Context Sketching for Memory-efficient Graph Representation Learning

Kai-Lang Yao and Wu-Jun Li[§]
National Key Laboratory for Novel Software Technology
Department of Computer Science and Technology, Nanjing University, China
yaokl@smail.nju.edu.cn, liwujun@nju.edu.cn

***Abstract*—Graph representation learning (GRL) is fundamental in multi-graph applications like molecular property prediction. Graph neural networks (GNNs) have emerged as a popular method for GRL. However, existing GRL methods primarily focus on designing GNN models with enhanced expressiveness while overlooking the memory efficiency of algorithms during training. The memory inefficiency problem is caused by a *contextual constraint* imposed on node representations, which requires each node to be context-dependent on its input graph. In this paper, we propose a novel method, called context sketching (COS), for memory-efficient graph representation learning in multi-graph scenarios. We first formally define the contextual constraint based on the enclosed ∞-hop subgraphs of nodes. Subsequently, we propose to relax the original contextual constraint by requiring each node to be context-dependent on its enclosed k-hop subgraph ($k \ll \infty$) which is a contextual sketch of the enclosed ∞-hop subgraph. Lastly, we prove that COS constructs an optimal solution to a memory-related objective associated with graph coarsening. Experiments on four widely used benchmark datasets demonstrate that COS can reduce the memory footprint of baselines by a large margin with almost no accuracy loss.**

***Index Terms*—Graph Representation Learning, Graph Neural Networks, Graph Deep Learning, Memory-efficient Learning.**

I. Introduction

GRAPHS are important data structures used to represent real-world objects such as chemical molecules and proteins. By taking multiple graphs as samples, graph representation learning (GRL) aims to transform each graph into a vector representation for various machine learning tasks, such as classification, regression and clustering [1]. In this paper, GRL refers to the case in multi-graph scenarios unless otherwise stated. GRL has been widely applied in diverse domains, including predicting molecular properties for molecular graphs [2], classifying taxonomic groups for proteins based on protein-protein association networks [3] and predicting subtokens that form method names using abstract syntax trees of Python method codes [4]. GRL also shows much potential in many other areas, such as brain network analysis in cognitive science [5]. Consequently, GRL has gained much attention within the machine learning community.

Among the various methods for GRL, graph neural networks (GNNs) based methods [6]–[10] have emerged as a dominant class in recent years. These methods apply GNN models to a graph to generate node representations which are then summarized into a graph representation with a readout function. Previous studies [10] show that early GNN models are effective in generating discriminative graph representation, as they possess the expressive power equivalent to the Weisfeiler-Leman (1-WL) graph isomorphism test [11]. However, their expressiveness is also upper-bounded by the 1-WL algorithm. Consequently, recent works propose various approaches to design GNN models with enhanced expressiveness. For example, some works [12]–[15] focus on specific domains, such as molecular graphs, and propose domain-specific models that exploit domain-specific properties. Some other works aim to approach the expressive power of the k-WL ($k>1$) algorithm and propose powerful and provable models. These include direct extensions of the k-WL algorithm [16]–[18], models with invariant and equivariant layers [19]–[21], and subgraph GNN models [22]–[26]. Additionally, some works go beyond restricting message passing solely between neighboring nodes and propose graph Transformer models [27]–[30].

Although existing GNN methods, including those mentioned above, have made much progress in various tasks, most of them suffer from memory inefficiency during training. This inefficiency problem arises from imposing a *contextual constraint* on node representations, which requires each node to be context-dependent on its input graph. As a result, nodes with the same features or similar contexts across different graphs have independent memory spaces, resulting in a massive memory footprint. By taking $L{=}10$ (number of model layers), $r{=}256$ (dimension of node representations), $n{=}250$ (average number of nodes per graph) and a small $B{=}1024$ (number of graphs in a mini-batch) as an example, existing GNN methods can consume over 48GB of graphics memory during training, surpassing the capacity of most GPU card. This massive memory footprint will impose undesirable limitations on GNN methods. For example, it limits the scale of GNN models in terms of depth and width. Considering that large models have become a dominant trend in artificial intelligence research [31]–[33], this limitation is problematic, as larger GNN models typically have the potential to achieve better accuracy performance [34]–[36]. Furthermore, a massive memory footprint limits the mini-batch size during training, leading to the underutilization of computing resources and undesirable increase in wall-clock time. Unfortunately, despite these limitations, the memory inefficiency problem in GRL has received little attention from researchers.

[§]Corresponding author

In this paper, we propose a novel method, called context sketching (COS), for memory-efficient graph representation learning in multi-graph scenarios. The contributions of this paper can be outlined as follows:

- This paper is the first to study the memory inefficiency problem in GNN methods for GRL.
- We formally define the contextual constraint and propose to improve the memory efficiency of GNN methods with a novel context sketching technique. Moreover, we prove that COS constructs an optimal solution to a memory-related objective associated with graph coarsening.
- Experiments on four widely used benchmark datasets demonstrate that COS can reduce the memory footprint of baselines by a large margin with almost no accuracy loss.

II. Preliminary

A. Notation

We use a boldface uppercase letter, such as $\mathbf{A}$, to denote a matrix. We use a boldface lowercase letter, such as $\boldsymbol{a}$, to denote a vector. We use the transpose of $\boldsymbol{a}_i$ to denote the ith row of $\mathbf{A}$. A_{ij} denotes the element of the ith row and the jth column in $\mathbf{A}$. $\mathbf{A}^\dagger$ denotes the pseudo inverse of $\mathbf{A}$. $\{\!\{\cdot\}\!\}$ denotes a multiset. $\|\cdot\|_F$ denotes the Frobenius norm of a matrix. $\mathrm{Tr}(\cdot)$ denotes a trace operation defined on a square matrix.

Let $\mathcal{G}=(\mathcal{V},\mathcal{E},\mathbf{X},\mathbf{E})$ denote a graph comprising a set $\mathcal{V}$ of nodes and a set $\mathcal{E}$ of edges. $\mathbf{X}$ and $\mathbf{E}$ are the feature matrix associated with $\mathcal{V}$ and $\mathcal{E}$, respectively. Each row of $\mathbf{X}$ denotes the feature vector of a node. Each row of $\mathbf{E}$ denotes the feature vector of an edge. We use $\mathbf{A}$ to denote the adjacency matrix of a graph $\mathcal{G}$, where $A_{ij}=1$ if $(i,j)\in\mathcal{E}$, otherwise $A_{ij}=0$. We use $\mathbf{L}=\mathbf{D}-\mathbf{A}$ to denote the graph Laplacian matrix of $\mathbf{A}$, where $\mathbf{D}$ is the diagonal degree matrix of $\mathbf{A}$, i.e., $D_{ii}=\sum_j A_{ij}$. We use different subscript t to distinguish different graphs, like $\mathcal{G}_t$, $\mathcal{V}_t$, $\mathbf{X}_t$, $\mathbf{E}_t$, $\mathbf{A}_t$, $\mathbf{D}_t$ and $\mathbf{L}_t$. Since our goal is to learn a vector representation for each input graph, we assume that each input graph is connected.

B. Problem Definition

The main idea of GNN models is to perform iterative message passing among neighboring nodes, enabling each node to encode its global structural role into its node representation after multiple iterations. From a unified perspective, most GNN models can be regarded as specific instances of message-passing neural networks (MPNN) [37]. For an input graph $\mathcal{G}$, one MPNN layer can be defined as follows:

$$\boldsymbol{m}_i^{(\ell)}=\square_{j\in\mathcal{N}_i}\phi^{(\ell)}\left(\boldsymbol{x}_i^{(\ell-1)},\boldsymbol{x}_j^{(\ell-1)},\boldsymbol{e}_{ij},A_{ij}^{(\ell)}\right), \tag{1}$$

$$\boldsymbol{x}_i^{(\ell)}=\gamma^{(\ell)}\left(\boldsymbol{x}_i^{(\ell-1)},\boldsymbol{m}_i^{(\ell)}\right), \tag{2}$$

where $\phi^{(\ell)}$ and $\gamma^{(\ell)}$ are learnable functions, $\square$ is an aggregation function (typically sum, mean or max), ℓ denotes the layer number, $\boldsymbol{x}_i^{(0)}$ is the input feature vector of node i, $\boldsymbol{e}_{ij}$ is the feature vector of edge (i,j) and $\mathcal{N}_i$ denotes the set of neighbors for node i. $\mathbf{A}^{(\ell)}$ denotes the preprocessed $\mathbf{A}$ in layer ℓ like row-normalization $\mathbf{D}^{-1}\mathbf{A}$, column-normalization $\mathbf{A}\mathbf{D}^{-1}$, symmetric normalization $\mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$ or attention-based normalization as in [9]. After L layers, GNN models transform all node representations within a graph $\mathcal{G}$ to a graph representation with a readout function, which is defined as follows:

$$\boldsymbol{g}=f_R\left(\left\{\boldsymbol{x}_1^{(L)},\cdots\boldsymbol{x}_n^{(L)}\right\}\right). \tag{3}$$

Here, f_R (permutation equivariant) represents sum, mean, max or a learnable function. We use different subscript t to distinguish different graphs, like $\boldsymbol{x}_{t,i}^{(\ell)}$, $\boldsymbol{e}_{t,ij}$, $\boldsymbol{g}_t$ and $A_{t,ij}^{(\ell)}$.

