Randomized Schur Complement Views for Graph Contrastive Learning

Vignesh Kothapalli [1]

Abstract

We introduce a randomized topological augmentor based on Schur complements for Graph Contrastive Learning (GCL). Given a graph laplacian matrix, the technique generates unbiased approximations of its Schur complements and treats the corresponding graphs as augmented views. We discuss the benefits of our approach, provide theoretical justifications and present connections with graph diffusion. Unlike previous efforts, we study the empirical effectiveness of the augmentor in a controlled fashion by varying the design choices for subsequent GCL phases, such as encoding and contrasting. Extensive experiments on node and graph classification benchmarks demonstrate that our technique consistently outperforms pre-defined and adaptive augmentation approaches to achieve state-of-the-art results.

1. Introduction

Understanding the structural properties and semantics of graph data typically requires domain expertise and efficient computational tools. Advances in machine learning techniques such as Graph Neural Networks (GNN) (Gori et al., 2005; Scarselli et al., 2008; Bruna et al., 2014; Henaff et al., 2015; Welling & Kipf, 2016; Niepert et al., 2016; Bronstein et al., 2017; Xu et al., 2018; Wu et al., 2020; Zhou et al., 2020) have paved a path for representation learning on internet scale graphs and significantly alleviated the human effort. However, real-world graphs such as social networks, citation networks, supply chains and media networks are continuously evolving, which makes labeling an extremely challenging task to accomplish. Self-supervised learning (SSL) addresses this issue by optimizing pretext objectives and learning generalizable representations for downstream tasks (Jin et al., 2020; Wu et al., 2021; Liu et al., 2022; Xie et al., 2022). This learning paradigm has been quite popular for vision (Gidaris et al., 2018; Hjelm et al., 2018; Chen et al., 2020; Jing & Tian, 2020), language domains (Mikolov et al., 2013; Devlin et al., 2018; Radford et al., 2018; Lan et al., 2019) and is relatively new to graphs.

Contrastive learning on graphs (Velickovic et al., 2019; Sun et al., 2019; Zhu et al., 2020; You et al., 2020; Hassani & Khasahmadi, 2020) is a variant of SSL, whose pretext task is aimed at maximizing representation agreement across augmented views. A typical GNN-based GCL framework comprises three main components: 1. Data augmentors, 2. GNN encoders and 3. Contrastive objectives with corresponding modes. Data augmentation techniques in the literature can be categorized based on: 1. Feature masking/perturbations (You et al., 2020; Thakoor et al., 2021), 2. Structural perturbations (You et al., 2020; Hassani & Khasahmadi, 2020; Zeng & Xie, 2021), 3. Sub-graph sampling (Hu et al., 2019; Qiu et al., 2020; Jiao et al., 2020; Zhu et al., 2021a), 4. Adaptive and learnable structural perturbations (Zhu et al., 2021c; Yin et al., 2022). Efforts leveraging these techniques tend to focus on the collective comparison of GCL frameworks and lack controlled experimentation pertaining to the augmentation phase.

A recent empirical study on GCL by Zhu et al. (2021b) partially addresses this issue and compares some of the widely used augmentors with a fixed encoder design and contrastive objective. In particular, the study focuses on techniques such as feature masking, node dropping, edge perturbations, subgraph sampling based on random walks, diffusions based on Personalized Page Rank (PPR), Heat Kernel (HK) and Markov Diffusion Kernel (MDK). From a topological augmentation perspective, pre-defined stochastic techniques such as node dropping and edge perturbations tend to be faster in practice but fail to preserve structural information of the original graph (Hübler et al., 2008). Surprisingly, views obtained by such simple techniques have been shown to outperform diffusion-based strategies when contrasting solely based on node embeddings. Despite this parity, Zhu et al. (2021b) show that, when diffusion-based approaches are combined with node dropping, there is a substantial increase in the unsupervised node and graph classification performance. An attempt to avoid such trial and error based design of augmentors was made by Yin et al. (2022), where learnable augmentors are trained in an end-to-end fashion to generate semantic label preserving views. Such a technique

[1]Courant Institute of Mathematical Sciences, New York University, New York, USA. Correspondence to: Vignesh Kothapalli <vk2115@nyu.edu>.

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

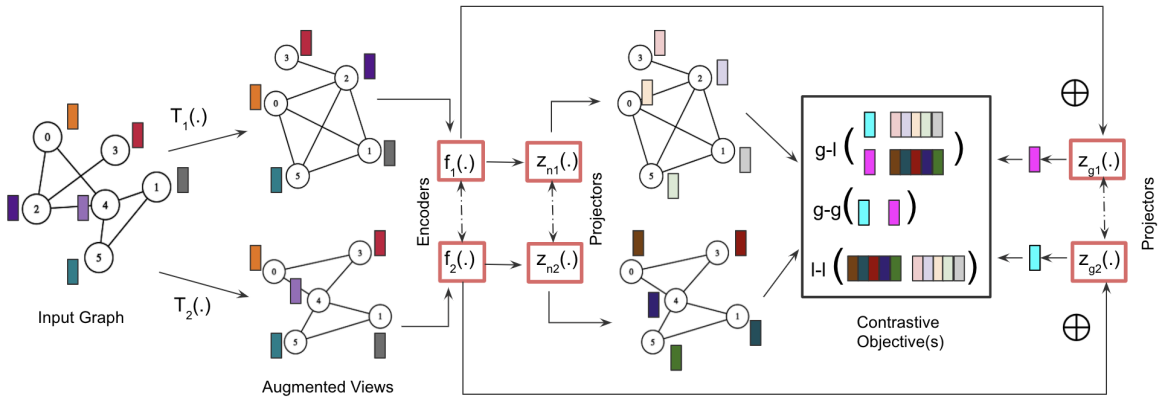


Figure 1. A generalized dual branch GCL framework. The input graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, w)$ is augmented by T_1, T_2 to generate graph views. These graph views are encoded using GNNs f_1, f_2 to compute node embeddings. The node embeddings are projected using MLPs z_{n1}, z_{n2} to generate node features. Additionally, the node embeddings are aggregated using a permutation invariant function $\bigoplus$ and projected using z_{g1}, z_{g2} to compute graph-level features. Depending on the contrastive mode, the graph and node features are contrasted.

is limited to node-dropping operations and induces non-negligible computational overheads if edge perturbations were to be included. Furthermore, since learnable augmentors modify learning objectives, leveraging them to explore and design new GCL objectives can be challenging. On the other hand, the adaptive augmentation technique proposed by Zhu et al. (2021c) drops edges based on their degree, page rank or eigenvector centrality scores and preserves pre-defined structural information across views. However, metrics such as eigenvector centrality are expensive to compute and negate the performance benefits of augmentors. These observations highlight the persisting augmentation problem of **identifying** and **preserving** structural properties across **stochastically** generated views in a **computationally efficient** manner.

In our work, we leverage the properties of Schur complements to address these issues. Schur complements are typically obtained during Gaussian Elimination (GE) and are pervasive in numerical analysis, probability and statistics (Cottle, 1974; Ouellette, 1981; Zhang, 2006). From a graph theoretic perspective, GE essentially leads to node dropping and edge perturbations such that the resulting Schur complements preserve random walk transition probabilities of the remaining nodes (with respect to the original graph). Our technique exploits this property and generates randomized yet unbiased approximations of Schur complements as the augmented views. Additionally, we exploit the role of Schur complements in matrix inversions and showcase computational optimizations for diffusion-based augmentors. Overall, the potential benefits of this class of topological augmentors remain unexplored for GCL and we take a step towards understanding them. To summarize, our contributions are as follows:

- We design a randomized Schur complement based topological augmentor for GCL which is fast, stochastic and effective on a wide range of framework designs.
- We show that by preserving the combinatorial properties of Schur complements in expectation, the augmentor achieves state-of-the-art results on unsupervised node and graph classification tasks.
- Using our technique, we present an efficient approach to achieve diffusion followed by sub-graph sampling.

2. Preliminaries

2.1. Notations

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E}, w)$ denote an undirected graph with nodes $\mathcal{V} = \{v_i\}_{i=1}^N$ and edges $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$. An edge between nodes v_i, v_j is given by $e_{v_i v_j} = e_{ij} \in \mathcal{E}$. We drop the subscript in e_{ij} when indexing can be avoided. The neighborhood of v_i is given by $\mathcal{N}(v_i)$. For notational convenience, $v_i \in e$ represents a node $v_i \in \mathcal{V}$ being on either end of an edge $e \in \mathcal{E}$. A weight function w maps an edge $e_{ij} \in \mathcal{E}$ to its weight $w(e_{ij}) = w(v_i, v_j) \in \mathbb{R}$. Also, w is overloaded to denote the total weight of a node as the sum of weights of edges incident on it: $w(v_i) = \sum_{e \in \mathcal{E}, v_i \in e} w(e)$. Similarly, the degree of a node $deg(v_i) = |\{e \in \mathcal{E} : v_i \in e\}|$ is the total number of edges incident on it. The feature matrix of $\mathcal{G}$ is given by $\mathbf{X} \in \mathbb{R}^{N \times d}$ where features of a node v_i are given by the row $\mathbf{x}_i \in \mathbb{R}^d$. Finally, we consider the graph $\mathcal{G}$ to be associated with a weighted positive semi-definite laplacian matrix $\mathbf{L} \in \mathbb{R}^{N \times N}$.

2.2. Graph Contrastive Learning Frameworks

In this section, we briefly describe the functionality of GCL framework components (see Figure 1) and defer additional details on the mathematical formulations to Appendix D.

2.2.1. Augmentors

Without loss of generality, let $\mathcal{T}$ represent a class of augmentation functions from which $T_1, T_2 \sim \mathcal{T}$ are chosen. These functions transform $\mathcal{G}$ to the respective augmented views $T_1(\mathcal{G}) = \widetilde{\mathcal{G}}_1, T_2(\mathcal{G}) = \widetilde{\mathcal{G}}_2$ by perturbing the topology, features or both. Hassani & Khasahmadi (2020) observed that, when $\mathcal{T}$ represents a class of diffusion-based augmentors, contrasting more than 2 views didn't improve the performance on the downstream node and graph classification tasks. Additionally, the approach of Velickovic et al. (2019) uses only a single view to contrast the node level and graph level embeddings. Analyzing the effect of the number of views based on choices of $\mathcal{T}$ is still an open problem. Meanwhile, for the rest of this paper, we default it to 2.

2.2.2. Encoders and Projectors

Depending on design choices for the framework, one can choose either a shared encoder (You et al., 2020) or dedicated encoders (Hassani & Khasahmadi, 2020) to compute the node embeddings for $\widetilde{\mathcal{G}}_1, \widetilde{\mathcal{G}}_2$. Typically, GNNs based on Graph Convolution Network (GCN) (Kipf & Welling, 2016a) or Graph Isomorphism Network (GIN) (Xu et al., 2018) are employed. We denote the node level embeddings of $\widetilde{\mathcal{G}}_1, \widetilde{\mathcal{G}}_2$ as $\mathbf{H}_{\widetilde{\mathcal{G}}_1}, \mathbf{H}_{\widetilde{\mathcal{G}}_2} \in \mathbb{R}^{N \times d_h}$. Next, an MLP-based projector is employed to project the embeddings onto a latent space. We denote the projected node level features of $\widetilde{\mathcal{G}}_1, \widetilde{\mathcal{G}}_2$ as $\mathbf{Z}_{\widetilde{\mathcal{G}}_1}, \mathbf{Z}_{\widetilde{\mathcal{G}}_2} \in \mathbb{R}^{N \times d_z}$. Additionally, the node embeddings are aggregated using a permutation invariant function $\bigoplus$ (eg: mean) and projected to compute graph level features[1] $\mathbf{z}_{\widetilde{\mathcal{G}}_1}, \mathbf{z}_{\widetilde{\mathcal{G}}_2} \in \mathbb{R}^{d_z}$.

2.2.3. Contrastive modes and objectives

With $\mathbf{Z}_{\widetilde{\mathcal{G}}_1}, \mathbf{Z}_{\widetilde{\mathcal{G}}_2} \in \mathbb{R}^{N \times d_z}$ and $\mathbf{z}_{\widetilde{\mathcal{G}}_1}, \mathbf{z}_{\widetilde{\mathcal{G}}_2} \in \mathbb{R}^{d_z}$ being available, a relevant contrastive objective is chosen for optimization. Popular choices include: Noise Contrastive Estimation based InfoNCE (Oord et al., 2018), Jensen-Shannon Divergence (JSD) (Lin, 1991), Bootstrapping Latent (BL) (Grill et al., 2020), Variance-Invariance-Covariance Regularization (VICReg) (Bardes et al., 2021) and Barlow Twins (BT) (Zbontar et al., 2021) losses. Finally, a mode is chosen to determine the granularity of contrasting:

1. **local-local (l-l)**: The objective is solely dependent on the node-level features.
2. **global-local (g-l)**: The objective is dependent on both node and graph-level features.
3. **global-global (g-g)**: The objective is solely dependent on the graph-level features.

The choice of mode and objective not only affects the GCL framework performance but also plays a key role in computational efficiency. For instance, Zhu et al. (2021b) empirically show that, in local-local mode, losses such as BL, BT, and VICReg consume $\approx 3\times$ less memory than InfoNCE and JSD while achieving comparable performance on node and graph classification tasks. Finally, when it comes to objectives such as InfoNCE and JSD, choosing the optimal set of negative samples is not straightforward. Especially, the difficulty lies in achieving a balance between computational overheads of negative sampling (Chuang et al., 2020) (with hard negatives if required by design (Robinson et al., 2020)) and performance improvements.

2.3. Schur Complements and Gaussian Elimination

Every edge $e_{ij} \in \mathcal{E}$ of graph $\mathcal{G}$ forms an elementary laplacian given by the outer product: $\mathbf{\Delta}_{ij} = (\delta_i - \delta_j)(\delta_i - \delta_j)^*$. Here, $\delta_i \in \mathbb{R}^N$ denotes the standard basis vector at index i and * denotes the adjoint. The weighted Laplacian $\mathbf{L}$ can now be formulated as follows:

$$\mathbf{L} = \sum_{e_{ij} \in \mathcal{E}} w(e_{ij}) \mathbf{\Delta}_{ij} \tag{1}$$

Since every $\mathbf{\Delta}_{ij}$ is positive semi-definite, so is their weighted sum $\mathbf{L}$. The off-diagonal elements of $\mathbf{L}$ are given by: $(\mathbf{L})_{ij} = -w(e_{ij}), i \neq j$ and the diagonal elements by $(\mathbf{L})_{ii} = w(v_i) = \sum_{v_j \in \mathcal{N}(v_i)} w(e_{ij})$. Now, without loss of generality, consider the matrix representation of $\mathbf{L}$ as:

$$\mathbf{L} = \begin{bmatrix} a_{11} & \mathbf{b}^* \\ \mathbf{b} & \mathbf{L}_2 \end{bmatrix} \tag{2}$$

Where $a_{11} \in \mathbb{R}, \mathbf{b} \in \mathbb{R}^{N-1}$ and $\mathbf{L}_2 = \mathbb{R}^{N-1 \times N-1}$. The Schur complement of $\mathbf{L}$ with respect to nodes $\mathcal{V} \backslash v_1$ is a positive semi-definite matrix that is obtained when v_1 is eliminated via a GE step:

$$\begin{aligned} SC(\mathbf{L}, \mathcal{V} \backslash v_1) &= \mathbf{L} - \frac{1}{a_{11}} \begin{bmatrix} a_{11} \\ \mathbf{b} \end{bmatrix} \begin{bmatrix} a_{11} \\ \mathbf{b} \end{bmatrix}^* \\ &= \begin{bmatrix} 0 & \mathbf{0}^* \\ \mathbf{0} & \mathbf{L}_2 - a_{11}^{-1} \mathbf{b}\mathbf{b}^* \end{bmatrix} \end{aligned} \tag{3}$$

Alternatively, a graph theoretic interpretation of GE states that $SC(\mathbf{L}, \mathcal{V} \backslash v_1)$ is obtained by removing the *Star* graph corresponding to v_1 from $\mathcal{G}$ and adding the induced *Clique* graph. Formally:

$$SC(\mathbf{L}, \mathcal{V} \backslash v_1) = \mathbf{L} - \textit{STAR}(\mathbf{L}, v_1) + \textit{CLIQUE}(\mathbf{L}, v_1) \tag{4}$$

[1] Designs such as MVGRL (Hassani & Khasahmadi, 2020) employ MLP projections to improve the graph level representations.

Where $STAR(\mathbf{L}, v_i), CLIQUE(\mathbf{L}, v_i)$ for any node $v_i \in \mathcal{V}$ are given by:

$$STAR(\mathbf{L}, v_i) = \sum_{e_{ij} \in \mathcal{E}, v_j \in \mathcal{N}(v_i)} w(e_{ij}) \mathbf{\Delta}_{ij}$$
$$CLIQUE(\mathbf{L}, v_i) = \frac{1}{2w(v_i)} \sum_{e_{ij} \in \mathcal{E}} \sum_{e_{ik} \in \mathcal{E}} w(e_{ij}) w(e_{ik}) \mathbf{\Delta}_{jk} \tag{5}$$

This result indicates that $SC(\mathbf{L}, \mathcal{V} \backslash v_1)$ is a weighted Laplacian matrix with an associated graph. For simplicity, we denote $SC(\mathbf{L}, \mathcal{V} \backslash v_1)$ as $SC(\mathcal{G}, \mathcal{V} \backslash v_1)$ when the notion of a graph is suitable for the context. Schur complements hold an interesting graph theoretic property that, the random walk transition probabilities of the remaining nodes $\mathcal{V} \backslash v_1$ through the eliminated vertex v_1 (with respect to $\mathcal{G}$) are preserved in $SC(\mathcal{G}, \mathcal{V} \backslash v_1)$. Intuitively, the list of nodes visited by random walks on $SC(\mathcal{G}, \mathcal{V} \backslash v_1)$ is equivalent in distribution to the list of nodes in $\mathcal{V} \backslash v_1$ visited by random walks on $\mathcal{G}$. Refer to section 3 in Durfee et al. (2019) and section 4.1 in Gao et al. (2022) for detailed discussions on this property.

Algorithm 1 $rLap$

Input: graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, w)$, node drop fraction γ, node elimination scheme o_v, neighbor ordering scheme o_n.
set $\mathbf{R} = \mathbf{L}$
repeat
  $\mathcal{X}_{\mathbf{R}}(v_i) = o_n(\mathcal{N}_{\mathbf{R}}(v_i))$
  $\mathbf{C} = \mathbf{0}$
  for $l = 1$ **to** $|\mathcal{X}_{\mathbf{R}}(v_i)| - 1$ **do**
    // choose x_q based on the conditional probability
    $P(x_q|x_l) = \frac{w_{\mathbf{R}}(v_i, x_q)}{w_{\mathbf{R}}(v_i) - \sum_{k=1}^{l} w_{\mathbf{R}}(v_i, x_k)}; x_{l<q} \in \mathcal{X}_{\mathbf{R}}(v_i)$
    // compute weight of the new edge between x_l, x_q
    $w_{\mathbf{R}}(x_l, x_q) = \frac{w_{\mathbf{R}}(v_i, x_l) \cdot (w_{\mathbf{R}}(v_i) - \sum_{k=1}^{l} w_{\mathbf{R}}(v_i, x_k))}{w_{\mathbf{R}}(v_i)}$
    $\mathbf{C} = \mathbf{C} + w_{\mathbf{R}}(x_l, x_q) \cdot \mathbf{\Delta}_{x_l x_q}$
  end for
  $\mathbf{R} = \mathbf{R} - \sum_{e_{ij} \in \mathcal{E}_{\mathbf{R}}, v_j \in \mathcal{N}_{\mathbf{R}}(v_i)} w_{\mathbf{R}}(e_{ij}) \mathbf{\Delta}_{ij}$
  $\mathbf{R} = \mathbf{R} + \mathbf{C}$
until all the $\gamma|\mathcal{V}|$ nodes are eliminated based on o_v
return $\mathbf{R}$

3. Methodology

To leverage these combinatorial properties of Schur complements in our contrastive views, two issues need to be addressed: The $O(N^2)$ overhead to compute a clique and a lack of stochasticity in the gaussian elimination procedure. We address both these issues by developing a clique approximation procedure that inherently introduces randomness. Thus, by eliminating nodes via GE and computing unbiased approximations of the induced cliques, our augmentation technique generates randomized Schur complements which preserve the random walk transition probabilities of the remaining nodes in expectation.

The need for stochasticity is based on the empirical results of Zhu et al. (2021b), where introducing randomness to the views (eg: combining diffusion with node dropping) improved contrastive learning performance on downstream classification tasks. Although a rigorous analysis of such behavior is not present in the literature, we provide interesting insights on GCL performance due to randomness introduced by our approximation procedure.

3.1. Randomized Schur Complements

We present our Schur complement approximation procedure in Algorithm 1.[2] We name it $rLap$ to denote the laplacian nature of its output. The input to $rLap$ is the graph $\mathcal{G}$, the fraction of nodes to eliminate γ, node elimination scheme o_v and the neighbor ordering scheme o_n. The output is a randomized Schur complement of $\mathcal{G}$ after eliminating $\gamma|\mathcal{V}|$ nodes[3]. The scheme o_v indicates the order in which $\gamma|\mathcal{V}|$ nodes are eliminated. The possibilities for such a scheme are huge but we limit our analysis to random ordering and ordering based on node degree. In the 'random' scheme, a node is randomly selected at each step of the outer loop. In the 'degree' scheme, a priority queue is maintained to eliminate nodes based on their degree, i.e. nodes with lower degrees are eliminated before the highly connected ones.

Now, without loss of generality, consider the ith iteration of the outer loop where node v_i is being eliminated. The first step is to order the neighbors $\mathcal{N}_{\mathbf{R}}(v_i)$ and store the result in $\mathcal{X}_{\mathbf{R}}(v_i)$ [4]. The order decided by o_n affects the arrangement of edges in the approximated clique $\mathbf{C}$. Specifically, while iterating over the neighbors $x_l \in \mathcal{X}_{\mathbf{R}}(v_i), l \in \{1, \cdots, |\mathcal{X}_{\mathbf{R}}(v_i)| - 1\}$ in the inner loop, we compute a conditional probability $P(x_q|x_l)$ of choosing a neighbor $x_q \in \mathcal{X}_{\mathbf{R}}(v_i), q \in \{l+1, \cdots, |\mathcal{X}_{\mathbf{R}}(v_i)|\}$, based on which, a new edge between x_l, x_q is created. The choice of o_n affects this conditional probability and can lead to sparser or denser variants of approximated cliques. Now, the weighted elementary laplacian of the newly formed edge $w_{\mathbf{R}}(x_l, x_q)\mathbf{\Delta}_{x_l x_q}$ is computed and added to $\mathbf{C}$ [5]. Note that, the inner loop iterates over $O(N)$ nodes for approximating the clique and avoids the $O(N^2)$ overhead for exact compu-

[2]The code is available at: https://github.com/kvignesh1420/rlap

[3]Elimination in this context indicates that a node is disconnected from all its neighbors. The dimensions of $\mathbf{R}$ remain $\mathbb{R}^{N \times N}$ with corresponding row and column set to $\mathbf{0}$.

[4]$\mathcal{E}_{\mathbf{R}}, w_{\mathbf{R}}, \mathcal{N}_{\mathbf{R}}$ are indexed by $\mathbf{R}$ to denote the current state.

[5]Here x_l, x_q are indexed on $\mathcal{X}_{\mathbf{R}}(v_i)$ for notational convenience, but have a one-to-one correspondence with our original node set $\mathcal{V}$

tation. Finally, $STAR(\mathbf{R}, v_i)$ is subtracted and $\mathbf{C}$ is added to $\mathbf{R}$ to continue the elimination process.

