GSCAN: Graph Stability Clustering Using Edge-Aware Excess-of-Mass

Etzion Harari **Naphtali Abudarham** **Roee Litman**
RAFAEL Advanced Defense Systems Ltd.
{etzionh,naphtali,roeel}@rafael.co.il

Abstract

Graph Clustering is required for the identification of communities and groups within a given network. In recent years, various attempts have been made to develop tools suitable for this purpose. Most recently, these attempts are based on the latest advancements in deep learning and especially in Graph Neural Networks (GNN). While some methods take into account the graph intrinsic topological structure throughout, surprisingly, the leading clustering methods ignore this during the final cluster assignment stage, which leads to sub-optimal results. In this paper, we propose GSCAN: a Graph Stability Clustering for Applications with Noise, which is based both on node features and on the graph structure. We base our approach on the celebrated method of Exess-of-Mass (EoM), which is based the principle of maximizing cluster stability. This method has additional desirable properties like resilience to outliers and the fact it doesn't require an a-priory definition of the number of clusters. We extend EoM to work on the *intrinsic* graph structure and propose two possible post-processes to deal with one of EoM's shortcomings - its tendency to over-flagging data-points as outliers. These post processes harness the graph topology and lead to superior performance, even compared to leading clustering approaches that are trained end-to-end. We show that the proposed approach can be implemented in a fast and scalable manner. Our claims are backed on three well-known benchmark datasets. Our code is available here: https://github.com/GraphEoM/GSCAN

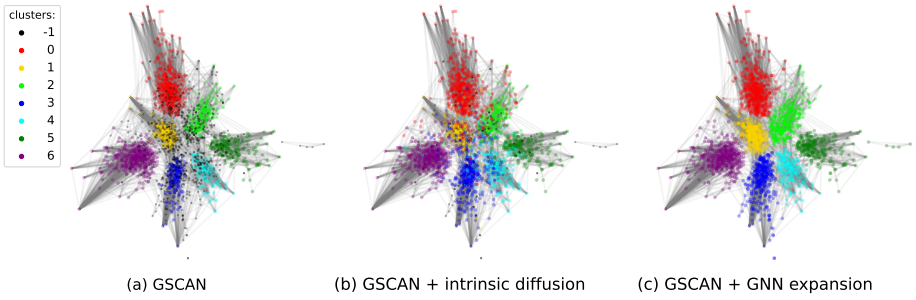

Figure 1: Two dimensional embeddings of the Cora dataset [1] using PCA, colored by clustering results of the three flavours of GSCAN. (a) GSCAN without a post process; One can notice the abundance of black points, which indicate data points that are flagged as outliers. (b) GSCAN with intrinsic diffusion post-process (see Section 3.2.1); Here, the only remaining outliers are the ones that have no connectivity to any of the clusters. (c) GSCAN with GNN-expansion post-process (see Section 3.2.2); This result has no outliers remaining.

1 Introduction

Clustering is the task of grouping a set of data points in such a way that points in the same group (called a cluster) are more similar (in some sense) to each other than to those in other groups (clusters).

E. Harari et al., GSCAN: Graph Stability Clustering Using Edge-Aware Excess-of-Mass. *Proceedings of the Second Learning on Graphs Conference (LoG 2023)*, PMLR 231, Virtual Event, November 27–30, 2023.

The goal of Graph Clustering is to divide graph nodes into sub-graphs, or 'communities', where similar nodes will belong to the same subgroup. This task has many applications, such as in the fields of social sciences [2], biology [3], and citation networks research [4].

As in the case of tabular data clustering, similarity between graph nodes is measured using some mathematical function (for example Euclidean distance, or cosine-similarity), which is based on the vectors of feature values describing data points (i.e. data-points representations, or embeddings); this is called the *extrinsic* distance. However, uniquely for graphs, similarity between nodes can also be measured by the distance between them within the graph - the *intrinsic* distance. Moreover, based on the assumption that graphs have high homophily [5, 6], the existence of an edge between two nodes indicates also a similarity between their features. Therefore, edge information can be extremely valuable when we come to determine whether two nodes should belong to the same cluster or not.

Graph clustering is a well-studied topic and there are many methods that harness the structure of the graph to improve clustering. Typically, when machine learning approaches are applied to improve clustering performance, they are used to yield better *features* of the data points, which in turn should improve similarity measurements. To illustrate this point, Figure 2 depicts an example of the dichotomy between *extrinsic* distances, which ignore the connectivity information, and *intrinsic* distances. However, to the best of our knowledge, some of the leading graph clustering methods use the graph structure *only* during the creation of node features and perform the subsequent clustering phase based on extrinsic distances [7]. This is illustrated in Table 1. By ignoring the intrinsic structure of the graph in the clustering phase, these methods implicitly assume that this information is fully captured in the node features. However, we show that this assumption does not hold in practice, as using our EoM edge-aware clustering, over the learned features improves the performance of most existing clustering methods, including ones that are trained end-to-end.

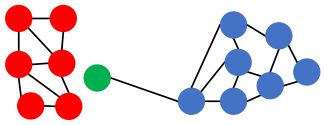

Figure 2: Examples of the difference between intrinsic and extrinsic definition of distances. The green point is to be added one of two clusters, red or blue, according to the one it is more similar to. Measuring the extrinsic two dimensional Euclidean distance, the green point should probably be part of the cluster of red cluster, while if we limit the distances to be measured only between points with edges between them, then it must belong to the blue cluster.

To address the aforementioned gap, we present GSCAN, an edge aware clustering algorithm. We propose to divide the full graph clustering procedure into two logical steps - a representation generation step, followed by a clustering step - and to use a clustering method that is edge-aware, regardless of the underlying representation. To this end we present an edge-aware version of the Excess of Mass (EoM) [8] method. However, since EoM might leave nodes unassigned, we introduce two label expansion methods, as a post process, which also leverage graph connectivity to effectively expand EoM clusters to unassigned nodes.

To compare our method to existing literature, we artificially break down previous methods into the aforementioned two steps (see Table 1) - representation generation and clustering - and show that edge-aware clustering can improve results when using different representations, even representations that are learned in an end-to-end fashion.

In summary, our study makes the following key contributions. First, we show that EoM yields competitive results when used on the intrinsic graph topology rather than the (commonly used) extrinsic feature space. Second, we provide two optional approaches to address unlabeled nodes in the EoM output, for cases these are undefined or undesirable. Finally, we show SOTA clustering results on diverse datasets.

2 Related Work

We now provide a short background for the task of graph clustering, which is by no means exhaustive, and the reader may turn to recent surveys in this area such as [7].

2.1 Problem definition

A graph $G = \{V, E\}$ consists of a set of N vertices $V = \{v_i\}_{i=1,\dots,N}$ (where $N = |V|$) and a set of edges $E = \{e_{ij} = (i, j)$: if i & j connected $\}$. The edges can also be represented by an adjacency matrix A where $A \in \mathbb{R}^{N\times N}$ and $A_{ij} \neq 0$ if there exists an edge between nodes i and j and $A_{ij} = 0$

Table 1: Usage of graph data in different approaches. This table is a non-exhaustive list of possible graph clustering approaches, aimed to highlight how each is utilizing the graph data. For each method we indicate whether it is using the original node features and/or the graph structure, for either feature learning and/or the subsequent clustering phase. It is clear to see that the top four approaches do not use the graph structure for clustering. Below them, two end-to-end methods that are aware of the intrinsic structure, but do not outperform methods like [9]. * indicates the clustering method that each approach used, which we replace with GSCAN in this paper in some of our experiments.

