Robust Graph Clustering via Meta Weighting for Noisy Graphs

Hyeonsoo Jo
Kim Jaechul Graduate School of AI
KAIST
Seoul, Republic of Korea
hsjo@kaist.ac.kr

Fanchen Bu
School of Electrical Engineering
KAIST
Daejeon, Republic of Korea
boqvezen97@kaist.ac.kr

Kijung Shin
Kim Jaechul Graduate School of AI
KAIST
Seoul, Republic of Korea
kijungs@kaist.ac.kr

ABSTRACT

How can we find meaningful clusters in a graph robustly against noise edges? Graph clustering (i.e., dividing nodes into groups of similar ones) is a fundamental problem in graph analysis with applications in various fields. Recent studies have demonstrated that graph neural network (GNN) based approaches yield promising results for graph clustering. However, we observe that their performance degenerates significantly on graphs with noise edges, which are prevalent in practice. In this work, we propose MetaGC for robust GNN-based graph clustering. MetaGC employs a decomposable clustering loss function, which can be rephrased as a sum of losses over node pairs. We add a learnable weight to each node pair, and MetaGC adaptively adjusts the weights of node pairs using meta-weighting so that the weights of meaningful node pairs increase and the weights of less-meaningful ones (e.g., noise edges) decrease. We show empirically that MetaGC learns weights as intended and consequently outperforms the state-of-the-art GNN-based competitors, even when they are equipped with separate denoising schemes, on five real-world graphs under varying levels of noise. Our code and datasets are available at https://github.com/HyeonsooJo/MetaGC.

CCS CONCEPTS

• **Information systems** → **Data mining**; **Clustering**; • **Computing methodologies** → **Machine learning**; **Neural networks**.

KEYWORDS

Graph Clustering; Meta Weighting; Robust Learning

ACM Reference Format:
Hyeonsoo Jo, Fanchen Bu, and Kijung Shin. 2023. Robust Graph Clustering via Meta Weighting for Noisy Graphs. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management (CIKM '23), October 21–25, 2023, Birmingham, United Kingdom.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3583780.3615038

1 INTRODUCTION

Graphs are a powerful way to represent various systems in physics, bioinformatics, social science, etc. Real-world graphs usually contain substructures called clusters, where each cluster is a set of similar nodes. Clusters in real-world graphs have useful implications, such as social groups in friendship networks [2, 17], functional modules in protein-interaction networks [8], and groups of papers on the same topic in citation networks [49]. Recently, graph neural networks (GNNs), which are a class of deep learning models designed to perform inference on graph-structured data with side information (e.g., node attributes), have received considerable attention. GNN-based representation learning has shown remarkable performance in various tasks, including node classification, link prediction, and graph classification [20, 28, 59].

Several GNN-based approaches have been developed also for graph clustering [4, 16, 53]. In them, GNNs are trained for objectives of graph clustering (e.g., cut and modularity) to produce a (soft) clustering assignment of nodes. These approaches are effective, especially when abundant node attributes are given, because GNNs are trained end-to-end to exploit both node attributes and graph topology for a considered task.

GNN-based approaches in general are known to be vulnerable to noise edges in graphs, since message passing, the fundamental building block of GNNs, is performed through both meaningful edges and noise edges. Therefore, GNN-based graph-clustering methods also have a common problem of being vulnerable to noise edges. We observe that their performance degenerates greatly on graphs with noise edges, as detailed in the experiment section.

However, real-world graphs, including social networks [3, 15], auction networks [43], SMS networks [46], review networks [56], computer networks [51], are often contaminated by noise edges [12, 36] due to click errors [19], bots [7, 48], and spam [33], to name a few. Recently, several trials have been made on training GNNs to be robust to structural noise. To alleviate structural noise such as noise edges, some methods [13, 21, 36, 57] eliminate noise edges by using similarity of node attributes or some assumptions such as low rank, sparsity, and attribute-smoothness. Another related line of research has focused on enhancing the robustness of GNNs by modifying the message-passing schemes without explicit graph denoising. Specifically, in those works, GNNs are designed to be robust for node classification [14, 60, 61]. However, methods for improving the robustness of graph clustering have been underexplored.

To address the above problems, we propose MetaGC (**Meta**-weighting based **G**raph **C**lustering) for robust GNN-based graph clustering against noise edges. MetaGC employs a decomposable clustering loss function, with theoretical justification, and it uses a meta-model to adaptively adjust the weight of each pair in the corresponding loss term (spec., lowering the weights of noise edges). Both the meta-model and the GNN-based clustering model in MetaGC are trained end to end for graph clustering. Consequently, MetaGC is able to produce better clusters than separately applying graph denoising schemes before graph clustering.

CIKM '23, October 21–25, 2023, Birmingham, United Kingdom

ACM ISBN 979-8-4007-0124-5/23/10…$15.00
https://doi.org/10.1145/3583780.3615038

Table 1: Frequently-used symbols and definitions.

Symbol	Definition
$G = (V, E)$	an input graph with nodes V and edges E
$N = \|V\|$	the number of nodes
$A \in \{0,1\}^{N \times N}$	the adjacency matrix of G
F	the dimension of each node's attribute vector
$X \in \mathbb{R}^{N \times F}$	the node attribute matrix of G
$D = \text{diag}(A\mathbf{1}_N)$	the degree matrix of A
$d_i = \sum_{i'=1}^{N} A_{ii'}$	the degree of node v_i
K	the number of clusters
$P \in \mathcal{P} \subseteq [0,1]^{N \times K}$	a (soft) cluster assignment matrix

Moreover, we demonstrate the effectiveness of meta-weighting in comparison to non-meta-weighting-based end-to-end approaches [21, 36], which we adapt for graph clustering.

Our contributions are listed as follows:

- **Observations**: We show that GNN-based clustering approaches are vulnerable to noise edges. Theoretically, we define a class of decomposable clustering loss functions (e.g., modularity-based ones) and prove that they are suitable for continuous relaxation needed by GNN-based end-to-end learning.
- **Methodology**: We design MetaGC for improving the robustness of GNN-based graph clustering. To the best of our knowledge, we are the first (a) to use meta-weighting for the *robustness* of GNNs and (b) to use meta-weighting specialized in *graph clustering*.
- **Extensive Experiments**: In our experiments on 5 real-world graphs under 3 levels of noise, we show the advantages of MetaGC over its state-of-the-art competitors, even when they use separate denoising schemes.

The rest of this paper is organized as follows. In Sec. 2, we provide some preliminaries and give a brief survey of related work. In Sec. 3, we describe our proposed method. In Sec. 4, we review our experiments. In Sec. 5, we conclude our work.

2 PRELIMINARIES & RELATED WORK

In this section, we provide some mathematical preliminaries used throughout this paper and review some related studies.

2.1 Mathematical Background

Let $G = (V, E)$ be an unweighted, undirected,[1] and self-loop-free graph with node set $V = \{v_1, \cdots, v_{|V|}\}$ and edge set $E \subseteq \binom{V}{2}$. Each edge $(v_i, v_j) = (v_j, v_i) \in E$ joins two nodes $v_i \in V$ and $v_j \in V$. Let $N = |V|$ denote the *number of nodes*. Let $A \in \{0,1\}^{N \times N}$ denote the *adjacency matrix* of G, where for two nodes v_i and v_j, $A_{ij} = 1$ if and only if $(v_i, v_j) \in E$, i.e., v_i and v_j are joined by an edge. The *degree matrix* of G is $D = \text{diag}(A\mathbf{1}_N)$, where each diagonal entry $D_{ii} = d_i = \sum_{i'=1}^{N} A_{ii'}$ is the *degree* of node v_i.

We assume that an attribute vector of dimension F is given for each node, and we use $X \in \mathbb{R}^{N \times F}$ to denote the corresponding *node attribute matrix*, where the i-th row of X, denoted by X_i, is the node attribute vector of node v_i.

Let K be the *number of clusters*. We call a matrix $P \in [0,1]^{N \times K}$ a (soft) *cluster assignment matrix* if $\sum_{x=1}^{K} P_{ix} = 1, \forall 1 \leq i \leq N$, where each element P_{ix} can be interpreted as the probability that we assign node v_i to cluster x. If $P \in \{0,1\}^{N \times K}$ further holds, then we call P a *deterministic* cluster assignment matrix. Let $\mathcal{P}$ be the set of all (soft) cluster assignment matrices and $\mathcal{P}^* \subseteq \mathcal{P}$ be the set of all deterministic ones. We list frequently-used symbols in Table 1.

[1] For simplicity and due to the nature of the datasets, we focus on unweighted and undirected graphs. Our method is easily extended to weighted and/or directed graphs.

2.2 Graph Clustering & Quality Functions

Given a graph, the goal of *graph clustering* is to divide the nodes into disjoint and exhaustive (i.e., every node is assigned to one group) groups (namely, *clusters*) so that nodes in the same group are more similar to each other than to those in different groups. Many algorithms have been developed for the problem, and they can be categorized largely into partitioning methods [24], agglomerative methods [5], divisive methods [17], and spectral methods [41].

Several measures, including normalized cut [58] and modularity [40], have been used to measure the structural quality (the homogeneity within-cluster nodes and/or the dissimilarity between the cross-cluster nodes) of a given clustering, and they also have been used as objectives for a specific formulation of optimization problems. Especially, a number of approaches [5, 32, 39] directly aim to maximize modularity.