Theorem 3.1. *Given an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, w)$, node dropping fraction γ, node elimination scheme o_v and neighbor ordering scheme o_n, the output of $rLap(\mathcal{G}, \gamma, o_v, o_n)$ is an unbiased estimator of the schur complement $SC(\mathcal{G}, \mathcal{V}_{\mathbf{R}_{\gamma|\mathcal{V}|}})$, where $\mathcal{V}_{\mathbf{R}_{\gamma|\mathcal{V}|}} \subset \mathcal{V}$ denotes the set of nodes that remain after $\gamma|\mathcal{V}|$ eliminations.*

Proof. Without loss of generality, let $\mathbf{R}_{i-1}$ be the state after eliminating $i-1$ nodes. Let v_i indicate the node which is being eliminated in the ith iteration of the outer loop. The proof is based on the loop invariant that $\mathbb{E}[\mathbf{R}_i] = SC(\mathcal{G}, \mathcal{V}_{\mathbf{R}_{i-1}} \backslash v_i)$ after the end of this iteration. With this guarantee, we continue the elimination process for $\gamma|\mathcal{V}|$ iterations and achieve the desired randomized Schur complement which equals $SC(\mathcal{G}, \mathcal{V}_{\mathbf{R}_{\gamma|\mathcal{V}|}})$ under expectation. The proof is available in Appendix B along with the tail bounds of deviation for the laplacian matrix martingale. □

3.2. Connections with Graph Diffusion

For a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, w)$, the diffusion operator is a polynomial filter on $\mathcal{G}$ which mitigates the effect of noisy neighbors. The diffused output $\mathbf{S}$ is given by:

$$\mathbf{S} = \sum_{i=0}^{\infty} \rho_i \mathbf{T}^i \tag{6}$$

Where $\mathbf{T}$ is a transition matrix (Klicpera et al., 2019) and can be chosen as either $\mathbf{T}_{rw} = \mathbf{A}\mathbf{D}^{-1}$ or $\mathbf{T}_{sym} = \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$, $\mathbf{A}, \mathbf{D}$ are the adjacency and degree matrices for $\mathcal{G}$ respectively, ρ_i is the scaling coefficient with $\rho_i^{PPR} = \alpha \cdot (1-\alpha)^i, \alpha \in (0,1)$ and $\rho_i^{HK} = e^{-t} \cdot \frac{t^i}{i!}$ representing the coefficients for personalized page rank (Page et al., 1999) and heat kernels with diffusion time t (Kondor & Lafferty, 2002; Chung, 2007) respectively. When $\mathbf{T} = \mathbf{T}_{rw} = \mathbf{A}\mathbf{D}^{-1}$, the closed forms of diffusions are given by:

$$\begin{aligned} \mathbf{S}^{PPR} &= \alpha(\mathbf{I} - (1-\alpha)\mathbf{A}\mathbf{D}^{-1})^{-1} \\ \mathbf{S}^{HK} &= \exp(t \cdot \mathbf{A}\mathbf{D}^{-1} - t\mathbf{I}) \end{aligned} \tag{7}$$

Note that, for $\mathbf{S}^{PPR}$, computing $(\mathbf{I} - (1-\alpha)\mathbf{A}\mathbf{D}^{-1})^{-1}$ is an expensive operation over the entire graph and leads to undesired bottlenecks when the desired augmented view during contrasting is just a relatively smaller sub-graph. We address this issue in the following theorem using $rLap$.

Theorem 3.2. *Let $\rho_i = c \cdot \beta^i \in \mathbb{R}, \beta \to 1$. The sub-graph with nodes $\mathcal{V}_s$ sampled from $\mathbf{S} = (\mathbf{I} - (1-\alpha)\mathbf{A}\mathbf{D}^{-1})^{-1}$ is given by $\mathbf{S}_{\mathcal{V}_s} = \left(\lim_{K\to\infty} \sum_{k=0}^{K} c \cdot \beta^k \cdot (\mathbf{A}\mathbf{D}^{-1})^k \right)_{\mathcal{V}_s} = \mathbb{E}\left[\lim_{K\to\infty} \sum_{k=0}^{K} c \cdot \beta^k \cdot \mathbf{D}_{\mathcal{V}_s} \widetilde{\mathbf{D}}_{\mathbf{R}}^{-1} (\widetilde{\mathbf{A}}_{\mathbf{R}} \widetilde{\mathbf{D}}_{\mathbf{R}}^{-1})^k \right]$, where $\mathbf{R}$ is the approximation of schur complement $SC(\mathcal{G}, \mathcal{V}_s), \mathcal{V}_s \subset \mathcal{V}$ given by $rLap$, $\mathbf{D}_{\mathcal{V}_s}$ is a diagonal matrix formed by selecting entries corresponding to $\mathcal{V}_s$ from $\mathbf{D}$ and $\widetilde{\mathbf{A}}_{\mathbf{R}}, \widetilde{\mathbf{D}}_{\mathbf{R}}$ represent the adjacency and degree matrices w.r.t $\mathbf{R}$.*

Proof. The proof is a simple expansion of the limit:

$$\begin{aligned} &\mathbb{E}\left[\lim_{K\to\infty} \textstyle\sum_{k=0}^{K} c \cdot \beta^k \mathbf{D}_{\mathcal{V}_s} \widetilde{\mathbf{D}}_{\mathbf{R}}^{-1} (\widetilde{\mathbf{A}}_{\mathbf{R}} \widetilde{\mathbf{D}}_{\mathbf{R}}^{-1})^k \right] \\ &= c\mathbf{D}_{\mathcal{V}_s} \mathbb{E}\big[\widetilde{\mathbf{D}}_{\mathbf{R}}^{-1} \lim_{K\to\infty} \textstyle\sum_{k=0}^{K} (\beta \widetilde{\mathbf{A}}_{\mathbf{R}} \widetilde{\mathbf{D}}_{\mathbf{R}}^{-1})^k\big] \\ &= c\mathbf{D}_{\mathcal{V}_s} \mathbb{E}\big[\widetilde{\mathbf{D}}_{\mathbf{R}}^{-1} \cdot (\mathbf{I} - \beta \widetilde{\mathbf{A}}_{\mathbf{R}} \widetilde{\mathbf{D}}_{\mathbf{R}}^{-1})^{-1}\big] \\ &= c\mathbf{D}_{\mathcal{V}_s} \mathbb{E}\big[(\widetilde{\mathbf{D}}_{\mathbf{R}} - \beta \widetilde{\mathbf{A}}_{\mathbf{R}})^{-1}\big] \\ &= c\mathbf{D}_{\mathcal{V}_s} \cdot SC(\mathbf{D} - \beta\mathbf{A}, \mathcal{V}_s)^{-1} \\ &= c\mathbf{D}_{\mathcal{V}_s} (\mathbf{D} - \beta\mathbf{A})_{\mathcal{V}_s}^{-1} = \mathbf{S}_{\mathcal{V}_s} \end{aligned}$$

The equality $\mathbb{E}[(\widetilde{\mathbf{D}}_{\mathbf{R}} - \beta\widetilde{\mathbf{A}}_{\mathbf{R}})^{-1}] = SC(\mathbf{D} - \beta\mathbf{A}, \mathcal{V}_s)^{-1}$ is based on the guarantees of Theorem 3.1 when $\beta \to 1$ and $\mathbf{D} - \beta\mathbf{A} \to \mathbf{L}$. The equality $SC(\mathbf{D} - \beta\mathbf{A}, \mathcal{V}_s)^{-1} = (\mathbf{D} - \beta\mathbf{A})_{\mathcal{V}_s}^{-1}$ is based on the standard matrix inversion lemma. Refer to Gallier (2010) and Appendix C.4 in Boyd et al. (2004) for further details. □

When β is not close to 1, $\mathbf{D} - \beta\mathbf{A}$ doesn't represent a graph laplacian but instead represents a symmetric diagonally dominant matrix (SDDM). Similar to the laplacian property of Schur complements, it can be shown that Schur complements of SDDM matrices are also SDDM (refer Appendix A, Lemma A.1 in Fahrbach et al. (2020) for the proof). To better understand the implications of Theorem 3.2, consider a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, w)$ on which the PPR diffusion operator needs to be applied, followed by a sub-graph sampling operation with respect to nodes $\mathcal{V}_s \subset \mathcal{V}$. Such a requirement is pervasive in GCL when views are based on diffusion matrices. However, the overheads of computing $\mathbf{S}$ for medium-large scale graphs can be significant. Based on the empirical analysis of Klicpera et al. (2019), the optimal choice of α for $\rho_i = \alpha \cdot (1-\alpha)^i$ typically lies in a close range of $\alpha \in [0.05, 0.2]$. This implies $\beta = 1 - \alpha$ is close to 1, which justifies our assumption in Theorem 3.2 for practical settings. Thus, instead of diffusion followed by sub-graph sampling, one can apply $rLap$ on $\mathcal{G}$ to obtain the randomized Schur complement which is relatively sparse, followed by the diffusion operator (up to $\mathbf{D}_{\mathcal{V}_s} \widetilde{\mathbf{D}}_{\mathbf{R}}^{-1}$ scaling). See Figure 2 for an illustration.

3.3. Augmentation with rLap

We present a generalized GCL framework with $rLap$ as the augmentor in Algorithm 2. Given a collection of graphs $G = \{\mathcal{G}^i, i \in \{1, \cdots, M\}, M \geq 1$, the blueprint presented in Algorithm 2 can be used for learning node and graph features via different design choices for encoders (including projectors) and contrastive objectives. For every graph $\mathcal{G}^i \in G$, $rLap$ generates the randomized schur complement

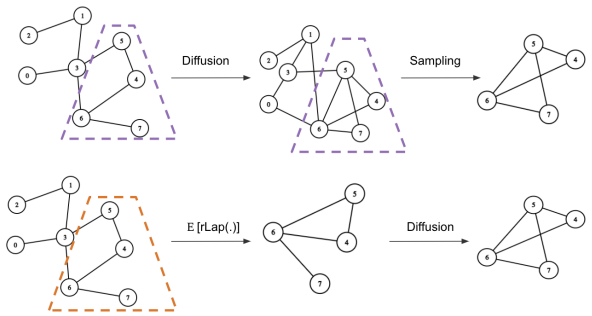


Figure 2. Illustration of diffusion + sampling vs $rLap$ + diffusion with a toy example. In the first row, diffusion is applied on a graph $\mathcal{G}$ followed by sampling the sub-graph corresponding to nodes: $\mathcal{V}_s = \{4, 5, 6, 7\}$. In the second row, the expected output of $rLap$ on $\mathcal{G}$ with $\mathcal{V}_s = \{4, 5, 6, 7\}$ followed by diffusion on the randomized Schur complement gives the same desired sub-graph.

views $\widetilde{\mathcal{G}}_1^i, \widetilde{\mathcal{G}}_2^i$, for which, the graph features $\mathbf{z}_{\widetilde{\mathcal{G}}_1^i}, \mathbf{z}_{\widetilde{\mathcal{G}}_2^i}$ and node features $\mathbf{Z}_{\widetilde{\mathcal{G}}_1^j}, \mathbf{Z}_{\widetilde{\mathcal{G}}_2^j}$ are learnt by the encoders and projectors. The learning objective depends on the contrastive mode m, which can be either 'l-l', 'g-l' or 'g-g' as defined in the preliminaries. Most of the GCL frameworks in the literature can be derived from this blueprint. For instance, by replacing $rLap$ with a uniform node dropping augmentor, using shared GCN-based encoders $f_1 = f_2 = f(.)$, MLP-projectors $z_{g1} = z_{g2} = z_g(.)$, 'g-g' contrastive mode and InfoNCE loss $\mathcal{L}_{l-l}$, we obtain the widely used GraphCL framework (You et al., 2020). See Appendix D for illustrations and details.

However, recent efforts in developing augmentation techniques tend to restrict the design choices of encoders and objectives when comparing the overall GCL performance with prior works. Such an approach fails to isolate the effectiveness and limitations of an augmentor from other design choices. We break this norm and follow established benchmark practices of Zhu et al. (2021b) to perform controlled experiments in the following section.

4. Experiments

Owing to the framework-agnostic nature of augmentors, we conduct controlled experiments and evaluate their effectiveness on unsupervised node and graph classification tasks under the linear evaluation protocol (Velickovic et al., 2019). The controlled settings pertain to fixing the shared/dedicated nature of encoder(s), contrastive modes, and objectives (see Table 1). Especially, we leverage the designs of 4 widely used GCL frameworks: GRACE (Zhu et al., 2020), MVGRL (Hassani & Khasahmadi, 2020), GraphCL (You et al., 2020) and BGRL (Thakoor et al., 2021). Comprehensive details on experimental settings, datasets, augmentors, design choices, evaluation protocols and baselines are available in

Algorithm 2 A generalized GCL framework with $rLap$.

Input: Graph(s) $G = \{\mathcal{G}^i, i \in \{1, \cdots, M\}$, node elimination scheme o_v, neighbor ordering scheme o_n, fraction of nodes to eliminate for both views γ_1, γ_2, encoders f_1, f_2, node level projectors z_{n1}, z_{n2}, aggregation function $\bigoplus$, graph level projectors z_{g1}, z_{g2}, mode m.
repeat
  for $\mathcal{G}^i$ in G **do**
    $\widetilde{\mathcal{G}}_1^i = rLap(\mathcal{G}^i, \gamma_1, o_v, o_n)$ // first view
    $\mathbf{H}_{\widetilde{\mathcal{G}}_1^i} = f_1(\widetilde{\mathcal{G}}_1^i)$ //node embeddings
    $\mathbf{Z}_{\widetilde{\mathcal{G}}_1^i}, \mathbf{z}_{\widetilde{\mathcal{G}}_1^i} = z_{n1}(\mathbf{H}_{\widetilde{\mathcal{G}}_1^i}), z_{g1}(\bigoplus(\mathbf{H}_{\widetilde{\mathcal{G}}_1^i}))$ // features
    $\widetilde{\mathcal{G}}_2^i = rLap(\mathcal{G}^i, \gamma_2, o_v, o_n)$ // second view
    $\mathbf{H}_{\widetilde{\mathcal{G}}_2^i} = f_2(\widetilde{\mathcal{G}}_2^i)$ //node embeddings
    $\mathbf{Z}_{\widetilde{\mathcal{G}}_2^i}, \mathbf{z}_{\widetilde{\mathcal{G}}_2^i} = z_{n2}(\mathbf{H}_{\widetilde{\mathcal{G}}_2^i}), z_{g2}(\bigoplus(\mathbf{H}_{\widetilde{\mathcal{G}}_2^i}))$ // features
  end for
  // compute objectives based on contrastive modes
  set $\ell = 0$
  for $j = 1, 2, \cdots, M$ and $k = 1, 2, \cdots, M$ **do**
    if $m =$ "*l-l*" **then**
      $\ell = \ell + \mathcal{L}_{l-l}(\mathbf{Z}_{\widetilde{\mathcal{G}}_1^j}, \mathbf{Z}_{\widetilde{\mathcal{G}}_2^j}, \mathbf{Z}_{\widetilde{\mathcal{G}}_1^k}, \mathbf{Z}_{\widetilde{\mathcal{G}}_2^k})$
    else if $m =$ "*g-l*" **then**
      $\ell = \ell + \mathcal{L}_{g-l}(\mathbf{Z}_{\widetilde{\mathcal{G}}_1^j}, \mathbf{Z}_{\widetilde{\mathcal{G}}_2^k}, \mathbf{z}_{\widetilde{\mathcal{G}}_1^j}, \mathbf{z}_{\widetilde{\mathcal{G}}_2^k})$
    else if $m =$ "*g-g*" **then**
      $\ell = \ell + \mathcal{L}_{g-g}(\mathbf{z}_{\widetilde{\mathcal{G}}_1^j}, \mathbf{z}_{\widetilde{\mathcal{G}}_2^j}, \mathbf{z}_{\widetilde{\mathcal{G}}_1^k}, \mathbf{z}_{\widetilde{\mathcal{G}}_2^k})$
    end if
  end for
  // compute gradients with appropriate normalization $\mathcal{Z}$
  $\nabla_{f_1, f_2, z_{n1}, z_{n2}, z_{g1}, z_{g2}} \frac{1}{\mathcal{Z}}(\ell)$
until convergence/tolerance

Appendix D. In the following results for $rLap$, we employ a 'random' node selection scheme as o_v and an ordering scheme based on increasing order of edge weights as o_n. A detailed ablation study on these schemes is presented in Appendix C.

Table 1. Control settings for evaluating augmentors.

SETTING	DESIGN CHOICES
DATASET	**EVAL**
AUGMENTOR	**EVAL**
ENCODERS	SHARED, DEDICATED
MODE	*l-l, g-l, g-g*
OBJECTIVE	INFONCE, JSD, BL

4.1. Node Classification Results

For node classification tasks, we report the performance using GRACE and MVGRL-based designs in Table 2, 3 respectively. The GRACE design corresponds to shared encoders, 'l-l' contrastive mode, and InfoNCE objective.

Table 2. Evaluation (in accuracy) on benchmark node datasets with **GRACE** based design.

AUGMENTOR	CORA	AMAZON-PHOTO	PUBMED	COAUTHOR-CS	COAUTHOR-PHY
EDGEADDITION	83.34 ± 1.69	86.99 ± 0.57	84.03 ± 0.97	89.94 ± 0.72	95.53 ± 0.35
EDGEDROPPING	82.87 ± 2.39	90.8 ± 0.55	84.24 ± 0.87	92.53 ± 0.4	95.31 ± 0.43
EDGEDROPPINGDEGREE	83.42 ± 3.57	88.58 ± 1.13	83.88 ± 0.77	92.49 ± 0.49	95.47 ± 0.38
EDGEDROPPINGEVC	82.32 ± 1.81	90.74 ± 0.9	84.2 ± 0.6	92.3 ± 0.31	95.60 ± 0.37
EDGEDROPPINGPR	82.39 ± 1.31	92.01 ± 0.87	84.22 ± 0.74	92.45 ± 0.51	95.51 ± 0.25
MARKOVDIFFUSION	82.1 ± 3.11	90.91 ± 0.89	83.96 ± 0.91	92.69 ± 0.59	95.13 ± 0.43
NODEDROPPING	82.9 ± 1.8	92.15 ± 1.33	84.02 ± 0.9	92.81 ± 0.89	95.63 ± 0.32
PPRDIFFUSION	79.85 ± 2.07	91.16 ± 0.91	83.31 ± 1.26	92.75 ± 0.58	95.09 ± 0.34
RANDOMWALKSUBGRAPH	82.13 ± 2.89	89.76 ± 1.26	84.37 ± 0.59	92.51 ± 0.29	95.12 ± 0.33
RLAP	$\mathbf{83.75 \pm 2.64}$	$\mathbf{92.59 \pm 1.05}$	$\mathbf{84.56 \pm 0.71}$	$\mathbf{93.1 \pm 0.54}$	$\mathbf{95.83 \pm 0.44}$

Table 3. Evaluation (in accuracy) on benchmark node datasets with **MVGRL** based design.

AUGMENTOR	CORA	AMAZON-PHOTO	PUBMED	COAUTHOR-CS	COAUTHOR-PHY
EDGEADDITION	81.62 ± 4.01	87.64 ± 1.68	83.35 ± 0.75	91.44 ± 0.65	94.23 ± 0.21
EDGEDROPPING	83.49 ± 1.32	87.87 ± 1.33	83.73 ± 1.14	91.18 ± 0.43	94.68 ± 0.38
EDGEDROPPINGDEGREE	83.57 ± 2.29	88.11 ± 1.25	84.13 ± 0.79	91.32 ± 0.52	94.57 ± 0.39
EDGEDROPPINGEVC	82.79 ± 2.73	88.81 ± 1.46	84.14 ± 0.77	91.71 ± 0.46	94.62 ± 0.56
EDGEDROPPINGPR	83.42 ± 2.47	88.04 ± 1.36	83.76 ± 0.91	91.52 ± 0.47	94.51 ± 0.35
MARKOVDIFFUSION	$\mathbf{84.3 \pm 2.91}$	90.2 ± 0.98	84.0 ± 1.03	91.61 ± 0.49	94.22 ± 0.33
NODEDROPPING	84.16 ± 1.69	86.46 ± 1.51	83.71 ± 1.22	91.6 ± 0.58	94.54 ± 0.29
PPRDIFFUSION	84.05 ± 2.72	$\mathbf{90.84 \pm 1.67}$	82.7 ± 0.85	90.9 ± 1.06	94.03 ± 0.5
RANDOMWALKSUBGRAPH	83.53 ± 2.46	88.31 ± 1.01	83.36 ± 0.94	91.70 ± 0.49	94.6 ± 0.49
RLAP	83.68 ± 2.04	87.14 ± 1.34	$\mathbf{84.21 \pm 0.46}$	$\mathbf{91.73 \pm 0.53}$	$\mathbf{94.81 \pm 0.31}$

In this setting, $rLap$ achieves the best performance across all datasets, followed by EdgeDropping variants and NodeDropping. It is interesting to observe that PPRDiffusion and MarkovDiffusion approaches tend to perform relatively poorly in this setting. This is due to the fixed structure of the augmented views during training that a shared encoder is attempting to contrast. Intuitively, the lack of stochasticity fails to present noisy views to the model and improve its contrastive ability. On the other hand, for every node that is eliminated by $rLap$, it incorporates the lost information in the remaining sub-graph and preserves the random walk-based combinatorial properties in expectation. This approach of leveraging global information in augmented views seems to benefit the GNN encoders. Additionally, since randomized Schur complements are based on node and edge perturbations, one can intuitively consider it as an effective combination of the two.

On the other hand, MVGRL design corresponds to dedicated encoders, 'g-l' contrastive mode, and JSD objective. In this setting, diffusion-based methods performed relatively better on datasets such as CORA and AMAZON-PHOTO when compared to $rLap$ and NodeDropping. During our experiments on PUBMED, COAUTHOR-CS, and COAUTHOR-PHY, the PPRDiffusion technique led to out-of-memory (OOM) issues on a 32GB GPU, which we addressed using sub-graph sampling approaches. One can observe that our technique is outperformed by diffusion and adaptive augmentors such as EdgeDroppingEVC on CORA and AMAZON-PHOTO and the margin of improvement with $rLap$ on PUBMED, COAUTHOR-CS and COAUTHOR-PHY is minimal. However, we emphasize the observation that computational overheads of PPRDiffusion, MarkovDiffusion and EdgeDroppingEVC techniques are significantly large when compared to $rLap$ (see Table 18). In this setting, $rLap$ achieves the best trade-off between performance and computational overheads. Furthermore, based on Theorem 3.2, one can combine $rLap$ with PPRDiffusion to reduce the computational overheads (see Appendix F). An interesting observation related to MVGRL design and non-diffusion based augmentors is that a perturbation ratio γ_1 close to 0 for one of the views was preferred in most of the experiments. Intuitively, one can think of this setting as contrasting a highly perturbed graph with a relatively less perturbed one, which reflects the essence of multi-view contrasting.

4.2. Graph Classification Results

For graph classification tasks, we report the performance using GraphCL and BGRL designs in Table 4, 5 respectively. GraphCL design corresponds to shared encoders, 'g-g' contrastive mode and InfoNCE objective. Similar to our observations with shared encoders and InfoNCE loss in the node classification setting, $rLap$ outperformed other aug-

Table 4. Evaluation (in accuracy) on benchmark graph datasets with **GraphCL** based design.

AUGMENTOR	PROTEINS	IMDB-BINARY	MUTAG	IMDB-MULTI	NCI1
EDGEADDITION	71.34 ± 3.28	68.7 ± 3.55	81.0 ± 8.89	47.13 ± 4.33	73.36 ± 1.81
EDGEDROPPING	71.79 ± 3.37	70.4 ± 5.37	83.0 ± 5.15	46.4 ± 4.81	71.29 ± 3.08
EDGEDROPPINGDEGREE	71.96 ± 3.93	67.4 ± 4.76	85.0 ± 11.62	48.03 ± 5.42	74.4 ± 1.65
EDGEDROPPINGEVC	74.2 ± 2.97	71.3 ± 4.71	80.5 ± 10.11	45.94 ± 5.86	72.65 ± 2.32
EDGEDROPPINGPR	74.11 ± 4.24	$\mathbf{71.4 \pm 2.87}$	83.5 ± 8.38	46.8 ± 2.81	74.6 ± 2.29
MARKOVDIFFUSION	72.68 ± 4.43	64.9 ± 5.54	80.0 ± 6.71	41.47 ± 3.69	71.83 ± 2.5
NODEDROPPING	73.57 ± 1.84	69.0 ± 4.27	82.0 ± 6.03	47.33 ± 4.86	73.65 ± 2.61
PPRDIFFUSION	73.48 ± 5.15	69.8 ± 2.96	85.0 ± 6.71	42.87 ± 3.63	73.36 ± 2.34
RANDOMWALKSUBGRAPH	74.55 ± 4.59	70.0 ± 3.35	84.0 ± 9.17	43.93 ± 3.55	74.01 ± 2.47
RLAP	$\mathbf{75.27 \pm 3.34}$	70.9 ± 5.01	$\mathbf{87.5 \pm 9.86}$	$\mathbf{48.66 \pm 2.68}$	$\mathbf{75.06 \pm 1.65}$

Table 5. Evaluation (in accuracy) on benchmark graph datasets with **BGRL** based design.

