We thank the reviewers for the constructive comments! Please refer to both **the revised submission** and **the Appendix included in our full version [1]**, where we have addressed all comments, corrected presentation issues and typos, and made additional improvements, including enhanced clarity and expanded experimental results. We highlighted the changes for each reviewer with a different color in the text (blue: R1; red: R2; orange: R3). The rest changes are marked in blue by default.

REVIEWER 1 (R1).

[W1] *I think several important experiments are missing. In the paper, only the accuracies of several compression methods are compared. The accuracy of the GNNs on the original graph is not compared in the experiments. This comparison is important to show whether this algorithm is meaningful.*

[A] Please refer to the updated Figure 8 in Section 7.2. We report the inference accuracy on the original graph G for all three GNNs variants, each represented in a different color, across all four datasets. Our new results verified that (0.5, 1)-SPGC achieves inference accuracy on par with direct inference on the original graph G across all GNNs variants and all four datasets, while consistently outperforming the baseline methods DSpar and FGC in preserving inference results.

[W2] *Similar to weak point 1, the paper does not show the accuracy of* ASPGC. *From the sections that introduce* ASPGC, *I doubt its accuracy, because as the embeddings of the neighbors of anchored nodes cannot be preseved, the embeddings of the anchored nodes might be inaccurate. Consequently, showing the accuracy of* ASPGC *is really important.*

[A] Thanks. We have revised "Compression and Inference" paragraph in Section 5.1 to clarify the following. (1) The compression scheme of ASPGC following its counterpart in SPGC, which enforces "embedding equivalence". (2) ASPGC adopts the inference process that also consistently recovers the neighborhood information of up to L-hop neighbors of a given set of anchored nodes V_A, without the need of complete reconstruction of their neighborhood. The "absolute" values of accuracy alone of ASPGC vary for different choices of V_A; and in practice, V_A can be set as the task-specific test nodes V_T. We remark that ASPGC does not aim to optimize the absolute accuracy of GNNs, yet to speed up their inference process while preserving their accuracy.

We have reported the accuracy of ASPGC with a default setting ($V_A = V_T$) and summarize our finding below, as suggested. As shown in the revised Figure 8 in Section 7.2, we highlighted the inference accuracy of ASPGC as colored lines (marked in blue for GCN, red for GAT, and yellow for GraphSAGE). We observe that the inference accuracy of ASPGC exactly matches the the inference accuracy derived by inferencing on original G. For example, on **Cora**, ASPGC achieves an inference accuracy of 0.72 (in MAE) using a 3-layers GCN, which is the same to its counterpart of the same 3-layers GCN on original G. Our tests over other datasets consistently verified this strong results. These results, among others we have shown for ASPGC, verified that ASPGC can very well preserve the accuracy of GNNs in practice for different inference queries.

[W3] *I cannot understand the "inference process with decompression" section in section 6.1. I think the whole paragraph should be rewritten. For example, "it prioritizes the decompression of nodes having the most shared neighbors with others in G": how can you achieve this? This example is not the only confusing sentence in the paragraph. All sentences in this paragraph after the word "Specifically" are not readable.*

[A] We have rewrote the whole paragraph in Section 5.1 and Appendix [1] where we have clarified how procedure decompG prioritizes the decompression of nodes with the most shared neighbors. We enriched the explanation of how the decompression algorithm prioritizes nodes with the most shared neighbors and how it integrates with the original Re-Pair to form our proposed extended Re-Pair method. Furthermore, due to the limited space, we include the relevant details algorithm (α, r)-SPGC, procedure decompG, example 8 (deriving decompressed G_{cd} from G_c), and Figure 18 (compression by Re − Pair and decompression by decompG) in the Appendix [1].

[D1] *I cannot find the number of anchored vertices in* ASPGC *in every dataset. This information is important for evaluating the performance of* ASPGC.

[A] Please refer to the revised Section 5.2 and Section 7.1. We have clarified the size of the set of anchored nodes V_A and its relationship to the set of test nodes V_T. By default, $V_A = V_T$ for every dataset, we set $|V_A| = 0.05 * |V|$.

[D2] *There are several typos. For example, in page 2, "with x ranges over s,d,n and p". I think "p" should be replaced by "v". Another example: Page 7, Section 5.3, the "compression time memoization" section: sometimes the subscript T is in a normal font, but most of the times it is cursive.*

[A] Thanks! We have corrected all these typos.

[D3] *The memory footprints of GNNs of compression methods should be shown. Simply showing the compression ratio on the graph is not sufficient.*

[A] We have added the results for memory footprints. Please refer to the new Table 9 and Memory Cost Analysis section in Appendix [1] for detailed experimental results and analysis. We define a new metric Memory Compression Rate (mcr) as mcr = $1 - \frac{|M_c|}{|M|}$. It quantifies the fraction of memory cost that is "reduced": the larger, the better. We measure the peak memory footprints of the GNNs (a 3-layers GCN model with a hidden size of 32) during the inference process. Compared to the original graph G, the compressed G_c achieves the mcr of 14.58%, 8.09%, and 65.19% for ***Arxiv***, **Yelp**, and ***Ogbn-Products*** datasets respectively. Notably, as the size of original graph G increases, the reduction in peak memory of GNNs becomes more pronounced, reflecting the overall reductions in both graph size and GNNs memory footprint. Besides, please also refer to A(R2(W1)) and A(R2(D4)).

REVIEWER 2 (R2).

[W1] *No experimental study of space.*

[A] Please refer to Table 9 and Memory Cost Analysis Section in Appendix [1] for detailed experimental results and analysis. We conduct the following three tests to evaluate spacing savings from SPGC: (1) GM: the graph memory cost of G versus G_c derived by SPGC measured in PyTorch Geometric Graph Data format; (2) MSM: memory cost of memoziation structure $\mathcal{T}$; (3) PM: the peak memory consumption of GNNs models inference operation on G versus G_c. Besides, please also refer to A(R1(D3)) and A(R2(D4)).

[D1] *Sec 1, "(1) Inference in large networks.": What is d in the O() complexity? All other variables are introduced; d is not.*

[A] We added the definition of d in Section 1 for clarification.

[D2] *Figure 1: Why $M((G_c))$ has double parentheses.*

[A] We have fixed the typo and revised Figure 1 accordingly.

[D3] *Figure 2: Not sure why the example engages in the question whether a nurse is female or male. It adds nothing and would be easy to avoid. You could use years of experience or salary as a third attribute if a third attribute is badly needed to make the example work.*

[A] Please refer to revised figure and example. We have revised this example and the descriptions of this example. We include age, years of experience (YoE), and Department (Dpt) as new node features.

[D4] *Sec 5.3, Memoization structure $\mathcal{T}$: The size of $\mathcal{T}$, as illustrated in Figure 6, depends on the number of vertices in G. It would be interesting to have size comparisons of G vs. $G_c + T$. The compression certainly will come with space savings. But how much?*

[A] Please refer to Table 9 and Memory Cost Analysis Section in Appendix [1] for detailed experimental results and analysis. We observe the followings: i) the memory cost of $\mathcal{T}$ of G_c accounts for 10.21% of that of G_c

and only 4.33% of G on average across all the three datasets; and ii) The combined memory cost of G_c + $\mathcal{T}$ still remains significantly smaller than that of G, achieving mcr of 34.56%, 16.72%, and 71.58% for ***Arxiv***, **Yelp**, and ***Ogbn-Products***. Besides, please also refer to A(R1(D3)) and A(R2(W1)).

[D5] *Sec 5.3, "CompressG": CompressG seems to pretty straightforward computation of a summary graph.*

[A] We have proposed the light-weighted compression approach CompressG which enables an easy-to-implement, and feasible way to derive compressed graph G_c from EC for large-scale graph G. In addition, different from above methods, we derive the memoization structure $\mathcal{T}$ while compressing G, which stores information to ensure inference-preserving guarantee of SPGC. While we have theoretically shown desired properties such as minimality and uniqueness of compressed graphs, we consider having a solution that is both easy to implement and meanwhile ensure the above properties in PTIME an advantage, comparing with other alternatives that introduce and solve intractable optimization problems.

[D6] *Sec 7.1, "Arxiv [25], an academic collaboration network": Why is Arxiv portrayed as a "collaboration network" then it is, according to the description following right after, clearly a citation network? (Or conversely, if we run by the classification of Arxiv, what makes Cora not a collaboration network?)*

[A] Nice suggestion! We have revised the descriptions to clarify that both **Arxiv** and **Cora** datasets are depicting citation networks. Please refer to the revised Section 7.1.

[D7] *Sec 7.1, "Yelp": What do edges denote in the Yelp network?*

[A] We included the definition of edges of **Yelp** dataset to Section 7.1.

[D8] *Sec 6.1 and Sec 7.2, Exp-2: What happens at /alpha = 0 and r = diameter of the graph? This probably compresses the whole graph into a single node, doesn't it? At which parameter setting does the accuracy fall off a cliff? Explore the weak spots of the approach and the corner case of parameter settings.*

[A] Yes, when $\alpha = 0$ and $r = d$ (diameter of the graph), all nodes collapse into one equivalence class such that $EC = \{V\}$. d is on average equal to 6 for most of the small-world graphs [49]. Since most GNNs do not have more than three layers ($L \leq 3$) due to the issue of over-smoothing [30], r cannot go all the way up to be equal to the diameter of graph since $r \leq L$ must hold. In most cases, r = 1, 2, or 3 is sufficient for most GNNs. In practice, the user can select a suitable value of r (typically 2 or 3, depending on the dataset and model) to balance inference speed-up and accuracy, as demonstrated in Figure 10(e) and Figure 10(f).

[D9] *Figure 9: Why has FGC no results for Arxiv, Yelp, and Products? The text says "SPGC achieves ncr up to 74.5% while [...] FGC achieves [...] 30.7%". This number, 30.7%, is not plotted in the figure, it seems. Why?*

[A] We set the longest waiting limit for the compression cost associated with baselines Dspar and FGC as 5 hours. The numbers, 46.2% and 30.7%, computed based on the definition of ncr, are not plotted since the compression costs of Dspar and FGC on ***ogbn-products*** as well as **Yelp** and **Arxiv** are over 5-hours. The reason we set 5-hours as the upper limit is that the compression time already surpasses the time cost of training a new GNNs from scratch for the corresponding datasets. For FGC, only **Cora** is shown since the compression time of FGC on **Cora** is less than five hours.

[D10] *Figure 12(c): Arguing scalability based on an increase from "6M to 10M" is not very convincing. That is not even a 100% increase. How about an increased range that spans like 2 orders of magnitude? In fixed domain, at some scale there are likely no new cases coming in with further growth of the graph that are interesting to whatever the GNN is trying to classify. So, at some point, you would expect the compression rate to start to increase with the graph size. The experiments do not look into if such as effect exists.*

[A] We have conducted a large-scale experiment to evaluate how the linear increase of graph size $|G|$ affects the compression cost of DPP, SPGC, (0.5, 1)-SPGC, and ASPGC. Please refer to Figure 11(c) and Impact analysis of $|G|$ in Section 7.2., along with the detailed experimental results and analysis. Figure 11(c) shows the compression cost of four data points (0.5, 1)-SPGC corresponding to the graph size $|G|$ of 6M, 204M, 492M, and 600M, spanning 2 orders of magnitude. Our observation remain consistent.
We also remark that we treat GNNs as a function that learns to best represent node representations for the entire graph, unless anchored nodes (or test nodes) are explicitly specified – which specifies the fraction of the GNNs "the interesting fraction the GNN is trying to classify"; and for the latter, we specialize anchored compression scheme accordingly. Hence we have provided a full treatment to scale GNN inference over the growth of the graphs, and have considered both possibilities with three compression schemes.

[D11] *Sec 7.2, "Expr-4", "We present additional tests with details in [1].": Well, at the time of review, [1] contains the exact same sentence, which is weird. [1] does not seem to have any additional experiments or experiment details. It has additional proofs and algorithms, but nothing additional that is easily identifiable about the experiments.*

[A] Additional tests here refer to two additional experiments (1) the impact of aggregation method on inference accuracy of SPGC; and (2) a pilot study of applying SPGC to GNNs training achieving improved training cost. We have included these two experiments to the Appendix [1].

REVIEWER 3 (R3).

[W1] *The motivation for similarity merging is unclear. (D1)*

[D1] *It is not well explained why graph data would contain a large number of similar vertices, and why both their vertex features and the features of vertices within multi-hop subgraphs are similar. It would be better to conduct both theoretical analysis and experimental validation.*

[A] Please refer to the Appendix [1]. We added a new paragraph to clarify our idea of similarity merging which leverages both **structural similarity** and **embedding similarity**. Compared to traditional clustering or classification-based merging, which relies solely on structural properties or embeddings, SPGC prioritizes structural similarity using our proposed structural equivalence. This is followed by fine-tuning with embedding similarity based on input node features. Hence, SPGC does not necessarily require the existence of similar nodes in terms of their features and the similarities among their nodes within multi-hops. We applied a featurization pre-processing step to discretize input features. This ensures that numerical values within the same ranges are categorized together after featurization, improving consistency in similarity measures. We introduce a new metric, the percent of similar nodes psn to quantify similar nodes within a graph. It is defined as psn = $1 - \frac{|V_{dis}|}{|V|}$, where $|V_{dis}|$ represent the number of stand-alone nodes in the equivalence relation R^S that cannot be merged with any other nodes in V. The psn of a given graph G depends on its structural and embedding similarities. Using the equivalence relation R^S derived from SPGC, we directly compute the psn for representative datasets. For example, in the **Arxiv**, 43.06% nodes in **Arxiv** are similar whereas 87.42% nodes in ***Ogbn-Products*** and 77.51% nodes in **Cora** are similar. To enhance the performance of compression for any arbitrary graph, our variant (α, r)-SPGC, introduces configurable parameters α and r, enabling fine-tuned control over the merging process and improving the compression ratio.

[W2] *The compression algorithms based on the similarity principle may not be general enough to other types of graphs. (D2-D4)*

[D2] *It is unclear to me what types of input graphs are suitable for similarity merging. Therefore, it is not easy to believe that* SPGC *can achieve good compression performance across all graph data, as the assumption that multi-hop subgraph structures and their contained vertex features are similar is strict.*

[D3] *On the other hand, although the authors claim that* SPGC *is a compression method that does not require consideration of the model architecture, I think the model layers and the complexity of the model still influence* SPGC. *First, the embedding similarity between two vertices is dependent on the model depths. The similarity between vertices may vary across different model layers, as the receptive field of graph aggregation continually expands. Related experiments have also shown a decrease in accuracy for SPGC in deeper models. In fact, GNN models encounter the problem of over-smoothing as the number of layers increases, because the vertex embeddings gradually converge, making it difficult to distinguish between vertices [TOWARDS DEEP GRAPH CONVOLUTIONAL NETWORKS ON NODE CLASSIFICATION ICLR'20]. It is unclear whether the similarity vertex merging in SPGC exacerbates the over-smoothing issue, as even vertices that are not fully similar might be merged together. On the other hand, complex models like GAT involve edge-based neural network computations, where the weight of each edge is related to the model parameters. This suggests that calculating the embedding similarity between vertices may not be feasible without considering the model itself.*

[D4] *The graphs are too small to validate the efficiency of SPGC. It would be better to evaluate SPGC on larger datasets, such as Ogbn-Papers100M.*

[A] For D2, please also refer to Appendix [1]. We have clarified our proposed similarity merging by highlighting its difference from traditional clustering and classification (please also refer to A(R3(W1))). Our similarity merging leverages both structural similarity and embedding similarity to construct similarity-based merging. It prioritizes structural similarity as defined by structural equivalence and fine-tunes the equivalence relation by embedding similarity. Even if multi-hop subgraph structures and their contained vertex features are not similar, SPGC can still achieve desirable compression ratio.

We also would like to point out that the "generality" of our methods, like any other data compression scheme, should be interpreted as its capacity to be applicable to various types of graph data, but do not indicate that it always guarantee a high compression ratio – as the latter indeed depends on the task-specific embedding approaches and data properties. This is what motivated us to propose configurable (α, r)-SPGC, allowing the user to flexibly increase the compression ratio to strike a balance between the inference speed-up and inference accuracy over graphs with high heterogeneity or GNNs with over-smoothing issues. Therefore, our SPGC and its variants are general-purpose graph compression frameworks, and are generally applicable for GNN classes specified by a layer bound L and node update function, across graphs with different levels of embedding heterogeneity.

For D3, please refer to revised Section 5.1. (1) We have clarified the relationship between r and L (number of layers). Based on the definition of SPGC, the embedding similarity between two vertices is dependent on r rather than L. In other words, SPGC is model-agnostic as long as $r \leq L$. In practice, since L is small, the safe choice of r is usually 1 or 2 (please also refer to A(R2(D8))); (2) As shown in Figure 10(e) and Figure 10(f), fixing L of GNNs, increasing r reduces the inference accuracy in trade for better inference speed-up. This is because increasing r allowing more nodes being merged together, resulting in larger compression ratio, may lead to decreasing inference accuracy. However, this is not caused by exacerbating over-smoothing issue since we fix the number of layers of all GNNs to be 3; (3) For GAT, we need the attention weights from model M to construct the auxiliary memoziation structure $\mathcal{T}$. However, the process of calculating embedding similarity calculation in SPGC and its variants does not need the model weights or any model information since this computation only depends on the choice of α and r (please refer to Algorithm (α, r)-SPGC in Appendix [1]).

For D4, due to hardware constraints, we are unable to host a large-scale dataset like Ogbn-Papers100M. Having this said, with all our theoretical guarantees remain intact, we will conduct experiments on datasets at this scale when we can access the sufficient computing resources. Another topic is to investigate parallel graph compression scheme to facilitate large-scale GNN inference. We have enriched Section 8 to include these as future work.

