebingen.de**Keywords:** graph neural networks, community detection, graph clustering, unsupervised learning

Abstract

In community detection, many methods require the user to specify the number of clusters in advance since an exhaustive search over all possible values is computationally infeasible. While some classical algorithms can infer this number directly from the data, this is typically not the case for graph neural networks (GNNs): even when a desired number of clusters is specified, standard GNN-based methods often fail to return the exact number due to the way they are designed. In this work, we address this limitation by introducing a flexible and principled way to control the number of communities discovered by GNNs. Rather than assuming the true number of clusters is known, we propose a framework that allows the user to specify a plausible range and enforce these bounds during training. Additionally, if the user wants an exact number of clusters, it may also be specified and reliably returned.

1. Introduction

Graph neural networks (GNNs) have become an increasingly popular choice for graph-based tasks (see [2, 11, 19, 23] for example applications). Network classification is a common task and requires a model to match a network to its label. To do so, it transforms the input network into an initial guess of the label and then improves that guess based on its similarity to the true label. The transformation requires a component in the GNN known as a pooling layer. The purpose of the pooling layer is to reduce the size of the input data so that it may ultimately become a label. We refer the reader to [7] for a more comprehensive discussion of pooling in GNNs.

Recently, community detection has been proposed as a method for pooling (i.e. to reduce the size of the network data) [1, 21]. Community detection is the process of identifying meaningful collections of nodes (i.e. communities or clusters) in a network so that they can be replaced or summarized by a singular (super)node. Connections between supernodes are then formed from the connections between the communities that they represent. To perform community detection in the pooling layer of a GNN, another GNN is used. While other classic non-neural network styled methods, in principle, may also be used, this would no longer allow end-to-end learning — a hallmark of the success of neural networks.

Community detection using GNNs was originally developed in the context of a pooling layer to reduce the size of the network. This process is a supervised learning task that requires labeled data. However, community detection is traditionally an unsupervised learning task; there are no labels for the data. When evaluated in a classical community detection setting, GNNs present a surprising flaw: they do not return the number of communities specified by the user. [1, 21] hint at this problem by their inclusion of regularization terms in the GNN's loss function to influence the number of communities returned. The first part of our contribution addresses this problem by enabling a GNN to output the specified number of communities.

The need to specify the number of communities when using a community detection algorithm is a common feature [5]. As stated previously, community detection is the process of finding meaningful collections of nodes in a network. The task requires both identifying a meaningful collection of nodes

and determining a meaningful number of collections. The second problem influences the first in that part of what makes a collection of nodes meaningful is the *number* of collections. To circumvent the difficulty of addressing both problems, many algorithms require that the number of communities be specified and, as such, they focus on finding a meaningful arrangement of nodes within those communities. However, other algorithms are able to address both problems. The second part of our contribution aims to push GNNs toward this class of algorithms by removing the need to specify the exact number of communities.

For our contribution, we propose a constraint that we incorporate into the loss function of a GNN. The constraint allows the output number of communities to either be specified exactly or within a given range. Our approach extends the capabilities of GNNs for community detection by introducing required functionality and additional flexibility.

2. Related work

Our general goal is to augment the loss function of GNNs used for community detection. To this end, we build upon notable examples of such GNNs, namely: DiffPool [24], MinCutPool [1], and DMoN [21].

An earlier work, DiffPool [24] introduced GNNs as a pooling technique within a larger GNN used for network classification. It supplemented the loss function of the GNN used for pooling with a link prediction function. MinCutPool [1] built on this idea and introduced the minimum cut problem as a differentiable loss function, learning soft cluster assignments that minimize edges between clusters while maintaining balanced partitions. This approach enables effective graph pooling by incorporating spectral clustering principles directly into the neural network architecture. More recently, DMoN [21] developed a GNN approach for community detection that directly optimizes modularity — a measure quantifying the difference between the number of edges within communities and the expected number under a random null model. In contrast to MinCutPool, DMoN was developed as a standalone community detection technique.

We use a similar approach to MinCutPool and, in particular, to DMoN with the addition of a constraint term to the loss function.

3. Notation

Let a graph G be defined by a set of nodes V and edges E , where n is the total number of nodes and m is the total number of edges. The adjacency matrix is represented as $\mathbf{A} \in \{0, 1\}^{n \times n}$ where $\mathbf{A}_{ij} = 1$ indicates an edge between nodes i and j (and 0 indicates no edge). Additionally, let $d_i := \sum_j \mathbf{A}_{ij}$ be the degree of a node i — the total number of edges starting at node i . We consider the cluster assignment matrix $\mathbf{S} \in [0, 1]^{n \times c}$ where c is the maximum number of clusters. The cluster assignment matrix indicates which cluster (indexed by the columns) each node (indexed by the rows) is assigned to. Let C_k be the set of nodes in cluster $k \in \{0, 1, \dots, c\}$ and $|C_k|$ is the number of distinct elements in C_k . Lastly, let $\mathbf{e}_k$ be the canonical vector of $\mathbb{R}^c$ for some $k \in \{1, \dots, c\}$.

4. Community detection in GNNs via modularity maximization

Similar to neural networks, GNNs are non-linear function approximators. However, unlike neural networks, they leverage the topology of the input data via message passing to generate the output. They are commonly used for tasks such as node and graph classification and, although less common, for community detection. We focus on community detection.

In the context of GNNs, community detection is formulated as an optimization problem in which the GNN initially assigns nodes to communities and then iteratively improves this assignment according to an objective (or loss) function. The loss function serves as a measure of the quality of the community assignment, encoding the definition of community for which the GNN optimizes. In general, two common quality functions are used: one based on modularity and another based on the minimum cut problem. In this paper, we focus on modularity as the quality function because it is the most

widely adopted community measure (in general) and it is used in the current state-of-the-art GNN for community detection [21].

Modularity [13] formulates the idea of a community from a statistical perspective. In general, a community is seen as a densely connected collection of nodes. The question that follows is: what do we mean by “densely connected”? Modularity takes the stance that nodes are densely connected if there are more edges than one would expect to see at random. Mathematically, it is typically represented using Q and formulated as

$$Q = \frac{1}{2m} \sum_{ij}^n [\mathbf{A}_{ij} - \mathbf{P}_{ij}] \delta_{ij}(k_i, k_j), \quad (1)$$

where $\mathbf{P}_{ij}$ is the probability that there is an edge between node i and j ; k_i indicates that node i is part of cluster k . The Kronecker delta function $\delta_{ij}(k_i, k_j)$ is 1 when nodes i and j are part of the same cluster k and 0 when they are part of different clusters. In the classic formulation of modularity, $\mathbf{P}_{ij} = \frac{d_i d_j}{2m}$ and $\delta_{ij}(k_i, k_j) = \sum_{ik}^c s_{ik} s_{jk}$ where $s_{ik} \in \mathbf{S}$ and $s_{ik} = 1$ if node i is assigned to cluster k (0 otherwise). Thus,

$$Q = \frac{1}{2m} \sum_{ij}^n \left[\mathbf{A}_{ij} - \frac{d_i d_j}{2m} \right] \sum_k^c s_{ik} s_{jk}. \quad (2)$$

Note that when all nodes are placed in a single community, the two terms in (1) cancel each other (since $\sum_{ij} \mathbf{A}_{ij} = \sum_{ij} \mathbf{P}_{ij} = 2m$), and thus modularity is zero. Additionally, $\mathbf{S} \in \{0, 1\}^{n \times c}$ has discrete values (either 0 or 1) in the classical formulation. However, a continuous formulation is used ($\mathbf{S} \in [0, 1]^{n \times c}$) in GNNs to ensure modularity is differentiable.

An alternative function that measures the quality of community structure is derived from the minimum cut problem. The minimum cut problem is the task of finding a specified number of disjoint subgraphs of a network by removing as few edges as possible. These disjoint subgraphs are seen as the communities of the network because they are densely connected collections of nodes. Here, as opposed to modularity, densely connected simply means more edges inside the community than between communities. Thus, the problem can be reformulated as maximizing the number of edges inside communities. Mathematically, we can express this function as

$$\sum_{k=1}^c \frac{\sum_{i,j \in C_k}^n \mathbf{A}_{ij}}{\sum_{i \in C_k, j \notin C_k}^n \mathbf{A}_{ij}}, \quad (3)$$

where the numerator is the number of edges inside community k and the denominator is the number of edges leaving the same community. Equation (3) is usually normalized by a constant to prevent the undesirable solution of having a community with $n - 1$ nodes and another with 1 node. When it is normalized by the number of nodes, it is called *RatioCut* [8], and when it is normalized by the edge weights, it is referred to as *Ncut* [18].

Again, we focus on modularity as a measure of community quality.

4.1. The problem of choosing the number of clusters

Many community detection algorithms require the desired number of clusters as an input. GNNs are no different. In practice, however, the specified number of clusters may be interpreted as an upper bound, rather than a guarantee that exactly that many clusters will be found. To understand why this is the case, we need to keep in mind that community detection in GNNs is formulated as an optimization problem: the neural network seeks the nodes' cluster assignments that maximize a desired community quality function (for example, modularity). To that end, the entries of the cluster assignment matrix are continuous, not discrete (for the sake of differentiability) and, thus, represent a soft assignment of the nodes. However, we still desire a hard clustering of nodes and therefore need to convert the soft assignment to a hard assignment. An intuitive and common approach to achieve this is to apply the max function to the rows of the cluster assignment matrix: we assign each node to the cluster to which it is most likely to belong. Effectively, we treat the continuous values in the cluster assignment matrix as probabilistic memberships, where each entry represents the probability that a node is assigned to a given cluster. And in so doing, we divide nodes into clusters based on the *highest* probability of being assigned to a cluster. Of course, this is not the only way one might transition from soft to hard clustering but, as previously mentioned, it intuitively makes sense, especially considering that our aim in the optimization

Algorithm 1. Constraint for finding the minimum number of clusters — $\text{constraint}(\mathbf{S}, l)$.

