RGE: A Repulsive Graph Rectification for Node Classification via Influence

Jaeyun Song [*1] **SungYub Kim** [*1] **Eunho Yang** [1 2]

Abstract

In real-world graphs, noisy connections are inevitable, which makes it difficult to obtain unbiased node representations. Among various attempts to resolve this problem, a method of estimating the counterfactual effects of these connectivities has recently attracted attention, which mainly uses influence functions for single graph elements (i.e., node and edge). However, in this paper, we argue that there is a strongly interacting group effect between the influences of graph elements due to their connectivity. In the same vein, we observe that edge groups connecting to the same train node exhibit significant differences in their influences, hence no matter how negative each is, removing them at once may have a rather negative effect as a group. Based on this motivation, we propose a new edge-removing strategy, Repulsive edge Group Elimination (RGE), that preferentially removes edges with no interference in groups. Empirically, we demonstrate that RGE consistently outperforms existing methods on the various benchmark datasets.

1. Introduction

Learning effective node representations becomes increasingly crucial as the need for graph-structured data processing increases in diverse industrial applications such as social networks (Mohammadrezaei et al., 2018), bioinformatics (Hamilton et al., 2017), and recommendation systems (Perozzi et al., 2016) to name a few. Recently, various Graph Neural Networks (GNNs) have been proposed to integrate the topological information of graphs into node representation, and have achieved unprecedented performance improvements in many node classification tasks (Kipf & Welling, 2017; Veličković et al., 2018; Hamilton et al., 2017; Wu et al., 2019). However, GNNs are still susceptible to structural noises, such as task-irrelevant edges, in real-world graphs (Zheng et al., 2020; Luo et al., 2021). Consequently, these noises in graphs can hamper the use of GNNs in the industry due to the deterioration in the generalization performance.

Recently, several algorithms have been proposed to resolve these structural noises using topological information from graphs. ReNode (Chen et al., 2021) and TAM (Song et al., 2022) modify loss functions (e.g., adjusting loss weights or margins) to decrease the signals of deteriorating nodes. On the other hand, NeuralSparse (Zheng et al., 2020) and PTDNet (Luo et al., 2021) remove edges by learning auxiliary networks to sparsify graphs in an end-to-end manner. To the best of our knowledge, none of the existing methods explain why each edge is helpful or harmful, as they do not assess the counterfactual effects of removing edges.

To mitigate this issue, Chen et al. (2022a) proposed an influence function for individual nodes and edges to estimate the counterfactual effects of removing these elements. Based on this, they found that removing negative influence edges, called opponent edges (Pruthi et al., 2020), could significantly improve classification performance. However, this approach is intractable on graphs with many edges, as they retrain GNNs for each accumulative opponent edge set and select one using the validation accuracy of retrained GNNs. Furthermore, this type of graph rectification cannot consider the group effect of edges (i.e., how an edge's influence changes as neighboring edges are removed). For example, if removing an edge makes the influence of surrounding edges positive, preserving its neighbors will help generalize, and vice versa.

Therefore, to develop more efficient and effective strategies, we study a simple approach, Exhaustive edge Group Elimination (EGE), that eliminates all opponent edges at once. This approach can discover the most opponent set if influence additivity holds (i.e. there does not exist any group effect). Nevertheless, we hypothesize that neighboring opponent edges mainly cause significant group effects. In fact, we demonstrate that, due to group influence estimation error (the gap between group influence and the sum of individual

[*]Equal contribution [1]Graduate School of Artificial Intelligence, Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Korea [2]AITRICS, Seoul, Korea. Correspondence to: Jaeyun Song <mercery@kaist.ac.kr>, SungYub Kim <sungyub.kim@kaist.ac.kr>, Eunho Yang <eunhoy@gmail.com>.

Proceedings of the 40^{th} International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023.

influence), this approach degrades performance on synthetic graphs. Furthermore, neighboring edges exhibit much larger gaps compared to distant edges.

In light of this observation, we propose a new method, **R**epulsive edge **G**roup **E**limination (RGE), that prefers to remove distant edges over neighboring edges. As a single iteration of RGE does not remove opponent edges at once, we repeat removing the opponent edges using RGE and retraining GNNs until no opponent edges are left. We show that the edge group discovered by RGE has a smaller group influence estimation error than EGE on diverse synthetic graphs, and multiple iterations of RGE significantly improve performance. Furthermore, RGE outperforms baselines on standard benchmark datasets, such as citation networks (Sen et al., 2008), commercial graphs (Shchur et al., 2018), WebKB, Actor Network (Pei et al., 2020), and Wikipedia networks (Rozemberczki et al., 2021).

Our contribution is threefold:

- We demonstrate that there are substantial differences between group influence and the sum of individual influence for graph neural networks when an edge group connects to the same train node.
- We propose an iterative framework for removing a repulsive group of edges to avoid the aftermath of group effects.
- Empirically, our method shows superior performance to baselines on various benchmark datasets.[1]

2. Preliminary

2.1. Graph Neural Networks

Let us consider an undirected graph $G = (V, E)$ where V is a set of vertices (or nodes), and E is a set of edges. We denote the adjacency matrix of G as $A \in \mathbb{R}^{|V|\times|V|}$ and the degree matrix as $D \in \mathbb{R}^{|V|\times|V|}$.[2] Specifically, A is a binary symmetry matrix where $A_{ij} = A_{ji} = 1$ for $e_{ij} \in E$ and 0 otherwise, and D is a diagonal matrix whose v-th diagonal element is the node degree of $v \in V$ (i.e., $D_{ii} = \sum_{j\in V} A_{ij}$). The set of nodes whose distance from $v \in V$ is less than or equal to $k \in \mathbb{N}$ is called a k-hop neighborhood of v and is denoted by $\mathcal{N}_k(v)$. We refer to the notation table in Appendix A for mathematical terms used in the paper.

We target node classification tasks on G where each node $v \in V$ has representation $x_v \in \mathbb{R}^M$ and the corresponding label $y_v \in \mathbb{R}^C$. We assume nodes are divided into three different subsets for training and evaluation: train nodes V_{Tr} for training parameters, validation nodes V_{Val} for selecting hyperparameters and eliminating edges, and test nodes V_{Test} for evaluation.

The most significant difference between a classical neural network and a GNN is whether each layer aggregates (intermediate) features over neighboring nodes. For example, the l-th layer of Graph Convolution Networks (GCN; Kipf & Welling (2017)) pre-processes its input representation z_v^{l-1} as

$$\tilde{z}_v^l = \sum_{u\in\mathcal{N}_1(v)} \frac{z_u^{l-1}}{\sqrt{(D_{uu}+1)(D_{vv}+1)}}$$

where x_v is simply defined as z_v^0 for notational convenience. Then, standard feature transformation (with non-linear activation) is applied for the aggregated feature as

$$z_v^l = \sigma(\tilde{z}_v^l \Theta^l)$$

where Θ^l are learnable parameters (i.e., weight and bias) of l-th layer, and $\sigma(\cdot)$ is a component-wise non-linear activation function (e.g., ReLU). In summary, the output of L-layer GCN can be written concisely as

$$\hat{y}_v = \text{softmax}\left(S\sigma(S\cdots\sigma(Sx_v\Theta^1)\cdots\Theta^{L-1})\Theta^L\right). \quad (1)$$

Here, $S = \tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}$ is the renormalized graph Laplacian (Kipf & Welling, 2017) where $\tilde{A} = A + I_{|V|}$ and $\tilde{D}$ is the degree matrix of $\tilde{A}$, and $\text{softmax}(z) = \exp(z)/\sum_{k=1}^K \exp(z)_k$ for $z \in \mathbb{R}^K$. Recently, Wu et al. (2019) proposed L-hop Simple Graph Convolution (SGC) by removing activation functions in (1) so that we have

$$\hat{y}_v = \text{softmax}\left(S^L x_v \Theta\right) \quad (2)$$

where $\Theta = \prod_{l=1}^L \Theta^l$. Hereafter, we denote all parameters of any GNN as $\theta \in \mathbb{R}^P$ to represent it as a single vector.

The empirical risk minimization (ERM) of node classification tasks solves the following optimization problem

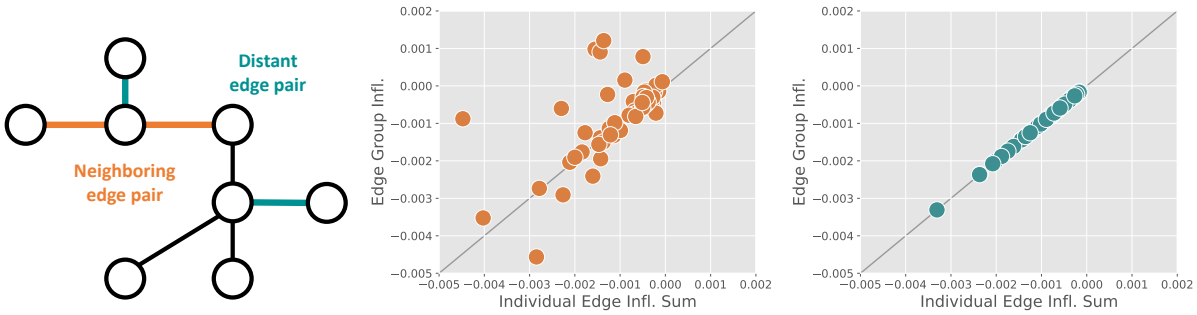
$$\theta^* := \underset{\theta}{\text{argmin}}\, \mathcal{L}_{\text{Tr}}(G, \theta) \quad (3)$$

where $\mathcal{L}_{\text{Tr}}(G,\theta) := \sum_{v\in V_{\text{Tr}}} \ell\big(\hat{y}_v(G,\theta), y_v\big)/|V_{\text{Tr}}|$ for cross-entropy loss $\ell(\cdot,\cdot)$ for the training data V_{Tr}. Here, note that we use $\hat{y}_v(G,\theta)$ for predictions to clarify the dependency on G and θ. While both GCN and SGC refer to the $\mathcal{N}_L(v)$ for the prediction of $v \in V$ in node classification, training of SGC is more efficient and stable than GCN as the loss function of SGC is convex w.r.t. θ.