[W3] *The advantages over related work are insignificant and some related work is missed. (D5)*

[D5] *The authors discuss related work in Section 2, including graph sparsification and graph coarsening, and mention that SPGC is orthogonal to existing methods. This looks like that SPGC addresses graph compression from a different perspective rather than offering a superior solution compared to existing work. Additionally, some graph compression techniques applied for training [Graph-Skeleton WWW'24, NeutronSketch KBS'24], should also be discussed, as they can be used for inference as well. In summary, the strength of the contribution and its impact should be further highlighted.*

[A] We added a new category of related works, Graph *Sketch*, to related work. We discussed Graph-Skeleton [12] and NeutronSketch [35] and clarified the important differences between our proposed SPGC and these two approaches. Graph-Skeleton can effectively reduces the memory usage for a pre-defined set of target nodes. However, it lacks flexibility and generalizability, as it must be tailored to specific nodes. The compression of NeutronSketch is restricted to the training phase, it does not accelerate inference, as validation and test nodes remain uncompressed. Even if applied directly to test nodes, it does not guarantee inference accuracy for the compressed subgraph containing test nodes. By contrast, our SPGC has following the advantages compared to existing approaches: (1) more efficient: the compression cost of SPGC and its variants are linear to size of graph by performing one-time compression that applies universally to any set of test nodes; (2) inference-preseving with provable inference accuracy guarantee for any $V_T \in V$.

[D6] *I am also unsure whether* SPGC *can be applied to accelerate training and what limitations it might encounter. In practice,* GNNs *training typically involves much higher computational overhead compared to inference, requiring hundreds of epochs, which makes it a more promising target for computational acceleration.*

[A] Please refer to the new experiment of Training Cost and Accuracy of SPGC in the Appendix [1]. We applied SPGC to train a 3-layers GCN using **Arxiv**. SPGC reduces the training time by 40.57% using G_c compared to training on the original graph G while retaining comparable inference accuracy. In addition, we added discussions of extending SPGC from inference to GNNs training in Section 1. We pinpointed applying SPGC to GNNs training in two learning settings: (1) transductive-learning: training GNNs with compressed G_c and leveraging $\mathcal{T}$ for accuracy-preserving inference; and (2) inductive-learning: learnable G_c fine-tuned by optimizing training loss. For the latter one, it is more costly but potentially performs better for more challenging inductive-learning setting. We consider this as a promising pilot result. While the scope of this paper is to accelerate GNN inference, we plan to explore graph compression for accelerating GNN training. We have enriched Section 8 to include this as an interesting future work.

Inference-friendly Graph Compression for Graph Neural Networks

Yangxin Fan, Haolai Che, Yinghui Wu
Case Western Reserve University
Cleveland, Ohio, USA
{yxf451,hxc859,yxw1650}@case.edu

ABSTRACT

Graph Neural Networks (GNNs) have demonstrated promising performance in graph analysis. Nevertheless, the inference process of GNNs remains costly, hindering their applications for large graphs. This paper proposes *inference-friendly graph compression* (IFGC), a graph compression scheme to accelerate GNNs inference. Given a graph G and a GNN M, an IFGC computes a small compressed graph G_c, to best preserve the inference results of M over G, such that the result can be directly inferred by accessing G_c with no or little decompression cost. (1) We characterize IFGC with a class of inference equivalence relation. The relation captures the node pairs in G that are not distinguishable for GNN inference. (2) We introduce three practical specifications of IFGC for representative GNNs: structural preserving compression (SPGC), which computes G_c that can be directly processed by GNN inference without decompression; (α, r)-compression, that allows for a configurable trade-off between compression ratio and inference quality, and anchored compression that preserves inference results for specific nodes of interest. For each scheme, we introduce compression and inference algorithms with guarantees of efficiency and quality of the inferred results. We conduct extensive experiments on diverse sets of large-scale graphs, which verifies the effectiveness and efficiency of our graph compression approaches.

1 INTRODUCTION

Graph Neural Networks (GNNs) have shown promising performance in various analytical tasks such as node classification [30] and link prediction [54]. In general, a GNN $\mathcal{M}$ converts an input graph G (as a pair (X, A) of node feature matrix X and an adjacency matrix A), to a vector representation ("embeddings") $\mathcal{M}(G)$ via multiple layers. For each node, each layer applies a same "node update function" to uniformly update its embedding as a weighted aggregation of embeddings from its neighbors, subsequently transforming it towards an output embedding. The training of $\mathcal{M}$ is to optimize its model parameters ("weights") and obtain a proper update function to make it best fits a set of training data in a training graph. Given an input (test) graph G, the *inference* of $\mathcal{M}$ applies the node update function to generate output embeddings $\mathcal{M}(G)$ (a matrix of node embeddings). $\mathcal{M}(G)$ can be post-processed to task-specific output, such as class labels for node classification.

Despite their promising performances, GNNs incur expensive inference process when G is large [13, 52, 57]. The emerging need for large-scale testing, fine-tuning and benchmarking of graph learning models, require fast inferences of GNNs under various configurations. Consider the following scenarios.

(1) *Inference in large networks.* For a graph G with $|V|$ nodes and $|E|$ edges (where V and E refers to its node and edge set), with on average F features per node, an L-layered GNN M may typically take $O(L|E|dF^2 + L|V|F^2)$ time [13] (as summarized in Table 3). This can be prohibitively expensive for large real-world graphs.

(2) *Real-time Inference.* GNN models have been developed for *e.g.,* traffic analysis [41], social recommendation [17], energy forecasting [2, 18, 29], cybersecurity [59], computer vision [45], and edge devices [57]. Such scenarios often require real-time response at *e.g.,* milliseconds [57]. In such cases, even a linear time inference of "small" GNNs (when F and L are small constants) may still not be feasible for large graphs (when $|V|$ and $|E|$ are large).

(3) *Fine-tuning & Benchmarking.* Fine-tuning and testing pre-trained GNNs to adapt them for various domain-specific tasks is a routine process in GNN-based data analysis for *e.g.,* materials sciences, biomedicine, social science, and geosciences [24, 42, 47, 56, 58]. Inference tests of large pool of "candidate" GNNs over domain-specific graph data (such as knowledge graphs) is a cornerstone in such context. Fast GNN inference can accelerate large-scale domain-specific testing and benchmarking.

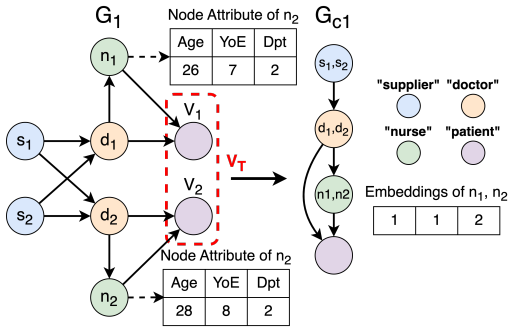
Several approaches have been developed to accelerate GNN inference, by simplifying model architecture [40], (learning) to optimize inference process [46], or data sampling [14]. These methods typically works with specific GNN $\mathcal{M}$, requires prior knowledge of its internals (*e.g.,* model parameter values), and may incur new computation overhead each time a different GNN $\mathcal{M}$ is specified.

Compressing graphs for GNN inference. Unlike prior "model-specific" approaches, we propose a model-agnostic, "*once-for-all*" graph compression scheme to accelerate GNN inference, for a *set* of GNNs. Consider a set of GNNs $\mathbb{M}$ (a GNN "class") with the same form of inference, which apply the same "type" of the node update function but only differs in model weights (see Example 1). The inference of a GNN M over G can be characterized as an "inference query" [6, 22], which invokes the inference of $\mathcal{M}$ to compute $\mathcal{M}(G)$.

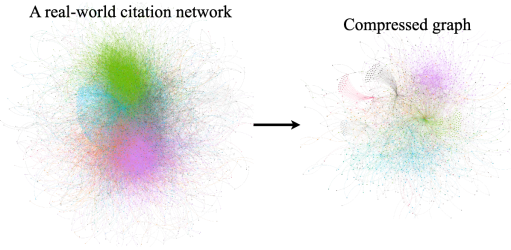
We advocate an *"inference-friendly" graph compression* scheme for GNN inference at scale. Given $\mathbb{M}$ and a large graph G,

- It uses a compression function $\mathcal{C}$ to compute a smaller counterpart G_c of G "once-for-all", for any GNN$\mathcal{M} \in \mathbb{M}$;
- For any inference query that requests $\mathcal{M}(G)$ for a specific GNN $\mathcal{M} \in \mathbb{M}$, it performs an inference directly over G_c instead of G to compute $\mathcal{M}(G_c)$, with a reduced time cost, such that $\mathcal{M}(G_c)$ (approximately) equals $\mathcal{M}(G)$.

Such a compression is desirable: (1) It readily reduces the cost for any single inference query that computes $\mathcal{M}(G)$; (2) An inference query often does not require the entire output $\mathcal{M}(G)$ but only a fraction $\mathcal{M}(G, V_T) \subseteq \mathcal{M}(G)$ of a specified test (node) set V_T of interests; (3) Multiple inference queries can be posed to request output from different GNNs in $\mathbb{M}$ in G. For any workload with



(a) Social role classification: a hospital network (G_1) can be "compressed" by merging nodes with equivalent social roles for testing a GCN-based classifier (adapted from [11], node attributes include Age, YoE (years of experience), and Dpt (department)).



(b) Visualization of a fraction of real-world citation network [27] ($|G| = 1,335,586$) and its compressed counterpart ($|G_c| = 148,887$) for a GraphSAGE-based node classification. 88.6% of nodes and edges are compressed, reducing inference cost by 92.0%, achieving 12.5 times speed-up with up to 6.7% loss of accuracy.

Figure 1: Compression Scheme to scale node classification.

inference queries that specify any GNN $\mathcal{M} \in \mathbb{M}$ and any V_T from G, one only need to compute G_c once, to reduce the total inference cost of the workload. These benefits applications in large-scale tests over large graphs, real-time inference and benchmarking, as aforementioned.

While desirable, *is such a compression scheme doable?* We illustrate a case in the following example.

Example 1: Consider a 3-layer Vanilla GNN $\mathcal{M}$ [44] as a node classifier that assigns role labels {supplier, doctor, nurse, patient} in a social healthcare network G_1 (illustrated in Fig. 1). Each node in G_1 has attributes such as role, age group, department, etc. To infer the roles of test nodes $V_T = \{v_1, v_2\}$, an inference process of $\mathcal{M}$ starts by propagating a node feature matrix X with a node update function M_v. Via a 3-layer forward message passing, the embeddings of v_1 and v_2 are obtained, quantifying the likelihood of them being assigned to one of the labels. As the probability of "patient" is the highest for both, it infers both labels as "patient".

We take a closer look at the update function M_v at layer k:

$$X_v^k = \sigma(\Theta \cdot \sum_{u \in N(v)} X_u^{k-1})$$

where X_v^{k-1} (resp. X_v^k) is the embedding of a node v at the $(k-1)$-th (resp. k-th) layer; σ is an activation function, $N(v)$ refers to the neighbors of node v, and Θ refers to the learned weight matrix (same across all layers in the GNN).

If the input features of a pair of nodes x_1 and x_2 are the same (with x ranges over s, d, n and v), then the embedding of x_1 and x_2 will be the same during the inference computation, as long as the above the node update function is applied for any fixed model weights Θ and any fixed total number of layers. That is, x_1 and x_2 are "indistinguishable" for the inference of any 3-layered GNN $\mathcal{M}$ that adopts the above node update function M with the same aggregator AGG, regardless of how its Θ changes.

Note that feature equivalence does not necessarily mean that x_1 and x_2 have exactly the same attribute values as input. For example, while n_1 and n_2 refer to a 26 years old and a 28 years old nurse, respectively, their ages, years of experience (YoE), and department (Dpt) fall in the same group via (categorical or one-hot) encoding. Hence, they have the same input feature.

By "recursively" merging all such node pairs into a "group node" that are connected to neighbors that are also indistinguishable groups (*e.g.*, $[s] = \{s_1, s_2\}$, $[d] = \{d_1, d_2\}$, $[n] = \{n_1, n_2\}$, and $[v] = \{v_1, v_2\}$), a smaller graph G_c can be obtained. An inference directly over G_c can yield the same output for the test nodes v_1 and v_2 for $\mathcal{M}$ without decompression. To see this, one just need to "recover" their original value with a constant factor 2 (their original degrees) as auxiliary information at query time, for each layer:

$$X_{v_1}^k = X_{v_2}^k = 2 \times X_{[v]}^k = 2 \times \sigma(\Theta \cdot X_{[n]}^{k-1})$$

Such aggregated neighborhood information (*e.g.*, degrees, edge weights/attentions, or hyper-parameters) can be readily "remembered" at compression time, and be retrieved in constant time. This indicates an overall cheaper inference cost, and an exact query-time restore of the original embedding, for any node in G.

Better still, we only need to compute G_c "once-for-all", to reduce the unnecessary inference cost for any set of inference queries that specify a 3-layer GNN $\mathcal{M}$ (regardless of their weights Θ) that uses the same node update function M, and for any V_T in G. □

The above example verifies the possibility of a graph compression scheme by finding and merging node pairs that are indistinguishable for the inference process of GNNs. Our study verifies that real-world graphs are indeed highly compressible with such structures, and if compressed, well preserve inference output with no or small loss of accuracy, and meanwhile significantly reduce unnecessary inference computation (see Fig. 1 (b)).

Contributions. Our main contributions are as follows.

(1) We formally introduce *inference-friendly graph compression scheme* (IFGC), a as a general scheme to scale GNN inference to large graphs. We characterize IFGC with an *inference equivalence* relation, which captures the nodes with embeddings that are indistinguishable for the inference process using the same type of node update function. We then introduce a sufficient condition for the existence of IFGC, which specifies G_c as the quotient graph of G induced by the inference equivalence relation.

(2) We specify IFGC for representative GNNs classes. We first introduce structural preserving compression (SPGC), that enforces node embedding equivalence and neighborhood connectivity. We show it computes a compressed graph G_c in $O(|E| \log |V|)$ time, which can be directly processed by the inference process to retrieve the original results *without* decompression. We further justify SPGC by showing that it can produce a unique, smallest G_c up to graph

Methods	Category	LB	MA	IFGS	Compression Cost
Dspar [36]	S	×	✓	✓	$O(\frac{\|V\|log\|V\|}{\epsilon^2})$
AdaptiveGCN [33]	S	✓	×	×	n/a
NeuralSparse [55]	S	✓	✓	×	$O(q\|E\|)$
SCAL [28]	C	✓	✓	×	n/a
FGC [31]	C	✓	✓	×	$O(\|V\|^2\|V_c\|)$
SPGC (Ours)	C	×	✓	✓	$O(\|V\| + \|E\|)$
(α, r)-SPGC (Ours)	C	×	✓	✓	$O(\|V\|\|N_r\| + \|E\|)$
ASPGC (Ours)	C	×	✓	✓	$O(\|G_L\|)$

Table 1: Graph compression to accelerate GNN inference. S: Sparsification, C: Coarsening, LB: Learning-Based, MA: Model-Agnostic, IP: Inference-friendly w. guarantee, ϵ: a constant that controls approximation error, q: # of visits of the neighbors per node. N_r: the largest r-hop neighbor set for a node in G. G_L: the subgraph of G induced by L-hop of anchored node set V_A for L-layered GNNs. .

isomorphism among compressed graphs. We also show that SPGC preserves the discriminability of the GNNs.

(3) We further introduce two *configurable* variants of IFGC to allow flexible trade-off between compression ratio and the quality of inference output. (a) The (α, r)-SPGC groups nodes with similar features (determined by a threshold α), that also have similar counterparts within their r-hop neighbors. (b) The *anchored*-SPGC (ASPGC) adapts SPGC to an "anchored" node set of user's interests and preserve the inference results for such nodes only rather than the entire node set. For both variants, we introduce efficient compression and inference algorithms.

(4) We experimentally verify the effectiveness and efficiency of our graph compression schemes. We show that with cheap "once-for-all" compression, our compression methods can significantly reduce the inference cost of representative GNNs such as GCNs, GAT and GraphSAGE by 55%-85%, with little to no sacrifice of their accuracy.

Related work. Several approaches have been developed to accelerate GNN inference in large graphs [14, 20, 40, 57]. Closer to our approach is graph reduction, which simplifies graphs at a small sacrifice of model accuracy [25, 34]. There are three strategies.

Graph Sparsification. These methods (learn to) remove task-irrelevant edges from input graphs, such that the remaining part preserves the performance of GNNs. For example, AdaptiveGCN [33] learns an edge predictor to determine and remove task-irrelevant edges to accelerate GNN inference on CPU/GPU clusters. NeuralSparse [55] learns supervised DNNs to remove task-irrelevant edges. [14] proposed a framework to incorporates both model optimization and graph sparsification, which leverages lottery ticket hypothesisto identify subnetworks that can perform as well as the full network. Dspar [36] induces smaller subgraphs by removing edges that have similar "importance" (quantified by approximating a resistance measure as in circuits) to preserve graph spectrum.

Graph Coarsening. These methods group and amalgamate nodes into groups, without removing nodes. For example, SCAL [28] proposed the use of off-the-shelf coarsening methods LV[38] for scaling up GNN training and theoretically proved that coarsening can be considered a type of regularization and may improve the generalization as well as reduce the number of nodes by up to a factor of ten without causing a noticeable downgrade in classification accuracy. [31] introduced an optimization-based framework (FGC) that incorporates graph matrix and node features to jointly learn a coarsened graph while preserving desired properties such as spectral similarity [38]. GRAPE [50] is a GNN variant enhanced with sampled subgraph features from ego networks of automorphic equivalent nodes. It has a different goal of improving accuracy rather than reducing inference costs. [10] compresses graphs to accelerate GNN learning, using color refinement (with a case of bisimulation) to merge nodes within bounded radius. Node groups are iteratively refined based on a label encoding that concatenate node label and neighboring group colors. This is similar with SPGC. Nevertheless, no inference algorithm is provided. We show that our (α, r)-compression subsumes bisimulation compression.

Graph Sketch. These methods reduce the redundancy in the original graph, generating a skeleton graph that retains essential structural information. For instance, Graph-Skeleton [12] constructs a compact, synthetic, and highly-informative graph for the target nodes classification by eliminating redundant information in the background nodes. While this approach effectively reduces the memory usage for a pre-defined set of target nodes, it lacks flexibility and generalizability, as it must be tailored to specific nodes. In contrast, our SPGC performs one-time compression that applies universally to any set of test nodes $V_T \subset V$. NeutronSketch [35] focuses on eliminating the redundant information from the training portion of the graph, yet its compression is restricted to the training phase rather than accelerating inference, and does not ensure inference equivalence for the compressed graphs.