Input: The cluster assignment matrix $\mathbf{S}$ and the desired lower bound l

Output: Difference between l and $p \in \mathbb{R}_+$, the predicted number of clusters. Note that $p \leq l$

1 Normalize rows of $\mathbf{S}$ by the largest element of each row;

2 Find the largest element in each column of the row-normalized $\mathbf{S}$;

3 $p \leftarrow$ sum the l largest elements;

4 **return** $l - p$;

problem is to find the cluster assignment that maximizes our community quality function. However, when we assign nodes to the cluster they are most likely to belong to, the subsequent number of clusters is at most the number of communities specified (c) or fewer. This phenomenon may be exacerbated by a quality function such as modularity, which struggles to identify smaller communities and, as such, is biased towards fewer, larger clusters [4–6]. In our experiments, the GNN with only modularity as a loss function consistently finds fewer than the specified number of communities (refer to figures 2, Figures A.1–A.3)

Given that, in practice, we specify an upper bound on the output number of clusters, one might naturally and prematurely conclude that the GNN yields the optimal clustering (and thus the optimal number of clusters) below the upper bound. Ultimately, this is dependent on the quality function used. In the case of modularity, this is rarely the case because it is a non-convex function with multiple near-optimal solutions, making it difficult to optimize [6].

In light of this problem, researchers typically augment the GNN’s loss function to influence the number of clusters identified. This is done by adding a regularization term that has two effects: discouraging an undesirable, trivial solution to the quality function and encouraging more balanced clusters (we discuss the regularization terms of the DMoN and MinCutPool models in section A.8). The first effect aims to penalize an undesirable solution to the community detection problem, namely, assigning all nodes to a single cluster. It is trivially true that we could maximize the number of intra-community edges if we assign all nodes to a single cluster. This is not a problem if one uses modularity as a quality function because it is, by definition, zero when all nodes are placed in the same cluster. Additionally, the second effect encourages balanced clusters: each cluster should have the same number of nodes, with c clusters in total. Notably, the inclusion of this second effect is not justified in the literature [1, 21]. Perhaps it is to encourage c number of clusters. In our experimentation, it seems to encourage more clusters, although rarely c .

4.2. Our contribution: bounding the output number of clusters

We propose a constraint that enforces a lower bound (l) on the number of clusters returned. Inherently, c behaves like an upper bound and, thus, the addition of our constraint creates a range for the output number of clusters. When the upper and lower bounds are equal, the GNN will return an exact number of clusters.

The constraint acts on the cluster assignment matrix $\mathbf{S}$. Mathematically, it normalizes each row of $\mathbf{S}$ by the maximum of each row, resulting in $\mathbf{s}'_{ik} = \mathbf{s}_{ik} / \max_{1 \leq k \leq c} \mathbf{s}_{ik}$ where $\mathbf{s}_{ik} \in \mathbf{S}$. The k th row entry with value $\mathbf{s}'_{ik} = 1$ represents the cluster with the highest membership for node i . This would also be the cluster to which it belongs if we were to consider only hard membership. Using this row-normalized matrix, we can define our constraint for the optimization problem as

$$\begin{aligned} \text{constraint}(\mathbf{S}, l) &= l - \sum_{k=1}^l \max_{1 \leq i \leq n} \mathbf{s}'_{ik} \\ &= l - \sum_{k=1}^l \max_{1 \leq i \leq n} \left[\mathbf{s}_{ik} / \max_{1 \leq k \leq c} \mathbf{s}_{ik} \right], \end{aligned} \quad (4)$$

where the elements $\max_{1 \leq i \leq n} \mathbf{s}'_{ik}$ are sorted from largest to smallest and l is the specified lower bound. For further ease of comprehension, we provide an algorithmic description of our constraint in algorithm 1.

The constraint serves as a continuous count of the number of communities in relation to a specified lower bound. Intuitively, it may be understood as a continuous measure of how close empty clusters are to becoming non-empty. When finding the largest element in each column, a non-empty cluster is

Table 1. A summary of our contribution in relation to different GNN methods for community detection.

Method	Loss		
	Community measure	Regularization	Constraint
MinCutPool	Minimum cut	✓	×
DMoN	Modularity	✓	×
Approach taken in paper	Modularity	✓	✓

represented as 1, and an empty cluster has a value between 0 and 1, indicating that none of the nodes has its maximum membership in that cluster. The closer the value is to 1, the closer it is to becoming non-empty. In this way, the constraint identifies the empty clusters most likely to become non-empty and encourages them to have at least one member. We also offer an upgraded version of our constraint (equation (4)) that allows the user to specify the minimum number of nodes per cluster. We include it in the appendix (algorithm 2).

The constraint has computational complexity $\mathcal{O}(n(c+l))$, which is linear in the number of nodes n . Specifically, computing normalization factors requires $\mathcal{O}(nc)$ operations, and evaluating the constraint over l clusters requires $\mathcal{O}(ln)$ operations. This linear scaling is small compared to the quadratic complexity $\mathcal{O}(n^2c)$ from the modularity computation (given that c is much smaller than n), which dominates the overall computational cost.

In addition to equation (4), we introduce a regularization term to encourage balanced clusters (i.e. clusters of equal size). It is a modified version of the MinCutPool [1] regularization term, which is easier to satisfy since it no longer requires the column vectors of the clusters assignment matrix to be orthogonal. It may be described as

$$\text{balance}(\mathbf{S}) = \frac{\|\text{diag}(\mathbf{S}^T \mathbf{S}) - \frac{n}{c} \mathbf{1}_c\|_2}{\|n \mathbf{e}_k - \frac{n}{c} \mathbf{1}_c\|_2}, \quad (5)$$

where $\mathbf{1}_c \in \mathbb{R}^c$ denotes the vector of all ones, $\mathbf{e}_k \in \mathbb{R}^c$ is the k th canonical vector, and $\|\cdot\|_2$ denotes the ℓ_2 norm (Euclidean norm). The denominator normalizes the function to have a maximum value of 1. As such, it attains its maximum value of 1 when all nodes are in a single cluster, and its minimum value of 0 when each cluster has an equal number of nodes. For our experiments, it is important to include the balanced-cluster regularization with the constraint. The constraint only requires that there is one node per cluster and singleton clusters have a poor modularity. Introducing the balanced-cluster regularization term encourages more nodes per cluster.

We combine equations (1), (4) and (5) into a single loss function that may be used in a GNN:

$$L(\mathbf{S}, \mu, \lambda, l) = -\text{modularity}(\mathbf{S}) + \mu \cdot \text{constraint}(\mathbf{S}, l) + \lambda \cdot \text{balance}(\mathbf{S}), \quad (6)$$

where the constants μ and λ adjust the influence of the constraint and balance regularization terms, respectively. Note that maximizing modularity is equivalent to minimizing the negative of modularity. As such, modularity is negative in equation (6) since it is optimized in GNNs using gradient descent.

In summary, we propose a constraint (equation (4)) that enforces a lower bound l and, coupled with the upper bound c , enforces a range for the output number of clusters. The range may be tightened at the user's discretion, with the case $l = c$ yielding an exact number of clusters. We incorporate the constraint into the loss function of a GNN used for community detection, along with a regularization term (equation (5)) that encourages balanced clusters. Table 1 summarizes our contribution in the context of prominent GNN architectures upon which our work developed.

In addition to creating a fixed range, a lower bound may also be used to search for an optimal number of clusters by progressively tightening the range and selecting the output number of clusters with the highest modularity. We do not explore such a procedure in this work and leave it as an open direction of inquiry. Instead, we posit the idea to further highlight the benefit of imposing a constraint on the returned number of clusters.

Table 2. A summary of the synthetic network statistics. 10 networks are generated for each network type and, as such, the number of edges and density are approximations.

Size label	Clusters	Density label	Nodes	Edges	Density
Small	5	Low	10^2	~ 450	~ 0.04
		Medium	10^2	~ 1000	~ 0.1
Medium	5	Low	10^3	$\sim 25\,000$	~ 0.02
		Medium	10^3	$\sim 100\,000$	~ 0.1
		High	10^3	$\sim 150\,000$	~ 0.15
	10	Medium	10^3	$\sim 70\,000$	~ 0.07
	20	Medium	10^3	$\sim 50\,000$	~ 0.05
Large	5	Low	10^4	$\sim 1\,000\,000$	~ 0.01

5. Experiments

We assess our proposed solution on both synthetic and real-world datasets.

In our experiments, we compare the performance of four GNNs that differ only in their loss functions. The first model uses a loss function based solely on the community quality function modularity; we refer to it as *GNN*. It serves as our baseline and provides a reference point for comparison. The second model is the standard approach currently in use. Its loss function combines a quality function with a regularization term that encourages balanced clusters, and we refer to it as *GNN+REG*. The third model extends *GNN* by incorporating our constraint, and we refer to it as *GNN+CONSTRAINT*. It provides an additional baseline for illustrating only the effect of the constraint. Finally, our proposed method augments the loss function with both the regularization term for balanced clusters, and our constraint that enforces a minimum number of clusters. We refer to this model as *GNN+REG+CONSTRAINT*. Using models with different terms in the loss function helps delineate the effect of the balance-cluster regularization term and the constraint.

Although each model differs in the loss function, they share a common model architecture. Each model consists of a GNN that transforms the input data to a transitional embedding that is then fed to a multilayer perceptron (MLP); this outputs the cluster assignment matrix. We use GraphSage (with mean aggregation) [9] as our GNN of choice because it is a common and well-established model. Additionally, we set the parameter constants $\mu = \lambda = 1$ from equations (4) and (5). We also use a random weight initialization for both the GNN and the MLP. As is standard, we initialize each model 3 times to assess the consistency of model performance (i.e. how much of the performance is influenced by the random weight initialization).