		Representation uses...		Clustering uses...	
Representation	Clustering	node features	structure	node features	structure
Original	KMeans	✓	✗	✓	✗
node2vec [10]	KMeans	✗	✓	✓	✗
DCN [11]	KMeans*	✓	✗	✓	✗
DAEGC [9]	KMeans*	✓	✓	✓	✗
DMoN [12]	pooling*	✓	✓	✓	✓
TVGNN [13]	$\arg\max$*	✓	✓	✓	✓
Original	GSCAN	✓	✗	✓	✓
node2vec [10]	GSCAN	✗	✓	✓	✓
DCN [11]	GSCAN	✓	✗	✓	✓
DAEGC [9]	GSCAN	✓	✓	✓	✓
DMoN [12]	GSCAN	✓	✓	✓	✓
TVGNN [13]	GSCAN	✓	✓	✓	✓

otherwise. An attributed graph also has a matrix $X \in \mathbb{R}^{N\times d}$ representing the d features of each node v_i in vector x_i. These features may be given, or can be the output of a learned model. The task of graph clustering is to assign each node v_i to a single cluster label $y_i \in \{1,\ldots,K\}$, and $C = \{c_k\}_{k=1,\ldots,K}$ is the set of clusters. $D_{ij} = \|x_i - x_j\|$ represents the distance metric between nodes v_i and v_j in the *feature space*.

2.2 Classic clustering methods

Kmeans. [14, 15] is the de-facto gold standard for clustering. KMeans is an iterative clustering algorithm that aims to partition a dataset into K distinct clusters. It minimizes the within-cluster sum of squares (WCSS) of the distances between each sample and the closest centroid, by iteratively assigning samples to the nearest centroid $\mu_k \in \mathbb{R}^d$ and updating the centroids until convergence. The label y_i of node v_i is calculated by minimizing the expression $y_i = argmin_k||x_{i\cdot} - \mu_k||^2$, and the d coordinates of μ_k are calculated in each iteration as follows: $\mu_k = \sum_{i=1}^{N} x_i \mathbb{I}_{\{y_i==k\}} / \sum_{i=1}^{N} \mathbb{I}_{\{y_i==k\}}$.

Even though the KMeans algorithm is widely used, it has several disadvantages: It is sensitive to outliers, it is highly influenced by the K parameter selection, and it relies on the assumption that the underlying distribution of the data is distributed as K compact hyper-spheres [16]. In addition, common KMeans implementations are not naturally adapted to running on the intrinsic structure of the graph - because KMeans's cluster centers μ_k may not necessarily lay on data points.

HDBSCAN. [17–19] is a hierarchical clustering algorithm that identifies clusters based on density estimation, specifically addresses outlier samples, and handles arbitrarily shapes clusters. To determine the density of samples, the algorithm employs a density value, denoted as $\lambda = \frac{1}{D_{ij}}$, which is inversely proportional to the distance D_{ij} Next, the algorithm constructs a Single-Linkage hierarchy [20] of clusters using a minimum spanning tree (MST) algorithm. This condensed tree [19] (also known as a density-contour tree [17, 18], component tree [21] or level-set tree [22]) represents the set of clusters for each density value λ, where each branch of the tree represents a cluster split into two child clusters as the density increases. Clusters must have a minimal size, m, which is an input parameter of the method.

To determine the optimal set of clusters, HDBSCAN uses the concept of Excess of Mass (EoM) [8], which enables it to quantify *cluster stability* which is defined as follows:

$$S(c_k) = \sum_{v_j \in c_k} (\lambda_{max}(v_j, c_k) - \lambda_{min}(c_k)) = \sum_{x_j \in c_k} (\frac{1}{D_{min}(x_j, c_k)} - \frac{1}{D_{max}(c_k)}), \qquad (1)$$

where $\lambda_{min}(c_k) = \min_{v_i, v_j \in c_k} 1/D_{ij}$ marks the smallest density in the cluster c_k, and λ_{max} is defined similarly on the maximal density. Using $S(c_k)$, we can compare each cluster in this binary

tree hierarchy, and select the cluster set with the highest stability:

$$C = \{c_k : max(S(c_k), S(c_k^{(right)}) + S(c_k^{(left)}))\}, \tag{2}$$

where $c_k^{(right)}$ and $c_k^{(left)}$ refer to the two children clusters of the cluster c_k. The recursive expression in Equation 2 compares the stability of each cluster with the combined stability of its children. It then selects only the clusters that maximize the stability within the hierarchy.

2.3 Deep clustering

Deep tabular clustering. In recent years, Deep Neural Networks (DNN) have made significant advancements in supervised learning tasks. Deep learning has also been applied to clustering [23], particularly in the context of dimensionality reduction (DR) as a preprocessing step before applying a clustering algorithm. Typically this approach involves training a Deep Autoencoder (DAE) with a reconstruction loss, where the model learns to reconstruct the original data from its latent representation. Subsequently, clustering algorithms are applied to the resulting latent embeddings [24–26]. Various methods have been proposed to learn low-dimensional latent space embeddings that are suitable for clustering tasks. However, since the DNN and clustering objectives are trained **separately**, the output results may not always be well-suited for clustering data samples.

To improve the clustering results, recent architectures propose training the DNN with a specific focus on clustering. One such architecture is the Deep Embedded Clustering model (DEC) [27], which utilizes the Kullback-Leibler (KL) divergence loss. This loss is inspired by the KMeans [14, 15] clustering approach. Another improvement comes from the end-to-end architecture of the Deep Clustering Network (DCN) [11]. This model combines both the reconstruction loss and the clustering KL loss in its objective. By doing so, it learns a 'KMeans Friendly' representation for each sample and extracts suitable clusters within a single framework.

Deep graph clustering. When attempting to cluster nodes in a given graph, traditional methods often struggle to capture and represent the underlying topological structure of the graph in a reduced latent representation. Previous works have primarily focused on representing the topology alone. For example, architectures like DeepWalk [28] and Node2Vec [10] (N2V) utilize deep learning models to assign each node a representation based on its neighboring nodes. However, these models do not take advantage of node attributes during the learning process, resulting in embeddings that are limited to capturing only the topological features of the graph.

The emergence of Graph Convolutional Networks (GCN) [29] has paved the way for the development of new architectures such as Graph Autoencoders (GAE) and Variational Graph Autoencoders (VGAE) [30]. These models encode node features into a latent space combined with graph edges and then decode the latent representation back into the graph adjacency matrix: $\hat{A} = \text{sigmoid}(zz^T)$, where z represents the latent representation in the GAE's bottleneck. This approach enables the model to learn a lower-dimensional representation for each node in the graph by leveraging both the node attributes X and the graph edges E. To obtain clustering results using GAE, all that is left is to apply some classical clustering algorithms to the output embedding.

In order to enhance the clustering results obtained after applying GAE, Zhang et al. [9] propose a novel architecture called Deep Attentional Embedded Graph Clustering (DAEGC). This model follows an end-to-end approach, similar to DCN [11] for tabular data. DAEGC combines the use of GAE composed of Graph Attention Network (GAT) layers [31], and incorporates a KL clustering loss in addition to the reconstruction loss objective. The learned output embedding is optimized for reconstructing the adjacency matrix A of the graph and is suited for KMeans clustering as well.