Our work differs from existing graph reduction approaches (summarized in Table. 1) in the following. (1) Our methods are model-agnostic and apply to any GNN that adopt the same inference process, without requiring model parameters, and incur no learning overhead. (2) We specify IFGC with variants that (approximately) preserve inferred results with invariant properties such as uniqueness and minimality, as well as fast compression and inference algorithms. These are not discussed in prior work. On the other hand, we remark that our scheme can be applied orthogonally: One can readily apply these approaches over compressed graphs from our method to further improve GNN training and inference. Our proposed SPGC can be potentially extended to the following learning settings: (1) transductive-learning: training on a compressed graph G_c while utilizing our proposed memoization structure $\mathcal{T}$ to enable accuracy-preserving inference; and (2) inductive-learning: applying semi-supervised or unsupervised learning techniques to refine the graph structure of G_c for a specific task.

2 GRAPHS AND GRAPH NEURAL NETWORKS

Graphs. A directed graph $G = (V, E)$ has a set of nodes V and a set of edges $E \subseteq V \times V$. Each node v carries a tuple $T(v)$ of attributes and their values. The size of G, denoted as $|G|$, refers to the total number of its nodes and edges, *i.e.*, $|G| = |V| + |E|$.

Graph Neural Networks. A graph neural network (GNN) $\mathcal{M}$ is a mapping that takes as input a featurized representation $G = (X, A)$ to an output embedding matrix Z, *i.e.*, $\mathcal{M}(G) = Z$. Here X is a matrix of node features, and A is a (normalized) adjacency matrix of G. [1]

[1] A feature vector X_v of a node v can be a word embedding or one-hot encoding [21] of $T(v)$. A is often normalized as $\hat{A} = A + I$, where I is the identity matrix.

Notation	Description
$G = (X, A)$	graph G, X: feature matrix, A: adjacency matrix
$\|G\|$	size of G; $\|G\| = \|V\| + \|E\|$
$\mathcal{M}$	a GNNs model
$\mathcal{M}(G)$ (resp. $\mathcal{M}(G, V_T)$)	output of $\mathcal{M}$ over G (resp. test set V_T)
$\mathcal{C}, \mathcal{P}$	compression, post-processing function
M_v	node update function
x_v^k	embedding of node v at layer k
G_c	compressed graph of G
α, r	similarity threshold, # hops
V_T, V_A	test node set, anchored nodes
$R^S, R^{(\alpha,r)}, R_L^A$	structural equivalence, (α, r) relation & anchored relation

Table 2: Summary of Notations.

Inference. We take a query language perspective [6, 22] to characterize the inference process of GNNs. A GNN inference process is specified as a composition of *node update functions*.

Node update function. Given a GNN $\mathcal{M}$ with L layers, a node update function M_v uniformly computes the embedding of each node v at each layer k ($k \in [1, L]$), with a general recursive formula as

$$x_v^k = M_v^k(\Theta^k, \text{AGG}(X_u^{k-1}, x_v^{k-1}, \forall u \in N^k(v)))$$

which is specified by (1) the learned model parameters Θ^k, (2) an aggregation function AGG (*e.g.,* $\sum$, CONCAT), and (3) the neighbors of v that participate in the inference computation at the k-th layer (denoted as $N^k(v)$). When k=1, X_v^0=$X_v \in X$, *i.e.,* the input features.

The *inference process* of a GNN $\mathcal{M}$ with L layers takes as input a graph $G = (X, A)$, and computes the embedding x_v^k for each node $v \in V$ at each layer $k \in [1, L]$, by recursively applying the node update function. A GNN $\mathcal{M}$ has a *fixed* inference process, if its node update function is specified by fixed input model parameters, layer number, and aggregator. It has a *deterministic* inference process, if $M(\cdot)$ always generates the same embedding for the same input.

We consider GNNs with fixed, deterministic inference processes. In practice, such GNNs are desired for consistent and robust performance. For simplicity, we assume that M_v specifies a proper set of neighbors that participate the inference process as $N(v) \subseteq \{u | (u, v) \in E \text{ or } (v, u) \in E\}$. This allows us to include GNNs that exploits neighborhood sampling (such as GraphSAGE), and directed message passing into discussion. In general, inferences of representative GNNs are in PTIME [13, 57] (see Table 3).

Classes of GNNs. We say a set of fixed, deterministic GNNs $\mathbb{M}$ belongs to a *class* of GNNs $\mathbb{M}^L$, if for every GNN $\mathcal{M} \in \mathbb{M}$, (1) $\mathcal{M}$ has L layers, and (2) $\mathcal{M}$ uses the same form of node update function M_v^k, for each node $v \in V$ and $k \in [1, L]$.

Table 3 summarizes several node update functions in their general forms for mainstream GNN classes. For example, Graph Convolution Networks (GCNs) [30] adopt a node update function as $X_v^k = \sigma(\Theta^k(\sum_{u \in N(v)} \frac{1}{\sqrt{d_u d_v}} x_u^{k-1}))$. Here d_u or d_v denotes the degree of node u or v. $\sigma(.)$ is the non-linear activation function. A class of GNNs GCN^3 contains 3-layered GCNs that adopt such node update function. Note that two GNNs in the same class can have different Θ and output, given the same input.

3 INFERENCE-FRIENDLY COMPRESSION

Given a graph $G = (V, E)$, a *compressed graph* of G, denoted as $G_c = (V_c, E_c)$, is a graph where (1) each node $[v] \in V_c$ is a nonempty subset of V, and $V = \bigcup_{[v] \in V_c} [v]$; and (2) there is an edge $([v], [v']) \in E_c$, if there are at least one node $v \in [v]$ and $v' \in [v']$, such that $(v, v') \in E$.

Note that $|V_c| \leq |V|$ and $|E_c| \leq |E|$. Hence, $|G_c| \leq |G|$.

Inference-friendly Graph Compression. Given a set of GNNs $\mathbb{M}$ and a graph G, an *inference-friendly graph compression*, denoted as IFGC, is a pair $(\mathcal{C}, \mathcal{P})$ where

- $\mathcal{C}$ is a compression function that computes a compressed graph G_c of G ($G_c = \mathcal{C}(G)$);
- $\mathcal{P}$ is a function that restore the auxiliary information of nodes in G_c to their counterparts in G; and moreover,
- $\mathcal{M}(G) = \mathcal{M}(\mathcal{P}(G_c))$, for *any* GNN $\mathcal{M} \in \mathbb{M}$.

An IFGC aims to generate a compressed graph G_c with a smaller size, such that an inference query that requests output $\mathcal{M}(G, V_T)$ for any $V_T \subseteq V$ can be computed by a faster inference process of $\mathcal{M}$ over G_c only, even with a query-time overhead incurred by $\mathcal{P}$.

A Sufficient Condition. We next introduce a sufficient condition for the existence of IFGC. To this end, we start with a notion of *inference-equivalent* relation.

Inference equivalence. Given a class of GNN $\mathbb{M}^L$ and a graph G, a pair of nodes (v, v') in G are *inference equivalent w.r.t.* $\mathbb{M}^L$, denoted as $v \sim_M^L v'$, if for any $M \in \mathbb{M}^L$, $X_v^k = X_{v'}^k$ for any $k \in [1, L]$.

One can readily infer that for any two nodes $v \sim_M^L v'$, $M(v, G) = M(v', G)$. That is, inference equivalence of nodes ensure that they are all "indistinguishable" for the inference of any GNN $\mathcal{M} \in \mathbb{M}^L$.

Denote the binary relation (v, v') induced by inference equivalence as R_M^L, *i.e.,* $(v, v') \in R_M^L$ if and only if $v \sim_M^L v'$. We say R_M^L is *nontrivial* if there is at least one pair $(v, v') \in R_M^L$, where $v \neq v'$. We can readily verify the following result.

Lemma 1: *Given $\mathbb{M}$ and G, the binary relation R_M^L is an equivalence relation,* i.e., *it is reflexive, symmetric, and transitive.* □

The *equivalent class* of v under an equivalence relation R_M^L, denoted as $[v]$, refers to the set $\{v' | (v, v') \in R_M^L\}$. The equivalent classes induced by the inference equivalence relation R_M^L forms a node partition V_R of V. The *quotient graph* induced by R_M^L is a graph G_R with nodes V_R and edges E_R, where each node in V_R is a distinct equivalent class induced by R_M^L, and there is an edge $([v], [v']) \in E_R$ if and only if there exists a node $v \in [v]$ and $v' \in [v']$, such that $(v, v') \in E$.

Lemma 2: *Given a class of* GNNs $\mathbb{M}^L$ *and a graph G, a graph compression scheme $(\mathcal{C}, \mathcal{P})$ is an* IFGC w.r.t. $\mathbb{M}^L$ *and G, if for any $\mathcal{M} \in \mathbb{M}^L$, (1) $\mathcal{C}(G)$ computes a quotient graph G_c induced by a nontrivial inference equivalent relation R_M^L* w.r.t. $\mathbb{M}^L$ *and G, and (2) $\mathcal{P}$ is a function that restores X_v^k with $X_{[v]}^k$ by a scaling factor derived from auxiliary information of v, for each layer $k \in [1, L]$.* □

Proof sketch: Let R_M^L be a non-empty inference equivalent relation *w.r.t.* $\mathbb{M}^L$ and G, and G_c be the quotient graph induced by $\mathbb{M}^L$. (1) Given **Lemma 1**, R_M^L is a nontrivial equivalence relation. Hence there exists at least one equivalent class $[v]$ with size larger

GNNs Classes	Node Update Function (general form)	Training Cost	Inference Cost
Vanilla [44]	$X_v^k = \sigma(\Theta \cdot \text{AGG}(X_u^{k-1}, \forall u \in \mathcal{N}(v)))$	$O(L\|E\| + L\|V\|)$	$O(L\|E\| + L\|V\|)$
GCN [13, 30]	$X_v^k = \sigma(\Theta^k(\sum_{u\in\mathcal{N}(v)} \frac{1}{\sqrt{d_u d_v}} x_u^{k-1}))$	$O(L\|E\|F + L\|V\|F^2)$	$O(L\|E\|F + L\|V\|F^2)$
GAT [9, 48]	$X_v^k = \sigma(\sum_{u\in\mathcal{N}(v)} \alpha_{uv}\Theta^k X_u^{k-1}))$	$O(L\|E\|dF^2 + L\|V\|F^2)$	$O(L\|E\|dF^2 + L\|V\|F^2)$
GraphSAGE [13, 23]	$X_v^k = \sigma(\Theta^k \cdot (X_v^{k-1}\|\|\text{AGG}(X_u^{k-1}, \forall u \in \mathcal{N}(v))))$	$O(L\|V\|dF + L\|E\|dF^2)$	$O(L\|V\|dF + L\|E\|dF^2)$
GIN [9, 51]	$X_v^k = \sigma(\text{MLP}((1+\gamma)x_v^{k-1} + \sum_{u\in\mathcal{N}(v)} x_u^{k-1})$	$O(L\|E\|F + L\|V\|F^2)$	$O(L\|E\|F + L\|V\|F^2)$

Table 3: Comparison of Representative GNNs with node update functions, training cost, and inference cost. σ: an activation function *e.g.,* ReLU or LeakyReLU. AGG: aggregation function; can be *e.g.,* sum ($\sum$), average (Avg), or concatenation (||). L, $|E|$, $|V|$, F, and d denote the number of layers, edges, nodes, features per node, and maximum node degree of G, respectively.

than one, *i.e.,* $|G_c| < |G|$. As function $\mathcal{P}$ does not introduce new node or edge to G_c, we have $|\mathcal{C}(G))| = |G_c| < |\mathcal{P}(G_c)| < |G|$. (2) To see $\mathcal{M}(G) = \mathcal{M}(\mathcal{P}(\mathcal{C}(G)))$, *i.e.,* G_c preserves inference result, it suffices to show that for every node $v \in G$, $\mathcal{M}(G, \{v\}) = M(\mathcal{P}(\mathcal{C}(G)), \{[v]\})$. This is ensured by (a) the fixed deterministic inference process that applies the same node update function M_v, and (b) $\mathcal{P}$ restores X_v^k with only $X_{[v]}^k$ and a scaling factor, for any layer $k \in [1, L]$. We list examples of $\mathcal{P}$ and scaling factors for mainstream GNNs in Table 4. Hence $(\mathcal{C}, \mathcal{P})$ is an IFGC. □

We next introduce practical IFGC for representative GNN classes, with efficient compression (implementing $\mathcal{C}$) and inference (involving $\mathcal{P}$) algorithms. We summarize notations in Table 2.

4 STRUCTURAL-PRESERVING COMPRESSION

We introduce a first IFGC for GNN inference. We specify R_M^L as an extended version of *structural equivalence*. The latter has origins in role equivalence in social science [37], and simulation equivalence of Kripke structures in model checking [4, 16]. By enforcing equivalence on embeddings and neighborhood connectivity, it ensures an IFGC to accelerate GNN inference *without* decompression.

4.1 Compression Scheme

Structural equivalence. Given a graph G=(X, A), a *structural equivalence* relation, denoted as R^S, is a non-empty binary relation such that for any node pair (v, v') in G, $(v, v') \in R^S$, if and only if:

- $X_v^0 = X_{v'}^0$, *i.e.,* v and v' have the same input features;
- for any neighbor u of v ($u \in N(v)$), there exists a neighbor u' of v' (u' in $N(v')$), such that $(u, u') \in R^S$; and
- for any neighbor u'' of v' in $N(v')$, there exists a neighbor u''' of v in $N(v)$, such that $(u'', u''') \in R^S$.

Structural-preserving Compression. Given a GNN class $\mathbb{M}^L$ and graph G, a *structural-preserving compression*, denoted as SPGC *w.r.t.* $\mathbb{M}^L$, is a pair $(\mathcal{C}, \mathcal{P})$ where (1) $\mathcal{C}$ computes G_c as the quotient graph of R^S, where R^S is the non-empty, maximum structural equivalence relation in G, and (2) $\mathcal{P}$ is a function that restores node embeddings with matching scaling factors for $\mathbb{M}^L$.

Example 2: Consider the graphs G_2 and G_3 in Fig. 2, and their compressed counterpart obtained by SPGC, G_{c_2} and G_{c_3}, respectively.

(1) G_2 has 10 nodes and 10 edges. The nodes having the same labels $'a', 'c', 'd'$ also have the same input features, respectively. For example, $X_{a_1}^0 = X_{a_2}^0$, and $X_{d_1}^0 = X_{d_2}^0 = X_{d_3}^0$. For nodes labeled with $'b'$, $X_{b_1}^0 = X_{b_3}^0 \neq X_{b_2}^0$. One can verify that $R^S = \{(b_1, b_3), (c_1, c_2), (d_1, d_3)\}$. A compressed graph G_{c_2} is illustrated with 7 nodes and 7 edges.

Observe that despite d_1, d_2 and d_3 have the same input features, $d_2 \not\sim_M^L d_1$, and $d_2 \not\sim_M^L d_3$ for GNNs with $L \geq 1$. Indeed, d_2 has a neighbor b_2 that has no counterpart in the neighbors of d_1 or d_3 that share the same embedding; hence the output embedding of d_2 may be different from either d_1 or d_3, and should be separated from equivalent class $\{d_1, d_3\}$ in G_c.

(2) G_3 is a cycle in the form of $\{c_n, b_n, a_n, \ldots, c_1, b_1, a_1\}$, where $X_{a_i}^0 = X_{a_j}^0$, $X_{b_i}^0 = X_{b_j}^0$, and $X_{c_i}^0 = X_{c_j}^0$, for any $i, j \in [1, n]$. We can verify that $R^S = \bigcup_{i,j\in[1,n]}\{(a_i, a_j), (b_i, b_j), (c_i, c_j)\}$. A smallest compressed graph G_{c_3} is illustrated with only three nodes: $A = \{a_i\}$, $B = \{b_i\}$, $C = \{c_i\}$, $\forall i \in [1, n]$, regardless of how large n is. □

The result below tells us that any two nodes that are structural equivalent are "indistinguishable" for GNN inference process.

Theorem 3: *Given a class of* GNNs $\mathbb{M}^L$ *and graph G, the relation R^S over G is an inference equivalence relation* w.r.t. $\mathbb{M}^L$. □

Proof sketch: First, R^S is an equivalence relation. It then suffices to show that for any pair $(v, v') \in R^S$, $v \sim_M^L v'$. We perform an induction on the number of layers k for GNNs. Consider a "matching" relation h between a pair $(v, v') \in R^S$, such that $h(v) = v'$. At any layer, for any node v and every neighbor $u \in N(v)$, there exists a "match" $h(v)$ and a "match" $h(u) \in N(h(v))$ with the same (intermediate) embedding. This ensures the equivalence of aggregated embedding computed by the node update function at v and $h(v)$, and vice versa. Hence for any pair $(v, v') \in R^S$, $v \sim_M^L v'$. R^S is thus an inference-friendly relation by definition. □

Following Lemma 2 and Theorem 3, SPGC is an IFGC.

Example 3: We also compare SPGC with several other possible graph compression scheme. We illustrate three more compressed graphs, G_{c2}^1, G_{c2}^2, G_{c2}^3 of G_2, following Exact Compression [10], Bisimulation [16], and Automorphism [50], respectively.

(1) Exact Compression [10] applies an iterative color refinement process[2] starting with groups that contains nodes agreeing on embeddings and color encoding (labels). It then iteratively split the groups, where each node is updated by concatenating its color encoding with those from their neighbors, and refine groups. It finally derives $G_{c_2}^1$ with 8 nodes and 9 edges after two rounds, where only node pairs d_1, d_3 and a_1, a_2 share the same node representation. The concatenation is more sensitive to the impact of *e.g.,* degrees, preventing more possible merge, hence less can be compressed.

(2) Bisimulation [16] ignores embedding equivalence and groups

[2]The original method is used to simplify GNNs learning problems [10]; we make a comparison by applying color refinement for graph compression alone.