We now define several relations among edges, including the affected train node set, distant edges, and neighbor edges.

[1] Code will be available at https://github.com/Jaeyun-Song/RGE.git

[2] We use $|S|$ to indicate the cardinality of set S.



(a) Neighboring and distant edge pairs (b) Influence of neighboring edge pairs (c) Influence of distant edge pairs

Figure 1. The group influences plotted against the sum of individual influences. (a) indicates which kinds of edge pairs are a neighboring edge pair or a distant edge pair under 1-hop SGC. (b) and (c) represent graphs between group influences and the sum of individual influences for neighboring edge pairs and for distant edge pairs, respectively.

Definition 2.1. (Affected train node-set) For a given edge set $E' \subset E$, the affected train node-set is defined as:

$$V_{\text{Tr}}(E') := \left\{ v \in V_{\text{Tr}} \mid \hat{y}_v\big(G(E'), \theta^*\big) \neq \hat{y}_v\left(G, \theta^*\right) \right\}, \tag{4}$$

where $G(E') = (V, E \setminus E')$ is the graph when E' is eliminated from G.

With abuse of notation, we also use V_{Tr} for a single edge as $V_{\text{Tr}}(e) := V_{\text{Tr}}\big(\{e\}\big)$.

Definition 2.2. (Distant edges) For given edges $e, e' \in E$, we say e, e' are distant edges if these edges satisfy the following condition:

$$V_{\text{Tr}}(e) \cap V_{\text{Tr}}(e') = \emptyset. \tag{5}$$

In contrast, we say edges e, e' are **neighboring edges** if (5) does not hold. The concept figure for neighboring edges and distant edges under 2-hop SGC is described in Figure 1(a).

2.2. Influence Estimation on Edges

Our goal is to find harmful edges, called opponent edges, that can improve the test performance when removed from the original graph G. Formally, we would like to identify an edge set $E' \subset E$ such that θ', the ERM solution (3) on the modified graph $G(E') := (V, E \setminus E')$, outperforms θ^* in terms of test-time evaluation metrics such as the node classification accuracy. However, since test nodes are not available during the training phase, we define the influence of E' on validation loss as a surrogate:

$$\mathcal{I}_{\texttt{True}}(E') := \mathcal{L}_{\text{Val}}(G, \theta') - \mathcal{L}_{\text{Val}}(G, \theta^*) \tag{6}$$

where $\mathcal{L}_{\text{Val}}(G, \theta) := \sum_{v \in V_{\text{Val}}} \ell\big(\hat{y}_v(G, \theta), y_v\big) / |V_{\text{Val}}|$.[3] That is, $\mathcal{I}_{\texttt{True}}(E')$ is the change in the loss after removing edges in E', hence, the more negative it is, the more opponent E' is. Here, since retraining for an arbitrary edge set requires a substantial computational cost where we have a huge number of edges in graph datasets in practice, Chen et al. (2022a) proposed an efficient approximation of ground truth influence as follows

$$\mathcal{I}(E') := \tilde{g}^\top H^{-1}\left(g - g'\right) \tag{7}$$

where $g = \nabla_\theta \mathcal{L}_{\text{Tr}}(G, \theta^*)$, $\tilde{g} = \nabla_\theta \mathcal{L}_{\text{Val}}(G, \theta^*)$, $H := \nabla^2_\theta \mathcal{L}_{\text{Tr}}(G, \theta^*)$, and $g' = \nabla_\theta \mathcal{L}_{\text{Tr}}\big(G(E'), \theta^*\big)$. Intuitively, this approximation can be understood as a two-step approximation of ground truth influence in the sense that

$$\begin{aligned} \mathcal{I}_{\texttt{True}}(E') &\approx \mathcal{L}^{\text{lin}}_{\text{Val}}(G, \theta') - \mathcal{L}^{\text{lin}}_{\text{Val}}(G, \theta^*) && (8) \\ &= \tilde{g}^\top(\theta' - \theta^*) \\ &\approx \tilde{g}^\top H^{-1}\left(g - g'\right) && (9) \end{aligned}$$

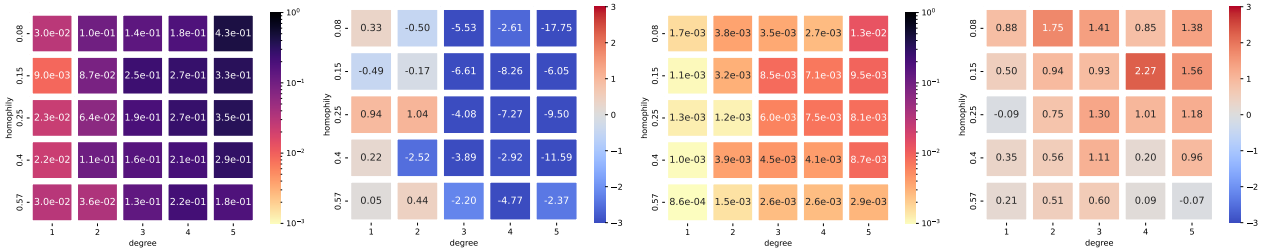
where $L^{\text{lin}}_{\text{Val}}(G, \psi) := \mathcal{L}_{\text{Val}}(G, \theta^*) + \tilde{g}^\top(\psi - \theta^*)$. Note that (8) applies linearization to the validation loss and (9) approximates parameter difference $\theta' - \theta^*$ as follows

$$\theta' \approx \theta^* + \underbrace{H^{-1}g}_{\text{revert opt. for } G} - \underbrace{H^{-1}g'}_{\text{modified opt. with } G(E')}.$$

Hereafter, we also use $G, \mathcal{I}$ as $G(e) := G(\{e\})$ and $\mathcal{I}(e) := \mathcal{I}(\{e\})$ for notational simplicity.

Armed with this approximation, Chen et al. (2022a) searches for an opponent edge set, based on the individual edge influence, among the relatively small pool of edge sets. To construct the pool, they first sort edges in the ascending order of edge influence $\mathcal{I}(e)$, and then they accumulate one by one from the most negative influence edge: $E_1 = \{e_1\}, ..., E_{i+1} = E_i \cup \{e_{i+1}\}$ where e_{i+1} is the $(i+1)$-th smallest edge in the sorting. Then, they prepare the pool B by including all accumulated subsets: $B = \{E_1, E_2, \ldots, E_n\}$, where n is the number of opponent edges. Next, they retrain (or fine-tuned) GNNs on the

[3] By using `True`, we indicate that we are computing the *ground truth* influence.



(a) Influence difference of EGE (b) Performance gain of EGE (c) Influence difference of RGE (d) Performance gain of RGE

Figure 2. The group influence estimation error and performance gain over the properties of synthetic graphs such as degree and homophily. In (a) and (c), each cell indicates the difference between group influence and the sum of individual influence. As the color is darker, the difference is larger. The difference for EGE (single iteration) is much larger than RGE (single iteration). In (b) and (d), each cell represents performance gain (%) compared to the test accuracy of the corresponding original model. As the color is close to dark red, the method shows better performance than the original model. In contrast, as the color is close to dark blue, the method is inferior to the original model. Although EGE (single iteration) deteriorates performance, RGE (single iteration) improves performance.

graph after removing each subset $E_i \in B$ from the original graph and evaluate the validation set. Finally, they select the best set $\hat{E}$ among B according to the accuracy of the validation set. This method is, however, **intractable** for graphs with many edges since it requires retraining for every set in B accumulating each opponent edge. Furthermore, the edge set identified by this approach might be significantly far from the most opponent set in that the range of subsets is limited to B and B is curated without consideration of group effects.

3. Repulsive Edge Group Elimination

In this section, to identify opponent edge sets effectively and efficiently, we design and scrutinize a practical algorithm by considering multiple edges' group effects. In Section 3.1, we observe that there are significant differences between the group influence and the sum of individual influences, and neighboring opponent edges primarily cause these differences. Based on this observation, in Section 3.2, we propose a method of preferentially removing distant edges that may have less interference in groups.

3.1. Problem: Group Influence Estimation Error

As mentioned in Section 2, our goal is to identify the most opponent edge set E^*:

$$E^* = \underset{E' \subset E}{\operatorname{argmin}} \mathcal{I}(E').$$

Since there are exponentially many possible edge subsets from E, it is computationally intractable to find the optimal edge subset by enumerating all possible cases. Thus, it is necessary to develop efficient approaches to solve this problem practically.

A simple strategy is to collect all opponent edges, a process we call **E**xhaustive edge **G**roup **E**limination (EGE):

$$E^* \approx \hat{E} = \left\{ e \in E \mid \mathcal{I}(e) < 0 \right\}.$$

By using this method, we can find the optimal edge set if the following additivity condition for individual influences holds:

$$\mathcal{I}(\{e, e'\}) = \mathcal{I}(e) + \mathcal{I}(e') \tag{10}$$

for all different edges $e, e' \in E$. Under this additivity assumption, the sum of individual edge influences is equal to the group influence of all edges.

However, in reality, this condition is unlikely to hold due to the fact that edge influence would be significantly changed after adjacent edges are removed. Thus, the set found by EGE may have less negative influence than the sum of individual influences. We hypothesize that there are significant gaps between group influence and the sum of individual edge influences (group influence estimation error), and these differences mainly stem from neighboring edges rather than distant edges.