In (1), (2) and (3), each node i within an input graph $\mathcal{G}_t$ has its own memory space and computation process that depend on $\mathcal{G}_t$. As a result, nodes with the same features and similar contexts across distinct graphs have independent memory spaces and computation process. That is why we state that a *contextual constraint* (a formal definition is given in Section III-A) is imposed on each node i within $\mathcal{G}_t$. If the mini-batch size (number of graphs in a mini-batch) is B, $\boldsymbol{x}_{t,i}^{(\ell)}\in\mathbb{R}^r$ and the average number of nodes per graph is n, the memory complexity of GNN methods is at least $\mathcal{O}(LBnr)$ during training. For large-scale applications with a large n, even a small B will incur a large memory footprint.

III. Context Sketching for GRL

In this section, we begin by formally defining the contextual constraint in GNN methods. Next, we introduce the memory-related objective of COS, which is associated with graph coarsening in multi-graph scenarios. After that, we present the memory-efficient solution with COS and prove that COS constructs an optimal solution for the memory-related objective. Then, we demonstrate the memory and computation efficiency of COS through complexity analysis. Finally, we discuss the relationship between our proposed COS and the most related works.

A. Formal Definition for Contextual Constraint

The contextual constraint existing in (1) and (2) guarantees a GNN model to learn a context-dependent representation for each node i with respect to its input graph $\mathcal{G}_t$, thus ensuring an independent memory space and computation process associated with $\mathcal{G}_t$. In this subsection, we provide a formal definition for the contextual constraint described above and realize it with an injective function.

We use $\mathcal{D}_\mathcal{G}=\{\mathcal{G}_1,\mathcal{G}_2,\cdots,\mathcal{G}_T\}$ to denote a multi-graph dataset. $\mathcal{G}^{(\infty,i)}=\left(\mathcal{V}^{(\infty,i)},\mathcal{E}^{(\infty,i)},\mathbf{X}^{(\infty,i)},\mathbf{E}^{(\infty,i)}\right)$ denotes the enclosed ∞-hop subgraph of node i induced from a graph $\mathcal{G}\in\mathcal{D}_\mathcal{G}$. $\mathcal{V}^{(\infty,i)}$ represents the union of ∞-hop neighbors of i within $\mathcal{G}$. $(u,v)\in\mathcal{E}^{(\infty,i)}$ iff $(u,v)\in\mathcal{E}$. Since $\mathcal{G}$ is connected, it is easy to verify that $\mathcal{G}^{(\infty,i)}=\mathcal{G}$. We refer to $\mathcal{G}^{(\infty,i)}$ as the *global context* of node i because the computation of $\boldsymbol{x}_i^{(L)}$ depends on $\mathcal{G}^{(\infty,i)}$. Let $N=\sum_t|\mathcal{V}_t|$. Then an injective

function for realizing the contextual constraint is defined as follows:

$$\pi : \left(i, \boldsymbol{x}_i, \mathcal{G}^{(\infty,i)}\right) \rightarrow \{1, \cdots, N\}, \quad i \in \mathcal{V} \wedge \mathcal{G} \in \mathcal{D}_{\mathcal{G}}, \quad (4)$$

where π maps nodes across distinct graphs to different variables because distinct graphs induce different ∞-hop subgraphs $\mathcal{G}^{(\infty,i)}$. Moreover, π can also distinguish different nodes within a graph $\mathcal{G}$ by including the node identifier i and node feature $\boldsymbol{x}_i$ as inputs. As a result, π maps each node within $\mathcal{D}_{\mathcal{G}}$ to a unique variable, thus assigning each node an independent memory space and computation process associated with its input graph. Equivalently, π maps $\mathcal{D}_{\mathcal{G}}$ to a single graph $\mathcal{G}$ as follows:

$$p = \pi\left(i, \boldsymbol{x}_{t,i}, \mathcal{G}_t^{(\infty,i)}\right), \quad i \in \mathcal{V}_t \wedge 1 \leq t \leq T, \quad (5)$$

$$q = \pi\left(j, \boldsymbol{x}_{t,j}, \mathcal{G}_t^{(\infty,j)}\right), \quad j \in \mathcal{V}_t \wedge 1 \leq t \leq T, \quad (6)$$

$$A_{pq} = A_{t,ij}, \quad \boldsymbol{e}_{pq} = \boldsymbol{e}_{t,ij}, \quad \boldsymbol{x}_p = \boldsymbol{x}_{t,i}, \quad \boldsymbol{x}_q = \boldsymbol{x}_{t,j}, \quad (7)$$

where we abuse the notations $\mathcal{G}$, $\mathbf{A}$, $\mathbf{X}$ and $\mathbf{E}$ for the equivalent transformation of $\mathcal{D}_{\mathcal{G}}$. In the end, one MPNN layer with contextual constraint for a multi-graph dataset $\mathcal{D}_{\mathcal{G}}$ can be formulated as follows:

$$\boldsymbol{m}_p^{(\ell)} = \square_{q \in \mathcal{N}_p} \phi^{(\ell)}\left(\boldsymbol{x}_p^{(\ell-1)}, \boldsymbol{x}_q^{(\ell-1)}, \boldsymbol{e}_{pq}, A_{pq}^{(\ell)}\right), \quad (8)$$

$$\boldsymbol{x}_p^{(\ell)} = \gamma^{(\ell)}\left(\boldsymbol{x}_p^{(\ell-1)}, \boldsymbol{m}_p^{(\ell)}\right), \quad (9)$$

$$\boldsymbol{g}_t = f_R\left(\left\{\boldsymbol{x}_p^{(L)} | i \rightarrow p, i \in \mathcal{V}_t\right\}\right), \quad 1 \leq t \leq T, \quad (10)$$

where $i \rightarrow p$ means π maps node i in $\mathcal{G}_t$ to node p in $\mathcal{G}$.

B. Memory-related Objective of COS

As analyzed in Section II-B, existing GNN methods have a memory complexity of $\mathcal{O}(LBnr)$. After transforming a multi-graph dataset $\mathcal{D}_{\mathcal{G}}$ to an equivalent single-graph dataset $\mathcal{G}$ by (5), (6) and (7), one possible way to reduce $\mathcal{O}(LBnr)$ is to coarsen $\mathcal{G}$ to a smaller one.

Graph coarsening [38], [39] aims to learn a smaller-tractable graph for a large graph while preserving its properties. Many works have proposed to learn such a small graph by designing various optimization problems that ensure desired properties. Suppose $\mathbf{L} \in \mathbb{R}^{N \times N}$ and $\mathbf{X} \in \mathbb{R}^{N \times u}$ are the graph Laplacian matrix and node features associated with $\mathcal{G}$, respectively. u denotes the feature dimension of nodes. Let $\tilde{\mathcal{G}} = (\tilde{\mathcal{V}}, \tilde{\mathcal{E}}, \tilde{\mathbf{X}})$ denote the coarsened graph by leaving out the edge feature. Similarly, we denote the graph Laplacian matrix associated with $\tilde{\mathcal{G}}$ by $\tilde{\mathbf{L}} \in \mathbb{R}^{M \times M}$, where M represents the number of nodes in $\tilde{\mathcal{G}}$. Let $\mathbf{P} \in \mathbb{R}_+^{M \times N}$ be the coarsening matrix which realizes the linear mapping $\tilde{\pi}: \mathcal{V} \rightarrow \tilde{\mathcal{V}}$ such that $\tilde{\mathbf{X}} = \mathbf{P}\mathbf{X}$. A non-zero P_{ij} indicates mapping the node j within $\mathcal{G}$ to the node i within $\tilde{\mathcal{G}}$. Each column of $\mathbf{P}$ has only one non-zero entry, indicating that each node within $\mathcal{G}$ can only be mapped to one node within $\tilde{\mathcal{G}}$. Then, the original graph $\mathcal{G}$ and its coarsened graph $\tilde{\mathcal{G}}$ satisfy the following properties [38], [39]:

$$\tilde{\mathbf{L}} = \mathbf{C}^\top \mathbf{L} \mathbf{C}, \quad \tilde{\mathbf{X}} = \mathbf{P}\mathbf{X}, \quad (11)$$

where $\mathbf{C} \in \mathbb{R}_+^{N \times M}$ is the pseudo inverse of $\mathbf{P}$ and is known as the loading matrix. Moreover, $\mathbf{C}$ belongs to the following set:

$$\mathcal{C} = \big\{\mathbf{C} \in \mathbb{R}_+^{N \times M} | \mathbf{C}_{*i}^\top \mathbf{C}_{*j} = 0 \; \forall \, i \neq j, \; \mathbf{C}_{*i}^\top \mathbf{C}_{*i} = |\tilde{\pi}^{-1}(i)|, \; \|\mathbf{C}_{*i}\|_0 \leq 1, \; \|\mathbf{C}_{i*}\|_0 = 1\big\} \quad (12)$$

In the end, graph coarsening is to learn a $\mathbf{P}$ or $\mathbf{C}$ such that $\tilde{\mathbf{L}}$ can preserve some desired properties of $\mathbf{L}$.

