Total Variation Graph Neural Networks

Jonas Berg Hansen [*1] **Filippo Maria Bianchi** [*1 2]

Abstract

Recently proposed Graph Neural Networks (GNNs) for vertex clustering are trained with an unsupervised minimum cut objective, approximated by a Spectral Clustering (SC) relaxation. However, the SC relaxation is loose and, while it offers a closed-form solution, it also yields overly smooth cluster assignments that poorly separate the vertices. In this paper, we propose a GNN model that computes cluster assignments by optimizing a tighter relaxation of the minimum cut based on graph total variation (GTV). The cluster assignments can be used directly to perform vertex clustering or to implement graph pooling in a graph classification framework. Our model consists of two core components: i) a message-passing layer that minimizes the ℓ_1 distance in the features of adjacent vertices, which is key to achieving sharp transitions between clusters; ii) an unsupervised loss function that minimizes the GTV of the cluster assignments while ensuring balanced partitions. Experimental results show that our model outperforms other GNNs for vertex clustering and graph classification.

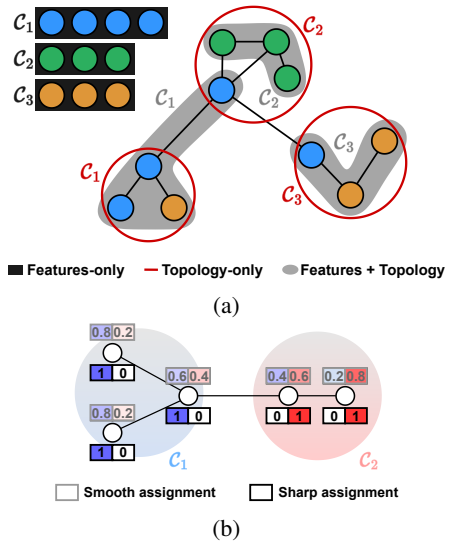


Figure 1: **a)** Most clustering methods partition the data only based on the features (■). SC partitions the vertices of the graph based on its topology (■). GNNs account both for the vertex features and the graph topology(■). **b)** Difference between smooth and sharp cluster assignments. Especially at the edge of a cluster, the smooth assignments give large weights to more than one cluster.

1. Introduction

Traditional clustering techniques partition samples based on their features or on suitable data representations computed, for example, with deep learning models (Tian et al., 2014; Min et al., 2018; Su et al., 2022). Spectral clustering (SC) (Von Luxburg, 2007) is a popular technique that first encodes the similarity of the data features into a graph and then creates a partition based on the graph topology. Such a graph is just a convenient representation of the similarity among the samples and has no attributes on its vertices. On the other hand, an attributed graph can represent both the relationships among samples and their features. Graph Neural Networks (GNNs) are deep learning architectures specifically designed to process and make inference on such data (Hamilton, 2020). Therefore, contrarily to traditional clustering methods, a GNN-based approach for clustering can account for both the features and the relationships among samples to generate partitions (see Fig.1a).

Similarly to other deep learning architectures for clustering (Shaham et al., 2018; Kampffmeyer et al., 2019), GNNs can be trained end-to-end and return soft cluster assignments as part of the output. Several existing GNN clustering approaches compute cluster assignments from the vertex representations generated by message passing (MP) layers and, then, optimize the assignments with an unsupervised loss inspired by SC (Bianchi et al., 2020; Tsitsulin et al., 2020; Duval & Malliaros, 2022). The SC objective benefits from the smoothing operations performed by the MP layers, which minimize the local quadratic variation of adjacent vertex features. However, this approach produces smooth cluster assignment vectors that are less informative as they

[*]Equal contribution [1]Department of Mathematics and Statistics, UiT the Arctic University of Norway [2]NORCE, The Norwegian Research Centre AS. Correspondence to: Filippo Maria Bianchi <filippo.m.bianchi@uit.no>.

Proceedings of the 40th International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023. Copyright 2023 by the author(s).

do not separate well the samples (see Fig.1b). Indeed, the SC objective is known to give a loose approximation of the optimal partition defined in terms of the minimum cut (Rangapuram et al., 2014).

Contributions. We propose a novel GNN-based clustering approach that generates cluster assignments by optimizing the graph total variation (GTV). Compared to SC, GTV trades a closed-form solution with a tighter continuous relaxation of the optimal minimum cut partition (Hein & Setzer, 2011). Notably, an optimization objective with a closed-form solution is not particularly useful for a GNN trained with gradient descent.

We design an unsupervised loss function for minimizing the GTV that is GNN-friendly as it avoids numerical issues during gradient descent and assumes values in a well-defined range, making it suitable to be combined with other losses. Our GNN yields sharp cluster assignments by minimizing the ℓ_1 norm of their differences. Clearly, the minimization of GTV is hindered when cluster assignments are derived from smooth representations computed by traditional MP layers. To address this issue, we propose a new MP layer that minimizes the ℓ_1 norm of the difference between adjacent vertex features.

By optimizing the proposed loss, we can train a GNN end-to-end to perform vertex clustering. In addition, by coarsening the graph according to the learned partition we can perform hierarchical graph pooling (Grattarola et al., 2022) in deep GNN architectures for graph-level tasks, such as graph classification. In this case, the proposed loss is combined with an additional supervised loss, such as the cross-entropy. Experiments show the superiority of the proposed approach compared to other GNN methods for clustering and graph pooling.

2. Background

A graph is represented by a tuple $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ and $\mathcal{E}$ are the vertex and edge sets, respectively. The cardinality of the sets are given as $|\mathcal{V}| = N$ and $|\mathcal{E}| = E$. The adjacency matrix $\mathbf{A} \in \mathbb{R}^{N\times N}$ with elements $a_{ij} \in \{0, 1\}$ defines the graph connectivity. In an attributed graph, each vertex i is associated with a feature vector $\boldsymbol{x}_i \in \mathbb{R}^F$. Feature vectors are often grouped in a matrix $\mathbf{X} \in \mathbb{R}^{N\times F}$. The soft cluster assignment matrix is $\mathbf{S} \in \mathbb{R}^{N\times K}$, where K is the number of clusters and $s_{ij} \in [0, 1]$ is the membership of vertex i to cluster j. The combinatorial Laplacian is $\mathbf{L} = \mathbf{D} - \mathbf{A}$ and $\tilde{\mathbf{A}} = \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$ is the symmetric degree normalization of $\mathbf{A}$.

2.1. Graph cuts

The task of finding K clusters of similar size can be cast into the balanced K-cut problem, defined as a ratio of two set functions:

$$\mathcal{C} = \min_{C_1,\dots,C_K} \sum_{k=1}^{K} \frac{\text{cut}(C_k, \bar{C}_k)}{\hat{S}(C_k)} \quad \text{s.t.} \quad C_i \cap C_j = \emptyset, \tag{1}$$

where $\text{cut}(C_i, C_j)$ counts the volume of edges connecting the two sets of vertices $C_i, C_j \subset \mathcal{V}$, $\bar{C}_k$ is the complement of set C_k, and $\hat{S}(\cdot) : 2^{\mathcal{V}} \to \mathbb{R}_+$ is a submodular set function that balances the size of the clusters in the partition (Hein & Setzer, 2011). Depending on the choice of $\hat{S}(\cdot)$ different cuts are obtained, such as the ratio cut for $\hat{S}(C_k) = |C_k|$, and the normalized cut for $\hat{S}(C_k) = \text{vol}(C_k)$ (Von Luxburg, 2007). Of particular interest for us is the Cheeger cut, also known as Balanced cut or Ratio Cheeger cut, where $\hat{S}(C_k) = \min\{|C_k|, |\bar{C}_k|\}$, which penalizes the formation of very large clusters. A variant of the Cheeger Cut, called *Asymmetric Cheeger cut*, encourages an even partition by letting $\hat{S}(C_k) = \min\{(K-1)|C_k|, |\bar{C}_k|\}$ (Bresson et al., 2013).

2.2. Tight relaxation of the Asymmetric Cheeger cut

Let the numerator in (1) be expressed in matrix form as

$$\text{cut}(C_k, \bar{C}_k) = \sum_{i\in C_k, j\in \bar{C}_k} a_{ij}(1 - z_i z_j) = \boldsymbol{z}^T \mathbf{L} \boldsymbol{z}, \tag{2}$$

with $z_i, z_j \in \{-1, 1\}$ (derivation in A.1). The common relaxation done in spectral clustering (SC) is:

$$\min_{\boldsymbol{z}\in\{-1,1\}^N} \frac{\boldsymbol{z}^T \mathbf{L} \boldsymbol{z}}{\hat{S}(C_k)} \to \min_{\boldsymbol{s}\in\mathbb{R}^N} \frac{\boldsymbol{s}^T \mathbf{L} \boldsymbol{s}}{S(C_k)} \tag{3}$$

where $S(C_k)$ is the continuous counterpart of $\hat{S}(C_k)$. What the SC relaxation actually does, is to apply Laplacian smoothing to a graph signal $\boldsymbol{s}$ by minimizing its *local quadratic variation* (LQV), i.e., the quadratic variation of $\boldsymbol{s}$ across adjacent vertices. The LQV defined in terms of the combinatorial Laplacian reads

$$\boldsymbol{s}^T \mathbf{L} \boldsymbol{s} = \frac{1}{2} \sum_{(i,j)\in\mathcal{E}} a_{ij}(s_i - s_j)^2. \tag{4}$$

From the graph signal processing perspective, Laplacian smoothing applies to a graph signal a low-pass filter with response $(1-\lambda_i)$, where λ_i is the i-th eigenvalue of the Laplacian (Tremblay et al., 2018). Despite offering a closed-form solution, SC gives a loose approximation of the solution of the discrete optimization problem (Hein & Setzer, 2011).

Let $\boldsymbol{s}_k \in \mathbb{R}^N$ be the soft assignment vector for the k-th cluster. A tighter continuous relaxation of the Asymmetric Cheeger cut problem is given by (Bresson et al., 2013):

$$\min_{\boldsymbol{s}_1,\dots,\boldsymbol{s}_K\in\mathbb{R}^N} \sum_{k=1}^{K} \frac{||\boldsymbol{s}_k||_{\text{GTV}}}{||\boldsymbol{s}_k - \text{quant}_\rho(\boldsymbol{s}_k)||_{1,\rho}} \quad \text{s.t.} \quad \sum_{k=1}^{K} \boldsymbol{s}_k = \mathbf{1}_N. \tag{5}$$

In the numerator, $||\boldsymbol{s}_k||_{\text{GTV}} = \sum_{i,j} a_{ij}|s_{i,k} - s_{j,k}|$ measures the graph total variation (GTV) of the soft assignments to cluster k. In the denominator, $\text{quant}_\rho(\boldsymbol{s}_k)$ denotes the ρ-quantile of $\boldsymbol{s}_k$, i.e., the $(q+1)^{\text{st}}$ largest value in $\boldsymbol{s}_k$ with $q = \lfloor N/(\rho+1) \rfloor$, while $||\cdot||_{1,\rho}$ denotes an asymmetric ℓ_1 norm, which for a vector $\boldsymbol{x} \in \mathbb{R}^N$ is defined as

$$||\boldsymbol{x}||_{1,\rho} = \sum_{i=1}^{N} |x_i|_\rho, \text{ where } |x_i|_\rho = \begin{cases} \rho x_i, & x_i \geq 0 \\ -x_i, & x_i < 0 \end{cases}. \quad (6)$$

When $\rho = K - 1$, the denominator in (5) encourages balanced partitions, i.e., clusters having similar sizes, and prevents the two common *degenerate solutions*: i) all samples collapsing into the same cluster (e.g., $\boldsymbol{s}_i = [1, 0, \ldots, 0], \forall i$), and ii) samples uniformly assigned to all clusters ($\boldsymbol{s}_i = [1/K, 1/K, \ldots, 1/K], \forall i$).

Differently from SC, (5) minimizes the ℓ_1 rather than the ℓ_2 distance between components of $\boldsymbol{s}$ that are adjacent on the graph. Minimizing GTV yields sharper cluster assignments and achieves a tighter relaxation of the balanced K-cut problem compared to SC (Rangapuram et al., 2014; Bresson et al., 2013). However, contrarily to SC, (5) it is non-convex and is optimized through iterative updates.

2.3. Clustering with Graph Neural Networks

To leverage the graph topology for learning vertex representations, GNNs implement message-passing (MP) layers. Typically, an MP layer collects information from the neighbours to update the representation of each vertex (Hamilton, 2020). An example of an MP layer is the Graph Convolutional Network (GCN) by (Kipf & Welling, 2017):

$$\mathbf{X}^{(\text{out})} = \sigma(\tilde{\mathbf{A}}'\mathbf{X}^{(\text{in})}\boldsymbol{\Theta}_{\text{MP}}) \quad (7)$$

where $\mathbf{A}' = \mathbf{A} + \mathbf{I}_N$ and $\boldsymbol{\Theta}_{\text{MP}} \in \mathbb{R}^{F_{\text{in}} \times F_{\text{out}}}$ are learnable parameters. It can be shown that a GCN layer applies Laplacian smoothing on the vertex features $\mathbf{X}$; see (Wu et al., 2019; Nt & Maehara, 2019; Li et al., 2018; Bianchi et al., 2021) for detailed discussions. For example, (Ma et al., 2021) shows that the l-th GCN layer updates of the vertex features as

$$\mathbf{X}^{(l+1)} = (\mathbf{I} - \delta\tilde{\mathbf{A}}')\mathbf{X}^{(l)}, \quad (8)$$

which is actually one gradient descent step of size δ in the optimization of the problem

$$\min_{\mathbf{X}} \text{Tr}(\mathbf{X}^T(\mathbf{I} - \tilde{\mathbf{A}}')\mathbf{X}). \quad (9)$$

After applying a stack of L MP layers, the soft assignment matrix $\mathbf{S}$ is obtained by passing the updated vertex features to some function f, typically a multi-layer perceptron (MLP) with a `Softmax` activation, that produces a K-dimensional vector for each vertex. Importantly, since the MP layers perform Laplacian smoothing and f is a smooth function, the cluster assignments generated from $\mathbf{X}^{(L)}$ comply with the SC definition. Nevertheless, minimizing the LQV of $\mathbf{X}^{(L)}$ is not the same as minimizing the LQV of $\mathbf{S}$ directly. In addition, the size of the clusters must be balanced and degenerate solutions avoided. For these reasons, the assignments $\mathbf{S}$ are further optimized through unsupervised loss functions. For instance, MinCutPool (Bianchi et al., 2020) optimizes the following loss

$$\underbrace{-\frac{\text{Tr}(\mathbf{S}^T\tilde{\mathbf{A}}\mathbf{S})}{\text{Tr}(\mathbf{S}^T\tilde{\mathbf{D}}\mathbf{S})}}_{\mathcal{L}_c} + \underbrace{\left\| \frac{\mathbf{S}^T\mathbf{S}}{\|\mathbf{S}^T\mathbf{S}\|_F} - \frac{\mathbf{I}_K}{\sqrt{K}} \right\|_F}_{\mathcal{L}_o}, \quad (10)$$

where $\|\cdot\|_F$ indicates the Frobenius norm and $\tilde{\mathbf{D}} = \text{diag}(\tilde{\mathbf{A}}\mathbf{1})$. The $\mathcal{L}_c$ term encourages strongly connected components to be clustered together, while $\mathcal{L}_o$ is a balancing term that promotes equally-sized clusters and helps to avoid degenerate solutions. Similarly, DMoN (Tsitsulin et al., 2020) optimizes a two-termed loss

$$\underbrace{-\frac{\text{Tr}(\mathbf{S}^T\mathbf{A}\mathbf{S} - \mathbf{S}^T\mathbf{d}^T\mathbf{d}\mathbf{S})}{2|\mathcal{E}|}}_{\mathcal{L}_m} + \underbrace{\frac{\sqrt{K}}{N}\left\|\sum_i \mathbf{S}_i^T\right\|_F - 1}_{\mathcal{L}_r}, \quad (11)$$

where $\mathbf{d}$ is the degree vector of $\mathbf{A}$, $\mathcal{L}_m$ pushes strongly connected components to the same cluster, and $\mathcal{L}_r$ is a regularization term that penalizes the degenerate solutions.