Experimental setup To validate our hypothesis, we conduct extensive experiments on synthetic graphs. We generate various graphs by changing (node) degree and homophily of graphs, through GraphWorld (Palowitch et al., 2022). We assign 20/230/250 nodes per class for train/valid/test nodes, respectively, then train 2-hop SGCs on these graphs. We note that homophily indicates how frequently nodes connect to nodes belonging to the same class. We then find opponent edges and compute the group influences and the sum of individual influences of edge pairs. For each edge pair, we compare the group influence and the sum of individual influences. This experiment is conducted for two categories of edge pairs such as neighboring edges and distant edges. The detailed experimental setting is provided in Appendix F.1.

Performance degradation in EGE We first examine the performance of EGE before verifying our hypothesis. In Figure 2(b), performance gains over the original SGC are indicated by colors on the heatmap. When a cell's color is close to dark red, the method performs better than the original SGC on the corresponding graph. In contrast, the blue color signifies a drop in performance. We observe that there are significant performance drops on most of the synthetic graphs. To identify the cause of the performance degradation, we also inspect the gap between group influence and the sum of the individual influences for edges found by EGE. In Figure 2(a), the color of each cell on the graph indicates the difference between group influence and the sum of individual influences where we use the darker color as the gap increases. As a result of the experiment, there are substantial gaps on all graphs. This result implies that the edge set found by EGE might be far from the optimal set due to group influence estimation errors, so EGE deteriorates the performance on the graphs as already shown in Figure 2(b).

Group influence estimation error We now validate our hypothesis. We first divide edge pairs into two groups: neighboring edges and distant edges. Then, we investigate the group influence estimation errors for each group. In Figure 1(b)-1(c), we show the sum of individual influences as an x-axis, and group influence as a y-axis on the graph. In Figure 1(b), we can observe significant group influence estimation errors for neighboring edge pairs. Surprisingly, there are some cases where group influence is positive even when all individual edges are negative. This indicates that removing neighboring opponent edges at once may degrade performance. In contrast, for distant edge pairs, most pairs are well aligned and lie on $y = x$ in Figure 1(c). Based on these results, we conclude that group influence estimation error is larger in neighboring edge pairs than in distant edge pairs. We note that consistent results are obtained for other synthetic and real-world graphs, which are deferred to Appendix G.

A key observation in the previous experiment is that influence additivity holds for distant edges. Indeed, we can prove it as follows.

Proposition 3.1 (Influence additivity for distant edges)**.** *Let edges $e, e' \in E$ be distant edges and, for them, assume that the removal of edge e does not change the gradient of unaffected train node v:*

$$\nabla_\theta \ell\big(\hat{y}_v\left(G\left(\{e, e'\}\right), \theta^*\right), y_v\big) = \nabla_\theta \ell\big(\hat{y}_v\left(G(e'), \theta^*\right), y_v\big) \quad (11)$$

for $e \neq e'$, $v \in V_{Tr}$, and $v \notin V_{Tr}(e)$. Then, influence additivity in (10) holds.

We refer to Appendix B for the proof. Note that the assumption (11) is prevalent for many GNNs, such as GCN (Kipf & Welling, 2017), GAT (Veličković et al., 2018), GraphSAGE (Hamilton et al., 2017), and SGC (Wu et al., 2019).

Algorithm 1 Repulsive edge Group Elimination (RGE)

Input: Graph $G = (V, E)$, set of train nodes V_{Tr}, model parameters θ, loss function $\mathcal{L}_{\text{Tr}}$ for train set, patience ρ
Initialize: Model parameter $\theta \in \mathbb{R}^d$.
$E' \leftarrow E$
# Multi-step edge elimination
for $k = 0, 1, 2, \cdots$ **do**
  # Retrain θ on the rectified graph
  $G' \leftarrow (V, E')$
  $\theta' \leftarrow \text{argmin}_\theta \mathcal{L}_{\text{Tr}}(G', \theta)$

  # Sort edges according to the influence $\mathcal{I}(e)$
  $E'_{\text{sorted}} \leftarrow$ sort E' as ascending order based on $\mathcal{I}$
  $n_{\text{negative}} \leftarrow \left|\left\{e_t \in E'_{\text{sorted}} \mid \mathcal{I}(e_t) < 0\right\}\right|$

  # Repulsive selection rule
  $e_1 \leftarrow$ the first edge in E'_{sorted}
  $\hat{E} \leftarrow \{e_1\}$
  $E_{\text{cache}} \leftarrow \{e_1\}$
  $c \leftarrow 0$
  for $i = 2, \cdots, n_{\text{negative}}$ **do**
    $e_i \leftarrow$ the i-th edge in E'_{sorted}

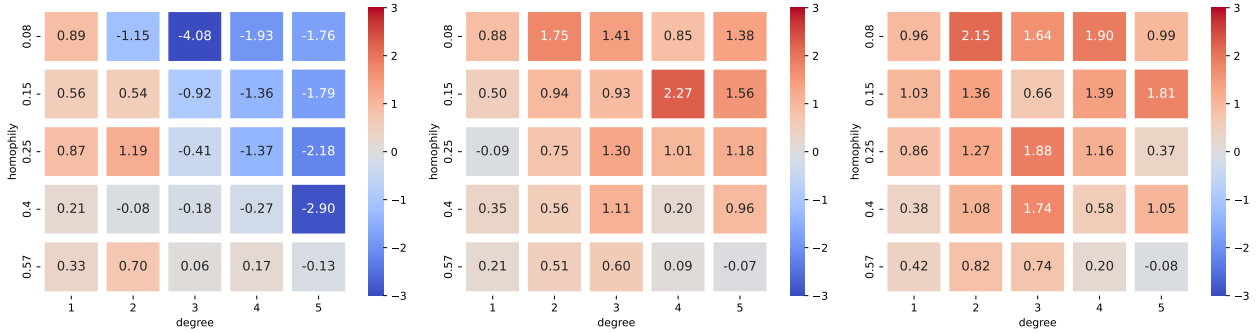
    # Patience mechanism
    if $(V_{\text{Tr}}(e_i) \cap V_{\text{Tr}}(E_{\text{cache}}) \neq \emptyset) \wedge (c \leq \rho)$ **then**
      $c \leftarrow c + 1$
    else
      $\hat{E} \leftarrow \hat{E} \cup \{e_i\}$
      $c \leftarrow 0$
      $E_{\text{cache}} \leftarrow E_{\text{cache}} \cup \{e_i\}$
    end if
    Keep the cache size $|E_{\text{cache}}|$ constant
  end for

  if $\left(|\hat{E}| = 1\right) \wedge \left(\mathcal{I}(e_1) \geq 0\right)$ **then**
    break
  end if
  $E' \leftarrow E' \setminus \hat{E}$
end for
Output: θ'

For example, for a given train node v and L-hop GCN, the presence of edge e connecting to $(L+2)$-hop node from v does not change the node representation of v, so it does not alter the gradient for v. Empirically, Figure 1(c) provides evidence of our proposition.

3.2. Proposed Method

We now propose a new approach, **R**epulsive edge **G**roup **E**limination (RGE), to find an opponent edge set while minimizing the risk of erroneous group influence estimation. The key concept is that RGE preferentially selects distant opponent edges over neighboring opponent edges (Repulsive selection rule). To improve performance further, RGE removes chosen edges and retrains GNNs repeatedly to eliminate remaining opponent edges (multi-step edge elimination). Consequently, RGE would reduce the chance of



(a) EGE (multiple iterations) (b) RGE (single iteration) (c) RGE (multiple iterations)

Figure 3. The performance gain over the properties of synthetic graphs such as degree and homophily. The values of cells in heatmaps represent performance gain (%) compared to the test accuracy of SGC on the corresponding graph. As the color is similar to dark red, the method shows better performance than the original model. Utilizing only multi-step edge elimination still suffers performance degradation, but RGE for a single iteration improves performance.

finding a sub-optimal opponent edge set due to group effects. To further boost efficiency, we introduce a patience mechanism to allow RGE to choose a few neighboring edges if they satisfy some criterion (Patience mechanism). Each major component is described in detail below and the overall algorithm is summarized in Algorithm 1.

Repulsive selection rule We here describe how to discover an opponent edge set $\hat{E}$. The main process is to investigate whether a candidate edge shares affected train nodes with already selected edges. If it does, we do not select this candidate since our influence estimation in this regime is not faithful, and removing this edge might not be actually beneficial due to the group effect with already selected edges. Specifically, RGE first sorts edges in ascending order by edge influence (we use E_{sorted} to indicate a sorted edge set). We add the most opponent edge $e_1 \in E_{\text{sorted}}$ to $\hat{E}$ (edge set to be removed). Then, from the second opponent edge to the least opponent edge, we inspect whether the i-th opponent edge $e_i \in E_{\text{sorted}}$ shares affected train nodes with already chosen edges: $V_{\text{Tr}}(e_i) \cap V_{\text{Tr}}(\hat{E}) \neq \emptyset$. If e_i does, we do not take it and move to the next candidate edge. Otherwise, we add e_i to $\hat{E}$. However, it turns out that, if all previously selected edges are used in this process, edge removal tends to be too conservative in the case of graphs with many connections. Hence, we maintain and utilize only the most recently selected edges (we use E_{cache} to indicate this set) when we inspect each candidate edge. We keep $|E_{\text{cache}}| \leq |V_{\text{Tr}}|/2$ for all experiments.

Multi-step edge elimination Since we do not select neighbor opponent edges at once, there are still residuals, so we suggest repeating RGE multiple times. In other words, we reiterate removing the edge set determined by RGE and retraining models until there are no more negative influence edges. Through repetitions, we would further eliminate noisy connections and improve generalization performance.

Patience mechanism We also introduce a patience mechanism to select more edges at each stage since the number of selected edges might be too small compared to the number of whole edges on graphs with many connections. The core idea of this mechanism is to allow RGE to even select a neighboring edge if we fail consecutively in the process of testing the affected train nodes. Specifically, we initialize the counting number c as zero. If $V_{\text{Tr}}(e_i) \cap V_{\text{Tr}}(E_{\text{cache}}) \neq \emptyset$ and c is less or equal to the pre-defined patience ρ, then we increase c as 1. Otherwise, we choose e_i and reset c to zero. To avoid selecting too many neighboring edges on graphs with a high degree, we devise the following rule, which empirically works well: $\rho = \log_2(d) + 3$. This rule logarithmically increases the patience scale with the average degree of the graph, d. We note that a log scale is adopted since a linear scale severely reduces the number of selected edges. For all experiments, we adopt this patience scale.