As stated in [39], ϵ-similarity is one desired property often required to be preserved by $\tilde{\mathbf{L}}$. It is because node features between connected nodes within a graph are usually imposed with a smooth assumption in graph machine learning [40], [41]. As a result, ϵ-similarity quantifies such a smoothness variation from $\mathbf{L}$ to $\tilde{\mathbf{L}}$. A formal definition of ϵ-similarity is presented in Definition 1.

Definition 1. *ϵ-similarity [39]. A coarsened graph $\tilde{\mathcal{G}} = (\tilde{\mathcal{V}}, \tilde{\mathcal{E}}, \tilde{\mathbf{X}})$ is ϵ similar to its original graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{X})$ if there exists an $\epsilon \geq 0$ such that:*

$$(1 - \epsilon)\|\mathbf{X}\|_{\mathbf{L}} \leq \|\tilde{\mathbf{X}}\|_{\tilde{\mathbf{L}}} \leq (1 + \epsilon)\|\mathbf{X}\|_{\mathbf{L}}, \quad (13)$$

where $\|\mathbf{X}\|_{\mathbf{L}} = \sqrt{\mathrm{Tr}(\mathbf{X}^\top \mathbf{L} \mathbf{X})}$ and $\|\tilde{\mathbf{X}}\|_{\tilde{\mathbf{L}}} = \sqrt{\mathrm{Tr}(\tilde{\mathbf{X}}^\top \tilde{\mathbf{L}} \tilde{\mathbf{X}})}$.

Here, $\mathrm{Tr}(\mathbf{X}^\top \mathbf{L} \mathbf{X})$ quantifies the smoothness of node features between connected nodes and it is also known as Dirichlet energy. Definition 1 indicates that the smaller ϵ is, the more similar $\tilde{\mathbf{L}}$ is to $\mathbf{L}$ regarding the smoothness property. To achieve a small ϵ when learning $\mathbf{C}$, one suitable option is to optimize the following objective function:

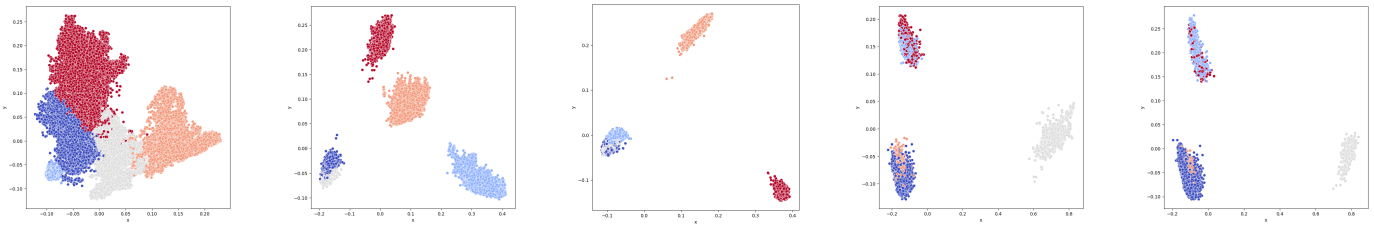
$$\begin{aligned} \min_{\mathbf{C}} \quad & (\|\mathbf{X}\|_{\mathbf{L}} - \|\tilde{\mathbf{X}}\|_{\tilde{\mathbf{L}}})^2 \\ \text{s.t.} \quad & \mathbf{C} \in \mathcal{C}, \quad \tilde{\mathbf{L}} = \mathbf{C}^\top \mathbf{L} \mathbf{C}, \quad \tilde{\mathbf{X}} = \mathbf{P}\mathbf{X}, \quad \mathbf{P} = \mathbf{C}^\dagger \end{aligned} \quad (14)$$

Many works [38], [39], [42], [43] have proposed approximate solutions for (14) or proposed to optimize a surrogate optimization problem. However, existing methods for graph coarsening are typically designed for single-graph datasets and hence are suboptimal for multi-graph datasets in this paper. In particular, multi-graph datasets have repeated structural patterns across different graphs [14], such as nodes with the same node features or neighbor patterns, which can be utilized to facilitate finding optimal solutions for (14). Moreover, existing methods for graph coarsening all involve a time-consuming optimization process before learning graph representation with GNN models, which is undesirable.

C. Memory-efficient Solution with COS

Instead of solving (14) through an optimization process, COS directly constructs an optimal solution to (14), thereby introducing minimal computation overhead before proceeding to learn graph representation with GNN models. COS achieves this by exploiting repeated structural patterns in multi-graph datasets. We present the details of COS in the rest of this subsection.

As discussed in Section III-A, depending on the global context to generate a node representation leads to the memory inefficiency problem in GNN methods. Here, we visualize



(a) k=1. (b) k=2. (c) k=3. (d) k=4. (e) k=5.

Fig. 1: Visualization of node representations in multi-graph scenarios. We train a GNN model on dataset ogbg-pcba with existing GNN methods and generate node representations for all graphs using the learned model. To visualize these node representations, we project them onto a 2-dimensional vector space using PCA. In the figures, each solid circle represents a node in graphs. In each figure, nodes with the same color indicate that they share the same node features and enclosed k-hop subgraphs. We find that nodes with similar contexts (the same enclosed k-hop subgraphs) clump together in clusters.

the node vector representation generated by existing GNN methods in Fig. 1. We find that nodes with the same node features and enclosed k-hop ($k \ll \infty$) subgraphs clump together in clusters. Inspired by this phenomenon, we propose to relax the original hard contextual constraint by replacing the global context with its sketch as the input for π in (4). More specifically, we replace a node's enclosed ∞-hop subgraph with its enclosed k-hop subgraph as the input for π. After transforming a multi-graph dataset $\mathcal{D}_{\mathcal{G}}$ to an equivalent single-graph dataset $\mathcal{G}$ by (5), (6) and (7), we redefine π in (4) as follows:

$$\pi : \left(\boldsymbol{x}_i, \mathcal{G}^{(k,i)}\right) \to \{1, \cdots, M\}, \quad i \in \mathcal{V}, \tag{15}$$

where $\mathcal{G}^{(k,i)}$ represents the enclosed k-hop subgraph of node i induced from $\mathcal{G}$. M denotes the number of nodes with a unique $\left(\boldsymbol{x}_i, \mathcal{G}^{(k,i)}\right)$. It is easy to verify that (15) imposes a soft contextual constraint on each node by mapping nodes with the same features and enclosed k-hop subgraphs to the same variables, thus enabling them to share memory spaces during training. A visual illustration of COS is presented in Fig. 2. Given π in (15), we need to determine the coarsening matrix $\mathbf{P} \in \mathbb{R}_+^{M \times N}$ or the loading matrix $\mathbf{C} \in \mathbb{R}_+^{N \times M}$ to obtain a small coarsened graph $\tilde{\mathcal{G}}$. More specifically, we define $\mathbf{P}$ and obtain its pseudo inverse $\mathbf{C}$ as follows:

$$\pi^{-1}(p) = \left\{ i | \pi(\boldsymbol{x}_i, \mathcal{G}^{(k,i)}) = p, \, i \in \mathcal{V} \right\}, \quad p \in \mathcal{V} \tag{16}$$

$$P_{pi} = \begin{cases} \frac{1}{|\pi^{-1}(p)|}, & \text{if } i \in \pi^{-1}(p) \\ 0, & \text{else} \end{cases}, \tag{17}$$

$$C_{ip} = \begin{cases} 1, & \text{if } i \in \pi^{-1}(p) \\ 0, & \text{else} \end{cases}, \tag{18}$$

where $\mathbf{C}$ and $\mathbf{P}$ satisfy that $\mathbf{PC}{=}\mathbf{I}$ and $\mathbf{C} \in \mathcal{C}$. Then, we can obtain $\tilde{\mathbf{A}} \in \mathbb{R}^{M \times M}$, $\tilde{\mathbf{X}} \in \mathbb{R}^{M \times u}$ and $\tilde{\mathbf{E}}$ associated with the coarsened graph $\tilde{\mathcal{G}}$ as follows:

$$\tilde{\mathbf{A}} = \mathbf{C}^\top \mathbf{A} \mathbf{C}, \quad \tilde{\mathbf{X}} = \mathbf{P}\mathbf{X}, \tag{19}$$

$$\mathcal{E}_{pq} = \{(i,j) | \pi(p){=}i \wedge \pi(q){=}j \wedge (i,j) {\in} \mathcal{E}\}$$

$$\tilde{\boldsymbol{e}}_{pq} = \sum_{(i,j) \in \mathcal{E}_{pq}} \frac{\boldsymbol{e}_{ij}}{|\mathcal{E}_{pq}|}. \tag{20}$$