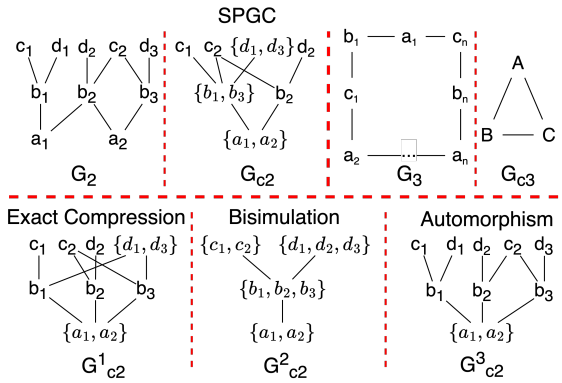


Figure 2: Compressing graphs with SPGC and a comparison with exact compression [10] , bisimulation [16], and automorphism (adapted from [50]). Exact compression uses node features for initial coloring and $d = 2$ for color refinement.

nodes only with topology-level equivalence. In this case, b_2 can be merged with b_1 and b_3 due to bisimulation in connectivity. This leads to smaller compressed graph $G^2_{c_2}$ with 4 nodes and 3 edges compared with G_{c_2}, yet at the cost of inaccurate inference at *e.g.*, nodes c_1, c_2, b_2, and d_2 due to "overly" compressed structure.

(3) Automorphism [50] partitions nodes into same automorphism equivalence sets, which poses strong topological equivalence on graph isomorphism in their neighbors. By enumerating the automorphism groups of G_2 and considering the embedding similarities, only a_1 and a_2 can be merged as shown in G^3_{c2} while other node pairs like c_1, d_1 or b_1, b_3 cannot be merged due to different embeddings or different connection patterns of the neighbors. This can be an overkill for reducing unnecessary inference computation. In addition, computing automorphism remains NP-hard, which indicates more expensive compression cost; while SPGC computes maximum structural equivalence in PTIME (see Section 4.3). □

4.2 Properties and Guarantees

We next justify SPGC by showing a *minimality and uniqueness* property. We show that an SPGC can generate a smallest G_c, which is "unique" up to graph isomorphism. That is, if there is another smallest compressed graph G'_c generated by an SPGC, then G^*_c and G'_c are isomorphic.

Lemma 4: *Given a* GNN *class* $\mathbb{M}^L$ *and* G*, there is an* SPGC *that computes a smallest* G^*_c *which is unique up to graph isomorphism.* □

Proof sketch: We show the minimality property by a constructive proof as follows: (1) given $\mathbb{M}^L$ and G, there exists a unique, largest inference equivalence relation R^{*S}; (2) we construct an SPGC that computes G^*_c as the quotient graph of R^{*S}. The uniqueness of G^*_c can be shown by a contradiction: if there exists another smallest G'_c that is not graph isomorphic to G^*_c, then either G'_c is not smallest in sizes, or R^{*S} is not the (largest) inference equivalence relation, *i.e.,* there is a pair (v, v'), such that either $v \sim^R_M v'$ but are not in $[v]$, or $v \nsim^R_M v'$, but are included in $[v]$. Either leads to contradiction. □

We next justify SPGC by showing that it properly preserves the discriminative set of GNNs, which has been used as one way to characterize the expressiveness power of GNNs as queries [6, 22].

Algorithm 1 : SPGC

Input: Graph G, node feature matrix X, a class of GNNs $\mathbb{M}^L$ with node update function M_v;
Output: A compressed graph G_c with memoization structure $\mathcal{T}$;
1: set $R^S := \emptyset$; set $EC := \{V\}$; set $\mathcal{T} := \emptyset$; graph $G_c := \emptyset$;
2: $R^S :=$ DPP(G);
3: $R^S := R^S \setminus \{(v, v') \mid X^0_v \neq X'^0_v\}$;
4: $EC := V/R^S$; /* induce partition EC from refined R^S*/
5: $(G_c, \mathcal{T}) :=$ CompressG $(\mathcal{T}, G_c, EC, G, M)$;
6: **return** G_c and $\mathcal{T}$;

Figure 3: Algorithm SPGC

Discriminative set of GNNs [22]. Given a set of graphs $\mathcal{G}$, the *discriminative set* of a GNN $\mathcal{M}$, denoted as $\mathcal{G}_\mathcal{M}$, refers to the maximum set of pairs $\{(G, G')\}$, where $G, G' \in \mathcal{G}$, such that $M(G) = M(G')$. In the case of equivariant GNNs [43], the strongest discriminativeness can be achieved, for which the set contains all pairs (G, G') such that G and G' are isomorphic [5]. In other words, these GNNs can "solve" graph isomorphic problem: one can issue a Boolean inference query to test if an input pair of graphs are isomorphic.

Given a set of graphs $\mathcal{G}$, denote the set of corresponding compressed graphs generated by SPGC as $\mathcal{G}_c$, *i.e.,* $\mathcal{G}_c = \{G_c | G_c = \mathcal{C}(G); G \in \mathcal{G}\}$. We have the following result.

Lemma 5: *Given* $\mathbb{M}^L$ *and a set of graphs* $\mathcal{G}$*, an* SPGC *can compute a compressed set* $\mathcal{G}_c$*, such that for every* GNN $\mathcal{M} \in \mathbb{M}^L$*, and any pair* $(G, G') \in \mathcal{G}_\mathcal{M}$*, there exists a pair* $(G_c, G'_c) \in \mathcal{G}_{c_\mathcal{M}}$. □

This result tells us that SPGC "preserves" the discriminativeness of GNNs. Moreover, it suggests a practical compression scheme for large-scale graph classification. One can apply SPGC to compress $\mathcal{G}$ to a smaller counterpart $\mathcal{G}_c$. As the discriminativeness set is preserved over $\mathcal{G}_c$ for every GNN $M \in \mathcal{M}^L$, SPGC reduces the overall classification overhead, via a post-processing $\mathcal{P}$ that readily groups $\mathcal{G}$ by corresponding label groups over $\mathcal{G}_c$.

Due to limited space, we present the detailed proofs in [1].

4.3 Compression Algorithm

We next present a compression algorithm (function $\mathcal{C}$) in SPGC.

General idea. The algorithm, simply denoted as SPGC, follows Lemma 4 to construct the smallest G^*_c induced by the maximum structure equivalence relation R^*_S. To ensure efficient inferences that only refer to G_c without decompression, it (1) uses a *memoization* structure $\mathcal{T}$ to cache the neighborhood statistics specified by node update function M_v, and (2) *rewrites* M_v to an equivalent counterpart $M_{[v]}$ (see Table 4 for examples), such that the inference can directly process on each $[v]$ in G_c, and "looks up" $\mathcal{T}$ at runtime, to obtain the embeddings for all the nodes in $[v]$, in a single batch.

Compression Algorithm. The SPGC algorithm, as illustrated in Fig. 3, takes as input a featurized input $G = (X, A)$ and a GNN class $\mathbb{M}^L$ with node update function M_v. (1) It first extends Dovier-Piazza-Policriti (DPP) algorithm [16] to compute the maximum structural equivalence relation R^*_S, by enforcing embedding equivalence as an additional equivalence constraint (lines 2-4). This induces a set of equivalence classes EC (a node partition). It then invokes a

Algorithm 2 Procedure CompressG($\mathcal{T}$, G_c, EC, G, M)

1: **for** $[v] \in EC$ **do**
2: $V_c = V_c \cup \{[v]\}$;
3: initialize $[v]_T$; /* with row pointers as $v \in [v]$*/
4: **for** edge $(u, v) \in E$ **do**
5: $E_c = E_c \cup \{([u], [v])\} \mid u \in [u], v \in [v]$;
6: **if** $M.\phi$ is topology sensitive **then**
7: $[v]_T(v, [u])$ += $\frac{1}{\sqrt{deg(u)}}$;
8: **else if** $M.\phi$ is weight sensitive **then**
9: $[v]_T(v, [u])$ += $\alpha_{v,u}$;
10: **else**
11: $[v]_T(v, [u])$ += 1;
12: $\mathcal{T} = \bigcup_{[v] \in V_c} [v]_T$;
13: **return** G_c and $\mathcal{T}$;

Figure 4: Procedure CompressG

procedure CompressG to construct G_c as the quotient graph of R^*_S, as well as the memoization structure $\mathcal{T}$ (line 5).

Procedure CompressG. CompressG is a light-weighted compression approach enabling an easy-to-implement and feasible way to derive compressed graph G_c from large-scale G while computing the memoization structure $\mathcal{T}$. Given the induced equivalence classes EC and an encoding of the node update function M_v, procedure CompressG (illustrated in Fig. 4), generates the compressed graph G_c and memoization structure $\mathcal{T}$. For each equivalent class $[v]$ in EC, CompressG initializes a node in G_c, (lines 1-3). For each edge $(u, v) \in E$, CompressG adds an edge $([u], [v])$ (lines 4-5).

Compression time memoization. CompressG dynamically maintains a memoization structure $\mathcal{T}$ that is shared by all GNNs in $\mathcal{M}^L$, to cache useful auxiliary neighborhood information used by the node update function for efficient inference (see Section 4.4). For each node $[v] \in V_c$, it assigns $[v]_{\mathcal{T}}$, a compact table, such that for every $v \in [v]$, and every neighbor $[u] \in N([v])$, an entry $[v]_{\mathcal{T}}(v, [u])$ records an aggregation of auxiliary neighborhood information (*e.g.*, sum of node degree, edge weights) of $N(v) \subseteq N([v])$.

When processing an edge $(u, v) \in E$, it follows a case analysis of $\mathbb{M}^L$ with node update function M_v. For example, (1) "topology sensitive" means the degrees of neighbors of v is required, as seen in GCNs; (2) "weight sensitive" means additional edge weights, such as edge attentions in GATs. For GATs, the edge weights from pre-trained M are optional such that if given the information of all edge weights, the inference accuracy can be preserved based on the formula shown in Table. 4.4. Such information can be readily obtained by tagging the input GNN class $\mathbb{M}^L$ or encoded as rules. It then updates the entry $[v]_{\mathcal{T}}(v, [u])$ accordingly (lines 6-12).

Example 4: Consider a GNNs class GIN, and graph G_4 shown in Fig. 5. (1) SPGC invokes the DPP algorithm to compute R^S. It next refines R^S based on feature embeddings and returns the induced partition $EC = \{[a], [b], [c], [d]\}$, where nodes with same labels are merged, *e.g.*, $[a] = \{a_1, a_2\}$. (2) We illustrate how CompressG dynamically updates the memoization structure $\mathcal{T}$ by considering the processing of two edges (b_1, a_1) and (b_2, a_1). For $[a] \in EC$, it first initializes $[a]_{\mathcal{T}}$ as an empty table. It next iterates over edges in

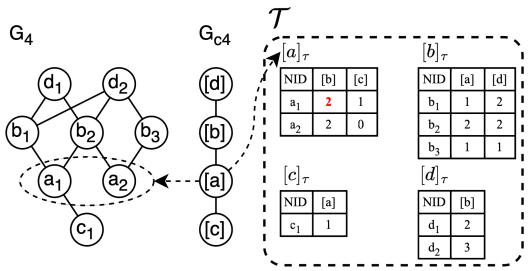


Figure 5: Run-time generation of Memoization structure $\mathcal{T}$.

E. As the node update function of GIN does not require exact degree or additional edge weight (not topology or weight sensitive), the entry $[a]_T(a_1, [b])$ is updated to 1, to "memoize" that there is one neighbor of a_1 in $N([b])$ that will contribute to a "unit" value to the embedding computation of a_1, via edge (b_1, a_1). Similarly, when it reaches edge (b_2, a_1), $[a]_T(a_1, [b])$ is updated to 2. Following this processes, all entries in $\mathcal{T}$ will be updated to memoize neighbors' information while compressing the graph. □

Correctness and cost. SPGC correctly computes G^*_c as ensured by (1) the correctness of DPP algorithm and (2) the follow-up refinement by enforcing embedding equivalence. For time cost, it takes SPGC $O(|V| + |E|)$ time to initialize R^S with DPP algorithm. The refinement of R^S and EC takes $O(|V|)$ time (lines 3-4). Procedure CompressG processes each equivalent class in EC ($|EC| \leq |V|$) and each edge in G once, hence in $O(|V| + |E|)$ time to construct G^*_c and update $\mathcal{T}$. The total cost is thus in $O(|V| + |E|)$ time.

4.4 Inference Process

Inference algorithm. We outline an algorithm that directly obtains $M(G)$ by referring to G^*_c only, without decompression. Our strategy rewrites the node update function M_v to an equivalent counterpart $M_{[v]}$, that takes as input $[v]$ and the corresponding tuple $[v]_{\mathcal{T}}(v)$ in $\mathcal{T}$, to "scale" the embedding computation with the memorized edge weights. The algorithm performs inference directly in G^*_c with $M_{[v]}$, and simply "scale up" the results at $[v]$ for each node $v \in [v]$ with a scaling factor. The scaling factor can be directly looked up from the table $[v]_{\mathcal{T}}(v)$ (function $\mathcal{P}$). Table 4 illustrates the scaling factors for mainstream GNN classes.

Example 5: Continuing with Example. 4, an inference at node a_1 looks up, in constant time, the values from entries $[a]_T(a_1, [b])$ and $[a]_T(a_1, [c])$ which are 2 and 1 separately (as shown in Fig. 5). It next assigns the values as scaling factors (Table. 4) to restore messages and the embedding of node a_1 as in original G_4. □

Inference cost. As SPGC requires no decompression on neighborhood structures of nodes, an inference query can be directly applied to G_c without incurring additional overhead. The overall inference cost is in $O(L|E_c|F + L|V_c|F^2)$. We remark that this result is derived by scaling down a common upper bound of inference costs for mainstream GNNs in Table 3. For other and more complex GNNs variants, the inference costs can be derived similarly by scaling down from their counterparts over G.

GNNs	Node Update Function M_v	equivalent rewriting $M_{[v]}$; scaling factors are marked in red	notes
Vanilla [44]	$X_v^k = \sigma(\Theta \cdot \text{AGG}(X_u^{k-1}, \forall u \in \mathcal{N}(v)))$	$X_v^k = \sigma(\Theta \cdot \text{AGG}(\mathbf{[v]_T(v,[u])} X_{[u]}^{k-1}, \forall [u] \in \mathcal{N}([v])))$	AGG: $\sum$ or AVG; for AVG, need to multiple by RF_v
GCN [30]	$X_v^k = \sigma(\Theta^k(\sum_{u\in\mathcal{N}(v)} \frac{1}{\sqrt{deg_u deg_v}} x_u^{k-1}))$	$X_v^k = \sigma(\Theta^k(\sum_{[u]\in\mathcal{N}([v])} \frac{1}{\sqrt{\mathbf{deg_v}}} \mathbf{[v]_T(v,[u])} x_{[u]}^{k-1}))$	deg_v: degree of node v in G, topology sensitive
GAT [48]	$X_v^k = \sigma(\sum_{u\in\mathcal{N}(v)} \alpha_{uv} \Theta^k X_u^{k-1}))$	$X_v^k = \sigma(\sum_{[u]\in\mathcal{N}([v])} \mathbf{[v]_T(v,[u])} \Theta^k X_{[u]}^{k-1}))$	weight sensitive
GraphSAGE [23]	$X_v^k = \sigma(\Theta^k \cdot (X_v^{k-1} \Vert \text{AGG}(X_u^{k-1}, \forall u \in \mathcal{N}(v))))$	$X_v^k = \sigma(\Theta^k \cdot (X_{[v]}^{k-1} \Vert \text{AGG}(\mathbf{RF_v} \times \mathbf{[v]_T(v,[u])} X_{[u]}^{k-1}, \forall [u] \in \mathcal{N}([v]))))$	$\Vert$: concatenation; AGG: AVG
GIN [51]	$X_v^k = \sigma(\text{MLP}((1+\gamma)x_v^{k-1} + \sum_{u\in\mathcal{N}(v)} x_u^{k-1}))$	$X_v^k = \sigma(\text{MLP}((1+\gamma)x_{[v]}^{k-1} + \sum_{[u]\in\mathcal{N}([v])} \mathbf{[v]_T(v,[u])} x_{[u]}^{k-1}))$	

Table 4: Rewriting of node update functions for mainstream GNN classes (scaling factors highlighted in red).

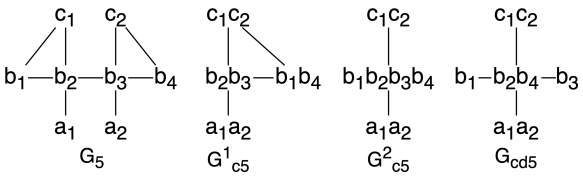


Figure 6: Compressing G_5 with $(0.5, 2)$-SPGC.

5 CONFIGURABLE GRAPH COMPRESSION

While SPGC generates G_c that can be directly queried by inference queries without decompression, it enforces node embedding equivalence, which may be an overkill for nodes with similar embeddings and can be processed in a single batch with tolerable difference in query outputs. Users may also want to *configure* the compression schemes to balance among accuracy and speed up, or to contextualize the compression with inference queries that specifies a set of test nodes $V_T \subseteq V$ of interests, such that $M(G, V_T) = M(G_c, V_T)$.

In response, we next introduce two variants of SPGC: (α, r)-SPGC (Section 5.1), and anchored SPGC (Section 5.2), respectively.

5.1 Compression with Structural and Embedding Similarity

We start with a relation called (α, r)-relation, which approximates R^S by lifting its equivalence constraints.

(α, r)-relation. Given graph G, a configuration (xsim, α, r) is a triple where $\text{xsim}(\cdot)$ is a *feature similarity function* that computes a similarity score of two node embeddings, α is a similarity threshold ($\alpha \in [0, 1]$), and r an integer. Let $N_r(v)$ be the nodes within r-hop neighbors of v. A binary relation $R^{(\alpha,r)} \subseteq V \times V$ is an (α, r)-relation if for any node pair $(v, v') \in R^{(\alpha,r)}$,

- $\text{xsim}(X_v^0, X_{v'}^0) \geq \alpha$;
- for any node $u \in N(v)$, there exists a node $u' \in N_r(v'))$, such that $(u, u') \in R^{(\alpha,r)}$; and
- for any node $u'' \in N(v')$, there exists a node $u''' \in N_r(v)$, such that $(u'', u''') \in R^{(\alpha,r)}$.

Note that $R^{(1,1)}$ is an R^S, as $\alpha = 1$ ensures embedding equivalence, and $r = 1$ preserves indistinguishable neigbhors for node update functions in GNN inference. On the other hand, (α, r)-relation is no longer an equivalence relation, as it "relaxes" structural equivalence by lifting both embedding equality, and the strict neighborhood-wise equivalence, in trade for better compression ratio. We next clarified the relationship between r and L. Based on the definition of SPGC, the embedding similarity between two vertices is dependent on r instead of L. In other words, SPGC is model-agnostic as long as $r \leq L$. In practice, since L is small due to over-smoothing issue of larger L, r is usually a small integer like 1 or 2.