Similarly to our repulsive selection rule, Hermsdorff & Gunderson (2019) suggests a rule that avoids selecting directly adjacent edges at each iteration to reduce the size of graphs. However, the objectives of edge selection are different in that our goal is to identify and eliminate opponent edges. Furthermore, in Appendix D, we observe that using this rule for edge group elimination exhibits inferior performance in graphs with a high degree since these graphs already show high estimation errors as depicted in Figure 2(c). The estimation errors resulting from considering only directly adjacent edges become more critical to identify the opponent edge set in high-degree graphs.

4. Experiment

4.1. Experimental Setting

Datasets We demonstrate the effectiveness of RGE on various benchmark datasets, including citation networks (Sen et al., 2008), commercial graphs (Shchur et al., 2018), We-

Table 1. The performance on homophilous graphs under SGC (Wu et al., 2019) in node classification. **Bold** and underline denote the first-highest model and the second-highest model, respectively. RGE shows superior performance on most graphs. We repeat experiments 10 times and report the average accuracy and standard error. † indicates that we report the best accuracy in Luo et al. (2021).

Method (Acc.)	Cora	CiteSeer	PubMed	Photo	Computers
SGC (Wu et al., 2019)	81.00 ±0.00	71.90 ±0.00	78.90 ±0.00	90.22 ±0.26	86.65 ±0.33
ReNode (Chen et al., 2021)	80.82 ±0.01	71.30 ±0.00	80.00 ±0.00	90.34 ±0.23	86.55 ±0.24
TAM (Song et al., 2022)	81.20 ±0.00	71.40 ±0.00	79.40 ±0.00	89.41 ±0.31	85.81 ±0.29
DropEdge (Rong et al., 2020)	80.80 ±0.07	71.60 ±0.11	76.88 ±0.14	90.19 ±0.23	86.62 ±0.33
NeuralSparse (Zheng et al., 2020)†	83.4 ±1.5	72.4 ±2.6	78.8 ±1.8	-	-
PTDNet (Luo et al., 2021)†	84.4 ±2.3	73.7 ±3.1	79.8 ±2.4	-	-
Chen et al. (2022a)	83.40 ±0.01	**74.10** ±0.00	82.29 ±0.01	90.62 ±0.21	87.67 ±0.21
RGE	**84.50** ±0.14	73.75 ±0.13	**82.80** ±0.09	**91.64** ±0.18	**88.85** ±0.18

Table 2. The accuracy comparison on heterophilous graphs under SGC (Wu et al., 2019) in node classification. **Bold** and underline indicate the first-highest model and the second-highest model, respectively. RGE exhibits higher accuracy than baselines on most graphs. Since graphs in WebKB are small and experiments on these graphs have high variance, we reiterate experiments 100 times for WebKB, 10 times for Actor, and Squirrel. We provide the average accuracy and standard error.

Method (Acc.)	Cornell	Wisconsin	Texas	Actor	Squirrel
SGC (Wu et al., 2019)	54.32 ±0.72	64.90 ±0.66	**63.24** ±0.65	31.21 ±0.24	40.54 ±0.48
ReNode (Chen et al., 2021)	53.24 ±0.66	62.75 ±0.55	62.70 ±0.60	31.18 ±0.35	**40.93** ±0.53
TAM (Song et al., 2022)	51.89 ±0.57	66.47 ±0.61	45.68 ±0.61	31.14 ±0.26	36.58 ±0.56
DropEdge (Rong et al., 2020)	49.43 ±0.60	67.08 ±0.41	62.84 ±0.61	31.01 ±0.37	40.38 ±0.49
Chen et al. (2022a)	52.43 ±0.79	67.25 ±0.44	62.97 ±0.51	31.32 ±0.39	39.55 ±0.55
RGE	**56.32** ±0.38	**68.84** ±0.32	**63.24** ±0.63	**31.64** ±0.39	39.59 ±0.38

bKB[4], Actor network (Pei et al., 2020), and Wikipedia networks (Rozemberczki et al., 2021). These graphs are categorized by degree and homophily. Cora, CiteSeer, and PubMed (Sen et al., 2008) are homophilous networks with a low degree. AmazonPhoto, and AmazonComputers (Shchur et al., 2018) are homophilous graphs with a high node degree. In contrast, Cornell, Wisconsin, and Texas are heterophilous graphs with a few connections. Actor (Pei et al., 2020) and Squirrel (Rozemberczki et al., 2021) belong to heterophilous graphs possessing many edges compared to previous heterophilus graphs. We use the splits in Yang et al. (2016) for citation networks, and in Pei et al. (2020) for WebKB, Actor, and Wikipedia networks. For AmazonPhoto and AmazonComputers, we adopt ten random splits. The detailed setup is provided in Appendix F.

Baselines Competitive methods to alleviate the bias induced by noisy structures on graphs, such as ReNode (Chen et al., 2021), TAM (Song et al., 2022), DropEdge (Rong et al., 2020), NeuralSparse (Zheng et al., 2020), PTDNet (Luo et al., 2021), and Chen et al. (2022a), are compared to RGE. ReNode and TAM adjust the learning signals of the train nodes under noisy local topologies by modifying the sample weights or margins in the loss function. DropEdge randomly removes edges to prevent models from overfitting structures. NeuralSparse and PTDNet learn to drop edges by introducing additional networks which determine which edges would not be used while training GNNs. Chen et al. (2022a) removes opponent edges based on the edge influence function after training GNNs and retrains models on rectified graphs. We use the officially released codes for ReNode[5], and TAM[6].

4.2. Results

Group influence estimation error We experiment with single iteration RGE on various synthetic graphs to prove that the edge set produced by RGE has a small difference between the sum of individual influence and group influence. In fact, we observe that the group-effect gap of RGE is much smaller than EGE in Figure 2(c). It indicates that this group-effect gap can be effectively reduced by avoiding selecting all of an opponent's neighboring edges at once. Furthermore, single iteration RGE significantly improves performance while EGE degrades it in Figure 2(d). The results show that RGE can discover more harmful edge sets

[4] http://www.cs.cmu.edu/afs/cs.cmu.edu/project/theo-11/www/wwkb/

[5] https://github.com/victorchen96/ReNode

[6] https://github.com/Jaeyun-Song/TAM

Table 3. The performance on homophilous graphs under GCN and GAT in node classification. **Bold** and underline denote the best and second-best models, respectively. RGE shows superior performance on most graphs. We repeat experiments ten times and report the average accuracy and standard error. † indicates that we report the accuracy in Luo et al. (2021).

Method (Acc.)	GCN			GAT		
	Cora	CiteSeer	PubMed	Cora	CiteSeer	PubMed
Cross Entropy	82.74 ±0.13	71.82 ±0.07	79.25 ±0.09	82.79 ±0.13	71.64 ±0.17	78.62 ±0.14
ReNode (Chen et al., 2021)	82.74 ±0.10	72.65 ±0.12	80.14 ±0.10	83.21 ±0.12	72.44 ±0.07	79.47 ±0.16
TAM (Song et al., 2022)	82.31 ±0.09	73.10 ±0.16	80.09 ±0.06	82.51 ±0.16	72.72 ±0.20	79.26 ±0.14
DropEdge (Rong et al., 2020)	81.81 ±0.19	72.39 ±0.14	78.51 ±0.12	82.97 ±0.12	71.81 ±0.24	78.20 ±0.14
NeuralSparse (Zheng et al., 2020)†	82.1 ±1.8	71.5 ±1.7	78.8 ±1.8	83.4 ±1.5	72.4 ±2.6	78.0 ±1.7
PTDNet (Luo et al., 2021)†	82.8 ±2.6	72.7 ±1.8	79.8 ±2.4	<u>84.4</u> ±2.3	**73.7** ±3.1	79.3 ±1.5
Chen et al. (2022a)	<u>84.55</u> ±0.05	<u>73.58</u> ±0.13	<u>82.20</u> ±0.05	83.72 ±0.13	73.28 ±0.14	<u>81.19</u> ±0.10
RGE	**85.30** ±0.15	**73.70** ±0.18	**83.26** ±0.10	**84.70** ±0.18	<u>73.46</u> ±0.23	**82.50** ±0.13

than EGE by avoiding the selection of neighboring edge sets, which can cause a large group influence estimation error. In fact, among 25 synthetic graphs, Spearman's correlation between the difference in influence estimation and the performance gap among EGE and RGE is 0.93. This indicates that performance drops are highly correlated with group influence estimation errors.

Main results We validate RGE compared to baselines on 10 real-world networks. We observe that influence-based graph rectification such as Chen et al. (2022a) and ours show superior performance compared to learning-based rectification including NeuralSparse and PTDNet. Based on this result, we infer that these methods utilizing the influence function effectively discover noisy connectivities near train nodes. We also observe that RGE outperforms Chen et al. (2022a) on most homophilous graphs except for CiteSeer. Furthermore, RGE improves performance on heterophilous, albeit Chen et al. (2022a) fails. This indicates that the edge set found by RGE is more harmful than Chen et al. (2022a). For Squirrel, due to huge group influence errors, RGE exhibits a slight decline. Specifically, we observe that group influence estimation errors grow larger as the degree of a graph increases and homophily decreases, as shown in Figure 2(c), indicating that the edge set found by RGE might not consist of genuinely harmful edges. Thus, as the iterative process of RGE progresses, these errors intensify, leading to a performance drop in Squirrel.