The losses in (10) and (11) are closely related to SC, from which the problem of smooth cluster assignments is inherited. In addition, being trained with gradient descent, the GNNs do not exploit the closed-form solution of SC. This motivates relying on GTV to perform clustering with a GNN.

3. Total Variation Graph Neural Network

In this section, we present the two core components of the Total Variation Graph Neural Network (TVGNN). First, we introduce the unsupervised clustering loss inspired by the GTV relaxation. Then, we present a novel MP layer to be used in conjunction with the proposed loss. We conclude by showing two specific TVGNN architectures that can be used for vertex clustering and for graph classification.

3.1. The loss function

In principle, the optimization of the objective in (5) yields a balanced partition with sharp transitions in the assignment vectors of adjacent vertices belonging to different clusters. However, the relaxed Asymmetric Cheeger Cut expression is ill-suited for the stochastic gradient descent used to train a GNN, due to potential numerical issues in the proximity

of the degenerate solutions. Specifically, all cluster assignments become similar when a degenerate solution is approached, bringing both the numerator and the denominator in (5) close to zero and creating numerical instability in the gradients. Adding small constants to avoid zero division can mitigate the issue only partially as it hinders the effect of the balancing term, which is what actually prevents degenerate solutions. Finally, we desire a loss that could be easily combined with other losses (e.g., the cross-entropy) when the cluster assignments are used to implement graph pooling in a deep GNN for graph classification (see Sec. 3.3). For this purpose, it is desirable to control the range of possible values it can assume and, therefore, a denominator taking arbitrary small values should be avoided.

To satisfy these requirements while retaining the desired properties of the objective function, we construct a loss by adding the GTV and balancing terms rather than taking their ratio. First, we define the GTV loss term as

$$\mathcal{L}^*_{\text{GTV}} = ||\mathbf{S}||_{\text{GTV}} = \sum_{k=1}^{K}\sum_{i=1}^{N}\sum_{j=i}^{N} a_{ij}|s_{ik} - s_{jk}|, \tag{12}$$

Then, we define the asymmetrical norm term as

$$\mathcal{L}^*_{\text{AN}} = \sum_{k=1}^{K} ||\boldsymbol{s}_{:k} - \text{quant}_\rho(\boldsymbol{s}_{:k})||_{1,\rho}. \tag{13}$$

To control the range of values of the loss, the two terms are rescaled as follows:

$$\mathcal{L}_{\text{GTV}} = \frac{\mathcal{L}^*_{\text{GTV}}}{2E} \in [0,1], \tag{14}$$

$$\mathcal{L}_{\text{AN}} = \frac{\beta - \mathcal{L}^*_{\text{AN}}}{\beta} \in [0,1], \tag{15}$$

where E is the number of edges in the graph and

$$\beta = \begin{cases} N\rho & \text{when } \rho = K-1, \\ N\rho\min(1, K/(\rho+1)) & \text{otherwise.} \end{cases} \tag{16}$$

The final loss reads:

$$\mathcal{L} = \alpha_1\mathcal{L}_{\text{GTV}} + \alpha_2\mathcal{L}_{\text{AN}}, \tag{17}$$

where $\alpha_1, \alpha_2 \in \mathbb{R}$ are hyperparameters that weigh the relative and total (in case there are other losses) contribution of the loss components. For the experiments in Sec. 4 we set $\rho = K-1$, for which $\mathcal{L}_{\text{AN}}$ will be minimized for balanced clusters. Without prior knowledge, a balanced clustering is a reasonable bias. Nevertheless, the balancing term only poses a soft constraint that can be violated if needed, *e.g.*, if the data show a clustering structure that is clearly uneven (see the additional results in C.1).

The cluster assignments are computed by an `MLP` fed with the vertex representations produced by a stack of L MP layers

$$\mathbf{S} = \texttt{Softmax}(\texttt{MLP}(\mathbf{X}^{(L)}; \mathbf{\Theta}_{\text{MLP}})) \tag{18}$$

As discussed in Sec. 2.3, common MP layers minimize the LQV of vertex features and the `MLP`, which is a smooth function, will naturally preserve the Laplacian smoothing effect when computing $\mathbf{S}$. While this was suitable for an SC objective, the optimization problem expressed by $\mathcal{L}_{\text{GTV}}$ entails minimizing the ℓ_1 norm $\|\boldsymbol{s}_i - \boldsymbol{s}_j\|_1$ of each pair of adjacent vertices i and j. Thus, the MP layers should ideally minimize the discrepancy in the features of adjacent vertices in a ℓ_1 norm sense. To satisfy this requirement, we design a new MP layer starting from the definition of the gradient of an approximated GTV function.

3.2. The GTVConv layer

Our goal is to design an MP layer that minimizes the GTV of the vertex features. While the GTV can be expressed by means of an incidence matrix (Wang et al., 2016) or a (nonlinear) ℓ_1-Laplacian operator (Bai et al., 2018; Zhou & Schölkopf, 2005), these formulations are difficult to integrate within a standard MP layer that operates on a connectivity matrix. Furthermore, GTV is non-differentiable for $x_i = x_j$.

To address these issues, we define the GTV Laplacian as

$$\mathbf{L}_{\mathbf{\Gamma}} = \mathbf{D}_{\mathbf{\Gamma}} - \mathbf{\Gamma}, \quad \text{with} \quad \mathbf{D}_{\mathbf{\Gamma}} = \text{diag}(\mathbf{\Gamma}\mathbf{1}), \tag{19}$$

where $\mathbf{\Gamma}$ is a connectivity matrix matching the sparsity pattern of the adjacency matrix, with elements

$$[\mathbf{\Gamma}]_{ij} = \gamma_{ij} = \frac{a_{ij}}{\max\{|x_i - x_j|, \epsilon\}}, \tag{20}$$

where ϵ is a small constant, whose purpose will be clarified soon. To minimize the GTV, we compute the partial derivative of $||\boldsymbol{x}||_{\text{GTV}}$ with respect to vertex k for some $\boldsymbol{x} \in \mathbb{R}^N$. According to the definition of $\mathbf{\Gamma}$ and assuming that the graph is undirected, the partial derivative reads

$$\frac{\partial}{\partial x_k}(||\boldsymbol{x}||_{\text{GTV}})_\epsilon = 2\sum_{j=1}^{N}\gamma_{kj}\cdot(x_k - x_j), \tag{21}$$

and the full GTV gradient is

$$\nabla(||\boldsymbol{x}||_{\text{GTV}})_\epsilon = 2\mathbf{D}_{\mathbf{\Gamma}}\boldsymbol{x} - 2\mathbf{\Gamma}\boldsymbol{x} = 2(\mathbf{D}_{\mathbf{\Gamma}} - \mathbf{\Gamma})\boldsymbol{x} = 2\mathbf{L}_{\mathbf{\Gamma}}\boldsymbol{x}. \tag{22}$$

By looking at (21), we now see that the constant ϵ ensures numerical stability by avoiding the discontinuity in the derivative of $||\boldsymbol{x}||_{\text{GTV}}$. Indeed, using $\max\{|x_i - x_j|, \epsilon\}$ rather than $|x_i - x_j|$ as the denominator in (20) corresponds to modifying the GTV function and its derivative in the proximity of $|x_i - x_j| = 0$, as shown in Fig. 2. A detailed discussion and the derivations are deferred to Appendix A.2.

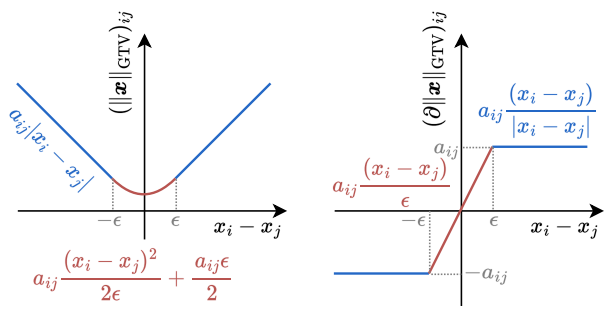


Figure 2: The GTV function (in blue) is modified (parts in red) near zero to avoid the discontinuity in the derivative.

Based on (22), the ϵ-approximation of $||\boldsymbol{x}||_{\text{GTV}}$ is minimized by taking the following gradient descent update

$$\boldsymbol{x}^{(t+1)} = \left(\mathbf{I} - 2\delta \mathbf{L}_{\mathbf{\Gamma}}{}^{(t)}\right) \boldsymbol{x}^{(t)} \tag{23}$$

where δ is the step size. The update in (23) closely resembles the update in (8) and it can be implemented by a GCN layer operating on the connectivity matrix $\mathbf{I} - 2\delta \mathbf{L}_{\mathbf{\Gamma}}^{(t)}$. We note that the superscript (t) in $\mathbf{L}_{\mathbf{\Gamma}}^{(t)}$ indicates the dependency on the features $\boldsymbol{x}^{(t)}$ in the denominator of (20). Without loss of generality, from now on we replace index t, which indicates the t-th step in the gradient descent update, with layer index l, meaning that each MP layer in the GNN performs a gradient descent update to minimize the GTV.

The aggregation procedure in (23) is only valid for univariate vertex features $\boldsymbol{x} \in \mathbb{R}^N$. For a graph with multi-dimensional features $\mathbf{X} \in \mathbb{R}^{N \times F}$ the partial derivative is

$$\frac{\partial}{\partial x_{kf}}(||\mathbf{X}||_{\text{GTV}})_\epsilon = \sum_{j=1}^{N} \gamma_{kjf} \cdot (x_{kf} - x_{jf}), \tag{24}$$

where

$$\gamma_{kjf} = \frac{a_{kj}}{\max\{|x_{kf} - x_{jf}|, \epsilon\}}, \; f = 1, \ldots, F. \tag{25}$$

The dependence on f implies that we need a distinct $\mathbf{\Gamma}_f$ for each feature and that the update in (23) must be done feature-wise. Notably, each time an MP layer maps the vertex features in a new F'-dimensional space, F' different connectivity matrices are required. Clearly, this introduces two major drawbacks. First, building, storing, and applying many $\mathbf{\Gamma}_f$ matrices is computationally expensive, especially for large graphs. Second, since each feature is updated independently, the minimization of the approximated $\|\mathbf{X}\|_{\text{GTV}}$ can exhibit erratic behaviors and fail to converge to the optimal solution. See A.3 for a detailed discussion and an example.

To address these issues, we modify the gradient descent step by defining a single operator that computes the ℓ_1 distance over the full feature vectors:

$$\hat{\gamma}_{ij} = \frac{a_{ij}}{\max\{||\boldsymbol{x}_i - \boldsymbol{x}_j||_1, \epsilon\}}. \tag{26}$$

By letting

$$\hat{\mathbf{L}}_{\mathbf{\Gamma}} = \hat{\mathbf{D}}_{\mathbf{\Gamma}} - \hat{\mathbf{\Gamma}}, \quad \text{with} \quad [\hat{\mathbf{\Gamma}}]_{ij} = \hat{\gamma}_{ij} \quad \text{and} \quad \hat{\mathbf{D}}_{\mathbf{\Gamma}} = \text{diag}(\hat{\mathbf{\Gamma}}\mathbf{1}), \tag{27}$$

the vertex features update at step/layer $l+1$ becomes

$$\mathbf{X}^{(l+1)} = \left(\mathbf{I} - 2\delta \hat{\mathbf{L}}_{\mathbf{\Gamma}}^{(l)}\right) \mathbf{X}^{(l)}. \tag{28}$$

By referring to the notation in (7), the proposed GTVConv layer reads

$$\mathbf{X}^{(l+1)} = \sigma\left[\left(\mathbf{I} - 2\delta \hat{\mathbf{L}}_{\mathbf{\Gamma}}^{(l)}\right) \mathbf{X}^{(l)} \mathbf{\Theta}_{\text{MP}}^{(l)}\right] \tag{29}$$

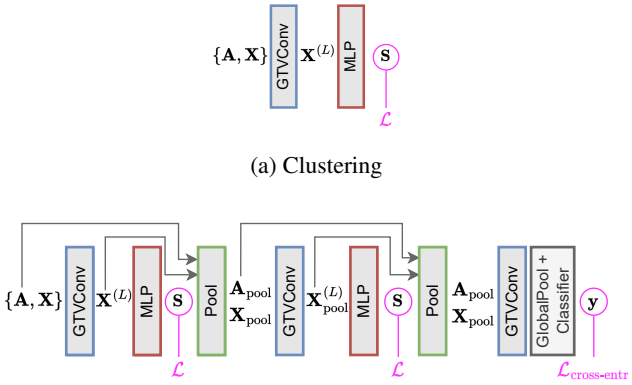
Remarks There is a clear analogy between the LQV in (4), minimized by common MP layers, and the GTV defined in terms of $\hat{\mathbf{L}}_{\mathbf{\Gamma}}$, minimized by GTVConv. While the MP layers based on Laplacian smoothing perform low-pass filtering (Bo et al., 2021), GTVConv is closely related to graph trend filtering (Wang et al., 2016; Liu et al., 2021), which implements a total variation smoother based on the ℓ_1 Laplacian. While linear smoothers cannot handle heterogeneous smoothness, a total variation smoother encompasses both globally smooth functions, said to have homogeneous smoothness, and functions with different levels of smoothness at different graph locations (Sadhanala et al., 2016). From the graph signal processing perspective, it means applying low- and high-pass filtering on the graph signal at the same time (Fu et al., 2022). In our case, when driven by the $\mathcal{L}_{\text{GTV}}$ term in (17), GTVConv applies low-pass filtering to central vertices in the cluster and high-pass filtering at the clusters' boundary, enabling sharp cluster transitions.