Based on the relation $R^{(\alpha,r)}$, we introduce a variant of SPGC.

(α, r)-SPGC. Given a graph G, and a configuration α and r *w.r.t.* an embedding similarity measure and a threshold, an (α, r)-SPGC is a graph compression scheme if C computes a graph G_c induced by the relation $R^{(\alpha,r)}$. Specifically,

- for any node pair $(v, v') \in R^{(\alpha,r)}$, $v \in [v]$, $v' \in [v]$; and
- there is an edge between $([u], [v])$ if $(u, v) \in E$.

Lemma 6: *Given a* GNN *class $\mathcal{M}^L$ and a graph G, an (α, r)-*SPGC *incurs compression cost in $O(|V||N_r| + |E|)$ time ($|N_r|$ refers to the largest size of r-hop neighbors of a node in G), and an inference cost in $O(L|E|F + L|V_c|F^2)$ time.* □

As a constructive proof, we next introduce algorithms that implements (α, r)-SPGC with the above guarantees.

Compression Algorithm. We describe the compression algorithm (α, r)-SPGC. It follows the same principle to compute (α, r)-relation and induce a compressed graph. The difference is that rather than inducing equivalence class and quotient graph from G, (1) it first induces a graph G_r by linking nodes to their r-hop neighbors, (2) it then computes an R^S relation by invoking DPP algorithm, and refines it by a re-grouping of nodes determined by similarity function $\text{xsim}(\cdot)$ with α as similarity threshold. (3) It generates $[v]$ to include all the pairs (v, v') in $R^{(\alpha,r)}$, and accordingly the edges. It updates the memoization structure $\mathcal{T}$ following the edges in G_r, similarly as in SPGC. Here $\mathcal{T}$ caches the statistics from the r-hop neighbors of each node v in the original graph G.

Example 6: Consider graph G_5 shown in Fig. 6. A $(0.5, 2)$-SPGC invokes DPP to initialize a $(1, 1)$-relation, and refines it to $R^{(0.5,2)} = \{(a_1, a_2), (c_1, c_2), (b_1, b_2), (b_1, b_3), (b_1, b_4), (b_2, b_3), (b_2, b_4), (b_3, b_4)\}$. This yields a compressed graph $G_{c_5}^2$ with only 3 nodes and 2 edges.

We also illustrate $G_{c_5}^1$, a compression graph from SPGC for G_5 (induced by an R^S as a $(1, 1)$-relation). Due to strictly enforced embedding equivalence, b_1 and b_2 cannot be merged, and similarly for b_3 and b_4. This yields $G_{c_5}^1$ with more nodes and edges. □

Compression Cost. It takes $O(|V| \cdot |N_r(v)|)$ time to derive G_r for $v \in V$. It then takes $O(|V| + |E_r|)$ to compute and refine $R^{(\alpha,r)}$, for G_r with edge set E_r. Here $|E_r| \leq |V| \cdot |N_r|$, where N_r refers to the largest r-hop neighbor set for a node in G. Procedure CompressG constructs G_c in $O(|V| + |E_r|)$ time, and generates memoization structure $\mathcal{T}$ in $O(|E|)$ time. The total cost is thus in $O(|V||N_r| + |E|)$.

As (α, r)-SPGC is specified by $R^{(\alpha,r)}$ that approximates an inference-friendly relation, it is no longer an IFGC, hence a direct inference over the G_c from it may not preserve the original output. To mitigate accuracy loss, the inference specifies a procedure $\mathcal{P}$ to perform run-time decompression with small overhead.

Inference process with decompression. The inference algorithm directly processes each node $[v]$ in G_r as in SPGC. For (α, r)-SPGC. the difference is that it ad-hocly invokes a decompression procedure decompG (the decompG algorithm and its example illustrated in Figure. 18 are shown in the Appendix [1]) to reconstruct the neighbors of v in G_r, and performs an inference using original 1-hop neighbors of v to obtain an embedding X_v as close as its original counterpart in G_r. To minimize decompression cost, when compressing the graph, algorithm (α, r)-SPGC (shown in Appendix B) incorporates *Re-Pair*, a reference encoding method [15, 32], to derive AL_c and rules for later fast decompression from a compact encoding structure. Specifically, within each *EC*, procedure decompG sorts the processing order of nodes by their degrees in G_r. In other words, procedure decompG prioritizes the decompression of nodes having the most shared r-hop neighbors with others in G_r, to (1) maximally reduce redundant computation in decompression process, and (2)"maximize" the likelihood for more accurate inference computation. For example, a partially decompressed graph G_{cd_5} that resolves 1-hop neighbors of b_2 (in trade for more accurate embedding) is illustrated in Fig. 6 (please refer to Example. 8 and Figure. 18 in Appendix for details [1]). The decompressed neighbors are kept in G_{cd} until the inference terminates.

As the decompression restores at most $|E|$ edges, the overall inference process takes $O(L|E|F + L|V_c|F^2)$ time, including the decompression overhead. We present the details of decompression algorithm in [1]. The above analysis completes the proof of Lemma 6.

5.2 Anchored Graph Compression

We next introduce our second variant of SPGC, notably, *anchored* SPGC, which permits a decompression-free, inference friendly compression, *relative* to a specific set of nodes of interests.

We present our main result below.

Theorem 7: *Given $\mathcal{M}^L$ and G with a set of targeted nodes V_A, there exists an* IFGC *that computes a compressed graph in $O(|G_L|)$ time to preserve the inference output for every node in V_A at an inference cost in $O(L|E_c|F + L|V_c|F^2)$ time. Here $|G_L|$ refers to the subgraph of G induced by L-hop of V_A, and $|V_c|$ and $|E_c|$ are bounded by $|G_L|$.* □

As a constructive proof, we introduce a notion of anchored relation, and construct such an IFGC. Given a graph G with a set of designated targeted nodes V_A, and a class of GNNs $\mathcal{M}^L$, a graph compression scheme $(C, \mathcal{P})$ is an IFGC relative to V_A, if (1) $|\mathcal{P}(C(G))| < |G|$; and (2) for any GNN $M \in \mathcal{M}^L$, and any $v \in V_A$, $M(G, \{v\}) = M(\mathcal{P}(C(G), \{v\})$.

Anchored relation. Given graph G, an integer L, and a designated *anchor set* $V_A \subseteq V$, we define the L-hop neighbors of V_A, denoted as $N_L(V_A)$, as $\bigcup_{v \in V_A} N_L(v)$, where $N_L(v)$ refers to the set of nodes within L-hop of v in G. An *anchored relation* R_L^A *w.r.t.* V_A refers to the structural equivalence relation defined over the subgraph G_L of G induced by $N_L(V_A)$.

One may verify that (1) R_L^A is an equivalence relation over V, and (2) $R^S = R_L^A$ if $V_A = V$, and L is larger than the diameter of G.

Anchored SPGC. Given graph G and an anchor set V_A from G, an anchored SPGC, denoted as ASPGC, is a graph compression where C computes a compressed graph that is the quotient graph of R_L^A. The computation of compressed graph G_c using ASPGC follows its counterpart in SPGC. The difference is that it induces a subgraph G_L of G with the L-hop neighbors of all the nodes in V_A. It then invokes the compression algorithm of SPGC to derive R_L^A and applies the compression algorithm of SPGC on G_L to compute the G_c and memoization structure $\mathcal{T}$. The inference process over G_c, similarly, follows its SPGC counterpart over G_c, which consistently leverages $\mathcal{T}$ to efficiently recover the auxiliary information of neighborhoods. Hence, it preserves the inference results of the nodes in V_A for GNNs classes with up to L layers. In practice, one may set V_A simply as a set of test nodes V_T to adapt ASPGC for specific inference queries. We present details and an example in [1].

Analysis. The correctness of ASPGC follows from the data locality of L-layered GNN inference when V_A is specified, which only involves the subgraph G^L of G induced by L-hop neighbors of V_A. ASPGC next follows SPGC to correctly compute G_c from G_L. For compression cost, it takes $O(|N^L(V_A)|+|E|)$ time to induce G_L, and $O(|V_L| + |E_L|)$ time to construct G_c from G_L. (3) The inference cost is consistently $O(L|E_c|F + L|V_c|F^2)$, where both $|E_c|$ and $|V_c|$ are bounded by $|G_L|$.

Given the above analysis, Theorem 7 follows.

6 PARALLEL INFERENCE

SPGC and its variants go through the following modules and algorithms to parallelize inference, as illustrated in Fig. 7. (1) **Coordinator**. Given the configuration tuple (M^L, V_T) provided by the user, coordinator conducts the following steps: (1) retrieves the corresponding G and $\mathcal{T}$ from the compressor; (2) generates parallel joblets $< \mathcal{T}_i, G_{c_i}, M^L >$ and sends them to processor P_i for all $v_i \in V_T$. (2) **Compressor**. Given a graph G and a class of GNNs $\mathcal{M}^L$, the compressor produces the compressed graph G_c and the memoization structure $\mathcal{T}$. It performs offline "once-for-all" graph compression by leveraging a built-in library of compression methods, including our structural-preserving compression SPGC as well as the state-of-the-art graph compression approaches [31, 36]. (3) **Parallel Inference**. Given the joblet $< \mathcal{T}_i, G_{c_i}, M^L >$, the module computes the inference result $M^L(v_i, G_{c_i})$ in parallel.

Coordinator. The coordinator optimizes query workload with three components. (1) *Partitioner*: Upon receiving an inference query (M^L, V_T), it initializes a set of processors $\mathcal{P}$. At each $P_i \in \mathcal{P}$, it partitions V_T, the memoization table $\mathcal{T}$ to $\mathcal{T}_i$, and induces G_{c_i} accordingly to $v_i \in V_T$. (2) *Load Balancer*: It assigns joblets to processors by distributing query workload evenly among the processors, preventing bottlenecks. This approach minimizes the total time cost of inference on the entire set of V_T. (3) *Job Scheduler*: It reschedules joblets $< \tau_i, G_{c_i}, M^L >$ based on estimated workload and current processor status, and allocate computational resources to dynamically rebalance the parallel inference computation.

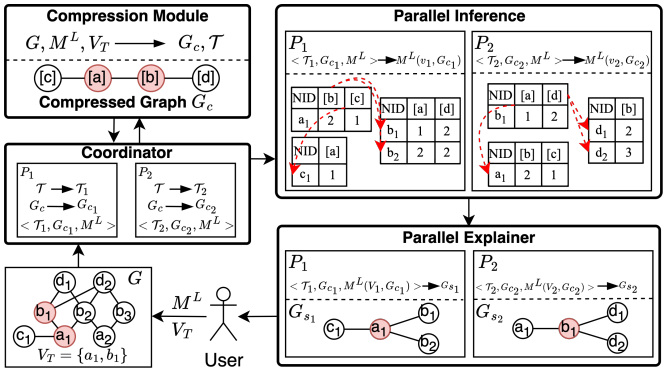


Figure 7: Parallel Inference and Explainer with a running example: $M^L \in \text{GCN}^2$; $V_T = \{a_1, b_1\}$; $\mathcal{T}_1$, $\mathcal{T}_2$: partitioned subsets of $\mathcal{T}$ for a_1 and b_1; G_{s_1}, G_{s_2}: explanation graphs for a_1 and b_1.

Pipelining. At each processor P_i, given a joblet $< \mathcal{T}_i, G_{c_i}, M^L >$, it performs the inference for assigned $v_i \in V_T$ in parallel to generate output $M^L(v_i, G_{c_i})$. The inference joblets follow a pipelined parallelism among all processes, to ensure that users receive inference result incrementally, rather than waiting or all inference queries to be evaluated.

Parallel Time Cost. The parallel cost per inference query is $O(\frac{L|G_c|F^2}{n})$ where L is the number of layers and F is number of features per node in G_c, and n the number of processors.

7 EXPERIMENTAL STUDY

Using both real-world graph datasets and large synthetic graphs, we conducted four sets of experiments, to understand (1) effectiveness of our compression methods, in terms of compression ratio, and the trade-off between inference cost and accuracy loss; (2) their efficiency, in terms of the compression cost and inference cost, (3) impact of critical factors, and (4) an ablation study to evaluate the effectiveness of memoization, and decompression overhead.

7.1 Experimental Settings

Datasets. We employ four real-world graph benchmark datasets (summarized in Table. 5): (1) **Cora** [39], a citation network where nodes represent documents, and edges are citations among the documents. Node features are described by a 0/1-valued word vector indicating the absence/presence of the corresponding word from the dictionary; (2) **Arxiv** [27], a citation network with nodes representing arXiv papers and edges denoting one paper cites another one. The features of each node includes a 128-dimensional feature vector obtained by averaging the embeddings of words in its title and abstract; (3) **Yelp** [53] is prepared from the raw json data of businesses, comprising a dense network of user-business interactions. Nodes represent users and edges are generated if two users are friends. Node features are the word embeddings derived by reviews of products; and (4) ***Products*** [27], a product co-purchase network, where nodes represent products sold in Amazon and edges denote products purchased together. Node features are generated by extracting embeddings from the product descriptions followed by a Principal Component Analysis reducing the dimension to 100. The size of test nodes $|V_T| = 0.05 * |V|$ across all datasets. Note that $|V_A| = |V_T|$ since by default $V_A = V_T$ in ASPGC.

Dataset	|V|	|E|	# node types	# attributes
Cora	2,708	5,429	7	1,433
Arxiv	169K	1.2M	40	128
Yelp	717K	7.9M	100	300
WS-MAG	2M	8M	153	768
Products	2.4M	61.9M	47	100

Table 5: Summary of Datasets.

Besides real-world benchmark datasets, we also generated a large synthetic dataset **WS-MAG**, by extending a core of real MAG240M network [26] (a citation network) with a small world generator [49].

Graph Neural Networks. We have pre-trained three classes of representative GNNs: GCNs[3] [30], GATs[3] [48], and GraphSAGE[3] [23], for each dataset. For a fair comparison, (1) for all the datasets, we consider node classification, and (2) all compression methods are applied for the same set of GNNs

Compression Methods. We compare SPGC and its variants, (α, r)-SPGC and ASPGC, with two state-of-the-art compression methods: (1) DSpar [36], a graph sparsification method that performs edge down-sampling to preserve graph spectral information, and (2) FGC [31], a latest learning-based graph coarsening approach that learns a coarsened graph matrix and feature matrix to preserve desired graph properties, such as homophily. We are aware of other learning-based approaches, yet they are model-specific and require the learned model parameters. Our work is orthogonal to these methods, and is not directly comparable. When conducing the experiments using DSpar and FGC, we set the longest waiting limit = 5 hours which means if the compression graph cannot be generated after 5 hours. We simply exclude them since the compression cost is already comparable to training a new GNNs model.

Evaluation Metrics. Given graph G, a class of GNNs $\mathcal{M}^L$, a graph compression scheme $(C, \mathcal{P})$ that computes a compressed graph G_c, and its matching inference process over G_c, we use the following metrics. (1) For efficiency, we evaluate (a) the time cost of compression, and (b) the speed-up of inference, which is defined by $\frac{T_{MG}}{T_{MC}}$, where T_{MG} refers to the inference time cost over G, and T_{MC} represents its counterpart over G_c. (2) For effectiveness, we report (a) a normalized compression ratio, which is defined as $\text{ncr} = 1 - \frac{|G_c|}{|G|}$; Intuitively, it quantifies the fraction of G that is "reduced": the larger, the better; and (b) the model performance quantified by accuracy and $F1$-score over G_c. In particular for Yelp, over which the benchmark task is a multi-class node classification, we report micro $F1$-score. We report the average performance of 200 inference tests, for each GNN model over each dataset.

Environment. SPGC and its variants are developed in Python with PyTorch Geometric [19] and BisPy [3] libraries. All tests are conducted on 4 Intel(R) Xeon(R) Silver 4216 CPU @ 2.10GHz, 128 GB Memeory, 16 cores, and 1 32GB NVIDIA V100 GPU. Our source code, datasets, and a full version of the paper are made available[3].

[3]https://github.com/Yangxin666/SPGC

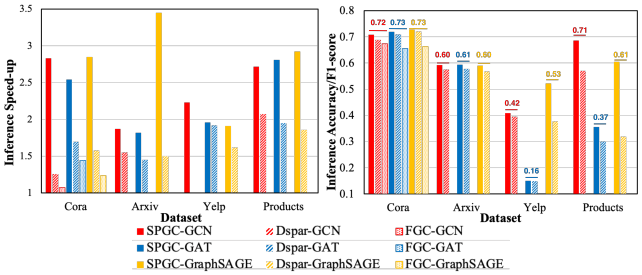
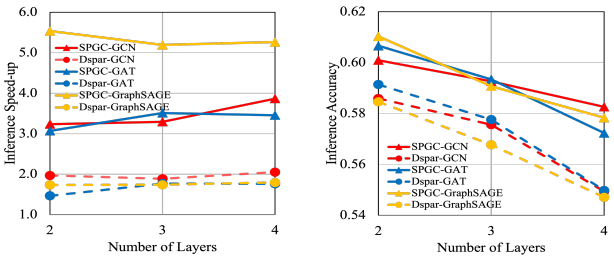


Figure 8: Comparison of $(0.5, 1)$-SPGC with the Baselines in Inference Speed-up and Inference Accuracy/F1-score. (the inference accuracy of ASPGC marked in colored lines, *e.g.,* the red lines show inference accuracy of ASPGC using GCN; the inference accuracy on original graph G marked in corresponding colors for difference GNNs, *e.g.,* the blue 0.73 indicates the inference accuracy of GAT on original G of Cora).



(a) Num. Layers v.s. Speed-up **(b) Num. Layers v.s. Accuracy**

Figure 9: Varying Num. Layers in GNNs (Arxiv).