AUGMENTOR	PROTEINS	IMDB-BINARY	MUTAG	IMDB-MULTI	NCI1
EDGEADDITION	81.57 ± 6.12	76.1 ± 5.63	73.5 ± 11.63	59.0 ± 9.58	72.77 ± 3.01
EDGEDROPPING	80.5 ± 7.94	78.3 ± 10.83	72.5 ± 10.55	56.13 ± 8.31	74.89 ± 2.36
EDGEDROPPINGDEGREE	81.5 ± 6.51	$\mathbf{78.5 \pm 6.32}$	74.0 ± 5.39	59.07 ± 5.0	73.67 ± 3.98
EDGEDROPPINGEVC	82.8 ± 3.94	75.8 ± 6.26	79.5 ± 9.63	55.0 ± 7.01	74.5 ± 1.86
EDGEDROPPINGPR	80.86 ± 8.49	76.8 ± 7.48	74.5 ± 8.52	57.8 ± 9.96	72.7 ± 2.69
MARKOVDIFFUSION	81.84 ± 7.48	75.3 ± 8.36	79.6 ± 8.79	56.8 ± 7.71	68.69 ± 2.99
NODEDROPPING	80.59 ± 6.12	70.3 ± 7.29	72.0 ± 9.81	57.53 ± 8.5	71.78 ± 2.4
PPRDIFFUSION	79.32 ± 7.69	73.5 ± 8.22	76.5 ± 10.26	57.13 ± 7.72	70.49 ± 2.23
RANDOMWALKSUBGRAPH	83.09 ± 3.85	69.5 ± 10.31	73.5 ± 10.51	57.8 ± 9.47	74.92 ± 2.88
RLAP	$\mathbf{84.34 \pm 4.03}$	74.8 ± 10.17	$\mathbf{81.5 \pm 5.39}$	$\mathbf{59.47 \pm 7.42}$	$\mathbf{75.15 \pm 3.57}$

mentors on PROTEINS, MUTAG, IMDB-MULTI and NCI1 datasets and is quite close to the EdgeDroppingPR technique on IMDB-BINARY dataset. EdgedroppingPR was the best adaptive edge augmentor on PROTEINS, IMDB-BINARY and NCI1 whereas EdgeDroppingDegree performed well on MUTAG and IMDB-MULTI. One should note that leveraging PageRank information in EdgeDroppingPR to perform stochastic augmentations consistently led to better performance than the PPRDiffusion approach, which underscores the benefits of stochasticity. Interestingly, we found that this particular design preferred augmentation ratios (γ_1, γ_2) to be nearly equal (see Table 14), indicating that in 'g-g' mode, the shared encoders prefer to contrast similar sized graphs.

In the next setting, we use a modified BGRL design which corresponds to shared encoders, 'g-l' contrastive mode and BL objective. Unlike previously mentioned designs, BGRL employs an online and target encoder with weight-sharing based on a moving average technique. The original BGRL design (Thakoor et al., 2021) uses an 'l-l' contrastive mode without negative samples and focuses on node classification settings. In our work, we leverage the modularity of the PyGCL library and incorporate the 'g-l' contrastive mode for graph classification. With this design, we report some of the highest observed performances on the PROTEINS dataset in unsupervised settings, with $rLap$ achieving the highest classification accuracy of $\approx 84.34\%$. Additionally, it achieves the best results on MUTAG, IMDB-MULTI and NCI1 but continues to suffer on IMDB-BINARY. However, in our ablation studies (see Appendix C), we observed that by changing the node elimination scheme from being a 'random' selection to a minimum 'degree' selection, this $rLap$ variant achieves $\approx 79.5\%$ accuracy on IMDB-BINARY and surpasses other techniques.

5. Future Research

• **Towards distributed augmentation:** Current approaches to augment large-scale graphs tend to rely on simpler techniques such as edge dropping (Thakoor et al., 2021). Thus, by extending $rLap$ augmentation to a distributed setting and leveraging its connection with graph diffusion, future efforts can explore and unlock the potential benefits of a richer class of augmentations for large-scale GCL. However, as a current challenge to achieving this goal, we observed that the resulting views of $rLap$ + PPRDiffusion + $sampling$ can be denser than the PPRDiffusion + $sampling$ approach. Thus, $rLap$ can address the bottlenecks of PPRDiffusion by saving memory during augmentation, but the GNN encoders would eventually consume additional memory due to the message-passing operations on relatively more edges. Although this overhead might not be significant for small-medium scale graphs, we believe that the reader should

be aware of these practical trade-offs when dealing with large-scale graphs.

• **Towards randomized second-order optimizers:** Computing the inverse of the Hessian for large neural networks poses significant computational overheads to optimizers that aim to leverage the curvature of the loss landscape. A popular technique to address this issue is to compute layer-wise block diagonal approximations of the hessian (Martens & Grosse, 2015; Osawa et al., 2019; Hoefler et al., 2021), which is relatively easy to invert. Alternatively, to capture the second-order information for a subset of parameters (for example, pertaining to a single layer), one can leverage the matrix inversion lemma and compute randomized Schur complements with respect to these parameters to negate the need to invert large Hessian matrices. Especially, by extending $rLap$, this approach can be computationally efficient while also incorporating unbiased curvature information into the parameter updates. This strategy can be of significant interest when only a subset of neural network layers need to be fine-tuned for downstream tasks.

• **Towards fine-grained augmentor benchmarks:** Our paper presents an extensive augmentor benchmark with respect to various GCL design choices. Thus, future efforts can leverage and extend these fine-grained comparisons for reproducible and fair comparison of augmentors.

6. Conclusion

In this work, we designed a fast, stochastic augmentor based on randomized Schur complements and demonstrated its flexibility and effectiveness on a variety of GCL designs and benchmark datasets. Our theoretical analysis and extensive experiments have proved that $rLap$ achieves a perfect balance with respect to performance and computational overheads while achieving state-of-the-art accuracies on unsupervised node and graph classification tasks. Especially, we achieved an unsupervised graph classification accuracy of $\approx 84.34\%$ on the PROTEINS dataset under linear evaluation and exceeded current state-of-the-art results. To conclude, we emphasize the potential of this new class of augmentation techniques that enable GCL frameworks to leverage the stochasticity and combinatorial properties of randomized Schur complements.

Acknowledgements

VK would like to thank Prof. Jonathan Weare, Zhengdao Chen, and the anonymous reviewers for their constructive feedback and prompt discussions during the preparation of this manuscript.

References

Adhikari, B., Zhang, Y., Ramakrishnan, N., and Prakash, B. A. Sub2vec: Feature learning for subgraphs. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pp. 170–182. Springer, 2018.

Bardes, A., Ponce, J., and LeCun, Y. Vicreg: Variance-invariance-covariance regularization for self-supervised learning. *arXiv preprint arXiv:2105.04906*, 2021.

Borgwardt, K. M., Ong, C. S., Schönauer, S., Vishwanathan, S., Smola, A. J., and Kriegel, H.-P. Protein function prediction via graph kernels. *Bioinformatics*, 21(suppl_1): i47–i56, 2005.

Boyd, S., Boyd, S. P., and Vandenberghe, L. *Convex optimization*. Cambridge university press, 2004.

Bronstein, M. M., Bruna, J., LeCun, Y., Szlam, A., and Vandergheynst, P. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4): 18–42, 2017.

Bruna, J., Zaremba, W., Szlam, A., and Lecun, Y. Spectral networks and locally connected networks on graphs. In *International Conference on Learning Representations (ICLR2014), CBLS, April 2014*, 2014.

Chen, C., Liang, T., and Biros, G. Rchol: Randomized cholesky factorization for solving sdd linear systems. *SIAM Journal on Scientific Computing*, 43(6): C411–C438, 2021.

Chen, T., Kornblith, S., Norouzi, M., and Hinton, G. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pp. 1597–1607. PMLR, 2020.

Chow, Y. S. and Teicher, H. *Probability theory: independence, interchangeability, martingales*. Springer Science & Business Media, 2003.

Chuang, C.-Y., Robinson, J., Lin, Y.-C., Torralba, A., and Jegelka, S. Debiased contrastive learning. *Advances in neural information processing systems*, 33:8765–8775, 2020.

Chung, F. The heat kernel as the pagerank of a graph. *Proceedings of the National Academy of Sciences*, 104 (50):19735–19740, 2007.

Cohen, M. B., Kyng, R., Miller, G. L., Pachocki, J. W., Peng, R., Rao, A. B., and Xu, S. C. Solving sdd linear systems in nearly m log1/2 n time. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*, pp. 343–352, 2014.

Cohen, M. B., Madry, A., Tsipras, D., and Vladu, A. Matrix scaling and balancing via box constrained newton's method and interior point methods. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pp. 902–913. IEEE, 2017.

Cohen, M. B., Kelner, J., Kyng, R., Peebles, J., Peng, R., Rao, A. B., and Sidford, A. Solving directed laplacian systems in nearly-linear time through sparse lu factorizations. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pp. 898–909. IEEE, 2018.

Cohen, M. B., Lee, Y. T., and Song, Z. Solving linear programs in the current matrix multiplication time. *Journal of the ACM (JACM)*, 68(1):1–39, 2021.

Cottle, R. W. Manifestations of the schur complement. *Linear algebra and its Applications*, 8(3):189–211, 1974.

Debnath, A. K., Lopez de Compadre, R. L., Debnath, G., Shusterman, A. J., and Hansch, C. Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity. *Journal of medicinal chemistry*, 34 (2):786–797, 1991.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Durfee, D., Kyng, R., Peebles, J., Rao, A. B., and Sachdeva, S. Sampling random spanning trees faster than matrix multiplication. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pp. 730–742, 2017.

Durfee, D., Gao, Y., Goranci, G., and Peng, R. Fully dynamic spectral vertex sparsifiers and applications. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pp. 914–925, 2019.

Fahrbach, M., Goranci, G., Peng, R., Sachdeva, S., and Wang, C. Faster graph embeddings via coarsening. In *International Conference on Machine Learning*, pp. 2953–2963. PMLR, 2020.

Fey, M. and Lenssen, J. E. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

Freedman, D. A. On tail probabilities for martingales. *the Annals of Probability*, pp. 100–118, 1975.

Gallier, J. H. Notes on the schur complement. 2010.

Gao, Y., Liu, Y. P., and Peng, R. Fully dynamic electrical flows: Sparse maxflow faster than goldberg-rao. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pp. 516–527. IEEE, 2022.

Gidaris, S., Singh, P., and Komodakis, N. Unsupervised representation learning by predicting image rotations. *arXiv preprint arXiv:1803.07728*, 2018.

Gori, M., Monfardini, G., and Scarselli, F. A new model for learning in graph domains. In *Proceedings. 2005 IEEE international joint conference on neural networks*, volume 2, pp. 729–734, 2005.

Grill, J.-B., Strub, F., Altché, F., Tallec, C., Richemond, P., Buchatskaya, E., Doersch, C., Avila Pires, B., Guo, Z., Gheshlaghi Azar, M., et al. Bootstrap your own latent-a new approach to self-supervised learning. *Advances in neural information processing systems*, 33:21271–21284, 2020.

Grover, A. and Leskovec, J. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 855–864, 2016.

Hassani, K. and Khasahmadi, A. H. Contrastive multi-view representation learning on graphs. In *International Conference on Machine Learning*, pp. 4116–4126. PMLR, 2020.

Henaff, M., Bruna, J., and LeCun, Y. Deep convolutional networks on graph-structured data. *arXiv preprint arXiv:1506.05163*, 2015.

Hjelm, R. D., Fedorov, A., Lavoie-Marchildon, S., Grewal, K., Bachman, P., Trischler, A., and Bengio, Y. Learning deep representations by mutual information estimation and maximization. *arXiv preprint arXiv:1808.06670*, 2018.

Hoefler, T., Alistarh, D., Ben-Nun, T., Dryden, N., and Peste, A. Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. *The Journal of Machine Learning Research*, 22(1):10882–11005, 2021.

Hu, W., Liu, B., Gomes, J., Zitnik, M., Liang, P., Pande, V., and Leskovec, J. Strategies for pre-training graph neural networks. *arXiv preprint arXiv:1905.12265*, 2019.

Hübler, C., Kriegel, H.-P., Borgwardt, K., and Ghahramani, Z. Metropolis algorithms for representative subgraph sampling. In *2008 Eighth IEEE International Conference on Data Mining*, pp. 283–292. IEEE, 2008.

Jiao, Y., Xiong, Y., Zhang, J., Zhang, Y., Zhang, T., and Zhu, Y. Sub-graph contrast for scalable self-supervised graph representation learning. In *2020 IEEE international conference on data mining (ICDM)*, pp. 222–231. IEEE, 2020.

Jin, W., Derr, T., Liu, H., Wang, Y., Wang, S., Liu, Z., and Tang, J. Self-supervised learning on graphs: Deep insights and new direction. *arXiv preprint arXiv:2006.10141*, 2020.

Jing, L. and Tian, Y. Self-supervised visual feature learning with deep neural networks: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 43(11): 4037–4058, 2020.

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016a.

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

Klicpera, J., Weißenberger, S., and Günnemann, S. Diffusion improves graph learning. *arXiv preprint arXiv:1911.05485*, 2019.

Kondor, R. and Pan, H. The multiscale laplacian graph kernel. *Advances in neural information processing systems*, 29, 2016.

Kondor, R. I. and Lafferty, J. Diffusion kernels on graphs and other discrete structures. In *Proceedings of the 19th international conference on machine learning*, volume 2002, pp. 315–322, 2002.

Koutis, I., Miller, G. L., and Peng, R. A nearly-m log n time solver for sdd linear systems. In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, pp. 590–598. IEEE, 2011.

Kyng, R. and Sachdeva, S. Approximate gaussian elimination for laplacians-fast, sparse, and simple. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pp. 573–582. IEEE, 2016.

Kyng, R., Lee, Y. T., Peng, R., Sachdeva, S., and Spielman, D. A. Sparsified cholesky and multigrid solvers for connection laplacians. In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, pp. 842–850, 2016.

Lan, Z., Chen, M., Goodman, S., Gimpel, K., Sharma, P., and Soricut, R. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*, 2019.

Lin, J. Divergence measures based on the shannon entropy. *IEEE Transactions on Information theory*, 37(1):145–151, 1991.

Liu, Y., Jin, M., Pan, S., Zhou, C., Zheng, Y., Xia, F., and Yu, P. Graph self-supervised learning: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 2022.

Martens, J. and Grosse, R. Optimizing neural networks with kronecker-factored approximate curvature. In *International conference on machine learning*, pp. 2408–2417. PMLR, 2015.

McCallum, A. K., Nigam, K., Rennie, J., and Seymore, K. Automating the construction of internet portals with machine learning. *Information Retrieval*, 3(2):127–163, 2000.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

Narayanan, A., Chandramohan, M., Venkatesan, R., Chen, L., Liu, Y., and Jaiswal, S. graph2vec: Learning distributed representations of graphs. *arXiv preprint arXiv:1707.05005*, 2017.

Niepert, M., Ahmed, M., and Kutzkov, K. Learning convolutional neural networks for graphs. In *International conference on machine learning*, pp. 2014–2023. PMLR, 2016.

Oord, A. v. d., Li, Y., and Vinyals, O. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

Osawa, K., Tsuji, Y., Ueno, Y., Naruse, A., Yokota, R., and Matsuoka, S. Large-scale distributed second-order optimization using kronecker-factored approximate curvature for deep convolutional neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 12359–12367, 2019.

Ouellette, D. V. Schur complements and statistics. *Linear Algebra and its Applications*, 36:187–295, 1981.

Page, L., Brin, S., Motwani, R., and Winograd, T. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

Peng, R. and Vempala, S. Solving sparse linear systems faster than matrix multiplication. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 504–521. SIAM, 2021.

Perozzi, B., Al-Rfou, R., and Skiena, S. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 701–710, 2014.

Qiu, J., Dong, Y., Ma, H., Li, J., Wang, K., and Tang, J. Network embedding as matrix factorization: Unifying deepwalk, line, pte, and node2vec. In *Proceedings of the eleventh ACM international conference on web search and data mining*, pp. 459–467, 2018.

Qiu, J., Chen, Q., Dong, Y., Zhang, J., Yang, H., Ding, M., Wang, K., and Tang, J. Gcc: Graph contrastive coding for graph neural network pre-training. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1150–1160, 2020.

Radford, A., Narasimhan, K., Salimans, T., Sutskever, I., et al. Improving language understanding by generative pre-training. 2018.

Robinson, J., Chuang, C.-Y., Sra, S., and Jegelka, S. Contrastive learning with hard negative samples. *arXiv preprint arXiv:2010.04592*, 2020.

Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., and Monfardini, G. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008.

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

Shchur, O., Mumme, M., Bojchevski, A., and Günnemann, S. Pitfalls of graph neural network evaluation. *arXiv preprint arXiv:1811.05868*, 2018.

Shervashidze, N., Vishwanathan, S., Petri, T., Mehlhorn, K., and Borgwardt, K. Efficient graphlet kernels for large graph comparison. In *Artificial intelligence and statistics*, pp. 488–495. PMLR, 2009.

Shervashidze, N., Schweitzer, P., Van Leeuwen, E. J., Mehlhorn, K., and Borgwardt, K. M. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(9), 2011.

Spielman, D. A. and Srivastava, N. Graph sparsification by effective resistances. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pp. 563–568, 2008.

Spielman, D. A. and Teng, S.-H. Solving sparse, symmetric, diagonally-dominant linear systems in time o (m/sup 1.31. In *44th Annual IEEE Symposium on Foundations of Computer Science, 2003. Proceedings.*, pp. 416–427. IEEE, 2003.

Spielman, D. A. and Teng, S.-H. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pp. 81–90, 2004.

Sun, F.-Y., Hoffmann, J., Verma, V., and Tang, J. Infograph: Unsupervised and semi-supervised graph-level representation learning via mutual information maximization. *arXiv preprint arXiv:1908.01000*, 2019.

Tang, J., Qu, M., and Mei, Q. Pte: Predictive text embedding through large-scale heterogeneous text networks. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 1165–1174, 2015a.

Tang, J., Qu, M., Wang, M., Zhang, M., Yan, J., and Mei, Q. Line: Large-scale information network embedding. In *Proceedings of the 24th international conference on world wide web*, pp. 1067–1077, 2015b.

Thakoor, S., Tallec, C., Azar, M. G., Munos, R., Veličković, P., and Valko, M. Bootstrapped representation learning on graphs. In *ICLR 2021 Workshop on Geometrical and Topological Representation Learning*, 2021.

Tropp, J. A. User-friendly tail bounds for sums of random matrices. *Foundations of computational mathematics*, 12 (4):389–434, 2012.

Tropp, J. A. Matrix concentration & computational linear algebra. 2019.

Tropp, J. A. et al. An introduction to matrix concentration inequalities. *Foundations and Trends® in Machine Learning*, 8(1-2):1–230, 2015.

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

Velickovic, P., Fedus, W., Hamilton, W. L., Liò, P., Bengio, Y., and Hjelm, R. D. Deep graph infomax. *ICLR (Poster)*, 2(3):4, 2019.

Wale, N., Watson, I. A., and Karypis, G. Comparison of descriptor spaces for chemical compound retrieval and classification. *Knowledge and Information Systems*, 14 (3):347–375, 2008.

Welling, M. and Kipf, T. N. Semi-supervised classification with graph convolutional networks. In *J. International Conference on Learning Representations (ICLR 2017)*, 2016.

Williams, D. *Probability with martingales*. Cambridge university press, 1991.

Wu, L., Lin, H., Tan, C., Gao, Z., and Li, S. Z. Self-supervised learning on graphs: Contrastive, generative, or predictive. *IEEE Transactions on Knowledge and Data Engineering*, 2021.

Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., and Philip, S. Y. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 32(1):4–24, 2020.

Xie, Y., Xu, Z., Zhang, J., Wang, Z., and Ji, S. Self-supervised learning of graph neural networks: A unified review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.

Xu, D., Cheng, W., Luo, D., Chen, H., and Zhang, X. Infogcl: Information-aware graph contrastive learning. *Advances in Neural Information Processing Systems*, 34: 30414–30425, 2021.

Xu, K., Hu, W., Leskovec, J., and Jegelka, S. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

Yanardag, P. and Vishwanathan, S. Deep graph kernels. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 1365–1374, 2015.

Yin, Y., Wang, Q., Huang, S., Xiong, H., and Zhang, X. Autogcl: Automated graph contrastive learning via learnable view generators. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pp. 8892–8900, 2022.

You, Y., Chen, T., Sui, Y., Chen, T., Wang, Z., and Shen, Y. Graph contrastive learning with augmentations. *Advances in Neural Information Processing Systems*, 33: 5812–5823, 2020.

Zbontar, J., Jing, L., Misra, I., LeCun, Y., and Deny, S. Barlow twins: Self-supervised learning via redundancy reduction. In *International Conference on Machine Learning*, pp. 12310–12320. PMLR, 2021.

Zeng, J. and Xie, P. Contrastive self-supervised learning for graph classification. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 10824–10832, 2021.

Zhang, F. *The Schur complement and its applications*, volume 4. Springer Science & Business Media, 2006.

Zhou, J., Cui, G., Hu, S., Zhang, Z., Yang, C., Liu, Z., Wang, L., Li, C., and Sun, M. Graph neural networks: A review of methods and applications. *AI Open*, 1:57–81, 2020.

Zhu, H. and Koniusz, P. Simple spectral graph convolution. In *International Conference on Learning Representations*, 2020.

Zhu, Q., Yang, C., Xu, Y., Wang, H., Zhang, C., and Han, J. Transfer learning of graph neural networks with ego-graph information maximization. *Advances in Neural Information Processing Systems*, 34:1766–1779, 2021a.

Zhu, Y., Xu, Y., Yu, F., Liu, Q., Wu, S., and Wang, L. Deep graph contrastive representation learning. *arXiv preprint arXiv:2006.04131*, 2020.

Zhu, Y., Xu, Y., Liu, Q., and Wu, S. An empirical study of graph contrastive learning. *arXiv preprint arXiv:2109.01116*, 2021b.

Zhu, Y., Xu, Y., Yu, F., Liu, Q., Wu, S., and Wang, L. Graph contrastive learning with adaptive augmentation. In *Proceedings of the Web Conference 2021*, pp. 2069–2080, 2021c.

A. Related Work

Earlier works on representation learning techniques on graphs with limited labels leveraged exploration-based approaches such as DeepWalk (Perozzi et al., 2014) and node2vec (Grover & Leskovec, 2016). Inspired by the skip-gram model of word2vec (Mikolov et al., 2013), such techniques generate similar embeddings for nodes that co-occur in a random walk. On the other hand, techniques such as Large-scale Information Network Embedding (LINE) (Tang et al., 2015b) and Predictive Text Embedding (PTE) (Tang et al., 2015a) learn representations by contrasting nodes with negative samples drawn from a noisy distribution. Eventually, in the work of Qiu et al. (2018), DeepWalk, node2vec, LINE and PTE were unified under a common matrix factorization scheme for computing the network/graph embeddings. However, their limitations on scalability and incorporating global information in the learning process persisted.

Recent efforts in GCL leverage GNNs as scalable feature encoders and are inspired by techniques/frameworks in vision (Gidaris et al., 2018; Hjelm et al., 2018; Chen et al., 2020; Jing & Tian, 2020) and language domains (Mikolov et al., 2013; Devlin et al., 2018; Radford et al., 2018; Lan et al., 2019). Some of the notable efforts by Velickovic et al. (2019); Sun et al. (2019); Zhu et al. (2020); You et al. (2020); Hassani & Khasahmadi (2020) employ these GCL frameworks and aim to minimize objectives based on mutual information. With this growing line of work, a critical aspect of most GCL frameworks that is devised based on trial and error is the augmentation phase. The node/edge perturbation techniques introduced in You et al. (2020) have been widely employed due to their simplicity and are empirically effective on unsupervised, self-supervised node and graph classification tasks (Zhu et al., 2021b; Thakoor et al., 2021). To the contrary, Hassani & Khasahmadi (2020) showed that augmentations such as graph diffusions can outperform the stochastic techniques with dedicated encoders. Since then, efforts have been made to devise augmentations that can leverage structural information during the training process either in a pre-defined (Zhu et al., 2021c) or learnable fashion (Yin et al., 2022). The adaptive augmentation technique by Zhu et al. (2021c) takes a step in this direction and focuses on edge-dropping mechanisms based on centrality measures. We observed in our experiments that, it is difficult to choose a particular centrality measure that can work for both node and graph-based tasks. Furthermore, measures such as eigenvector centrality are computationally expensive, which makes such adaptive techniques orders of magnitude slower than simple edge dropping.