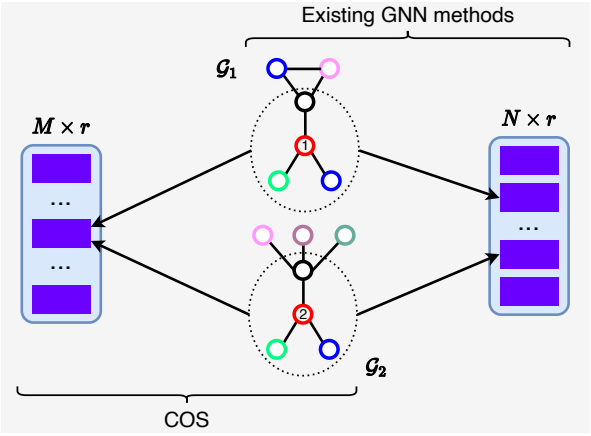


Fig. 2: A visual illustration of COS. In the figure, each circle represents a node, and nodes with different colors indicate nodes with different features. Nodes with the same features are shown in the same color. $\mathcal{G}_1$ and $\mathcal{G}_2$ are two input graphs. The blue rectangle represents a node representation of dimension r. N represents the total number of nodes in $\mathcal{G}_1$ and $\mathcal{G}_2$, and M ($< N$) represents the total number of distinct nodes after applying COS. We highlight the difference between COS and existing GNN methods using two red circles marked with numbers 1 and 2. The enclosed 1-hop subgraphs of node 1 and node 2 are marked with dashed circles. We can see that node 1 and node 2 have the same features and enclosed 1-hop subgraphs. Existing GNN methods map node 1 and node 2 to different vector representations. In contrast, COS maps node 1 and node 2 to the same vector representation.

It is easy to verify that the graph Laplacian matrix $\tilde{\mathbf{L}}$ associated with $\tilde{\mathbf{A}}$ satisfies $\tilde{\mathbf{L}}{=}\mathbf{C}^\top \mathbf{L} \mathbf{C}$. As a result, $\mathbf{C}$, $\mathbf{P}$, $\tilde{\mathbf{X}}$ and $\tilde{\mathbf{L}}$ constructed above satisfy the constraints in (14). It is worth noting that $\tilde{\mathbf{X}}$ satisfies $\tilde{\mathbf{X}}_{p*}{=}\mathbf{X}_{i*} \, \forall \, i{=}\pi^{-1}(p)$. It is because nodes with the same features and enclosed k-hop subgraphs are mapped into the same nodes within the coarsened graph. We will show in Section III-D that such a property within $\tilde{\mathbf{X}}$ is the key ensuring an optimal solution to (14). Similarly, $\tilde{\mathbf{E}}$ satisfies that $\tilde{\mathbf{e}}_{pq}{=}\tilde{\mathbf{e}}_{ij} \, \forall \, \pi(p){=}i \wedge \pi(q){=}j \wedge (i,j){\in}\mathcal{E}$.

Before proceeding to learn graph representation with GNN models, we need to realize π defined in (15). The key is to distinguish distinct $\mathcal{G}^{(k,i)}$ and then map them to integers, which equals to a graph isomorphism test problem. Therefore, π can be approximated with the 1-WL or color-refinement algorithm, which is summarized in Algorithm 1. $hash(x)$=$hash(y)$ iff x=y. The outputs of $hash(\cdot)$ are forced to be integers. After obtaining $\tilde{\mathcal{G}}$, we feed $\tilde{\mathcal{G}}$ to a GNN model and learn graph representation $\{\mathbf{g}_t\}$ as in (8), (9) and (10).

We denote the training set by $\mathcal{D}$=$\{(\mathcal{G}_t, \boldsymbol{y}_t)\}_{t=1}^{T}$. We use $\mathcal{W}$ to represent the set of all learnable parameters in COS. By taking multi-class classification as an example, the optimization problem of applying COS to learn graph representation is as follows:

$$\min_{\mathcal{W}} \frac{1}{T}\sum_{t}\sum_{c} - Y_{tc}\log \hat{Y}_{tc} + \frac{\lambda}{2}\sum_{\mathbf{W}\in\mathcal{W}} \|\mathbf{W}\|_F^2, \tag{21}$$

where $\hat{\boldsymbol{y}}_t$ denotes the prediction of $\mathcal{G}_t$ by performing a learnable projection on $\boldsymbol{g}_t$. λ is a hyper-parameter. The entire learning algorithm of COS is summarized in Algorithm 2.

D. Theoretical Analysis

In this subsection, we prove that COS provides a constructive optimal solution to (14). We summarize the theoretical analysis in Theorem 1.

Theorem 1. *Suppose a multi-graph dataset $\mathcal{D}_{\mathcal{G}} = \{\mathcal{G}_1, \mathcal{G}_2, \cdots, \mathcal{G}_T\}$ are transformed to an equivalent single-graph dataset $\mathcal{G}$ by (5), (6) and (7). Suppose COS defines $\mathbf{C}$, $\mathbf{P}$, $\tilde{\mathbf{A}}$ and $\tilde{\mathbf{X}}$ based on $\mathcal{G}$ by (16), (17), (18) and (19). Then, $\mathbf{C}$ defined by COS is an optimal solution to (14).*

Proof. Firstly, we prove that $\mathbf{C}\tilde{\mathbf{X}}$=$\mathbf{C}\mathbf{P}\mathbf{X}$=$\mathbf{X}$. For a given p, we have $\tilde{\mathbf{X}}_{p*}$=$\mathbf{X}_{i*}\ \forall i$=$\pi^{-1}(p)$ because nodes mapped to node p within $\tilde{\mathcal{G}}$ have the same features. For a given i and p=$\pi(i)$, we have $\mathbf{C}_{i*}\tilde{\mathbf{X}}$=$C_{ip}\tilde{\mathbf{X}}_{p*}$=$\mathbf{X}_{i*}$ because π is an injective function and C_{ip}=1. Therefore, we have $\mathbf{C}\tilde{\mathbf{X}}$=$\mathbf{C}\mathbf{P}\mathbf{X}$=$\mathbf{X}$. Secondly, we prove that $\tilde{\mathbf{X}}^\top\tilde{\mathbf{L}}\tilde{\mathbf{X}} = \mathbf{X}^\top\mathbf{L}\mathbf{X}$. According to $\tilde{\mathbf{L}}$=$\mathbf{C}^\top\mathbf{L}\mathbf{C}$, we have:

$$\begin{aligned}\tilde{\mathbf{X}}^\top\tilde{\mathbf{L}}\tilde{\mathbf{X}} &= \tilde{\mathbf{X}}^\top(\mathbf{C}^\top\mathbf{L}\mathbf{C})\tilde{\mathbf{X}}\\ &= (\mathbf{C}\tilde{\mathbf{X}})^\top\mathbf{L}(\mathbf{C}\tilde{\mathbf{X}})\\ &= \mathbf{X}^\top\mathbf{L}\mathbf{X}.\end{aligned}$$

Lastly, we have $\|\tilde{\mathbf{X}}\|_{\tilde{\mathbf{L}}}$=$\sqrt{\mathrm{Tr}(\tilde{\mathbf{X}}^\top\tilde{\mathbf{L}}\tilde{\mathbf{X}})}$=$\sqrt{\mathrm{Tr}(\mathbf{X}^\top\mathbf{L}\mathbf{X})}$=$\|\mathbf{X}\|_{\mathbf{L}}$. Since the minimum value of (14) will not be less than zero, we can claim that $\mathbf{C}$ defined by COS is an optimal solution to (14). □

The above theorem shows that exploiting repeated structural patterns across graphs can facilitate finding an optimal solution to (14) without an additional optimization process before proceeding to learn graph representation with GNN models.