To further demonstrate the effectiveness of RGE, we compare performance under other architectures, including GCN (Kipf & Welling, 2017) and GAT (Veličković et al., 2018) in Table 3. Specifically, for Chen et al. (2022a) and RGE, we first obtain the opponent edge set under SGC and train GCN and GAT on graphs where the identified edges are removed. We observe that RGE also performs better than baselines under GCN and GAT. These results imply that the opponent sets found under SGC also harm GCN and GAT, leading to performance improvement.

RGE iteration counts We investigate how many iterations are required for RGE to reach the stopping condition where there are no more opponent edges. In Appendix C, RGE takes 20-40 iterations in sparse graphs, while over 100 are needed in dense graphs. Nevertheless, for graphs with a high degree, only 5-10 iterations are sufficient to improve performance significantly.

Ablation study We provide the ablation study on the repulsive selection rule and multi-step edge elimination. In Figure 3, heatmaps show performance gains over the original SGC, with colors indicating the gains. We observe that, for EGE, there is a performance drop on graphs with a high degree in Figure 2(b). In graphs with many edges, EGE exhibits inferior performance in that edges connecting to the same train node are likely to be removed. In contrast, applying RGE only for a single iteration improves performance across all graphs in Figure 3(b) and further improvements can be expected when RGE is applied to multiple steps in Figure 3(c) in that RGE outperforms RGE (single iteration) on 20 of 25 graphs. Our method is, therefore, able to identify edge sets with a more negative influence than EGE, based on the results of this study.

The performance drop in EGE (single iterations) is also alleviated by only using multi-step edge elimination in Figure 3(a). Nevertheless, multi-step edge elimination still performs less well on heterophilous graphs with a high degree than the original SGC. Due to the possibility that multi-step edge elimination can also eliminate neighboring opponent edges, which is more likely if there are many opponent edges, multi-step edge elimination suffers from performance degradation. As a result, the repulsive selection rule is the core component for improving performance.

We examine the patience mechanism on various datasets in Appendix E. We observe that the patience mechanism

effectively reduces the number of iterations without compromising performance.

5. Related Works

Noisy structure handling Several works adjust the learning signals of train nodes to reduce the detrimental effects of noisy structures on graphs. ReNode (Chen et al., 2021) identifies train nodes suffering from influence conflicts and reduces the loss weight of these samples. TAM (Song et al., 2022) adjusts margins for train nodes that deviate from the overall connectivity pattern. TopoImb (Zhao et al., 2022) discovers topology groups and mitigates the bias toward major topology groups with the additional learning objective. In BA-GNN (Chen et al., 2022b), structural environments are determined and GNNs are trained to learn node representations that are invariant to these environments. As a result, BA-GNN resolves the bias to only several structural environments. Nevertheless, these approaches cannot directly determine which connections should be removed.

To obtain unbiased representations of nodes, several works remove edges directly. DropEdge(Rong et al., 2020) stochastically removes edges at each epoch to alleviate overfitting, but it cannot identify noisy edges. NeuralSparse (Zheng et al., 2020) introduces sparsification networks to learn and determine which edges should be removed and train the target GNNs on rectified graphs by sparsification networks. PTDNet (Luo et al., 2021) learns dropping probabilities through parametrized networks to avoid high variance and biased gradients and constrains the adjacent matrix of graphs to have a low rank to produce more realistic sparsified graphs. Despite the fact that these methods show improvement, they cannot estimate the counterfactual effect of edges, so it is likely that removing edges can be beneficial.

Chen et al. (2022a) devises edge influence function to estimate the counterfactual effect of a single graph element such as node and edge on the validation set and eliminates opponent edges based on this function. However, Chen et al. (2022a) does not consider the group effect when removing edges, so the edge group found by this method may not be significant. Several studies (Bajaj et al., 2021; Abrate & Bonchi, 2021) have attempted to predict counterfactual effects on graphs, but their methods are mainly designed for GNN interpretability.

Influence function Measuring the influences of training samples for predictions of models dates back to Cook's distance (Cook, 1977) for linear regression problems. Recently, Koh & Liang (2017) extended this concept to the deep learning models by introducing IFs. IF can be applied to various tasks for understanding and improving the pre-trained neural networks (NNs): detecting proponents (helpful samples) and opponents (obstructive samples) of given training data (Pruthi et al., 2020), finding noisy labeled samples in training dataset (Koh & Liang, 2017) and removing (Wang et al., 2018) or relabeling (Kong et al., 2021) these samples, tracing the origins of word embeddings (Brunet et al., 2019), and dataset pruning (or dataset summarization) task (Harutyunyan et al., 2021; Borsos et al., 2020).

Recent studies on IF have focused on improving the computational complexity of IF. IF contains inverse Hessian-vector product (IHVP) of training loss for parameters. Since inverting the Hessian matrix has $\mathcal{O}(p^3)$ time complexity and $\mathcal{O}(p^2)$ space complexity, Koh & Liang (2017) used LiSSA (Agarwal et al., 2016), which requires multiple Hessian-vector products (HVP), an algorithm to approximate IHVP. Nevertheless, approximating IHVP with LiSSA is still computationally intractable to large-scale models and datasets. To mitigate this issue, Pruthi et al. (2020) proposed Random Projection (RP) methods to avoid pair-wise gradient inner product and Schioppa et al. (2022) proposed Arnoldi iteration (Arnoldi, 1951) to extract the curvature information of Hessian directly. These methods will be helpful when applying RGE to more complex GNNs (Veličković et al., 2018; Chen et al., 2020).

6. Conclusion

To discover opponent edge sets, we devised and investigated practical approaches. We observed that there are substantial group effects among edges and these effects concentrate on neighboring opponent edges. Based on this observation, we proposed a new approach to choose preferentially distant edges over neighboring edges and reiterate removing them and retraining GNNs until there are no more opponent edges. We showed that our method outperforms baselines on various synthetic graphs and benchmark datasets.

Limitations When there is a non-trivial long-term interaction, our method requires more iterations. Thus, acquiring the final edge set might not be feasible.

Acknowledgements

This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grants (No.2019-0-00075, Artificial Intelligence Graduate School Program (KAIST), No.2021-0-02068, Artificial Intelligence Innovation Hub, No.2022-0-00984, Development of Artificial Intelligence Technology for Personalized Plug-and-Play Explanation and Verification of Explanation, No.2022-0-00713, Meta-learning applicable to real-world problems), and National Research Foundation of Korea (NRF) grants (No.2018R1A5A1059921, No.RS-2023-00209060, A Study on Optimization and Network Interpretation Method for Large-Scale Machine Learning) funded by the Korea government (MSIT).

References

Abrate, C. and Bonchi, F. Counterfactual graphs for explainable classification of brain networks. In *KDD '21: The 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, Singapore, August 14-18, 2021*, pp. 2495–2504. ACM, 2021.

Agarwal, N., Bullins, B., and Hazan, E. Second-order stochastic optimization in linear time. *stat*, 1050:15, 2016.

Arnoldi, W. E. The principle of minimized iterations in the solution of the matrix eigenvalue problem. *Quarterly of applied mathematics*, 9(1):17–29, 1951.

Bajaj, M., Chu, L., Xue, Z. Y., Pei, J., Wang, L., Lam, P. C., and Zhang, Y. Robust counterfactual explanations on graph neural networks. In *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pp. 5644–5655, 2021.

Borsos, Z., Mutny, M., and Krause, A. Coresets via bilevel optimization for continual learning and streaming. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

Brunet, M.-E., Alkalay-Houlihan, C., Anderson, A., and Zemel, R. Understanding the origins of bias in word embeddings. In *International conference on machine learning*, pp. 803–811. PMLR, 2019.

Chen, D., Lin, Y., Zhao, G., Ren, X., Li, P., Zhou, J., and Sun, X. Topology-imbalance learning for semi-supervised node classification. In *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pp. 29885–29897, 2021.

Chen, M., Wei, Z., Huang, Z., Ding, B., and Li, Y. Simple and deep graph convolutional networks. In *International conference on machine learning*, pp. 1725–1735. PMLR, 2020.

Chen, Z., Li, P., Liu, H., and Hong, P. Characterizing the influence of graph elements. *CoRR*, abs/2210.07441, 2022a. doi: 10.48550/arXiv.2210.07441. URL https://doi.org/10.48550/arXiv.2210.07441.

Chen, Z., Xiao, T., and Kuang, K. BA-GNN: on learning bias-aware graph neural network. In *38th IEEE International Conference on Data Engineering, ICDE 2022, Kuala Lumpur, Malaysia, May 9-12, 2022*, pp. 3012–3024. IEEE, 2022b.

Cook, R. D. Detection of influential observation in linear regression. *Technometrics*, 19(1):15–18, 1977.

Hamilton, W. L., Ying, Z., and Leskovec, J. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pp. 1024–1034, 2017.

Harutyunyan, H., Achille, A., Paolini, G., Majumder, O., Ravichandran, A., Bhotika, R., and Soatto, S. Estimating informativeness of samples with smooth unique information. *arXiv preprint arXiv:2101.06640*, 2021.

Hermsdorff, G. B. and Gunderson, L. M. A unifying framework for spectrum-preserving graph sparsification and coarsening. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pp. 7734–7745, 2019.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

Koh, P. W. and Liang, P. Understanding black-box predictions via influence functions. In *International conference on machine learning*, pp. 1885–1894. PMLR, 2017.

Kong, S., Shen, Y., and Huang, L. Resolving training biases via influence-based data relabeling. In *International Conference on Learning Representations*, 2021.

Luo, D., Cheng, W., Yu, W., Zong, B., Ni, J., Chen, H., and Zhang, X. Learning to drop: Robust graph neural network via topological denoising. In *WSDM '21, The Fourteenth ACM International Conference on Web Search and Data Mining, Virtual Event, Israel, March 8-12, 2021*, pp. 779–787. ACM, 2021.

Mohammadrezaei, M., Shiri, M. E., and Rahmani, A. M. Identifying fake accounts on social networks based on graph analysis and classification algorithms. *Secur. Commun. Networks*, 2018:5923156:1–5923156:8, 2018.