To address these issues, we designed the $rLap$ algorithm, which is inspired by the approximate gaussian elimination (AGE) technique introduced by Kyng & Sachdeva (2016). The AGE technique leverages the idea of effective resistances (Spielman & Srivastava, 2008) and falls in the line of work on fast approximate linear system solvers for SDDM matrices (Spielman & Teng, 2003; 2004; Koutis et al., 2011; Cohen et al., 2014; Kyng et al., 2016). It has been influential in efficiently solving linear systems of equations (Cohen et al., 2018; Peng & Vempala, 2021; Cohen et al., 2021; Chen et al., 2021), sampling random spanning trees (Durfee et al., 2017) and matrix scaling (Cohen et al., 2017). The technique of Kyng & Sachdeva (2016) was later adopted by Fahrbach et al. (2020) for generating node embeddings using sparse matrix factorization techniques. However, to the best of our knowledge, techniques along this line of work have not been explored for GCL.

B. Proof for Theorem 3.1

Recall from the preliminaries that:

$$\textit{CLIQUE}(\mathbf{L}, v_i) = \frac{1}{2w(v_i)} \sum_{e_{ij} \in \mathcal{E}} \sum_{e_{ik} \in \mathcal{E}} w(e_{ij}) w(e_{ik}) \mathbf{\Delta}_{jk}$$

The factor 2 in the denominator addresses duplicate laplacians in the quadratic sum. We assume that $\mathcal{G}$ doesn't have self-loops and eliminate these redundant summations to rewrite $\textit{CLIQUE}(\mathbf{L}, v_i)$ as:

$$\textit{CLIQUE}(\mathbf{L}, v_i) = \sum_{x_l \in \mathcal{N}(v_i)} \sum_{x_q \in \mathcal{N}(v_i) \setminus \{x_1, \cdots, x_l\}} \frac{w(v_i, x_q) \cdot w(v_i, x_l)}{w(v_i)} \cdot \mathbf{\Delta}_{x_l x_q} \tag{8}$$

To employ this reformulation in Algorithm 1, we assume $\mathbf{R}_{i-1}$ as the Schur complement at the beginning of ith iteration of the outer loop and v_i as the selected node for elimination. Then:

$$\textit{CLIQUE}(\mathbf{R}_{i-1}, v_i) = \sum_{x_l \in \mathcal{N}_{\mathbf{R}_{i-1}}(v_i)} \sum_{x_q \in \mathcal{N}_{\mathbf{R}_{i-1}}(v_i) \setminus \{x_1, \cdots, x_l\}} \frac{w_{\mathbf{R}_{i-1}}(v_i, x_q) \cdot w_{\mathbf{R}_{i-1}}(v_i, x_l)}{w_{\mathbf{R}_{i-1}}(v_i)} \cdot \mathbf{\Delta}_{x_l x_q} \tag{9}$$

In Eq. 9, we eliminated half the effort needed to compute *CLIQUE*$(\mathbf{R}_{i-1}, v_i)$, but introduced an ordering of the neighbors $\{x_1, \cdots, x_l\} \in \mathcal{N}_{\mathbf{R}_{i-1}}(v_i)$. This ordering is determined by o_n in Algorithm 1 and leads to different conditional probabilities $P(x_q|x_l)$ while computing the approximate clique $\mathbf{C}_i = C(\mathbf{R}_{i-1}, v_i)$. The ordered nodes are given by $\mathcal{X}(v_i) = o_n(\mathcal{N}_{\mathbf{R}_{i-1}}(v_i))$ and $w_{\mathbf{R}_{i-1}} = \hat{w}$ for notational convenience. Then, $\mathbb{E}C(\mathbf{R}_{i-1}, v_i)$ is computed as follows:

$$\mathbb{E}C(\mathbf{R}_{i-1}, v_i) = \sum_{x_q \in \mathcal{X}(v_i)\setminus\{x_1\}} P(x_q|x_1) \cdot \hat{w}(x_1, x_q) \cdot \mathbf{\Delta}_{x_1 x_q} + \sum_{x_q \in \mathcal{X}(v_i)\setminus\{x_1, x_2\}} P(x_q|x_2) \cdot \hat{w}(x_2, x_q) \cdot \mathbf{\Delta}_{x_2 x_q} + \cdots$$
$$+ \sum_{x_q \in \mathcal{X}(v_i)\setminus\{x_1, \ldots, x_{|\mathcal{X}(v_i)|-1}\}} P(x_q|x_{|\mathcal{X}(v_i)|-1}) \cdot \hat{w}(x_{|\mathcal{X}(v_i)|-1}, x_q) \cdot \mathbf{\Delta}_{x_{|\mathcal{X}(v_i)|-1} x_q}$$

By substituting the conditional probabilities and edge weights as per Algorithm 1, we get:

$$\mathbb{E}C(\mathbf{R}_{i-1}, v_i) = \sum_{x_q \in \mathcal{X}(v_i)\setminus\{x_1\}} \frac{\hat{w}(v_i, x_q)}{\hat{w}(v_i) - \hat{w}(v_i, x_1)} \cdot \frac{\hat{w}(v_i, x_1) \cdot (\hat{w}(v_i) - \hat{w}(v_i, x_1))}{\hat{w}(v_i)} \cdot \mathbf{\Delta}_{x_1 x_q}$$
$$+ \sum_{x_q \in \mathcal{X}(v_i)\setminus\{x_1, x_2\}} \frac{\hat{w}(v_i, x_q)}{\hat{w}(v_i) - \hat{w}(v_i, x_1) - \hat{w}(v_i, x_2)} \cdot \frac{\hat{w}(v_i, x_2) \cdot (\hat{w}(v_i) - \hat{w}(v_i, x_1) - \hat{w}(v_i, x_2))}{\hat{w}(v_i)} \cdot \mathbf{\Delta}_{x_2 x_q}$$
$$+ \ldots$$
$$+ \sum_{x_q \in \mathcal{X}(v_i)\setminus\{x_1, \ldots, x_{|\mathcal{X}(v_i)-1|}\}} \frac{\hat{w}(v_i, x_q)}{\hat{w}(v_i) - \sum_{k=1}^{|\mathcal{X}(v_i)|-1} \hat{w}(v_i, x_k)} \cdot \frac{\hat{w}(v_i, x_{|\mathcal{X}(v_i)|-1}) \cdot (\hat{w}(v_i) - \sum_{k=1}^{|\mathcal{X}(v_i)|-1} \hat{w}(v_i, x_k))}{\hat{w}(v_i)} \cdot \mathbf{\Delta}_{x_{|\mathcal{X}(v_i)|-1} x_q}$$

The factors cancel out and simplify the expectation to:

$$\mathbb{E}C(\mathbf{R}_{i-1}, v_i) = \sum_{x_q \in \mathcal{X}(v_i)\setminus\{x_1\}} \frac{\hat{w}(v_i, x_q) \cdot \hat{w}(v_i, x_1)}{w(v_i)} \cdot \mathbf{\Delta}_{x_1 x_q} + \sum_{x_q \in \mathcal{X}(v_i)\setminus\{x_1, x_2\}} \frac{\hat{w}(v_i, x_q) \cdot \hat{w}(v_i, x_2)}{\hat{w}(v_i)} \cdot \mathbf{\Delta}_{x_2 x_q} + \ldots$$
$$+ \sum_{x_q \in \mathcal{X}(v_i)\setminus\{x_1, \ldots, x_{|\mathcal{X}(v_i)-1|}\}} \frac{\hat{w}(v_i, x_q) \cdot \hat{w}(v_i, x_{|\mathcal{X}(v_i)|-1})}{\hat{w}(v_i)} \cdot \mathbf{\Delta}_{x_{|\mathcal{X}(v_i)|-1} x_q}$$
$$\implies \mathbb{E}\mathbf{C}_i = \mathbb{E}C(\mathbf{R}_{i-1}, v_i) = \sum_{x_l \in \mathcal{X}(v_i)} \sum_{x_q \in \mathcal{X}(u_i)\setminus\{x_1, \cdots, x_l\}} \frac{\hat{w}(v_i, x_q) \cdot \hat{w}(v_i, x_l)}{\hat{w}(v_i)} \cdot \mathbf{\Delta}_{x_l x_q} \tag{10}$$

Thus, after node v_i has been eliminated, $\mathbf{C}_i = C(\mathbf{R}_{i-1}, v_i)$ is an unbiased estimator of the actual clique. Now, by removing the star and adding $\mathbf{C}_i$ to $\mathbf{R}_{i-1}$, we get $\mathbf{R}_i$, which is the unbiased schur complement approximation after i iterations:

$$\mathbf{R}_i = \mathbf{R}_{i-1} - \textit{STAR}(\mathbf{R}_{i-1}, v_i) + \mathbf{C}_i \tag{11}$$

By repeating this process for $\gamma|\mathcal{V}|$ iterations, we get the desired randomized schur complement $\mathbf{R}_{\gamma|\mathcal{V}|}$.

B.1. Deviation Analysis

Considering $\mathbf{L}_0 = \mathbf{R}_0 = \mathbf{L}$, one can observe that $\mathbf{R}_i$ is a laplacian matrix, satisfying the following laplacian approximations:

$$\mathbf{L}_i = \mathbf{R}_i + \sum_{k=1}^{i} \mathbf{s}_k \mathbf{s}_k^*, \forall i = 1, 2, \ldots, N \tag{12}$$

This implies that at every step we remove a rank 1 matrix corresponding to $\mathbf{s}_k = \frac{1}{\sqrt{(\mathbf{R}_{k-1})_{v_k v_k}}}\mathbf{R}_{k-1}\delta_{v_k}$. Thus, the sequence $\{\mathbf{L}_i\}$ forms a random process such that:

$$\mathbb{E}_{i-1}[\mathbf{L}_i - \mathbf{L}_{i-1}] = \mathbb{E}_{i-1}\mathbb{E}_{i-1}[(\mathbf{R}_i - \mathbf{R}_{i-1} + \mathbf{s}_i\mathbf{s}_i^*)|v_i] = \mathbb{E}_{i-1}\mathbb{E}_{i-1}[(\mathbf{C}_i - \textit{CLIQUE}(\mathbf{R}_{i-1}, v_i))|v_i] = 0 \tag{13}$$

The last equality is based on the result that $\mathbb{E}_{i-1}[\mathbf{C}_i|v_i] = \textit{CLIQUE}(\mathbf{R}_{i-1}, v_i)$ as shown above. This implies, $\mathbb{E}\mathbf{L}_i = \mathbf{L}$ for $i = 1, 2, 3, \ldots, \gamma|\mathcal{V}|$ and the 'deviation' sequence $\{\mathbf{L}_i - \mathbf{L}_0\}$ is a ***matrix martingale*** with $\mathbf{0}$ initial value. The analysis for the 'deviation' martingale $\{\mathbf{L}_i - \mathbf{L}_0\}$ aids in understanding the randomness that is induced due to a series of Schur complement approximations.

Our line of analysis is inspired by the approach of Kyng & Sachdeva (2016) which was later presented in Tropp (2019). We relax the assumptions and transformations for spectral approximations and bounding the effective resistances. These assumptions were critical in the analysis of Kyng & Sachdeva (2016); Tropp (2019) as the motivation was to solve the laplacian system of linear equations. However, since we are primarily concerned with augmentations of graphs, we avoid such constraints.

Analysis Sketch: We primarily leverage the corrector process approach introduced in Tropp (2019). By building the corrector process for the martingale $\{\mathbf{L}_i - \mathbf{L}_0\}$, we provide a tail bound for the singular values of these deviations, which is a direct application of Theorem 7.4 in Tropp (2019). We start by building the individual corrector matrices and then extend them to build the corrector process.

B.1.1. Correctors

Definition B.1. Corrector process (Tropp, 2019)(Sec 7.2): Consider a function $g : [0, \infty] \to [0, \infty]$, a martingale $\{\mathbf{Y}_k \in \mathbb{R}^{N\times N} : k = 0, 1, \cdots\}$ and a predictable random process $\{\mathbf{W}_k \in \mathbb{R}^{N\times N} : k = 0, 1, \cdots\}$ of self-adjoint matrices. We define $\{g\mathbf{W}_k\}$ as a corrector process of martingale $\{\mathbf{Y}_k\}$ if the real-valued random process $\{\text{tr}(\exp(\theta\mathbf{Y}_k - g(\theta)\mathbf{W}_k))\}, \forall\theta \geq 0$ is a positive supermartingale (Williams, 1991; Chow & Teicher, 2003).

Definition B.2. Corrector (Tropp, 2019)(Sec 7.3): Consider a function $g : [0, \infty] \to [0, \infty]$, a random self-adjoint matrix $\mathbf{U} \in \mathbb{R}^{N\times N}$, a fixed matrix $\mathbf{V} \in \mathbb{R}^{N\times N}$. We define the corrector $g\mathbf{V}$ as the matrix satisfying:

$$\mathbb{E}\big[\,\text{tr}\big(\exp(\mathbf{M} + \theta\mathbf{U} - g(\theta)\mathbf{V})\big)\big] \leq \text{tr}\big(\exp(\mathbf{M})\big), \theta \geq 0, \forall\mathbf{M} \in \mathbb{H}_N \tag{14}$$

This bound must hold true for all fixed matrices $\mathbf{M} \in \mathbb{H}_N$, where $\mathbb{H}_N$ denotes a space of real $N \times N$ self-adjoint matrices.

Corollary B.3. *(Tropp, 2019)(Corollary 7.7): Let $\mathbf{M} \in \mathbb{R}^{N\times N}$ be a self-adjoint matrix and $\mathbf{U} \in \mathbb{R}^{N\times N}$ be a random self-adjoint matrix. Then for $\theta \in \mathbb{R}$,*

$$\mathbb{E}[\,tr(\exp(\mathbf{M} + \theta\mathbf{U} - \log\mathbb{E}e^{\theta\mathbf{U}}))] \leq \mathbb{E}[\,tr(\exp(\mathbf{M}))] \tag{15}$$

This corollary is specific to corrector matrices of the form $g\mathbf{V} = \log\mathbb{E}e^{\theta\mathbf{U}}$ and will be used to build the Bernstein and Chernoff correctors for random matrices with arbitrary spectral norm.

B.1.2. The Bernstein Corrector

From these definitions, let $\mathbf{U}_i = \mathbf{C}_i - \mathbb{E}\mathbf{C}_i$ be the random self-adjoint matrix. Let $\|\mathbf{U}_i\|$ indicate the spectral norm of $\mathbf{U}_i$ (l_2 operator norm), which indicates its largest singular value. Since $\mathbb{E}[\mathbf{U}_i] = \mathbf{0}$, we can directly apply a generalized version of the Bernstein corrector for $\mathbf{U}_i$. The result in Proposition 7.8 of Tropp (2019) assumes $\|\mathbf{U}_i\| \leq 1$ and requires some form of normalization in the analysis. We avoid such assumptions and extend the result to any arbitrary bound $\|\mathbf{U}_i\| \leq \mathcal{B}_{\mathbf{U}_i}$.

We begin with a scalar case and extend it to the matrix case. If $|x| \leq b$, then the exponential $e^{\theta x}$ can be bounded by:

$$e^{\theta x} = 1 + \theta x + \sum_{j=2}^{\infty}\frac{\theta^j}{j!}x^j \leq 1 + \theta x + \left(\sum_{j=2}^{\infty}\frac{b^{j-2}|\theta|^j}{2\cdot 3^{j-2}}\right)x^2 = 1 + \theta x + \frac{\theta^2}{2(1 - b|\theta|/3)}x^2$$

Similarly, if $\|\mathbf{U}_i\| \leq \mathcal{B}_{\mathbf{U}_i}$ then, we can extend the above bound to matrix form as follows:

$$e^{\theta\mathbf{U}_i} \preccurlyeq \mathbf{I} + \theta\mathbf{U}_i + \frac{\theta^2}{2(1 - \mathcal{B}_{\mathbf{U}_i}|\theta|/3)}\mathbf{U}_i^2 \implies \log\mathbb{E}e^{\theta\mathbf{U}_i} \preccurlyeq \frac{\theta^2}{2(1 - \mathcal{B}_{\mathbf{U}_i}|\theta|/3)}\mathbb{E}\mathbf{U}_i^2$$

Where $\preccurlyeq$ indicates inequality in the semi-definite order. For example: if $\mathbf{J} \preccurlyeq \mathbf{Q}$, then $\mathbf{x}^\top \mathbf{J}\mathbf{x} \leq \mathbf{x}^\top \mathbf{Q}\mathbf{x}, \forall \mathbf{x}$. Also, the second inequality is based on the operator monotone property of logarithms (see Chapter 8 in Tropp et al. (2015)). From Corollary B.3, observe that by replacing $\log \mathbb{E}e^{\theta \mathbf{U}_i}$ with $\frac{\theta^2}{2(1-\mathcal{B}_{\mathbf{U}_i}|\theta|/3)}\mathbb{E}\mathbf{U}_i^2$ the inequality is still preserved. Thus, we can use $\frac{\theta^2}{2(1-\mathcal{B}_{\mathbf{U}_i}|\theta|/3)}\mathbb{E}\mathbf{U}_i^2$ as our corrector matrix for $\mathbf{U}_i$. The variance term $\mathbb{E}\mathbf{U}_i^2$ can be given by:

$$\mathbb{E}\mathbf{U}_i^2 = \mathbb{E}[\mathbf{C}_i - \mathbb{E}[\mathbf{C}_i]]^2 = \mathbb{E}[\mathbf{C}_i^2] - (\mathbb{E}[\mathbf{C}_i])^2 \preccurlyeq \mathbb{E}[\mathbf{C}_i^2] \preccurlyeq \mathbb{E}[\|\mathbf{C}_i\| \cdot \mathbf{C}_i] \tag{16}$$

The second equality is due to the zero mean of $\mathbf{C}_i - \mathbb{E}[\mathbf{C}_i]$. The last semi-definite order inequality is straightforward since $\|\mathbf{C}_i\|$ indicates the largest singular value of $\mathbf{C}_i$, which bounds $\mathbf{x}^\top \mathbf{C}_i\mathbf{C}_i\mathbf{x} \leq \|\mathbf{C}_i\| \mathbf{x}^\top \mathbf{C}_i\mathbf{x}, \forall \mathbf{x}$. Additionally, if we assume the spectral norm of the approximated cliques to be bounded $\|\mathbf{C}_i\| \leq \mathcal{B}_{\mathbf{C}_i}$, we get $\mathbb{E}\mathbf{U}^2 \preccurlyeq \mathbb{E}[\|\mathbf{C}_i\| \cdot \mathbf{C}_i] \preccurlyeq \mathcal{B}_{\mathbf{C}_i} \cdot \textit{CLIQUE}(\mathbf{R}_{i-1}, v_i)$. Putting all these results together, we get the Bernstein corrector as follows:

$$\left(\frac{\theta^2}{2(1-\mathcal{B}_{\mathbf{U}_i}|\theta|/3)}\mathcal{B}_{\mathbf{C}_i}\right)\textit{CLIQUE}(\mathbf{R}_{i-1}, v_i) \tag{17}$$

The Bernstein corrector handles the randomness associated with the clique approximation after a node v_i is selected (i.e, due to the ordering scheme o_n and its implications on the edge formation probabilities). Now, to handle the randomness associated with this node selection operation, we build the Chernoff corrector.

B.1.3. The Chernoff Corrector

Based on the formulations of $\textit{CLIQUE}(\mathbf{R}_{i-1}, v_i), \textit{STAR}(\mathbf{R}_{i-1}, v_i)$, we obtain a relation between them as follows:

$$\textit{CLIQUE}(\mathbf{R}_{i-1}, v_i) = \textit{STAR}(\mathbf{R}_{i-1}, v_i) - \mathbf{s}_i\mathbf{s}_i^* \tag{18}$$

Where $\mathbf{s}_i = \frac{1}{\sqrt{(\mathbf{R}_{i-1})_{v_iv_i}}}\mathbf{R}_{i-1}\delta_{v_i}$ as introduced before. Since $\mathbf{s}_i\mathbf{s}_i^*$ is positive semi-definite and $\textit{STAR}(\mathbf{R}_{i-1}, v_i)$ is formed by a subset of edges of $\mathbf{R}_{i-1}$, we obtain the following semi-definite order inequalities:

$$\mathbf{0} \preccurlyeq \textit{CLIQUE}(\mathbf{R}_{i-1}, v_i) \preccurlyeq \textit{STAR}(\mathbf{R}_{i-1}, v_i) \preccurlyeq \mathbf{R}_{i-1} \tag{19}$$

This result can now be used to bound $\mathbb{E}_{v_i}[\textit{CLIQUE}(\mathbf{R}_{i-1}, v_i)]$ over the possible choices of $v_i \in \mathcal{V}_{\mathbf{R}_{i-1}}$. Here $\mathcal{V}_{\mathbf{R}_{i-1}}$ represents the set of nodes from which v_i is chosen uniformly at random for elimination. This setting corresponds to the case of the 'random' node elimination scheme o_v.

$$\mathbb{E}_{v_i}[\textit{CLIQUE}(\mathbf{R}_{i-1}, v_i)] \preccurlyeq \mathbb{E}_{v_i}[\textit{STAR}(\mathbf{R}_{i-1}, v_i)] = \frac{1}{|\mathcal{V}_{\mathbf{R}_{i-1}}|}\sum_{v_i \in \mathcal{V}_{\mathbf{R}_{i-1}}}\sum_{e \in \textit{STAR}(\mathbf{R}_{i-1}, v_i)} w_{\mathbf{R}_{i-1}}(e)\mathbf{\Delta}_e \tag{20}$$

Where $\mathbf{\Delta}_e$ is the elementary laplacian formed by the nodes at either end of edge e. The last term in the above equation indicates that, for every node $v_i \in \mathcal{V}_{\mathbf{R}_{i-1}}$, we add the weighted laplacians for all edges in $\textit{STAR}(\mathbf{R}_{i-1}, v_i)$. Thus, we are iterating over all the edges in $\mathcal{E}_{\mathbf{R}_{i-1}}$ and adding the weighted elementary laplacians twice. Formally:

$$\mathbb{E}_{v_i}[\textit{CLIQUE}(\mathbf{R}_{i-1}, v_i)] \preccurlyeq \frac{2}{|\mathcal{V}_{\mathbf{R}_{i-1}}|}\sum_{e \in \mathcal{E}_{\mathbf{R}_{i-1}}} w_{\mathbf{R}_{i-1}}(e)\mathbf{\Delta}_e = \frac{2}{|\mathcal{V}_{\mathbf{R}_{i-1}}|}\mathbf{R}_{i-1} \tag{21}$$

Now, we prove an extension of the Chernoff bound lemma to random matrices as follows:

Lemma B.4. *Let* $\mathbf{N}$ *be a random self-adjoint matrix satisfying:* $0 \preccurlyeq \mathbf{N} \preccurlyeq r\mathbf{I}$*, Then* $\log \mathbb{E}e^{\theta \mathbf{N}} \preccurlyeq \frac{(e^{r\cdot\theta}-1)}{r}(\mathbb{E}\mathbf{N})$*,* $\forall \theta \in \mathbb{R}$*.*

Proof. The classical version on this statement uses $r = 1$ and has been proved in Tropp (2012) and Lemma 1.12 in Tropp (2019). The proof sketch for the custom case is straightforward and can be derived from the observation that the function $\mathbf{N} \to e^{\theta \mathbf{N}}$ is convex, singular values of $\mathbf{N}$ lie in $[0, r]$ and $\log(1 + x) \leq x$ for valid x. This gives us:

$$e^{\theta \mathbf{N}} \preccurlyeq \mathbf{I} + \frac{(e^{r\cdot\theta}-1)}{r}\mathbf{N} \implies \log \mathbb{E}e^{\theta \mathbf{N}} \preccurlyeq \frac{(e^{r\cdot\theta}-1)}{r}(\mathbb{E}\mathbf{N})$$

□

By leveraging Lemma B.4 and Eq. 19 , we can bound $\mathit{CLIQUE}(\mathbf{R}_{i-1}, v_i)$ as $0 \preccurlyeq \mathit{CLIQUE}(\mathbf{R}_{i-1}, v_i) \preccurlyeq \|\mathbf{R}_{i-1}\| \cdot \mathbf{I}$. Now, if we assume a bound on the largest singular values of $\mathbf{R}_{i-1}$ as $\|\mathbf{R}_{i-1}\| \leq \mathcal{B}_{\mathbf{R}_{i-1}}$, the Chernoff corrector can be given using Eq. 21 as follows:

$$\frac{(e^{\mathcal{B}_{\mathbf{R}_{i-1}} \cdot \theta} - 1)}{\mathcal{B}_{\mathbf{R}_{i-1}}} \cdot \frac{2}{|\mathcal{V}_{\mathbf{R}_{i-1}}|} \mathbf{R}_{i-1} \tag{22}$$

B.1.4. Composite Corrector

Putting together the Bernstein corrector in Eq. 17 and the Chernoff corrector in Eq. 22, we can create a composite corrector that accounts for the randomness determined by o_n and o_v. The composition rule stated and proved in section 7.3.7 of Tropp (2019) is defined here for context.