7.2 ExperimentalResults

Exp-1: Effectiveness: Accuracy vs. Speed-up. Fig. 8 compares $(0.5, 1)$-SPGC with DSpar and FGC, in terms of inference speed-up in left figure and inference accuracy/F1-score in the right figure, over all four real datasets. Note that FGC can only generate results for **Cora** since its compression cannot be completed within 5-hours for all other datasets. Here a test annotated as "compression method-GNN" refers to the setting that a GNN inference is applied on a compressed graph generated by the method. (1) $(0.5, 1)$-SPGC outperforms DSpar and FGC across all four datasets for all the GNN classes on inference speed-up. It can improve the inference efficiency better over larger graphs. For example, for ***Arxiv***, $(0.5, 1)$-SPGC achieves a speed-up of 3.4 for inference with GraphSAGE, while DSpar achieves a speed-up to 1.5. (2) Consistently, we found that SPGC achieves higher ncr than DSpar and FGC. For example, for **Cora**, SPGC achieves ncr up to 74.5% while DSpar and FGC achieves 46.2% and 30.7% respectively. (3) SPGC outperforms DSpar and FGC in terms of its ability to preserve inference results for GNNs. We observe that i) $(0.5, 1)$-SPGC achieves inference accuracy comparable to direct inference on the original graph G across all four datasets (as illustrated in Fig. 8); and ii) $(0.5, 1)$-SPGC retains highest inference accuracy/F1-scores across all datasets and models. For example, for **Cora**, $(0.5, 1)$-SPGC achieves 0.71 accuracy which outperforms 0.68 and 0.67 achieved by DSpar and FGC.

Exp-2: Effectiveness: Impact of Factors. We first investigate the impact of number of layers L, which evaluates whether the quality of SPGC-based compression is affected by the complexity of GNNs classes. Then we evaluate the performance of configurable compression (α, r)-SPGC, in terms of the impact of α and r.

Varying Number of Layers. We varied the number of layers of GNNs from 2 to 4 over **Arxiv** and report its impact on inference speed-up (resp. accuracy) in Fig. 9(a) (resp. 9(b)). (1) $(0.5, 1)$-SPGC consistently outperforms all the baselines in both inference speed-up and accuracy, due to that it preserves the inference results with small compressed graphs. (2) In general, the speed-up of inference achieved by $(0.5, 1)$-SPGC is not sensitive to the number of layers. This verifies our theoretical analysis that it preserves inference results with unique smallest compressed graphs, which is independent of model complexity. (3) While the accuracy of GNNs drops as the number of layers become larger, in all cases, $(0.5, 1)$-SPGC preserves the accuracy with smallest "gap" compared with other methods, for the same class of GNNs.

Varying α. Fixing $r = 1$, we varied α from 0.2 to 1, and report the results in Figs. 10(a) to 10(d). It tells us the followings.

(1) As α is increased from 0.2 to 1, ncr drops as illustrated in Fig. 10(a). As expected, larger α makes it harder for (α, r)-SPGC to merge nodes that are less close in their representations, leaving more nodes in compressed graphs, hence worsening compression ratio. (2) Consistently for larger α and for all the three GNNs classes, it is harder for (α, r)-SPGC to improve their inference efficiency due to larger compressed structure G_c as shown in Fig. 10(b). We observe consistent results for Yelp and Products (not shown; see [1]).

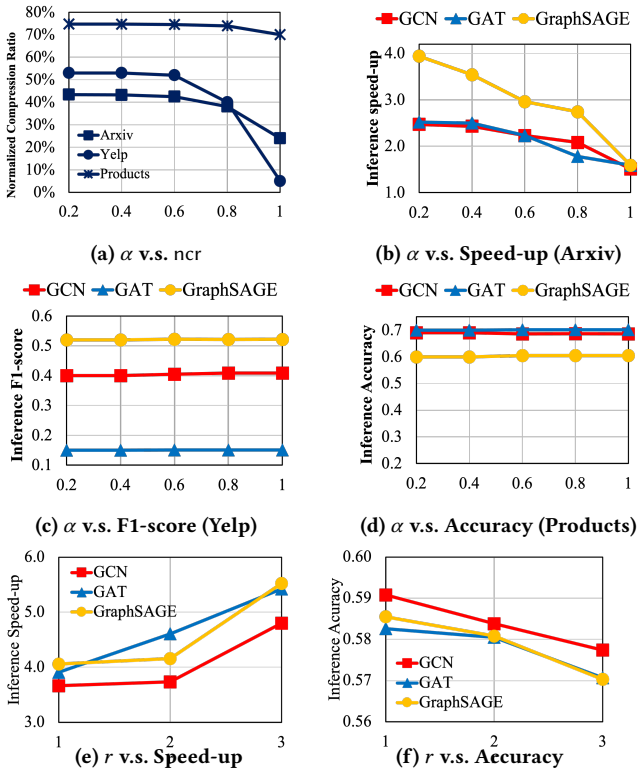
(3) As α increases, the F1-score (resp. accuracy) of the inference results over **Yelp** (resp. **Products**) remains insensitive, as shown in Figs.10(c) (resp. 10(d)). Our observation over **Arxiv** remains consistent, and we omitted it due to limited space. This indicates that (α, r)-SPGC does not lose much on the quality of the inference while significantly improved inference efficiency.

Varying r in SPGC. Fixing $\alpha = 0.25$, we vary r from 1 to 3 over **Arxiv** and report the result in Fig. 10. (1) As r is varied from 1 to 3, the inference speed-up achieved by $(0.25, r)$-SPGC for all GNNs classes notably increased. Indeed, larger r allows (α, r)-SPGC to find and merge more node pairs with equivalent embeddings, which may not be direct neighbors of another pair in the $(\alpha, 1)$-relation. (2) As r increases, the inference accuracy for all GNNs classes slightly drops, and all within a small range of 0.02. This demonstrates that (α, r)-SPGC is capable of preserving inference accuracy while increasing r in trading for larger speed up.

Exp-3: Efficiency & Scalability. We next evaluate the compression and inference costs of SPGC and its variants.

Average degree and "small-world" effect v.s. Inference Speed-up. We simulate **WS-MAG** based on (K, P) Watts-Strogatz algorithm [49] with fixed $|V| = 2M$ from **MAG240M** dataset. K and P represent average degree and re-wiring probability respectively. As P goes up, the less "small-world" (*i.e.,* more random) the graphs become.

Fixing $P = 0.6$, we vary the average degree from 2 to 4. Fig. 11(a) shows that the inference speed-up for different GNNs types. (1) As the average degree of the graph increases from 2 to 4, inference speed-up achieved by SPGC increases approximately linearly from 3.0×-4.0× to 5.0×-7.0×. (2) Inference speed-up achieved by DSpar

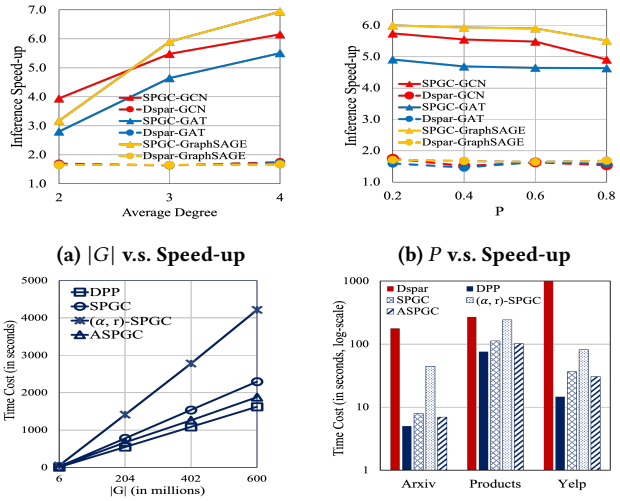


(a) α v.s. ncr (b) α v.s. Speed-up (Arxiv)

(c) α v.s. F1-score (Yelp) (d) α v.s. Accuracy (Products)

(e) r v.s. Speed-up (f) r v.s. Accuracy

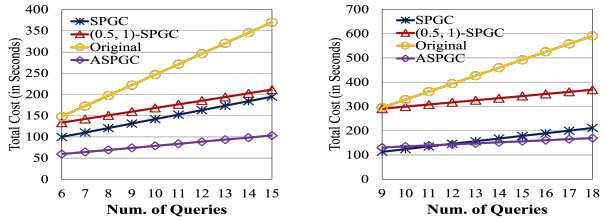
Figure 10: Varying α and r in (α, r)-SPGC



(a) $|G|$ v.s. Speed-up (b) P v.s. Speed-up

(c) $|G|$ v.s. Compression Cost (d) Compression Cost

Figure 11: Compression: Scalability and Efficiency.



(a) Yelp (b) Products

Figure 12: Total Cost: Varying Size of Queries

remains relatively stable (1.6×-1.8×) and smaller than SPGC. As the average degree goes up, SPGC may takes the advantage of higher density that makes nodes more likely to be merged, resulting in a smaller G_c and greater speed-up.

Next, fixing $|G| = 8M$, we increase P from 0.2 to 0.8. We have the following observations. (1) As P increases, the inference speed-up achieved by SPGC on three GNNs exhibits a slight drop, albeit still much larger than the speed-up by DSpar. (2) Inference speed-ups achieved by DSpar on three GNNs stay flat without even breaking 2× speed-up. This indicates that SPGC may perform better to compress "small-world" graphs, which are common in real-world networks such as social networks [7, 49].

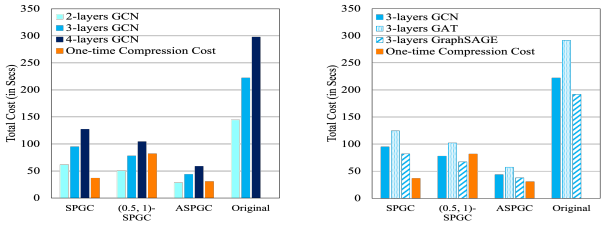
Impact of $|G|$. We increased the size of the simulated graph $|G|$ from $6M$ to $600M$, spanning two orders of magnitude. As shown in Fig. 11(c), (1) SPGC, ASPGC, and DPP all scale well with $|G|$, which is consistent with the compression cost presented in Table. 1. (2) (α, r)-SPGC has relatively higher and more sensitive compression cost as $|G|$ increases, as (α, r)-SPGC incurs more time to refine the (α, r)-relation, yet achieves better compression ratio and in turn, more reduction of the inference cost on compressed graph G_c.

Compression cost: real world datasets. Next, we compare the one-time compression cost induced by the SPGC or its variants with the cost of DSpar as shown in Fig. 11(d). Note that the compression cost of SPGC and its variants include the time cost of DPP. We have the following discoveries. (1) SPGC, (α, r)-SPGC and ASPGC outperforms DSpar on all the real-world datasets. In particular, SPGC are 95.52% and 58.45% times faster than DSpar over **Arxiv** and **Products**, respectively. For **Yelp**, DSpar does not run to completion after 1,000 seconds. (2) For SPGC and its variants, ASPGC outperforms SPGC and (α, r)-SPGC by 12.71% and 68.34% on average. This is because ASPGC exploits data locality of GNN inference and perform compression up to certain hops of the anchored nodes

Total cost: Varying query workloads. We compare the total inference costs of SPGC, (α, r)-SPGC, ASPGC to the total cost induced by the original G. For SPGC and its variants, the total cost is the sum of "one-time" compression cost and the inference cost for all inference queries; for "Original", it refers to the inference time on the original graph. Varying the number of inference queries from 6 to 18 (each with fixed number of test node $|V_T| = 0.05|V|$), Fig. 12 reports the total cost of a 3-layers GCN on **Yelp** and **Products**.

(1) All SPGC methods are able to reduce the original inference cost, and all can improve the inference efficiency better with larger amount of queries. For example, for **Yelp**, SPGC reduces 32.58% - 47.41% amount of inference cost as the number of queries varies from 6 to 15, at a one-time compression cost. Moreover, GNN inference starts to enjoy such improvement with only a few queries (6 for $(0.5, 1)$-SPGC, 3 for SPGC, and 2 for ASPGC; not shown).

(2) Among SPGC variants, for queries with given anchored nodes, ASPGC performs best in improving the inference efficiency. We also observe that (α, r)-SPGC is more sensitive to different datasets, compared with SPGC and ASPGC. Indeed, SPGC enforces rigidly embedding equivalence, while ASPGC benefits most from data locality from anchored node set of fixed size, hence both are less sensitive to datasets. (α, r)-SPGC on the other hand is most adaptive



(a) Varying GNN Layer Number **(b) Varying GNN class**

Figure 13: Total Cost: Varying GNN Layers and Class.

for tunable trade-off between inference speed up and compression ratio, depending on the heterogeneity of node embeddings.

Total cost: varying GNN complexity. We next investigate the impact of GNN model complexity, in terms of number of layers, and types. Using GCNs, we varying the number of layers L from 2 to 4 (We denote GCN_2 as the 2-layers GCN; similarly for others), and a query workload of 3 different inference queries with the same size $|V_T|$. Fig. 13(a) tells us that as the number of layers increase, the inference cost grows as expected. ASPGC outperforms SPGC and $(0.5, 1)$-SPGC consistently for all layers in compression efficiency.

Fixing query workload size as 3, and the number of layers as 3 for all GNN types, we report the total cost over GCN, GAT and GraphSAGE. As shown in Fig. 13(b), all three types of GNNs consistently and significantly benefit from SPGC, with an improvement of inference time by a factor of 2.21, 2.36 and 2.11, respectively; similar for (α, r)-SPGC and ASPGC. The one-time compression costs are consistent with our prior observations in Fig. 13(a). In general, GAT may benefit most from inference-friendly compression. A possible reason is that the memoization effectively cached its additional edge weights as part of the auxiliary information, which are a source of overhead for inference over G.

We have also investigated (1) how memoization structure $\mathcal{T}$, and neighbor recovery affect the inference accuracy and speed-up achieved by SPGC; (2) the impact of aggregation method and improvement of model training cost. Due to limited space, we present additional tests with details in [1].

8 CONCLUSION

We have proposed IFGC, an inference friendly graph compression scheme to generate compressed graphs that can be directly processed by GNN inference process to obtain original inference output. We have introduced a sufficient condition, and introduced three practical specifications of IFGC, SPGC, for inference without decompression, and configurable (α, r)-compression, to achieve better compression ratio and reduction of inference cost, and anchored SPGC, that preserves inference results for specified node set. Our theoretical analysis and experimental study have verified that IFGC-based approaches can significantly accelerate the inference on real-world and large graphs with small loss on inference accuracy. A future topic is to extend our compression scheme to accelerate GNN training. Another topic is to develop parallel compression scheme for large graphs.

REFERENCES

[1] 2025. Full version. http://github.com/Yangxin666/SPGC/blob/main/SPGC_full.pdf

[2] Arman Ahmed, Sajan K Sadanandan, Shikhar Pandey, Sagnik Basumallik, Anurag K Srivastava, and Yinghui Wu. 2022. Event Analysis in Transmission Systems Using Spatial Temporal Graph Encoder Decoder (STGED). *IEEE Transactions on Power Systems* (2022).

[3] Francesco Andreuzzi. 2021. BisPy: Bisimulation in Python. *Journal of Open Source Software* (2021).

[4] Adnan Aziz, Vigyan Singhal, Felice Balarin, Robert K Brayton, and Alberto L Sangiovanni-Vincentelli. 1994. Equivalences for fair kripke structures. In *Automata, Languages and Programming: 21st International Colloquium, ICALP 94 Jerusalem, Israel, July 11–14, 1994 Proceedings 21*. 364–375.

[5] Waïss Azizian and Marc Lelarge. 2021. Expressive Power of Invariant and Equivariant Graph Neural Networks. In *ICLR*.

[6] Pablo Barceló, Egor V Kostylev, Mikaël Monet, Jorge Pérez, Juan L Reutter, and Juan-Pablo Silva. 2020. The expressive power of graph neural networks as a query language. *SIGMOD Record* (2020).

[7] Danielle S Bassett and Edward T Bullmore. 2017. Small-world brain networks revisited. *The Neuroscientist* (2017).

[8] Maciej Besta and Torsten Hoefler. 2018. Survey and taxonomy of lossless graph compression and space-efficient graph representations. *arXiv preprint arXiv:1806.01799* (2018).

[9] Maciej Besta and Torsten Hoefler. 2024. Parallel and distributed graph neural networks: An in-depth concurrency analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2024).

[10] Jeroen Bollen, Jasper Steegmans, Jan Van den Bussche, and Stijn Vansummeren. 2023. Learning graph neural networks using exact compression. In *Proceedings of the 6th Joint Workshop on Graph Data Management Experiences & Systems (GRADES) and Network Data Analytics (NDA)*. 1–9.

[11] Stephen P Borgatti and Martin G Everett. 1992. Notions of position in social network analysis. *Sociological methodology* (1992).

[12] Linfeng Cao, Haoran Deng, Yang Yang, Chunping Wang, and Lei Chen. 2024. Graph-Skeleton:˜ 1% Nodes are Sufficient to Represent Billion-Scale Graph. In *Proceedings of the ACM on Web Conference 2024*. 570–581.

[13] Ming Chen, Zhewei Wei, Bolin Ding, Yaliang Li, Ye Yuan, Xiaoyong Du, and Ji-Rong Wen. 2020. Scalable graph neural networks via bidirectional propagation. *NeurIPS* (2020).

[14] Tianlong Chen, Yongduo Sui, Xuxi Chen, Aston Zhang, and Zhangyang Wang. 2021. A unified lottery ticket hypothesis for graph neural networks. In *ICML*.

[15] Francisco Claude and Gonzalo Navarro. 2007. A fast and compact Web graph representation. In *International Symposium on String Processing and Information Retrieval*. 118–129.

[16] Agostino Dovier, Carla Piazza, and Alberto Policriti. 2001. A fast bisimulation algorithm. In *Proceedings of the 13th International Conference on Computer Aided Verification*.

[17] Wenqi Fan, Yao Ma, Qing Li, Yuan He, Eric Zhao, Jiliang Tang, and Dawei Yin. 2019. Graph neural networks for social recommendation. In *WWW*.

[18] Yangxin Fan, Xuanji Yu, Raymond Wieser, David Meakin, Avishai Shaton, Jean-Nicolas Jaubert, Robert Flottemesch, Michael Howell, Jennifer Braid, et al. 2023. Spatio-Temporal Denoising Graph Autoencoders with Data Augmentation for Photovoltaic Data Imputation. *SIGMOD* (2023).

[19] Matthias Fey and Jan E. Lenssen. 2019. Fast Graph Representation Learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*.

[20] Xinyi Gao, Wentao Zhang, Yingxia Shao, Quoc Viet Hung Nguyen, Bin Cui, and Hongzhi Yin. 2022. Efficient Graph Neural Network Inference at Large Scale. *arXiv preprint arXiv:2211.00495* (2022).