Similarly to attention-based GNNs (Veličković et al., 2017), GTVConv can learn edge weights in a data-driven fashion: by looking at (26), we notice that $\hat{\gamma}_{ij}$ depend on the features $\boldsymbol{x}_i, \boldsymbol{x}_j$. Therefore, the output features at layer l will influence the edge weights at layer $l+1$.

A variant to the proposed GTVConv layer is obtained from a GTV weighted by the vertex degrees, which gives an expression related to the LQV defined in terms of the symmetric normalized Laplacian (details in A.4).

3.3. TVGNN architectures for clustering and classification

In the following, we describe the GNN architectures we used in two downstream tasks: unsupervised vertex clustering and supervised graph classification.



(a) Clustering

(b) Classification

Figure 3: Schematic depiction of the architectures used for vertex clustering and graph classification.

Vertex clustering The GNN used for clustering is depicted in Fig. 3a. The architecture is rather simple: a stack of L GTVConv layers ($L \geq 1$) generates the feature vectors $\mathbf{X}^{(L)}$ that are used by an MLP to compute the cluster assignments $\mathbf{S}$. Since clustering is an unsupervised task, the GNN is trained using only the loss function $\mathcal{L}$ defined in (17).

Graph classification Graph classification is a graph-level task, where a class label y_i is assigned to the i-th graph $\{\mathbf{A}_i, \mathbf{X}_i\}$. GNN architectures for graph classification often alternate MP layers with graph pooling layers, which gradually distill the global label information from the vertex representations (Du et al., 2021).

The key challenge in graph pooling is to generate a coarsened graph that summarizes well the properties of the original one (Bianchi & Lachi, 2023). Similarly to previous work (Ying et al., 2018; Bianchi et al., 2020), the cluster assignment matrix $\mathbf{S}$ computed in (18) is used to coarsen the adjacency matrix and to compute pooled vertex features as

$$\mathbf{A}^{\text{pool}} = \mathbf{S}^T\mathbf{A}\mathbf{S} \in \mathbb{R}^{K\times K};\ \mathbf{X}^{\text{pool}} = \mathbf{S}^T\mathbf{X} \in \mathbb{R}^{K\times F}. \quad (30)$$

According to a recently proposed taxonomy by (Grattarola et al., 2022), ours is a trainable, dense, fixed, hierarchical pooling method.

Graph pooling can be applied multiple times, to obtain smaller and smaller coarsened graphs. The features on the final coarsened graph are globally pooled and passed to a classifier that predicts the class label. A different instance of the loss in (17) is used to optimize the cluster assignments at each pooling layer. The total loss is given by combining the clustering losses, which in this case act as regularizers, and a supervised cross-entropy loss $\mathcal{L}_{\text{cross-entr}}$ between true and predicted class labels. Fig. 3b shows a schematic depiction.

By construction, each entry a^{pool}_{ij} of $\mathbf{A}^{\text{pool}}$ is the volume of edges across clusters i and j. Since the minimization of $\mathcal{L}_{\text{GTV}}$ pushes connected components to the same cluster, $\mathbf{A}^{\text{pool}}$ gradually turns into a diagonally dominant matrix. This would limit the contribution of any MP layer operating on $\mathbf{A}^{\text{pool}}$, since the vertices will share information mostly with themselves. However, in the Laplacian (27) used by the GTVConv layer, the diagonal of the coarsened adjacency is removed, which avoids this potential issue.

4. Experiments

We evaluate the proposed TVGNN model on unsupervised vertex clustering and supervised graph classification tasks. The code to implement TVGNN is publicly available[1].

4.1. Unsupervised vertex clustering

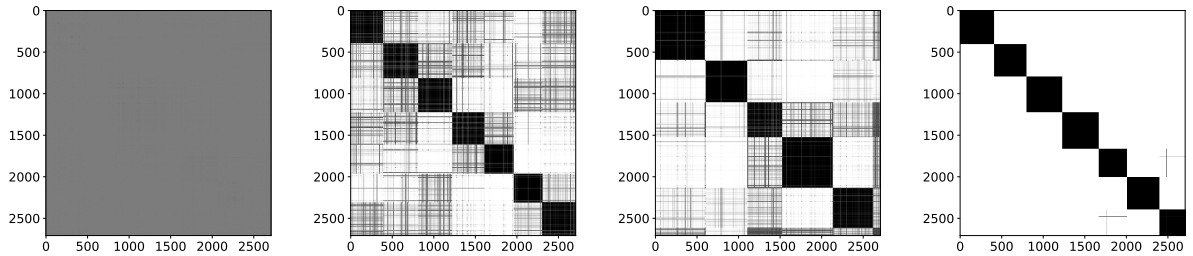
In this experiment, we evaluate the capability of TVGNN to create cluster assignments that are sharp and match well the true vertex class. The performance of TVGNN is compared against three classes of methods. The first, are algorithms that generate vertex embeddings based only on the adjacency matrix. The vertex embeddings are then clustered with k-means. Representatives of this category are Spectral Clustering (SC), DeepWalk (Perozzi et al., 2014), Node2vec (Grover & Leskovec, 2016), and NetMF (Qiu et al., 2018). The second class of methods generates vertex embeddings by accounting both for the adjacency matrix and for the vertex features. Afterward, the learned embeddings are clustered with k-means. The chosen representatives for this category are the Graph AutoEncoder (GAE) and Variational Graph AutoEncoder (VGAE) (Kipf & Welling, 2016), TADW (Yang et al., 2015), BANE (Yang et al., 2018), and TENE (Yang & Yang, 2018). Finally, the last class of methods consists of end-to-end GNN models that directly generate soft cluster assignments $\mathbf{S}$ by accounting both for the graph connectivity and the vertex features. In this case, k-means is not required and the discrete cluster assignments are simply obtained as $\boldsymbol{c} = \texttt{argmax}(\mathbf{S})$. DiffPool (Ying et al., 2018), DMoN (Tsitsulin et al., 2020), MinCutPool (Bianchi et al., 2020), and the proposed TVGNN belong to this class. The GNNs equipped with DiffPool, DMoN, and MinCutPool have the same general architecture depicted in Fig 3a: a stack of MP layers as in (7) followed by a layer that computes $\mathbf{S}$. The GNNs are trained only by minimizing unsupervised losses, such as those in Eq. 10 (MinCutPool), Eq. 11 (DMoN), and Eq. 17 (TVGNN). The hyperparameters of each model are in B.3.

The methods are tested on 3 citation and 1 collaboration networks (details in B.2). The number of clusters K is set to be equal to the number of vertex classes. Averaged results

[1] https://github.com/FilippoMB/Total-variation-graph-neural-networks

Table 1: NMI and ACC results for vertex clustering. The highest averages are in bold and the second highest are underlined.

Method	Cora		Citeseer		Pubmed		DBLP		Tot. Average	
	NMI	ACC	NMI	ACC	NMI	ACC	NMI	ACC	NMI	ACC
SC	0.029±0.017	29.8±0.7	0.014±0.003	21.7±0.3	0.183±0.000	59.0±0.0	0.023±0.005	45.8±0.2	0.062	39.1
DeepWalk	0.064±0.024	23.0±2.1	0.005±0.001	19.4±0.3	0.001±0.000	36.1±0.1	0.001±0.000	26.7±0.1	0.018	26.3
node2vec	0.060±0.030	23.0±2.5	0.004±0.001	19.5±0.3	0.001±0.000	36.2±0.1	0.001±0.000	27.3±0.1	0.017	26.5
NetMF	0.251±0.000	38.9±0.0	0.127±0.000	27.7±0.0	0.059±0.000	44.8±0.0	0.037±0.000	45.6±0.0	0.119	39.3
TADW	0.012±0.000	19.3±0.0	0.002±0.000	18.8±0.0	0.031±0.000	42.8±0.0	0.012±0.000	29.8±0.0	0.014	27.7
BANE	0.291±0.000	49.5±0.0	0.260±0.000	49.4±0.0	0.121±0.000	50.9±0.0	0.177±0.000	50.6±0.0	0.212	50.1
TENE	0.115±0.000	25.5±0.0	0.005±0.000	20.8±0.0	0.002±0.000	39.9±0.0	0.003±0.000	40.2±0.0	0.125	31.6
GAE	0.328±0.051	46.5±6.2	0.163±0.029	38.1±3.8	0.235±0.044	58.9±7.2	0.111±0.029	41.6±3.5	0.209	46.3
VGAE	0.437±0.029	57.2±5.4	0.156±0.034	36.0±3.9	**0.245**±0.043	**61.1**±6.1	0.213±0.021	50.7±4.7	0.263	51.3
DiffPool	0.307±0.006	47.3±1.0	0.180±0.008	33.6±0.8	0.084±0.002	41.8±0.3	0.045±0.044	37.1±4.3	0.154	40.0
MinCutPool	0.406±0.029	53.4±4.1	0.295±0.029	49.8±4.9	0.209±0.015	57.3±3.5	0.297±0.025	53.8±3.4	0.302	53.6
DMoN	0.357±0.043	48.8±6.4	0.196±0.030	36.4±4.3	0.193±0.049	55.9±4.2	0.335±0.027	59.0±4.0	0.270	50.0
TVGNN	**0.488**±0.016	**63.2**±1.8	**0.361**±0.018	**58.6**±3.0	0.216±0.027	60.0±2.0	**0.342**±0.011	**60.8**±1.5	**0.352**	**60.7**
ABL1	0.376±0.018	48.0±2.0	0.210±0.019	41.1±3.8	0.222±0.014	57.7±3.8	0.260±0.035	52.2±3.3	0.267	49.8
ABL2	0.388±0.024	50.7±4.9	0.285±0.038	49.3±5.2	0.191±0.035	58.5±3.3	0.262±0.044	50.4±4.4	0.282	52.2



(a) DiffPool (b) MinCutPool (c) DMoN (d) TVGNN

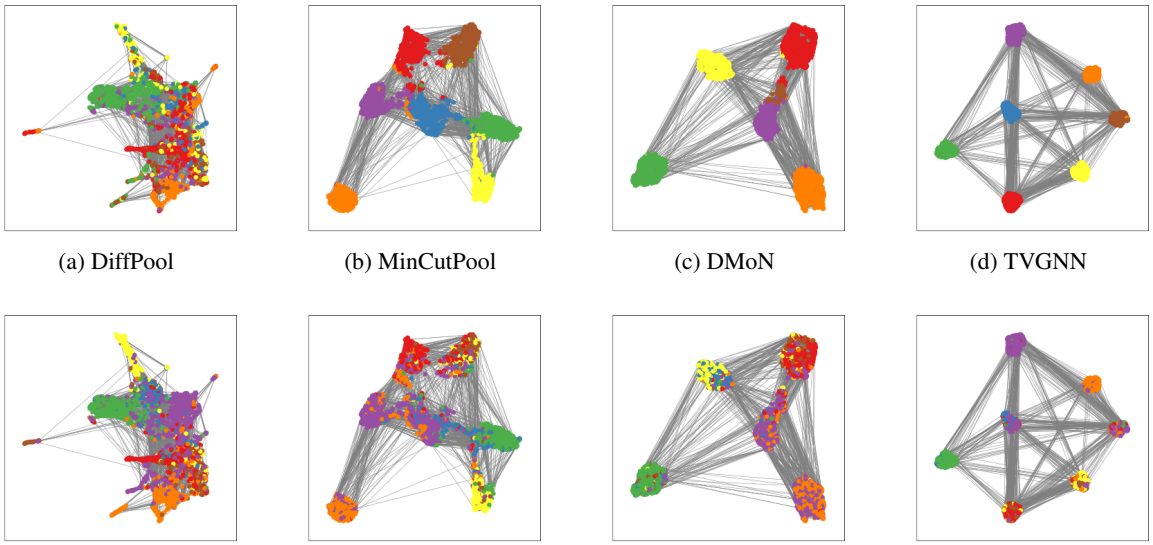
Figure 4: Visualization of the logarithm of $\mathbf{S}\mathbf{S}^T$ for Cora.

Table 2: Graph classification accuracy. The highest mean accuracy for each dataset is in bold, and the second highest is underlined. We report the p-value of the difference between the two highest means (* and ** denote significance at 95% and 99% confidence levels).

Dataset	Top-K	SAGPool	DiffPool	MinCutPool	DMoN	TVGNN	p-value
Bench-easy	53.8±31.8	53.8±31.8	99.0±0.3	99.0±0.3	98.8±0.5	**99.6**±0.6	.011*
Bench-hard	30.5±0.7	29.5±0.0	72.8±0.2	70.9±1.7	71.8±1.9	**75.3**±0.8	.001**
MUTAG	77.5±8.4	76.8±9.7	86.4±7.6	85.2±7.2	86.7±7.0	**88.4**±7.5	.606
Mutagenicity	68.4±8.4	68.2±7.8	78.5±1.5	78.4±1.4	77.1±1.3	**80.0**±1.3	.028*
NCI1	54.0±4.1	59.2±7.7	74.1±1.8	75.2±1.8	74.3±1.3	**77.3**±1.8	.018*
Proteins	69.6±2.7	70.4±2.5	74.6±4.2	75.7±3.0	75.2±3.3	**77.1**±2.9	.302
D&D	62.0±5.6	64.2±7.0	77.7±3.0	78.2±3.4	78.0±3.3	**79.5**±2.2	.323
COLLAB	73.4±6.9	75.6±2.5	78.4±1.6	79.3±1.1	79.5±0.7	**79.8**±1.1	.476
REDDIT-BINARY	54.0±10.0	50.0±0.1	80.9±2.7	82.3±3.2	82.6±2.9	**86.5**±2.8	.007**

from 10 independent runs of each method are in Tab. 1, which reports the Normalized Mutual Information (NMI) and the accuracy (ACC) between the vertex labels and the cluster assignments sorted with the Kuhn-Munkres algorithm. Overall, TVGNN outperforms every other method in terms of both NMI and ACC.