5.1. Results on synthetic data

To generate synthetic data, we use the stochastic block model, which we parameterize by the community sizes and the edge probabilities within and between communities, denoted by p_{in} and p_{out} , respectively. We use unequal cluster sizes to provide a more realistic challenge to our models, given our inclusion of the balance-cluster regularization term. The cluster sizes are generated randomly from a uniform distribution (we provide the cluster sizes in table A.3 in the appendix). The controlled setup allows us to systematically evaluate the performance of our method under varying structural properties of the network. We vary three main properties in the synthetic networks: network size (considered as the number of nodes), the number of ground-truth communities, and the network density (defined in equation (A.2) in the appendix). For ease of discussion, we provide labels for the size and density of the network. The sizes of the network varies between *small* (10^2 nodes), *medium* (10^3 nodes) and *large* (10^4 nodes). Moreover, the density is labeled as *low*, *medium* and *high*, where $\text{low} < 0.05 \leq \text{medium} \leq 0.1 < \text{high}$. Again, these labels are used solely as a means to facilitate communication. We generated 10 networks for each network type (for example, there are 10 *medium*-sized networks with 5 clusters and *low* density). We summarize the combinations of parameters used in table 2. Additionally, GNNs require node features. Therefore, we use the column vector from the adjacency matrix that is associated with a node as its feature vector. This was done to simply provide topological information to the model and not bias it based on node feature similarity.

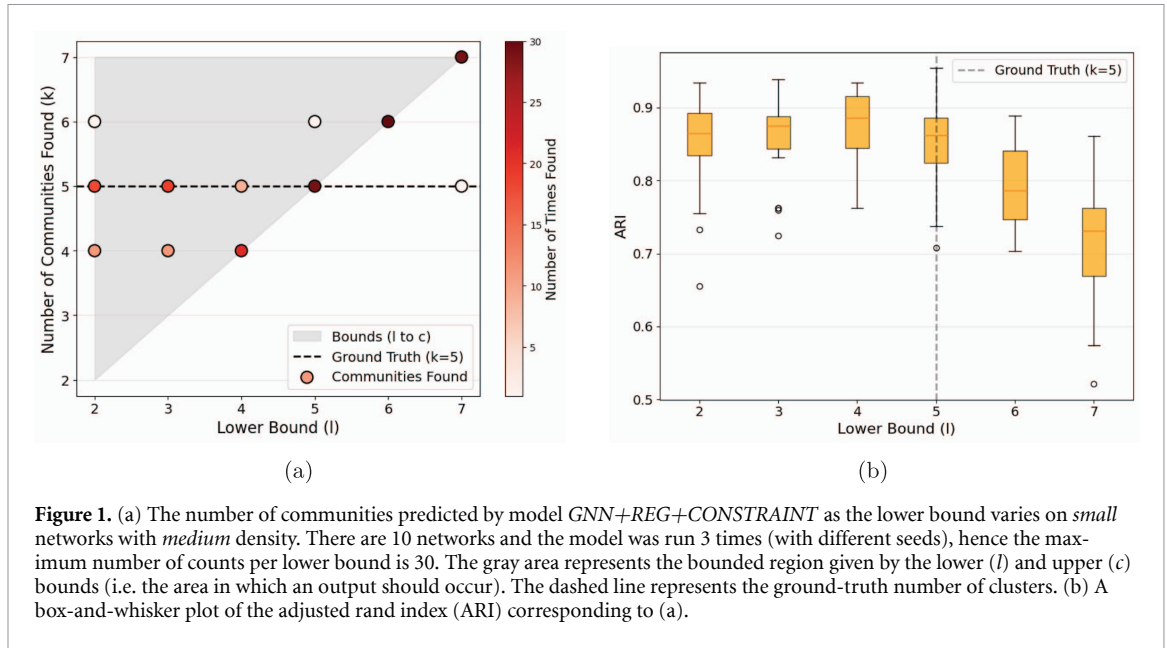


Figure 1. (a) The number of communities predicted by model *GNN+REG+CONSTRAINT* as the lower bound varies on *small* networks with *medium* density. There are 10 networks and the model was run 3 times (with different seeds), hence the maximum number of counts per lower bound is 30. The gray area represents the bounded region given by the lower (*l*) and upper (*c*) bounds (i.e. the area in which an output should occur). The dashed line represents the ground-truth number of clusters. (b) A box-and-whisker plot of the adjusted rand index (ARI) corresponding to (a).

Our proposed constraint is successfully enforced in our synthetic experiments.

In our first set of experiments, we asked the following questions: does the constraint work and, if so, how well? To address these questions, we applied our model (*GNN+REG+CONSTRAINT*) to 10 *small*-sized, *medium* density networks and executed it 3 separate times, yielding 30 experiments per lower bound. From figure 1(a), we see that in all but one experiment, the lower bound was successfully enforced. The scenario in which the constraint was not satisfied occurred when the lower bound ($l=7$) exceeded the ground-truth number of clusters ($k=5$). To fix the issue, we increased the constant μ (setting it to $\mu=1000$) from equation (4) and thus increased the effect of the constraint. Additionally, we note from figure 1(a) that the output number of communities is more diverse when the lower bound is below the ground truth. However, when the lower bound equals or exceeds the ground truth, the returned number of communities is almost exclusively on the lower bound. This is expected if the true number of communities ($k=5$) yields the highest modularity score.

The next question is: how well does the constraint work? In other words, does it produce meaningful communities? We plot the adjusted rand index (ARI) (described in equation (A.1)) for the varying-lower-bound experiments in figure 1(b) to assess the quality of the clustering. The plot shows high

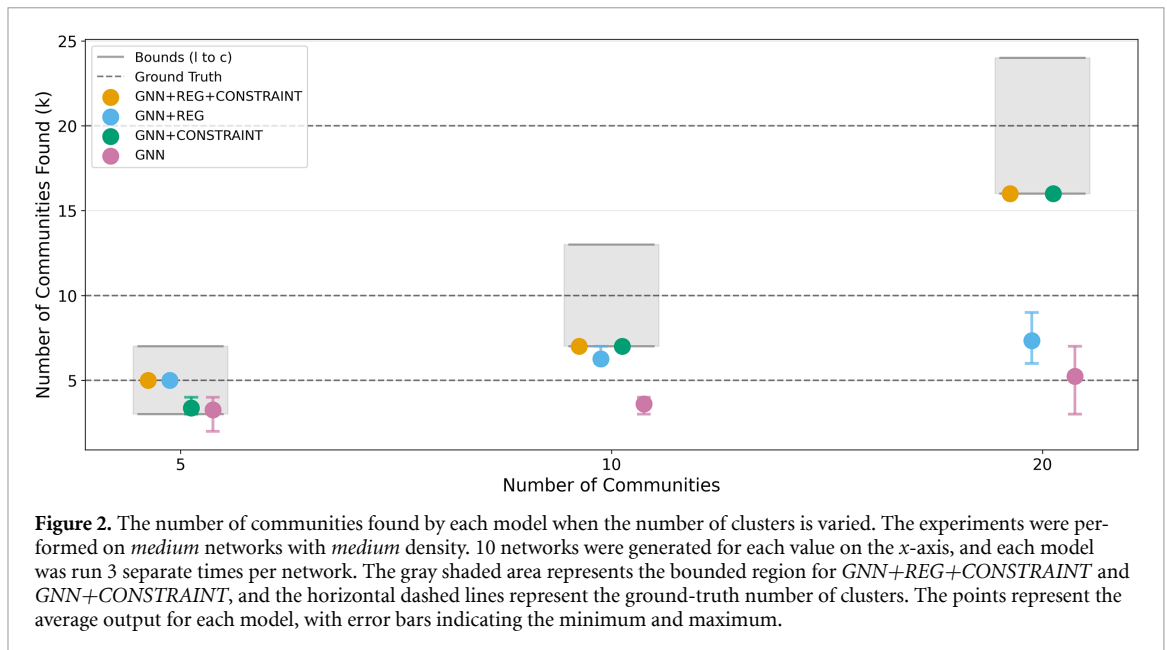
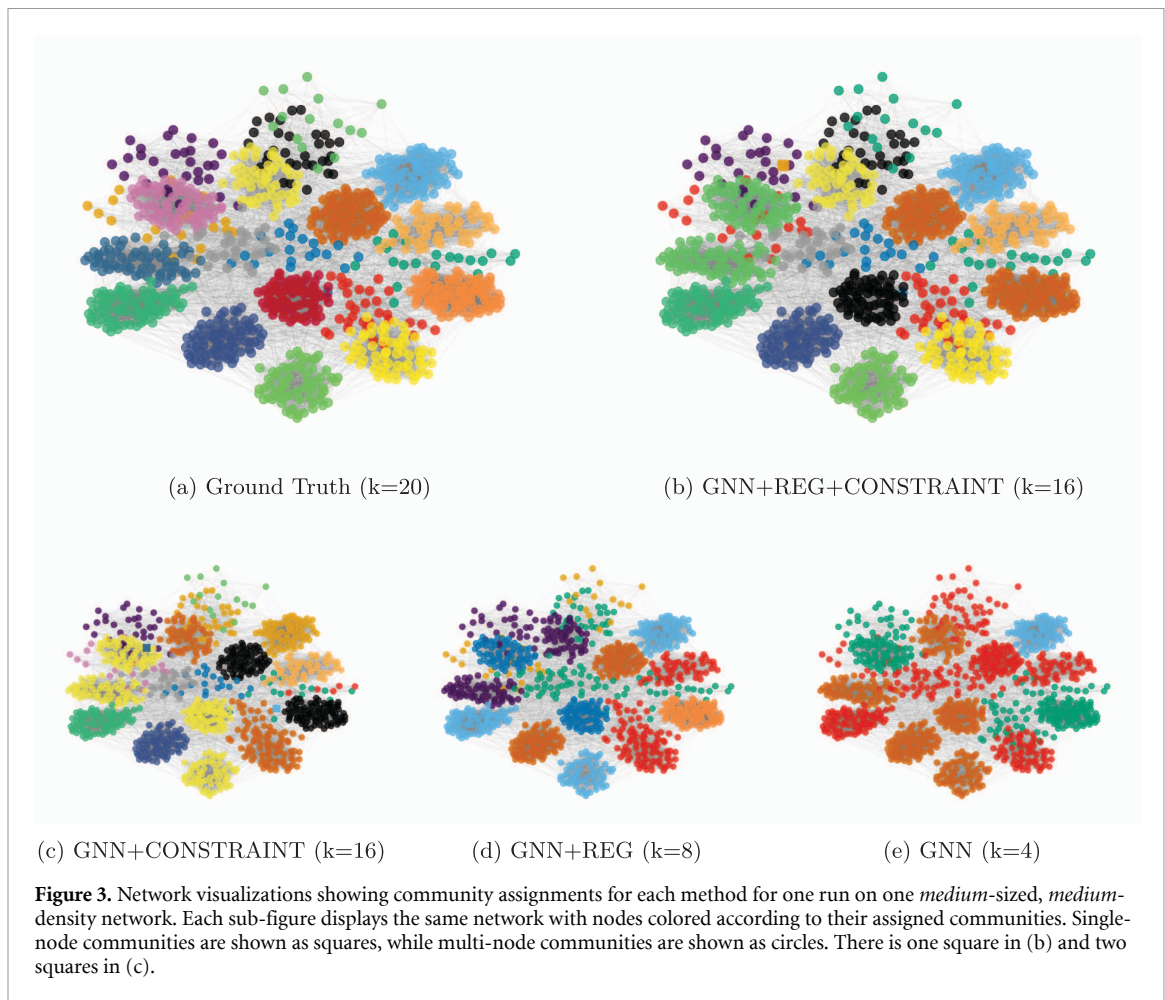


Figure 2. The number of communities found by each model when the number of clusters is varied. The experiments were performed on *medium* networks with *medium* density. 10 networks were generated for each value on the x-axis, and each model was run 3 separate times per network. The gray shaded area represents the bounded region for *GNN+REG+CONSTRAINT* and *GNN+CONSTRAINT*, and the horizontal dashed lines represent the ground-truth number of clusters. The points represent the average output for each model, with error bars indicating the minimum and maximum.