2.3 Meta-weighting

Meta-weighting is a method of learning the weights of training samples while minimizing an objective function based on meta-learning. The weights usually represent the different importance of samples, and they are useful for alleviating class imbalance and reducing the noise in labels. The weight of each training sample is obtained by a meta-model, whose parameters are optimized using a small amount of high-quality data without biases and noises along with the learning process of the main model. Recently, meta-weighting-based schemes outperform traditional rule-based weighting schemes [11, 37] in various tasks, including image classification and recommendation [25, 26, 47, 52].

Our contributions in the context of meta-weighting. To the best of our knowledge, we are first to apply meta-weighting to graph learning. It should be noted, however, that the above techniques are not directly applicable to graph-level tasks (e.g., graph clustering) since (a) a graph is not naturally divided into independent samples, and (b) which part of a graph is of high quality is typically unknown. Regarding (a), rather than decomposing data into components, we suggest a novel idea of decomposing a loss function that satisfies the conditions in Definition 3.1. Regarding (b), our study demonstrates that, despite the presence of noise, a meta-model can be effectively trained by using simply distinct batches for it and the clustering model. That is, we show that a noise-free validation dataset is not mandatory, at least in the context of our problem.

2.4 GNN-Based Graph Clustering Methods

Recently, several approaches based on graph neural networks (GNNs) have been proposed for graph clustering [4, 16, 53]. Those approaches are particularly effective when abundant node attributes are given, as GNNs learn to combine node attributes and graph topology for a considered task through end-to-end training.

MinCutPool [4] uses a graph convolutional network (GCN) followed by a multi-layer perceptron (MLP) and softmax activation. Its output is a (soft) cluster assignment matrix P (see Sec. 2.1) that is relaxed to be continuous for end-to-end trainability. The objective

function consists of a continuous relaxation of the normalized cut objective. DMoN [53] employs a similar architecture while it uses a continuous relaxation of modularity (see Sec. 2.2) instead of normalized cut. GCC [16] leverages GCN and the k-means clustering loss [35] to perform node embedding and clustering simultaneously.

2.5 Graph Denoising and Robust GNNs

GNNs in general are vulnerable to noise edges in graphs because message passing, which is the basic operation of GNNs, is performed not only through valid edges but also through noise edges. Thus, several methods have been proposed to improve the robustness of GNNs by removing potential noise edges from an input graph. GCN-Jaccard [57] removes an edge if the Jaccard similarity between the node attributes of its two endpoints is below a predetermined threshold. GCN-SVD [13] uses a low-rank approximation of the given adjacency matrix instead of the original adjacency matrix. ProGNN [21] further assumes the sparsity of the denoised adjacency matrix. Specifically, in ProGNN, the denoised adjacency matrix and GNN parameters are learned end to end to minimize jointly (a) a classification loss, (b) the ℓ_1 norm (for sparsity) and nuclear norm (for low-rankness) of the adjacency matrix, and (c) the difference between attributes of adjacent nodes. GDC [30] creates edges between nodes with high proximity (measured by heat kernel and personalized PageRank) and uses them, and uses such edges, instead of the original edges, for message passing in GNNs. PTDNet [36] produces a denoised adjacency matrix through a parameterized denoising network. The denoised adjacency matrix is subsequently optimized jointly with downstream-task models. FGC [22] employs spectral clustering for graph clustering by generating a node-similarity matrix. This matrix is obtained through a learning process that minimizes a loss function based on both node proximity and filtered attributes, where the filtered attributes are obtained through a graph Laplacian filter. All these methods, except for FGC, yield only a denoised adjacency matrix without providing clustering results. Therefore, on top of these methods, except for FGC, we apply DMoN (see Sec. 2.4) to obtain clustering results so that they can be directly compared with our proposed method for the purpose of clustering (see Sec. 4.2).

Another related line of research has focused on enhancing the robustness of GNNs by modifying the message-passing schemes without explicit graph denoising. However, they are designed specifically for node classification [14, 60, 61].

3 PROPOSED METHOD: METAGC

In this section, we introduce our proposed method, MetaGC (**Meta**-weighting based **G**raph **C**lustering), for robust GNN-based graph clustering. As shown in Figure 1, MetaGC consists of a GNN-based clustering model C (see Sec. 3.1) and a meta-model M (see Def. 3.1 and Sec. 3.2), where M adjusts the weights of node pairs in a decomposable clustering loss function (see Sec. 3.3) that is used to train C. The key idea of MetaGC is to let M learn to properly adjust the weights of the node pairs (spec., to lower the weights of noise edges) so that the clustering performance of C becomes robust.

Below, we first introduce the detailed structures of both models (C and M). Then, we discuss the details of the objective function with theoretical analyses. Lastly, we describe the training process.

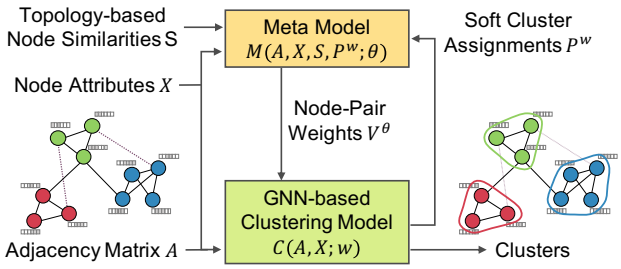


Figure 1: Overall procedure of MetaGC. MetaGC consists of a GNN-based clustering model C and a meta-model M. The meta-model M adjusts the weights of the terms in the modularity-based loss function, which is used to train C.

3.1 GNN-based Clustering Model

Given an adjacency matrix A, a node attribute matrix X, and the number of clusters K, the clustering model C outputs a (soft) cluster assignment matrix $P^{\boldsymbol{w}} \in [0,1]^{N\times K}$, where N is the number of nodes. The clustering model C consists of a GNN, a multilayer perceptron (MLP), and a softmax activation; and C is parameterized by $\boldsymbol{w}$. Specifically,

$$P^{\boldsymbol{w}} = C(A, X, K; \boldsymbol{w}) = \text{Softmax}(\text{MLP}(\text{GNN}(A, X))). \tag{1}$$

While any GNN models can be used for this purpose, we use a variant with skip connections of GCN [28]. Formally, each GNN layer is formulated as the following transformation:

$$H^{(i+1)} = \sigma_C(D^{-1/2}AD^{-1/2}H^{(i)}W_1 + H^{(i)}W_2),$$

where W_1 and W_2 are learnable parameters, D is the degree matrix (see Sec. 2.1), σ_C is an activation function (spec., SELU [29]), and $H^{(i)}$ for each $i > 1$ is the output of the i-th GNN layer and $H^{(0)} = X$. We use $P^{\boldsymbol{w}}$ to compute a *decomposable* clustering loss function $L^{\boldsymbol{w}}$. We give the definition of decomposable clustering loss functions first, and the detailed theoretical analyses are deferred to Sec. 3.3.

Definition 3.1 (Decomposable clustering loss functions). Given $G = (V, E)$ with $|V| = N$, $K \in \mathbb{N}$, and a (soft) cluster assignment matrix $P \in \mathcal{P}$, a clustering loss function $f : \mathcal{P} \to \mathbb{R}$ is **decomposable**, if there exist constants $c_{ij} = c_{ij}(G), \forall i, j \in [N]$ s.t.

$$f(P) = \sum_{i=1}^{N}\sum_{j=1}^{N} c_{ij} P_i \cdot P_j, \tag{2}$$

where P_i denotes the i-th row of P.[2]

3.2 Meta-Model

The inputs of the meta-model M are (a) the adjacency matrix A, (b) the node attribute matrix X, (c) the topology-based node-similarity matrix S (spec., we use the Adamic-Adar indices [1]), and (d) the soft cluster assignment matrix $P^{\boldsymbol{w}}$ (i.e., the output of C); and M outputs the node-pair weight matrix $V^{\boldsymbol{\theta}} \in \mathbb{R}^{N\times N}_{>0}$ for weighting the terms in the final loss function (i.e., Eq. (7) in Sec. 3.3). The parameters of M are denoted by $\boldsymbol{\theta}$. Specifically,

$$\begin{aligned} V^{\boldsymbol{\theta}} &= M(A, X, S, P^{\boldsymbol{w}}; \boldsymbol{\theta}) \\ &= \sum_{r=1}^{R} \alpha^{(r)} \cdot \sigma_M\left((Z^{(r)}(Z^{(r)})^T) \odot Y^{(r)}\right), \end{aligned} \tag{3}$$

[2] WLOG, we assume f does not contain constant terms.

where $\odot$ denotes the element-wise product, σ_M is an activation function (spec., we use sigmoid), and R is the number of the features of node pairs. For each $1 \leq r \leq R$, $Y^{(r)} \in \mathbb{R}^{N\times N}$ is a feature matrix for node pairs, the scalar $\alpha^{(r)}$ is the learnable weight[3] for $Y^{(r)}$, and the symmetric matrix $Z^{(r)}(Z^{(r)})^T \in \mathbb{R}^{N\times N}$ with $Z^{(r)} = \text{MLP}^{(r)}(X)$ is used as the attention matrix for $Y^{(r)}$. While any features of node pairs can be employed, after a preliminary study, we use the following three features (i.e., $R = 3$) for each node pair (v_i, v_j): (a) if $A_{ij} = 0$, $(Y^{(1)}_{ij}, Y^{(2)}_{ij}, Y^{(3)}_{ij}) = (1, 0, 0)$; (b) if $A_{ij} = 1$, $(Y^{(1)}_{ij}, Y^{(2)}_{ij}, Y^{(3)}_{ij}) = (1, S_{ij}, L^{\boldsymbol{w}}_{ij})$, where S is the topology-based node-similarity matrix, and $L^{\boldsymbol{w}}_{ij}$ is a decomposable clustering loss function computed using $P^{\boldsymbol{w}}$ (i.e., the output of C), whose details with theoretical analyses will be provided in Sec. 3.3.