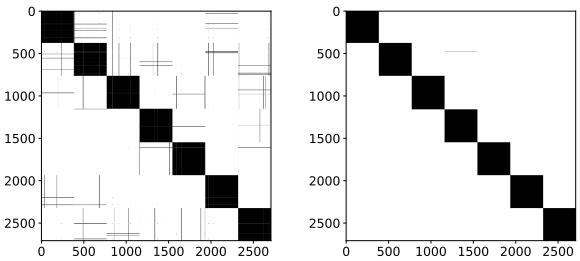
While NMI and ACC quantify of how well the clusters match the true vertex labels, they do not measure the sharpness of the soft cluster assignments $\mathbf{S}$ generated by the GNN-based methods. To evaluate sharpness, we propose the following procedure. Let $\hat{\boldsymbol{y}}$ be the assignments obtained from the Kuhn-Munkres algorithm, which minimizes the mismatch between the discrete cluster assignments $\boldsymbol{c}$ and the



(a) DiffPool (b) MinCutPool (c) DMoN (d) TVGNN

(e) DiffPool (f) MinCutPool (g) DMoN (h) TVGNN

Figure 5: 2D UMAP transform of $\mathbf{X}^{(L)}$ for Cora with the edges from the original adjacency matrix. In (a)-(d) vertex colors correspond to cluster assignments $\hat{\boldsymbol{y}}$ from Kuhn-Munkres and in (e)-(h) colors correspond to the true labels.

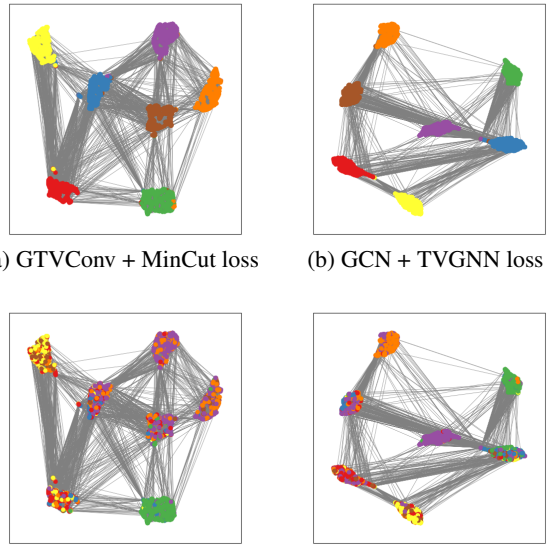


(a) GTVConv + MinCut loss (b) GCN + TVGNN loss

Figure 6: Visualization of the logarithm of $\mathbf{S}\mathbf{S}^T$ for configurations used in the ablation study.

labels $\boldsymbol{y}$. By sorting the rows of $\mathbf{S}$ according to the indices given by $\texttt{argsort}(\hat{\boldsymbol{y}})$, $\mathbf{S}\mathbf{S}^T$ will exhibit a block-diagonal structure if the vertices are assigned with high confidence to only one cluster, i.e., if the cluster assignments are sharp. Instead, if the assignments in $\mathbf{S}$ are smooth, non-zero elements will appear on the off-diagonal of $\mathbf{S}\mathbf{S}^T$. In addition, the size of each block in $\mathbf{S}\mathbf{S}^T$ indicates the cluster size. Fig. 4 shows $\log(\mathbf{S}\mathbf{S}^T)$ for the cluster assignments obtained for the Cora dataset. The soft assignments given by TVGNN are much sharper than those of DiffPool, MinCutPool, and DMoN, since most of the non-zero values lie within the blocks on the diagonal. Notably, the assignments given by DiffPool are so smooth that discerning any structure in $\mathbf{S}\mathbf{S}^T$ is impossible.

The cluster assignments are computed from vertex features



(a) GTVConv + MinCut loss (b) GCN + TVGNN loss

(c) GTVConv + MinCut loss (d) GCN + TVGNN loss

Figure 7: Equivalent plots to Figure 5 for the ablation study configurations. Colors correspond to the cluster assignments in the top row and the true labels in the bottom row.

$\mathbf{X}^{(L)}$ generated by the last MP layer according to (18). To show the separation of the vertex communities before computing the assignments, we project $\mathbf{X}^{(L)}$ in two dimensions using UMAP (McInnes et al., 2018). Fig 5 shows the projected features for Cora. In the first row, the vertices are colored according to the cluster assignments $\hat{\boldsymbol{y}}$ from Kuhn-

Munkres; in the second row, according to the true labels $\boldsymbol{y}$. Compared to the other methods, TVGNN yields clusters that are better separated and more compact. In addition, with TVGNN the class distribution is better aligned with the clustering partition.

Appendices C.1, C.2, and C.3 report additional results and experiments, which further highlight the capabilities of TVGNN in learning sharp and meaningful cluster assignments.

4.2. Ablation experiment

To verify the effectiveness of the proposed TVGNN architecture, which combines the loss function in (17) with the GTVConv layers in (29), we conduct an ablation study with two modified configurations. The numerical results are reported at the bottom of Tab. 1. Here, ABL1 denotes a configuration where the proposed loss is replaced by the MinCutPool loss function in (10). The MinCutPool loss was chosen because it is closely related to the spectral clustering loss, which we aim at improving by basing our loss on the components of a tighter relaxation of the Asymmetric Cheeger cut. On the other hand, ABL2 denotes a configuration where the GTVConv layers, which minimize the GTV, are replaced by GCN layers, which minimize the LQV (4) instead. The plots of $\mathbf{S}\mathbf{S}^T$ and the UMAP transform of the latent representation are in Fig. 6 and 7, respectively. Compared to the proposed model, in both ablation experiments, the clusters are less separated and compact. In addition, there is a worse correspondence between clusters and class labels.

4.3. Supervised graph classification

This task consists in assigning each graph $\mathcal{G}_i$ to a class y_i. We adopt the deep architecture described in Sec. 3.3, where MP layers are interleaved with a pooling layer. As pooling methods, we consider DiffPool, MinCutPool, DMoN, SAGPool (Lee et al., 2019), Top-K (Cangea et al., 2018; Gao & Ji, 2019), and the proposed TVGNN. All GNN architectures follow the same general configuration depicted in Fig. 3b, with the main difference that TVGNN adopts GTVConv rather than standard MP layers (hyperparameters and other details are in Appendix B.3). The unsupervised clustering losses in DiffPool, MinCutPool, DMoN, and TVGNN are computed at each pooling layer and then combined with the supervised cross-entropy loss at the end. The GNNs with SAGPool and Top-K do not have auxiliary losses and are trained only with the cross-entropy.

We consider 9 graph classification datasets (details in B.2). Training and testing are done with a stratified 5-fold train/test split. In addition, 10% of the training set is used as a validation set using a random stratified split. For each fold, we perform 3 independent runs and we train until we reach early stopping by measuring the validation loss. To make the comparison fair, all methods are evaluated on the exact same splits. The classification accuracy for each dataset is reported in Table 2. The GNN models based on TVGNN achieve the highest mean accuracy on all datasets and the differences with the second-best performing method are statistically significant in most of the cases.

5. Conclusions

We introduced a novel graph neural network that significantly improves the performance of previous GNN models based on spectral clustering. To obtain compact and well-separated clusters, we derived an unsupervised loss from the Asymmetric Cheeger cut, which minimizes the graph total variation of the cluster assignments. Remarkably, this is the first attempt to adapt a tighter relaxation of the K-cut problem to neural networks and to apply the Asymmetric Cheeger cut relaxation for clustering vertices of attributed graphs.

To facilitate the minimization of the proposed loss function, we introduced GTVConv, a message-passing layer that updates the vertex features by following the gradient of their graph total variation. The formal derivation demands GTVConv to use a different connectivity matrix to process each one of the vertex features, which is intractable. Therefore, we approximated the gradient descent step of GTV with a single connectivity matrix that accounts for all the vertex features at once, reducing the model complexity and facilitating its training. An appealing property of the GTVConv layer is its capability to adjust the edge weights based on the vertex representations, which are learned in a data-driven fashion. Our extensive experimental evaluation showed that TVGNN outperforms every other competing method in vertex clustering and graph classification tasks. In particular, our model always separates well the vertex features and generates sharp cluster assignments.

In this work, the GTVConv was used in conjunction with the proposed unsupervised loss in GNN architectures for vertex clustering and graph classification. However, GTVConv could also be used as a stand-alone MP layer in GNNs for tasks such as semi-supervised vertex classification.

ACKNOWLEDGEMENTS

The authors gratefully acknowledge the support of Nvidia Corporation with the donation of the RTX A6000 GPUs used to perform the experimental evaluation.

References

Bai, Y., Cheung, G., Liu, X., and Gao, W. Graph-based blind image deblurring from a single photograph. *IEEE Trans-*

actions on Image Processing, 28(3):1404–1418, 2018.

Bianchi, F. M. and Lachi, V. The expressive power of pooling in graph neural networks. *arXiv preprint arXiv:2304.01575*, 2023.

Bianchi, F. M., Grattarola, D., and Alippi, C. Spectral clustering with graph neural networks for graph pooling. In *International Conference on Machine Learning*, pp. 874–883. PMLR, 2020.

Bianchi, F. M., Grattarola, D., Livi, L., and Alippi, C. Graph neural networks with convolutional arma filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

Bianchi, F. M., Gallicchio, C., and Micheli, A. Pyramidal reservoir graph neural network. *Neurocomputing*, 470: 389–404, 2022.

Bo, D., Wang, X., Shi, C., and Shen, H. Beyond low-frequency information in graph convolutional networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 3950–3957, 2021.

Bresson, X., Laurent, T., Uminsky, D., and von Brecht, J. Multiclass total variation clustering. In Burges, C., Bottou, L., Welling, M., Ghahramani, Z., and Weinberger, K. (eds.), *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc., 2013.

Cangea, C., Veličković, P., Jovanović, N., Kipf, T., and Liò, P. Towards sparse hierarchical graph classifiers. *arXiv preprint arXiv:1811.01287*, 2018.

Du, J., Wang, S., Miao, H., and Zhang, J. Multi-channel pooling graph neural networks. In *IJCAI*, pp. 1442–1448, 2021.

Duval, A. and Malliaros, F. Higher-order clustering and pooling for graph neural networks. *arXiv preprint arXiv:2209.03473*, 2022.

Fey, M. and Lenssen, J. E. Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*, 2019.

Fu, G., Zhao, P., and Bian, Y. p-laplacian based graph neural networks. In *International Conference on Machine Learning*, pp. 6878–6917. PMLR, 2022.

Fu, X., Zhang, J., Meng, Z., and King, I. Magnn: Metapath aggregated graph neural network for heterogeneous graph embedding. In *Proceedings of The Web Conference 2020*, pp. 2331–2341, 2020.

Gao, H. and Ji, S. Graph u-nets. In *international conference on machine learning*, pp. 2083–2092. PMLR, 2019.

Grattarola, D. and Alippi, C. Graph neural networks in tensorflow and keras with spektral. *arXiv preprint arXiv:2006.12138*, 2020.

Grattarola, D., Zambon, D., Bianchi, F. M., and Alippi, C. Understanding pooling in graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

Grover, A. and Leskovec, J. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, 2016.

Hamilton, W. L. Graph representation learning. *Synthesis Lectures on Artifical Intelligence and Machine Learning*, 14(3):1–159, 2020.

Hein, M. and Setzer, S. Beyond spectral clustering-tight relaxations of balanced graph cuts. In *NIPS*, pp. 2366–2374. Citeseer, 2011.

Ivanov, S., Sviridov, S., and Burnaev, E. Understanding isomorphism bias in graph data sets. *arXiv preprint arXiv:1910.12091*, 2019.

Kampffmeyer, M., Løkse, S., Bianchi, F. M., Livi, L., Salberg, A.-B., and Jenssen, R. Deep divergence-based approach to clustering. *Neural Networks*, 113:91–101, 2019.

Kipf, T. N. and Welling, M. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*, 2016.

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. *International Conference of Learning Representations (ICLR)*, 2017.

Lee, J., Lee, I., and Kang, J. Self-attention graph pooling. In *International conference on machine learning*, pp. 3734–3743. PMLR, 2019.

Li, Q., Han, Z., and Wu, X.-M. Deeper insights into graph convolutional networks for semi-supervised learning. In *Thirty-Second AAAI conference on artificial intelligence*, 2018.

Liu, X., Jin, W., Ma, Y., Li, Y., Liu, H., Wang, Y., Yan, M., and Tang, J. Elastic graph neural networks. In Meila, M. and Zhang, T. (eds.), *Proceedings of the 38th International Conference on Machine Learning*, volume 139, pp. 6837–6849. PMLR, 18–24 Jul 2021.

Ma, Y., Liu, X., Zhao, T., Liu, Y., Tang, J., and Shah, N. A unified view on graph neural networks as graph signal denoising. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pp. 1202–1211, 2021.

McInnes, L., Healy, J., and Melville, J. Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*, 2018.

Min, E., Guo, X., Liu, Q., Zhang, G., Cui, J., and Long, J. A survey of clustering with deep learning: From the perspective of network architecture. *IEEE Access*, 6: 39501–39514, 2018.

Morris, C., Kriege, N. M., Bause, F., Kersting, K., Mutzel, P., and Neumann, M. Tudataset: A collection of benchmark datasets for learning with graphs. In *ICML 2020 Workshop on Graph Representation Learning and Beyond (GRL+ 2020)*, 2020. URL www.graphlearning.io.

Nt, H. and Maehara, T. Revisiting graph neural networks: All we have is low-pass filters. *arXiv preprint arXiv:1905.09550*, 2019.

Perozzi, B., Al-Rfou, R., and Skiena, S. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2014.

Qiu, J., Dong, Y., Ma, H., Li, J., Wang, K., and Tang, J. Network embedding as matrix factorization: Unifying deepwalk, line, pte, and node2vec. In *Proceedings of the 11th ACM international conference on web search and data mining*, 2018.

Rangapuram, S. S., Mudrakarta, P. K., and Hein, M. Tight continuous relaxation of the balanced k-cut problem. In *NIPS*, pp. 3131–3139, 2014.

Rozemberczki, B., Kiss, O., and Sarkar, R. Karate Club: An API Oriented Open-source Python Framework for Unsupervised Learning on Graphs. In *Proceedings of the 29th ACM International Conference on Information and Knowledge Management (CIKM '20)*, pp. 3125–3132. ACM, 2020.

Sadhanala, V., Wang, Y.-X., and Tibshirani, R. J. Total variation classes beyond 1d: Minimax rates, and the limitations of linear smoothers. *Advances in Neural Information Processing Systems*, 29, 2016.

Shaham, U., Stanton, K., Li, H., Basri, R., Nadler, B., and Kluger, Y. Spectralnet: Spectral clustering using deep neural networks. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=HJ_aoCyRZ.

Su, X., Xue, S., Liu, F., Wu, J., Yang, J., Zhou, C., Hu, W., Paris, C., Nepal, S., Jin, D., et al. A comprehensive survey on community detection with deep learning. *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

Tian, F., Gao, B., Cui, Q., Chen, E., and Liu, T.-Y. Learning deep representations for graph clustering. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 28, 2014.