E. Complexity Analysis

Since a graph $\mathcal{G}$ of N nodes is coarsened to a graph $\tilde{\mathcal{G}}$ of M nodes, COS has a memory complexity of $\mathcal{O}\left(LBnr\cdot(M/N)\right)$. As a result, COS is more memory-efficient than existing GNN methods because N>M. In real multi-graph datasets (see Fig. 3), it is common that there exist many repeated structural patterns across graphs [14], like nodes with the same node features or neighbor patterns. As we will show in experiments, such a property enables COS to achieve a value of 50% for M/N on some datasets during training, indicating that COS can reduce memory footprint by 50% compared to baselines.

From Algorithm 2, we find that COS introduces additional computation overheads in operations for performing Algorithm 1, operations for transforming a multi-graph dataset to an equivalent single-graph dataset, and operations for coarsening compared to existing GNN methods. However, these computation overheads are negligible in comparison to that for training. Firstly, when given one dataset, we only perform Algorithm 1 once and store π for reuse when tuning hyper-parameters. Moreover, Algorithm 1 traverses a multi-graph dataset only

Algorithm 1 Sketching Algorithm of COS

Require: $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{X}, \mathbf{E})$, k.

$c_i^{(0)} \leftarrow hash\left(\boldsymbol{x}_i\right), \quad \forall i \in \mathcal{V}$;
$c_{ij} \leftarrow hash\left(\boldsymbol{e}_{ij}\right), \quad \forall (i,j) \in \mathcal{E}$;
for $\ell = 1 : k$ **do**
  $c_i^{(\ell)} \leftarrow hash(c_i^{(\ell-1)}, \{\!\{(c_i^{(\ell-1)}, c_{ij}) | j \in \mathcal{N}_i\}\!\}), \ \forall i \in \mathcal{V}$;
end for
$M = |\{c_i^{(k)}\}|$;
$c_i \leftarrow hash(c_i^{(k)}) \wedge c_i \in \{1, 2, \cdots, M\}, \ \forall i \in \mathcal{V}$;
$\pi : (\boldsymbol{x}_i, \mathcal{G}^{(k,i)}) \rightarrow c_i, \quad i \in \mathcal{V}$;
return π

Algorithm 2 Learning Algorithm of COS

Require: $\mathcal{D}_{\mathcal{G}}$: a multi-graph dataset, k: a hyper-parameter for context sketching, B: mini-batch size, E: maximum number of epoches.

Ensure: A GNN model with parameters $\mathcal{W}$.

Preprocessing Step:
Obtain $\mathcal{G}$ by transforming $\mathcal{D}_{\mathcal{G}}$ to an equivalent single-graph dataset with (5), (6) and (7);
Obtain π defined in (15) with Algorithm 1; /*We only need to perform the above preprocessing step for once during training and store π for reuse.*/
Learning Step:
for $e = 1 : E$ **do**
  for $s = 1 : T/B$ **do**
    Sample B graphs from $\mathcal{D}_{\mathcal{G}}$ and transform them to an equivalent single-graph $\mathcal{G}$ as in (5), (6) and (7);
    Obtain $\mathbf{C}$ and $\mathbf{P}$ by (16), (17) and (18);
    Obtain $\tilde{\mathcal{G}}$ by coarsening $\mathcal{G}$ with $\mathbf{C}$ and $\mathbf{P}$ as in (19) and (20);
    Obtain $\{\mathbf{g}_t\}$ of B graphs by applying (8), (9) and (10) on $\tilde{\mathcal{G}}$;
    Update the model parameters $\mathcal{W}$ by optimizing the mini-batch version of (21);
  end for
end for

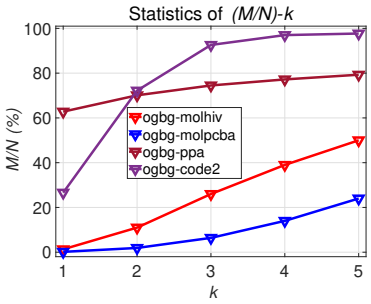


Fig. 3: Statistics of repeated structural patterns on datasets ogbg-molhiv, ogbg-molpcba, ogbg-ppa and ogbg-code2. N represents the total number of nodes in a multi-graph dataset, and M represents the number of nodes with unique enclosed k-hop subgraphs in a multi-graph dataset. A smaller M/N indicates more repetitive structural patterns in multi-graph datasets.

k (typically less than 5) times and $hash(\cdot)$ operations are more computation-efficient than the operations in (8), (9) and (10). Secondly, we can transform $\mathcal{D}_{\mathcal{G}}$ to $\mathcal{G}$ by reindexing nodes within $\mathcal{D}_{\mathcal{G}}$, which introduces almost no computation overhead. Lastly, operations for coarsening as in (16) - (20) are more computation-efficient than the operations in (8), (9) and (10).

F. Comparison to Related Work

This subsection discusses the relationship between our proposed COS and the most related works, including subgraph GNNs, graph pooling and graph coarsening.

a) Subgraph GNNs: Various subgraph GNNs [22]–[26] have been proposed to define a node representation as a function of its rooted subgraphs. While COS also involves operations on subgraphs, there are notable differences between COS and subgraph GNNs. Firstly, COS imposes a soft contextual constraint on node representations, whereas subgraph GNNs still enforce a hard contextual constraint. Secondly, subgraph GNNs impose an additional hard contextual constraint on nodes within a subgraph, leading to worse memory efficiency.

b) Graph Pooling: Graph pooling [8], [44]–[49] first applies a GNN model to learn node representations and then hierarchically coarsens a graph to generate its graph representation. Therefore, graph pooling is designed as a readout function, which can also be applied to COS as an alternative $f_R(\cdot)$ in (10). As a result, it still suffers from memory inefficiency when learning node representations with a GNN model. Moreover, graph pooling studies how to coarsen a single graph, while COS studies how to coarsen multiple graphs by exploiting unique properties in multi-graph datasets.

c) Graph Coarsening: Graph coarsening [38], [39], [42], [43], a popular type of graph reduction, contracts disjoint sets of nodes to super nodes within a coarsened graph, thus transforming a large graph into a small one. However, existing graph coarsening methods are tailored for single-graph datasets and leave unique properties in multi-graph datasets underutilized, resulting in suboptimal solutions to the corresponding optimization problem. By contrast, COS exploits repeated structural patterns across multiple graphs to achieve optimal solutions. Moreover, existing graph coarsening methods involve a time-consuming optimization process before proceeding to learn graph representation with GNN models, while COS directly constructs an optimal solution without an additional optimization process for graph coarsening.

IV. Experiment

In this section, we evaluate COS on four benchmark datasets. We implement all methods with Pytorch [50] and Pytorch-Geometric [51]. We run all experiments on an NVIDIA TRX A6000 GPU server with 48GB of graphics memory.[1]

A. Datasets

We evaluate the performance of COS on four diverse graph classification datasets, including ogbg-molhiv, ogbg-molpcba, ogbg-ppa and ogbg-code2[2]. For datasets ogbg-molhiv and ogbg-molpcba, each graph represents a molecule in which nodes represent atoms and edges denote chemical bonds. These two datasets are extracted from MOLECULENET [2] and their task is to predict molecular properties for each graph. For dataset ogbg-ppa, each graph is a protein-protein association network in which nodes represent proteins and edges indicate biologically meaningful associations between proteins. This dataset is extracted from the protein-protein association networks of various species, covering a wide range of taxonomic groups. Its task is to predict the taxonomic groups that each graph originates from. For dataset ogbg-code2, each graph is the abstract syntax trees (ASTs) of a Python method definition in which nodes represent AST nodes and edges represent AST edges. This dataset is extracted from different repositories on GitHub and its task is to predict the sub-tokens forming the Python method name. TABLE III at the end of main text presents detailed statistics for all the datasets. Following existing GNN methods, we adopt the evaluation metric proposed by [52] for our experiments.

B. Baselines and Settings

a) Baselines: We apply COS to GIN [10], SAGE [7], GCN [6] and GAT [9], which are widely used representative GNN methods in GRL and serve as building blocks for many other GNN variants. It is worth noting that COS can also be applied to other GNN methods but we focus on the most representative ones for illustrative purpose. Since the memory inefficiency problem has yet to be studied in existing GNN methods, we introduce a naive baseline to demonstrate the effectiveness of COS further. More specifically, we realize π in (15) with a random mapping strategy and refer to this baseline as RANDOM in our experiments. Like COS, RANDOM does not involve a time-consuming optimization process for graph coarsening before proceeding to learn graph representation.

[1]Source code is released at https://github.com/yaokl-nju/COS.

[2]Publicly available at https://ogb.stanford.edu/docs/graphprop/.