ARI scores, indicating that the model found meaningful communities within the bounds of the constraint. The ARI also has a steady trend with a slight peak at 4 and 5 communities. As the lower bound increases past 5 communities (the true number of communities), the ARI decreases monotonically. This behavior is expected because, as the true number of communities is exceeded, there are fewer nodes with the correct neighbors and, as such, there will be a lower ARI score.

For the next batch of experiments, we tested whether the constraint still held if the network properties were varied. We varied the network size, density, and number of true communities. In all cases, the constraint was satisfied. We examine the results of varying the number of true communities in figure 2. We used 10 *medium* networks with *medium* density for each number of true communities. For example, there are 10 networks with *medium* density that have 5 true communities. As the number of communities increases, the number of output communities remains on the lower bound for *GNN+REG+CONSTRAINT* and *GNN+CONSTRAINT*. On the other hand, the models without the constraint diverge further from the ground truth. This is especially evident when comparing 10 and 20 true communities. The models *GNN* and *GNN+REG* output a similar number of communities in both cases, even though there are twice the number of communities in the latter case. This is somewhat expected, given that the communities are unequal in size and randomly generated (we provide the community sizes in table A.3). As the number of communities increases, there is a greater chance for smaller communities that are difficult to detect with modularity. Modularity is known to have a resolution limit [4–6] that ultimately biases modularity maximization approaches toward larger communities. This further illustrates the value of a lower bound in that it forced the models *GNN+REG+CONSTRAINT* and *GNN+CONSTRAINT* to be closer to the desired number of communities (i.e. the ground truth) and, as a result, counteracted the bias of modularity. This is an indirect consequence of the constraint that is dependent on the lower bound chosen by the user. Other methods address the resolution limit more explicitly, like [16, 22], which introduce a hyperparameter that can be adjusted to identify smaller communities. We discuss the constraint and its relation to modularity's resolution limit further in section A.7.

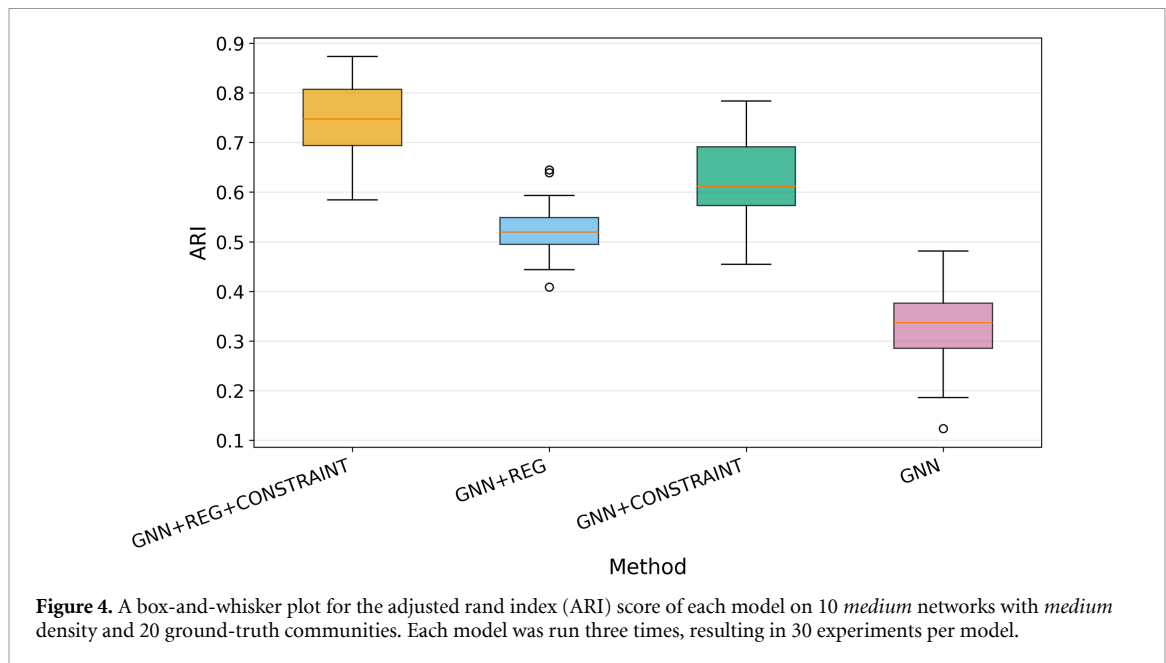


Table 3. Runtime (in seconds) of each model on synthetic datasets. Each entry reports the mean $\pm$ standard deviation over three runs.

Size label	Clusters	Density label	GNN+REG+CONSTRAINT	GNN+CONSTRAINT	GNN+REG	GNN
Small	5	Low	7.36 ± 0.05	7.06 ± 0.09	7.09 ± 0.16	6.69 ± 0.08
		Medium	7.51 ± 0.01	7.16 ± 0.02	7.13 ± 0.03	6.89 ± 0.15
Medium	5	Low	96.16 ± 6.03	70.93 ± 14.80	76.07 ± 2.93	76.21 ± 11.57
		Medium	316.26 ± 10.58	318.85 ± 8.53	210.70 ± 94.60	238.49 ± 57.46
		High	329.22 ± 141.87	526.20 ± 0.69	377.39 ± 118.24	342.75 ± 148.17
	10	Medium	209.06 ± 45.72	149.10 ± 62.83	214.92 ± 27.75	192.93 ± 39.13
	20	Medium	125.86 ± 35.65	143.05 ± 25.50	109.57 ± 34.71	122.12 ± 21.76
Large	5	Low	$33\,935.11 \pm 6\,138.76$	$34\,362.55 \pm 9\,141.21$	$32\,890.65 \pm 11\,032.64$	$22\,623.14 \pm 7\,883.28$

To elucidate the scenario of 20 true communities in figure 2, we visualize the output of each model in figure 3 and plot the ARI in figure 4. In figure 3, we plot the predicted labels of each model for a single run on a single *medium* network with *medium* density. *GNN+REG+CONSTRAINT* found 9 communities that perfectly match the ground truth; *GNN+CONSTRAINT* found 7 and *GNN+REG* and *GNN* found 1. We see this reflected in figure 4 as *GNN+REG+CONSTRAINT* and *GNN+CONSTRAINT* have higher ARI scores than *GNN+REG* and *GNN*. This implies that the constraint allowed the models to find more meaningful communities. Furthermore, our proposed approach, *GNN+REG+CONSTRAINT*, has the highest ARI. This is also expected since the addition of the regularization term encourages more nodes per community, which would lead to a better ARI. In contrast, the constraint in *GNN+CONSTRAINT* only requires one node per community.

We show the results obtained by varying the network size and density in the [appendix](#). Furthermore, we assessed the special case where the upper bound equals the lower bound and present the results in the appendix (figure A.3). Again, in all cases, the constraint was successfully enforced.

In table 3, we note the runtime of each model in seconds. We observe that all models have comparable runtime and, as a result, the addition of the constraint and regularization term have little effect.

Lastly, the four models, each using a different loss function, help assess the influence of the balance and constraint terms separately. We observe that the constraint is enforced when it is the only addition to the *GNN*, confirming that the constraint's performance is independent of the balance-cluster regularization term. However, the constraint requires only that each community contains at least one node, which is not optimal for modularity. Based on our experiments, the modularity term is not enough to encourage more nodes per community, as may be seen in figure 2 when comparing the *GNN+CONSTRAINT* and *GNN+REG+CONSTRAINT* for 5 ground-truth communities. The *GNN+CONSTRAINT* results remain on the lower bound while the *GNN+REG+CONSTRAINT* results

Table 4. A summary of the real-network statistics.

Dataset	Nodes	Edges
PubMed	19 717	44 338
Actor	7 600	30 019
Citeseer	3 312	4 715
Cora	2 708	5 429

consistently find the ground truth. Moreover, the addition of only the balance term to *GNN* also seems to help the model find more communities, illustrated again in figure 2 when comparing the models *GNN* and *GNN+REG*. *GNN+REG* is consistently closer than *GNN* to the true number of communities. Hence, this implies that the balance term encourages more nodes per community and is effective even in cases of unbalanced communities, ultimately leading to a higher-quality clustering.

It is important to note that ARI can only be assessed when the true communities are known (typically in synthetic data). In contrast, we rarely know the true communities in real data and, as such, the use of metrics (such as ARI) is no longer helpful or applicable.

5.2. Results on real data

We use a collection of widely studied real-world graph datasets, summarized in table 4. Cora, Citeseer, and PubMed are citation networks in which nodes correspond to scientific publications, edges represent citation links, node features are bag-of-words representations of document content, and communities are given by the topic labels of the papers (e.g. seven classes in Cora, six in Citeseer, and three in PubMed) [17]. The Actor dataset is a co-occurrence network in which nodes are actors and edges indicate co-occurrence in films; node features are derived from textual attributes, and communities correspond to actor or role categories [15]. In all cases, the real data come with labels. These are often used for supervised learning tasks but may not necessarily be linked to the community structure found in the network, as in real networks where the ground truth is unknown. For our purposes, the labels serve as an arbitrary reference point for the bounds, instead of a true or accurate number of communities that we could use to assess our constraint.