Tremblay, N., Gonçalves, P., and Borgnat, P. Design of graph filters and filterbanks. In *Cooperative and Graph Signal Processing*, pp. 299–324. Elsevier, 2018.

Tsitsulin, A., Palowitch, J., Perozzi, B., and Müller, E. Graph clustering with graph neural networks. *arXiv preprint arXiv:2006.16904*, 2020.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., and Bengio, Y. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

Von Luxburg, U. A tutorial on spectral clustering. *Statistics and computing*, 17(4):395–416, 2007.

Wang, Y.-X., Sharpnack, J., Smola, A. J., and Tibshirani, R. J. Trend filtering on graphs. *Journal of Machine Learning Research*, 17(105):1–41, 2016. URL http://jmlr.org/papers/v17/15-147.html.

Wu, F., Souza, A., Zhang, T., Fifty, C., Yu, T., and Weinberger, K. Simplifying graph convolutional networks. In *International conference on machine learning*, pp. 6861–6871. PMLR, 2019.

Yang, C., Liu, Z., Zhao, D., Sun, M., and Chang, E. Y. Network representation learning with rich text information. In *International Joint Conference on Artificial Intelligence*, IJCAI'15, pp. 2111–2117. AAAI Press, 2015. ISBN 9781577357384.

Yang, H., Pan, S., Zhang, P., Chen, L., Lian, D., and Zhang, C. Binarized attributed network embedding. In *2018 IEEE International Conference on Data Mining (ICDM)*, pp. 1476–1481. IEEE, 2018.

Yang, S. and Yang, B. Enhanced network embedding with text information. In *2018 24th International Conference on Pattern Recognition (ICPR)*, pp. 326–331. IEEE, 2018.

Yang, Z., Cohen, W. W., and Salakhutdinov, R. Revisiting semi-supervised learning with graph embeddings. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning*, ICML'16, pp. 40–48. JMLR.org, 2016.

Ying, Z., You, J., Morris, C., Ren, X., Hamilton, W., and Leskovec, J. Hierarchical graph representation learning with differentiable pooling. *Advances in neural information processing systems*, 31, 2018.

Zhou, D. and Schölkopf, B. Regularization on discrete spaces. In *Joint Pattern Recognition Symposium*, pp. 361–368. Springer, 2005.

A. Derivations and analyses

A.1. Graph cut in matrix form

To see that the cut between C_k and its conjugate $\bar{C}_k$ can be expressed in matrix form, we first write the cut as

$$\text{cut}(C_k, \bar{C}_k) = \sum_{i \in C_k, j \in \bar{C}_k} a_{ij}(1 - z_i z_j),$$

where $z_i, z_j \in \{-1, 1\}$ are cluster indicators, i.e., $z_i = 1$ if vertex $i \in C_k$ and $z_i = -1$ if $i \notin C_k$.

Then,

$$\begin{aligned}
&\sum_{i \in C_k, j \in \bar{C}_k} a_{ij}(1 - z_i z_j) = \sum_{i \in C_k, j \in \bar{C}_k} a_{ij} \left(\frac{z_i^2 + z_j^2}{2} - z_i z_j \right) \\
&= \frac{1}{2} \sum_{i \in C_k} \left[\sum_{j \in \bar{C}_k} a_{ij} \right] z_i^2 + \frac{1}{2} \sum_{j \in \bar{C}_k} \left[\sum_{i \in C_k} a_{ij} \right] z_j^2 - \sum_{i \in C_k, j \in \bar{C}_k} a_{ij} z_i z_j \\
&= \frac{1}{2} \sum_{i \in C_k} d_{ii} z_i^2 + \frac{1}{2} \sum_{j \in \bar{C}_k} d_{jj} z_j^2 - \boldsymbol{z}^T \mathbf{A} \boldsymbol{z} \\
&= \boldsymbol{z}^T \mathbf{D} \boldsymbol{z} - \boldsymbol{z}^T \mathbf{A} \boldsymbol{z} = \boldsymbol{z}^T \mathbf{L} \boldsymbol{z}.
\end{aligned}$$

A.2. Derivation of the GTVConv aggregation

The graph total variation of a graph with univariate node attributes is defined as

$$||\boldsymbol{x}||_{\text{GTV}} = \sum_{i=1}^{N} \sum_{j=1}^{N} a_{ij} |x_i - x_j|.$$

If the graph has no self-loops ($a_{ii} = 0 \ \ \forall\, i$) and $x_i \neq x_j \ \ \forall\, j \neq i$, the partial derivative of GTV with respect to x_k is

$$\frac{\partial}{\partial x_k}(||\boldsymbol{x}||_{\text{GTV}}) = \sum_{\substack{j=1 \\ j \neq k}}^{N} a_{kj} \frac{x_k - x_j}{|x_k - x_j|} + \sum_{\substack{j=1 \\ j \neq k}}^{N} a_{jk} \frac{x_k - x_j}{|x_j - x_k|}$$

To achieve differentiability at all points, we define an approximated GTV function as

$$(||\boldsymbol{x}||_{\text{GTV}})_\epsilon = \sum_{i=1}^{N} \sum_{j=1}^{N} \left[I(|x_i - x_j| \geq \epsilon) \left(a_{ij} |x_i - x_j| \right) + I(|x_i - x_j| < \epsilon) \left(a_{ij} \frac{(x_i - x_j)^2}{2\epsilon} + \frac{a_{ij}\epsilon}{2} \right) \right], \tag{31}$$

where $I(\cdot)$ is the indicator function. The function in (31) is the one displayed on the left in Fig. 2. Compared to the original GTV function, (31) is smooth in the vicinity of $|x_i - x_j| = 0$. Importantly, this approximation removes the discontinuity in the derivative at $|x_i - x_j| = 0$, which now becomes a piece-wise linear function, as shown on the right in Fig. 2.

By defining the matrix $\mathbf{\Gamma}$ whose (ij)-th entry is

$$\gamma_{ij} = \frac{a_{ij}}{\max\{|x_i - x_j|, \epsilon\}},$$

the partial derivative of the approximate GTV can be expressed as

$$\begin{aligned}
\frac{\partial}{\partial x_k}(||\boldsymbol{x}||_{\text{GTV}})_\epsilon &= \sum_{j=1}^{N}\gamma_{kj}(x_k - x_j) + \sum_{j=1}^{N}\gamma_{jk}(x_k - x_j) \\
&= x_k\sum_{j=1}^{N}\gamma_{kj} - \sum_{j=1}^{N}\gamma_{kj}x_j + x_k\sum_{j=1}^{N}\gamma_{jk} - \sum_{j=1}^{N}\gamma_{jk}x_j \\
&= x_k d_k - \sum_{j=1}^{N}\gamma_{kj}x_j + x_k d_k^* - \sum_{j=1}^{N}\gamma_{jk}x_j, \quad \text{where } d_k = \sum_j \gamma_{kj}, \;\; d_k^* = \sum_j \gamma_{jk} \\
&= \mathbf{D}_{\mathbf{\Gamma}_k}\boldsymbol{x} - \mathbf{\Gamma}_k\boldsymbol{x} + \mathbf{D}^*_{\mathbf{\Gamma}_k}\boldsymbol{x} - (\mathbf{\Gamma}^T)_k\boldsymbol{x},
\end{aligned}$$

where $\mathbf{\Gamma}_k$ and $\mathbf{D}_{\mathbf{\Gamma}_k}$ represent the k-th row of $\mathbf{\Gamma}$ and $\mathbf{D}_{\mathbf{\Gamma}} = \text{diag}(\mathbf{\Gamma}\mathbf{1})$, respectively. The full gradient of the approximated GTV of $\boldsymbol{x}$ is furthermore

$$\nabla(||\boldsymbol{x}||_{\text{GTV}})_\epsilon = \mathbf{D}_{\mathbf{\Gamma}}\boldsymbol{x} - \mathbf{\Gamma}\boldsymbol{x} + \mathbf{D}^*_{\mathbf{\Gamma}}\boldsymbol{x} - \mathbf{\Gamma}^T\boldsymbol{x}.$$

If the graph is undirected, i.e., $\gamma_{ij} = \gamma_{ji}$, the gradient can be simplified as

$$\nabla(||\boldsymbol{x}||_{\text{GTV}})_\epsilon = 2(\mathbf{D}_{\mathbf{\Gamma}} - \mathbf{\Gamma})\boldsymbol{x} = 2\mathbf{L}_{\mathbf{\Gamma}}\boldsymbol{x},$$

where $\mathbf{L}_{\mathbf{\Gamma}}$ is the Laplacian associated with $\mathbf{\Gamma}$. Finally, the gradient descent step update of the approximated GTV of $\boldsymbol{x}$ with step size δ becomes

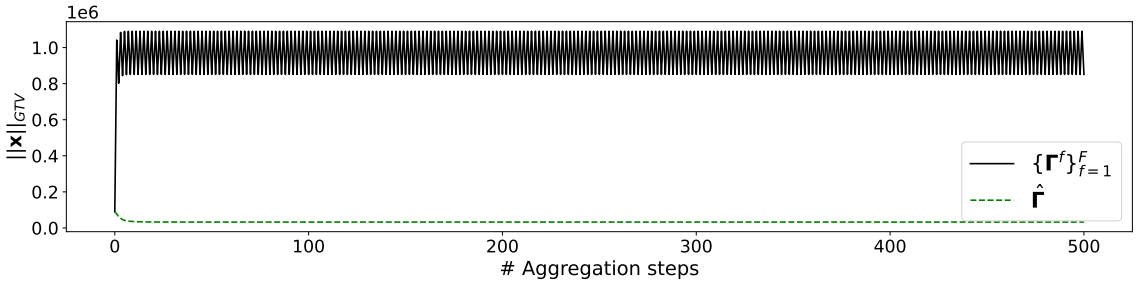
$$\begin{aligned}
\boldsymbol{x}^{(t+1)} &= \boldsymbol{x}^{(t)} - \delta\nabla(||\boldsymbol{x}||_{\text{GTV}})_\epsilon^{(t)} \\
&= \boldsymbol{x}^{(t)} - 2\delta\mathbf{L}_{\mathbf{\Gamma}}^{(t)}\boldsymbol{x}^{(t)} \\
&= \left(\mathbf{I} - 2\delta\mathbf{L}_{\mathbf{\Gamma}}^{(t)}\right)\boldsymbol{x}^{(t)}.
\end{aligned}$$

A.3. Replacing multiple $\mathbf{\Gamma}^f$ with a single $\hat{\mathbf{\Gamma}}$

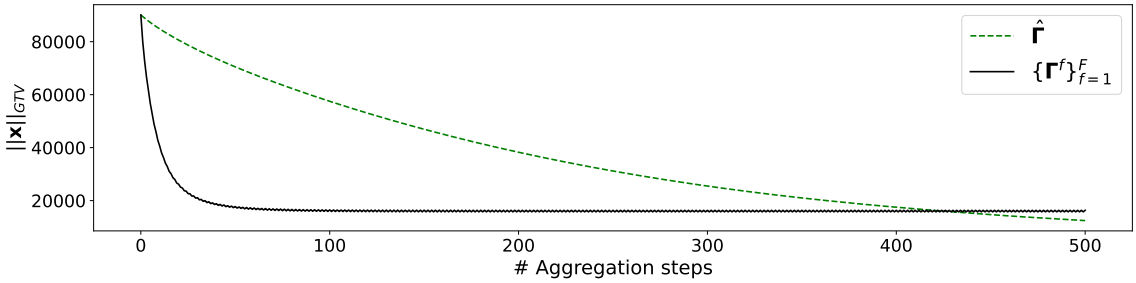
As discussed in Sec. 3.2 and according to (24), in the presence of multiple vertex features a different connectivity matrix $\mathbf{\Gamma}^f, f = 1, \ldots, F$ must be used to independently aggregate each feature f. We propose to simplify the procedure, by aggregating all features using the single connectivity matrix $\hat{\mathbf{\Gamma}}$ defined in (26). The proposed simplification offers two important advantages. The first and more obvious involves computational complexity: only one connectivity matrix needs to be stored and a single aggregation procedure must be performed. A second advantage is that using $\hat{\mathbf{\Gamma}}$ facilitates the minimization of the GTV across the vertex features. We illustrate this second point with an example.

In the following, we consider the Cora dataset but the same behavior is observed also for the other datasets. To reproduce a typical GNN setting, the features $\mathbf{X}^{(0)}$ are constructed by mapping the original one-hot encoded features into a 32-dimensional space through the multiplication with a kernel, whose values are randomly sampled from a standard normal distribution. Then, we apply several times the aggregation defined in equations (24) and (26). Fig. 8 depicts how the GTV of the vertex features, $\|\mathbf{X}\|_{\text{GTV}}$, decreases when the vertex features are aggregated using multiple matrices $\{\mathbf{\Gamma}^f\}_{f=1}^F$ (black line) or a single connectivity matrix $\hat{\mathbf{\Gamma}}$ (green dashed line). In the first plot, Fig. 8(a), we use gradient step $2\delta = 0.311$, which is the same used in the vertex clustering experiment. In Fig. 8(b) we set $2\delta = 0.005$. First, we notice that for the larger gradient step, the aggregation based on multiple connectivity matrices performs poorly as the minimization gets stuck oscillating around some local minima. This can happen because the gradients of the GTV for each feature f are pulling toward different directions. Decreasing step size (Fig. 8(b)), which corresponds to lowering the contribution from neighbors when aggregating, makes the scheme based on multiple $\mathbf{\Gamma}^f$ more stable. Nevertheless, even in this case the aggregation scheme based on the single $\hat{\mathbf{\Gamma}}$ matrix eventually converges to a better solution.

In conclusion, when using $\hat{\mathbf{\Gamma}}$ the optimization of GTV is generally more stable, allows to find better minima, and is more robust to the selection of the gradient step δ.



(a)



(b)

Figure 8: Evolution of GTV for the vertex features of Cora with respect to the number of aggregation steps when using a connectivity matrix $\mathbf{\Gamma}^f$ for each feature or a single connectivity matrix $\hat{\mathbf{\Gamma}}$. In (a), we use gradient step $2\delta = 0.311$; In (b), we use gradient step $2\delta = 0.005$.