Palowitch, J., Tsitsulin, A., Mayer, B., and Perozzi, B. Graphworld: Fake graphs bring real insights for gnns. In *KDD '22: The 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, August 14 - 18, 2022*, pp. 3691–3701. ACM, 2022.

Pei, H., Wei, B., Chang, K. C.-C., Lei, Y., and Yang, B. Geom-gcn: Geometric graph convolutional networks. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=S1e2agrFvS.

Perozzi, B., Schueppert, M., Saalweachter, J., and Thakur, M. When recommendation goes wrong: Anomalous link discovery in recommendation networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, pp. 569–578. ACM, 2016.

Pruthi, G., Liu, F., Kale, S., and Sundararajan, M. Estimating training data influence by tracing gradient descent. *Advances in Neural Information Processing Systems*, 33: 19920–19930, 2020.

Rong, Y., Huang, W., Xu, T., and Huang, J. Dropedge: Towards deep graph convolutional networks on node classification. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=Hkx1qkrKPr.

Rozemberczki, B., Allen, C., and Sarkar, R. Multi-scale attributed node embedding. *J. Complex Networks*, 9(2), 2021.

Schioppa, A., Zablotskaia, P., Vilar, D., and Sokolov, A. Scaling up influence functions. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pp. 8179–8186, 2022.

Sen, P., Namata, G., Bilgic, M., Getoor, L., Gallagher, B., and Eliassi-Rad, T. Collective classification in network data. *AI Mag.*, 29(3):93–106, 2008.

Shchur, O., Mumme, M., Bojchevski, A., and Günnemann, S. Pitfalls of graph neural network evaluation. *CoRR*, abs/1811.05868, 2018. URL http://arxiv.org/abs/1811.05868.

Song, J., Park, J., and Yang, E. TAM: topology-aware margin loss for class-imbalanced node classification. In *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, volume 162 of *Proceedings of Machine Learning Research*, pp. 20369–20383. PMLR, 2022.

Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. Dropout: a simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, 2014.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., and Bengio, Y. Graph attention networks. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=rJXMpikCZ.

Wang, T., Huan, J., and Li, B. Data dropout: Optimizing training data for convolutional neural networks. In *2018 IEEE 30th international conference on tools with artificial intelligence (ICTAI)*, pp. 39–46. IEEE, 2018.

Wu, F., Jr., A. H. S., Zhang, T., Fifty, C., Yu, T., and Weinberger, K. Q. Simplifying graph convolutional networks. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pp. 6861–6871. PMLR, 2019.

Yang, Z., Cohen, W. W., and Salakhutdinov, R. Revisiting semi-supervised learning with graph embeddings. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, volume 48 of *JMLR Workshop and Conference Proceedings*, pp. 40–48. JMLR.org, 2016.

Zhao, T., Luo, D., Zhang, X., and Wang, S. Topoimb: Toward topology-level imbalance in learning from graphs. In *The First Learning on Graphs Conference*, 2022. URL https://openreview.net/forum?id=nR3rZ4ODtQ.

Zheng, C., Zong, B., Cheng, W., Song, D., Ni, J., Yu, W., Chen, H., and Wang, W. Robust graph representation learning via neural sparsification. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pp. 11458–11468. PMLR, 2020.

Zhu, J., Yan, Y., Zhao, L., Heimann, M., Akoglu, L., and Koutra, D. Beyond homophily in graph neural networks: Current limitations and effective designs. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

A. Notations

Table 4. Notations used in the main paper

$G = (V, E)$	$\triangleq$	graph G with node set V and edge set E
$G(E') = (V, E \setminus E')$	$\triangleq$	modification of graph G by elimination of $E' \subset E$
$A, D \in \mathbb{R}^{\lvert V\rvert \times \lvert V\rvert}$	$\triangleq$	adjacency matrix and degree matrix of renormalized graph
$S \in \mathbb{R}^{\lvert V\rvert \times \lvert V\rvert}$	$\triangleq$	renormalized graph Laplacian
$\tilde{A}, \tilde{D} \in \mathbb{R}^{\lvert V\rvert \times \lvert V\rvert}$	$\triangleq$	adjacency matrix and degree matrix of G
$\mathcal{N}_k(v) \subset V$	$\triangleq$	k-hop neighborhood of v.
$x_v \in \mathbb{R}^M, y_v \in \mathbb{R}^C$	$\triangleq$	node representation and label of node $v \in V$
$V_{\text{Tr}}, V_{\text{Val}}, V_{\text{Test}} \subset V$	$\triangleq$	train, valid, test nodes for node classification
z_v^{l-1}	$\triangleq$	hidden representation of l-th layer of GNN
Θ^l	$\triangleq$	trainable parameters of l-th layer of GNN
$\theta \in \mathbb{R}^P$	$\triangleq$	trainable parameters of GNN in a single vector
$\hat{y}_v$ (or $\hat{y}_v(G, \theta)$)	$\triangleq$	prediction of GNN for node v given graph G and parameter θ
$\mathcal{L}_{\text{Tr}}(G, \theta)$, $\mathcal{L}_{\text{Val}}(G, \theta)$, $\mathcal{L}_{\text{Test}}(G, \theta)$	$\triangleq$	train, valid, and test loss given G and θ
$\theta^* \in \mathbb{R}^P$	$\triangleq$	pre-trained parameter given full graph G
$V_{\text{Tr}}(E') \subset V$	$\triangleq$	affected train node set of edge set $E' \subset E$
$\mathcal{I}_{\text{True}}$	$\triangleq$	ground truth influence function obtained by retraining
$\mathcal{I}$	$\triangleq$	approximated influence function with Hessian
$g = \nabla_\theta \mathcal{L}_{\text{Tr}}(G, \theta^*)$	$\triangleq$	gradient w.r.t. parameters given train dataset and θ^*
$g' = \nabla_\theta \mathcal{L}_{\text{Tr}}\big(G(E'), \theta^*\big)$	$\triangleq$	train gradient for modified graph $G(E')$
$\tilde{g} = \nabla_\theta \mathcal{L}_{\text{Val}}(G, \theta^*)$	$\triangleq$	gradient w.r.t. parameters given validation dataset and θ^*
$H := \nabla_\theta^2 \mathcal{L}_{\text{Tr}}(G, \theta^*)$	$\triangleq$	Hessian w.r.t. parameters given train dataset and θ^*

B. Proof of Influence Additivity for Distant Edges

To clarify the dependence on E', we define $g'(E') = \nabla_\theta \mathcal{L}_{\text{Tr}}(G(E'), \theta^*)$. From the definition of distant edges in (5), the intersection of train nodes affected by each edge is empty: $V_{\text{Tr}}(\{e\}) \cap V_{\text{Tr}}(\{e'\}) = \emptyset$. Based on the assumption in (11) and the fact that e, e' are distant edges,

$$\sum_{v \in V_{\text{Tr}}(\{e\})} \nabla_\theta \ell(\hat{y}_v(G(\{e, e'\}), \theta^*), y_v) = \sum_{v \in V_{\text{Tr}}(\{e\})} \nabla_\theta \ell(\hat{y}_v(G(\{e\}), \theta^*), y_v)$$
$$\sum_{v \in V_{\text{Tr}}(\{e'\})} \nabla_\theta \ell(\hat{y}_v(G(\{e, e'\}), \theta^*), y_v) = \sum_{v \in V_{\text{Tr}}(\{e'\})} \nabla_\theta \ell(\hat{y}_v(G(\{e'\}), \theta^*), y_v).$$

From the definition, we can calculate gradients for the train loss as:

$$(g - g'(\{e\})) = \sum_{v \in V_{\text{Tr}}(\{e\})} (\nabla_\theta \ell(\hat{y}_v(G, \theta^*), y_v) - \nabla_\theta \ell(\hat{y}_v(G(\{e\}), \theta^*), y_v)) / |V_{\text{Tr}}|$$
$$(g - g'(\{e'\})) = \sum_{v \in V_{\text{Tr}}(\{e'\})} (\nabla_\theta \ell(\hat{y}_v(G, \theta^*), y_v) - \nabla_\theta \ell(\hat{y}_v(G(\{e'\}), \theta^*), y_v)) / |V_{\text{Tr}}|.$$

Then, we can induce the below equations:

$$\begin{aligned}
(g - g'(\{e, e'\})) &= \sum_{v \in V_{\text{Tr}}(\{e, e'\})} (\nabla_\theta \ell(\hat{y}_v(G, \theta^*), y_v) - \nabla_\theta \ell(\hat{y}_v(G(\{e, e'\}), \theta^*), y_v)) / |V_{\text{Tr}}| \\
&= \sum_{v \in V_{\text{Tr}}(\{e\})} (\nabla_\theta \ell(\hat{y}_v(G, \theta^*), y_v) - \nabla_\theta \ell(\hat{y}_v(G(\{e, e'\}), \theta^*), y_v)) / |V_{\text{Tr}}| \\
&+ \sum_{v \in V_{\text{Tr}}(\{e'\})} (\nabla_\theta \ell(\hat{y}_v(G, \theta^*), y_v) - \nabla_\theta \ell(\hat{y}_v(G(\{e, e'\}), \theta^*), y_v)) / |V_{\text{Tr}}|, \\
&= \sum_{v \in V_{\text{Tr}}(\{e\})} (\nabla_\theta \ell(\hat{y}_v(G, \theta^*), y_v) - \nabla_\theta \ell(\hat{y}_v(G(\{e\}), \theta^*), y_v)) / |V_{\text{Tr}}| \\
&+ \sum_{v \in V_{\text{Tr}}(\{e'\})} (\nabla_\theta \ell(\hat{y}_v(G, \theta^*), y_v) - \nabla_\theta \ell(\hat{y}_v(G(\{e'\}), \theta^*), y_v)) / |V_{\text{Tr}}| \\
&= (g - g'(\{e\})) + (g - g'(\{e'\})).
\end{aligned}$$

From the above equation, we can derive influence additivity for distant edge pairs.

$$\begin{aligned}
\mathcal{I}(\{e, e'\}) &= \tilde{g}^\top H^{-1} (g - g'(\{e, e'\})) \\
&= \tilde{g}^\top H^{-1} (g - g'(\{e\}) + g - g'(\{e'\})) \\
&= \tilde{g}^\top H^{-1} (g - g'(\{e\})) + \tilde{g}^\top H^{-1} (g - g'(\{e'\})) \\
&= \mathcal{I}(\{e\}) + \mathcal{I}(\{e'\})
\end{aligned}$$

C. The Number of RGE Iterations

We examine the iteration counts of RGE across various datasets in Table 5 and 6. We observe that sparsely connected graphs (including Cora, CiteSeer, PubMed, Cornell, Wisconsin, and Texas) take 20-40 iterations to achieve the stopping criteria where no more opponent edges are present. For graphs with high degrees (such as AmazonPhoto, AmazonComputers, Actor, and Squirrel), more than 100 iterations are required to reach the same stopping condition. However, we observe that, in graphs with high degrees, 5-10 iterations are sufficient to reach comparable performance to RGE in the stopping condition (N_{stop}), as depicted in Table 7.