3.3 Learning Objective

The objective function of MetaGC includes a *decomposable* clustering loss function (see Def. 3.1). We shall show that decomposable clustering loss functions are *expectation-conforming*, which is a desirable property for continuous relaxation needed by GNN-based end-to-end learning. Given a soft clustering assignment matrix P, we can interpret each entry P_{ix} as the probability that we assign node v_i to the cluster x. Thus, we may see P as a random variable taking values in $\mathcal{P}^*$ (see Sec. 2.1) and the probability that P corresponds to a deterministic $P^* \in \mathcal{P}^*$ is

$$\mathbb{P}(P = P^*) = \prod_{i=1}^{n} P_{i,S(i;P^*)}, \tag{4}$$

where $S(i; P^*)$ is the cluster to which P^* assigns v_i (i.e., $P^*_{ix} = 1$ if and only if $x = S(i; P^*)$). Using such a perspective, we first give the formal definitions of expectation-conforming and decomposable clustering loss functions.

Definition 3.2 (Expectation-conforming clustering loss functions). Given $G = (V, E)$ with $|V| = N$, $K \in \mathbb{N}$, and a (soft) cluster assignment matrix $P \in \mathcal{P}$, a clustering loss function $f : \mathcal{P} \to \mathbb{R}$ is **expectation-conforming**, if

$$f(P) = \sum\nolimits_{P^*\in\mathcal{P}^*} \mathbb{P}(P = P^*) f(P^*), \forall P \in \mathcal{P}. \tag{5}$$

The intuition is that, if f satisfies Eq. (5), then for each (soft) cluster assignment matrix P, the value of f directly taken on P is equal to the *expectation* of the value of f when we see P as a random variable taking values in $\mathcal{P}^*$.

Lemma 3.3. *If $f : \mathcal{P} \to \mathbb{R}$ is expectation-conforming, then (1) $\min_{P\in\mathcal{P}} f(P) = \min_{P^*\in\mathcal{P}^*} f(P^*)$, and (2) for each $P \in \arg\min_{P\in\mathcal{P}} f(P)$, $\mathbb{P}(P = P^*) > 0 \Rightarrow P^* \in \arg\min_{P^*\in\mathcal{P}^*} f(P^*)$.*

Proof. Since $\mathcal{P}^* \subseteq \mathcal{P}$, it is trivial that $\min_{P\in\mathcal{P}} f(P) \leq \min_{P^*\in\mathcal{P}^*} f(P^*)$. On the other hand, since for each $P \in \mathcal{P}$, $\sum_{P^*\in\mathcal{P}^*} \mathbb{P}(P = P^*) = 1$, $f(P) = \sum_{P^*\in\mathcal{P}^*} \mathbb{P}(P = P^*) f(P^*) \geq \min_{P^*\in\mathcal{P}^*} f(P^*)$, completing the proof for (1).

Regarding (2), suppose the opposite, i.e., $\exists P \in \arg\min_{P\in\mathcal{P}} f(P)$, $P^*_0 \in \mathcal{P}^* \notin \arg\min_{P^*\in\mathcal{P}^*} f(P^*), \mathbb{P}(P = P^*) > 0$, then it is easy to see that $f(P) = \sum_{P^*\in\mathcal{P}^*} \mathbb{P}(P = P^*) f(P^*) > \min_{P^*\in\mathcal{P}^*} f(P^*)$, which, combined with (1), completes the proof. □

[3]The weights are normalized so that $\sum_{r=1}^{R} \alpha^{(r)} = 1$ holds.

Why is being expectation-conforming desirable? When we use an expectation-conforming clustering loss function f, if we reach a global minimum after training, then it is guaranteed that we can obtain an optimal solution to the graph clustering problem (i.e., an optimal deterministic clustering assignment) from the trained (soft) clustering assignment matrix.

Lemma 3.4. *If $f : \mathcal{P} \to \mathbb{R}$ is decomposable, then f is expectation-conforming.*

Proof. By Def. 3.1, Eq. 4, and linearity of expectation, it suffices to show that each term of f is expectation-conforming. For the $P_i \cdot P_j$ terms, $\sum_{P^*\in\mathcal{P}^*} \mathbb{P}(P = P^*) P^*_i \cdot P^*_j = \sum_{P^*_i, P^*_j} \mathbb{P}(P_i = P^*_i)\mathbb{P}(P_j = P^*_j) P^*_i \cdot P^*_j = \sum_x \mathbb{P}(P_i = P^*_i, P^*_{ix} = 1)\mathbb{P}(P_j = P^*_j, P^*_{jx} = 1) = \sum_x P_{ix}P_{jx} = P_i \cdot P_j$, where we have used the independence between P_i and P_j, for all $i \neq j$, completing the proof. □

Specifically, in MetaGC, we use a modularity-based loss function (Eq. (6)).[4] As mentioned before, modularity [40] is a representative objective for clustering. We use the output $P^{\boldsymbol{w}}$ of the clustering model C (see Sec. 3.1) to compute the loss function. Formally,

$$\tilde{Q}(\boldsymbol{w}) = \sum\nolimits_{i,j=1}^{N} \underbrace{\frac{1}{2|E|}\left(A_{ij} - \frac{d_i d_j}{2|E|}\right) P^{\boldsymbol{w}}_i \cdot P^{\boldsymbol{w}}_j}_{=\tilde{Q}_{ij}(\boldsymbol{w})}, \tag{6}$$

where we have decomposed modularity over node pairs. Note that, $\tilde{Q}(\boldsymbol{w})$ is one specific example of decomposable clustering loss functions, which are denoted above by $L^{\boldsymbol{w}}$.

Lemma 3.5. *The function $\tilde{Q}$ is decomposable, and thus it is expectation-conforming.*

Proof of Lem. 3.5. $\tilde{Q}$ is decomposable with $c_{ij} = \frac{1}{2|E|}(A_{ij} - \frac{d_i d_j}{2|E|}), \forall i, j \in [N]$. □

Remark. Due to the NP-hardness of modularity optimization [6], finding a globally optimal solution is non-trivial in general, and we are not claiming that using an expectation-conforming function makes it easier to reach an optimal solution. Instead, we are claiming that applying a continuous extension to an expectation-conforming function does not introduce any "bad" minima [23] (minima of the continuous extension but not the original problem).

In contrast, the normalized cut [4, 58] is not decomposable nor expectation-conforming in general.

Lemma 3.6. *In general,* $\text{NC} = \text{NC}(P) = -\text{Tr}(P^T\tilde{A}P)/\text{Tr}(P^T\tilde{D}P)$ *is not expectation-conforming, and thus it is not decomposable either.*

Proof. We expand NC as $\text{NC}(P) = -\frac{\sum_{i,j}(P_i \cdot P_j)\tilde{A}_{ij}}{\sum_i (P_i \cdot P_i)\tilde{D}_{ii}}$. Consider a simple case where G consists of two nodes and one edge between them with $K = 2$. We have $\text{NC}(P) = -\frac{2P_1 \cdot P_2}{P_1 \cdot P_1 + P_2 \cdot P_2} \neq \sum_{P^*\in\mathcal{P}^*} \mathbb{P}(P = P^*)\,\text{NC}(P^*) = P_{11}P_{21}\,\text{NC}(\left(\begin{smallmatrix}1&0\\1&0\end{smallmatrix}\right)) + P_{12}P_{22}\,\text{NC}(\left(\begin{smallmatrix}0&1\\0&1\end{smallmatrix}\right)) = -P_{11}P_{21} - P_{12}P_{22} = -P_1 \cdot P_2$. The counterexamples when G has more nodes or edges can be constructed similarly. □

[4]Modularity measures how much a cluster's edge count (A_{ij}'s) exceeds the expected edge count ($d_i d_j / 2|E|$'s) under a null model preserving the degree distribution.