Proposition B.5. *(Tropp, 2019) : The composite rule states that, given the sigma fields $\mathcal{F}_0 \subset \mathcal{F}_1 \subset \mathcal{F}_2$, Let $\mathbf{P}$ be a random matrix measurable over $\mathcal{F}_2$, $\mathbf{V}_1, \mathbf{M}_1$ are measurable over $\mathcal{F}_1$, $\mathbf{V}_0, \mathbf{M}_0$ are measurable over $\mathcal{F}_0$. For $\theta > 0$, suppose:*

$$\mathbb{E}\big[\mathrm{tr}\big(\exp(\mathbf{M}_1 + \theta\mathbf{P} - g(\theta)\mathbf{V}_1)\big)|\mathcal{F}_1\big] \leq \mathrm{tr}\big(\exp(\mathbf{M}_1)\big)$$
$$\mathbb{E}\big[\mathrm{tr}\big(\exp(\mathbf{M}_0 + \theta\mathbf{V}_1 - h(\theta)\mathbf{V}_0)\big)\big] \leq \mathrm{tr}\big(\exp(\mathbf{M}_0)\big)$$

holds true, then $(h \circ g)\mathbf{V}_0$ is the corrector for $\mathbf{P}$.

Where $(h \circ g)$ represents function composition. To use this proposition, we can compose the Bernstein corrector $g(\theta)V_1 = \big(\frac{\theta^2}{2(1-\mathcal{B}_{\mathbf{U}_i}|\theta|/3)}\mathcal{B}_{\mathbf{C}_i}\big)\mathit{CLIQUE}(\mathbf{R}_{i-1}, v_i)$ from Eq. 17 and the Chernoff corrector $h(\theta)V_0 = \frac{(e^{\mathcal{B}_{\mathbf{R}_{i-1}}\theta}-1)}{\mathcal{B}_{\mathbf{R}_{i-1}}} \cdot \frac{2}{|\mathcal{V}_{\mathbf{R}_{i-1}}|}\mathbf{R}_{i-1}$ from Eq. 22 to define the composite corrector as follows:

$$\frac{2}{\mathcal{B}_{\mathbf{R}_{i-1}}|\mathcal{V}_{\mathbf{R}_{i-1}}|} \cdot \left[\exp\left(\frac{\mathcal{B}_{\mathbf{R}_{i-1}}\mathcal{B}_{\mathbf{C}_i}\theta^2}{2(1-\mathcal{B}_{\mathbf{U}_i}|\theta|/3)}\right) - 1\right] \cdot \mathbf{R}_{i-1} \tag{23}$$

B.1.5. Deviation bound

Definition B.6. Corrector to Corrector process: Proposition 7.10 in Tropp (2019): For a function $\widetilde{g} : [0, \infty] \to [0, \infty]$, let $\{\widetilde{\mathbf{Y}}_k\}$ be a matrix martingale of self-adjoint matrices with difference sequence $\{\widetilde{\mathbf{U}}_k\} = \{\widetilde{\mathbf{Y}}_k - \widetilde{\mathbf{Y}}_{k-1}, k = 1, 2, \dots\}$. Let $\{\widetilde{\mathbf{V}}_k\}$ be a predictable sequence of self-adjoint matrices. For each k, suppose $\widetilde{g}\widetilde{\mathbf{V}}_k$ is a corrector for $\widetilde{\mathbf{U}}_k$, conditional on the sigma field $\mathcal{F}_{k-1}$, then the predictable process $\widetilde{\mathbf{W}}_k = \sum_{i=1}^{k} \widetilde{\mathbf{V}}_i$ generates a corrector $\{\widetilde{g}\widetilde{\mathbf{W}}_k\}$ for the martingale $\{\widetilde{\mathbf{Y}}_k\}$.

Since we are interested in the $\{\mathbf{L}_i - \mathbf{L}_0\}$ martingale, each matrix in this sequence can be expanded as:

$$\begin{aligned}\mathbf{L}_i - \mathbf{L}_0 &= \mathbf{L}_i - \mathbf{L}_{i-1} + \mathbf{L}_{i-1} - \mathbf{L}_{i-2} + \cdots + \mathbf{L}_1 - \mathbf{L}_0 \\ &= \mathbf{C}_i - \mathbb{E}[\mathbf{C}_i|v_i] + \mathbf{C}_{i-1} - \mathbb{E}[\mathbf{C}_{i-1}|v_{i-1}] + \cdots + \mathbf{C}_1 - \mathbb{E}[\mathbf{C}_1|v_1]\end{aligned}$$

Where the second equality follows from Eq. 13. Therefore, based on Definition B.6, the corrector process $\{\widetilde{g}\widetilde{\mathbf{W}}_i, i = 1, \dots\}$ for martingale $\{\mathbf{L}_i - \mathbf{L}_0\}$ is given by:

$$\widetilde{g}\widetilde{\mathbf{W}}_i = \sum_{j=1}^{i} \frac{2}{\mathcal{B}_{\mathbf{R}_{j-1}}|\mathcal{V}_{\mathbf{R}_{j-1}}|} \cdot \left[\exp\left(\frac{\mathcal{B}_{\mathbf{R}_{j-1}}\mathcal{B}_{\mathbf{C}_j}\theta^2}{2(1-\mathcal{B}_{\mathbf{U}_j}|\theta|/3)}\right) - 1\right] \cdot \mathbf{R}_{j-1} \tag{24}$$

Finally, we state the master tail bound theorem for matrix martingales (see Theorem 7.4 in Tropp (2019) and Freedman (1975) for details on the proof based on martingale stopping time).

Theorem B.7. *(Tropp, 2019) If $\{\widetilde{g}\widetilde{\mathbf{W}}_k\}$ is a corrector process for a martingale $\{\widetilde{\mathbf{Y}}_k \in \mathbb{R}^{N\times N}\}$ of self-adjoint matrices and $\sigma_{max}(.)$ returns the largest singular value, then:*

$$P(\exists k \geq 0 : \sigma_{max}(\widetilde{\mathbf{Y}}_k) \geq t \text{ and } \sigma_{max}(\widetilde{\mathbf{W}}_k) \leq q) \leq N \cdot \inf_{\theta>0} e^{-\theta t + \widetilde{g}(\theta)q}, \forall t, q \in \mathbb{R} \tag{25}$$

The deviation tail bound for the martingale $\{L_i - L_0\}$ can now be given as:

$$\begin{aligned} P(\|\mathbf{L}_i - \mathbf{L}_0\| > \epsilon) &\leq P(\exists j : \|\mathbf{L}_j - \mathbf{L}_0\| > \epsilon\} \\ &\leq P(\exists j : \sigma_{max}(\mathbf{L}_j - \mathbf{L}_0) \geq \epsilon) + P(\exists j : \sigma_{max}(-(\mathbf{L}_j - \mathbf{L}_0)) \geq \epsilon) \\ &\leq 2N \cdot \inf_{\theta > 0} \exp(-\epsilon\theta + \widetilde{g}(\theta)\eta^2) \end{aligned} \tag{26}$$

Where $\sigma_{max}(\widetilde{\mathbf{W}}_i) \leq \eta^2$. The first inequality is a split based on the singular values so that Theorem B.7 can be applied, to obtain the tail bound. The factor $2N$ instead of N (as per Theorem B.7) is obtained due to the two cases of singular value bounds in the second inequality of Eq. 26 and based on the validity of the corrector for the negation of the martingale $-(\mathbf{L}_j - \mathbf{L}_0)$. The $\widetilde{g}(\theta)$ term in the bound is obtained from the θ dependent scalar terms of the composite corrector in Eq. 24. Observe that, based on Eq. 11 and 19, it is sufficient if we can track $\mathcal{B}_{\mathbf{R}_{i-1}}$ during the elimination process as $\mathbf{R}_i$ essentially captures the clique approximations $\mathbf{C}_i$. Thus, by controlling $\mathcal{B}_{\mathbf{R}_{i-1}}$, we can have a provable bound on the randomness induced during the view generation steps using $rLap$.

The analysis by Kyng & Sachdeva (2016); Tropp (2019) provides one such approach. Their work applies a normalization map to the clique approximations $\Phi(\mathbf{C}_i) = \mathbf{L}^{-1/2}\mathbf{C}_i\mathbf{L}^{-1/2}$ and leverages the concept of effective resistances (Spielman & Srivastava, 2008) to split the edges in the graph into ξ multi-edges (each with weight $w_{\mathbf{L}}(e)/\xi$) and bound the spectral norm $\|\Phi(\mathbf{R}_i)\|$. A key takeaway from their approach is that the probability of deviation can be reduced by increasing ξ. However, note that this line of analysis aims at better pre-conditioning of the laplacian linear system with $\mathbf{L}$. In our analysis, since we are only concerned with introducing randomness into our graph augmentations and preserving the Schur complement properties in expectation, we present a generalized analysis and avoid strict assumptions.

Runtime: To analyse the runtime complexity of Algorithm 1, observe that at iteration i, when a node v_i is being eliminated, the corresponding *STAR*(v_i) can be removed from $\mathbf{R}_{i-1}$ in $O(deg(v_i))$ with efficient sparse tensor data types. Next, applying o_n can lead to $O(deg(v_i)\log(deg(v_i)))$ overheads. The clique sampling loop runs $O(deg(v_i))$ times with $O(1)$ overheads to sample a neighbor and add the new edge. This can be achieved by using a bi-directional linked list representation of the graph with an additional list for node pointer lookup. Finally, assuming a random node elimination scheme o_v (i.e, our default variant of $rLap$), $|\mathcal{V}| = N, |\mathcal{E}| = M, deg(v_i) = O(\frac{M}{N-i+1})$, the time complexity to eliminate $k = \gamma|\mathcal{V}|$ nodes is:

$$\begin{aligned} O\Big(\sum_{i=1}^{k} \frac{M}{N-i+1} \log\big(\frac{M}{N-i+1}\big)\Big) &= O\Big(\sum_{i=0}^{N-1} \frac{M}{N-i} \log\big(\frac{M}{N-i}\big)\Big) \\ &= O\Big(\sum_{i=0}^{N-1} \frac{M\log M}{N-i} - \frac{M\log(N-i)}{N-i}\Big) \\ &= O\Big(M\log M \sum_{i=0}^{N-1} \frac{1}{N-i} - M\sum_{i=0}^{N-1} \frac{\log(N-i)}{N-i}\Big) \\ &= O\Big(M\log M\log N - M\big(\frac{\log N}{N} + \cdots \frac{\log 1}{1}\big)\Big) \\ &= O\Big(M\log M\log N\Big) \end{aligned}$$

Note that if one employs a random neighbor ordering scheme for o_n, the time complexity can be reduced to $O(M\log N)$. Additionally, one can carefully design a priority queue that maintains the degrees of nodes and looks up the elimination node in $O(deg(v_i))$. Thus, o_v based on min-degree elimination can still be achieved in $O\Big(M\log M\log N\Big)$ time.

C. Ablation study on randomized Schur complements

In the following ablation study, we consider two variants of o_v and three variants of o_n as mentioned in Table 6 for $rLap$. The scheme $o_v = rand$ indicates a random selection of nodes for elimination and $o_v = deg$ indicates a selection based on the minimum degree nodes which haven't been eliminated. Formally, by employing a priority queue of nodes based on their

current degree values, the o_v scheme returns a node with the lowest degree at each iteration for elimination. Once a node v_i has been selected, the functionality of o_n is to order the neighbors $x_l \in \mathcal{N}_{\mathbf{R}_{i-1}}(v_i)$ based on weights of edges $w_{\mathbf{R}_{i-1}}(v_i, x_l)$. The schemes $o_n = asc, o_n = desc, o_n = rand$ indicate sorting of $x_l \in \mathcal{N}_{\mathbf{R}_{i-1}}(v_i)$ in the ascending, descending and random order of edge weights respectively. The variants of $rLap$ follow the $rLap - o_v - o_n$ naming convention.

Table 6. Variants of $rLap$ based on o_v, o_n.

VARIANT	o_v	o_v DESCRIPTION	o_n	o_n DESCRIPTION
RLAP-RAND-ASC	RAND	RANDOM SELECTION	ASC	ASCENDING ORDER OF $w_{\mathbf{R}_{i-1}}(v_i, x_l)$
RLAP-RAND-DESC	RAND	RANDOM SELECTION	DESC	DESCENDING ORDER OF $w_{\mathbf{R}_{i-1}}(v_i, x_l)$
RLAP-RAND-RAND	RAND	RANDOM SELECTION	RAND	RANDOM ORDER OF $w_{\mathbf{R}_{i-1}}(v_i, x_l)$
RLAP-DEG-ASC	DEG	MIN-DEGREE SELECTION	ASC	ASCENDING ORDER OF $w_{\mathbf{R}_{i-1}}(v_i, x_l)$
RLAP-DEG-DESC	DEG	MIN-DEGREE SELECTION	DESC	DESCENDING ORDER OF $w_{\mathbf{R}_{i-1}}(v_i, x_l)$
RLAP-DEG-RAND	DEG	MIN-DEGREE SELECTION	RAND	RANDOM ORDER OF $w_{\mathbf{R}_{i-1}}(v_i, x_l)$

On a related note, the $vertex - coarsening$ approach by Fahrbach et al. (2020) attains the same goal of approximating Schur complements in expectation and is closely related to our work. However, it differs from $rLap$ in its clique approximation approach and chooses the $o_v = deg$ scheme for node elimination. The notion of o_n doesn't apply to their edge sampling procedure when approximating the clique. To understand the benefits of our approach and the role of o_v, o_n, we compare the variants of $rLap$ (including $vertex - coarsening$) on all the benchmark datasets.

Table 7, 8 compare the unsupervised node classification performance of $rLap$ variants and $vertex - coarsening$ using the GRACE and MVGRL design. The setup and evaluation protocols are the same as per Appendix D. In a majority of cases, the $rLap - rand$ variants outperformed $rLap - degree$ variants, followed by $vertex - coarsening$. Thus, showcasing the effectiveness of our clique sampling strategy. A similar pattern can be observed for unsupervised graph classification tasks in Table 9, 10 using the GraphCL and BGRL designs respectively. However, the $vertex - coarsening$ approach dominates the $rLap$ variants when the BGRL design is employed for IMDB-BINARY, i.e the minimum degree-based elimination scheme for o_v seems to outperform the random variants in this case.

Table 7. Evaluation (in accuracy) on benchmark node datasets with **GRACE** design and $rLap$ variants.

AUGMENTOR	CORA	AMAZON-PHOTO	PUBMED	COAUTHOR-CS	COAUTHOR-PHY
RLAP-RAND-ASC	$\mathbf{83.75 \pm 2.64}$	$\mathbf{92.59 \pm 1.05}$	84.56 ± 0.71	$\mathbf{93.1 \pm 0.54}$	$\mathbf{95.83 \pm 0.44}$
RLAP-RAND-DESC	80.4 ± 3.43	92.37 ± 0.91	$\mathbf{85.49 \pm 0.88}$	92.71 ± 0.57	95.73 ± 0.25
RLAP-RAND-RAND	82.94 ± 3.88	92.52 ± 1.18	84.91 ± 0.7	92.67 ± 0.71	95.56 ± 0.39
RLAP-DEG-ASC	80.37 ± 1.8	89.7 ± 0.58	84.22 ± 0.85	92.87 ± 0.6	95.39 ± 0.4
RLAP-DEG-DESC	83.23 ± 2.35	90.98 ± 1.28	84.45 ± 0.66	92.84 ± 0.25	95.24 ± 0.4
RLAP-DEG-RAND	81.62 ± 3.6	88.27 ± 1.47	84.8 ± 0.77	92.63 ± 0.75	94.79 ± 0.38
VERTEX COARSENING	78.31 ± 2.25	90.58 ± 1.09	84.18 ± 1.0	92.77 ± 0.78	94.58 ± 0.36

Table 8. Evaluation (in accuracy) on benchmark node datasets with **MVGRL** design and $rLap$ variants.

AUGMENTOR	CORA	AMAZON-PHOTO	PUBMED	COAUTHOR-CS	COAUTHOR-PHY
RLAP-RAND-ASC	$\mathbf{83.68 \pm 2.04}$	87.14 ± 1.34	$\mathbf{84.21 \pm 0.46}$	91.73 ± 0.53	$\mathbf{94.81 \pm 0.31}$
RLAP-RAND-DESC	82.24 ± 1.66	86.14 ± 0.89	83.88 ± 0.51	91.6 ± 0.48	94.53 ± 0.2
RLAP-RAND-RAND	81.84 ± 1.26	87.28 ± 1.29	83.79 ± 0.97	91.57 ± 0.49	94.24 ± 0.43
RLAP-DEG-ASC	82.32 ± 2.29	$\mathbf{87.52 \pm 1.0}$	83.61 ± 0.74	91.74 ± 0.64	94.18 ± 0.34
RLAP-DEG-DESC	83.38 ± 1.65	86.47 ± 1.25	83.54 ± 0.64	91.82 ± 0.58	94.5 ± 0.35
RLAP-DEG-RAND	81.73 ± 2.47	87.11 ± 1.06	84.02 ± 0.49	$\mathbf{91.85 \pm 0.81}$	94.23 ± 0.33
VERTEX COARSENING	82.02 ± 2.18	87.36 ± 1.0	83.62 ± 0.67	90.79 ± 0.49	94.12 ± 0.35

C.1. Maximum singular value and edge count analysis

For a better understanding of the effectiveness of these variants, we set $\gamma = 0.5$ and vary the choices of o_v, o_n to track $\sigma_{max}(\mathbf{R}_i)$ as an indicator of $\mathcal{B}_{\mathbf{R}_i}$. As analyzing hundreds of Schur complements in graph classification settings defeats the

Table 9. Evaluation (in accuracy) on benchmark graph datasets with **GraphCL** design and $rLap$ variants.

AUGMENTOR	PROTEINS	IMDB-BINARY	MUTAG	IMDB-MULTI	NCI1
RLAP-RAND-ASC	$\mathbf{75.27 \pm 3.34}$	$\mathbf{70.9 \pm 5.01}$	$\mathbf{87.5 \pm 9.86}$	$\mathbf{48.66 \pm 2.68}$	$\mathbf{75.06 \pm 1.65}$
RLAP-RAND-DESC	72.61 ± 4.75	68.3 ± 4.29	85.5 ± 7.83	48.23 ± 2.73	74.14 ± 1.9
RLAP-RAND-RAND	73.34 ± 4.54	68.8 ± 4.12	83.5 ± 5.5	47.8 ± 4.16	74.43 ± 2.15
RLAP-DEG-ASC	74.29 ± 2.76	69.3 ± 3.85	85.0 ± 7.42	47.2 ± 4.31	74.2 ± 2.34
RLAP-DEG-DESC	73.04 ± 2.76	70.7 ± 5.88	82.0 ± 8.43	46.33 ± 4.91	74.89 ± 1.43
RLAP-DEG-RAND	73.66 ± 2.86	67.5 ± 4.2	81.5 ± 8.5	47.13 ± 4.15	73.6 ± 2.17
VERTEX COARSENING	71.61 ± 4.51	67.6 ± 3.85	80.0 ± 9.75	45.6 ± 4.32	74.09 ± 2.5

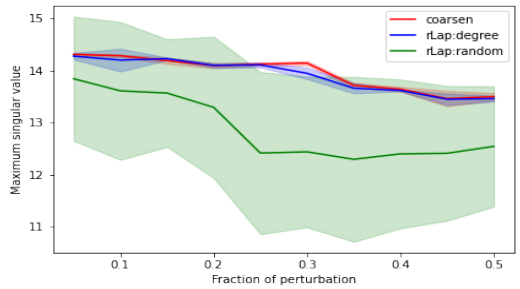
Table 10. Evaluation (in accuracy) on benchmark graph datasets with **BGRL** design and $rLap$ variants.

AUGMENTOR	PROTEINS	IMDB-BINARY	MUTAG	IMDB-MULTI	NCI1
RLAP-RAND-ASC	$\mathbf{84.34 \pm 4.03}$	74.8 ± 10.17	$\mathbf{81.5 \pm 5.39}$	$\mathbf{59.47 \pm 7.42}$	$\mathbf{75.15 \pm 3.57}$
RLAP-RAND-DESC	83.72 ± 4.3	75.1 ± 8.87	80.6 ± 8.66	58.73 ± 7.13	73.77 ± 3.34
RLAP-RAND-RAND	83.14 ± 6.38	74.2 ± 7.74	80.3 ± 5.39	57.07 ± 5.52	74.72 ± 2.97
RLAP-DEG-ASC	83.62 ± 5.44	79.4 ± 5.94	81.0 ± 10.26	58.6 ± 4.73	74.8 ± 2.89
RLAP-DEG-DESC	83.48 ± 4.91	79.2 ± 7.3	80.5 ± 5.22	58.53 ± 5.2	73.36 ± 4.17
RLAP-DEG-RAND	83.55 ± 5.62	79.0 ± 8.75	80.8 ± 10.2	59.36 ± 5.39	74.3 ± 2.36
VERTEX COARSENING	82.3 ± 6.72	$\mathbf{80.5 \pm 5.71}$	78.5 ± 7.76	56.67 ± 5.38	73.58 ± 2.37

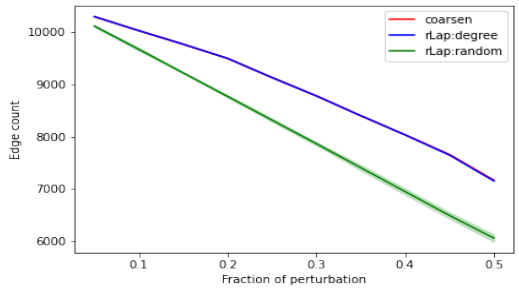
purpose of our analysis, we restricted our analysis to single graph node classification datasets to systematically track the Schur complements. Figure 3(a) plots the trend of $\sigma_{max}(\mathbf{R}_i)$ using $vertex - coarsening$ and $rLap$ variants with a default setting of $o_n = asc$ on the CORA dataset. Figure 3(b) plots the edge count trend of $\mathbf{R}_i$ for these three techniques. Since $rLap$ with $o_v = rand$ gave the best performance on most of the benchmark datasets, we tracked $\sigma_{max}(\mathbf{R}_i)$ for $rLap$ with $o_v = rand$ and $o_n = asc, desc, rand$ in Figure 3(c), with an edge count trend being presented in Figure 3(d). A similar analysis is conducted for AMAZON-PHOTO (see Figure 4), PUBMED (see Figure 5), COAUTHOR-CS (see Figure 6) and COAUTHOR-PHY (see Figure 7).

The $\sigma_{max}(\mathbf{R}_i)$ values for $rLap$ with $o_v = rand, o_n = asc$ tend to be relatively smaller than $o_v = deg, o_n = asc$ and $vertex - coarsening$ approaches on CORA and AMAZON-PHOTO. However, the variance in their values is larger than the latter two. On the contrary, the Schur complements have relatively higher $\sigma_{max}(\mathbf{R}_i)$ values for PUBMED, COAUTHOR-CS and COAUTHOR-PHY datasets. This indicates higher stochasticity in the augmented graph views when compared to $rLap$ with $o_v = deg, o_n = asc$, and $vertex - coarsening$. Also, note that degree-based elimination techniques lead to relatively dense $\mathbf{R}_i$ as they generate Schur complements corresponding to a set of highly connected nodes at every iteration. Thus, $\widetilde{\mathcal{G}}_1, \widetilde{\mathcal{G}}_2$ generated by such approaches lead to extra computation overheads for message passing in GNN encoders. For instance, $\mathbf{R}_i$ computed by $rLap$ with $o_v = rand, o_n = asc$ on COAUTHOR-PHY and AMAZON-PHOTO with $\gamma = 0.5$ is sparser by $\approx 50,000, \approx 60,000$ edges respectively when compared to $\mathbf{R}_i$ obtained using $rLap$ with $o_v = deg, o_n = asc$. Since $vertex - coarsening$ is based on minimum degree-based elimination, we see a close resemblance in its trends of $\sigma_{max}(\mathbf{R}_i)$ and edge counts with $rLap$ based on $o_v = deg, o_n = asc$.