While approaches like DAEGC and other end-to-end graph clustering methods [32–34] enjoy the benefits of optimizing the clustering objective in an end-to-end manner, they still have certain limitations due to their reliance on KMeans [16]. Importantly, KMeans disregards the underlying graph structure, and therefore these models may assign two nodes to the same cluster based on representations similarity, even if there is no existing path connecting them.

Recent methods employ end-to-end training strategies that consider the graph structure while assigning clusters. DMoN [12] is a pooling-based method that employs a regularized modularity loss to appropriately allocate nodes to clusters. Another method, TVGNN [13], utilizes total-variation (TV) loss to promote smooth (piece-wise constant) predictions across the graph, which are easier to

cluster by. Both approaches leverage graph topology within an end-to-end framework to learn cluster assignments. However, empirical experiments underscore that methods like the aforementioned [12, 13] do not yield improved results compared to KMeans-based approaches like DAEGC [9] (see Table 3). This gap emphasizes the necessity for alternative graph clustering methods that can harness the performance of methods like [9], while leveraging graph topology in the clustering process.

3 GSCAN Algorithm

In this section, we introduce our approach GSCAN, which utilizes a graph-adapted clustering algorithm provided node representations and, of course, topology. Our method incorporates a hierarchical clustering algorithm that leverages the Excess-of-Mass stability optimizer with an Edge-Aware approach. We also describe two post-processing steps to assign unlabeled nodes to existing clusters. These components have enabled our method to achieve superior results compared to the current state-of-the-art methods. See Section A in the appendix for a flow diagram of GSCAN.

3.1 Edge aware EoM clustering

We propose utilizing the EoM stability optimizer on the MST Hierarchy derived from the graph edges. This approach ensures that nodes lacking connecting paths between them are not assigned to the same cluster. The given edges in the graph allow us to construct a condensed tree[19], which represents the hierarchical structure of graph clusters. The length of each edge in this MST is used as an estimator for the density between two nodes in the graph. Short edges indicate high density, while long edges indicate sparsity. A separate tree will be constructed for each connected component of the graph. By leveraging this principle, we can identify sets of nodes with high density between them and treat them as clusters.

To accomplish this, we define a sparse distance matrix $D \in \mathbb{R}^{|V|\times|V|}$, which is derived from the distance function $\Delta(\cdot)$. Any conventional non-negative distance function (e.g. Euclidean or cosine similarity) can be used to calculate the weight (or length) D_{ij}, of the edge $e_{ij} = (i, j) \in E$:

$$D_{ij} = \begin{cases} \Delta(x_i, x_j) & (i,j) \in E \\ \infty & otherwise \end{cases} \tag{3}$$

The representation vector of node v_i is denoted as x_i. During the actual calculation, we utilize the provided edge set, eliminating the need to compute the entire matrix D. Using D, we construct the MST with the Kruskal Algorithm [35]. This algorithm generates the MST $E^{(mst)}$ by sorting the edges E based on their D values and connecting only those edges that link disjoint sets of nodes. The length D_{ij} of each edge in the MST is used as an estimator for the density λ_{ij}, where $\lambda_{ij} = 1/(D_{ij} + \epsilon)$. The inclusion of ϵ ensures a non-zero denominator, in cases where $D_{ij} = 0$.

It should be mentioned that the scalable version of HDBSCAN [19] utilizes the Boruvka algorithm [36] to construct the MST. By combining Boruvka with an approximate nearest neighbor algorithm, the runtime of HDBSCAN is substantially improved. However, since our method relies on the given graph edges, we can employ the Kruskal algorithm instead.

Our method solely relies on the existing graph edges, allowing us to leverage the inherent homophily within the original graph, which serves the clustering of similar nodes. The resulting tree $E^{(MST)}$ represents the hierarchical clustering dendrogram. By selecting a specific λ value, we can extract the corresponding cluster set C. As mentioned in Section 2.2, different clusters may exhibit varying density values λ, and thus, selecting a single-fixed λ value could lead to suboptimal clusters. Following the approach of Sander et al. [17, 18], we propose the use of an EoM stability optimizer.

By employing the stability measure outlined in Equation 1, we are able to compute the stability $S(c)$ for each cluster in the hierarchical tree structure. Utilizing these stability values, the optimizer identifies the optimal λ value for each branch in $E^{(MST)}$ that maximizes the overall stability. As explained in Equation 2, the subsequent process extracts the clusters set C, which represents the selection of a cluster that maximizes the stability within the hierarchical tree.

3.2 Label expansion

Our method utilizes the stability optimizer to assign labels by identifying the optimal clusters in the hierarchy. By preventing merging of two clusters, the optimizer ensures high stability values.

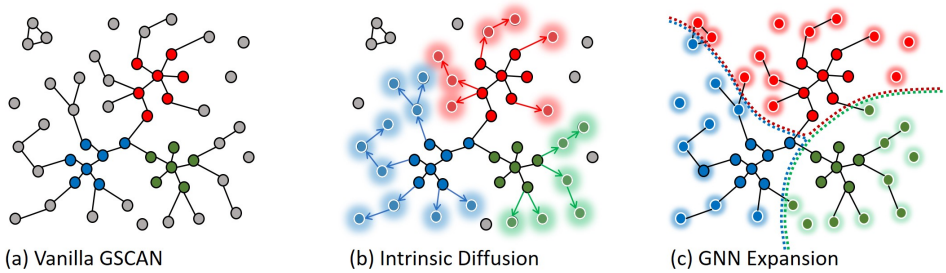

Figure 3: Illustration of GSCAN output and the two label expansion methods

This high-density criterion prevents the inclusion of low-density samples into the original clusters, as described in Figure 3 (a). These excluded samples can be considered as *non-core objects* which do not belong to any stable cluster, as opposed to *core objects* who do. Although these samples are not be classified as core objects, on some occasion it is desirable to assign them to some cluster nevertheless. A similar approach can be observed in tabular clustering methods such as HDBSCAN [17] and DBSCAN [37] which extend the assigned cluster labels also to non-core objects. In the context of graph clustering, this task becomes even simpler since some of these non-core nodes are connected to the core objects through the given edges set. We emphasize that the decision whether to expand the labels to non-core objects at all stems from the nature of the specific problem at hand.

Expanding the labels to these non-core objects on a graph clustering requires an adapted approach tailored to its unique nature. To achieve this, the method should leverage both the graph structure and the node representations. In the following sections, we propose two methods to accomplish this.

3.2.1 Intrinsic diffusion:

This simple approach assigns labels to the remaining nodes by leveraging the graph structure. The label diffusion method computes the closest labeled node for each unlabeled node. To accomplish this, we utilize the graph structure to calculate the intrinsic distance between node v_i and v_j. Using that distance, we assign each remaining node the corresponding label: $\hat{y}_i = \hat{y}_{j^*}$, where j^* represents the closest *intrinsically* labeled node $j^* = argmin_j \Delta(v_i, v_j)$, where Δ denotes the intrinsic distance calculated over the graph. Finding the closest-intrinsic node for each remaining node can be computationally expensive. Inspired by Dijkstra's algorithm [38], we propose a simple algorithm to efficiently assign each remaining node to its closest intrinsic node, as illustrated in Figure 3 (b).