Similar to the synthetic networks, the constraint is enforced in all cases. Figure 5(a) depicts the predicted number of communities for each dataset using the model *GNN+REG+CONSTRAINT*, where the gray vertical bars represent the bounded region demarcated by the upper and lower bounds. All points lie within the bounds. Figure 5(b) depicts the associated mean modularity output by the model for each dataset. The model yields high modularity for the Citeseer, Cora, and PubMed datasets, and low modularity for the Actor dataset. Notably, even though the model output a low modularity, it is not a result of the constraint. Table A.1 depicts the modularity for all models (with and without the constraint), each of which converged to a low modularity. This may be a result of poor performance of the GNNs, or the dataset has a poor community structure. This highlights the difficulty of evaluating models on real data where the ground truth is unknown.

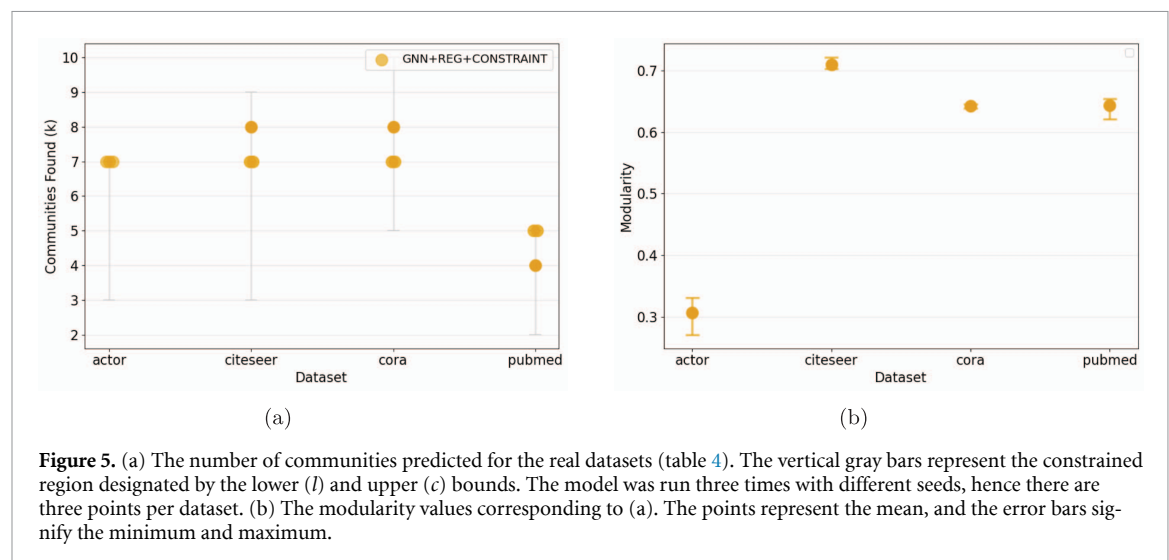


Figure 5. (a) The number of communities predicted for the real datasets (table 4). The vertical gray bars represent the constrained region designated by the lower (l) and upper (c) bounds. The model was run three times with different seeds, hence there are three points per dataset. (b) The modularity values corresponding to (a). The points represent the mean, and the error bars signify the minimum and maximum.

In addition, figure 6 illustrates the result of varying the lower bound on the Cora dataset. Again, the constraint held in each case (figure 6(a)). Interestingly, the modularity improved (figure 6(b)) as the bounds tightened. In table 5, we show the runtime for each model in seconds. We note that, as with the synthetic data, the addition of the constraint and balance regularization term has no meaningful effect on the runtime.

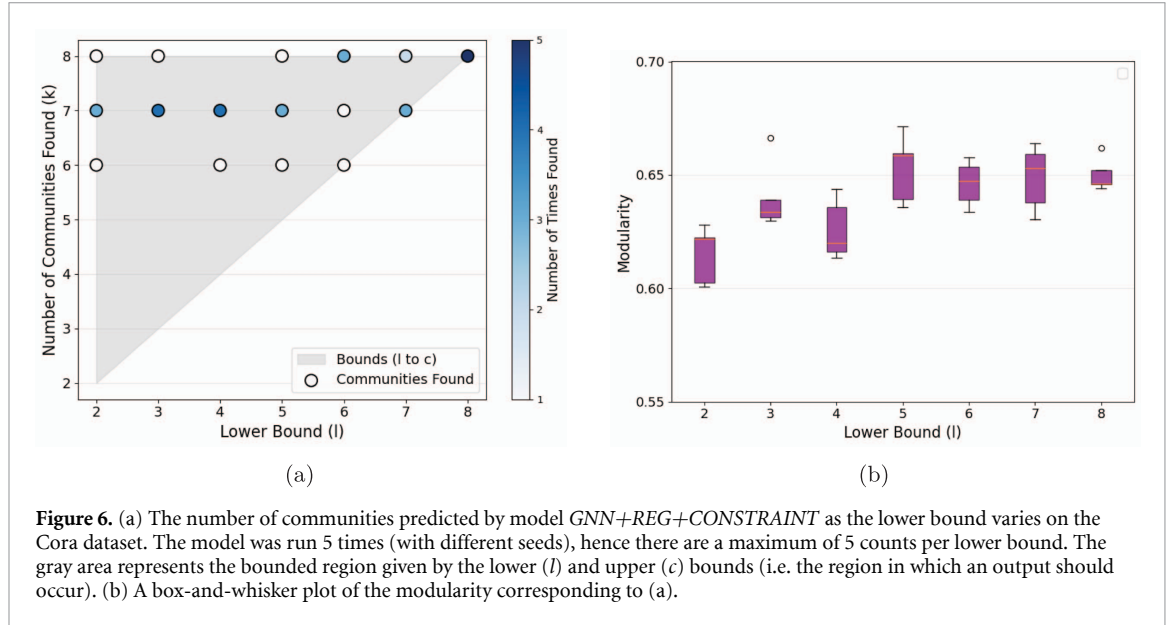


Figure 6. (a) The number of communities predicted by model $GNN+REG+CONSTRAINT$ as the lower bound varies on the Cora dataset. The model was run 5 times (with different seeds), hence there are a maximum of 5 counts per lower bound. The gray area represents the bounded region given by the lower (l) and upper (c) bounds (i.e. the region in which an output should occur). (b) A box-and-whisker plot of the modularity corresponding to (a).

Table 5. The runtime (in seconds) for real data. Each entry reports the mean $\pm$ standard deviation over three runs.

	Cora	Citeseer	PubMed	Actor
$GNN+REG+CONSTRAINT$	198 ± 24	464 ± 38	$5\,248 \pm 922$	$1\,116 \pm 180$
$GNN+CONSTRAINT$	215 ± 19	530 ± 45	$6\,403 \pm 294$	$1\,325 \pm 71$
$GNN+REG$	223 ± 23	555 ± 37	$6\,123 \pm 602$	$1\,290 \pm 109$
GNN	218 ± 21	492 ± 83	$5\,304 \pm 1659$	$1\,201 \pm 216$

Finally, from a more philosophical perspective, the constraint interprets the cluster assignment matrix as an accurate representation of the model's certainty of the cluster assignment. In other words, the constraint does not consider the quality of the assignment with respect to the community quality function. In practice, the cluster assignment matrix is not an accurate representation of the model's certainty about the cluster assignments (consider that the cluster assignment matrix is initialized with random weights). So, initially, the model guesses the cluster assignment. Instead, the cluster assignment matrix can be viewed as the set of values that maximizes the loss function. Figure 7 illustrates an example of the evolution of the cluster assignment during training. In practice, this difference in perspective appears to have no effect on the constraint's efficacy.

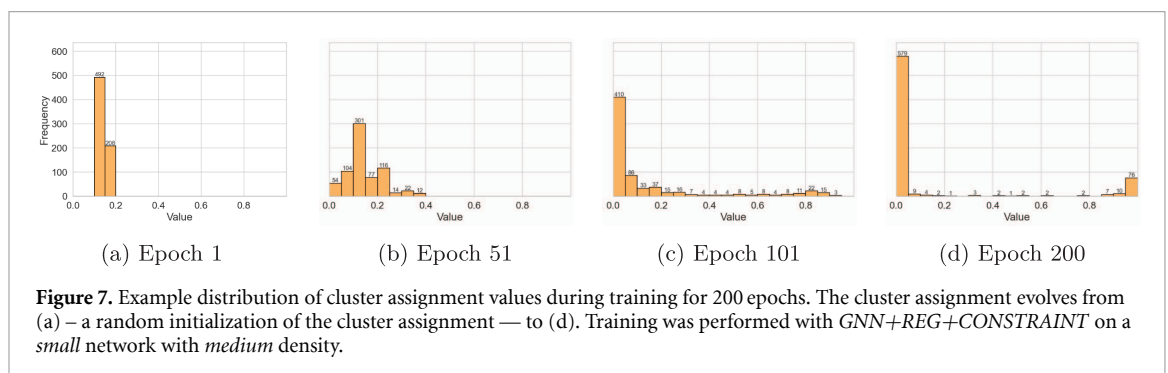


Figure 7. Example distribution of cluster assignment values during training for 200 epochs. The cluster assignment evolves from (a) – a random initialization of the cluster assignment – to (d). Training was performed with $GNN+REG+CONSTRAINT$ on a small network with medium density.

6. Conclusion, limitations, and future work

We proposed a constraint that allows a user to specify a desired number of output communities, or a range, when using GNNs for community detection. Such functionality did not exist previously, as GNNs would not return the specified number of communities. We empirically tested our constraint on a variety of real and synthetic networks, and it was successfully enforced. We recommend using the constraint jointly with a regularization term that encourages balanced communities, or another term that would encourage multiple nodes per community. This is because the constraint is satisfied if there is one node per community.