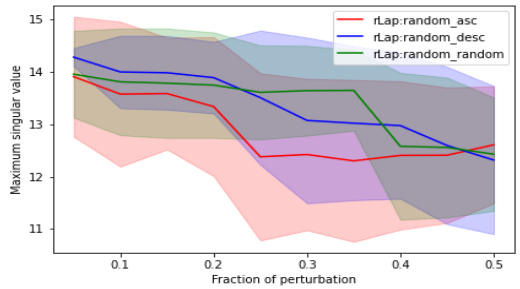
On the other hand, when $o_v = rand$ is fixed and o_n is varied, the $rLap$ variants tend to generate $\mathbf{R}_i$ with similar trends of $\sigma_{max}(\mathbf{R}_i)$ across datasets. However, we observe a noticeable difference in the edge count trend as $\gamma \to 0.5$. For instance, $\mathbf{R}_i$ with $o_v = rand, o_n = asc$ is sparser by $\approx 20,000$ edges when compared to $o_v = rand, o_n = desc$ on the AMAZON-PHOTO dataset. The relatively higher average degree of nodes in the AMAZON-PHOTO graph is one of the reasons for this observation. On the other hand, PUBMED has a relatively lower average degree which explains a minor difference of ≈ 2000 edges in Schur complements of these $rLap$ variants. Intuitively, nodes with higher weights (indicative of connectivity) have a higher conditional probability of having an edge in the approximate clique. When $o_n = desc$, such nodes are processed in the initial iterations of the inner loop and later iterations lead to new edges between nodes with relatively less degree. The scenario is flipped when $o_n = asc$ and the sampled edges tend to frequently merge with existing ones (note that we merge edges to avoid multi-graphs). Thus, leading to relatively sparser $\mathbf{R}_i$.



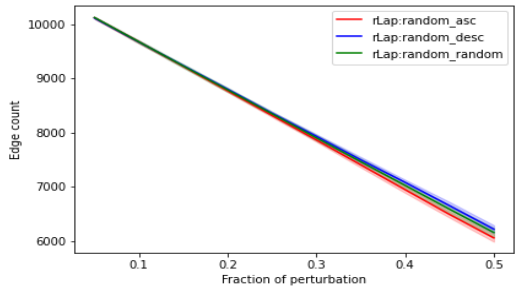
(a) Trend of $\sigma_{max}(\mathbf{R}_i)$ on CORA



(b) Trend of $\mathbf{R}_i$ edge count on CORA

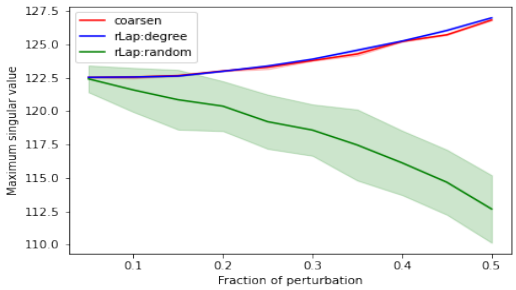


(c) Trend of $\sigma_{max}(\mathbf{R}_i)$ on CORA

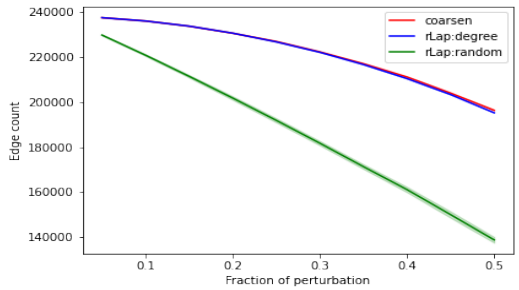


(d) Trend of $\mathbf{R}_i$ edge count on CORA

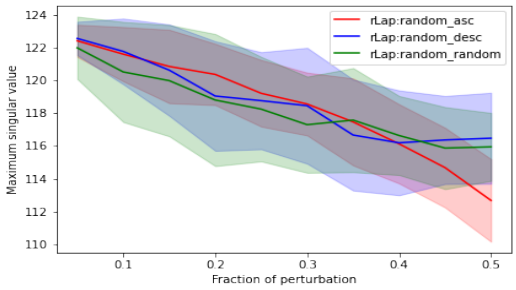
Figure 3. Plots of $\sigma_{max}(\mathbf{R}_i)$ and edge count vs fraction of perturbation γ for $rLap$ variants and $vertex - coarsening$ on CORA. For the $rLap$ variants, plots (a), (b) illustrate the trends when $o_n = asc$ and plots (c), (d) illustrate the trends when $o_v = rand$.



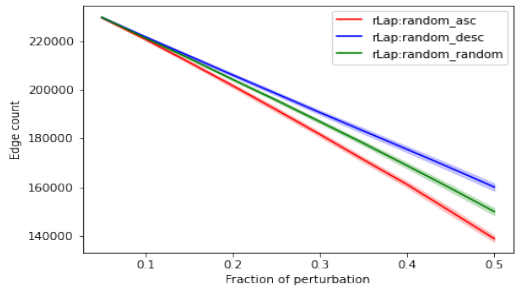
(a) Trend of $\sigma_{max}(\mathbf{R}_i)$ on AMAZON-PHOTO



(b) Trend of $\mathbf{R}_i$ edge count on AMAZON-PHOTO

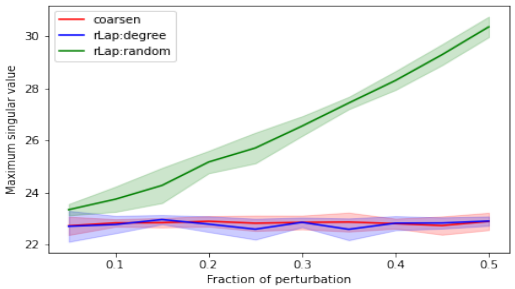


(c) Trend of $\sigma_{max}(\mathbf{R}_i)$ on AMAZON-PHOTO

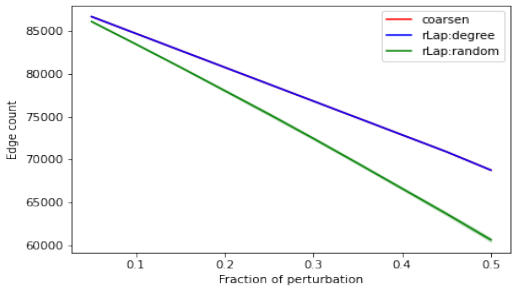


(d) Trend of $\mathbf{R}_i$ edge count on AMAZON-PHOTO

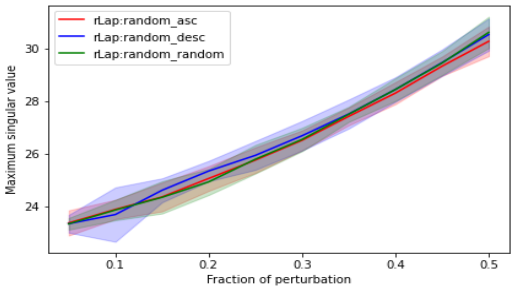
Figure 4. Plots of $\sigma_{max}(\mathbf{R}_i)$ and edge count vs fraction of perturbation γ for $rLap$ variants and $vertex - coarsening$ on AMAZON-PHOTO. For the $rLap$ variants, plots (a), (b) illustrate the trends when $o_n = asc$ and plots (c), (d) illustrate the trends when $o_v = rand$.



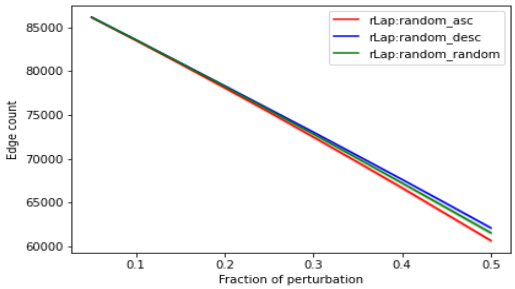
(a) Trend of $\sigma_{max}(\mathbf{R}_i)$ on PUBMED



(b) Trend of $\mathbf{R}_i$ edge count on PUBMED

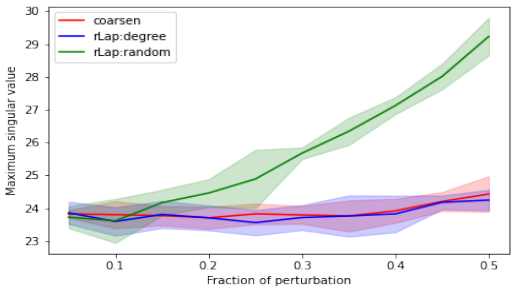


(c) Trend of $\sigma_{max}(\mathbf{R}_i)$ on PUBMED

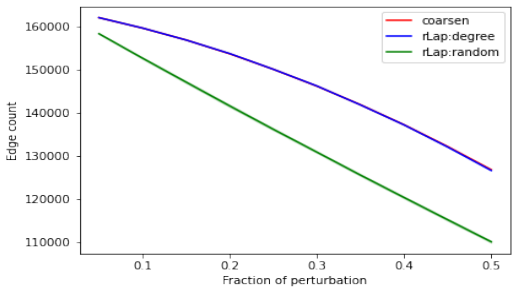


(d) Trend of $\mathbf{R}_i$ edge count on PUBMED

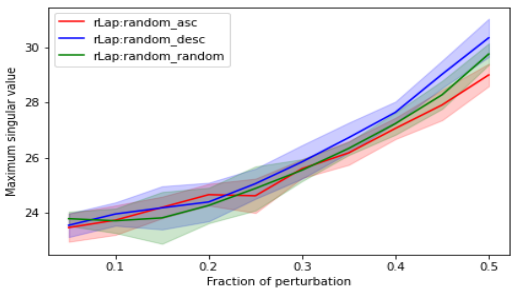
Figure 5. Plots of $\sigma_{max}(\mathbf{R}_i)$ and edge count vs fraction of perturbation γ for $rLap$ variants and $vertex - coarsening$ on PUBMED. For the $rLap$ variants, plots (a), (b) illustrate the trends when $o_n = asc$ and plots (c), (d) illustrate the trends when $o_v = rand$.



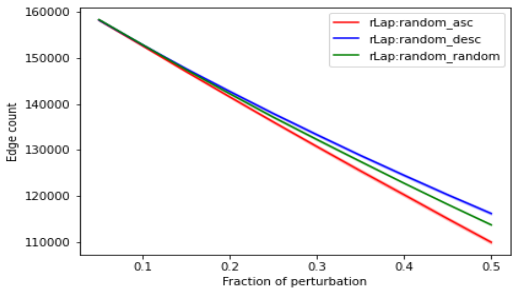
(a) Trend of $\sigma_{max}(\mathbf{R}_i)$ on COAUTHOR-CS



(b) Trend of $\mathbf{R}_i$ edge count on COAUTHOR-CS

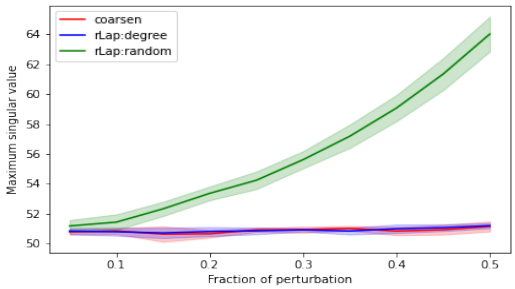


(c) Trend of $\sigma_{max}(\mathbf{R}_i)$ on COAUTHOR-CS

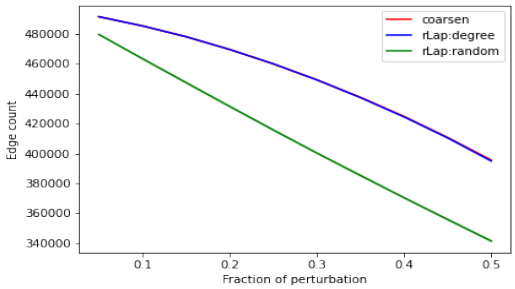


(d) Trend of $\mathbf{R}_i$ edge count on COAUTHOR-CS

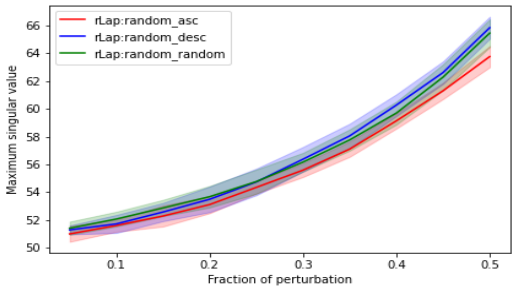
Figure 6. Plots of $\sigma_{max}(\mathbf{R}_i)$ and edge count vs fraction of perturbation γ for $rLap$ variants and $vertex - coarsening$ on COAUTHOR-CS. For the $rLap$ variants, plots (a), (b) illustrate the trends when $o_n = asc$ and plots (c), (d) illustrate the trends when $o_v = rand$.



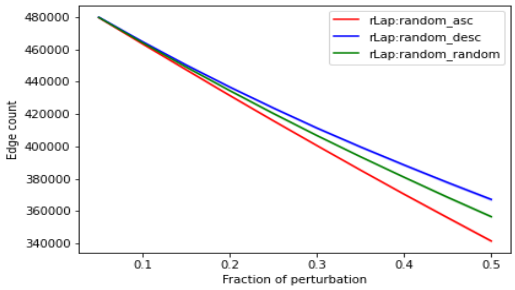
(a) Trend of $\sigma_{max}(\mathbf{R}_i)$ on COAUTHOR-PHY



(b) Trend of $\mathbf{R}_i$ edge count on COAUTHOR-PHY



(c) Trend of $\sigma_{max}(\mathbf{R}_i)$ on COAUTHOR-PHY



(d) Trend of $\mathbf{R}_i$ edge count on COAUTHOR-PHY

Figure 7. Plots of $\sigma_{max}(\mathbf{R}_i)$ and edge count vs fraction of perturbation γ for $rLap$ variants and $vertex-coarsening$ on COAUTHOR-PHY. For the $rLap$ variants, plots (a), (b) illustrate the trends when $o_n = asc$ and plots (c), (d) illustrate the trends when $o_v = rand$.

D. Experimental setup

D.1. Platform specifications

We conduct experiments on a virtual machine with 8 Intel(R) Xeon(R) Platinum 8268 CPUs, 24GB of RAM and 1 Quadro RTX 8000 GPU with 32GB of allocated memory. For reproducible experiments, we leverage the open-source PyGCL framework by Zhu et al. (2021b) along with PyTorch 1.12.1 (Paszke et al., 2019) and PyTorch-Geometric (PyG) 2.1.0 (Fey & Lenssen, 2019). To ensure low latencies, the $rLap$ algorithm is written in C++, uses the Eigen library for tensor operations, is built using Bazel and has Python bindings for wider adaptability.

D.2. Datasets

We experiment on 10 widely used node and graph classification datasets. For node classification, we experiment on CORA (McCallum et al., 2000), PUBMED (Sen et al., 2008), AMAZON-PHOTO, COAUTHOR-CS and COAUTHOR-PHY (Shchur et al., 2018). For graph classification, we use the MUTAG (Debnath et al., 1991), PROTEINS-full (PROTEINS) (Borgwardt et al., 2005), IMDB-BINARY, IMDB-MULTI (Yanardag & Vishwanathan, 2015) and NCI1 (Wale et al., 2008) datasets. All the datasets are available in PyG and can be imported directly. Table 11 provides a summary of their statistics.

D.3. Augmentors

EdgeAddition: To randomly add edges to the graph $\mathcal{G}$, a mask $\mathcal{M} \in \mathbb{R}^{N \times N}$ is sampled using the Bernoulli distribution as follows: $\mathcal{M}[i,j] \sim Bern(\gamma)$, where γ is the probability of adding an edge. Since we use two views for contrasting, masks $\mathcal{M}_1, \mathcal{M}_2$ are sampled using edge addition probabilities γ_1, γ_2 respectively and applied to the adjacency matrix $\mathbf{A}$ of $\mathcal{G}$ to obtain: $\widetilde{\mathbf{A}}_1 = \mathbf{A} \odot \mathcal{M}_1, \widetilde{\mathbf{A}}_2 = \mathbf{A} \odot \mathcal{M}_2$. Here $\odot$ indicates a bit-wise OR operator. As per the notation defined in preliminaries, the resulting views $\widetilde{\mathcal{G}}_1, \widetilde{\mathcal{G}}_2$ are the graphs corresponding to $\widetilde{\mathbf{A}}_1, \widetilde{\mathbf{A}}_2$.

EdgeDropping: This technique uses the same mask sampling process as EdgeAddition for $\mathcal{M}_1, \mathcal{M}_2$ but employs a bit-wise AND operator to mask and drop the edges.

EdgeDroppingDegree: Unlike the above-mentioned EdgeDropping scheme, this adaptive technique takes into account the

Table 11. Statistics of benchmark datasets from PyG

DATASET	DOMAIN	TASK	#GRAPHS	AVG #NODES	AVG #EDGES	#FEAT	#CLASSES
CORA	CS	NODE	1	2,708	10,556	1,433	7
AMAZON-PHOTO	E-COMMERCE	NODE	1	7,650	238,162	745	8
PUBMED	MEDICAL	NODE	1	19,717	88,648	500	3
COAUTHOR-CS	CS	NODE	1	18,333	163,788	6,805	15
COAUTHOR-PHY	PHY	NODE	1	34,493	495,924	8,415	5
MUTAG	BIO	GRAPH	188	≈ 17.9	≈ 39.6	7	2
PROTEINS	BIO	GRAPH	1,113	≈ 39.1	≈ 145.6	3	2
IMDB-BINARY	MOVIE	GRAPH	1,000	≈ 19.8	≈ 193.1	0	2
IMDB-MULTI	SOCIAL	GRAPH	1,500	≈ 13.0	≈ 131.88	0	3
NCI1	BIO	GRAPH	4,110	≈ 29.87	≈ 64.6	0	2

degree of nodes at either end of an edge and performs a weighted sampling. The probability of dropping an edge between nodes v_i, v_j is given by:

$$\gamma_{ij} = \min\left(\frac{(\log s^{deg})_{max} - \log s^{deg}_{ij}}{(\log s^{deg})_{max} - (\log s^{deg})_{avg}} \cdot \gamma_e, \gamma_\tau\right) \tag{27}$$

Where $s^{deg}_{ij} = [deg(v_i) + deg(v_j)]/2$ and $(\log s^{deg})_{max}, (\log s^{deg})_{avg}$ correspond to the maximum and average of log degree centrality scores for edges in the graph. Furthermore, γ_e is the desired (user-provided) probability of dropping an edge and γ_τ is the cut-off probability to truncate high probabilities of removal (Zhu et al., 2021c).

EdgeDroppingEVC: The leading eigen vector centrality of a node v_i is equal to $\mathbf{u}[i]$ where $\mathbf{A}\mathbf{u} = \lambda_{max}\mathbf{u}$ and $\mathbf{u}$ is the eigen vector corresponding to the largest eigen value λ_{max}. By replacing s^{deg}_{ij} in Eq. 27 with $s^{evc}_{ij} = [\mathbf{u}[i] + \mathbf{u}[j]]/2$, we get the eigenvector centrality based edge dropping scheme.

EdgeDroppingPR: The page-rank centrality vector represents the weights computed by the page rank algorithm (Page et al., 1999) and is given by $\mathbf{p} = \alpha\mathbf{A}\mathbf{D}^{-1}\mathbf{p} + \mathbf{I}$ (Zhu et al., 2021c). By replacing s^{deg}_{ij} in Eq. 27 with $s^{pr}_{ij} = [\mathbf{p}[i] + \mathbf{p}[j]]/2$, we get the page-rank centrality based edge dropping scheme.

NodeDropping: Similar to the edge dropping scheme, this technique drops a node v_i from the graph based on the $Bern(\gamma)$ distribution, where γ is the probability of dropping a node. Computationally, this is equivalent to setting the row and column values corresponding to the dropped node to $\mathbf{0}$.

RandomWalkSubgraph: This is a sub-graph sampling technique where a random walk with restart is performed on $\mathcal{G}$ and the sub-graph induced by this random walk is taken as the augmented view (Zhu et al., 2021b).

PPRDiffusion, MarkovDiffusion: The diffusion operator on a graph (introduced in Eq. 6) is an effective way of capturing global structural information. The PPRDiffusion and MarkovDiffusion schemes are based on this formulation as follows:

$$\begin{aligned} \mathbf{S}^{PPR} &= \alpha(\mathbf{I} - (1-\alpha)\mathbf{T})^{-1} \\ \mathbf{S}^{MD} &= \frac{1}{K}\sum_{i=1}^{K}(\alpha\mathbf{I} + (1-\alpha)\mathbf{T}^i) \end{aligned} \tag{28}$$

Where K represents the maximum step count for the markov diffusion (Zhu & Koniusz, 2020), α in $\mathbf{S}^{PPR}, \mathbf{S}^{MD}$ indicates the dampening factor and $\mathbf{T} = \mathbf{A}\mathbf{D}^{-1}$. We use $\alpha = 0.2$ and $K = 10$ in our experiments. Next, to address the increase in edge connectivity of the diffused graph, we perform sparsification based on thresholding with $\epsilon = 10^{-4}$ as proposed by Klicpera et al. (2019). Note that for diffusion based augmentors, the second view is the original graph without any augmentation and the notion of perturbation γ doesn't apply.

rLap: Based on our setup and Algorithm 1, the augmented view is generated by $rLap(\mathcal{G}, \gamma, o_v, o_n)$, where $\mathcal{G}$ is the input weighted graph, γ indicates the fraction of nodes to eliminate, o_v is the node elimination scheme and o_n is the neighbor ordering scheme. The algorithm leverages the weighted laplacian $\mathbf{L}$ to perform GE-based elimination for $\gamma|\mathcal{V}|$ iterations and returns the randomized Schur complement matrix (which is also a laplacian) and represents the augmented view of $\mathcal{G}$.

Feature Masking: Based on our setup, a graph $\mathcal{G}$ is associated with a feature matrix $\mathbf{X} = [\mathbf{x}_1, \cdots, \mathbf{x}_N]^\top \in \mathbb{R}^{N \times d}$. For every row $\mathbf{x}_i \in \mathbb{R}^d, i \in 1, \cdots, N$, a mask $m_i \in \mathbb{R}^d$ is sampled from a Bernoulli distribution as follows: $m_i[j] \sim Bern(1-\gamma), \forall j \in 1, \cdots, d$, where γ is the probability of masking an entry. These masks are then applied on the rows of $\mathbf{X}$ to obtain the masked features $\widehat{\mathbf{X}} = [\mathbf{x}_1 \odot m_1, \cdots, \mathbf{x}_N \odot m_N]^\top \in \mathbb{R}^{N \times d}$. Here $\odot$ indicates a bit-wise AND operator.

D.4. Contrastive objectives

We employ Information Noise Contrastive Estimation (InfoNCE), Jensen-Shannon Divergence (JSD), and Bootstrap Latent (BL) objectives in our experiments.

InfoNCE (Oord et al., 2018): Without loss of generality, consider a sample u_i for which $\mathcal{S}(u_i)$ and $\mathcal{D}(u_i)$ indicate the set of similar and dissimilar samples respectively. Samples can indicate nodes or graphs as per the context. The InfoNCE loss is now given by:

$$\ell_{InfoNCE} = -\frac{1}{|\mathcal{S}(u_i)|} \sum_{s_j \in \mathcal{S}(u_i)} \log \left(\frac{e^{\varphi(u_i, s_j)/t}}{e^{\varphi(u_i, s_j)/t} + \sum\limits_{d_j \in \mathcal{D}(u_i)} e^{\varphi(u_i, d_j)/t}} \right) \tag{29}$$

Where $t \in \mathbb{R}$ is the scaling/temperature parameter and $\varphi(.)$ measures the similarity of features $\psi(.)$ learned by the GNN encoder (followed by projector as per design) as follows:

$$\varphi(u_i, s_j) = \frac{\psi(u_i)^\top \psi(s_j)}{\|\psi(u_i)\|_2 \, \|\psi(s_j)\|_2} \tag{30}$$

JSD (Lin, 1991): With the same setup as InfoNCE, we use JSD loss based on softplus and sigmoid function $sig(.)$ as:

$$\ell_{JSD} = -\frac{1}{|\mathcal{S}(u_i)|} \sum_{s_j \in \mathcal{S}(u_i)} \log \left(1 + e^{-sig(\psi(u_i)^\top \psi(s_j))}\right) - \frac{1}{|\mathcal{D}(u_i)|} \sum_{d_j \in \mathcal{D}(u_i)} \log \left(1 + e^{sig(\psi(u_i)^\top \psi(d_j))}\right) \tag{31}$$

BL (Thakoor et al., 2021): The BL loss is independent of the dissimilar or negative samples $\mathcal{D}(u_i)$ and attempts to maximize the cosine similarity between embeddings of similar samples. We follow the design presented in Thakoor et al. (2021) and employ an online and target encoder to generate embeddings and minimize the following:

$$\ell_{BL} = -\frac{1}{|\mathcal{S}(u_i)|} \sum_{s_j \in \mathcal{S}(u_i)} \frac{\varphi_{on}(u_i)^\top \varphi_{tar}(s_j)}{\|\varphi_{on}(u_i)\|_2 \, \|\varphi_{tar}(s_j)\|_2} \tag{32}$$

We follow the standard benchmarking procedure by Zhu et al. (2021b) and incorporate additional constraints presented in Thakoor et al. (2021), such as the exponential moving average-based update of target encoder parameters.