Table 5. The iteration counts (N_{iter}) of RGE. We repeat experiments 10 times and report the average accuracy and standard error.

Dataset	Cora	CiteSeer	PubMed	Photo	Computers
N_{iter}	36.70 ± 0.95	29.20 ± 0.72	37.30 ± 0.88	133.20 ± 2.62	132.60 ± 2.65

Table 6. The iteration counts (N_{iter}) of RGE. We repeat experiments 10 times and report the average accuracy and standard error.

Dataset	Cornell	Wisconsin	Texas	Actor	Squirrel
N_{iter}	21.00 ± 0.30	22.24 ± 0.21	19.60 ± 0.45	58.50 ± 1.0	194.90 ± 6.59

Table 7. The accuracy across iteration counts (N_{iter}) of RGE. **Bold** indicates the best model in accuracy. N_{stop} presents RGE reaching the stopping criteria. We repeat experiments 10 times and report the average accuracy and standard error.

N_{iter}	Photo	Computers	Actor	Squirrel
1	90.39 ± 0.22	87.31 ± 0.23	31.25 ± 0.36	40.76 ± 0.47
5	90.82 ± 0.20	88.16 ± 0.20	31.37 ± 0.33	**41.24** ± 0.44
10	91.20 ± 0.21	88.67 ± 0.22	**31.74** ± 0.39	40.93 ± 0.49
N_{stop}	**91.64** ± 0.18	**88.85** ± 0.18	31.64 ± 0.39	39.59 ± 0.18

D. Comparison with Simple Selection Rule.

To demonstrate the effectiveness of RGE compared to the 1-hop rule of Hermsdorff & Gunderson (2019), we compare the accuracy and the number of iteration counts across various datasets. In Table 8, ours exhibits higher accuracy in all graphs. For iteration counts, the 1-hop rule necessitates fewer iterations in graphs with a low degree such as CiteSeer and PubMed. In contrast, the 1-hop rule requires a higher number of iterations in graphs with a high degree, including Photo and Actor. Since group influence estimation errors become larger in graphs with high-degree graphs, as shown in Figure 2(c), the additional errors from considering only directly adjacent edges are significant in these graphs, resulting in an increased iteration count.

Table 8. The accuracy and iteration counts (N_{iter}) of the 1-hop rule (Hermsdorff & Gunderson, 2019) and RGE. **Bold** indicates the best model in accuracy and the number of iterations. We repeat experiments 10 times and report the average accuracy and standard error.

Method	CiteSeer		PubMed		Photo		Actor	
	Acc.(↑)	N_{iter}(↓)	Acc.(↑)	N_{iter}(↓)	Acc.(↑)	N_{iter}(↓)	Acc.(↑)	N_{iter}(↓)
1-hop	72.36 ±0.05	**15.50** ±0.05	82.56 ±0.11	**30.20** ±0.37	91.17 ±0.21	384.50 ±84.10	31.47 ±0.48	97.90 ±32.62
RGE	**73.75** ±0.13	29.20 ±0.72	**82.80** ±0.09	37.30 ±0.88	**91.64** ±0.18	**133.20** ±2.62	**31.64** ±0.39	**58.50** ±1.00

E. Ablation Study for Patience Mechanism

We now investigate the effectiveness of the patience mechanism. We find that the patience mechanism significantly decreases the number of iterations across various datasets without sacrificing performance. Thus, this result indicates that the patience mechanism effectively enhances the efficiency of RGE.

Table 9. The performance and iteration counts (N_{iter}). **Bold** indicates the best model in accuracy and the number of iterations. We repeat experiments 10 times and report the average accuracy and standard error.

Method	CiteSeer		PubMed		Cornell		Wisconsin	
	Acc.(↑)	N_{iter}(↓)	Acc.(↑)	N_{iter}(↓)	Acc.(↑)	N_{iter}(↓)	Acc.(↑)	N_{iter}(↓)
RGE	73.75 ±0.13	**29.20** ±0.72	**82.80** ±0.09	**37.30** ±0.88	56.32 ±0.38	**21.00** ±0.30	**68.84** ±0.32	**22.24** ±0.21
w/o patience	**73.90** ±0.00	75.60 ±0.15	82.64 ±0.03	187.80 ±0.13	**57.30** ±0.59	53.60 ±0.65	68.65 ±0.25	55.01 ±1.75

F. Experimental Setting

F.1. Synthetic Graphs

To validate our approach on more diverse graphs, we generate diverse graphs from the perspective of degree and homophily, which is one of the important graph properties affecting performance in node classification. We utilize the recent graph synthesis approach (Palowitch et al., 2022) and each synthetic graph has degree among $\{1, 2, 3, 4, 5\}$ and edge homophily (Zhu et al., 2020) among $\{0.08, 0.14, 0.25, 0.40, 0.57\}$ with 4 types of node class (25 graphs in total). The numbers of train/valid/test nodes per class are 20/230/250 (2000 nodes in total). We validate methods on 10 random splits under SGC (Wu et al., 2019) for each graph. We note that the values of homophily are the results of the synthesis when we set the parameter regulating homophily (*p_to_q_ratio*) as $\{0.25, 0.5, 1, 2, 4\}$.

F.2. Evaluation Details

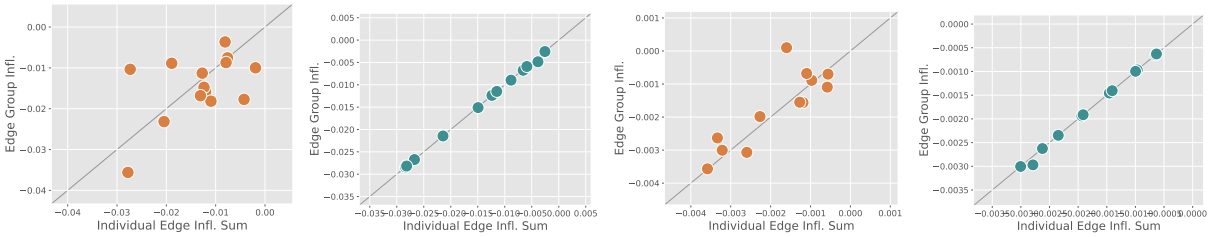
Our model is evaluated under SGC (Wu et al., 2019), GCN (Kipf & Welling, 2017), and GAT (Veličković et al., 2018). We train 2-hop SGC (Wu et al., 2019) for 200 epochs, while the 2-layer GCN and GAT are trained for 2000 epochs. We employ Adam optimizer (Kingma & Ba, 2015) with a learning rate of 0.2 for SGC except for CiteSeer (Sen et al., 2008) (0.5) and 0.01 for GCN and GAT. We choose the weight decay among 100 values ranging from 0.9 to 10^{-10} in the log scale according to the validation accuracy for SGC. For GCN and GAT, we select the weight decay from $\{10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}, 10^{-7}\}$ due to higher computational costs compared to SGC. We utilize dropout (Srivastava et al., 2014) of 0.5 in each layer of GCN and GAT. We adopt the hidden dimensions of 32 and 64 for GCN and GAT, respectively, and use eight heads for GAT. We note that we pre-process edges by removing self-loops and one-directional edges. For AmazonPhoto and AmazonComputers (Shchur et al., 2018), we use 10 random splits as the numbers of train/valid/test nodes per class are 20/60/200 to keep the class-balanced for each split.

F.3. Baselines

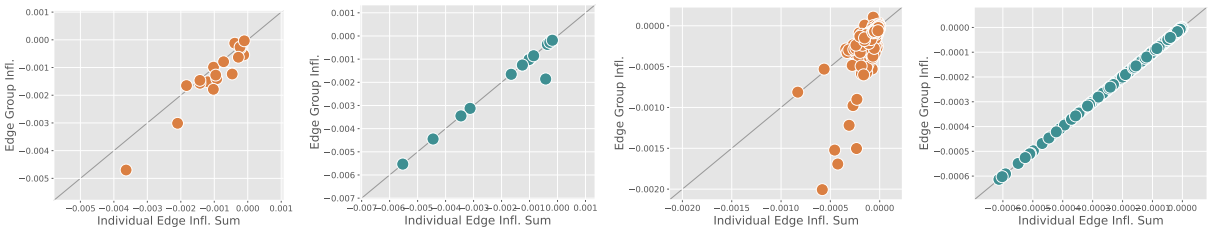
For ReNode (Chen et al., 2021), we search the best hyperparameter according to the validation accuracy among the space suggested in the paper. In specific, $\alpha \in \{0.05, 0.1, 0.15, 0.2\}$, $w_{min} \in \{0.25, 0.5, 0.75\}$, and $w_{max} \in \{1.25, 1.5, 1.75\}$. In TAM (Song et al., 2022), we find the best hyperparmeter among $\alpha \in \{0.5, 1.5, 2.5\}, \beta \in \{0.25, 0.5\}, \phi \in \{0.8, 1.2\}$. For DropEdge (Rong et al., 2020), we investigate the best hyperparameter in $p \in \{0.1, 0.2, \ldots, 0.9\}$ and stochastically drop two direction edges simultaneously at each epoch as Chen et al. (2022a) and RGE do.