A.4. Weighted LQV and GTV

Let us define the LQV weighted by the vertex degrees as

$$||\boldsymbol{x}||_{\mathrm{LQV_W}} = \frac{1}{4}\sum_{i=1}^{N}\sum_{j=1}^{N}\left(\sqrt{\frac{a_{ij}}{d_i}}x_i - \sqrt{\frac{a_{ij}}{d_j}}x_j\right)^2 = \frac{1}{4}\sum_{i=1}^{N}\sum_{j=1}^{N}a_{ij}\left(\frac{x_i}{\sqrt{d_i}} - \frac{x_j}{\sqrt{d_j}}\right)^2.$$

Then, under the assumption that the graph is undirected, we have that

$$\begin{aligned}
\frac{\partial}{\partial x_k}\left(||\boldsymbol{x}||_{\mathrm{LQV_W}}\right) &= \frac{1}{\sqrt{d_k}}\sum_j a_{kj}\left(\frac{x_k}{\sqrt{d_k}} - \frac{x_j}{\sqrt{d_j}}\right) \\
&= \frac{x_k}{d_k}\sum_j a_{kj} - \sum_j \frac{a_{kj}}{\sqrt{d_k}\sqrt{d_j}}x_j \\
&= (\mathbf{I}\boldsymbol{x})_k - (\mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}\boldsymbol{x})_k \\
\nabla\left(||\boldsymbol{x}||_{\mathrm{LQV_W}}\right) &= (\mathbf{I} - \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2})\boldsymbol{x},
\end{aligned}$$

where $(\mathbf{I} - \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2})$ is the symmetrically normalized Laplacian.

If we compute the gradient descent update we obtain

$$\begin{aligned}
\boldsymbol{x}^{(t+1)} &= \boldsymbol{x}^{(t)} - \delta\nabla\left(||\boldsymbol{x}||_{\mathrm{LQV_W}}\right)^{(t)} \\
&= \boldsymbol{x}^{(t)} - \delta(\mathbf{I} - \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2})\boldsymbol{x}^{(t)}.
\end{aligned}$$

When the gradient step size is $\delta = 1$, we get

$$\boldsymbol{x}^{(t+1)} = \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}\boldsymbol{x}^{(t)},$$

which closely resembles the aggregation function used by a GCN to update the vertex features.

Now, let us define the degree-weighted GTV as

$$||\boldsymbol{x}||_{\text{GTV}_\text{w}} = \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\left|\sqrt{\frac{a_{ij}}{d_i}}x_i - \sqrt{\frac{a_{ij}}{d_j}}x_j\right| = \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}|q_{ij}|,$$

with $q_{ij} = \sqrt{\frac{a_{ij}}{d_i}}x_i - \sqrt{\frac{a_{ij}}{d_j}}x_j$ and $d_i = \mathbf{D}_{ii}$, where $\mathbf{D} = \text{diag}(\mathbf{A}\mathbf{1})$. The numerically stable approximation to the derivative of q_{kj} w.r.t. x_k is then given by

$$\frac{\partial}{\partial x_k}(q_{kj})_\epsilon = \begin{cases} \sqrt{\frac{a_{kj}}{d_k}} \cdot \frac{q_{kj}}{|q_{kj}|}, & |q_{kj}| \geq \epsilon \\ \sqrt{\frac{a_{kj}}{d_k}} \cdot \frac{q_{kj}}{\epsilon}, & |q_{kj}| < \epsilon \end{cases}$$

which can be rewritten in a more compact form as

$$\frac{\partial}{\partial x_k}(q_{kj})_\epsilon = \sum_j \sqrt{\frac{a_{kj}}{d_k}}\frac{q_{kj}}{\max\{\epsilon, |q_{kj}|\}} = \sum_j \frac{a_{kj}}{\sqrt{d_k}}\frac{\frac{x_k}{\sqrt{d_k}} - \frac{x_j}{\sqrt{d_j}}}{\max\{\epsilon, |q_{kj}|\}}.$$

Now let

$$\gamma_{ij} = \frac{a_{ij}}{\max\{\epsilon, |q_{ij}|\}}$$

By substituting γ_{kj} and assuming, once again, that the graph is undirected we get

$$\begin{aligned}
\frac{\partial}{\partial x_k}\left(|\boldsymbol{x}||_{\text{GTV}_\text{w}}\right)_\epsilon &= \sum_j \frac{1}{\sqrt{d_k}}\gamma_{kj}\left(\frac{x_k}{\sqrt{d_k}} - \frac{x_j}{\sqrt{d_j}}\right) \\
&= \frac{x_k}{d_k}\sum_j \gamma_{kj} - \sum_j \frac{\gamma_{kj}}{\sqrt{d_k}\sqrt{d_j}}x_j \\
&= (\mathbf{D}^{-1}\mathbf{D}_\Gamma)_i\,\boldsymbol{x} - (\mathbf{D}^{-1/2}\boldsymbol{\Gamma}\mathbf{D}^{-1/2})_i\,\boldsymbol{x} \\
&= (\mathbf{D}^{-1/2}\mathbf{D}_{\boldsymbol{\Gamma}}\mathbf{D}^{-1/2})_i\,\boldsymbol{x} - (\mathbf{D}^{-1/2}\boldsymbol{\Gamma}\mathbf{D}^{-1/2})_i\,\boldsymbol{x},
\end{aligned}$$

where $\mathbf{D}_\Gamma = \text{diag}(\boldsymbol{\Gamma}\mathbf{1})$.

The gradient of the weighted GTV is

$$\begin{aligned}
\nabla\left(||\boldsymbol{x}||_{\text{GTV}_\text{w}}\right) &= \mathbf{D}^{-1/2}\mathbf{D}_{\boldsymbol{\Gamma}}\mathbf{D}^{-1/2}\boldsymbol{x} - \mathbf{D}^{-1/2}\boldsymbol{\Gamma}\mathbf{D}^{-1/2}\boldsymbol{x} \\
&= \mathbf{D}^{-1/2}(\mathbf{D}_{\boldsymbol{\Gamma}} - \boldsymbol{\Gamma})\mathbf{D}^{-1/2}\boldsymbol{x} \\
&= \mathbf{D}^{-1/2}\mathbf{L}_{\boldsymbol{\Gamma}}\mathbf{D}^{-1/2}\boldsymbol{x}
\end{aligned}$$

A gradient descent step for minimizing the weighted GTV is given by

$$\begin{aligned}
\boldsymbol{x}^{(t+1)} &= \boldsymbol{x}^{(t)} - \delta\nabla\left(||\boldsymbol{x}||_{\text{GTV}_\text{w}}\right)^{(t)} \\
&= \boldsymbol{x}^{(t)} - \delta\mathbf{D}^{-1/2}\mathbf{L}_{\boldsymbol{\Gamma}}^{(t)}\mathbf{D}^{-1/2}\boldsymbol{x}^{(t)} \\
&= (\mathbf{I} - \delta\mathbf{D}^{-1/2}\mathbf{L}_{\boldsymbol{\Gamma}}^{(t)}\mathbf{D}^{-1/2})\boldsymbol{x}^{(t)}
\end{aligned}$$

which is equivalent to the aggregation step of a GCN with $\tilde{\mathbf{A}} = \mathbf{I} - \delta \mathbf{D}^{-1/2} \mathbf{L}_{\mathbf{\Gamma}}^{(t)} \mathbf{D}^{-1/2}$.

We notice the analogy between the updates derived from the degree-weighted LQV, which is $\boldsymbol{x}^{(t+1)} = \boldsymbol{x}^{(t)} - \delta(\mathbf{I} - \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2})\boldsymbol{x}^{(t)}$, and from the degree-weighted GTV, which is $\boldsymbol{x}^{(t+1)} = (\mathbf{I} - \delta \mathbf{D}^{-1/2} \mathbf{L}_{\mathbf{\Gamma}}^{(t)} \mathbf{D}^{-1/2})\boldsymbol{x}^{(t)}$.

B. Details of the experimental setting

B.1. Software libraries

The GNN models were implemented using both Spektral[2] (Grattarola & Alippi, 2020) and Pytorch Geometric[3] (Fey & Lenssen, 2019). The methods for vertex embedding used in the vertex clustering experiment are based on the Karateclub[4] (Rozemberczki et al., 2020) implementation, and most of the datasets are taken from Pytorch Geometric.

B.2. Datasets details

Table 3: Details of the vertex clustering datasets.

Dataset	#Vertices	#Edges	#Vertex attr.	#Vertex classes
Cora	2,708	10,556	1,433	7
Citeseer	3,327	9,104	3,703	6
Pubmed	19,717	88,648	500	3
DBLP	17,716	105,734	1,639	4

Table 4: Details of the graph classification datasets.

Dataset	#Samples	#Classes	Avg. #vertices	Avg. #edges	Vertex attr.	Vertex labels
Bench-easy	1,800	3	147.82	922.67	–	yes
Bench-hard	1,800	3	148.32	572.32	–	yes
MUTAG	188	2	17.93	19.79	–	yes
Mutagenicity	4,337	2	30.32	61.54	–	yes
NCI1	4,110	2	29.87	64.60	–	yes
Proteins	1,113	2	39.06	72.82	1	yes
D&D	1,178	2	284.32	1,431.32	–	yes
COLLAB	5,000	3	74.49	4,914.43	–	no
REDDIT-BINARY	2,000	2	429.63	995.51	–	no

In the vertex clustering experiment, we considered the citation networks Cora, Pubmed, Citeseer (Yang et al., 2016) and DBLP (Fu et al., 2020). In the graph classification experiment, we analyzed seven TUD datasets (Morris et al., 2020) and two synthetic datasets, Bench-easy and Bench-hard (Bianchi et al., 2022).

Details about the datasets are reported in Tab. 3 and 4. In the graph classification datasets, the vertex feature matrix $\mathbf{X}$ consists of vertex attributes, vertex labels, or a concatenation of both. For the datasets where neither the vertex attributes nor the vertex labels are available, a one-hot encoded vertex degree matrix was used as a surrogate feature for $\mathbf{X}$. Furthermore, motivated by the work of (Ivanov et al., 2019), the datasets were cleaned such that they only contained non-isomorphic graphs.

B.3. Hyperparameters configuration

The hyperparameters for TVGNN for both the vertex clustering and graph classification tasks are reported in Tab. 5. The parameter ϵ which ensures numerical stability for $\mathbf{\Gamma}$ was set to 1e-3 in all experiments. For MinCutPool (Bianchi et al., 2020), GAE and VGAE (Kipf & Welling, 2016) the model configurations are those reported in the original papers. For DiffPool, Top-K, and SAGPool configurations were the same as in (Bianchi et al., 2020). The models with DMoN used

[2] https://graphneural.network
[3] https://pytorch-geometric.readthedocs.io
[4] https://karateclub.readthedocs.io

Table 5: Hyperparameters configuration for the vertex clustering and graph classification tasks. σ_{MP} indicates the activation of the MP layers, σ_{MLP} is the activation of the MLP layers, δ is the step-size in the GTVConv layer, α_1 is the coefficient for the total variation loss $\mathcal{L}_{\text{GTV}}$, α_2 is the coefficient for the balance loss $\mathcal{L}_{\text{AN}}$, ℓ_2 indicates the weight of the ℓ_2 regularization on the GNN weight parameters. The values of # MP layers and # MLP layers are the numbers of MP and MLP layers, respectively, in each of the blocks of the architectures for clustering and classification presented in Section 3.3. For example, the architecture used for vertex clustering uses 2×1 MP and 1×1 MLP layers, while the architecture used in Proteins uses 3×3 MP and 3×1 MLP layers.

Parameters	**Vertex Clustering**	**Bench-easy**	**Bench-hard**	**MUTAG**	**Mutagenicity**
# MP layers	2	1	1	1	3
# MP channels	512	32	32	32	32
σ_{MP}	ELU	ReLU	ReLU	ELU	ReLU
2δ	0.311	0.724	2.288	1.644	3.077
# MLP layers	1	3	1	3	2
# MLP channels	256	64	64	64	32
σ_{MLP}	ReLU	ReLU	ReLU	ReLU	ELU
α_1	0.785	0.594	0.188	0.623	0.726
α_2	0.514	0.974	0.737	0.832	0.982
ℓ_2	–	1e-5	0	1e-4	1e-5
Learning rate	1e-3	1e-3	5e-4	1e-2	5e-4

Parameters	**NCI1**	**Proteins**	**D&D**	**COLLAB**	**REDDIT-BINARY**
# MP layers	3	3	1	2	3
# MP channels	32	64	64	256	16
σ_{MP}	ReLU	ELU	ELU	ReLU	ReLU
2δ	2.411	2.073	0.622	0.554	1.896
# MLP layers	3	1	2	2	3
# MLP channels	32	16	32	128	16
σ_{MLP}	ReLU	ELU	ReLU	ReLU	ReLU
α_1	0.936	0.985	0.354	0.304	0.654
α_2	0.639	0.751	0.323	0.801	0.962
ℓ_2	1e-3	1e-3	1e-5	0	1e-3
Learning rate	5e-4	1e-3	1e-5	5e-5	1e-3

the same hyperparameter configuration as for MinCutPool and the regularization term $\mathcal{L}_r$ in the auxiliary loss is weighted by 1e-1. In the case of DeepWalk, node2vec, NetMF, and TADW we used the default configurations from the Karateclub library (Rozemberczki et al., 2020).

For the graph classification task, the GNNs with TVGNN, MinCutPool, DiffPool, and DMoN were trained with a batch size of 8 for all datasets, except D&D and REDDIT-BINARY, for which batch size was set to 1 due to memory constraints. The models with Top-K and SAGPool were trained with a batch size of 1 for all datasets. For the vertex clustering task, the GNNs were trained for 10,000 epochs. In the graph classification task, we performed early stopping on the validation set using patience of 20 epochs.

B.4. Training times

Table 6 and Table 7 report training times for the GNN-based methods for the vertex clustering task and graph classification task, respectively. All methods have been given similar capacities in terms of the number of layers and the size of the weight matrices. The reported times give a rough indication of the differences in the computational complexity, but they highly depend on how optimized is their implementation. Here, the different pooling methods were implemented using Pytorch Geometric (Fey & Lenssen, 2019) and, thus, all methods besides Top-K and SAGPool process dense representations of the graphs. Overall, we notice no significant differences between the execution time of TVGNN and the other methods.

Table 6: Training times in milliseconds per epoch for the vertex clustering task.