[21] Matt Gardner, Joel Grus, Mark Neumann, Oyvind Tafjord, Pradeep Dasigi, Nelson Liu, Matthew Peters, Michael Schmitz, and Luke Zettlemoyer. 2018. Allennlp: A deep semantic natural language processing platform. *arXiv preprint arXiv:1803.07640* (2018).

[22] Floris Geerts. 2023. A Query Language Perspective on Graph Learning. In *PODS*.

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

[24] Xueting Han, Zhenhuan Huang, Bang An, and Jing Bai. 2021. Adaptive transfer learning on graph neural networks. In *KDD*.

[25] Mohammad Hashemi, Shengbo Gong, Juntong Ni, Wenqi Fan, B Aditya Prakash, and Wei Jin. 2024. A Comprehensive Survey on Graph Reduction: Sparsification, Coarsening, and Condensation. *arXiv preprint arXiv:2402.03358* (2024).

[26] Weihua Hu, Matthias Fey, Hongyu Ren, Maho Nakata, Yuxiao Dong, and Jure Leskovec. 2021. OGB-LSC: A Large-Scale Challenge for Machine Learning on Graphs. *NeurIPS* (2021).

[27] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. 2020. Open graph benchmark: Datasets for machine learning on graphs. *NeurIPS* (2020).

[28] Zengfeng Huang, Shengzhong Zhang, Chong Xi, Tang Liu, and Min Zhou. 2021. Scaling up graph neural networks via graph coarsening. In *SIGKDD*.

[29] Ahmad Maroof Karimi, Yinghui Wu, Mehmet Koyuturk, and Roger H French. 2021. Spatiotemporal graph neural network for performance prediction of photovoltaic power systems. In *AAAI*.

[30] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).

[31] Manoj Kumar, Anurag Sharma, Shashwat Saxena, and Sandeep Kumar. 2023. Featured graph coarsening with similarity guarantees. PMLR.

[32] N Jesper Larsson and Alistair Moffat. 2000. Off-line dictionary-based compression. *Proc. IEEE* 88, 11 (2000), 1722–1732.

[33] Dongyue Li, Tao Yang, Lun Du, Zhezhi He, and Li Jiang. 2021. AdaptiveGCN: Efficient GCN through adaptively sparsifying graphs. In *CIKM*.

[34] Xin Liu, Mingyu Yan, Lei Deng, Guoqi Li, Xiaochun Ye, Dongrui Fan, Shirui Pan, and Yuan Xie. 2022. Survey on graph neural network acceleration: An algorithmic perspective. *arXiv preprint arXiv:2202.04822* (2022).

[35] Yajiong Liu, Yanfeng Zhang, Qiange Wang, Hao Yuan, Xin Ai, and Ge Yu. 2025. NeutronSketch: An in-depth exploration of redundancy in large-scale graph neural network training. *Knowledge-Based Systems* 309 (2025), 112786.

[36] Zirui Liu, Kaixiong Zhou, Zhimeng Jiang, Li Li, Rui Chen, Soo-Hyun Choi, and Xia Hu. 2023. DSpar: An Embarrassingly Simple Strategy for Efficient GNN Training and Inference via Degree-Based Sparsification. *TMLR* (2023).

[37] Francois Lorrain and Harrison C White. 1971. Structural equivalence of individuals in social networks. *The Journal of mathematical sociology* 1, 1 (1971), 49–80.

[38] Andreas Loukas and Pierre Vandergheynst. 2018. Spectrally approximating large graphs with smaller graphs. In *ICML*.

[39] Andrew Kachites McCallum, Kamal Nigam, Jason Rennie, and Kristie Seymore. 2000. Automating the construction of internet portals with machine learning. *Information Retrieval* (2000).

[40] Hongwu Peng, Deniz Gurevin, Shaoyi Huang, Tong Geng, Weiwen Jiang, Orner Khan, and Caiwen Ding. 2022. Towards sparsification of graph neural networks. In *2022 IEEE 40th International Conference on Computer Design (ICCD)*. 272–279.

[41] Hao Peng, Hongfei Wang, Bowen Du, Md Zakirul Alam Bhuiyan, Hongyuan Ma, Jianwei Liu, Lihong Wang, Zeyu Yang, Linfeng Du, Senzhang Wang, et al. 2020. Spatial temporal incidence dynamic graph neural networks for traffic flow forecasting. *Information Sciences* (2020).

[42] Patrick Reiser, Marlen Neubert, André Eberhard, Luca Torresi, Chen Zhou, Chen Shao, Houssam Metni, Clint van Hoesel, Henrik Schopmans, Timo Sommer, et al. 2022. Graph neural networks for materials science and chemistry. *Communications Materials* (2022).

[43] Vıctor Garcia Satorras, Emiel Hoogeboom, and Max Welling. 2021. E (n) equivariant graph neural networks. In *ICML*.

[44] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. 2008. The graph neural network model. *IEEE transactions on neural networks* (2008).

[45] Weijing Shi and Raj Rajkumar. 2020. Point-GNN: Graph Neural Network for 3D Object Detection in a Point Cloud. In *CVPR*.

[46] Shyam A Tailor, Javier Fernandez-Marques, and Nicholas D Lane. 2020. Degree-quant: Quantization-aware training for graph neural networks. *arXiv preprint arXiv:2008.05000* (2020).

[47] Petar Veličković. 2023. Everything is connected: Graph neural networks. *Current Opinion in Structural Biology* (2023).

[48] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, Yoshua Bengio, et al. 2017. Graph attention networks. *stat* (2017).

[49] Duncan J Watts and Steven H Strogatz. 1998. Collective dynamics of 'small-world'networks. *nature* (1998).

[50] Fengli Xu, Quanming Yao, Pan Hui, and Yong Li. 2021. Automorphic equivalence-aware graph neural network. *Advances in Neural Information Processing Systems* 34 (2021), 15138–15150.

[51] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2019. How powerful are graph neural networks?. In *ICLR*.

[52] Hanqing Zeng, Muhan Zhang, Yinglong Xia, Ajitesh Srivastava, Andrey Malevich, Rajgopal Kannan, Viktor Prasanna, Long Jin, and Ren Chen. 2021. Decoupling the depth and scope of graph neural networks. *NeurIPS* (2021).

[53] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. 2019. Graphsaint: Graph sampling based inductive learning method. *arXiv preprint arXiv:1907.04931* (2019).

[54] Muhan Zhang and Yixin Chen. 2018. Link prediction based on graph neural networks. *NeurIPS* (2018).

[55] Cheng Zheng, Bo Zong, Wei Cheng, Dongjin Song, Jingchao Ni, Wenchao Yu, Haifeng Chen, and Wei Wang. 2020. Robust graph representation learning via neural sparsification. In *ICML*.

[56] WANG Zhili, DI Shimin, CHEN Lei, and ZHOU Xiaofang. 2024. Search to fine-tune pre-trained graph neural networks for graph-level tasks. In *ICDE*.

[57] Hongkuan Zhou, Ajitesh Srivastava, Hanqing Zeng, Rajgopal Kannan, and Viktor Prasanna. 2021. Accelerating large scale real-time GNN inference using channel pruning. *arXiv preprint arXiv:2105.04528* (2021).

[58] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. 2020. Graph neural networks: A review of methods and applications. *AI open* (2020).

[59] Yaqin Zhou, Shangqing Liu, Jingkai Siow, Xiaoning Du, and Yang Liu. 2019. Devign: Effective vulnerability identification by learning comprehensive program semantics via graph neural networks. *NeurIPS* (2019).

9 APPENDIX

9.1 Appendix A: Proofs

Proof of Lemma ??. Given a class of GNNs $\mathbb{M}^L$ and a graph G, for any two nodes v and v' in G and any GNN $M \in \mathbb{M}^L$, $M(v, G) = M(v', G)$, if and only if $v \sim_M^L v'$.

Proof. We start by proving the following **If** condition: for any pair of nodes (v, v') from G, if $v \sim_M^L v'$, then $M(v, G) = M(v', G)$ for any GNN $\mathcal{M} \in \mathbb{M}^L$. By definition, $v \sim_M^L v'$ *w.r.t.* $\mathbb{M}^L$ indicates that for any GNN $\mathcal{M} \in \mathbb{M}^L$, the inferred representation ("embedding") of v at layer k of $\mathcal{M}$ is the same of its counterpart from v', *i.e.,* $X_v^k = X_{v'}^k$ for $k \in [0, L]$. Thus at the output layer (when k=L) of $\mathcal{M}$, for the inferred final node embeddings, $X_v^L = X_{v'}^L$. By definition of inference function $M(\cdot)$, $M(v, G) = X_v^L = X_{v'}^L = M(v', G)$. Thus $M(v, G) = M(v', G)$.

We prove the **Only If** condition by contradiction. As we consider fixed deterministic models, let $\mathbb{M}^L$ be adopting a same node update function M_v that is simply a fixed, linear function, *i.e.,* $X_v^k = c \cdot X_v^{k-1}$, where c is a positive constant. Assume $M(v, G) = M(v', G)$, yet $v \not\sim_M^L v'$. Then there exists a number $k \in [1, L]$, for which $X_v^k \neq X_v^k$. As the inference function $M(v, G)$ simulates a composition of node update functions, *i.e.,* $X_v^L = M_v^L \circ (M_v^{L-1}(\circ(\cdots M_v^1(X_v^0)) \cdots))$. Then $X_v^L = M_v^L \circ (M_v^{L-1}(\circ(\cdots M_v^{L-k}(X_v^k)) \cdots)) = c^k \cdot X_v^k$; and similarly, $X_{v'}^L = c^k \cdot X_{v'}^k$. Hence $M(v, G) = X_v^L = c^k \cdot X_v^k \neq c^k \cdot X_{v'}^k = X_{v'}^L = M(v', G)$. This contradicts to the assumption that $M(v, G) = M(v', G)$.

Given the above analysis, Lemma **??** follows. □

Proof of Lemma 1. Given $\mathbb{M}$ and G, the binary relation R_M^L is an equivalence relation, *i.e.,* it is reflexive, symmetric, and transitive.

Proof. (1) It is easy to verify that R_M^L is reflexive, *i.e.,* $(v, v) \in R_M^L$ for any $v \in V$: for any node v, and any of its neighbor v', an identity mapping verifies the reflexiveness of R_M^L. (2) We show the symmetric by definition. If $(v, v') \in R_M^L$, then there is a "matching relation" h, where $h(v)$=v', such that for every neighbor u of v, there is a neighbor $u' = h(u)$ such that $(u, h(u)) \in R_M^L$. Consider the inverse relation h^{-1}, we can verify that $(h(u), h^{-1}(h(u))) = (h(u), u) = (u', u) \in R^S$. This holds for every neighbor u' of v', which leads to that $(v', v) \in R^S$. (3) The transitivity of R_M^L states that for any pair (v_1, v_2) and (v_2, v_3) from G, if $(v_1, v_2) \in R_M^L$ and $(v_2, v_3) \in R_M^L$, then $(v_1, v_3) \in R_M^L$. Let h be the matching relation that induces R_M^L, where $h(v_1) = v_2$, and $h(v_2) = v_3$. Then consider a composite relation $h' = h \circ h$. We have $h'(v_1) = h(h(v_1)) = v_3$. Similarly, we can verify that $h'^{-1}(v_3) = v_1$. Hence $(v_1, v_3) \in R_M^L$ as induced by $h' = h \circ h$. The transitivity of R_M^L follows. □

Proof of Lemma 2.Given a class of GNNs $\mathbb{M}^L$ and a graph G, there exists a IFGC $(C, \mathcal{P})$ *w.r.t.* $\mathbb{M}^L$ and G, if there is a nontrivial

inference equivalent relation R_M^L *w.r.t.* (1) $C(G)$ computes a quotient graph G_c induced by a nontrivial inference equivalent relation R_M^L *w.r.t.* $\mathbb{M}^L$ and G, and (2) $|\mathcal{P}(G_c)| < |G|$.

Proof. Let R_M^L be a non-empty inference equivalent relation *w.r.t.* $\mathbb{M}^L$ and G, and G_c be the quotient graph induced by $\mathbb{M}^L$. (1) Given Lemma 1, R_M^L is a nontrivial equivalence relation. As $R_M^L \neq \emptyset$, there exists at least one equivalent class $[v]$ with size larger than one, *i.e.,* $|G_c| < |G|$. (2) As the post processing function $\mathcal{P}$ over G_c does not introduce additional nodes or edges, as ensured by $|\mathcal{P}(G_c)| < |G|$, we have $|\mathcal{P}(C(G))| < |G|$.

We next show that $M(G) = M(\mathcal{P}(C(G)))$. To see this, it suffices to show that for every node $v \in G$, $M(G, v) = M(\mathcal{P}(C(G)), [v])$. Indeed, if this holds, the output representation of $M(G_c, [v])$ can be readily assigned to every node $v \in [v]$ without decompression. Further, it suffices to show that for any nodes $(v_i, v_j) \in [v]$, $M(v_i, G) = M(v_j, G) = M([v], G_c)$.

We prove the above result by conducting an induction on the layers k of the GNNs. The general intuition is to show that it suffices for the compression function C to track some auxiliary information of the neighbors of v in G when generating the compressed graph G_c, such that the aggregation result of the node update function X_v for each node v can be readily computed by a weighted aggregation in G_c in a post processing function $\mathcal{P}$, *without* decompression (hence incurs no additional inference cost). Such information can be readily tracked and looked up as needed at inference time over G_c, by storing R_M^L equivalently as a "matching" relation h between each node v and its equivalence class $[v]$.

Given any two nodes v and v' such that $(v, v') \in R_M^L$, *i.e.,* $v \sim_M^L v'$, consider an equivalent characterization of R_M^L as a mapping h between G and G_c such that $h(v) = [v]$ in G_c, for each node $v \in G$. (1) Let $k = 0$. Then one can verify that $M(v_i, G) = X_{v_i}^0 = X_{v_j}^0 = M^0(v_j) = M^0([v]) = M([v], G_c)$.

(2) Let $k = i$, and assume the result holds for any GNNs in $\mathcal{M}^L$ at layer $k = i$. That is, for every node $v \in G$, $M^i(G, v) = M^i(\mathcal{P}(C(G)), [v])$. As $[v]$ is an equivalence class induced by R_M^i, $M^i(v, G) = M^i(v', G) = M^i([v], G_c)$ for i-layered GNNs $\mathcal{M}^i$ with the same node update function M_i (Lemma **??**). Consider the output of $M^{i+1}(v, G)$. The computation invokes the node update function as $X_v^{i+1} = M_v(\Theta^i, \text{AGG}, N(v), M_v^i)$, and $X_{[v]}^{i+1} = M_{[v]}(\Theta^i, \text{AGG}, N([v]), M_{[v]}^i)$. As we consider fixed, deterministic GNNs with the same node update function,

- the model weights $\Theta^{i+1} = \Theta^i$,
- operator AGG remain to be fixed,
- $M_v^i = M_{[v]}^i$ by induction; and
- the matching relation h ensures the invariant that for *every* neighbor $u \in N(v)$, there exists a counterpart $[u'] \in N(h(v)) = N([v])$ in the quotient graph G_c, such that $X_u^i = X_{[u]}^i$, ensured by R_M^i and the definition of quotient graphs.

We now show that $X_v^{k+1} = M_v(\Theta^k, \text{AGG}, N(v), M_v^k)$, $= M_{[v]}(\Theta^k, \text{AGG}, N([v]), M_{[v]}^k) = X_{[v]}^{k+1}$. To see this, $M_{[v]}$ only needs to "invoke" a fast look up function $\mathcal{P}$ that retrieves an edge weight (pre-stored during compression C; see "Notes") to adjust its direct aggregation over $N_{[v]}$ in G_c, as needed, without decompression. We illustrate below typical examples of the weighted update for major GNNs in Table 4 augmenting Table. 3; where the weights are highlighted in bold and red.

Hence, for any pair of nodes $(v, v') \in [v]$, $M(v, G) = M(v', G) = M([v], G_c)$ for any GNN $M \in \mathcal{M}^L$. By definition, the graph computation scheme $(C, \mathcal{P})$ is an IFGC for $\mathcal{M}^L$ and G. □

Remark. The above analysis verifies that G_c *preserves* the inference results, by showing that the computation of M_v in G can be simulated by an equivalent, re-weighted inference-friendly counterpart $M_{[v]}$ that directly process G_c without decompression. The weights can be easily bookkept during compression, tracked by a *run-time* look-up function $\mathcal{P}$ along with the inference process, over a (smaller) G_c, without a stacked run of $\mathcal{P}$, without incurring additional time cost, and without decompression. Moreover, this incurs only a small bounded memory cost of up to $|E|$ weights (numbers). We refer the implementation details in Sections 4, 5.1 and 5.2, respectively, for IFGC specifications.

Proof of Theorem 3. Given a class of GNN $\mathbb{M}^L$ and graph G, the relation R^S over G is an inference equivalence relation *w.r.t.* $\mathbb{M}^L$.

Proof. We first show that R^S is an equivalence relation, *i.e.,* it is reflexive, symmetric, and transitive. (1) It is easy to verify that R^S is reflexive and symmetric, *i.e.,* $(v, v) \in R^S$ for all the nodes in G, and for any node pair $(v, v') \in R^S$, $(v', v) \in R^S$, by definition. (2) To see the transitivity, let $(v_1, v_2) \in R^S$, and $(v_2, v_3) \in R^S$. Consider a matching relation h such that for each pair $(v, v') \in R^S$, $h(v) = v'$. Then $h(v_1) = v_2$, $h(v_2) = v_3$. We can verify that $h(v_1) = v_3$, which induces that $(v_1, v_3) \in R^S$, by the definition of structural equivalence. Hence R^S is an equivalence relation.

We next show that R^S is an inference equivalence relation *w.r.t.* $\mathcal{M}^L$, that is, for any pair $(v, v') \in R^S$, $v \sim_M^L v'$. Similarly as the analysis for Theorem 2, we perform an induction on the number of layers k of GNNs M.

(1) Let $k = 0$. Then $M(v, G) = X_{v_i}^0 = X_{v_j}^0 = M^0(v_j) = M^0([v]) = M([v], G_c)$.