TABLE I: Results when L=5 (number of model layers). 'MemR' represents the memory ratio between COS and baseline GNN methods during training. 'COS ($\triangle\%$)' indicates that COS achieves an accuracy loss of less than $\triangle\%$ compared to baseline GNN methods. In particular, when $\triangle$=0, COS achieves almost no accuracy loss. $\triangle$ of different values are achieved by choosing different k. RANDOM is a naive baseline that realizes π in (15) with a random mapping strategy. When achieving the same 'MemR', methods between COS and RANDOM that achieve the highest accuracy are marked in **boldface**.

(a) Apply COS to GIN.

Methods	ogbg-molhiv		ogbg-molpcba		ogbg-ppa		ogbg-code2	
	ROC-AUC (%) ↑	MemR ↓	AP (%) ↑	MemR ↓	Accuracy (%) ↑	MemR ↓	F1 score (%) ↑	MemR ↓
GIN	79.28 ± 0.76	100%	28.89 ± 0.26	100%	72.39 ± 0.33	100%	17.65 ± 0.19	100%
RANDOM	75.51 ± 2.40	61%	22.39 ± 0.58	74%	70.63 ± 0.28	88%	16.04 ± 0.13	83%
COS (0%)	**79.20 ± 0.34**	61%	**28.82 ± 0.21**	74%	**72.13 ± 0.43**	88%	**17.56 ± 0.11**	83%
RANDOM	69.21 ± 2.63	52%	18.71 ± 0.43	62%	68.77 ± 0.23	77%	13.35 ± 0.20	63%
COS (1%)	**78.41 ± 0.67**	52%	**28.09 ± 0.35**	62%	**71.30 ± 0.30**	77%	**16.71 ± 0.17**	63%

(b) Apply COS to SAGE.

Methods	ogbg-molhiv		ogbg-molpcba		ogbg-ppa		ogbg-code2	
	ROC-AUC (%) ↑	MemR ↓	AP (%) ↑	MemR ↓	Accuracy (%) ↑	MemR ↓	F1 score (%) ↑	MemR ↓
SAGE	79.73 ± 1.39	100%	28.73 ± 0.29	100%	72.40 ± 0.12	100%	17.67 ± 0.19	100%
RANDOM	73.65 ± 0.79	61%	21.37 ± 0.21	74%	70.41 ± 0.23	86%	15.68 ± 0.29	83%
COS (0%)	**79.49 ± 0.83**	61%	**28.67 ± 0.32**	74%	**72.56 ± 0.31**	86%	**17.55 ± 0.25**	83%
RANDOM	69.67 ± 1.30	54%	17.34 ± 0.36	62%	67.90 ± 0.04	77%	13.34 ± 0.17	63%
COS (1%)	**78.66 ± 0.72**	54%	**27.99 ± 0.23**	62%	**71.26 ± 0.34**	77%	**16.73 ± 0.15**	63%

(c) Apply COS to GCN.

Methods	ogbg-molhiv		ogbg-molpcba		ogbg-ppa		ogbg-code2	
	ROC-AUC (%) ↑	MemR ↓	AP (%) ↑	MemR ↓	Accuracy (%) ↑	MemR ↓	F1 score (%) ↑	MemR ↓
GCN	79.28 ± 0.35	100%	28.94 ± 0.29	100%	72.70 ± 0.43	100%	17.62 ± 0.11	100%
RANDOM	74.27 ± 1.11	61%	21.47 ± 0.23	74%	70.40 ± 0.32	86%	15.63 ± 0.13	83%
COS (0%)	**79.37 ± 0.59**	61%	**28.82 ± 0.28**	74%	**72.51 ± 0.18**	86%	**17.52 ± 0.12**	83%
RANDOM	68.60 ± 1.21	52%	18.03 ± 0.28	62%	67.31 ± 0.17	77%	12.86 ± 0.19	63%
COS (1%)	**78.41 ± 1.19**	52%	**28.07 ± 0.20**	62%	**71.59 ± 0.41**	77%	**16.73 ± 0.16**	63%

(d) Apply COS to GAT.

Methods	ogbg-molhiv		ogbg-molpcba		ogbg-ppa		ogbg-code2	
	ROC-AUC (%) ↑	MemR ↓	AP (%) ↑	MemR ↓	Accuracy (%) ↑	MemR ↓	F1 score (%) ↑	MemR ↓
GAT	79.79 ± 0.95	100%	27.98 ± 0.30	100%	70.98 ± 1.45	100%	17.43 ± 0.26	100%
RANDOM	77.12 ± 0.54	71%	22.39 ± 0.22	74%	71.19 ± 5.05	86%	15.91 ± 0.08	83%
COS (0%)	**79.79 ± 0.88**	71%	**27.90 ± 0.27**	74%	**71.65 ± 0.71**	86%	**17.45 ± 0.08**	83%
RANDOM	72.00 ± 2.09	55%	18.82 ± 0.09	58%	69.34 ± 3.48	77%	13.22 ± 0.26	63%
COS (1%)	**78.68 ± 1.14**	55%	**27.43 ± 0.21**	58%	**70.81 ± 2.58**	77%	**16.38 ± 0.07**	63%

b) Hyper-parameter Settings: The hyper-parameters include L (layer number), r (dimension of node representations), B (mini-batch size), λ (coefficient of regularization on parameters), η (learning rate), E (maximum number of epoches), ρ (dropout probability). On all datasets, $L \in \{5, 10\}$, r=256 and B=512. We set λ=10^{-5}, η=0.002, E=100 and ρ=0.45 on dataset ogbg-molhiv. We set λ=0, η=0.01, E=100 and ρ=0.2 on dataset ogbg-molpcba. We set λ=0, η=0.02, E=100 and ρ=0.45 on dataset ogbg-ppa. We set λ=0, η=0.002, E=50 and ρ=0.40 on dataset ogbg-code2. Please note that values of all hyper-parameters are selected according to the performance of GIN on the validation set. We use Adam [53] optimizer for model optimization. We run each setting 10 times and report mean values with standard deviations. The detailed settings for reproducing all experimental results are provided in our source codes.

C. Results

The comparison in test accuracy is presented in TABLE I (L=5) and TABLE II (L=10), and test accuracy-epoch curves

TABLE II: Results when L=10 (number of model layers). 'COS ($\triangle\%$)' and 'MemR' are defined in TABLE I.

(a) Apply COS to GIN.

Methods	ogbg-molhiv		ogbg-molpcba		ogbg-ppa		ogbg-code2	
	ROC-AUC (%) ↑	MemR ↓	AP (%) ↑	MemR ↓	Accuracy (%) ↑	MemR ↓	F1 score (%) ↑	MemR ↓
GIN	79.37 ± 1.25	100%	29.88 ± 0.28	100%	73.04 ± 0.23	100%	18.02 ± 0.14	100%
RANDOM	75.01 ± 0.64	76%	22.10 ± 0.20	72%	71.86 ± 0.24	88%	15.67 ± 0.17	83%
COS (0%)	**79.37 ± 0.90**	76%	**29.86 ± 0.22**	72%	**72.77 ± 0.43**	88%	**18.07 ± 0.12**	83%
RANDOM	73.01 ± 0.74	65%	15.41 ± 0.35	55%	70.13 ± 0.23	77%	14.04 ± 0.23	63%
COS (1%)	**78.43 ± 1.11**	65%	**28.89 ± 0.36**	55%	**71.82 ± 0.25**	77%	**17.29 ± 0.10**	63%

(b) Apply COS to SAGE.

Methods	ogbg-molhiv		ogbg-molpcba		ogbg-ppa		ogbg-code2	
	ROC-AUC (%) ↑	MemR ↓	AP (%) ↑	MemR ↓	Accuracy (%) ↑	MemR ↓	F1 score (%) ↑	MemR ↓
SAGE	79.96 ± 0.64	100%	29.59 ± 0.21	100%	73.19 ± 0.23	100%	18.11 ± 0.21	100%
RANDOM	75.00 ± 0.56	76%	21.35 ± 0.18	72%	72.50 ± 0.22	86%	14.77 ± 0.10	83%
COS (0%)	**79.87 ± 0.65**	76%	**29.65 ± 0.25**	72%	**73.09 ± 0.39**	86%	**18.18 ± 0.12**	83%
RANDOM	72.97 ± 1.43	65%	15.05 ± 0.59	55%	**72.49 ± 0.28**	77%	13.40 ± 0.22	63%
COS (1%)	**78.90 ± 1.41**	65%	**28.70 ± 0.27**	55%	72.34 ± 0.19	77%	**17.28 ± 0.09**	63%

(c) Apply COS to GCN.