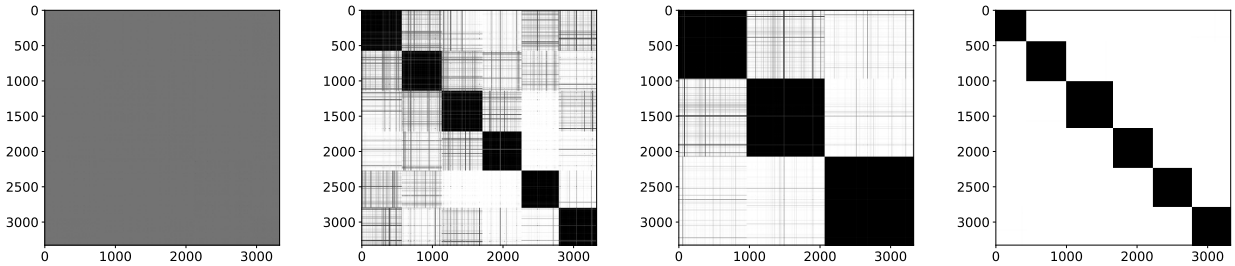
Method	Cora	Citeseer	Pubmed	DBLP
DiffPool	5.6	7.7	67.6	62.2
MinCutPool	8.9	13.8	272.5	227.4
DMoN	5.7	7.8	39.9	39.7
TVGNN	8.3	9.4	56.4	57.2

Table 7: Training times in seconds per epoch for the graph classification task.

Dataset	Top-K	SAGPool	DiffPool	MinCutPool	DMoN	TVGNN
Bench-easy	0.39	0.39	0.39	0.39	0.41	0.49
Bench-hard	0.37	0.37	0.34	0.38	0.37	0.41
MUTAG	0.14	0.10	0.09	0.10	0.09	0.10
Mutagenicity	0.72	0.81	0.71	0.69	0.74	0.85
NCI1	0.69	0.70	0.63	0.70	0.71	0.76
Proteins	0.28	0.23	0.21	0.24	0.24	0.25
D&D	0.32	0.34	0.61	0.63	0.45	0.50
COLLAB	1.03	1.18	0.99	1.12	1.11	1.52
REDDIT-BINARY	0.51	0.54	1.83	1.86	1.10	1.38

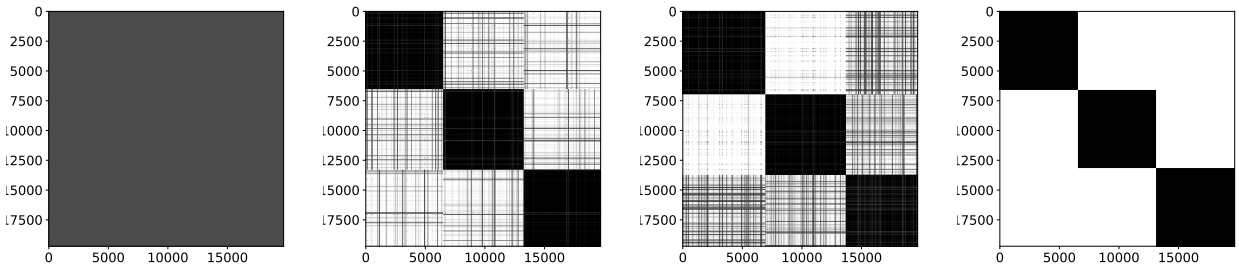
C. Additional results

C.1. Additional plots



(a) DiffPool (b) MinCutPool (c) DMoN (d) TVGNN

Figure 9: Logarithm of $\mathbf{S}\mathbf{S}^T$ for Citeseer.

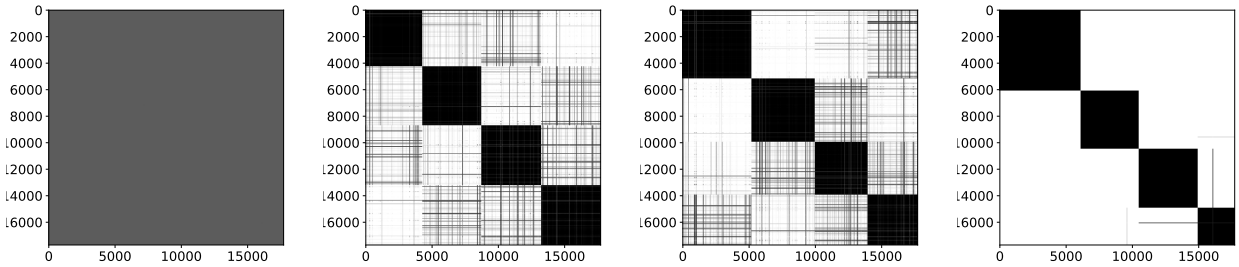


(a) DiffPool (b) MinCutPool (c) DMoN (d) TVGNN

Figure 10: Logarithm of $\mathbf{S}\mathbf{S}^T$ for Pubmed.

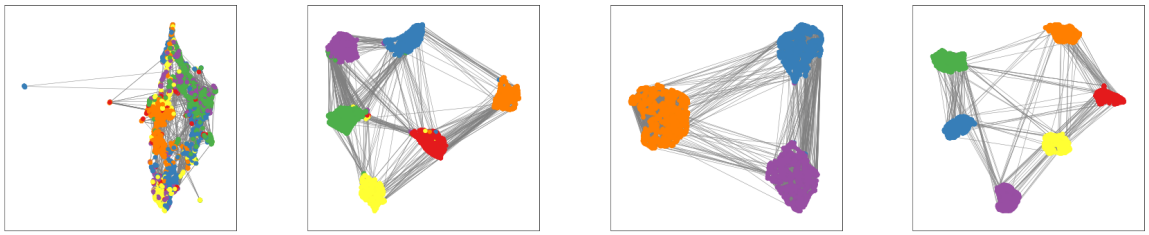
Plots with the logarithm of $\mathbf{S}\mathbf{S}^T$ for Citeseer, Pubmed, and DBLP are presented in Fig. 9, Fig. 10, and Fig. 11 respectively. The UMAP transform of $\mathbf{X}^{(L)}$ for Citeseer, Pubmed, and DBLP are presented in Fig. 12, Fig. 13, and Fig. 14 respectively.

As for the case of Cora, TVGNN manages to give better-separated clusters with sharper assignments for all three graphs when compared to the other three GNN-based clustering methods that produce soft assignments. These plots also show that

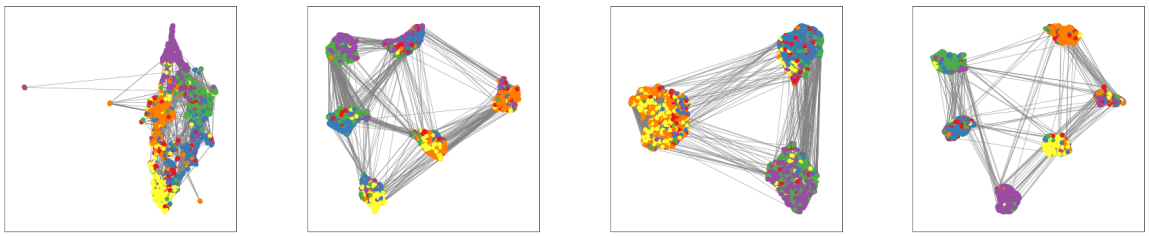


(a) DiffPool (b) MinCutPool (c) DMoN (d) TVGNN

Figure 11: Logarithm of $\mathbf{S}\mathbf{S}^T$ for DBLP.

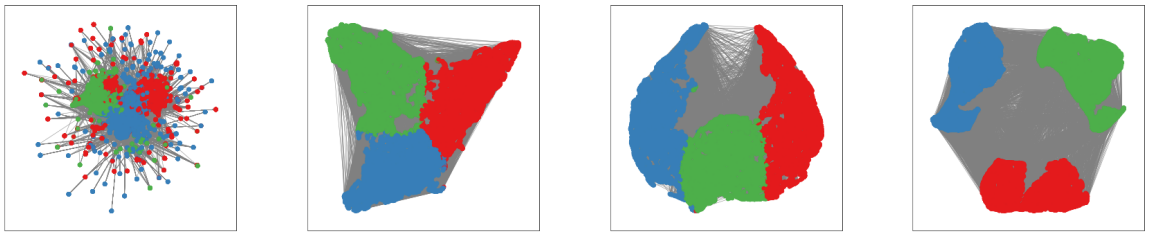


(a) DiffPool (b) MinCutPool (c) DMoN (d) TVGNN

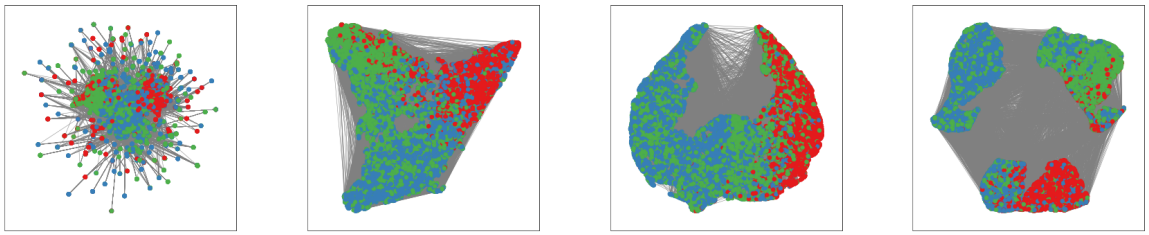


(e) DiffPool (f) MinCutPool (g) DMoN (h) TVGNN

Figure 12: UMAP transforms of $\mathbf{X}^{(L)}$ for Citeseer. Colors in the top row of each dataset correspond to cluster assignments, while the colors in the bottom row correspond to true labels.

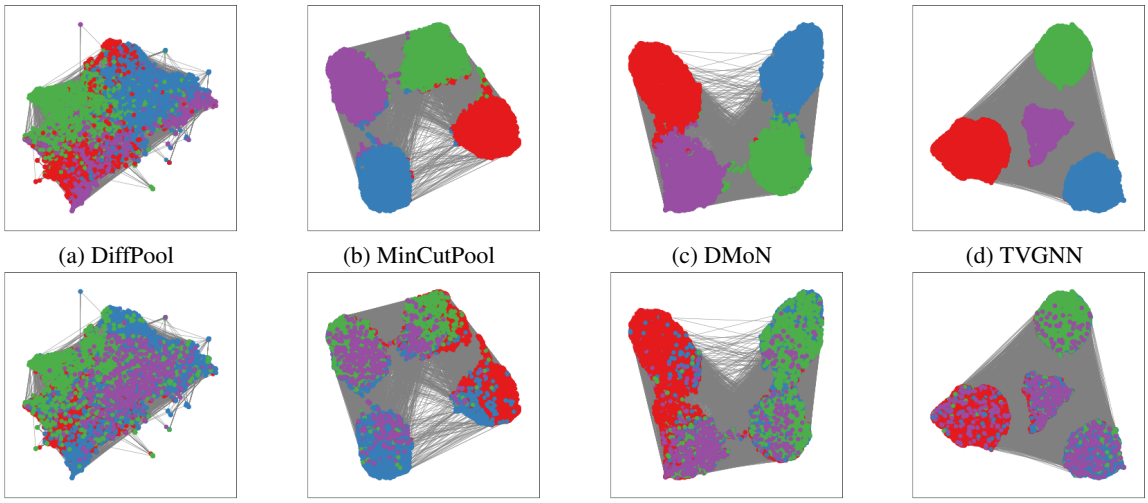


(a) DiffPool (b) MinCutPool (c) DMoN (d) TVGNN



(e) DiffPool (f) MinCutPool (g) DMoN (h) TVGNN

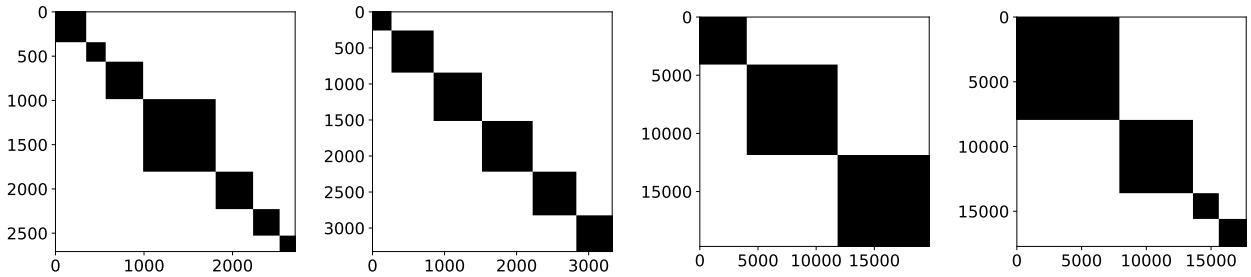
Figure 13: UMAP transforms of $\mathbf{X}^{(L)}$ for Pubmed. Colors in the top row of each dataset correspond to cluster assignments, while the colors in the bottom row correspond to true labels.



(a) DiffPool (b) MinCutPool (c) DMoN (d) TVGNN

(e) DiffPool (f) MinCutPool (g) DMoN (h) TVGNN

Figure 14: UMAP transforms of $\mathbf{X}^{(L)}$ for DBLP. Colors in the top row of each dataset correspond to cluster assignments, while the colors in the bottom row correspond to true labels.



(a) Cora (b) Citeseer (c) Pubmed (d) DBLP

Figure 15: Logarithm of $\mathbf{S}\mathbf{S}^T$ for the true labels of each dataset.

the cluster distribution given by TVGNN is not always balanced, see for instance Fig. 11. In fact, with respect to the true labels, all four datasets are imbalanced, which can be seen from Fig. 15.

The plots of $\mathbf{S}\mathbf{S}^T$ for the configurations used in the ablation study are presented in Fig. 16, while the UMAP plots for Citeseer, Pubmed, and DBLP are presented in Fig. 17, Fig. 18, and Fig. 19, respectively.

C.2. Denoising task

For this task, we generated a Stochastic-Block Model graph with three communities. The graph has 200 nodes, the probability of having a within-community edge is set to 0.3, and the probability of having an edge between communities is 0.005. We assigned vertex features $\boldsymbol{x}_1 = [1, 0, 0]$, $\boldsymbol{x}_2 = [0, 1, 0]$, and $\boldsymbol{x}_3 = [0, 0, 1]$ to the vertices of the first, second, and third community, respectively. Having three communities, allows us to use a convenient RGB color coding to visualize the vertex features. Next, we corrupted the features by adding Gaussian noise from $\mathcal{N}(0, 1.5)$. Afterward, we performed vertex clustering with Diffpool, MinCut, DMoN, and TVGNN. The results are shown in Fig. 20. Note that in panels (a-b), colors indicate node features $\mathbf{X}$, while in (c-f) the colors indicate the cluster assignments $\mathbf{S}$. As we can see, TVGNN manages to perfectly recover the original clusters, while the other methods do not. As for the other experiments, in Fig. 21 we visualize the matrix $\mathbf{S}\mathbf{S}^T$ which shows that TVGNN generates cluster assignments that are much sharper than those produced by the other GNN-based clustering methods.