D.5. GCL framework designs

GRACE: Figure 8 illustrates the design for a GRACE (Zhu et al., 2020) inspired framework. The input graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, w)$ is augmented by T_1, T_2 to generate views $\widetilde{\mathcal{G}}_1, \widetilde{\mathcal{G}}_2$. These graphs are encoded using a shared GNN f and a shared MLP projector z_n to generate node features. The node-level features across these two views are contrasted using the InfoNCE objective under 'l-l' mode. We employ this design for node classification where congruent counterparts of nodes form the positive/similar samples and the rest of the nodes are selected as negative/dissimilar samples. We deviate from the design of Zhu et al. (2020) by incorporating only inter-view dissimilar samples to reduce the computational overheads.

MVGRL: Figure 9 illustrates the design for a MVGRL (Hassani & Khasahmadi, 2020) inspired framework. The input graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, w)$ is augmented by T_1, T_2 to generate views $\widetilde{\mathcal{G}}_1, \widetilde{\mathcal{G}}_2$. These graphs are encoded using dedicated GNNs f_1, f_2 and projected using a shared MLP z_n to compute node features. The node embeddings computed by f_1, f_2 are aggregated using a permutation invariant function $\bigoplus$ (eg: mean) and projected using a shared MLP z_g to obtain graph features. Finally, these graph and node-level features and contrasted using the JSD objective under 'g-l' mode. In this mode, when only a single graph $\mathcal{G}$ is available in the dataset, the positive samples for graph features of $\widetilde{\mathcal{G}}_1$ include the node features of $\widetilde{\mathcal{G}}_2$ with negative samples being their noisy versions. The same procedure is applied to the graph features of $\widetilde{\mathcal{G}}_2$ and node features of $\widetilde{\mathcal{G}}_1$ (Velickovic et al., 2019; Hassani & Khasahmadi, 2020; Zhu et al., 2021b).

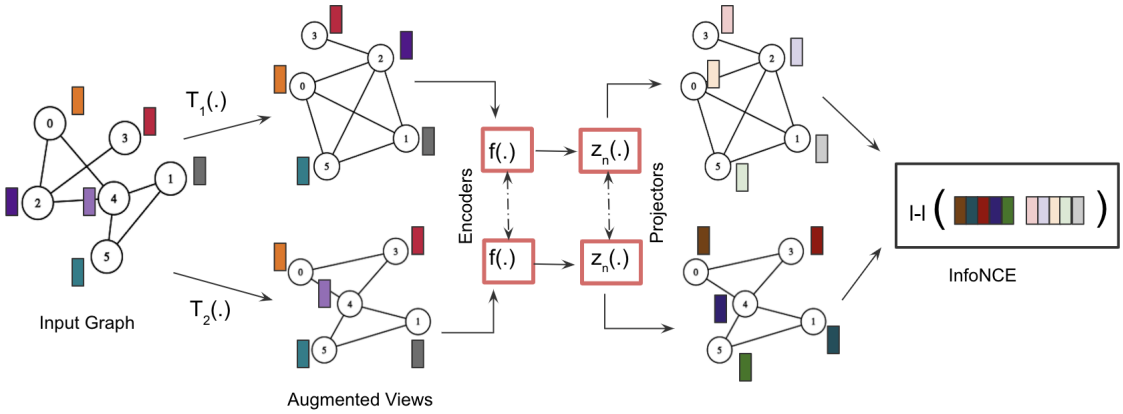


Figure 8. GRACE design with shared encoder f, shared node feature projector z_n, l-l contrastive mode and InfoNCE objective.

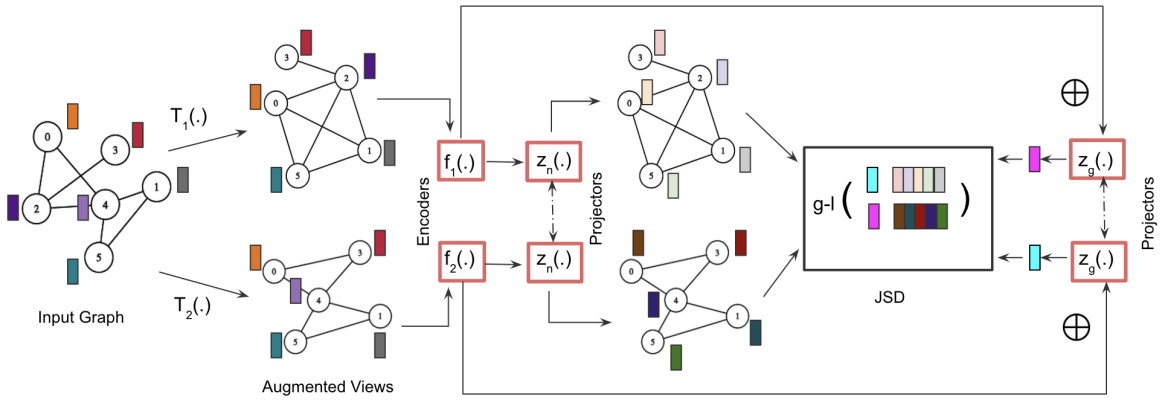


Figure 9. MVGRL design with dedicated encoders f_1, f_2, shared node feature projector z_n, shared graph feature projector z_g, g-l contrastive mode and JSD objective.

GraphCL: Figure 10 illustrates the design for a GraphCL (You et al., 2020) inspired framework. The input graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, w)$ is augmented by T_1, T_2 to generate views $\widetilde{\mathcal{G}}_1, \widetilde{\mathcal{G}}_2$. These graphs are encoded using a shared GNN f to obtain the node embeddings. These node embeddings are aggregated using a permutation invariant function $\bigoplus$ (eg: mean) and projected using a shared MLP z_g to obtain graph features. Finally, these graph features are contrasted using the InfoNCE objective under 'g-g' mode. For the graph features of $\widetilde{\mathcal{G}}_1$, the graph features of $\widetilde{\mathcal{G}}_2$ are considered as positive samples and the graph features of other graphs in the training batch are treated as negative samples.

BGRL: Figure 11 illustrates the design for a BGRL (Thakoor et al., 2021) inspired framework. The input graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, w)$ is augmented by T_1, T_2 to generate views $\widetilde{\mathcal{G}}_1, \widetilde{\mathcal{G}}_2$. Unlike the previously mentioned frameworks, these graphs are encoded using an online and target GNNs f_1, f_2 to compute node embeddings. The online encoder f_1 computes node-level embeddings for both the views and projects them using an MLP z_{n1}. The target encoder f_2 and MLP projector z_{n2} repeat the same procedure, followed by computing the graph-level features[6] using a permutation invariant function $\bigoplus$ (eg: mean). Finally, these graph and node-level features and contrasted using the BL objective under 'g-l' mode. The parameters of f_2 are updated based on an exponential moving average of f_1 parameters. This design is independent of negative samples and avoids the computational overheads of employing graph and node features across the training batch.

[6]Unlike MVGRL where MLP-based projections z_{g1}, z_{g2} are employed, BGRL skips the graph feature projection step.

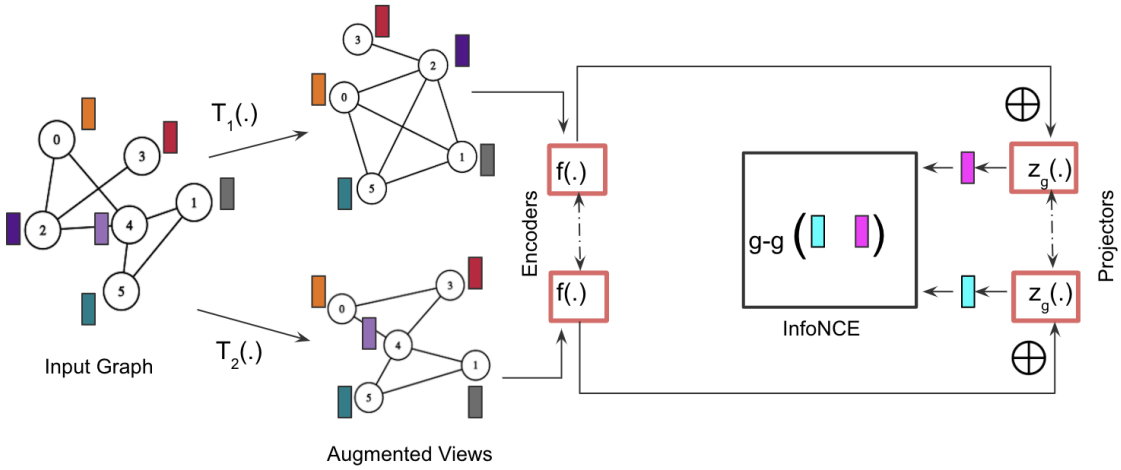


Figure 10. GraphCL design with shared encoder f, shared graph feature projector z_g, g-g contrastive mode and InfoNCE objective.

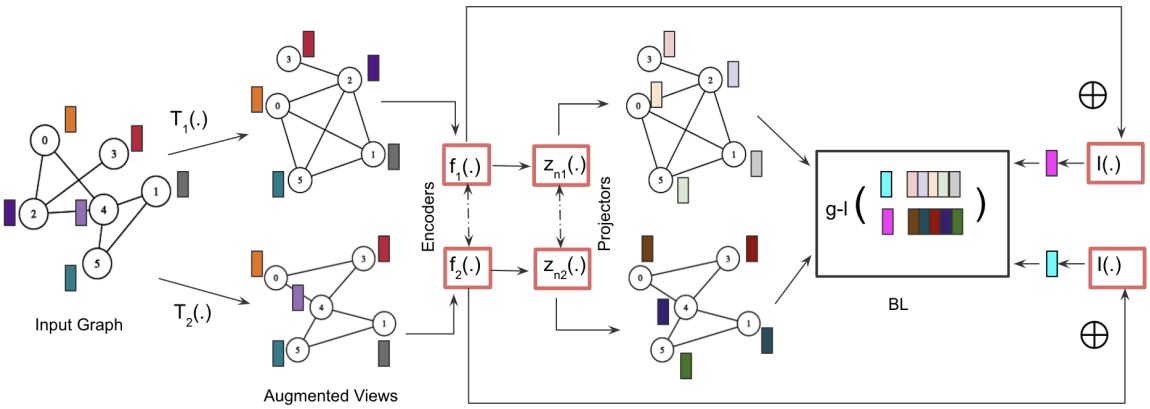


Figure 11. BGRL design with online, target encoders f_1, f_2, their node projectors z_{n1}, z_{n2}, g-l contrastive mode and BL objective.

D.6. Evaluation protocols

Generally, GCL frameworks differ in the choice of augmentors, encoder designs, contrastive modes and objective functions. Thus, it is unclear how the performance of an augmentor can be fairly tested. To address this issue, we fix as many configurations as possible and focus solely on the impact of the augmentor. Our reasoning is based on the well-established benchmark study by Zhu et al. (2021b). Adhering to this approach, we leverage the designs of widely adopted frameworks such as GRACE (Zhu et al., 2020), MVGRL (Hassani & Khasahmadi, 2020), GraphCL (You et al., 2020) and BGRL (Thakoor et al., 2021), where topological augmentation is necessary and evaluate on unsupervised node and graph classification tasks. These frameworks were chosen so as to extensively experiment on encoder designs, contrastive modes and objectives. We leverage graph convolution network (GCN) (Kipf & Welling, 2016a) and graph isomorphism network (GIN) (Xu et al., 2018) based encoders for node and graph-based tasks respectively. Also, based on the empirical benchmark observations in Zhu et al. (2021b), we perform feature masking with a constant masking ratio of 0.3 in all the experiments and perform an extensive grid-search over the hyper-parameters to find their ideal values. The perturbation ratios γ_1, γ_2 are selected from $\{0.0, 0.1, 0.2, 0.3, 0.4, 0.5\}$ to prevent excessive graph corruption. The number of layers in the encoder are chosen from $\{2, 4, 8\}$, hidden feature dimensions are chosen from $\{128, 256, 512\}$, learning rate is chosen from $\{10^{-2}, 10^{-3}, 10^{-4}\}$, weight decay is set to 10^{-5} and the maximum epoch count is set to 2000. An early stopping strategy with a patience interval

of 50 epochs with respect to contrastive loss is also employed. Finally, we choose the Adam optimizer for learning the embeddings. For testing, we follow a linear evaluation protocol (Velickovic et al., 2019; Zhu et al., 2021b) where the embeddings learned by the encoders in unsupervised settings are classified using a logistic regression classifier with a train, validation and test split of $10\%, 10\%, 80\%$ respectively. The process is repeated with 10 random splits and the classification accuracies are reported. The selected hyper-parameters based on a grid search are reported in Table 12, 13, 14, 15 and can be reproduced with the provided code and instructions.

Table 12. Hyper-parameters for GRACE-based design for node-classification

DATASET	MODE	LOSS	τ	γ_1	γ_2	L	LR	WD	HIDDEN DIM
CORA	L-L	INFONCE	0.2	0.5	0.4	2	10^{-4}	10^{-5}	256
AMAZON-PHOTO	L-L	INFONCE	0.2	0.3	0.4	2	10^{-2}	10^{-5}	256
PUBMED	L-L	INFONCE	0.2	0.4	0.1	2	10^{-3}	10^{-5}	256
COAUTHOR-CS	L-L	INFONCE	0.4	0.5	0.4	2	10^{-4}	10^{-5}	256
COAUTHOR-PHY	L-L	INFONCE	0.4	0.5	0.5	2	10^{-2}	10^{-5}	128

Table 13. Hyper-parameters for MVGRL-based design for node-classification

DATASET	MODE	LOSS	γ_1	γ_2	L	LR	WD	HIDDEN DIM
CORA	G-L	JSD	0.0	0.4	2	10^{-3}	10^{-5}	512
AMAZON-PHOTO	G-L	JSD	0.0	0.3	2	10^{-4}	10^{-5}	512
PUBMED	G-L	JSD	0.1	0.3	2	10^{-3}	10^{-5}	512
COAUTHOR-CS	G-L	JSD	0.0	0.2	2	10^{-4}	10^{-5}	512
COAUTHOR-PHY	G-L	JSD	0.1	0.4	2	10^{-3}	10^{-5}	512

Table 14. Hyper-parameters for GraphCL-based design for graph-classification

DATASET	MODE	LOSS	τ	γ_1	γ_2	L	LR	WD	HIDDEN DIM
PROTEINS	G-G	INFONCE	0.5	0.2	0.2	2	10^{-2}	10^{-5}	128
IMDB-BINARY	G-G	INFONCE	0.5	0.3	0.5	2	10^{-2}	10^{-5}	128
MUTAG	G-G	INFONCE	0.5	0.2	0.2	2	10^{-3}	10^{-5}	128
IMDB-MULTI	G-G	INFONCE	0.5	0.2	0.2	2	10^{-2}	10^{-5}	128
NCI1	G-G	INFONCE	0.2	0.2	0.3	2	10^{-3}	10^{-5}	128

Table 15. Hyper-parameters for BGRL-based design for graph-classification

DATASET	MODE	LOSS	γ_1	γ_2	L	LR	WD	HIDDEN DIM
PROTEINS	G-L	BL	0.1	0.1	2	10^{-2}	10^{-5}	256
IMDB-BINARY	G-L	BL	0.5	0.3	2	10^{-3}	10^{-5}	128
MUTAG	G-L	BL	0.2	0.2	2	10^{-3}	10^{-5}	128
IMDB-MULTI	G-L	BL	0.2	0.2	2	10^{-3}	10^{-5}	128
NCI1	G-L	BL	0.3	0.1	2	10^{-3}	10^{-5}	256

D.7. Baselines

We report the baseline node and graph classification performance from previously published reports (Hassani & Khasahmadi, 2020; You et al., 2020; Zhu et al., 2021c; Xu et al., 2021). For node classification, we leverage raw features (Velickovic et al., 2019), DeepWalk (Perozzi et al., 2014), Graph Auto Encoder (GAE) (Kipf & Welling, 2016b), Deep Graph Infomax (DGI) (Velickovic et al., 2019), GCN networks and Graph Attention Networks (GAT) (Veličković et al., 2017) (see Table

16). For graph classification, we leverage Graphlet Kernel (GK) (Shervashidze et al., 2009), Weisfeiler-Lehman Kernel (WL) (Shervashidze et al., 2011), Deep Graph Kernel (DGK) (Yanardag & Vishwanathan, 2015), Multi-scale Laplacian Graph Kernel (MLG) (Kondor & Pan, 2016), node2vec (Grover & Leskovec, 2016), sub2vec (Adhikari et al., 2018), graph2vec (Narayanan et al., 2017), InfoGraph (Sun et al., 2019), GIN, GCN and GAT networks (see Table 17).

Table 16. Baseline node classification accuracies from published reports.

METHOD	CORA	AMAZON-PHOTO	PUBMED	COAUTHOR-CS	COAUTHOR-PHY
RAW FEAT	47.90 ± 0.40	78.53 ± 0.00	69.10 ± 0.30	90.37 ± 0.00	93.58 ± 0.00
DEEPWALK	67.20 ± 0.00	89.44 ± 0.11	65.30 ± 0.00	84.61 ± 0.22	91.77 ± 0.15
DEEPWALK+FEAT	70.70 ± 0.60	90.05 ± 0.08	74.30 ± 0.90	87.70 ± 0.04	94.90 ± 0.09
GAE	71.50 ± 0.40	91.62 ± 0.13	72.10 ± 0.50	90.01 ± 0.71	94.92 ± 0.07
DGI	82.30 ± 0.60	91.61 ± 0.22	76.80 ± 0.60	92.15 ± 0.63	94.51 ± 0.52
GCN	81.50 ± 0.00	92.42 ± 0.22	79.0 ± 0.00	93.03 ± 0.31	95.65 ± 0.16
GAT	83.0 ± 0.70	92.56 ± 0.35	79.0 ± 0.30	92.31 ± 0.24	95.47 ± 0.15

Table 17. Baseline graph classification accuracies from published reports.

METHOD	MUTAG	PROTEINS	IMDB-BINARY	IMDB-MULTI	NCI1
GK	81.70 ± 2.10	–	65.87 ± 0.98	43.90 ± 0.40	–
WL	80.72 ± 3.00	72.92 ± 0.56	72.30 ± 3.44	47.0 ± 0.50	80.01 ± 0.50
DGK	87.44 ± 2.72	73.30 ± 0.82	66.96 ± 0.56	44.60 ± 0.50	80.31 ± 0.46
MLG	87.9 ± 1.60	–	66.6 ± 0.30	41.2 ± 0.00	80.8 ± 1.30
NODE2VEC	72.63 ± 10.20	57.49 ± 3.57	–	–	54.89 ± 1.61
SUB2VEC	61.05 ± 15.80	53.03 ± 5.55	55.26 ± 1.54	36.70 ± 0.80	52.84 ± 1.47
GRAPH2VEC	83.15 ± 9.25	73.30 ± 2.05	71.10 ± 0.54	50.40 ± 0.90	73.22 ± 1.81
INFOGRAPH	89.01 ± 1.13	74.44 ± 0.31	73.03 ± 0.87	49.70 ± 0.50	76.20 ± 1.06
GIN	89.4 ± 5.60	76.2 ± 2.80	75.1 ± 5.10	52.3 ± 2.80	82.7 ± 1.70
GCN	85.6 ± 5.80	75.2 ± 3.60	74.0 ± 3.4	51.9 ± 3.8	80.2 ± 2.0
GAT	89.4 ± 6.10	74.7 ± 4.00	70.5 ± 2.30	47.8 ± 3.10	66.6 ± 2.20

E. Augmentor Overheads

We establish an augmentor overhead benchmark in terms of memory usage and latency for an extensive range of techniques on the benchmark datasets. Specifically, we measure:

MEMORY: The memory (in MB) that is required solely by the augmentation phase.

LATENCY(CPU): The wall-clock time (in seconds) for performing the augmentation.

LATENCY(w/ GPU): The wall-clock time (in seconds) for performing the augmentation when GPU is available.

We run 10 experiments for each augmentor-dataset combination with a fixed perturbation ratio of 0.5 (and use a batch size of 128 for graph datasets) to report the mean and standard deviation of the metrics in Table 18, 19. From Table 18, we can observe that simple NodeDropping and EdgeDropping schemes are computationally the most efficient, with EdgeDroppingEVC and diffusion-based approaches being the least. The code for EdgeDroppingEVC has been adopted from the original paper by Zhu et al. (2021c) and leverages the NetworkX library to compute centrality scores. Furthermore, since this approach lacks GPU support, the hardware acceleration is minimal. In fact, the overheads of data transfers from GPU to CPU and the construction of NetworkX graphs significantly increase the memory consumption and latencies for this technique. Similar patterns can be observed for other augmentors as well where the GPU-CPU overheads dominate the parallelization benefits. On the contrary, diffusion-based approaches leverage GPU acceleration for heavy matrix operations and gain significant speedups to achieve reasonable latencies. A similar pattern can be observed in Table 19 for graph datasets with a batch size of 128. Note that some of the results show 0 variance due to numerical rounding.

The overheads of $rLap$ in Table 18, 19 lie within a very narrow margin of the most efficient techniques such as EdgeDropping, NodeDropping and RandomWalkSubgraph. These low latencies are achieved by designing the augmentor in C++ and

minimizing the interaction with Python APIs. Additionally, we compile Eigen without MKL optimizations and also do not support a GPU kernel for $rLap$ yet, which explains the lack of performance improvement when hardware acceleration is available. However, we tend to achieve minimal resource consumption by just leveraging the C++ APIs to the fullest. We leave the development of GPU support for $rLap$ as future work.

Table 18. Statistics of resource consumption by augmentors on node classification datasets.