Our method comprises of the following two steps [1] First, we construct a Priority Queue that contains only edges connecting labeled nodes to unlabeled nodes. The edges are sorted based on their length D_{ij}, where $D_{ij} = \Delta(v_i, v_j)$. This distance indicates the dissimilarity between two given samples. Second, we begin by extracting edges from the queue. The queue ensures that in each iteration, we obtain the shortest edge. Similar to the Dijkstra algorithm [38], we update the distances between labeled and unlabeled nodes in each iteration. This process guarantees that each unlabeled node is assigned its closest label in the graph. We assign the unlabeled node $\hat{y}_i$ the value of the labeled node $\hat{y}_{j^*}$. For each assigned node v_j, we add its unlabeled neighbors to the queue. We repeat these two stages process until the queue is empty.

Note that the latter process can be performed using the MST instead of the entire graph. Our experiments have shown that there is no significant performance difference (less than 0.1%) between the two options. However, the tests do show that using only $E^{(mst)}$ improves the run time by a factor of more than two.

It should be noted that our method relies solely on the intrinsic distances between nodes. As a result, graph components without any labeled nodes will remain completely unlabeled. However, the following post-processing method addresses this issue by ensuring that every node in the graph receives a label.

3.2.2 Label expansion using GNN

To address the limitations of the previous method, we can employ a straightforward GNN classification model. This model learns the missing labels by leveraging both the graph structure and node

[1] A more detailed description of this algorithm is included in Section C of the appendix

attributes. We can utilize GNN architectures composed of layers such as Graph Convolutional Networks (GCN)[29] or Graph Attention Networks (GAT)[31]. The training process involves using our core nodes as classification labels to train the GNN model. GNN layers such GCN and GAT based on Message passing [39, 40] mechanism and operates as follows:

$$H_{i\cdot}^{(\ell+1)} = \sigma(\sum_{j\in\mathcal{N}_i} \frac{1}{\gamma_{ij}} H_{j\cdot}^{(\ell)} W^{(\ell)}), \tag{4}$$

where $H_j^{(\ell)}$ is the representation of node v_j in specific layer ℓ (where $H^{(0)} = X$), and $W^{(\ell)}$ denote the learnable weights of the layer, $\mathcal{N}_i$ is the set of neighbors of node v_i (including v_i itself) and γ_{ij} is the weight of the edge (i, j). In GCN layers γ_{ij} is determined and fixed: $\gamma_{ij} = \sqrt{|\mathcal{N}_i| \cdot |\mathcal{N}_j|}$, and for GAT γ_{ij} calculating using learnable weights.

Once the network training is completed, we can use it for inference on our non-core nodes. This ensures that the expansion learning process takes into account both the features and the structure of the graph when assigning each non-core node to its appropriate cluster.

It should be noted that this part of the method is partially extrinsic, and can result in assigning the same label to two nodes that are not connected to each other, based solely on their tabular features. Examples of such errors can be observed in Figure 3(c). At a first glance, this post process seems to negate the very essence of our contribution, as it generates features intrinsically but assigns cluster labels extrinsically. However, is should be emphasized that this post process only applies to non-core objects, and initial labels where set intrinsically using GSCAN. This post process offers a compromise between the intrinsic and extrinsic approaches, and its performance compared to the previous diffusion approach will be dictated by the homophily of the specific data at hand. Additionally, it should be noted that the training process of the GNN is, like all deep learning cases, stochastic in nature, and each training session might yield different outcomes for the same train labels $\{y_i\}$.

4 Experiments

Benchmark datasets. The experiments were conducted using the three Planetoid citation networks dataset: Cora, Citeseer & Pubmed [1]. The properties of each graph dataset are described in Table 2.

Table 2: Properties of the individual dataset used in our experiments.

Dataset	nodes	features	edges	classes	homophily[2]
Cora	2708	1433	10556	7	81.0%
Citeseer	3327	3703	9104	6	73.6%
Pubmed	19717	500	88648	3	80.2%

Baseline methods & representations. We evaluate our method using the artificial division mentioned in the introduction and Table 1, where existing methods comprised of two phases: representation extraction and clustering. The following representation methods are included in our experiments: **DCN** [11] end-to-end DAE tabular method, that learns the representation without edges. **GAE** [30] Graph Autoencoder (& variational Graph Autoencoder) that learns the representation using both structures & features using graph convolutional network layers and reconstruction loss. **DAEGC** [9] end-to-end GAE that learn the representation using KL clustering loss. **DMoN** [12] end-to-end pooling GNN that learn clustering assignment using regulized-modularity loss. **TVGNN** [13] end-to-end GNN that utilizes loss function based on the graph topology using the concept of TV.

Clustering methods. The methods DCN, GAE and DAEGC use **KMeans** [14, 15] as the clustering method, which will be our baseline for comparison. We use the simple extrinsic KMeans implementation, which is identical to the version used as the final step of the end-to-end clustering methods. To compare our method to end-to-end methods, we run GSCAN on the latest node representation in the GNN, either before pooling for [12] or before the logits for [13]. **HDBSCAN** (EoM)[17–19] is included to emphasize the effect of our edge-aware adaptation thereof. We use three versions of HDBSCAN: The first is the vanilla version of the algorithm, and additionally reports the results of our two label expansion methods presented in Section 3.2. **GSCAN** indicates our edge-aware version of EoM, and it too is presented in its 'vanilla' form as well as with the two label expansion methods.

[2]Formal definition of the formula for homophily can be found in Section B.1 of the appendix

Table 3: Performance metrics on [1] datasets, Pubmed, Cora and Citeseer. We indicate the GNN-expansion and intrinsic diffusion by † and ⋆ respectively. MNI values are in the range $[0, 100]$. Best average result for each representation is marked in **bold** (even if not statistically significant), while the very best in the table is additionally colored in **blue**.