Methods	ogbg-molhiv		ogbg-molpcba		ogbg-ppa		ogbg-code2	
	ROC-AUC (%) ↑	MemR ↓	AP (%) ↑	MemR ↓	Accuracy (%) ↑	MemR ↓	F1 score (%) ↑	MemR ↓
GCN	79.16 ± 1.05	100%	29.88 ± 0.28	100%	73.47 ± 0.27	100%	18.18 ± 0.12	100%
RANDOM	75.79 ± 0.54	76%	22.11 ± 0.36	72%	71.88 ± 0.56	86%	15.06 ± 0.20	83%
COS (0%)	**79.50 ± 1.16**	76%	**29.88 ± 0.14**	72%	**73.21 ± 0.25**	86%	**18.24 ± 0.12**	83%
RANDOM	72.58 ± 0.77	61%	15.74 ± 0.08	55%	69.65 ± 0.32	77%	13.30 ± 0.19	63%
COS (1%)	**78.54 ± 0.75**	61%	**29.00 ± 0.16**	55%	**72.34 ± 0.24**	77%	**17.36 ± 0.16**	63%

(d) Apply COS to GAT.

Methods	ogbg-molhiv		ogbg-molpcba		ogbg-ppa		ogbg-code2	
	ROC-AUC (%) ↑	MemR ↓	AP (%) ↑	MemR ↓	Accuracy (%) ↑	MemR ↓	F1 score (%) ↑	MemR ↓
GAT	79.28 ± 1.11	100%	29.25 ± 0.27	100%	68.68 ± 2.79	100%	18.25 ± 0.13	100%
RANDOM	75.46 ± 0.40	79%	23.43 ± 0.19	72%	**69.68 ± 1.32**	86%	15.31 ± 0.11	83%
COS (0%)	**79.59 ± 0.76**	79%	**29.36 ± 0.28**	72%	68.62 ± 3.20	86%	**18.47 ± 0.16**	83%
RANDOM	74.95 ± 0.15	69%	10.61 ± 0.70	50%	65.11 ± 1.40	77%	13.82 ± 0.20	63%
COS (1%)	**78.48 ± 0.91**	69%	**27.90 ± 0.29**	50%	**67.79 ± 3.58**	77%	**17.68 ± 0.23**	63%

are shown in Fig. 4. It is worth noting that when COS achieves an accuracy falling within the standard deviations of baselines, we say that COS achieves no accuracy loss compared to baselines. From TABLE I, TABLE II and Fig. 4, we can find the following phenomena. Firstly, when achieving no accuracy loss, COS reduces the memory footprint of baseline GNN methods by $12\%\sim39\%$. Fig. 4 also demonstrates that 'COS (0%)' can match the accuracy performance of baseline GNN methods during training. Secondly, when achieving an accuracy loss of less than 1%, COS reduces the memory footprint of baseline GNN methods by $23\%\sim50\%$, which is larger than the reduction achieved when no accuracy loss is observed. Thirdly, when achieving the same reduction in memory footprint, RANDOM achieves much lower accuracy than COS in most cases. For example, RANDOM achieves $2\%\sim10\%$ lower accuracy than COS on dataset ogbg-molhiv and $5\%\sim13\%$ lower on dataset ogbg-molpcba. This phenomenon highlights the importance of exploiting repeated structural patterns for graph coarsening to achieve high accuracy. Fourthly, COS achieves varying reductions in memory footprint across different datasets. This variation is due to the differing amounts of repeated structural patterns presented in each dataset. The statistics of repeated structural patterns on benchmark datasets are presented in Fig. 3. Fifthly, COS achieves similar reductions in memory footprint when applied to different baseline GNN methods. This phenomenon demonstrates the potential of COS to be applied to other GNN variants, as baseline GNN methods in our experiments are

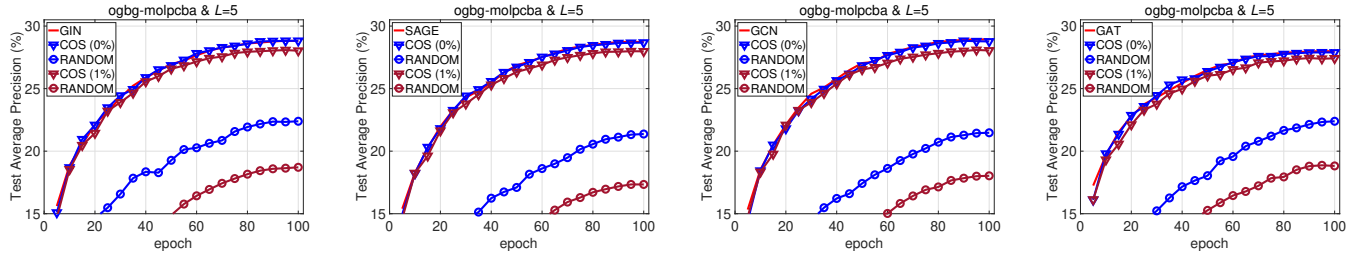


Fig. 4: Test accuracy-epoch curves on dataset ogbg-molpcba when L=5 (number of model layers). 'COS ($\triangle\%$)' and 'MemR' are defined in TABLE I. Methods achieving the same 'MemR' are marked in the same colors. Due to space limitations, we omit the test accuracy-epoch curves for L=10 and other three datasets (ogbg-molhiv, ogbg-ppa and ogbg-code2), as their trends are similar to the ones shown in the above figures.

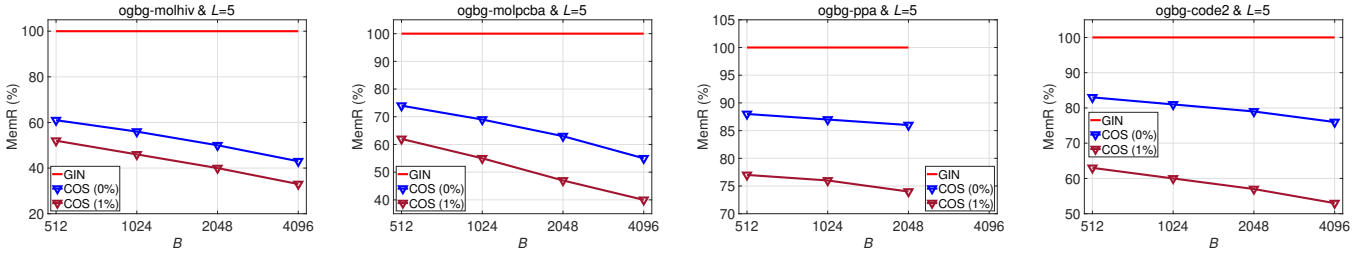


Fig. 5: Advantages with larger mini-batch size. Here, we only apply COS to GIN (L=5) for illustration, as similar phenomena can be observed when applying COS to other GNN methods. 'COS ($\triangle\%$)' and 'MemR' are defined in TABLE I. A smaller 'MemR' indicates higher memory efficiency. B represents the mini-batch size. It is worth noting that GIN encounters out-of-memory issues on dataset ogbg-ppa when B=4096, so this case is excluded from the figures.

building blocks for many other GNN variants. Lastly, when increasing L from 5 to 10, COS continues to achieve similar reductions in memory footprint. This phenomenon shows the potential of COS to be applied to deep and large GNN models.

D. Advantages with Larger Mini-batch Size

This subsection demonstrates the memory efficiency advantages that COS can achieve with larger mini-batch size B during training. The results are presented in Fig. 5. We can find that as B increases, COS achieves more significant reduction in memory footprint compared to baseline GNN methods. It is worth noting that maximum memory efficiency is achieved with a full-batch size.

V. Conclusion

Existing GNN methods for GRL suffer from memory inefficiency problem during training, mainly due to the *contextual constraint* imposed on node representations. Unfortunately, the memory inefficiency problem has received little attention from the GRL research community, despite the undesirable limitations it introduces. In this paper, we attempt to address this problem by first formally defining the contextual constraint and then proposing to improve the memory efficiency of GNN methods with a novel sketching technique. Furthermore, we prove that our proposed COS constructs an optimal solution to a memory-related objective associated with graph coarsening. To the best of our knowledge, we are the first to study the memory inefficiency problem in GNN methods for GRL. Experiments on four widely used benchmark datasets demonstrate that COS can reduce the memory footprint of baselines by a large margin with almost no accuracy loss.

Acknowledgment

This work is supported by NSFC Project (No.62192783) and Fundamental Research Funds for the Central Universities (No.020214380108).

References

[1] W. L. Hamilton, *Graph representation learning*. Morgan & Claypool Publishers, 2020.

[2] Z. Wu, B. Ramsundar, E. N. Feinberg, J. Gomes, C. Geniesse, A. S. Pappu, K. Leswing, and V. S. Pande, "MoleculeNet: a benchmark for molecular machine learning," *Chemical Science*, vol. 9, no. 2, pp. 513–530, 2018.

[3] D. Szklarczyk, A. L. Gable, D. Lyon, A. Junge, S. Wyder, J. Huerta-Cepas, M. Simonovic, N. T. Doncheva, J. H. Morris, P. Bork, L. J. Jensen, and C. von Mering, "STRING v11: protein-protein association networks with increased coverage, supporting functional discovery in genome-wide experimental datasets," *Nucleic Acids Research*, vol. 47, no. Database-Issue, pp. D607–D613, 2019.