A limitation of our work was that all synthetic networks had a strong community structure. A promising direction would be to systematically identify the phase transition at which performance begins to deteriorate as community structure weakens, and to investigate whether this transition can be related to structural properties of the underlying network. Finally, the introduction of a constraint naturally opens the possibility to search for an optimal number of communities. The addition of such functionality to GNNs would be a beneficial and welcome avenue for future work.

Data availability statement

The data that support the findings of this study are openly available at the following URL/DOI: <https://doi.org/10.5281/zenodo.17793>[12]; <https://pytorch-geometric.readthedocs.io/en/2.7.0/modules/datasets.html>[20].

Funding

This work was supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy–EXC number 2064/1–Project number 39 072 7645. D.B. acknowledges support from ERC Starting Grant No. 10 116 5497.

Appendix

A.1. Additional results on synthetic data

This section presents additional experimental results on synthetic networks that further validate the effectiveness of our constraint. We examine how the model performs under varying network densities and sizes, and explore the behavior when the upper and lower bounds are equal, effectively constraining the model to a fixed number of communities.

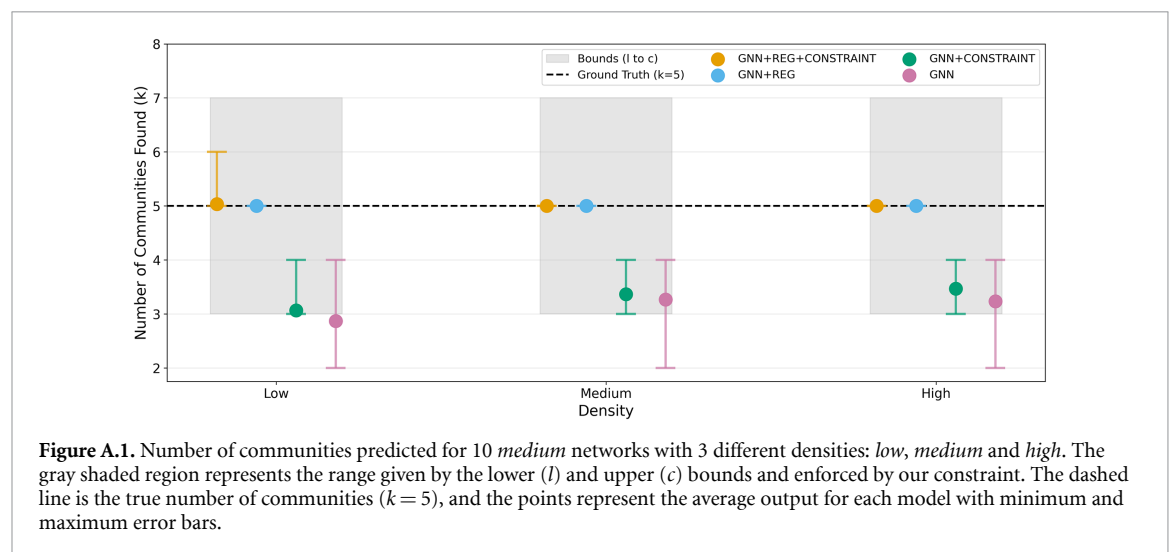
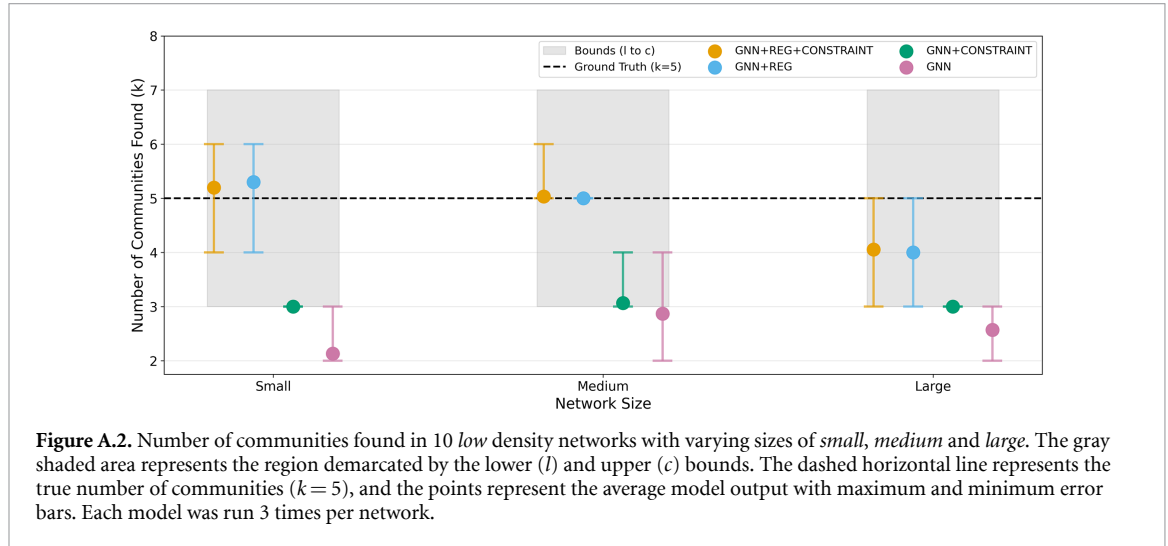


Figure A.1 demonstrates the robustness of our method across different network densities. The model maintains consistent performance in predicting the number of communities while respecting the specified bounds, regardless of whether the network has *low*, *medium*, or *high* edge density. The balance-cluster regularization term encouraged more clusters, resulting in *GNN+REG* finding the true number of communities. Thus, the constraint guaranteed that the returned number of clusters was within the bounds, and the balance regularization encouraged more clusters. This supports the use of the constraint and regularization terms in tandem.



We varied the network size and found similar results illustrated in figure A.2. However, the results for the *small* and *large* networks are more varied for *GNN+REG* and *GNN+REG+CONSTRAINT* than the *medium* network. Moreover, the results from the *large* network are further from the ground truth. Both the *small* and *large* networks have weaker community structure (illustrated by lower modularity values presented in table A.2) that could account for the larger spread of results. A weaker community structure would imply that it is more difficult to delineate communities, resulting in a wider range of results. In the case of the *large* networks, the difference in size between communities is greater (table A.3) and, as such, it is more challenging to identify the smaller communities (refer to [6] for an explanation of the resolution limits of modularity maximization). Hence, the output is (on average) lower than the ground truth.

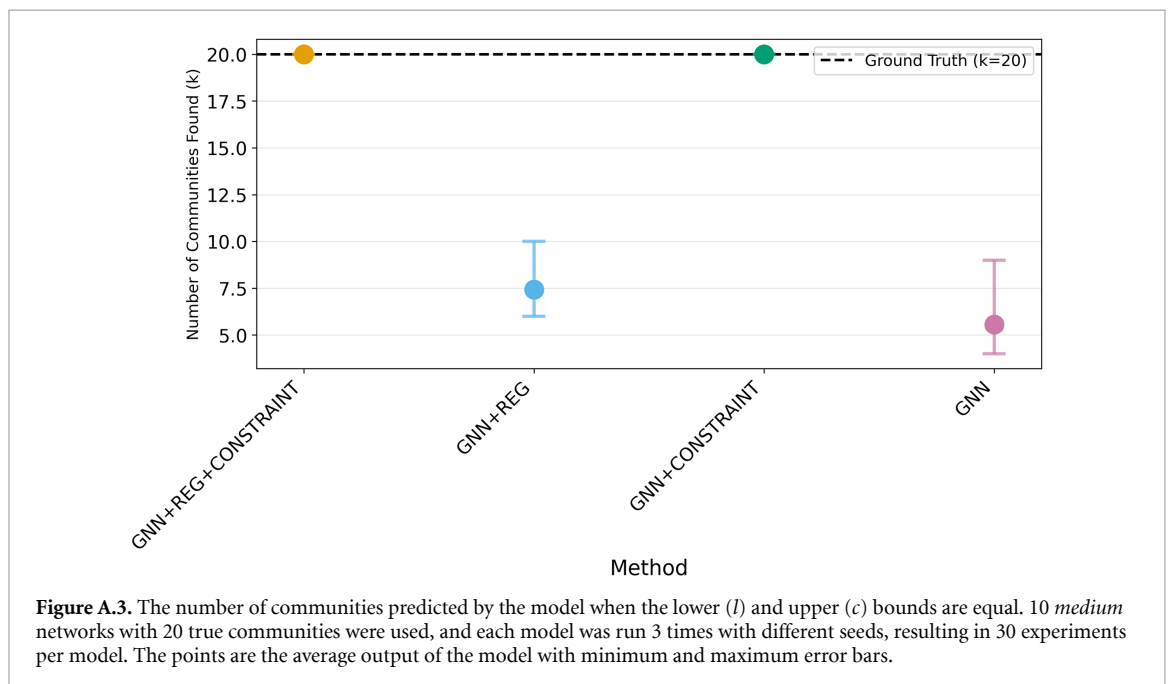


Table A.1. The modularity for each real dataset. Each entry reports the mean $\pm$ standard deviation over three runs. Note that a standard deviation of zero indicates a value smaller than 3 decimal places.

	Cora	Citeseer	PubMed	Actor
GNN+REG+CONSTRAINT	0.642 ± 0.003	0.710 ± 0.008	0.643 ± 0.015	0.306 ± 0.026
GNN+CONSTRAINT	0.517 ± 0.017	0.580 ± 0.022	0.429 ± 0.202	0.294 ± 0.076
GNN+REG	0.629 ± 0.007	0.703 ± 0.030	0.624 ± 0.000	0.274 ± 0.016
GNN	0.474 ± 0.019	0.552 ± 0.010	0.200 ± 0.000	0.296 ± 0.069

Algorithm 2. Constraint for finding the minimum number of clusters l with a minimum number of elements b per cluster — $f(\mathbf{S}, l, b)$.