repr.	method	edge aware	Pubmed			Cora			Citeseer		
			F1	ARI	NMI	F1	ARI	NMI	F1	ARI	NMI
DCN [11]	KMeans	No	39.9±2.2	0.9±0.6	0.9±0.6	22.5±0.7	0.2±0.4	0.8±0.5	23.7±1.1	0.4±0.2	0.7±0.2
	HDBSCAN	No	40.7±5.6	2.5±4.8	3.2±4.4	24.0±2.5	0.6±0.8	1.4±0.9	24.7±2.0	0.3±0.2	0.9±0.4
	HDBSCAN⋆	No	40.9±6.2	2.5±4.7	3.2±4.5	24.4±2.4	0.6±0.8	1.3±0.9	24.8±2.0	0.3±0.2	0.9±0.4
	HDBSCAN†	No	42.7±5.5	2.8±4.8	3.5±4.5	25.1±2.0	0.7±0.8	1.4±0.8	25.5±1.7	0.4±0.3	1.0±0.5
	GSCAN	Yes	48.1±4.1	3.8±6.9	5.3±6.3	32.9±1.2	3.1±1.3	9.3±2.5	34.5±1.2	1.8±0.6	10.4±1.8
	GSCAN⋆	Yes	**50.1±3.5**	5.0±6.7	6.0±6.4	**38.6±1.9**	**8.1±1.9**	**16.0±3.0**	**38.8±0.8**	**10.0±1.4**	**13.0±1.3**
	GSCAN†	Yes	50.1±4.1	**5.9±7.6**	**6.6±7.5**	32.1±1.9	3.6±1.8	5.7±1.6	32.3±1.0	3.0±1.1	4.0±1.2
GAE [30]	KMeans	No	**58.7±2.4**	16.6±2.4	20.7±2.5	60.8±2.1	35.1±1.9	45.6±1.4	45.5±2.3	16.1±2.4	21.9±1.8
	HDBSCAN	No	52.6±2.6	10.6±3.5	19.3±5.1	52.9±3.2	21.2±3.7	38.9±3.6	30.7±1.6	1.1±0.8	6.0±2.7
	HDBSCAN⋆	No	57.1±3.1	17.5±5.6	24.2±5.9	61.9±4.9	38.3±6.0	45.8±3.5	37.1±4.4	3.5±3.0	11.4±5.1
	HDBSCAN†	No	54.7±4.2	13.5±7.8	20.3±6.2	54.0±4.4	24.1±6.1	38.3±3.1	38.4±4.5	4.8±3.6	13.4±5.5
	GSCAN	Yes	50.4±1.5	7.1±2.4	19.5±1.3	61.4±1.3	28.0±2.2	46.0±1.1	45.3±1.6	9.6±2.2	25.0±1.0
	GSCAN⋆	Yes	57.8±0.9	20.1±1.0	25.2±1.6	**68.1±1.4**	**41.9±2.7**	**50.6±1.1**	48.7±0.9	16.2±1.1	24.8±0.9
	GSCAN†	Yes	57.8±1.2	**20.5±1.5**	**25.7±2.0**	65.0±1.9	38.2±3.2	47.1±2.2	**57.9±1.5**	**29.0±2.9**	**33.3±1.6**
DMoN [12]	pooling	Yes	43.8±4.4	11.3±2.0	17.3±2.9	55.7±2.1	29.3±1.5	46.5±1.4	45.3±3.0	19.5±1.9	27.9±1.7
	HDBSCAN	No	49.8±1.1	1.3±1.0	7.1±2.9	54.2±2.6	26.8±2.2	44.4±1.5	44.9±3.0	19.0±1.9	28.1±1.7
	HDBSCAN⋆	No	50.6±0.6	1.4±1.0	7.4±3.1	57.0±3.0	30.9±2.4	46.6±1.5	45.4±3.0	19.6±1.9	28.1±1.7
	HDBSCAN†	No	50.6±0.7	1.6±1.1	7.8±3.3	56.7±2.8	30.7±2.2	46.2±1.3	45.4±3.1	19.6±2.0	28.0±1.7
	GSCAN	Yes	49.1±1.8	14.4±1.2	8.2±1.6	57.1±3.7	27.4±3.5	46.0±1.7	41.3±2.5	7.6±1.3	25.2±1.9
	GSCAN⋆	Yes	**51.0±2.7**	**16.2±1.7**	**17.9±2.3**	**60.5±3.9**	34.1±4.0	**47.5±1.8**	43.7±2.9	12.9±1.8	24.3±1.9
	GSCAN†	Yes	50.3±1.8	14.4±1.5	13.6±2.6	59.6±4.8	**34.6±4.9**	47.1±2.2	**49.4±3.2**	**21.7±2.5**	**28.2±1.7**
TVGNN [13]	arg max	Yes	58.6±3.7	18.7±1.3	21.9±1.3	57.1±5.7	31.3±6.3	41.1±5.3	57.8±5.1	31.1±4.6	33.0±3.5
	HDBSCAN	No	53.9±4.3	12.1±6.7	13.1±6.8	41.4±4.2	9.2±3.4	23.6±5.9	55.0±4.2	26.0±4.0	28.5±2.8
	HDBSCAN⋆	No	58.9±3.4	15.6±8.1	17.7±8.7	56.3±9.1	28.2±12.5	39.2±9.7	57.0±4.5	32.0±4.3	33.2±3.5
	HDBSCAN†	No	58.8±3.4	15.5±8.1	17.6±8.8	56.0±8.8	28.0±11.7	38.4±9.3	58.9±4.6	32.0±4.3	33.1±3.5
	GSCAN	Yes	47.7±1.2	3.9±2.7	12.6±1.4	47.1±5.0	13.7±4.3	31.9±4.9	49.2±2.3	13.5±1.4	26.9±2.0
	GSCAN⋆	Yes	58.3±4.8	18.5±2.0	20.5±1.5	**58.8±6.7**	30.6±8.8	**43.5±4.9**	49.0±2.2	17.6±1.6	25.5±1.9
	GSCAN†	Yes	**59.3±4.7**	**20.4±2.4**	**23.2±1.9**	58.6±6.9	**32.1±9.1**	42.9±5.5	**58.9±4.6**	**32.0±4.3**	**33.5±3.5**
DAEGC [9]	KMeans	No	59.1±0.9	19.7±1.2	23.4±1.8	66.3±1.9	41.7±2.3	48.8±1.9	62.9±1.2	35.7±1.7	35.5±1.3
	HDBSCAN	No	58.1±3.9	13.8±4.2	19.7±5.1	57.5±4.4	26.7±3.8	43.8±4.8	50.7±3.9	12.1±4.1	30.7±3.2
	HDBSCAN⋆	No	62.7±5.1	26.4±8.2	30.0±7.0	65.0±7.0	41.1±8.2	49.5±6.1	60.6±3.0	34.0±3.8	35.1±2.4
	HDBSCAN†	No	62.5±4.1	22.5±8.0	25.3±7.4	59.9±6.6	34.4±7.9	44.3±5.9	57.1±3.9	30.2±4.3	32.5±2.8
	GSCAN	Yes	55.8±3.0	14.2±4.5	18.3±4.1	62.9±0.7	31.2±2.6	49.9±1.2	49.6±0.7	13.6±1.2	28.0±1.0
	GSCAN⋆	Yes	65.4±1.8	25.2±3.5	25.4±2.3	71.2±0.8	46.7±1.8	52.2±0.5	50.5±0.4	18.1±0.6	26.7±0.6
	GSCAN†	Yes	**67.6±4.0**	**31.0±5.0**	**31.7±3.7**	**71.7±1.1**	**49.6±2.4**	**52.4±1.4**	**64.7±2.1**	**38.2±3.2**	**39.9±2.3**

4.1 Evaluation metrics & parameter settings.

Metrics. We use three standard clustering evaluation metrics [41]: Normalized Mutual Information (NMI), Adjusted Rand Index (ARI) & F1-Score [42]. It is common practice to also use the Accuracy (ACC) measure. However, a drawback of ACC is that it relies on prior knowledge of the number of clusters. This poses a problem as some of the methods being evaluated use the number of clusters as a parameter while others do not. Hence, the use of this metric becomes problematic in such cases. We also report the percentage of samples that are assigned to a cluster (covered). For statistical significance, we ran each experiment 20 times, and report average values and standard-deviation.

Parameter settings. For our experiment, we employed the default parameters of the following methods: DAEGC, DMoN, TVGNN, and DCN, as described in their original papers. All Autoencoders utilized a 16-dimensional latent space for node embedding. GAEs employed two layers of GAT [31] layers. Whenever the number of cluster parameter K was required, we used the actual number of classes present in each dataset. The distance function $\Delta(\cdot)$ for our method is the cosine distance function. The parameter m (minimum number of samples in a cluster) for EoM clustering was set [3] to be about 2% of the number of nodes N. We ran our experiments on a machine with an 8-core Intel i7 processor with 64GB of RAM, and a GeForce RTX 3070 GPU card with 8GB of memory.