AUGMENTOR	DATASET	MEMORY	LATENCY(CPU)	LATENCY(W/ GPU)
EdgeAddition	CORA	3.23 ± 0.09	0.0024 ± 0.0	0.0033 ± 0.0001
EdgeDropping	CORA	0.51 ± 0.0539	0.0004 ± 0.0	0.0019 ± 0.0001
EdgeDroppingDegree	CORA	3.85 ± 1.1307	0.003 ± 0.001	0.0046 ± 0.0032
EdgeDroppingEVC	CORA	39.58 ± 0.14	0.2214 ± 0.0038	0.3542 ± 0.0021
EdgeDroppingPR	CORA	3.18 ± 0.0872	0.0023 ± 0.0001	0.0037 ± 0.0001
MarkovDiffusion	CORA	228.73 ± 38.1103	0.7966 ± 0.1115	0.2539 ± 0.0519
NodeDropping	CORA	1.31 ± 1.3729	0.0006 ± 0.0	0.0026 ± 0.0008
PPRDiffusion	CORA	140.0 ± 0.1342	0.2531 ± 0.0017	0.5933 ± 0.0589
RandomWalkSubgraph	CORA	3.26 ± 0.102	0.001 ± 0.0	0.0027 ± 0.0001
rLap	CORA	0.38 ± 0.04	0.0037 ± 0.0005	0.0048 ± 0.0002
EdgeAddition	AMAZON-PHOTO	20.94 ± 5.4471	0.0368 ± 0.0017	0.0071 ± 0.0003
EdgeDropping	AMAZON-PHOTO	5.64 ± 0.5953	0.0032 ± 0.0015	0.0022 ± 0.0001
EdgeDroppingDegree	AMAZON-PHOTO	26.13 ± 4.2223	0.0391 ± 0.0007	0.0041 ± 0.0001
EdgeDroppingEVC	AMAZON-PHOTO	2264.1 ± 0.004	1.554 ± 0.012	1.4576 ± 0.024
EdgeDroppingPR	AMAZON-PHOTO	11.32 ± 1.4999	0.0094 ± 0.0001	0.0043 ± 0.0003
MarkovDiffusion	AMAZON-PHOTO	158.42 ± 3.0019	30.6898 ± 0.127	1.7687 ± 0.0044
NodeDropping	AMAZON-PHOTO	1.53 ± 0.19	0.0017 ± 0.0012	0.0028 ± 0.0001
PPRDiffusion	AMAZON-PHOTO	389.33 ± 2.1808	1.6942 ± 0.0247	0.8229 ± 0.0112
RandomWalkSubgraph	AMAZON-PHOTO	7.26 ± 0.1356	0.0024 ± 0.0001	0.0029 ± 0.0007
rLap	AMAZON-PHOTO	32.6 ± 0.5119	0.0848 ± 0.0023	0.0891 ± 0.0014
EdgeAddition	PUBMED	10.93 ± 0.7785	0.0126 ± 0.0002	0.0045 ± 0.0001
EdgeDropping	PUBMED	1.89 ± 0.1136	0.002 ± 0.0003	0.0022 ± 0.0001
EdgeDroppingDegree	PUBMED	15.45 ± 2.1238	0.015 ± 0.001	0.0032 ± 0.0003
EdgeDroppingEVC	PUBMED	2236.23 ± 0.7537	1.2787 ± 0.0152	1.3101 ± 0.0156
EdgeDroppingPR	PUBMED	4.79 ± 0.2071	0.0059 ± 0.0005	0.0043 ± 0.0003
MarkovDiffusion	PUBMED	152.87 ± 2.0995	44.4244 ± 0.1306	2.5587 ± 0.0054
NodeDropping	PUBMED	1.78 ± 0.172	0.0021 ± 0.0002	0.0036 ± 0.0002
PPRDiffusion	PUBMED	438.42 ± 1.8595	19.0975 ± 0.3605	2.5488 ± 0.0135
RandomWalkSubgraph	PUBMED	1.74 ± 0.1625	0.0044 ± 0.0003	0.003 ± 0.0001
rLap	PUBMED	10.59 ± 3.1908	0.0269 ± 0.0008	0.0315 ± 0.0039
EdgeAddition	COAUTHOR-CS	16.71 ± 3.0409	0.0286 ± 0.0015	0.0057 ± 0.0003
EdgeDropping	COAUTHOR-CS	3.02 ± 0.5758	0.007 ± 0.0018	0.0025 ± 0.0003
EdgeDroppingDegree	COAUTHOR-CS	15.1 ± 2.3078	0.0363 ± 0.0014	0.0037 ± 0.0001
EdgeDroppingEVC	COAUTHOR-CS	1044.6 ± 0.5745	10.2603 ± 0.0266	10.6018 ± 0.1096
EdgeDroppingPR	COAUTHOR-CS	5.09 ± 0.359	0.0114 ± 0.0003	0.0052 ± 0.001
MarkovDiffusion	COAUTHOR-CS	177.9667 ± 7.0002	66.5972 ± 0.1857	3.3583 ± 0.1758
NodeDropping	COAUTHOR-CS	2.33 ± 0.1269	0.0035 ± 0.0004	0.0033 ± 0.0001
PPRDiffusion	COAUTHOR-CS	416.58 ± 3.9992	46.9031 ± 1.4907	2.2614 ± 0.1293
RandomWalkSubgraph	COAUTHOR-CS	5.63 ± 0.1792	0.0066 ± 0.0002	0.003 ± 0.0001
rLap	COAUTHOR-CS	13.6 ± 0.9798	0.0508 ± 0.0007	0.0602 ± 0.002
EdgeAddition	COAUTHOR-PHY	44.59 ± 8.3973	0.0975 ± 0.0048	0.0121 ± 0.0019
EdgeDropping	COAUTHOR-PHY	11.57 ± 1.5685	0.0098 ± 0.0009	0.002 ± 0.0001
EdgeDroppingDegree	COAUTHOR-PHY	20.7 ± 0.1483	0.1219 ± 0.0025	0.0046 ± 0.0003
EdgeDroppingEVC	COAUTHOR-PHY	2474.86 ± 0.4294	28.2138 ± 0.0304	28.9077 ± 0.2304
EdgeDroppingPR	COAUTHOR-PHY	16.96 ± 1.8205	0.0311 ± 0.0002	0.0041 ± 0.002
MarkovDiffusion	COAUTHOR-PHY	316.0 ± 0.4	316.1449 ± 0.1584	15.1789 ± 0.2203
NodeDropping	COAUTHOR-PHY	4.08 ± 1.7831	0.0067 ± 0.0004	0.0046 ± 0.0001
PPRDiffusion	COAUTHOR-PHY	695.0 ± 2.397	124.368 ± 2.9031	10.394 ± 0.2053
RandomWalkSubgraph	COAUTHOR-PHY	7.85 ± 0.9563	0.014 ± 0.0003	0.0057 ± 0.0047
rLap	COAUTHOR-PHY	43.9 ± 11.3004	0.191 ± 0.0258	0.2119 ± 0.0064

Table 19. Statistics of resource consumption by augmentors on graph classification datasets.

AUGMENTOR	DATASET	MEMORY	LATENCY(CPU)	LATENCY(W/ GPU)
EdgeAddition	IMDB-BINARY	4.78 ± 0.3628	0.0307 ± 0.0003	0.0191 ± 0.0121
EdgeDropping	IMDB-BINARY	2.15 ± 0.75	0.0079 ± 0.0038	0.0038 ± 0.0004
EdgeDroppingDegree	IMDB-BINARY	7.51 ± 0.6978	0.0346 ± 0.0002	0.0121 ± 0.001
EdgeDroppingEVC	IMDB-BINARY	2101.35 ± 2.3127	0.7977 ± 0.0068	0.9294 ± 0.0138
EdgeDroppingPR	IMDB-BINARY	2.27 ± 0.0458	0.0184 ± 0.0002	0.0166 ± 0.0001
MarkovDiffusion	IMDB-BINARY	195.43 ± 60.3246	8.2739 ± 0.0466	0.4589 ± 0.006
NodeDropping	IMDB-BINARY	2.03 ± 0.064	0.0046 ± 0.0001	0.0032 ± 0.0001
PPRDiffusion	IMDB-BINARY	170.13 ± 22.767	0.658 ± 0.0068	0.5295 ± 0.0105
RandomWalkSubgraph	IMDB-BINARY	2.09 ± 0.1136	0.0063 ± 0.0001	0.0096 ± 0.0016
rLap	IMDB-BINARY	6.96 ± 0.3583	0.0423 ± 0.0027	0.0412 ± 0.0009
EdgeAddition	IMDB-MULTI	5.25 ± 1.0249	0.0359 ± 0.0017	0.0193 ± 0.0001
EdgeDropping	IMDB-MULTI	2.18 ± 0.1327	0.0064 ± 0.0	0.0046 ± 0.0001
EdgeDroppingDegree	IMDB-MULTI	8.72 ± 1.1232	0.0365 ± 0.0001	0.0157 ± 0.0001
EdgeDroppingEVC	IMDB-MULTI	2119.91 ± 2.3364	0.806 ± 0.0104	0.8579 ± 0.0062
EdgeDroppingPR	IMDB-MULTI	4.45 ± 0.7159	0.0221 ± 0.0003	0.0236 ± 0.0004
MarkovDiffusion	IMDB-MULTI	28.02 ± 97.7836	5.5597 ± 0.0245	0.386 ± 0.0046
NodeDropping	IMDB-MULTI	2.17 ± 0.1005	0.0054 ± 0.0001	0.0045 ± 0.0012
PPRDiffusion	IMDB-MULTI	98.67 ± 41.6684	0.3907 ± 0.0183	0.7974 ± 0.0134
RandomWalkSubgraph	IMDB-MULTI	4.07 ± 0.9144	0.0075 ± 0.0001	0.0107 ± 0.0006
rLap	IMDB-MULTI	7.93 ± 0.6165	0.0421 ± 0.0023	0.0415 ± 0.0009
EdgeAddition	MUTAG	3.22 ± 0.14	0.0018 ± 0.0001	0.0046 ± 0.0001
EdgeDropping	MUTAG	1.61 ± 0.0943	0.0004 ± 0.0	0.0025 ± 0.0001
EdgeDroppingDegree	MUTAG	4.01 ± 0.7713	0.0021 ± 0.0001	0.0047 ± 0.0004
EdgeDroppingEVC	MUTAG	2118.92 ± 5.3619	0.0445 ± 0.0005	0.0683 ± 0.0061
EdgeDroppingPR	MUTAG	3.35 ± 0.1118	0.0025 ± 0.0001	0.006 ± 0.0004
MarkovDiffusion	MUTAG	164.0 ± 24.8769	0.3848 ± 0.0067	0.1774 ± 0.0075
NodeDropping	MUTAG	1.61 ± 0.0943	0.0008 ± 0.0	0.0033 ± 0.0001
PPRDiffusion	MUTAG	117.04 ± 0.2973	0.0829 ± 0.0019	0.602 ± 0.0101
RandomWalkSubgraph	MUTAG	2.7 ± 0.3821	0.0009 ± 0.0	0.0044 ± 0.0008
rLap	MUTAG	1.62 ± 0.3682	0.0021 ± 0.0	0.0041 ± 0.0001
EdgeAddition	PROTEINS	6.68 ± 0.3124	0.0288 ± 0.0014	0.0153 ± 0.0013
EdgeDropping	PROTEINS	2.58 ± 0.06	0.0058 ± 0.0012	0.004 ± 0.0001
EdgeDroppingDegree	PROTEINS	10.48 ± 1.6024	0.0319 ± 0.0003	0.0127 ± 0.0002
EdgeDroppingEVC	PROTEINS	2095.08 ± 0.6554	1.1132 ± 0.0209	1.3092 ± 0.2126
EdgeDroppingPR	PROTEINS	5.27 ± 0.064	0.0195 ± 0.0018	0.0182 ± 0.0001
MarkovDiffusion	PROTEINS	203.03 ± 84.7664	25.7943 ± 0.1671	1.1789 ± 0.0087
NodeDropping	PROTEINS	2.0 ± 0.6678	0.0079 ± 0.0019	0.0062 ± 0.0001
PPRDiffusion	PROTEINS	295.79 ± 26.6581	4.8792 ± 0.0261	1.6547 ± 0.549
RandomWalkSubgraph	PROTEINS	2.23 ± 0.064	0.0074 ± 0.0016	0.0099 ± 0.0001
rLap	PROTEINS	6.45 ± 0.3442	0.0405 ± 0.0024	0.0407 ± 0.0009
EdgeAddition	NCI1	4.26 ± 0.5122	0.0517 ± 0.0006	0.046 ± 0.0015
EdgeDropping	NCI1	1.72 ± 0.6447	0.0108 ± 0.0001	0.0085 ± 0.0002
EdgeDroppingDegree	NCI1	7.04 ± 0.6296	0.0574 ± 0.0013	0.0389 ± 0.0009
EdgeDroppingEVC	NCI1	2071.43 ± 5.9791	1.9465 ± 0.1448	2.3067 ± 0.0682
EdgeDroppingPR	NCI1	4.11 ± 0.1136	0.0548 ± 0.0064	0.0603 ± 0.0017
MarkovDiffusion	NCI1	141.825 ± 6.632	37.7575 ± 0.3818	2.0278 ± 0.0061
NodeDropping	NCI1	2.22 ± 0.1249	0.019 ± 0.0027	0.0219 ± 0.0008
PPRDiffusion	NCI1	157.16 ± 8.8786	8.8779 ± 0.0595	2.4582 ± 0.011
RandomWalkSubgraph	NCI1	2.68 ± 0.5862	0.0181 ± 0.0008	0.0229 ± 0.0014
rLap	NCI1	2.82 ± 0.5474	0.074 ± 0.0003	0.0801 ± 0.0011

F. Additional Experiments

F.1. PPR Diffusion and rLap

In our experiments, we observed that PPRDiffusion on medium scale graphs and MVGRL design leads to OOM and requires sub-graph sampling to proceed with the GCL phases. To this end, we observed a decrease in unsupervised node classification performance when the size of the sub-graph decreased (see Table 20). As the nodes for a sub-graph are randomly selected, one can't guarantee that the sub-graph will capture useful structural information for the GNN encoders to exploit. In a worst case scenario, if all the nodes of a sub-graph are disconnected, the GNN encoder essentially acts as an MLP. Thus, as the size of the sub-graph reduces, we can expect a heavy corruption of topological information in the augmented views.

Table 20. Evaluation (in accuracy) on benchmark node datasets with **MVGRL** based design and PPR Diffusion with varying sub-graph sizes. PPRDiffusion indicates the full graph view and PPRDiffusion-x indicates a sampled sub-graph with x nodes as the graph view.

AUGMENTOR	CORA	AMAZON-PHOTO	PUBMED	COAUTHOR-CS	COAUTHOR-PHY
PPRDIFFUSION	84.12 ± 2.72	90.65 ± 1.01	OOM	OOM	OOM
PPRDIFFUSION-8192	–	–	82.7 ± 0.85	90.9 ± 1.06	94.03 ± 0.5
PPRDIFFUSION-4096	–	87.38 ± 0.91	80.79 ± 1.4	89.0 ± 0.83	92.52 ± 0.39
PPRDIFFUSION-2048	80.93 ± 2.33	85.65 ± 1.41	76.4 ± 0.88	88.43 ± 0.77	89.24 ± 0.69

F.1.1. REDUCING COMPUTATIONAL OVERHEADS

For graphs such as COAUTHOR-PHY, the computational overheads of PPRDiffusion are relatively significant when compared to other augmentors (see Table 18). The standard practice to avoid such bottlenecks is to cache the output of the diffusion operator $\mathbf{S}^{PPR}$ and reuse it for sampling sub-graphs. To further reduce the overheads of computing $\mathbf{S}^{PPR}$ prior to caching, we leverage Theorem 3.2.

Sketch: The COAUTHOR-PHY graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ represents $|\mathcal{V}| = 34,493$ nodes and $|\mathcal{E}| = 495,924$ edges. To randomly sample a sub-graph of size 8192 from its diffused variant $\mathbf{S}^{PPR}$, one can compute a randomized Schur complement $\mathbf{R}_{\gamma|\mathcal{V}|}$ using $rLap$ with $\gamma = 0.5$, apply the PPR diffusion operator on $\mathbf{R}_{\gamma|\mathcal{V}|}$ to obtain $\widetilde{\mathbf{S}}^{PPR}_{\gamma}$ and cache it. $\widetilde{\mathbf{S}}^{PPR}_{\gamma}$ represents $\approx 17,000$ nodes from which a sub-graph of size 8192 can be sampled with relatively less overheads.

We benchmark the computational overheads of:

- PPRDiffusion on COAUTHOR-PHY graph followed by random sampling of a sub-graph with 8192 nodes.
- $rLap$ on COAUTHOR-PHY graph with $\gamma = 0.5, o_v = rand, o_n = asc$ followed by PPRDiffusion and random sampling of a sub-graph with 8192 nodes.

The results in Figure 12 demonstrate the effectiveness of the latter approach in achieving $\approx 0.25\times$ reduction in memory consumption, $\approx 10\times$ reduction in CPU only latencies and $\approx 5\times$ reduction in latencies when GPU is available. We also compared the unsupervised node classification accuracy and observed a slight performance improvement of $\approx 1\%$.

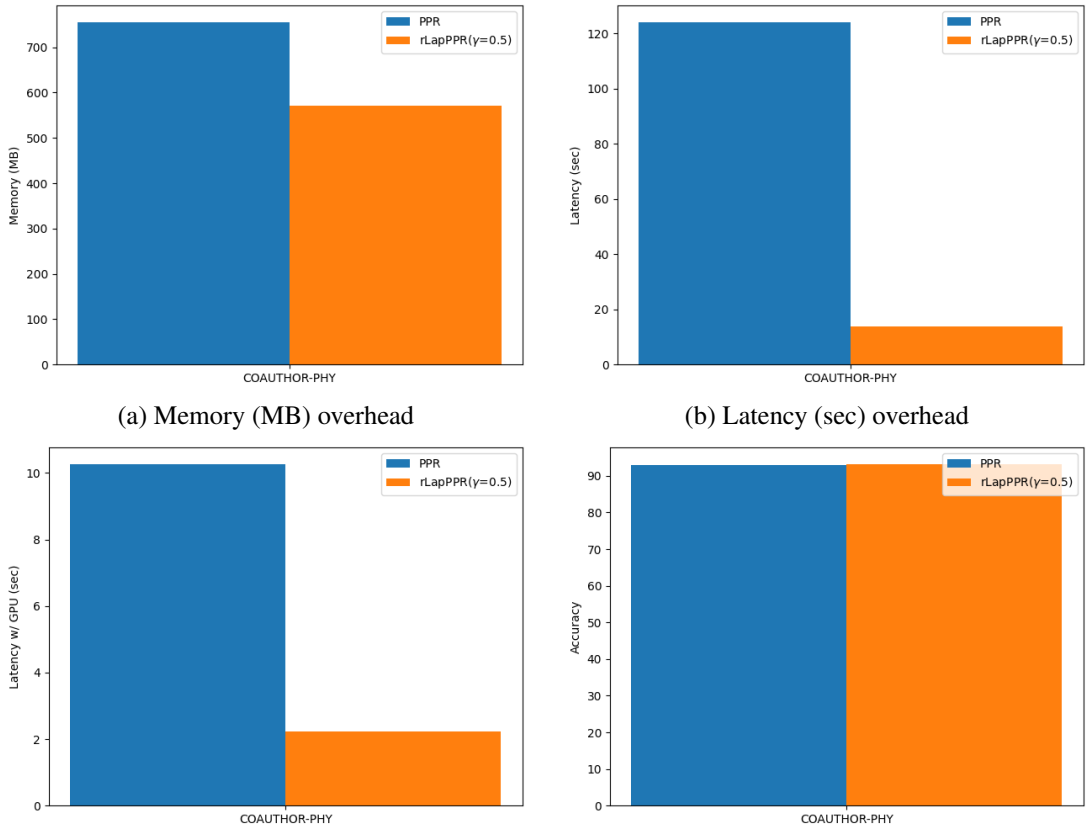
Limitations and Tradeoffs: From Algorithm 1, notice that after every outer loop iteration, the number of edges in the graph representing the current Schur complement state will be less than or equal to the number of edges at the beginning of the iteration. To quantify the impact of diffusion on these randomized Schur complements, we measured the edge counts of sub-graphs of size 8192 randomly sampled from the following graphs [7]:

- **COAUTHOR-CS:PPR**: PPRDiffusion applied on COAUTHOR-CS.
- **COAUTHOR-CS:rLapPPR**: $rLap$ on COAUTHOR-CS with $\gamma = 0.5$ followed by PPRDiffusion.
- **COAUTHOR-PHY:PPR**: PPRDiffusion applied on COAUTHOR-PHY.
- **COAUTHOR-PHY:rLapPPR**: $rLap$ on COAUTHOR-PHY with $\gamma = 0.5$ followed by PPRDiffusion.

[7] We consider the $o_v = rand, o_n = asc$ variant of $rLap$, which is the default setting.

The results in Figure 13 show that the randomly sampled sub-graph from **COAUTHOR-CS:rLapPPR** has $\approx 2.5\times$ the number of edges in **COAUTHOR-CS:PPR**. Similarly, the sub-graph from **COAUTHOR-PHY:rLapPPR** has $\approx 1.5\times$ the number of edges in **COAUTHOR-PHY:PPR**. Thus, although one can save memory and time to address the bottlenecks of PPRDiffusion, the GNN encoders would eventually consume additional memory due to the message passing operations on relatively more edges.

In such situations, if training latencies take precedence over memory constraints, then applying $rLap$ and proceeding with PPRDiffusion will lead to better training throughput. Else, based on the edge count pattern in Figure 14, the practitioner can use a lower γ value and consider the trade-offs between augmentation latencies and available memory as per their setup.



(a) Memory (MB) overhead

(b) Latency (sec) overhead

(c) Latency w/ GPU (sec) overhead

(d) Node classification accuracy

Figure 12. Computational overheads and unsupervised node classification accuracy of PPRDiffusion and $rLap$ + PPRDiffusion with $\gamma = 0.5$ on COAUTHOR-PHY dataset.

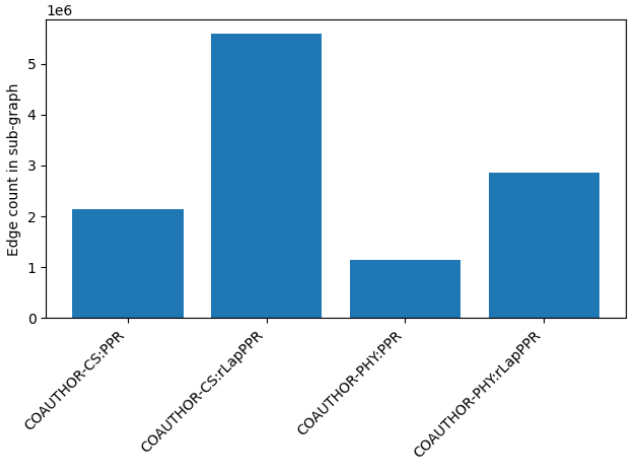
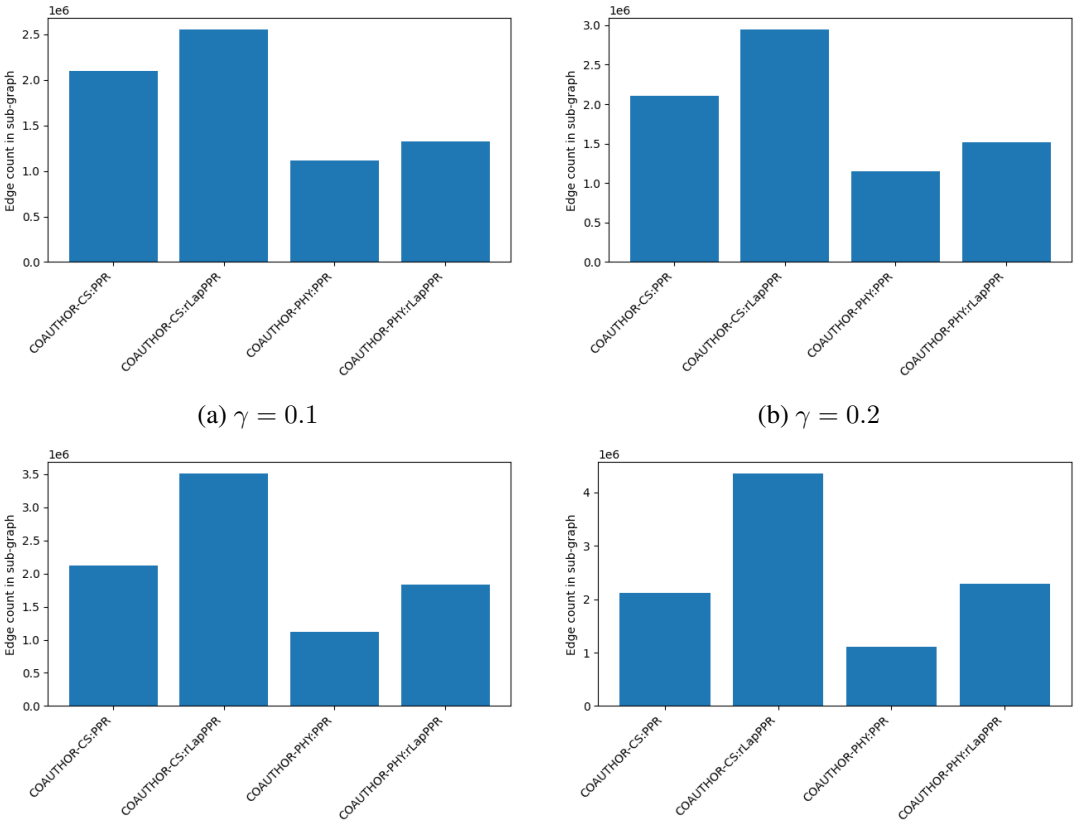


Figure 13. Plots of edge counts of randomly sampled sub-graphs of size 8192 with $\gamma = 0.5$ for $rLap$ based techniques.



(a) $\gamma = 0.1$ (b) $\gamma = 0.2$

(c) $\gamma = 0.3$ (d) $\gamma = 0.4$

Figure 14. Plots of edge counts of randomly sampled sub-graphs of size 8192 with varying γ for $rLap$ based techniques.