[4] M. Allamanis, E. T. Barr, P. T. Devanbu, and C. Sutton, "A survey of machine learning for big code and naturalness," *ACM Computing Surveys*, vol. 51, no. 4, pp. 81:1–81:37, 2018.

[5] X. Kan, W. Dai, H. Cui, Z. Zhang, Y. Guo, and C. Yang, "Brain network transformer," in *NeurIPS*, 2022.

[6] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *ICLR*, 2017.

[7] W. L. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *NeurIPS*, 2017.

[8] M. Zhang, Z. Cui, M. Neumann, and Y. Chen, "An end-to-end deep learning architecture for graph classification," in *AAAI*, 2018.

TABLE III: Statistics of benchmark datasets.

Datasets	ogbg-molhiv	ogbg-molpcba	ogbg-ppa	ogbg-code2
#Graphs	41,127	437,929	158,100	452,741
Average #Nodes per graph	25.5	26.0	243.4	125.2
Average #Edges per graph	27.5	28.1	2,266.1	124.2
#Classes	2	2	37	-
#Training Graphs	32,901	350,343	78,200	407,976
#Validation Graphs	4,113	43,793	45,100	22,817
#Test Graphs	4,113	43,793	34,800	21,948
Task Type	Binary-class	Binary-class	Multi-class	Sub-token prediction
Metric	ROC-AUC	Average Precision (AP)	Accuracy	F1 score

[9] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *ICLR*, 2018.

[10] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" in *ICLR*, 2019.

[11] B. Weisfeiler and A. Leman, "The reduction of a graph to canonical form and the algebra which appears therein," *NTI Series*, vol. 2, no. 9, pp. 12–16, 1968.

[12] C. Bodnar, F. Frasca, Y. Wang, N. Otter, G. F. Montúfar, P. Lió, and M. M. Bronstein, "Weisfeiler and lehman go topological: message passing simplicial networks," in *ICML*, 2021.

[13] C. Bodnar, F. Frasca, N. Otter, Y. Wang, P. Liò, G. F. Montúfar, and M. M. Bronstein, "Weisfeiler and lehman go cellular: CW networks," in *NeurIPS*, 2021.

[14] G. Bouritsas, F. Frasca, S. Zafeiriou, and M. M. Bronstein, "Improving graph neural network expressivity via subgraph isomorphism counting," *IEEE TPAMI*, vol. 45, no. 1, pp. 657–668, 2023.

[15] M. Fey, J. Yuen, and F. Weichert, "Hierarchical inter-message passing for learning on molecular graphs," in *ICML Workshop*, 2020.

[16] C. Morris, M. Ritzert, M. Fey, W. L. Hamilton, J. E. Lenssen, G. Rattan, and M. Grohe, "Weisfeiler and leman go neural: higher-order graph neural networks," in *AAAI*, 2019.

[17] H. Maron, H. Ben-Hamu, H. Serviansky, and Y. Lipman, "Provably powerful graph networks," in *NeurIPS*, 2019.

[18] L. Zhao, L. Härtel, N. Shah, and L. Akoglu, "A practical, progressively-expressive GNN," in *NeurIPS*, 2022.

[19] H. Maron, H. Ben-Hamu, N. Shamir, and Y. Lipman, "Invariant and equivariant graph networks," in *ICLR*, 2019.

[20] N. Keriven and G. Peyré, "Universal invariant and equivariant graph neural networks," in *NeurIPS*, 2019.

[21] W. Azizian and M. Lelarge, "Expressive power of invariant and equivariant graph neural networks," in *ICLR*, 2021.

[22] M. Zhang and P. Li, "Nested graph neural networks," in *NeurIPS*, 2021.

[23] L. Zhao, W. Jin, L. Akoglu, and N. Shah, "From stars to subgraphs: uplifting any GNN with local structure awareness," in *ICLR*, 2022.

[24] F. Frasca, B. Bevilacqua, M. M. Bronstein, and H. Maron, "Understanding and extending subgraph GNNs by rethinking their symmetries," in *NeurIPS*, 2022.

[25] C. Qian, G. Rattan, F. Geerts, C. Morris, and M. Niepert, "Ordered subgraph aggregation networks," in *NeurIPS*, 2022.

[26] B. Bevilacqua, F. Frasca, D. Lim, B. Srinivasan, C. Cai, G. Balamurugan, M. M. Bronstein, and H. Maron, "Equivariant subgraph aggregation networks," in *ICLR*, 2022.

[27] V. P. Dwivedi and X. Bresson, "A generalization of Transformer networks to graphs," in *AAAI Workshop*, 2021.

[28] D. Kreuzer, D. Beaini, W. L. Hamilton, V. Létourneau, and P. Tossou, "Rethinking graph Transformers with spectral attention," in *NeurIPS*, 2021.

[29] C. Ying, T. Cai, S. Luo, S. Zheng, G. Ke, D. He, Y. Shen, and T. Liu, "Do Transformers really perform badly for graph representation?" in *NeurIPS*, 2021.

[30] L. Rampásek, M. Galkin, V. P. Dwivedi, A. T. Luu, G. Wolf, and D. Beaini, "Recipe for a general, powerful, scalable graph Transformer," in *NeurIPS*, 2022.

[31] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," in *NeurIPS*, 2020.

[32] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. L. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, J. Schulman, J. Hilton, F. Kelton, L. Miller, M. Simens, A. Askell, P. Welinder, P. F. Christiano, J. Leike, and R. Lowe, "Training language models to follow instructions with human feedback," in *NeurIPS*, 2022.

[33] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "Lora: low-rank adaptation of large language models," in *ICLR*, 2022.

[34] M. Chen, Z. Wei, Z. Huang, B. Ding, and Y. Li, "Simple and deep graph convolutional networks," in *ICML*, 2020.

[35] G. Li, M. Müller, A. K. Thabet, and B. Ghanem, "DeepGCNs: can GCNs go as deep as CNNs?" in *IEEE/CVF ICCV*, 2019.

[36] S. Verma and Z. Zhang, "Towards deeper graph neural networks," in *ACM SIGKDD*, 2020.

[37] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *ICML*, 2017.

[38] A. Loukas, "Graph reduction with spectral and cut guarantees," *JMLR*, vol. 20, pp. 116:1–116:42, 2019.

[39] M. Kumar, A. Sharma, and S. Kumar, "A unified framework for optimization-based graph coarsening," *JMLR*, vol. 24, pp. 118:1–118:50, 2023.

[40] X. Dong, D. Thanou, P. Frossard, and P. Vandergheynst, "Learning laplacian matrix in smooth graph signal representations," *IEEE TSP*, vol. 64, no. 23, pp. 6160–6173, 2016.

[41] V. Kalofolias, "How to learn a graph from smooth signals," in *AISTATS*, 2016.

[42] Y. Jin, A. Loukas, and J. F. JáJá, "Graph coarsening with preserved spectral properties," in *AISTATS*, 2020.

[43] C. Cai, D. Wang, and Y. Wang, "Graph coarsening with neural networks," in *ICLR*, 2021.

[44] Z. Ying, J. You, C. Morris, X. Ren, W. L. Hamilton, and J. Leskovec, "Hierarchical graph representation learning with differentiable pooling," in *NeurIPS*, 2018.

[45] R. L. Murphy, B. Srinivasan, V. A. Rao, and B. Ribeiro, "Janossy pooling: learning deep permutation-invariant functions for variable-size inputs," in *ICLR*, 2019.

[46] H. Gao and S. Ji, "Graph U-nets," in *ICML*, 2019.

[47] J. Lee, I. Lee, and J. Kang, "Self-attention graph pooling," in *ICML*, 2019.

[48] J. Baek, M. Kang, and S. J. Hwang, "Accurate learning of graph representations with graph multiset pooling," in *ICLR*, 2021.

[49] D. Buterez, J. P. Janet, S. J. Kiddle, D. Oglic, and P. Liò, "Graph neural networks with adaptive readouts," in *NeurIPS*, 2022.

[50] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: an imperative style, high-performance deep learning library," in *NeurIPS*, 2019.

[51] M. Fey and J. E. Lenssen, "Fast graph representation learning with PyTorch Geometric," in *ICLR Workshop*, 2019.

[52] W. Hu, M. Fey, M. Zitnik, Y. Dong, H. Ren, B. Liu, M. Catasta, and J. Leskovec, "Open graph benchmark: datasets for machine learning on graphs," in *NeurIPS*, 2020.

[53] D. P. Kingma and J. Ba, "Adam: a method for stochastic optimization," in *ICLR*, 2015.