4.2 Experiment results

The experimental results are presented in Table 3. The results show significant variations across different representations. This finding highlights the considerable impact of representation choice on the outcomes of clustering algorithms. Notably, at least one of our two label expansion flavours over GSCAN consistently outperforms all other algorithms in the majority of experiments. This observation underscores the robustness of our approach in effectively handling diverse input representations. Additionally, our post-processing methods yield better results than the KMeans even for end-to-end methods DAEGC and DCN, despite the fact they were trained specifically to fit the KMeans algorithm. The improvement of our method over TVGNN, while significant, is smaller compared to the improvement over DAEGC. We attribute this to the fact that TVGNN is already fully edge-aware, which makes it challenging for our method to enhance results.

The representation generated by the DAEGC [9] achieves the highest performance, an observation that highlights its expressive power. The gap in performance of GSCAN over HDBSCAN emphasizes the importance of edge awareness for graph clustering. Finally, the performance of the DCN is relatively low compared to GNN-based algorithms. This observation highlights the significance of incorporating both the graph structure and node attributes in the graph clustering process.

Cover percentage. is displayed in Table 4, which is the proportion of nodes that are assigned to a cluster. It is evident that both our method and HDBSCAN may not assign cluster labels to every node in the graph. However, our proposed post-processing methods greatly enhance this aspect.

Table 4: Average covered percentage of each method, for each dataset

dataset	KMeans	HDBSCAN	HDBSCAN*	HDBSCAN†	GSCAN	GSCAN*	GSCAN†
cora	100	70.0±14.9	100	100	41.3±20.7	91.8	100
citeseer	100	54.1±24.8	100	100	40.6±11.5	63.7	100
pubmed	100	70.6±22.5	100	100	60.0±22.9	100	100

Run time. Our clustering calculation time is influenced mainly by the number of edges $|E|$ and also by the dimension of node features X. We can see in Table 5 that indeed the runtime grows proportionally with the number of edges $|E|$.

Table 5: Run times for some of the methods in Table 3 in seconds. See Table 3 for the meaning of † and ⋆.

dataset	KMeans	GSCAN	GSCAN†	GSCAN*
citeseer	0.14±0.01	0.03±0.01	0.49±0.05	0.02±0.0
cora	0.13±0.01	0.03±0.01	0.49±0.01	0.02±0.0
pubmed	0.2±0.03	0.28±0.03	0.59±0.02	0.21±0.05

5 Conclusion

In the current study we have shown that the leading recent deep graph clustering approaches use the graph structure for feature extraction only and ignore it in the final clustering phase. We have introduced GSCAN, a new edge-aware version of EoM, and additionally two possible label expansion approaches to handle unassigned nodes. We show that when coupled with the expansion methods, GSCAN consistently obtains SOTA results over several datasets.

At least one of the expansion methods consistently reaches the top results, including the vanilla GSCAN. The reason for either expansion to yield better results depends mainly on the problem at hand; For example, the Planetoid datsets [1] we used do not include outlier labels, which favors our label expansion over vanilla GSCAN. In more realistic scenarios, where one might deal with many outliers, the vanilla version might perform better.

While GSCAN indeed uses the graph structure, it is not incorporated into the feature training process. A future stage in our work would be to show how GSCAN can be incorporated into an end-to-end training process, i.e. to show it is differentiable, or at the very least be used in an alternating optimization framework as was done in [9] or [13].

[3] see Section B.2 of the appendix for more details on the selection and stability of the value of m

References

[1] Zhilin Yang, William Cohen, and Ruslan Salakhudinov. Revisiting semi-supervised learning with graph embeddings. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 40–48, New York, New York, USA, 20–22 Jun 2016. PMLR. URL https://proceedings.mlr.press/v48/yanga16.html. 1, 7, 8, 9, 14

[2] Jierui Xie and Boleslaw K Szymanski. Towards linear time overlapping community detection in social networks. In *Advances in Knowledge Discovery and Data Mining: 16th Pacific-Asia Conference, PAKDD 2012, Kuala Lumpur, Malaysia, May 29–June 1, 2012, Proceedings, Part II 16*, pages 25–36. Springer, 2012. 2

[3] Erzsébet Ravasz, Anna Lisa Somera, Dale A Mongru, Zoltán N Oltvai, and A-L Barabási. Hierarchical organization of modularity in metabolic networks. *science*, 297(5586):1551–1555, 2002. 2

[4] P Chen and Sidney Redner. Community structure of the physical review citation network. *Journal of Informetrics*, 4(3):278–290, 2010. 2

[5] Miller McPherson, Lynn Smith-Lovin, and James M Cook. Birds of a feather: Homophily in social networks. *Annual review of sociology*, 27(1):415–444, 2001. 2, 13

[6] David Easley and Jon Kleinberg. *Networks, crowds, and markets: Reasoning about a highly connected world*. Cambridge university press, 2010. 2, 13

[7] Yue Liu, Jun Xia, Sihang Zhou, Siwei Wang, Xifeng Guo, Xihong Yang, Ke Liang, Wenxuan Tu, Stan Z. Li, and Xinwang Liu. A survey of deep graph clustering: Taxonomy, challenge, and application, 2022. 2

[8] Dietrich Werner Müller and Günther Sawitzki. Excess mass estimates and tests for multimodality. *Journal of the American Statistical Association*, 86(415):738–746, 1991. 2, 3

[9] Chun Wang, Shirui Pan, Ruiqi Hu, Guodong Long, Jing Jiang, and Chengqi Zhang. Attributed graph clustering: A deep attentional embedding approach, 2019. 3, 4, 5, 7, 8, 9, 13

[10] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864, 2016. 3, 4

[11] Bo Yang, Xiao Fu, Nicholas D Sidiropoulos, and Mingyi Hong. Towards k-means-friendly spaces: Simultaneous deep learning and clustering. In *international conference on machine learning*, pages 3861–3870. PMLR, 2017. 3, 4, 7, 8

[12] Anton Tsitsulin, John Palowitch, Bryan Perozzi, and Emmanuel Müller. Graph clustering with graph neural networks. *arXiv preprint arXiv:2006.16904*, 2020. 3, 4, 5, 7, 8

[13] Jonas Berg Hansen and Filippo Maria Bianchi. Total variation graph neural networks. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 12445–12468. PMLR, 23–29 Jul 2023. 3, 4, 5, 7, 8, 9, 13, 14

[14] Edward W Forgy. Cluster analysis of multivariate data: Efficiency vs. interpretability of classifications. *biometrics*, 21:768–769, 1965. 3, 4, 7

[15] Stuart Lloyd. Least squares quantization in pcm. *IEEE transactions on information theory*, 28 (2):129–137, 1982. 3, 4, 7

[16] Abiodun M Ikotun, Absalom E Ezugwu, Laith Abualigah, Belal Abuhaija, and Jia Heming. K-means clustering algorithms: A comprehensive review, variants analysis, and advances in the era of big data. *Information Sciences*, 2022. 3, 4