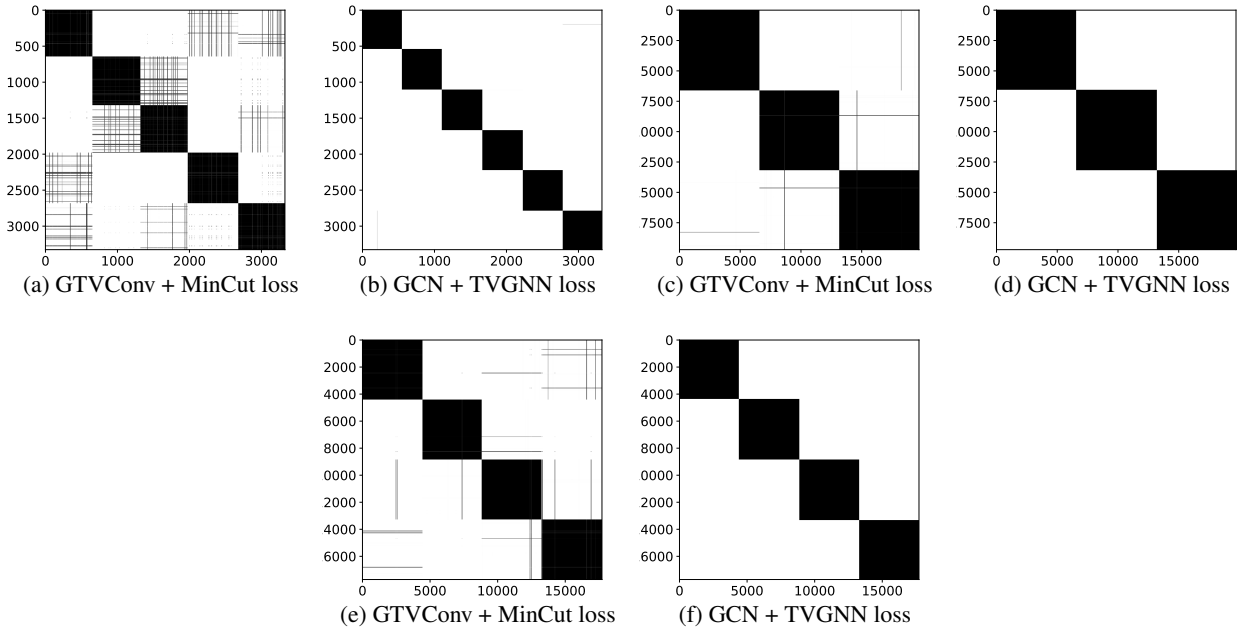


Figure 16: Logarithm of $\mathbf{S}\mathbf{S}^T$ for the ablation study on Citeseer (a, b), Pubmed (c, d), and DBLP (e, f).

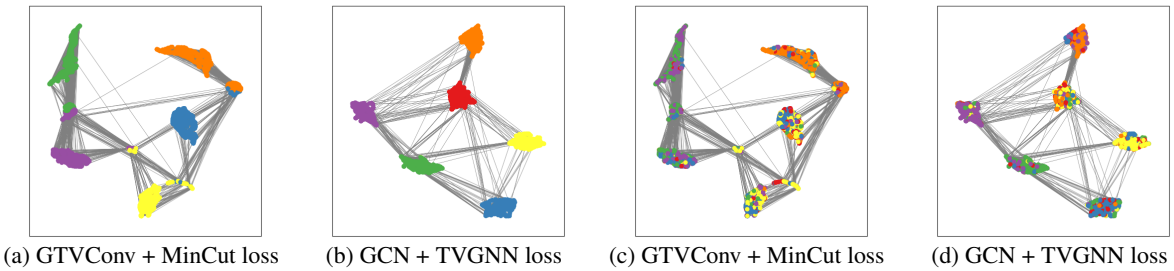


Figure 17: UMAP plots for the ablation study on Citeseer.

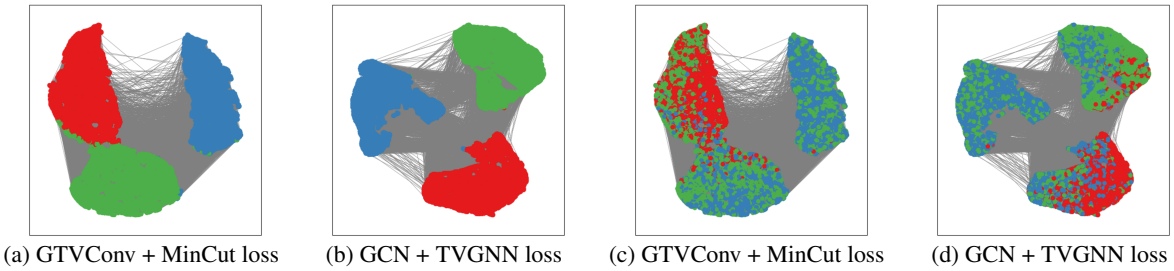
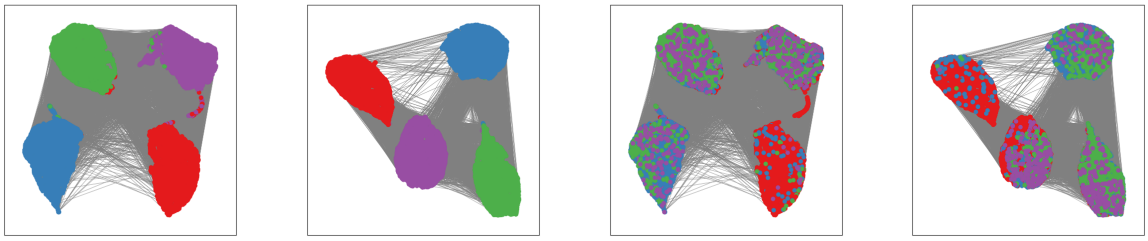


Figure 18: UMAP plots for the ablation study on Pubmed.

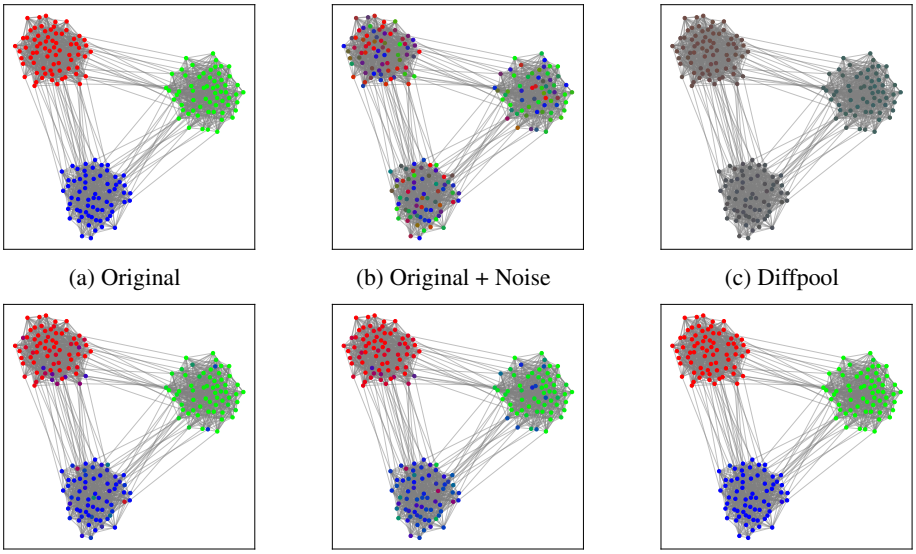
C.3. Clustering two simple point clouds

Fig. 22 and 23 show the largest soft assignment for each node when tasked with clustering a 2D ring graph and a 2D grid graph, respectively. The color of the node is chosen such that a sharp assignment of 1 gives a bright color (lightness equal to 0.5), while smoother assignments give paler colors, and an assignment of 0 is just white (lightness equal to 1). The number of desired clusters K for the ring and grid was 5 and 10, respectively. The models were trained using the same



(a) GTVConv + MinCut loss (b) GCN + TVGNN loss (c) GTVConv + MinCut loss (d) GCN + TVGNN loss

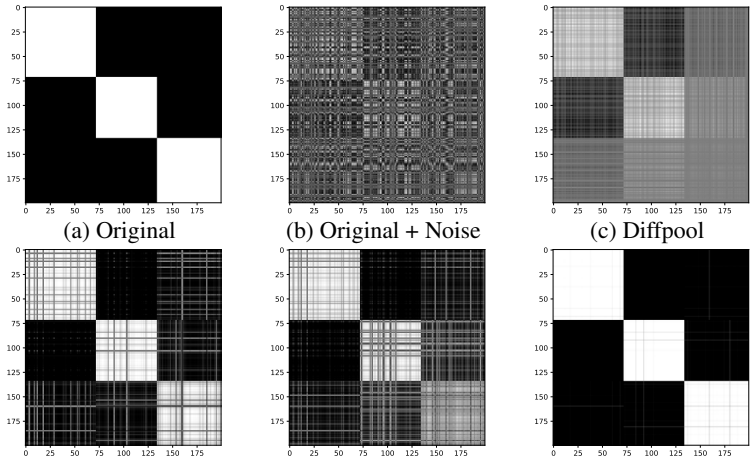
Figure 19: UMAP plots for the ablation study on DBLP.



(a) Original (b) Original + Noise (c) Diffpool

(d) MinCutPool (e) DMoN (f) TVGNN

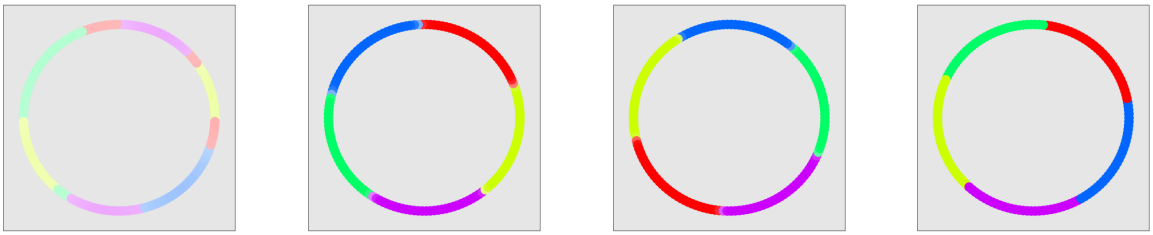
Figure 20: Denoising task. (a) the original vertex features; (b) the vertex features corrupted with Gaussian noise; (c-f) cluster labels identified by each method.



(a) Original (b) Original + Noise (c) Diffpool

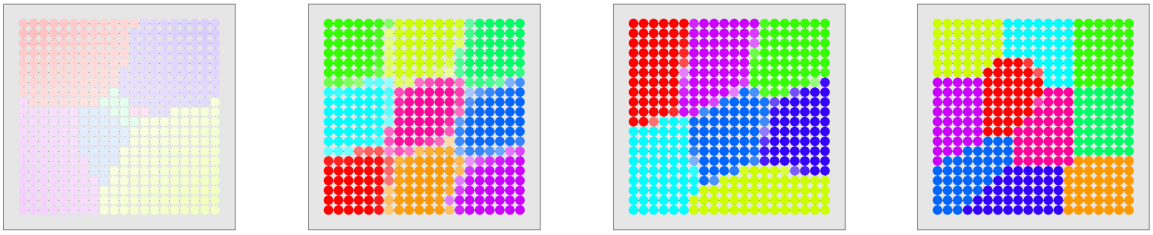
(d) MinCutPool (e) DMoN (f) TVGNN

Figure 21: Visualization of $\mathbf{S}\mathbf{S}^T$ for the denoising task. (a) the original vertex features; (b) the vertex features corrupted with Gaussian noise; (c-f) cluster labels identified by each method.



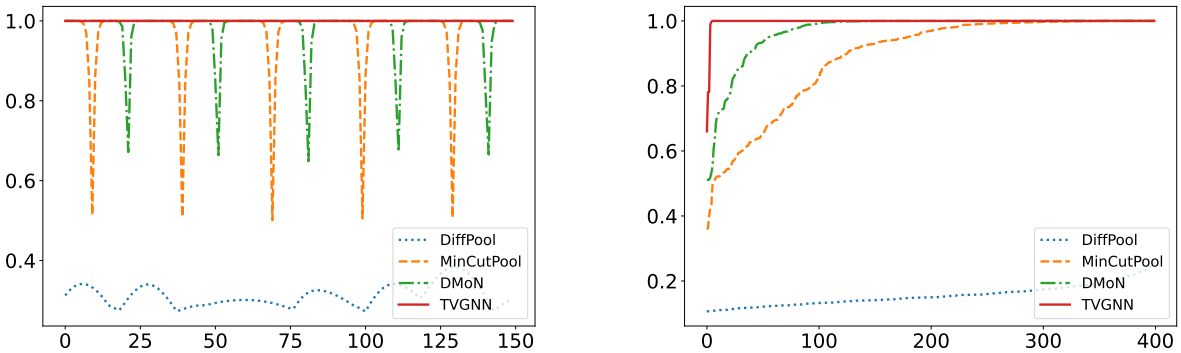
(a) DiffPool (b) MinCutPool (c) DMoN (d) TVGNN

Figure 22: Cluster assignments for the ring graph. The colors correspond to the index of the largest value in the soft cluster assignment vector. The brightness is proportional to the highest value in the assignment vector.



(a) DiffPool (b) MinCutPool (c) DMoN (d) TVGNN

Figure 23: Cluster assignments for the grid graph. The colors correspond to the index of the largest value in the soft cluster assignment vector. The brightness is proportional to the highest value in the assignment vector.



(a) Ring (b) Grid

Figure 24: Largest value in the soft cluster assignment vector as a function of the vertex index. For the ring in (a), the horizontal axis moves along the circumference of the ring. For the grid in (b), the node indices are sorted according to the largest value in the assignment vectors.

hyperparameters as for the vertex classification task in the experiments.

Again we see that DiffPool gives smooth assignments resulting in noticeably paler colors. In MinCutPool and DMoN we observe paler colors in the proximity of the cluster borders, while TVGNN exhibits sharp transitions from one cluster to the other. We also notice that TVGNN is the only method that on the grid generates a partition with the desired number of clusters $K = 10$.

To better quantify the sharpness of the transition between different clusters, in Fig. 24a we show the largest value in the soft

assignment vectors for the ring when moving along it. Here, the differences in cluster transitions are clear: the drops in the assignment value indicate the presence of smooth transitions. The cluster assignments of Diffpool are always very smooth; MinCutPool and DMoN exhibit smooth assignments only when crossing from one cluster to the other; with TVGNN the assignments are always sharp.

A similar plot for the grid is presented in Fig. 24b, but here the largest soft assignments for all nodes are sorted from lowest to highest, which indicates the overall proportion of smooth assignments. Also in this case, the cluster assignments of TVGNN are the sharpest, followed by DMoN, MinCutPool, and Diffpool.