Input: The cluster assignment matrix $\mathbf{S}$, the desired lower bound l and the minimum number of elements per cluster b

Output: Difference between $l \cdot b$ and $p \in \mathbb{R}_+$, the predicted number of clusters. Note that $p \leq l \cdot b$

- 1 Normalize rows of $\mathbf{S}$ by the largest element of each row;
 - 2 Find the b largest elements in each column of the row-normalized $\mathbf{S}$;
 - 3 Find the l largest row elements among the b largest column elements;
 - 4 $p \leftarrow$ sum collection of elements;
 - 5 **return** $l \cdot b - p$;
-

When the upper and lower bounds are equal, the constraint forces the model to predict an exact number of communities. Figure A.3 shows that the models with the constraint ($GNN+CONSTRAINT$ and $GNN+REG+CONSTRAINT$) successfully adhere to the tight bound. Without the constraint, the models' output is significantly further from the true number of communities.

A.2. Additional results on real data

We provide extra results from our experiments on real data.

A.3. Differentiability of constraint

Our approach relies on minimizing the constraint (equation (4)) and, as such, differentiability is a key consideration. The derivatives of the constraint with respect to the values of the cluster assignment matrix are of concern. In other words, how does the value of the constraint change as the cluster assignment changes? In this regard, the constraint depends upon three functions that are worth further explanation: max, sort, and index slicing (i.e. retrieving the top l elements). Notably, we use the Python package *PyTorch* version 2.2.2 [14] for our calculations and, subsequently, the behavior of each derivative should be understood in that context. max is a constant function that outputs the largest number in a set. Its derivative is zero except at the largest number, where it is one. Next, sort arranges the elements in a set, in our case, from largest to smallest, and it has a derivative of 1 everywhere. Lastly, index slicing retrieves l elements and keeps the derivatives associated with the indexed values.

The derivatives are calculated using the chain rule, and thus, max, sort, and index slicing do not influence the magnitude of the derivatives (consider that max and sort have derivatives of 1, and index slicing only restricts the derivatives available). Instead, specifically max and index slicing, limit the cluster assignment values that have an influence on the constraint. The part of the constraint that influences the magnitude of the derivative is $s_{ik} / \max_{1 \leq k \leq c} s_{ik}$. Therefore, the constraint is differentiable with respect to the elements of the cluster assignment matrix, but only a few entries influence it.

Another important point to consider is when there are multiple, identical maximum entries in the max function. In such an event, the first maximum value is output. Notably, it is a rare occurrence because the cluster assignment matrix contains real numbers between 0 and 1. For two values to be identical, they would need to be the same up to 4 decimal places.

A.4. Alternative constraint

Our proposed constraint (equation (4)) encourages that there is at least one node per cluster. A natural extension of this would be the option to include b minimum number of nodes per cluster. We express this upgraded functionality in the algorithm 2.

A.5. User guide for constraint

The constraint is most effective when used in tandem with a regularization term that encourages multiple nodes per cluster, such as the balance-cluster regularization term (equation (5)). Without such regularization, the constraint will be satisfied by assigning a single node to a community.

The constraint has a range of 1 to l (the lower bound) which is generally much larger than the balance-regularization term, which is between 0 and 1. This naturally leads the constraint to have more of an influence (since we would like to enforce a lower bound). However, the hyperparameters λ and μ allow for the strength of the regularization term and constraint to be adjusted. By default, we set $\lambda = \mu = 1$, but they may be altered by the user depending on the desired outcome. For example, if an output community has a single node, the value of λ should be increased. In the case where the constraint is not satisfied, μ should be increased.

Notably, the effectiveness of the constraint fundamentally depends on the quality of the cluster assignment matrix. When the cluster assignment values are very similar (often an indication of weak partitioning between clusters), satisfying the constraint becomes challenging. In such scenarios, the model may struggle to find assignments that simultaneously respect the constraint and maintain reasonable clustering quality. In such a case, we would recommend examining the similarity between values in the same row of the cluster assignment matrix, especially the two largest values. Then, increase the constant μ (equation (6)) that influences the strength of the constraint such that it is greater than the smallest difference between the two largest values in every row. The increase of μ might need to be orders of magnitude greater than the aforementioned difference.

A.6. Hyperparameter sensitivity

Within the loss function (equation (6)), the hyperparameters μ and λ adjust the influence of the constraint (equation (4)) and regularization (equation (5)), respectively. We performed a hyperparameter sensitivity analysis to better understand the influence of μ and λ (figure A.4). Figure A.4(a) illustrates a consistent performance as μ varies, with $\mu = 1$ having the lowest variance. The constraint is zero once it is satisfied and, therefore, it is expected that the ARI should not vary much as μ increases. On the other hand, figure A.4(b) depicts the ARI as λ varies and we observe that as it increases, the overall performance decreases (note that the variance increases). This is to be expected since, as λ increases, the balance regularization term has more influence during optimization. If its influence is too large, finding communities of equal size would take precedence over modularity and, thus, ARI would decrease.

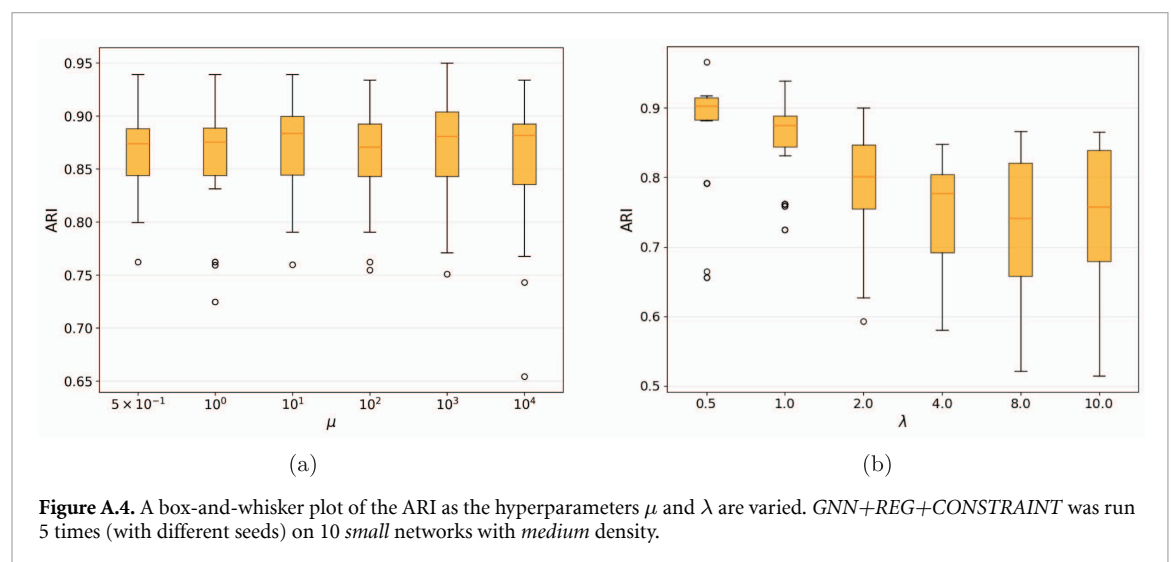


Figure A.4. A box-and-whisker plot of the ARI as the hyperparameters μ and λ are varied. *GNN+REG+CONSTRAINT* was run 5 times (with different seeds) on 10 *small* networks with *medium* density.

By default, we set $\mu = \lambda = 1$ for our experiments and reported the results of roughly 2000 synthetic and real experiments, out of which the constraint was violated only once (refer to figure 1 and a discussion in section 5.1). Furthermore, in figure 6(a), the ARI is high with the lowest variance when $\mu = 1$. Therefore, we believe that setting $\mu = 1$ is a good initial guess for the hyperparameter.

A.7. Constraint and modularity's resolution limit

A side-effect of our constraint (equation (4)), coupled with the regularization term (equation (5)), is that it can help counteract the inherent resolution limit of modularity. It is important to note that this is an indirect consequence that can only help in some scenarios. For example, imagine a scenario where there is a small community that is below the resolution limit. Identifying the community would not increase modularity. Now, imagine we know the true number of communities and, as such, set the lower and upper bounds to equal the number of true communities. The constraint would penalize the model until the specified number of communities was found. In other words, even though the small community would result in a sub-optimal modularity score, the overall loss (equation (6)) would be higher because the constraint would be satisfied.

Even though the constraint cannot serve as a general approach to counteract modularity's resolution limit, it may be paired with other approaches that explicitly do, such as γ -modularity [16]. The γ -modularity approach uses a hyperparameter γ that scales the term $\mathbf{P}_{ij}$ in equation (1) and, in so doing, adjusts the size of the community that can be detected. γ -modularity could replace the standard formulation (equation (1)) inside the loss function and, thus, serve as a general means to offset the resolution limit.

A.8. Regularization terms of MinCutPool and DMoN

As a reminder, $\mathbf{S} \in [0, 1]^{n \times c}$ is the cluster assignment matrix where n is the number of nodes and c is the number of clusters; $\mathbf{e}_k$ is the canonical vector of $\mathbb{R}^c$ for some $k \in \{1, \dots, c\}$. Note that $\sum_{k=1}^c s_{ik} = 1$ where $s_{ik} \in \mathbf{S}$.

The MinCutPool [1] penalty term is a normalized version of $\|\mathbf{S}^T \mathbf{S} - \frac{n}{c} \mathbf{I}\|_F$. Note that $\|\cdot\|_F$ is the Frobenius norm, $\mathbf{I}$ is the identity matrix, and $\frac{n}{c}$ is the number of nodes per cluster if they were divided equally. See [1] for the original formulation. The regularization term encourages the row vectors of the cluster assignment matrix to be orthogonal and balanced (the same number of nodes in every cluster). In [21], the authors show that the MinCutPool regularization term dominates optimization and offer an alternative — the DMoN penalty term.

The DMoN penalty term is

$$\frac{\sqrt{c}}{n} \left\| \sum_{i=1}^n \mathbf{s}_i \right\|_F - 1,$$

where $\|\cdot\|_F$ is the Frobenius norm and $\mathbf{s}_i$ is the i -th row vector of $\mathbf{S}$. Similar to the MinCutPool penalty, it was designed to balance clusters. In addition, the authors used it to discourage a trivial solution where all nodes were placed in one cluster. It was proposed as a less dominant penalty term in that the community quality function would be the focus of the optimization, *not* the penalty term. However, it is satisfied (i.e. has its lowest value) both when clusters are balanced and when the cluster assignment is uncertain (i.e. when each cluster has the same probability assignment). We illustrate this mathematically.