[17] Ricardo JGB Campello, Davoud Moulavi, and Sander. Density-based clustering based on hierarchical density estimates. In *Advances in Knowledge Discovery and Data Mining: 17th Pacific-Asia Conference, PAKDD 2013, Gold Coast, Australia, April 14-17, 2013, Proceedings, Part II 17*, pages 160–172. Springer, 2013. 3, 5, 6, 7

[18] Ricardo JGB Campello, Davoud Moulavi, Arthur Zimek, and Jörg Sander. Hierarchical density estimates for data clustering, visualization, and outlier detection. In *ACM Transactions on Knowledge Discovery from Data (TKDD)*, volume 10, pages 1–51. ACM New York, NY, USA, 2015. 3, 5

[19] Leland McInnes and John Healy. Accelerated hierarchical density based clustering. In *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*, pages 33–42. IEEE, 2017. 3, 5, 7

[20] Werner Stuetzle and Rebecca Nugent. A generalized single linkage method for estimating the cluster tree of a density. *Journal of Computational and Graphical Statistics*, 19(2):397–418, 2010. 3

[21] Laurent Najman and Michel Couprie. Building the component tree in quasi-linear time. *IEEE Transactions on image processing*, 15(11):3531–3539, 2006. 3

[22] Julie Digne, Jean-Michel Morel, Nicolas Audfray, and Charyar Mehdi-Souzani. The level set tree on meshes. In *Proc. 3DPVT*, volume 2, 2010. 3

[23] Erxue Min, Xifeng Guo, Qiang Liu, Gen Zhang, Jianjing Cui, and Jun Long. A survey of clustering with deep learning: From the perspective of network architecture. *IEEE Access*, 6: 39501–39514, 2018. 4

[24] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, Pierre-Antoine Manzagol, and Léon Bottou. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of machine learning research*, 11(12), 2010. 4

[25] Galen Andrew, Raman Arora, Jeff Bilmes, and Karen Livescu. Deep canonical correlation analysis. In *International conference on machine learning*, pages 1247–1255. PMLR, 2013.

[26] Andrew Ng et al. Sparse autoencoder. *CS294A Lecture notes*, 72(2011):1–19, 2011. 4

[27] Junyuan Xie, Ross Girshick, and Ali Farhadi. Unsupervised deep embedding for clustering analysis. In *International conference on machine learning*, pages 478–487. PMLR, 2016. 4

[28] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710, 2014. 4

[29] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016. 4, 7

[30] Thomas N Kipf and Max Welling. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*, 2016. 4, 7, 8, 14

[31] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, Yoshua Bengio, et al. Graph attention networks. *stat*, 1050(20):10–48550, 2017. 4, 7, 8

[32] Huiling Xu, Wei Xia, Quanxue Gao, Jungong Han, and Xinbo Gao. Graph embedding clustering: Graph attention auto-encoder with cluster-specificity distribution. *Neural Networks*, 142: 221–230, 2021. ISSN 0893-6080. doi: https://doi.org/10.1016/j.neunet.2021.05.008. URL https://www.sciencedirect.com/science/article/pii/S0893608021002008. 4

[33] Zhihao Peng, Hui Liu, Yuheng Jia, and Junhui Hou. Deep attention-guided graph clustering with dual self-supervision. *IEEE Transactions on Circuits and Systems for Video Technology*, 2022.

[34] Hongyuan Zhang, Pei Li, Rui Zhang, and Xuelong Li. Embedding graph auto-encoder for graph clustering. *IEEE Transactions on Neural Networks and Learning Systems*, 2022. 4, 14

[35] Joseph B Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical society*, 7(1):48–50, 1956. 5

[36] Otakar Boruvka. O jistem problemu minimalnim. In *Práce Moravské přírodovědecké společnosti*, 1926. 5

[37] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *kdd*, volume 96, pages 226–231, 1996. 6

[38] Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959. 6, 15

[39] Marco Gori, Gabriele Monfardini, and Franco Scarselli. A new model for learning in graph domains. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 2, pages 729–734. IEEE, 2005. 7

[40] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018. 7

[41] Rongkai Xia, Yan Pan, Lei Du, and Jian Yin. Robust multi-view spectral clustering via low-rank and sparse decomposition. In *Proceedings of the AAAI conference on artificial intelligence*, volume 28, 2014. 8

[42] Bjornar Larsen and Chinatsu Aone. Fast and effective text mining using linear-time document clustering. In *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 16–22, 1999. 8

[43] Sami Abu-El-Haija, Bryan Perozzi, Amol Kapoor, Nazanin Alipourfard, Kristina Lerman, Hrayr Harutyunyan, Greg Ver Steeg, and Aram Galstyan. Mixhop: Higher-order graph convolutional architectures via sparsified neighborhood mixing. In *international conference on machine learning*, pages 21–29. PMLR, 2019. 13

Appendix

A Flow diagram of GSCAN

For clarity, we include a high-level flow diagram 4 of the proposed method.

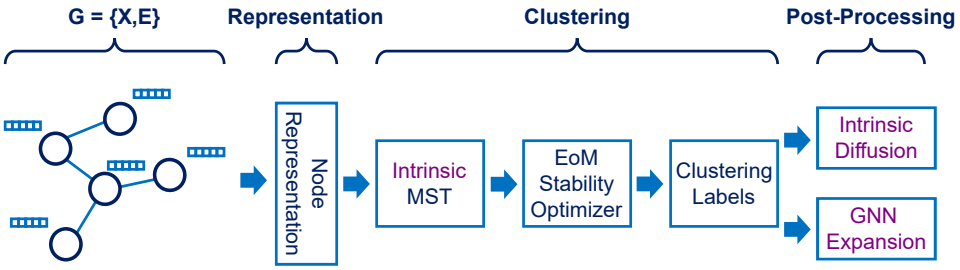

Figure 4: Flow diagram of GSCAN, where our main contributions are marked in violet therein. For a given graph G (with feature matrix X and edges set E), we use some method (such as DAEGC [9] or TVGNN [13]) to obtain node representation. Using this representation, we perform graph clustering using the Excess of Mass optimizer. To handle unlabeled nodes, we optionally perform one of the two proposed post-processing stages, to expand the labels to non-core objects in the graph.

B Additional experimental results

B.1 Effect of Homophily

Our intrinsic approach to graph clustering is based on the assumption of a highly homophilic topology. Homophily [5] [6] is the principle that the graph's topology reflects the similarity between pairs of nodes. The homophily ratio quantifies this relationship and is computed using the following formula:

$$\text{Homophily}(G) = \frac{1}{|E|} \sum_{(i,j)\in E} \mathbb{I}_{\{Y_i == Y_j\}} \tag{5}$$

where $\mathbb{I}$ is the indicator function. A high homophily ratio indicates a strong correlation between the topology of the graph and the partitioning of nodes. To investigate the impact of the homophily ratio on our method's performance, we conducted an experiment using the **MixHop** dataset [43]. This dataset consists 10 sub-datasets, all sharing the same feature matrix X but with distinct sets of edges E. Each sub-dataset represents a different homophily value ranging from 0 to 0.9. By applying our method to these datasets, we can assess the influence of the homophily ratio on performance, as illustrated in Figure 5.