G. Additional Results

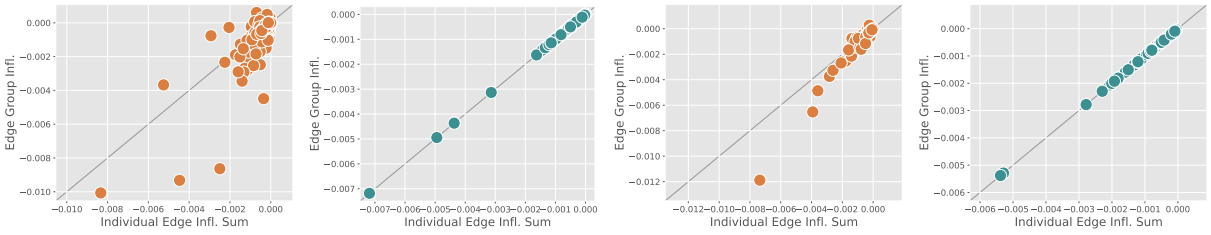
We provide the graphs between group influence and the sum of individual influence on real-world graphs in Figure 4 and synthetic graphs in Figure 5. As mentioned in Section 3.1, we observe that there are significant gaps for neighboring opponent edges, but edge pairs lie on $y = x$ for distant opponent edges.



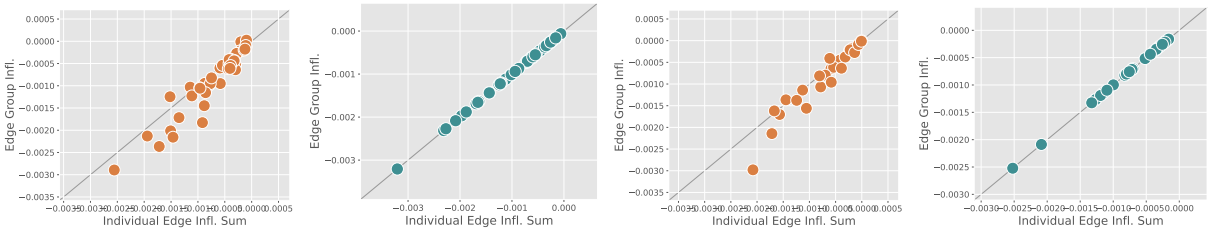
(a) Neighboring edges in Cornell (b) Distant edges in Cornell (c) Neighboring edges in Texas (d) Distant edges in Texas



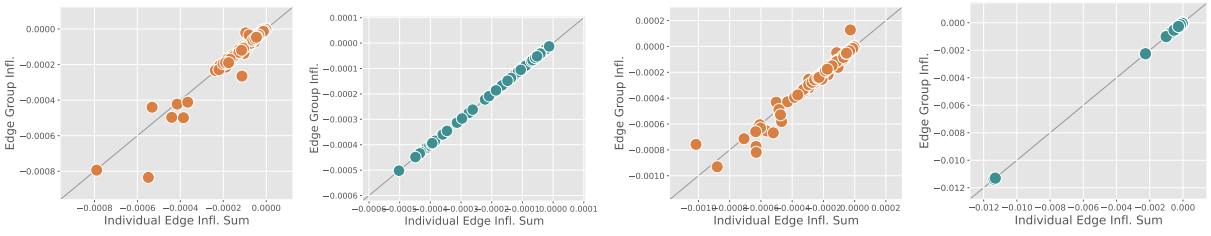
(e) Neighboring edges in Wisconsin (f) Distant edges in Wisconsin (g) Neighboring edges in Actor (h) Distant edges in Actor



(i) Neighboring edges in squirrel (j) Distant edges in squirrel (k) Neighboring edges in Cora (l) Distant edges in Cora

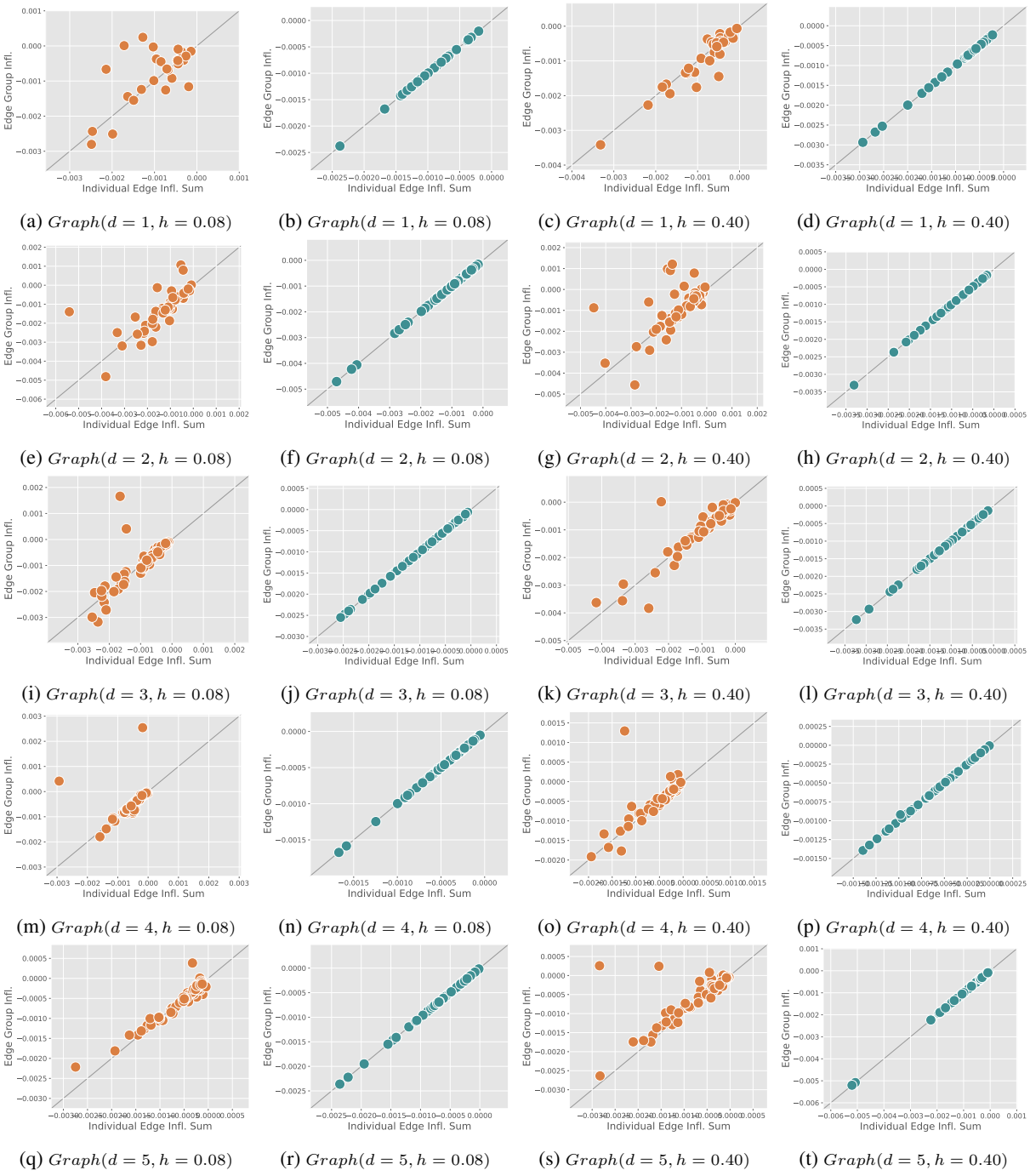


(m) Neighboring edges in CiteSeer (n) Distant edges in CiteSeer (o) Neighboring edges in PubMed (p) Distant edges in PubMed



(q) Neighboring edges in Photo (r) Distant edges in Photo (s) Neighboring edges in Computers (t) Distant edges in Computers

Figure 4. The group influences plotted with the sum of individual edge influences in 10 real-world graphs. The first and third columns indicate gaps for neighboring opponent edges. The second and fourth columns represent differences for distant opponent edges.



(a) $Graph(d = 1, h = 0.08)$ (b) $Graph(d = 1, h = 0.08)$ (c) $Graph(d = 1, h = 0.40)$ (d) $Graph(d = 1, h = 0.40)$

(e) $Graph(d = 2, h = 0.08)$ (f) $Graph(d = 2, h = 0.08)$ (g) $Graph(d = 2, h = 0.40)$ (h) $Graph(d = 2, h = 0.40)$

(i) $Graph(d = 3, h = 0.08)$ (j) $Graph(d = 3, h = 0.08)$ (k) $Graph(d = 3, h = 0.40)$ (l) $Graph(d = 3, h = 0.40)$

(m) $Graph(d = 4, h = 0.08)$ (n) $Graph(d = 4, h = 0.08)$ (o) $Graph(d = 4, h = 0.40)$ (p) $Graph(d = 4, h = 0.40)$

(q) $Graph(d = 5, h = 0.08)$ (r) $Graph(d = 5, h = 0.08)$ (s) $Graph(d = 5, h = 0.40)$ (t) $Graph(d = 5, h = 0.40)$

Figure 5. The group influences plotted with the sum of individual edge influences in 10 synthetic graphs. The first and third columns indicate gaps for neighboring opponent edges. The second and fourth columns represent differences for distant opponent edges. We note that d is degree and h is edge homophily (Zhu et al., 2020) of a graph.