Algorithm 1: Learning Algorithm of MetaGC

input : (a) graph $G = (V, E)$, (b) node attribute matrix X, (c) number of clusters K, (d) number of epochs T, (e) number of batches b
output : clustering model parameter $\boldsymbol{w}$

Initialize clustering/meta-model parameter $\boldsymbol{w}/\boldsymbol{\theta}$
for *epoch* $= 1$ *to* T **do**
  for $itr = 1$ *to* N/b **do**
    $B_C, B_M \leftarrow$ two disjoint subsets of V of size b sampled uniformly at random
    // Meta-model update
    $\boldsymbol{w}' \leftarrow \boldsymbol{w} - \eta \sum_{v_i \in B_C} \nabla_{\boldsymbol{w}} L_i(\boldsymbol{w}, \boldsymbol{\theta})$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \mu \sum_{v_i \in B_M} \nabla_{\boldsymbol{\theta}} \tilde{Q}_i(\boldsymbol{w}')$
    // Clustering model update
    $\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \sum_{v_i \in B_C} \nabla_{\boldsymbol{w}} L_i(\boldsymbol{w}, \boldsymbol{\theta})$
  end
end
return $\boldsymbol{w}$

In order to integrate the meta-weighting strategy into our decomposable clustering loss function, the loss at each node pair v_i and v_j is weighted by $V_{ij}^{\boldsymbol{\theta}} \in \mathbb{R}_{>0}$ produced by the meta-model M (see Sec. 3.2). Formally, the final loss function is

$$L(\boldsymbol{w}, \boldsymbol{\theta}) = \left(\sum_{i,j=1}^{N} V_{ij}^{\boldsymbol{\theta}} L_{ij}^{\boldsymbol{w}} \right) + \lambda R^{\boldsymbol{w}}, \tag{7}$$

where λ denotes the regularization rate, and $R^{\boldsymbol{w}} = \frac{\sqrt{K}}{N} || \sum_{i=1}^{N} P_i^{\boldsymbol{w}} ||_F - 1$ is a collapse regularization term introduced in [53]. For ease of explanation, we also define the loss at each node as follows:

$$L_i(\boldsymbol{w}, \boldsymbol{\theta}) = \left(\sum_{j=1}^{N} V_{ij}^{\boldsymbol{\theta}} L_{ij}^{\boldsymbol{w}} \right) + \frac{\lambda}{N} R^{\boldsymbol{w}}. \tag{8}$$

3.4 Overall Training Procedure

In Alg. 1, we provide the pseudocode of the training process, where we alternatingly optimize the meta-model and the clustering model, following the general procedure of meta-weighting [47, 52].

While the general procedure requires noise-free validation data, it is typically unknown which part of the input graph is noise-free. Thus, without such requirements, MetaGC employs distinct batches (i.e., subsets) of nodes and their corresponding incident edges for training the meta-model and the clustering model. This approach is founded on the belief that the input graph contains sufficient information to effectively train the meta-model, even in the presence of noise edges, and this belief is validated in Sec. 4.2.

Meta-model update. In each training step t, in order to update the parameters $\boldsymbol{\theta}_t$ of the meta-model M, the parameters $\boldsymbol{w}_t$ of clustering model C are employed. We first update $\boldsymbol{w}_t$ once using the weighted loss function (i.e., Eq. (7)) using a batch C_t of nodes as follows:

$$\boldsymbol{w}_t' \leftarrow \boldsymbol{w}_t - \eta \sum_{v_i \in C_t} \nabla_{\boldsymbol{w}} L_i(\boldsymbol{w}, \boldsymbol{\theta}_t) \big|_{\boldsymbol{w}=\boldsymbol{w}_t}, \tag{9}$$

where η is the learning rate for C.

Using the updated parameters $\boldsymbol{w}'$, we update the parameters $\boldsymbol{\theta}_t$ of the meta-model using another batch M_t of nodes and the unweighted loss function:[5]

$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \mu \sum_{v_i \in M_t} \nabla_{\boldsymbol{\theta}} \tilde{Q}_i(\boldsymbol{w}_t') \big|_{\boldsymbol{\theta}=\boldsymbol{\theta}_t}, \tag{10}$$

where μ is M's learning rate, and $\tilde{Q}_i = \sum_j \tilde{Q}_{ij}$ (see Eq. (6)).

Clustering model update. In each training step t, the clustering model C is updated after the meta-model update. The parameters $\boldsymbol{w}_t$ of C are updated using the weighted loss function (i.e., Eq. (7)) with a batch C_t of nodes and the meta-model M with its updated parameters $\boldsymbol{\theta}_{t+1}$ as follows:

$$\boldsymbol{w}_{t+1} \leftarrow \boldsymbol{w}_t - \eta \sum_{v_i \in C_t} \nabla_{\boldsymbol{w}} L_i(\boldsymbol{w}, \boldsymbol{\theta}_{t+1}) \big|_{\boldsymbol{w}=\boldsymbol{w}_t}. \tag{11}$$

Table 2: Summary of the real-world datasets

Name	# Nodes	# Edges	# Attributes	# Classes
Cora	2,708	5,278	1,433	7
Cora-ML	2,995	8,158	2,879	7
Citeseer	3,327	4,552	3,703	6
Amazon-Photo	7,535	119,081	745	8
Pubmed	19,717	44,324	500	3

4 EXPERIMENTS

In this section, we evaluate MetaGC to answer the Q1-Q3:

Q1. **Robustness & Accuracy**: Is MetaGC more robust and accurate than the competitors on noisy graphs?
Q2. **Effectiveness of Meta-Weighting**: Does the meta-model in MetaGC properly adjust the weights of loss terms?
Q3. **Ablation Study**: Does each component of MetaGC contribute to performance improvement?

4.1 Experiment Settings

Hardware. We run all experiments on a workstation with an Intel Xeon 4214 CPU, 512GB RAM, and RTX 8000 GPUs.

Datasets. We use 5 real-world datasets: 4 citation graphs (Cora, Cora-ML, Citeseer, Pubmed [49]) and a co-purchase graph (Amazon-Photo [50]). For all graphs, self-loops are ignored. Some basic statistics of the graphs are provided in Table 2.

Noisy-graph Generation. For each dataset, we generate noise edges that do not exist in the original graph and add them to the graph. Specifically, the noise edges are chosen uniformly at random among those whose endpoints belong to different classes. Noise levels I, II and III indicate that the ratio between the number of noise edges and that of the existing edges are 30%, 60%, and 90%, respectively. For each dataset and each noisy level, we generate five noisy graphs, and all experimental results are averaged over them.

Competitors. We compare MetaGC with 10 competitors that are divided into three categories: (a) four *node-embedding-based* methods (DeepWalk [45], Node2Vec [18], DGI [55], and GMI [44]), whose outputs are clustered by K-means++ [54], (b) three *GNN-based graph clustering* methods (MinCutPool [4], DMoN [53], and GCC [16]) (see Sec. 2.4 for the details), and (c) six *graph denoising* methods (GCN-Jaccard [57], GCN-SVD [13], ProGNN [21], GDC [30], PTDNet [36], and FGC [22]).[6] All the *graph denoising*

[5] Note that the weighted loss function (7) cannot be used to update $\boldsymbol{\theta}$ since it is trivially minimized when each $V_{ij}^{\boldsymbol{\theta}} = 0$.
[6] Technically speaking, GDC and FGC are graph augmentation methods.

methods, except for FGC, are combined with DMoN for graph clustering. Note that DMoN aims to maximize a continuous relaxation of modularity, which is used as an evaluation metric.

Some modifications are needed for the graph denoising methods so that they can be used for graph clustering. For GCN-SVD and GDC, which generate weighted graphs, we have observed that applying DMoN directly on weighted graphs impairs the performance of graph clustering. Therefore, we convert each generated weighted graph to an unweighted one by selecting the $|E|$ edges with the largest weights, where $|E|$ is the number of edges in the original graph. For ProGNN and PTDNet (see Sec. 2.5 for more details), we replace their classification loss with our clustering loss (see Eq. (6)).

For MetaGC and all its competitors, we use the ground-truth number of classes in each graph as the target number of clusters K. See Apppendix A for more hyperparameter settings.

Evaluation Metrics. To evaluate MetaGC and the competitors, we use both topology- and correlation-based metrics. As a topology-based metric, we use the modularity in the original graph without injected noises. As correlation-based metrics, we use the pairwise F1 Score (F1 Score) and the normalized mutual information (NMI) between the given classes of nodes and the cluster assignments of nodes. For each output (soft) cluster assignment matrix of the methods, we apply each of the evaluation metrics after converting it to a deterministic one by setting the maximum value of each row to 1 and the rest to 0. Notably, using each of the metrics alone, there exist cases where a higher value may not necessarily mean more meaningful clustering [9, 31, 34]. We also observe some specific cases in our experiments (see Sec. 4.2). Therefore, it is important to take all of them into consideration.

On each noisy graph, three independent trials are conducted, and thus for each dataset and each noise level, we report the average results over the 15 trials (5 noisy graphs × 3 independent trials).

4.2 Q1. Robustness & Accuracy

In Table 3, for each dataset, each noise level, each method, and each evaluation metric, we report the mean and standard deviation of the results over 15 trials.

For each dataset and each method, we also report the average rank (AR) over all the evaluation metrics. With regard to AR, tests of statistical significance are performed between the proposed method MetaGC and each competitor over 15 independent random trials. Specifically, for each competitor, the null hypothesis is that there is no significant superiority of MetaGC over the competitor w.r.t AR. One-tailed t-tests are employed to ascertain whether the AR of MetaGC is significantly better than that of each competitor. The results of the tests are reported as follows: * means $p < 0.05$, ** means $p < 0.01$, and *** means $p < 0.001$.

First of all, MetaGC performs best overall and ranks first in every dataset w.r.t the average rank. Specifically, MetaGC achieves an average rank of 1.2 to 3.3 among all the 14 considered methods on the five datasets. Below, we discuss the results in detail.

Since DeepWalk and Node2Vec use only graph topology without node attributes, their performance w.r.t modularity, which is a topology-based metric, is favorable. However, they are much less competitive than MetaGC w.r.t F1 Score and NMI, which are correlated-based metrics.

In some cases, DGI and GMI perform better than MetaGC w.r.t NMI, but in most cases, they perform significantly worse w.r.t the other two metrics, especially F1 Score. Moreover, their performance decreases on large graphs (Amazon-Photo and Pubmed).