In the worst case, when all nodes are assigned to the same cluster k , we have

$$\sum_{i=1}^n \mathbf{s}_i = n \mathbf{e}_k$$

and subsequently

$$\left\| \sum_{i=1}^n \mathbf{s}_i \right\|_F = n.$$

Hence,

$$\begin{aligned} \frac{\sqrt{c}}{n} \left\| \sum_{i=1}^n \mathbf{s}_i \right\|_F - 1 &= \frac{\sqrt{c}}{n} n - 1 \\ &= \sqrt{c} - 1. \end{aligned}$$

In the best case (when each cluster has the same number of nodes (i.e. balanced)):

$$\sum_{i=0}^n \mathbf{s}_i = \left[\frac{n}{c} \dots \frac{n}{c} \right].$$

Table A.2. Additional synthetic network statistics where p_{in} is the intra-cluster edge probability and p_{out} is the between-cluster edge probability. We generated 10 networks for each network type (i.e. there are 10 *small* networks with *low* density). Hence, the average node degree, density, and modularity are approximations.

Size label	Clusters	Density label	Average node degree	Density	p_{in}	p_{out}	Modularity
Small	5	Low	~ 4	~ 0.04	0.15	0.015	~ 0.49
		Medium	~ 11	~ 0.1	0.4	0.04	~ 0.48
Medium	5	Low	~ 25	~ 0.02	0.1	0.002	~ 0.67
		Medium	~ 100	~ 0.1	0.4	0.007	~ 0.67
		High	~ 154	~ 0.15	0.6	0.015	~ 0.65
	10	Medium	~ 72	~ 0.07	0.4	0.002	~ 0.69
	20	Medium	~ 51	~ 0.05	0.8	0.004	~ 0.84
Large	5	Low	~ 103	~ 0.01	0.033	0.0003	~ 0.59

Hence

$$\begin{aligned}
 \left\| \sum_{i=0}^n \mathbf{s}_i \right\|_F &= \sqrt{c \left(\frac{n}{c} \right)^2} \\
 &= \sqrt{\frac{n^2}{c}} \\
 &= \frac{n}{\sqrt{c}}.
 \end{aligned}$$

As such

$$\begin{aligned}
 \frac{\sqrt{c}}{n} \left\| \sum_{i=0}^n \mathbf{s}_i \right\|_F - 1 &= \frac{\sqrt{c}}{n} \times \frac{n}{\sqrt{c}} - 1 \\
 &= 1 - 1 \\
 &= 0.
 \end{aligned}$$

In the case where the node assignment is uncertain (i.e. $s_{ik} = \frac{1}{c}, \forall i \in \{0, \dots, n\}, k \in \{0, \dots, c\}$):

$$\sum_{i=0}^n \mathbf{s}_i = \left[\frac{n}{c} \dots \frac{n}{c} \right].$$

Hence

$$\frac{\sqrt{c}}{n} \left\| \sum_{i=0}^n \mathbf{s}_i \right\|_F - 1 = 0.$$

As such, the DMoN penalty term has the same value (i.e. 0) both when the nodes are perfectly balanced across clusters and when the node-cluster assignment is uncertain.

A.9. ARI

To evaluate clustering quality, we report the ARI, which measures the agreement between a predicted partition and a ground-truth partition while correcting for chance [10]. Given two partitions of n nodes, we form a contingency table with entries n_{ij} counting how many nodes are simultaneously assigned to cluster i in the first partition and cluster j in the second. Let $a_i = \sum_j n_{ij}$ and $b_j = \sum_i n_{ij}$ denote the row and column sums, respectively. The ARI is defined as

Table A.3. Cluster sizes of each network.

Size label	Clusters	Size of clusters
Small	5	25, 30, 10, 20, 15
Medium	5	39, 175, 236, 270, 280
	10	10, 23, 27, 32, 38, 97, 108, 170, 229, 266
	20	20, 31, 71, 73, 29, 19, 21, 32, 15, 65, 60, 53, 76, 80, 70, 27, 62, 61, 85, 50
Large	5	175, 478, 2358, 2989, 4000

$$\text{ARI} = \frac{\sum_{i,j} \binom{n_{ij}}{2} - \frac{\sum_i \binom{a_i}{2} \sum_j \binom{b_j}{2}}{\binom{n}{2}}}{\frac{1}{2} \left[\sum_i \binom{a_i}{2} + \sum_j \binom{b_j}{2} \right] - \frac{\sum_i \binom{a_i}{2} \sum_j \binom{b_j}{2}}{\binom{n}{2}}}. \quad (\text{A.1})$$

The ARI takes the value 1 for identical partitions, is close to 0 for random unrelated partitions, and can be negative when the agreement is worse than chance.

A.10. Hyperparameters

We summarize here the hyperparameters used in our experiments. For the real-world datasets, we employ a 4-layer GNN with regularization weights set to $\mu = \lambda = 1$. For the synthetic datasets, we use a 3-layer GNN, again with $\mu = \lambda = 1$. In all cases, we use a 2-layer MLP. Unless otherwise stated, all models are trained with a learning rate of 10^{-3} using the Adam optimizer for 3000 epochs. All experiments were run on Intel Xeon Gold CPUs with 16 cores.

A.11. Additional synthetic network statistics

Density is a measure of how close a network is to being fully connected. It is defined as:

$$\text{Density} = \frac{2m}{n(n-1)} \quad (\text{A.2})$$

where the numerator is the number of edges in the network, and the denominator is the total possible edges in a fully-connected network (with no self-loops).

Moreover, we provide additional statistics in tables A.2 and A.3 related to the synthetic networks, namely: the average node degree, community sizes, the p_{in} and p_{out} values used to generate the synthetic networks, as well as the ground-truth modularity. The average node degree is defined as:

$$\frac{2m}{n}. \quad (\text{A.3})$$

The p_{in} and p_{out} from table A.2 and the cluster sizes from table A.3 can be used to reproduce the synthetic networks using the Python package *PyTorch Geometric* version 2.5.2 [3].

ORCID iDs

Kibidi Neocosmos  0000-0001-7747-7980

Diego Baptista  0000-0003-2994-0138

Nicole Ludwig  0000-0003-3230-8918

References

- [1] Bianchi F M, Grattarola D and Alippi C 2020 Spectral clustering with graph neural networks for graph pooling *Proc. 37th Int. Conf. on Machine Learning* (PMLR) pp 874–83

- [2] Derrow-Pinion A, She J, Wong D, Lange O, Hester T, Perez L, Nunkesser M, Lee S, Guo X, Wiltshire B *et al* 2021 Eta prediction with graph neural networks in google maps *Proc. 30th ACM Int. Conf. on Information & Knowledge Management* pp 3767–76
- [3] Fey M *et al* 2025 PyG 2.0: Scalable learning on real world graphs *Temporal Graph Learning Workshop @ KDD*
- [4] Fortunato S and Barthelemy M 2007 Resolution limit in community detection *Proc. Natl Acad. Sci.* **104** 36–41
- [5] Fortunato S and Hric D 2016 Community detection in networks: A user guide *Phys. Rep.* **659** 1–44
- [6] Good B H, de Montjoye Y-A and Clauset A 2010 Performance of modularity maximization in practical contexts *Phys. Rev. E* **81** 046106
- [7] Grattarola D, Zambon D, Bianchi F M and Alippi C 2022 Understanding pooling in graph neural networks *IEEE Trans. on Neural Networks and Learning Systems* pp 1–11
- [8] Hagen L and Kahng A B 1992 New spectral methods for ratio cut partitioning and clustering *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **11** 1074–85
- [9] Hamilton W, Ying Z and Leskovec J 2017 Inductive representation learning on large graphs *Advances in Neural Information Processing Systems* vol 30 (Curran Associates, Inc.)
- [10] Hubert L and Arabie P 1985 Comparing partitions *J. Classifi.* **2** 193–218
- [11] Mirhoseini A *et al* 2021 A graph placement methodology for fast chip design *Nature* **594** 207–12
- [12] Neocosmos K, Baptista D and Ludwig N 2025 Synthetic network data for bounded graph clustering with graph neural networks Zenodo (<https://doi.org/10.5281/zenodo.17793018>)
- [13] Newman M E J 2006 Modularity and community structure in networks *Proc. Natl Acad. Sci.* **103** 8577–82
- [14] Paszke A *et al* 2019 Pytorch: an imperative style, high-performance deep learning library *Advances in Neural Information Processing Systems* vol 32
- [15] Pei H, Wei B, Chang K C-C, Lei Y and Yang B 2020 Geom-GCN: geometric graph convolutional networks *Proc. Int. Conf. on Learning Representations (ICLR)*
- [16] Reichardt J and Bornholdt S 2006 Statistical mechanics of community detection *Phys. Rev. E* **74** 016110
- [17] Sen P, Namata G, Bilgic M, Getoor L, Galligher B and Eliassi-Rad T 2008 Collective classification in network data *AI Mag.* **29** 93–106
- [18] Shi J and Malik J 2000 Normalized cuts and image segmentation *IEEE Trans. Pattern Anal. Mach. Intell.* **22** 888–905
- [19] Stokes J M *et al* 2020 A deep learning approach to antibiotic discovery *Cell* **180** 688–702
- [20] Team PyG Pytorch geometric datasets (available at: <https://pytorch-geometric.readthedocs.io/en/2.7.0/modules/datasets.html>) (accessed 10 November 2025)
- [21] Tsitsulin A, Palowitch J, Perozzi B and Müller E 2023 Graph clustering with graph neural networks *J. Mach. Learn. Res.* **24** 1–21
- [22] Shoujun X, Wang J, Zhang J and Zhang X-S 2011 Multiple resolution community structure analysis *Proc. 10th Int. Symp. on Operations Research and Its Applications* vol 14 pp 208–17
- [23] Ying R, Ruining H, Chen K, Eksombatchai P, Hamilton W L and Leskovec J 2018 Graph convolutional neural networks for web-scale recommender systems *Proc. 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* pp 974–83
- [24] Ying Z, You J, Morris C, Ren X, Hamilton W and Leskovec J 2018 Hierarchical Graph representation learning with differentiable pooling *Advances in Neural Information Processing Systems* vol 31 (Curran Associates, Inc.)