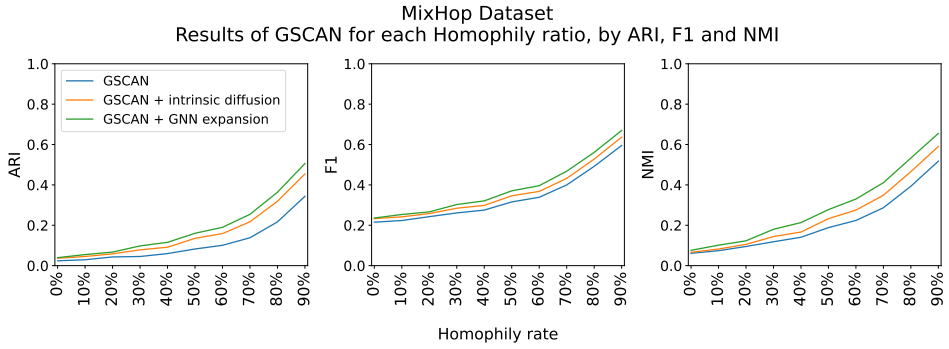

Figure 5: Homophily influence on GSCAN performance The charts shows the resulting NMI, F1 score and NMI for each homophily rate.

The results indicate a positive relationship between the homophily ratio and the performance of our method, as stated in the main text. As the homophily ratio increases, our algorithm consistently

delivers improved results. This property is not of GSCAN alone and is shared with most edge-aware methods. These findings underscore the significant influence of graph homophily on the quality of clustering outcomes. Specifically, when the intrinsic distance accurately reflects the similarity between nodes, the resulting clusters exhibit higher quality and improved clustering performance.

B.2 Selection of the parameter m

The parameter m indicates the minimal allowed cluster size in the EoM process. In our experiments, we use m approximately 2% of the number of nodes N. Figure 6 shows the performance of GSCAN over the features of TVGNN [13] (arbitrarily selected, other representations behave similarly). It can be seen that the performance stay quite constant well beyond the value we selected. We stress that the optimal value for m depends on the data at hand, and a smaller value of m might be needed in some cases.

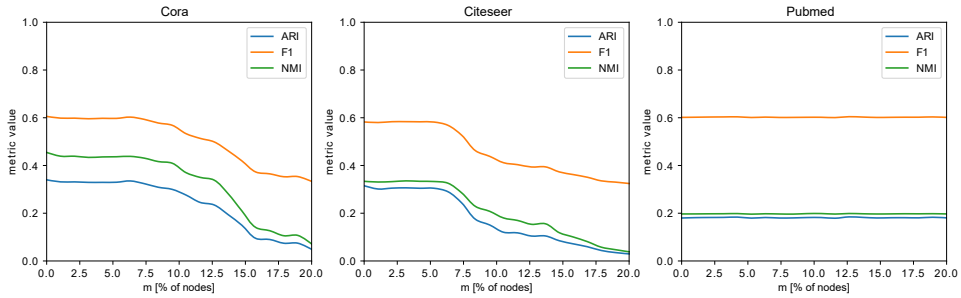

Figure 6: Sensitivity of GSCAN over the Planetoid [1] datasets to the parameter m (or lack thereof).We used $m \approx 2\%$ of the number of nodes N, which is well in the plateau region of the graphs.

B.3 Results using VGAE and EGAE representations

Due to space considerations, as well as architecture similarity to other methods, the results on VGAE [30] and EGAE [34] were omitted from Table 3 in the main text. We append them here for completeness.

Table 6: Performance metrics on [1] datasets, Pubmed, Cora and Citeseer, using representation of VGAE [30] and EGAE [34]. See Table 3 in the main text for more details.

			Pubmed			Cora			Citeseer		
repr.	method	edge aware	F1	ARI	NMI	F1	ARI	NMI	F1	ARI	NMI
VGAE [30]	KMeans	No	49.5±1.2	4.0±0.8	6.5±1.4	23.2±0.5	0.4±0.3	0.9±0.5	46.0±1.7	15.0±0.9	19.8±1.3
	GSCAN	Yes	50.4±0.5	0	2.9±1.2	30.0	0	2.5	43.1±1.6	6.8±1.2	22.8±1.4
	GSCAN*	Yes	**57.2±2.5**	**10.6±4.9**	**11.5±4.4**	**30.4±2.0**	0.3±2.7	**3.2±3.3**	48.0±0.8	15.9±1.1	24.3±1.0
	GSCAN†	Yes	56.6±2.5	9.7±4.6	11.0±5.0	30.1±2.2	**0.6±2.7**	0.9±4.1	**57.1±2.1**	**28.3±3.4**	**32.9±1.8**
EGAE [34]	KMeans	No	53.5±1.4	15.4±1.2	20.3±1.7	18.7±0.1	0	0.4	45.7±1.5	16.9±2.4	23.9±1.3
	GSCAN	Yes	61.5±1.8	19.9±2.8	20.6±2.4	31.6±0.7	1.3±0.6	7.8±2.2	44.7±0.9	9.4±1.3	**26.1±0.6**
	GSCAN*	Yes	62.5±1.9	20.1±3.1	20.5±2.3	**42.7±3.7**	**13.4±5.0**	**19.4±3.4**	47.1±0.4	14.8±0.8	25.8±0.6
	GSCAN†	Yes	**62.8±2.0**	**20.6±3.2**	**21.4±2.5**	30.9±0.9	2.1±1.7	3.1±1.1	**50.1±1.0**	**21.8±2.2**	26.0±1.6

C Intrinsic Diffusion Label Expansion Algorithm

In Section 3.2.1 of the main text, we described our Intrinsic Diffusion post-processing label expansion method. This algorithm allows the expansion of GSCAN's output labels to unlabeled nodes. The complete description of the diffusion algorithm is presented in Algorithm 1 below.

Algorithm 1 Graph Intrinsic Diffusion. By utilizing the Priority Queue in a similar manner to Dijkstra [38], we guarantee that each unlabeled node v_i obtains the closest intrinsic label y_j.

Require: Graph G with nodes V, edges E and nodes cluster labels $y_i{}_{i=1}^{N}$, all as defined in section 2.1 in the main text.

Ensure: Diffusing graph clustering of labels onto unlabeled nodes (marked as $y_i = -1$) that are connected to labeled nodes.

$Q \leftarrow PriorityQueue()$
for $(i, j) \in E$ **do**
  if $(y_i < 0$ & $y_j \geq 0)$ **then**
    $D_{ij} \leftarrow \Delta(v_i, v_j)$
    $Q.add((i, j), by = D_{ij})$
  end if
  if $(y_i \geq 0$ & $y_j < 0)$ **then**
    $D_{ij} \leftarrow \Delta(v_i, v_j)$
    $Q.add((j, i), by = D_{ij})$
  end if
end for
▷ now the Priority Queue ready and sorted by length
$R \leftarrow [\cdots]$ ▷ $|R| = |V|$, distance from a labeled node
$R_i = \infty$ if $y_i < 0$ else 0
while Q not empty **do**
  $(i, j) \leftarrow Q.pop()$
  if $y_j < 0$ **then**
    $y_j \leftarrow y_i$
    $R_j \leftarrow \Delta(v_i, v_j)$
    for $r \in Neighbors(v_j)$ **do**
      if $R_r > \Delta(v_j, v_r) + R_j$ **then**
        $D_{jr} \leftarrow \Delta(v_j, v_r)$
        $Q.add((j, r), by = D_{jr} + R_j)$
      end if
    end for
  end if
end while