Compared to the two GNN-based graph clustering methods *without meta-weighting* (MinCutPool, DMoN, and GCC), the performance superiority of MetaGC becomes more significant as the noise level increases, showing that meta-weighting indeed enhances the robustness (see Sec. 4.3 for more discussions on the effectiveness of meta-weighting). In some cases, MinCutPool and GCC output only one cluster that contains all nodes, which results in a high yet meaningless F1 Score (see the low NMI and modularity).

Recall that each of the graph denoising methods except FGC is actually DMoN with a separate or combined denoising process (see Sec. 4.1), and DMoN aims to maximize a continuous relaxation of modularity, which is used as an evaluation metric. MetaGC outperforms all of them w.r.t the average rank on every dataset. Notably, even though GCN-SVD and GDC use the ground-truth number of the edges in the original noise-free graph, which is unknown to MetaGC, they perform consistently worse than MetaGC in all respects. GCN-Jaccard sometimes shows slightly better performance than MetaGC w.r.t F1 Score, but MetaGC performs better w.r.t the other two metrics in almost all the other cases (except for the performance w.r.t modularity on Pubmed in noise level I). ProGNN and PTDNet both have a clear drawback compared to MetaGC. Moreover, ProGNN is not scalable for large graphs due to its $O(N^3)$ time complexity. PTDNet is not scalable for large graphs either due to its space complexity associated with its parameterized denoising network, which generates a denoised adjacency matrix.

4.3 Q2. Effectiveness of Meta-Weighting

We shall show that the meta-model in MetaGC effectively distinguishes real edges and noise edges by assigning high weights to real edges and low weights to noise edges. We see this as a binary classification task, and in Table 4, for each dataset and each noise level, we report the PRAUC (the area under the Precision-Recall curve) and HITS@10% (i.e., the proportion of real edges among the top-10% edges with highest weights) value. We also include a baseline representing the expected value of PRAUC and HITS@10% when we randomly assign the weights, which is equal to the proportion of real edges among all the edges. We can see that both PRAUC and HITS@10% of MetaGC are consistently higher than the expected value. Moreover, HITS@10% values are consistently close to 1, which means that in all the cases, almost all top-10% edges with the highest weights assigned by the meta-model are real edges, as intended. In Figure 2, we present the detailed Precision-Recall curves at different noise levels for each dataset.

4.4 Q3. Ablation Study

We have just shown that the meta-model in MetaGC is effective in distinguishing real edges and noise edges. We further examine how much it affects the performance of MetaGC by comparing MetaGC with two variants of it:

- **MetaGC-X**: MetaGC without the meta-model, i.e., the weight V_{ij} of every pair (v_i, v_j) is the same.

Table 3: (Q1) Robustness & Accuracy. MetaGC shows the best overall performance. It ranks the first in all datasets w.r.t the average rank (AR). Statistical significance of the AR differences between MetaGC and each of its competitors is reported at the following levels: $^{*}p < 0.05$, $^{**}p < 0.01$, $^{***}p < 0.001$. O.O.T.: out of time (> 6 hours). O.O.M: out of (GPU) memory. For each setting (each column), the best, second-best and third-best results are in red, blue, and green, respectively.

Noise Level	I			II			III			AR
Metric	F1 Score	NMI	Modularity	F1 Score	NMI	Modularity	F1 Score	NMI	Modularity	
DeepWalk	0.405±0.048	0.465±0.008	0.689±0.006	0.297±0.022	0.389±0.010	0.659±0.007	0.256±0.014	0.352±0.012	0.641±0.007	6.0***
Node2Vec	0.410±0.043	0.464±0.006	0.690±0.005	0.296±0.023	0.389±0.008	0.660±0.004	0.261±0.017	0.359±0.011	0.642±0.007	5.4***
DGI	0.230±0.010	0.287±0.003	0.151±0.007	0.198±0.009	0.239±0.004	0.141±0.010	0.183±0.006	0.203±0.013	0.122±0.013	9.7***
GMI	0.099±0.004	0.021±0.001	-0.003±0.001	0.103±0.005	0.025±0.001	-0.002±0.001	0.109±0.006	0.030±0.001	-0.002±0.001	11.7***
MinCutPool	0.464±0.000	0.000±0.000	0.000±0.000	0.464±0.000	0.000±0.000	0.000±0.000	0.464±0.000	0.000±0.000	0.000±0.000	9.4***
DMoN	0.556±0.049	0.533±0.041	0.609±0.036	0.528±0.028	0.494±0.025	0.599±0.023	0.470±0.033	0.425±0.036	0.531±0.050	3.3***
GCC	0.538±0.022	0.501±0.039	0.619±0.034	0.469±0.007	0.377±0.019	0.540±0.027	0.459±0.006	0.353±0.018	0.526±0.024	5.9***
GCN-Jaccard	0.557±0.049	0.533±0.040	0.610±0.036	0.525±0.034	0.493±0.028	0.597±0.032	0.473±0.034	0.431±0.038	0.538±0.052	3.1***
GCN-SVD	0.390±0.004	0.365±0.009	0.497±0.002	0.408±0.005	0.379±0.004	0.506±0.006	0.403±0.005	0.374±0.017	0.507±0.011	7.4***
GDC	0.514±0.073	0.502±0.054	0.572±0.043	0.474±0.057	0.447±0.052	0.547±0.059	0.463±0.033	0.418±0.031	0.532±0.050	4.9***
ProGNN	O.O.T.	O.O.T.	O.O.T.	O.O.T.	O.O.T.	O.O.T.	O.O.T.	O.O.T.	O.O.T.	N.A.
PTDNet	O.O.M.	O.O.M.	O.O.M.	O.O.M.	O.O.M.	O.O.M.	O.O.M.	O.O.M.	O.O.M.	N.A.
FGC	0.377±0.000	0.071±0.001	0.145±0.003	0.366±0.000	0.055±0.000	0.103±0.001	0.362±0.000	0.048±0.000	0.084±0.001	9.6***
MetaGC	0.562±0.015	0.566±0.017	0.675±0.008	0.528±0.020	0.520±0.013	0.664±0.007	0.508±0.014	0.498±0.009	0.658±0.006	**1.2**

(a) Amazon-Photo

Noise Level	I			II			III			AR
Metric	F1 Score	NMI	Moduarity	F1 Score	NMI	Moduarity	F1 Score	NMI	Moduarity	
DeepWalk	0.300±0.024	0.243±0.010	0.680±0.009	0.216±0.010	0.155±0.006	0.593±0.011	0.169±0.014	0.111±0.009	0.528±0.008	8.3***
Node2Vec	0.292±0.028	0.247±0.015	0.684±0.009	0.210±0.016	0.154±0.010	0.594±0.009	0.170±0.009	0.111±0.011	0.528±0.009	8.3***
DGI	0.351±0.040	0.415±0.011	0.619±0.015	0.294±0.027	0.330±0.012	0.547±0.017	0.248±0.018	0.240±0.017	0.412±0.033	6.1***
GMI	0.277±0.023	0.319±0.008	0.576±0.010	0.226±0.016	0.229±0.005	0.496±0.007	0.152±0.016	0.145±0.012	0.391±0.020	9.7***
MinCutPool	0.265±0.035	0.222±0.023	0.614±0.012	0.217±0.027	0.147±0.019	0.556±0.012	0.219±0.097	0.086±0.039	0.436±0.172	10.1***
DMoN	0.400±0.023	0.343±0.015	0.661±0.012	0.355±0.023	0.280±0.013	0.620±0.013	0.326±0.016	0.231±0.016	0.576±0.012	4.6***
GCC	0.375±0.017	0.230±0.013	0.486±0.011	0.364±0.023	0.114±0.014	0.312±0.053	0.364±0.041	0.076±0.016	0.252±0.073	9.6***
GCN-Jaccard	0.415±0.022	0.364±0.017	0.661±0.014	0.369±0.030	0.310±0.014	0.627±0.013	0.348±0.030	0.276±0.017	0.602±0.016	2.7**
GCN-SVD	0.313±0.025	0.207±0.019	0.487±0.022	0.291±0.031	0.172±0.023	0.468±0.016	0.288±0.024	0.156±0.017	0.458±0.018	8.9***
GDC	0.298±0.030	0.218±0.021	0.577±0.020	0.266±0.027	0.183±0.017	0.555±0.011	0.269±0.010	0.175±0.016	0.540±0.015	8.1***
ProGNN	0.405±0.023	0.348±0.015	0.631±0.015	0.370±0.022	0.296±0.011	0.590±0.016	0.341±0.018	0.248±0.017	0.544±0.019	4.2***
PTDNet	0.198±0.014	0.033±0.010	0.300±0.011	0.186±0.010	0.031±0.005	0.279±0.007	0.209±0.018	0.025±0.003	0.256±0.009	13.6***
FGC	0.388±0.005	0.145±0.005	0.337±0.006	0.374±0.005	0.123±0.006	0.314±0.006	0.364±0.010	0.112±0.008	0.295±0.005	8.7***
MetaGC	0.413±0.030	0.379±0.027	0.696±0.010	0.372±0.028	0.320±0.023	0.660±0.015	0.348±0.028	0.282±0.021	0.628±0.018	**1.7**

(b) Cora

Noise Level	I			II			III			AR
Metric	F1 Score	NMI	Moduarity	F1 Score	NMI	Moduarity	F1 Score	NMI	Moduarity	
DeepWalk	0.375±0.009	0.276±0.013	0.636±0.016	0.314±0.010	0.201±0.012	0.581±0.009	0.250±0.018	0.147±0.011	0.535±0.010	6.4***
Node2Vec	0.377±0.007	0.282±0.006	0.644±0.005	0.308±0.010	0.199±0.011	0.584±0.006	0.263±0.015	0.152±0.010	0.540±0.008	5.8***
DGI	0.379±0.063	0.295±0.037	0.340±0.045	0.242±0.009	0.131±0.015	0.167±0.021	0.210±0.015	0.063±0.017	0.083±0.027	9.8***
GMI	0.366±0.018	0.268±0.011	0.395±0.013	0.259±0.012	0.144±0.007	0.234±0.017	0.201±0.023	0.062±0.005	0.115±0.008	10.1***
MinCutPool	0.271±0.026	0.200±0.019	0.592±0.018	0.278±0.103	0.105±0.054	0.398±0.236	0.437±0.067	0.012±0.026	0.035±0.124	9.3***
DMoN	0.340±0.026	0.289±0.025	0.661±0.016	0.314±0.020	0.237±0.023	0.630±0.016	0.291±0.018	0.204±0.019	0.600±0.016	4.8***
GCC	0.461±0.022	0.299±0.024	0.441±0.041	0.415±0.014	0.165±0.049	0.306±0.086	0.379±0.030	0.105±0.036	0.232±0.063	5.7***
GCN-Jaccard	0.358±0.033	0.283±0.033	0.600±0.015	0.323±0.025	0.232±0.025	0.569±0.011	0.295±0.020	0.200±0.019	0.538±0.017	5.4***
GCN-SVD	0.275±0.022	0.165±0.024	0.403±0.029	0.247±0.017	0.142±0.011	0.365±0.015	0.261±0.016	0.129±0.011	0.338±0.016	9.4***
GDC	0.267±0.019	0.159±0.016	0.475±0.019	0.230±0.020	0.102±0.012	0.366±0.013	0.190±0.024	0.060±0.014	0.285±0.009	11.0***
ProGNN	0.345±0.026	0.297±0.025	0.662±0.016	0.319±0.020	0.244±0.023	0.631±0.015	0.297±0.018	0.212±0.020	0.603±0.015	3.6***
PTDNet	0.235±0.045	0.058±0.007	0.288±0.007	0.216±0.055	0.046±0.006	0.249±0.012	0.271±0.041	0.044±0.006	0.244±0.013	11.9***
FGC	0.395±0.013	0.035±0.011	0.111±0.029	0.415±0.002	0.017±0.003	0.059±0.005	0.424±0.003	0.010±0.001	0.038±0.006	9.8***
MetaGC	0.380±0.034	0.337±0.024	0.683±0.014	0.333±0.026	0.282±0.015	0.656±0.015	0.319±0.021	0.255±0.014	0.639±0.015	**1.8**

(c) Cora-ML

(continues on the next page)

Noise Level	I			II			III			AR
Metric	F1 Score	NMI	Moduarity	F1 Score	NMI	Moduarity	F1 Score	NMI	Moduarity	
DeepWalk	0.128±0.004	0.089±0.003	0.650±0.004	0.103±0.004	0.053±0.003	0.586±0.005	0.086±0.004	0.037±0.002	0.545±0.003	7.3***
Node2Vec	0.127±0.004	0.089±0.003	0.650±0.003	0.101±0.005	0.053±0.002	0.587±0.005	0.085±0.004	0.037±0.003	0.545±0.005	7.6***
DGI	0.199±0.019	0.108±0.003	0.169±0.005	0.160±0.007	0.064±0.002	0.153±0.007	0.130±0.006	0.043±0.003	0.160±0.007	9.2***
GMI	0.159±0.009	0.117±0.002	0.154±0.003	0.115±0.010	0.072±0.001	0.135±0.002	0.116±0.006	0.049±0.001	0.115±0.006	9.3***
MinCutPool	0.380±0.005	0.131±0.005	0.512±0.010	0.345±0.017	0.097±0.009	0.487±0.011	0.325±0.020	0.079±0.011	0.472±0.016	5.9***
DMoN	0.406±0.005	0.161±0.004	0.542±0.001	0.377±0.008	0.125±0.011	0.518±0.003	0.346±0.019	0.090±0.023	0.497±0.010	3.7***
GCC	0.522±0.002	0.052±0.003	0.313±0.005	0.459±0.004	0.039±0.001	0.421±0.004	0.505±0.053	0.019±0.010	0.276±0.149	7.4***
GCN-Jaccard	0.407±0.005	0.163±0.005	0.542±0.001	0.377±0.008	0.126±0.011	0.518±0.003	0.346±0.020	0.090±0.023	0.498±0.010	3.2***
GCN-SVD	0.372±0.043	0.094±0.022	0.379±0.006	0.351±0.038	0.076±0.014	0.372±0.006	0.332±0.035	0.060±0.014	0.365±0.005	7.1***
GDC	0.360±0.022	0.113±0.013	0.481±0.010	0.351±0.006	0.097±0.005	0.477±0.004	0.327±0.023	0.071±0.022	0.475±0.011	6.2***
ProGNN	O.O.T.	O.O.T.	O.O.T.	O.O.T.	O.O.T.	O.O.T.	O.O.T.	O.O.T.	O.O.T.	N.A.
PTDNet	O.O.M.	O.O.M.	O.O.M.	O.O.M.	O.O.M.	O.O.M.	O.O.M.	O.O.M.	O.O.M.	N.A.
FGC	0.598±0.000	0.000±0.000	0.000±0.000	0.568±0.000	0.059±0.000	0.261±0.000	0.576±0.000	0.044±0.000	0.218±0.000	7.1***
MetaGC	0.414±0.009	0.175±0.010	0.540±0.001	0.396±0.004	0.160±0.004	0.523±0.001	0.380±0.003	0.141±0.004	0.513±0.001	2.6

(d) Pubmed

Noise Level	I			II			III			AR
Metric	F1 Score	NMI	Moduarity	F1 Score	NMI	Moduarity	F1 Score	NMI	Moduarity	
DeepWalk	0.177±0.013	0.083±0.005	0.741±0.006	0.136±0.012	0.054±0.007	0.656±0.010	0.112±0.014	0.044±0.004	0.596±0.007	9.7***
Node2Vec	0.180±0.011	0.084±0.005	0.740±0.006	0.146±0.013	0.057±0.004	0.661±0.007	0.116±0.011	0.041±0.005	0.596±0.011	9.2***
DGI	0.256±0.021	0.294±0.006	0.704±0.015	0.204±0.013	0.228±0.006	0.648±0.020	0.183±0.020	0.176±0.004	0.562±0.032	6.0***
GMI	0.248±0.016	0.292±0.006	0.606±0.011	0.210±0.011	0.238±0.004	0.547±0.008	0.185±0.009	0.199±0.006	0.486±0.015	7.1***
MinCutPool	0.267±0.034	0.157±0.024	0.677±0.012	0.350±0.130	0.067±0.050	0.385±0.287	0.435±0.136	0.021±0.037	0.154±0.256	8.6***
DMoN	0.346±0.024	0.182±0.017	0.665±0.011	0.308±0.012	0.139±0.009	0.624±0.008	0.283±0.009	0.112±0.006	0.591±0.009	6.6***
GCC	0.410±0.011	0.191±0.031	0.545±0.066	0.415±0.024	0.147±0.039	0.448±0.114	0.419±0.054	0.078±0.039	0.312±0.161	6.3***
GCN-Jaccard	0.369±0.031	0.209±0.018	0.676±0.009	0.337±0.011	0.171±0.008	0.643±0.004	0.306±0.012	0.139±0.007	0.612±0.007	4.0***
GCN-SVD	0.280±0.033	0.120±0.013	0.448±0.021	0.248±0.027	0.084±0.008	0.422±0.022	0.237±0.027	0.062±0.010	0.398±0.022	9.8***
GDC	0.257±0.026	0.117±0.015	0.548±0.021	0.231±0.015	0.096±0.012	0.530±0.013	0.232±0.018	0.089±0.012	0.529±0.014	9.4***
ProGNN	0.359±0.025	0.191±0.017	0.636±0.012	0.326±0.016	0.153±0.009	0.587±0.013	0.302±0.012	0.125±0.006	0.544±0.012	5.9***
PTDNet	0.278±0.029	0.048±0.004	0.344±0.007	0.277±0.044	0.036±0.014	0.317±0.019	0.293±0.037	0.056±0.025	0.301±0.018	11.3***
FGC	0.410±0.004	0.131±0.005	0.409±0.007	0.398±0.005	0.112±0.007	0.381±0.008	0.400±0.005	0.105±0.005	0.370±0.008	7.4***
MetaGC	0.363±0.017	0.230±0.013	0.707±0.007	0.330±0.025	0.194±0.021	0.677±0.012	0.289±0.017	0.151±0.013	0.640±0.009	3.3

(e) Citeseer

- **MetaGC-A**: MetaGC with the metal model using only node attributes, spec., $(Y_{ij}^{(1)}, Y_{ij}^{(2)}, Y_{ij}^{(3)}) = (1, 0, 0)$ for every pair (v_i, v_j).

In Table 5, we report the results of the ablation study on Citeseer. For each noise level, each metric, and each variant including the original MetaGC, we report the mean and standard deviation of the results of the 15 trials. In all the settings and in all respects, the original MetaGC performs best, and MetaGC-X without the meta-model performs worst, validating the performance boost of the meta-model in MetaGC. Moreover, the comparison between MetaGC and MetaGC-A shows that using the additional information on the topology-based node similarity and the soft cluster assignment matrix from the clustering model is helpful. Interestingly, meta-weighting is more helpful w.r.t F1 Score and NMI than w.r.t modularity, although the meta-model is updated using the modularity-based objective.

5 CONCLUSION

In this work, we propose MetaGC for robust GNN-based graph clustering against noise edges. MetaGC consists of a GNN-based clustering model using a decomposable loss function with theoretical justification, and a meta-model that adaptively adjusts the weights of node pairs in the loss function. In our extensive experiments on the five datasets under three levels of noise, MetaGC is robust against noise edges, achieving an average rank of 1.2 to 2.7 among all the 11 considered methods. We also demonstrate the effectiveness of the meta-model by showing that it (a) assigns high weights to real edges and low weights to noise edges and (b) leads to a performance boost, especially when it uses richer information.

Acknowledgements: This work was supported by Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. 2022-0-00871, Development of AI Autonomy and Knowledge Enhancement for AI Agent Collaboration) (No. 2022-0-00157, Robust, Fair, Extensible Data-Centric Continual Learning) (No. 2019-0-00075, Artificial Intelligence Graduate School Program (KAIST)).

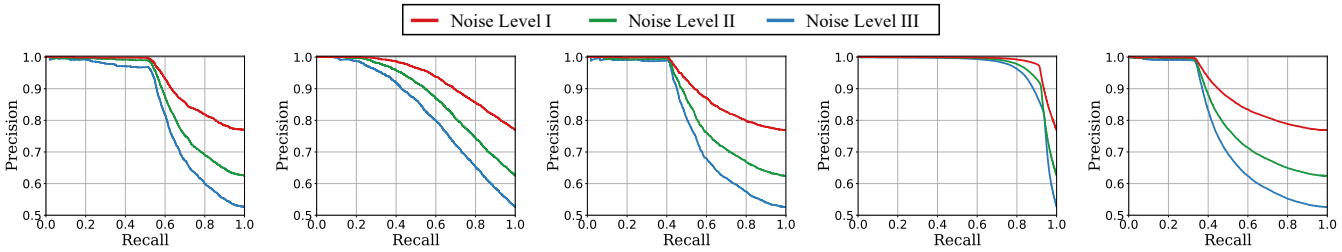
A APPENDIX: PARAMETER SETTINGS

In this section, we provide detailed parameter settings. For a fair comparison, we set the embedding dimension of all the considered methods (including MetaGC) to 64, unless otherwise stated.

MetaGC: MetaGC consists of a meta-model and a clustering model. The clustering model consists of a single-layer GCN with skip connections (see Sec. 3.2 of the main paper) and a single-layer perceptron. The number of hidden units is 64 in the GCN, and the single-layer perceptron outputs the final cluster assignment vector, whose length is equal to the number of clusters. In the meta-model, for each q, $H^{(q)}$ is obtained by a two-layer perceptron. We use ReLU activation [38] after the first layer, and we do not use any

Table 4: (Q2) Effectiveness of Meta-Weighting. MetaGC successfully assigns high weights to real edges and low weights to noise edges. The PRAUC and HITS@10% values are consistently and significantly higher than the baseline. The baseline is the expected value of PRAUC and HITS@10% when weights are randomly assigned.

Dataset	Cora			Cora-ML			Citeseer			Amazon-Photo			Pubmed		
Noise Level	I	II	III	I	II	III	I	II	III	I	II	III	I	II	III
PRAUC	0.927	0.875	0.831	0.934	0.878	0.825	0.908	0.843	0.793	0.993	0.985	0.976	0.890	0.813	0.757
HITS@10%	0.999	0.997	0.993	1.000	0.997	0.988	0.999	0.995	0.991	1.00	0.999	0.999	0.999	0.997	0.991
Baseline	0.769	0.625	0.526	0.769	0.625	0.526	0.769	0.625	0.526	0.769	0.625	0.526	0.769	0.625	0.526



(a) Cora (b) Cora-ML (c) Citeseer (d) Amazon-Photo (e) Pubmed

Figure 2: Detailed Precision-Recall curves used for Table 4.

Table 5: (Q3) Ablation Study. Meta-weighting in MetaGC improves the performance (compare MetaGC and MetaGC-X). Using the topology-based node similarity and the output of the clustering model is also helpful (compare MetaGC and MetaGC-A).

Noise Level	I			II			III		
Metric	F1 Score	NMI	Modularity	F1 Score	NMI	Modularity	F1 Score	NMI	Moduarity
MetaGC-X	0.340±0.022	0.203±0.016	0.695±0.006	0.308±0.017	0.173±0.014	0.662±0.007	0.280±0.018	0.142±0.013	0.634±0.009
MetaGC-A	0.346±0.020	0.214±0.014	0.701±0.007	0.324±0.019	0.187±0.016	0.674±0.011	0.288±0.017	0.150±0.012	0.638±0.009
MetaGC	**0.363±0.017**	**0.230±0.013**	**0.707±0.007**	**0.330±0.025**	**0.194±0.021**	**0.677±0.012**	**0.289±0.017**	**0.151±0.013**	**0.640±0.009**

activation after the second layer. We train MetaGC with the Adam optimizer [27]. The learning rates of the clustering model and the meta-model are fine-tuned via a grid search, where the range of both learning rates is $\{5e-4, 1e-3, 2e-3, 3e-3, 4e-3, 5e-3\}$. Similarly, the batch sizes of the two models are fine-tuned via a grid search, where the range of both batch sizes is $\{128, 256, 512, 1024, 2048\}$. MetaGC is trained with at least 200 epochs and at most 1500 epochs. We terminate the training if the modularity is not improved in the last 50 epochs, and use the parameters giving the highest modularity during the training.

Node embedding-based methods: For node-embedding-based methods, DGI, GMI, DeepWalk, and Node2Vec, we cluster their output embeddings using K-means++ [54], for which we set the maximum number of iterations to 300 and the tolerance to $1e-4$. For DGI and GMI, we increase the embedding dimension to 512,[7] as in the original papers; and we use the hyperparameters in the source code released by the authors. For DeepWalk and Node2Vec, we set the number of walks as 80, the length of the walks 80, and the window size 10.

GNN-based graph clustering methods: We compare MetaGC with three GNN-based graph clustering methods: MinCutPool, DMoN and GCC. MinCutPool and DMoN consist of a single-layer GCN with skip connections and a single-layer perceptron, as MetaGC does. We use ELU activation [10] in MinCutPool and SELU activation in DMoN (as in MetaGC), following the original papers. Moreover, following the original paper, in DMoN, we use a dropout layer before the softmax operation. The dropout ratio is set as 0.5. MinCutPool and DMoN are trained with at least 2000 epochs and at most 4000 epochs. We terminate the training if the modularity is not improved in the last 100 epochs, and use the parameters giving the highest modularity during the training. For GCC, we set the maximum number of optimization iterations as 30 and the tolerance $1e-7$. The propagation order is fine-tuned exhaustively in the range from 1 to 150.

Graph denoising methods: We compare MetaGC with four graph denoising methods: GCN-Jaccard, GCN-SVD, GDC, and ProGNN. GCN-Jaccard removes each edge such that the Jaccard similarity between the attributes of the two endpoints is 0. GCN-SVD generates a rank-100 approximation of the adjacency matrix, and uses such an approximation instead of the original matrix. In GDC, we use personalized PageRank [42], which performs best in the original paper. For ProGNN and PTDNet (see Sec. 2.5 for more details), we replace their GNN models with the clustering model employed by MetaGC and replace their classification loss with our clustering loss (see Eq. (6)). The hyperparameters of ProGNN and PTDNet are fine-tuned within the range specified in the source code provided by the authors. For GCN-SVD and GDC, which generate weighted graphs, we observe that applying DMoN directly on weighted graphs impairs the performance of graph clustering. Therefore, we convert each generated weighted graph to an unweighted one consisting only of the $|E|$ edges with the highest weights, where $|E|$ is the number of edges in the original noise-free graph. For FGC, the node similarity matrix is optimized using the loss function in the original paper. The order of the graph Laplacian filter is fine-tuned exhaustively in the range from 1 to 15, and the trade-off parameter α in the range $\{0.0001, 0.01, 1, 10, 100\}$.

[7] The performance of DGI and GMI degrades significantly if we set the embedding dimension as 64.

REFERENCES

[1] Lada A Adamic and Eytan Adar. 2003. Friends and neighbors on the web. *Social networks* 25, 3 (2003), 211–230.

[2] Punam Bedi and Chhavi Sharma. 2016. Community detection in social networks. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 6, 3 (2016), 115–135.

[3] Alex Beutel, Wanhong Xu, Venkatesan Guruswami, Christopher Palow, and Christos Faloutsos. 2013. Copycatch: stopping group attacks by spotting lockstep behavior in social networks. In *WWW*.

[4] Filippo Maria Bianchi, Daniele Grattarola, and Cesare Alippi. 2020. Spectral clustering with graph neural networks for graph pooling. In *ICML*.

[5] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. 2008. Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment* 2008, 10 (2008), P10008.

[6] Ulrik Brandes, Daniel Delling, Marco Gaertler, Robert Görke, Martin Hoefer, Zoran Nikoloski, and Dorothea Wagner. 2006. Maximizing modularity is hard. *arXiv preprint physics/0608255* (2006).

[7] Chiyu Cai, Linjing Li, and Daniel Zeng. 2017. Detecting social bots by jointly modeling deep behavior and content information. In *CIKM*.

[8] Jingchun Chen and Bo Yuan. 2006. Detecting functional modules in the yeast protein–protein interaction network. *Bioinformatics* 22, 18 (2006), 2283–2290.

[9] Davide Chicco and Giuseppe Jurman. 2020. The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation. *BMC genomics* 21, 1 (2020), 1–13.

[10] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. 2016. Fast and accurate deep network learning by exponential linear units (elus). In *ICLR*.

[11] Yin Cui, Menglin Jia, Tsung-Yi Lin, Yang Song, and Serge Belongie. 2019. Class-balanced loss based on effective number of samples. In *CVPR*.

[12] Enyan Dai, Wei Jin, Hui Liu, and Suhang Wang. 2022. Towards robust graph neural networks for noisy graphs with sparse labels. In *WSDM*.

[13] Negin Entezari, Saba A Al-Sayouri, Amirali Darvishzadeh, and Evangelos E Papalexakis. 2020. All you need is low (rank) defending against adversarial attacks on graphs. In *WSDM*.

[14] Boyuan Feng, Yuke Wang, and Yufei Ding. 2021. Uag: Uncertainty-aware attention graph neural network for defending adversarial attacks. In *AAAI*.

[15] Emilio Ferrara, Onur Varol, Clayton Davis, Filippo Menczer, and Alessandro Flammini. 2016. The rise of social bots. *Commun. ACM* 59, 7 (2016), 96–104.

[16] Chakib Fettal, Lazhar Labiod, and Mohamed Nadif. 2022. Efficient graph convolution for joint node representation learning and clustering. In *WSDM*.

[17] Michelle Girvan and Mark EJ Newman. 2002. Community structure in social and biological networks. *Proceedings of the national academy of sciences* 99, 12 (2002), 7821–7826.

[18] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *KDD*.

[19] Siyuan Guo, Lixin Zou, Yiding Liu, Wenwen Ye, Suqi Cheng, Shuaiqiang Wang, Hechang Chen, Dawei Yin, and Yi Chang. 2021. Enhanced doubly robust learning for debiasing post-click conversion rate estimation. In *SIGIR*.

[20] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *NeurIPS*.

[21] Wei Jin, Yao Ma, Xiaorui Liu, Xianfeng Tang, Suhang Wang, and Jiliang Tang. 2020. Graph structure learning for robust graph neural networks. In *KDD*.

[22] Zhao Kang, Zhanyu Liu, Shirui Pan, and Ling Tian. 2022. Fine-grained attributed graph clustering. In *SDM*.

[23] Nikolaos Karalias, Joshua Robinson, Andreas Loukas, and Stefanie Jegelka. 2022. Neural Set Function Extensions: Learning with Discrete Functions in High Dimensions. In *NeurIPS*.

[24] Brian W Kernighan and Shen Lin. 1970. An efficient heuristic procedure for partitioning graphs. *The Bell system technical journal* 49, 2 (1970), 291–307.

[25] Minseok Kim, Hwanjun Song, Doyoung Kim, Kijung Shin, and Jae-Gil Lee. 2021. PREMERE: Meta-Reweighting via Self-Ensembling for Point-of-Interest Recommendation. In *AAAI*.

[26] Minseok Kim, Hwanjun Song, Yooju Shin, Dongmin Park, Kijung Shin, and Jae-Gil Lee. 2022. Meta-Learning for Online Update of Recommender Systems. In *AAAI*.

[27] Diederik P Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *ICLR*.

[28] Thomas N Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. In *ICLR*.

[29] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. 2019. Self-normalizing neural networks. In *NeurIPS*.

[30] Johannes Klicpera, Stefan Weißenberger, and Stephan Günnemann. 2019. Diffusion improves graph learning. In *NeurIPS*.

[31] Andrea Lancichinetti and Santo Fortunato. 2011. Limits of modularity maximization in community detection. *Physical review E* 84, 6 (2011), 066122.

[32] Sune Lehmann and Lars Kai Hansen. 2007. Deterministic modularity optimization. *The European Physical Journal B* 60, 1 (2007), 83–88.

[33] Ao Li, Zhou Qin, Runshi Liu, Yiqun Yang, and Dong Li. 2019. Spam review detection with graph convolutional networks. In *CIKM*.

[34] Xin Liu, Hui-Min Cheng, and Zhong-Yuan Zhang. 2019. Evaluation of community detection methods. *IEEE Transactions on Knowledge and Data Engineering* 32, 9 (2019), 1736–1746.

[35] Stuart Lloyd. 1982. Least squares quantization in PCM. *IEEE transactions on information theory* 28, 2 (1982), 129–137.

[36] Dongsheng Luo, Wei Cheng, Wenchao Yu, Bo Zong, Jingchao Ni, Haifeng Chen, and Xiang Zhang. 2021. Learning to drop: Robust graph neural network via topological denoising. In *WSDM*.

[37] Chen Ma, Yingxue Zhang, Qinglong Wang, and Xue Liu. 2018. Point-of-interest recommendation: Exploiting self-attentive autoencoders with neighbor-aware influence. In *CIKM*.

[38] Vinod Nair and Geoffrey E Hinton. 2010. Rectified linear units improve restricted boltzmann machines. In *ICML*.

[39] Mark EJ Newman. 2004. Fast algorithm for detecting community structure in networks. *Physical review E* 69, 6 (2004), 066133.

[40] Mark EJ Newman. 2006. Modularity and community structure in networks. *Proceedings of the national academy of sciences* 103, 23 (2006), 8577–8582.

[41] Mark EJ Newman. 2013. Spectral methods for community detection and graph partitioning. *Physical Review E* 88, 4 (2013), 042822.

[42] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. 1999. *The PageRank citation ranking: Bringing order to the web.* Technical Report. Stanford InfoLab.

[43] Shashank Pandit, Duen Horng Chau, Samuel Wang, and Christos Faloutsos. 2007. Netprobe: a fast and scalable system for fraud detection in online auction networks. In *WWW*.

[44] Zhen Peng, Wenbing Huang, Minnan Luo, Qinghua Zheng, Yu Rong, Tingyang Xu, and Junzhou Huang. 2020. Graph representation learning via graphical mutual information maximization. In *WWW*.

[45] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *KDD*.

[46] Muhammad Zubair Rafique and Muhammad Abulaish. 2012. Graph-based learning model for detection of SMS spam on smart phones. In *IWCMC*.

[47] Mengye Ren, Wenyuan Zeng, Bin Yang, and Raquel Urtasun. 2018. Learning to reweight examples for robust deep learning. In *ICML*.

[48] Mohsen Sayyadiharikandeh, Onur Varol, Kai-Cheng Yang, Alessandro Flammini, and Filippo Menczer. 2020. Detection of novel social bots by ensembles of specialized classifiers. In *CIKM*.

[49] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. 2008. Collective classification in network data. *AI magazine* 29, 3 (2008), 93–93.

[50] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. 2018. Pitfalls of graph neural network evaluation. *arXiv preprint arXiv:1811.05868* (2018).

[51] Kijung Shin, Bryan Hooi, Jisu Kim, and Christos Faloutsos. 2017. Densealert: Incremental dense-subtensor detection in tensor streams. In *KDD*.

[52] Jun Shu, Qi Xie, Lixuan Yi, Qian Zhao, Sanping Zhou, Zongben Xu, and Deyu Meng. 2019. Meta-weight-net: learning an explicit mapping for sample weighting. In *NeurIPS*.

[53] Anton Tsitsulin, John Palowitch, Bryan Perozzi, and Emmanuel Müller. 2020. Graph clustering with graph neural networks. *arXiv preprint arXiv:2006.16904* (2020).

[54] Sergei Vassilvitskii and David Arthur. 2006. k-means++: The advantages of careful seeding. In *SODA*.

[55] Petar Velickovic, William Fedus, William L Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. 2019. Deep Graph Infomax.. In *ICLR*.

[56] Guan Wang, Sihong Xie, Bing Liu, and S Yu Philip. 2011. Review graph based online store review spammer detection. In *ICDM*.

[57] Huijun Wu, Chen Wang, Yuriy Tyshetskiy, Andrew Docherty, Kai Lu, and Liming Zhu. 2019. Adversarial examples for graph data: deep insights into attack and defense. In *IJCAI*.

[58] Zhenyu Wu and Richard Leahy. 1993. An optimal graph theoretic approach to data clustering: Theory and its application to image segmentation. *IEEE transactions on pattern analysis and machine intelligence* 15, 11 (1993), 1101–1113.

[59] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. 2019. GraphSAINT: Graph Sampling Based Inductive Learning Method. In *ICLR*.

[60] Xiang Zhang and Marinka Zitnik. 2020. Gnnguard: Defending graph neural networks against adversarial attacks. In *NeurIPS*.

[61] Dingyuan Zhu, Ziwei Zhang, Peng Cui, and Wenwu Zhu. 2019. Robust graph convolutional networks against adversarial attacks. In *KDD*.