(2) Assume R^S is an inference equivalence relation *w.r.t.* $\mathcal{M}^L$ for $k = i$. Given $(v, v') \in R^S$, for the layer $i + 1$, $X_v^{k+1} = M_v(\Theta^k, \text{AGG}, N(v), M_v^k)$, $= M_{[v]}(\Theta^k, \text{AGG}, N([v]), M_{[v]}^k) = X_{[v]}^{k+1}$, following the similar analysis as in R_M^L counterpart. Note that the h matching function of R^S is specified for R^S; and $\mathcal{P}$ is an identity function. □

Proof of Theorem 4. Given a class of GNNs $\mathbb{M}^L$ with the same node update function M_v, and a graph G, a SPGC produces a unique, minimum compressed graph G_c, up to graph isomorphism over the quotient graphs induced by R^S.

Proof. To see this, we specify the compression process C of SPGC to be a function that computes the *largest* structural-equivalence relation R^{S*} *w.r.t.* $\mathbb{M}^L$ and G. We first show that there exists a unique largest inference-friendly relation R^{S*} over G. We then show that the unique, largest R^{S*} induces a minimum compressed graph G_c^*, up to graph isomorphism over the quotient graphs induced by R^S.

(1) We prove the uniqueness property by contradiction. Assume there are two largest structural equivalence relations R^{S*} and R'^{S*},

where $|R^{S*}|=|R'^{S*}|$, and $R^{S*} \neq R'^{S*}$. Let $(v, v') \in R^{S*}$, and $(v, v') \neq R'^{S*}$. For the latter case, either $X_v^0 \neq X_{v'}^0$, or there exists a neighbor $u \in N(v)$ such that there is no neighbor u' in $N(v')$ such that $(u, u') \in R'^{S*}$. For the first case, clearly $(v, v') \neq R^{S*}$. For the second case, given Lemma **??**, there exists a GNN $\mathcal{M} \in \mathcal{M}^L$ such that $M(u, G) \neq M(u', G)$, as u' ranges over all the neighbors of v'. This indicates that $(v, v') \neq R^S$, which contradicts to that R^S is an inference equivalence relation. Hence $R^{S*} = R'^{S*}$.

(2) We consider the notion *graph isomorphism* defined over compressed graph. We say two compressed graphs G_c and G'_c are isomorphic, if there exists a bijective function h_c between G_c and G'_c, such that for any edge $([u], [v])$ in G_c, $(h_c([u]), h_c([v]))$ in G'_c.

Given that R^{S*} is the unique largest structural equivalence relation that is also an inference equivalence relation, let G_c^* be the corresponding quotient graph of R^{S*}. Assume there exists another quotient graph $G_c'^*$ induced from R^{S*} that is not isomorphic to R^{S*}. Then there exists at least an edge $([u], [v])$ in G_c^* for which no edge exists in $G_c'^*$. Given that G_c^* is the quotient graph induced by the maximum structural relation R^{S*}, then either $G_c'^*$ has a missing edge, which contradicts to that it is a quotient graph induced by the largest R^{S*}, or R^{S*} is not inference equivalence, which contradicts to that it is a structural equivalence relation, given Theorem 3. Hence there exists a unique, smallest quotient graph induced by R^{S*} up to graph isomorphism. □

Discussion of Compression Ratio by SPGC. Given GNNs $\mathbb{M}^L$ and graph G, SPGC achieves (1) an optimal compression ratio $\frac{|G|}{|G_c|}$, where G_c is the unique minimum quotient graph induced by the maximum R^S *w.r.t.* $\mathbb{M}^L$ and G; and (2) an optimal speed up of inference for $\mathbb{M}^L$ at $\frac{d|G|}{|G_c|}$ (where d is the maximum degree of G), independent of GNN configurations.

Proof. Given that there exists a unique smallest compressed graph G_c^* induced by the largest structural equivalence relation R^S, a theoretical optimal compression ratio cr can be provided as $\frac{|G|}{|G_c^*|}$.

Accordingly, considering an upperbound of the inference time cost of the mainstream GNNs as summarized in Table 3. A maximum speed up for inference cost can be computed as $\frac{O(LmdF^2+LnF^2)}{O(Lm'dF^2+Ln'F^2)} \leq \frac{md+n}{m'+n'} \leq d\frac{m+n}{m'+n'} = d \cdot \text{cr}$.

Interestingly, this result establishes a simple connection between a "best case" speed up and the theoretical optimal compression ratio, in terms of a single factor d that is the maximum degree of G. The intuition is that in the "ideal" case, every neighbor u of a node v in G is pairwise indistinguishable for the inference process, thus are "compressed" into a single node $[u]$, introducing a local inference cost reduction at most d times. □

Proof of Theorem 5. Given a class of GNNs $\mathbb{M}^L$ with the same node update function M_v, and a set of graphs $\mathcal{G}$, for any GNN $\mathcal{M} \in \mathbb{M}^L$, a SPGC $(C, \_)$ computes a unique compressed set $\mathcal{G}_c$, such that for any pair $(G, G') \in \mathcal{G}_{\mathcal{M}}$, there exists a pair $(G_c, G'_c) \in \mathcal{G}_{\mathcal{M}}$, *i.e.,* the discrminativeness of $\mathcal{M}$ is preserved by SPGC.

Proof. The uniqueness of the set $\mathcal{G}_c$ can be shown by verifying that each compressed graph $G_c \in \mathcal{G}_c$ is a corresponding unique, smallest compressed graph for an original counterpart $G \in \mathcal{G}$.

Let (G, G') be a pair in $\mathcal{G}_{\mathcal{M}}$ for a GNN $\mathcal{M} \in \mathcal{M}^L$. Then $M(G) = M(G')$. Given that G_c is the unique smallest compressed graph of G induced by an inference-equivalence relation, $M(v, G)$ can be computed via a weighted inference process $M([v], G_c)$, for every node v in G. Hence $M(G) = M(\mathcal{P}(G_c))$. Similarly, $M(G') = M(\mathcal{P}(G'_c))$. Thus $M(\mathcal{P}(G_c)) = M(\mathcal{P}(G'_c))$. Given that $\mathcal{M}$ remains a fixed model, $M(G'_c) = M(G_c)$ for each node $[v]$ in G_c, hence $(G'_c, G_c) \in \mathcal{G}_{c\mathcal{M}}$. □

9.2 Appendix B: Algorithms

Procedure DPP. Given G, procedure DPP computes the equivalence relation R that satisfy for any node pair (v, v') in R, if and only if the followings holds:

- for any neighbor u of v ($u \in N(v)$), there exists a neighbor u' of v' (u' in $N(v')$), such that $(u, u') \in R^S$; and
- for any neighbor u'' of v' in $N(v')$, there exists a neighbor u''' of v in $N(v)$, such that $(u'', u''') \in R^S$.

Procedure DPP first computes the rank for all nodes in G and identifies the maximum rank (line 1-3). It next induces the node partition Par based on the rank (line 4). Then it collapses nodes in $B_{-\infty}$ such that only one randomly selected node $v \in B_{-\infty}$ remains and all edges that were incident to the eliminated nodes are redirected to be incident to v (line 5). It next induces the equivalence relation R_i at each rank i (line 6-7). It next prunes each R_i based on whether there are edges incident to nodes in $B_{-\infty}$ and updates B_i accordingly (line 8-11). It next iterates over the rank equal to $0, \ldots, \phi$ (line 12). Within each iteration, it conducts the followings: 1) it computes the D_i and refines it using **Paige-Tarjan** and collapses all $X \in D_i$ (line 13-15); 2) it prunes each R_j where $j \in \{i+1, \ldots, \phi\}$ based on whether there are edges incident to nodes in B_i and updates B_j (line 16-19). It combines all R_i to derive R and returns R (line 20-21).

Inference process with decompression. Following the inference algorithm in Sec. 4.4, the users can directly query on V_T from G_c compressed from (α, r)-SPGC without any decompression and benefit from the accelerated inference. However, this may result in dropped inference accuracy since inference equivalence is not directly preserved. A pair (v, v') in an (α, r)-relation $R^{(\alpha,r)}$ is no longer conform to embedding equivalence, thus an (α, r)-SPGC $(C, \_)$ alone is not an IFGC, *i.e.,* no longer inference-friendly for a given G and GNNs class $\mathcal{M}^L$. We next show that with a cost-effective decompression process $\mathcal{P}$, a $(1, r)$-SPGC $(C, \mathcal{P})$ becomes inference friendly. The idea is to integrate a inference-time "restoring" of the r-hop neighbors up to a local range. We start by introducing an auxiliary structure called *neighbor correction table*. For each node $v \in G$, a neighbor correction table is a compressed encoding of its r-hop neighbors $N_r(v)$. There are a host of work on effective encoding of nodes and their neighbors (see [8]). We non-trivially extend Re-Pair compression, a reference encoding method [15, 32] for efficient decompression of r-hop neighbors with following two types of pointers that maintain: 1) equivalent class/cluster (nodes within same equivalence relation can be reached from each other)

Algorithm 3 Procedure DPP(G)

for $v \in V$ **do**
  compute rank(n);
ϕ := max{rank(n)};
node partition Par := $\{B_i : i = -\infty, 0, \ldots, \phi\}$
collapse $B_{-\infty}$;
for $i = -\infty, 0, \ldots, \phi$ **do**
  induce R_i at rank i;
for $n \in V \cap B_{-\infty}$ **do**
  for $i = 0, \ldots, \phi$ **do**
    $R_i := R_i \setminus \{(v, v') | (v, n) \in E$ and $(v', n) \notin E\}$;
    update B_i;
for $i = 0, , \ldots, \phi$ **do**
  $D_i := \{X \in$ Par $: X \subseteq B_i\}$;
  refine D_i with **Paige-Tarjan**;
  collapse $X \mid \forall X \in D_i$;
  for $n \in V \cap B_i$ **do**
    for $j = i + 1, \ldots, \phi$ **do**
      $R_j := R_j \setminus \{(v, v') | (v, n) \in E$ and $(v', n) \notin E\}$;
      update B_j;
$R := \bigcup_{i \in \{-\infty, 0, \ldots, \phi\}} R_i$;
return R

Figure 14: Procedure DPP

Algorithm 4 : (α, r)-SPGC

Input: Graph G, node feature matrix X, configuration (xsim, α, r); a class of GNNs $\mathbb{M}^L$ with node update function M;
Output: A compressed graph G_c and $\mathcal{T}'$, EC, compressed encodings $\mathsf{AL_c}$ and rules;

set $R^{(\alpha,r)} := \emptyset$; set $EC := \{V\}$; set $\mathcal{T}' := \emptyset$; set $AL := \emptyset$; set $\mathsf{AL_c}$ $:= \emptyset$; dictionary rules $:= \emptyset$; graph G_c, $G_r := \emptyset$;
$G_r := (V, E_r) \mid E_r := \{(u, v)\}, v \in V$ and $u \in N_r(v)$;
induce adjacency list AL from E_r;
$R^{(\alpha,r)} :=$ DPP(G_r);
$R^{(\alpha,r)} := R^{(\alpha,r)} \setminus \{(v, v') \mid \mathsf{xsim}(X_v^0, X_{v'}^0) < \alpha\}$;
$EC := V / R^{(\alpha,r)}$; /* induce partition EC from $R^{(\alpha,r)}$ */
$G_c, \mathcal{T} :=$ CompressG ($\mathcal{T}, G_c, EC, G, M$);
$\mathsf{AL_c}$, rules := Re − Pair(AL);
return $G_c, \mathcal{T}, EC, \mathsf{AL_c}$, rules;

Figure 15: Algorithm (α, r)-SPGC

and 2) common r-hops neighbors shared by nodes within in same equivalence relation.

Decompression Algorithm. We next outline decompG shown in Fig. 16 that implements a decompression function $\mathcal{P}$. Upon receiving an inference query defined on V, decompG visits every node v in $[v] \in V_c$ exactly once and identifies D such that it captures $N_r(v)$ that have not been decompressed by visited nodes. Then decompG adds new edges between the nodes in D and $[v]$.

Example 7: We continue with Example. 6. Given $G_c 5$, EC, $\mathsf{AL_c}$, and rules, decompG sorts $\mathsf{AL_c}$ in the descending order of node degrees

Algorithm 5 : decompG

Input: Compressed graph G_c, G_r, EC, $\mathsf{AL_c}$, rules;
Output: A de-compressed graph G_{cd};

$G_{cd} = G_c$;
sort $\mathsf{AL_c}$ by node degrees in G_r;
for $[v] \in G_c.V_c$ **do**
  while $\exists$ not visited $v \in [v]$ **do**
    $D = \{u | u \in N_r(v) \wedge \neg$ decompressed by $v \in [v]\}$;
    for $\forall u \in D$ **do**
      V_{cd}.add(u);
      E_{cd}.add(u, $[v]$);
    mark v as visited;
return G_{cd};

Figure 16: Algorithm decompG

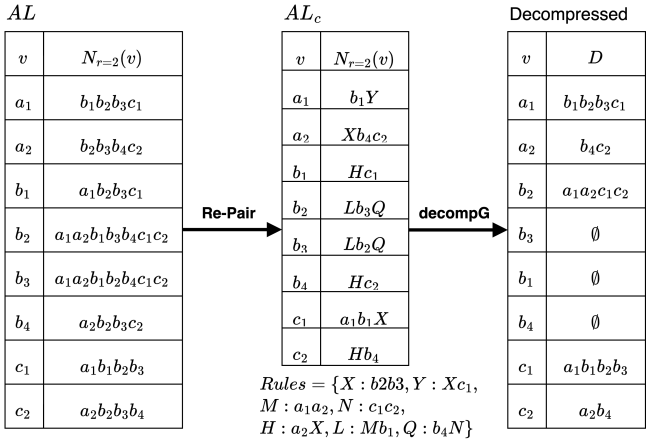


Figure 17: Illustration of the compression by Re − Pair **and decompression by** decompG **(G_5 in Fig. 6 as the example).**

while still keeping it in the order of equivalent class. decompG next decompresses $N_r([a])$ such that the nodes $b_1b_2b_3c_1$ are decompressed first by a_1 and then b_4c_2 are decompressed by a_2 as illustrated in the Fig. 17. Similarly, $N_r([b])$ and $N_r([c])$ are decompressed accordingly. decompG returns the decompressed graph G_{cd} as shown in the Fig. 6. □

Correctness. decompG recovers the union of up-to r-hop neighbor nodes for all nodes in $[v]$. decompG adds edges such that messages can be passed from up-to r-hop to all nodes in $[v]$ during GNNs inference. This ensures that lost information in compression can be recovered by added neighbor nodes and edges (decompression).

Inference Cost. decompG is in $O(|V|r)$ time. It takes $O(r)$ time to decompress its r-hop neighbor nodes of v. In the worst case, decompG decompresses G_c back to the size of G_r, thus the inference cost on the decompressed graph is in $O(L|E_r|F + L|V|F^2)$ time.

Lemma 8: G_c *from* $(1, 1)$ SPGC $\equiv G_c$ *from* SPGC. □

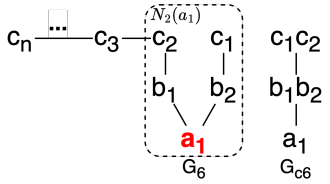


Figure 18: Compressing graph G_6 with anchored SPGC: for 2-layered GNNs, with anchored set $V_A = \{a_1\}$.

	GCN	GAT	GraphSAGE
AVG	**0.5908**	0.5796	**0.5855**
$\sum$	0.5096	0.5297	0.4602
Median	0.5901	**0.5804**	0.5849
Max	0.5766	0.5660	0.5584
Min	0.5758	0.5736	0.5641

Table 6: Comparison of Inference Accuracy with Different Aggregation Methods in SPGC (Arxiv).

Proof Sketch. Give $(1, 1)$ SPGC, we have $\alpha = 1$ and $r = 1$. When $r = 1$, the Partition computes the maximum bisimulation derived from A, which is same computation as the line 4 in SPGC. When $\alpha = 1$, we assign all nodes in each equivalent class the same labels. Therefore, there are no changes to EC from line 7 to line 11 in (α, r) SPGC. Finally, the line 12 is same as the line 6 in SPGC. Therefore, G_c derived from $(1, 1)$ SPGC is same as the one derived from SPGC.

An example to illustrate ASPGC. We consider the following example.

Example 8: Consider a class of GNNs $\mathcal{M}^2$, and graph G_6 with $V_A = \{a_1\}$ (shown in Fig. 18). For layer 2 GNNs, ASPGC first induces a subgraph G_6^L with 2-hop neighbors of a_1. It then follows a SPGC counterpart to compute the compressed graph G_{c_6}, with only three nodes. Observe that the compressed graph G_{c_6} does not guarantee to preserve the embedding of other nodes, but only a_1. For inference over node $b_1 \notin V_A$, since G_6^L do not cover within 2-hop neighbors of b_1, G_{c_6} cannot preserve its embedding. One can further verify that (1) the node pair $(c_1, c_2) \in R_L^A$ but $\notin R^S$, and (2) the anchored compression does not need to consider nodes beyond L-hop of anchored nodes (such as the chain from c_3 to c_n), as it best exploits the data locality of GNN inference process "centered" at V_A. □

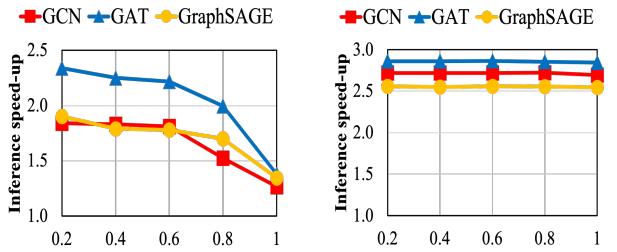
9.3 Appendix C: Additional Experiments

The impact of aggregation methods on SPGC. Fixing both $\alpha = 0.25$ and $r = 1$, we select five different aggregation methods: Mean, Sum, Median, Max, and Min used for node embedding construction of node $[v] \in G_c$ to compare the performance of SPGC over ***ogbn-arxiv*** with different aggregation methods. As illustrated in the Table. 7, Mean aggregation achieves the best overall inference accuracy (highest in GCN and GraphSAGE), followed by Median (highest in GAT) while Max and Min fall behind other methods.

Training Cost and Accuracy of SPGC. We use **Arxiv** to compare the training cost and accuracy of a 3-layers GCN using the compressed graph G_c from SPGC and $(0.5, 1)$-SPGC to training cost and accuracy using the original graph G. Table. 7 reports the

	Original G	$(0.5, 1)$ − SPGC	SPGC
$\|V\|$	169,343	127,508	143,939
$\|E\|$	1,166,243	205,132	607,989
ncr	0.00%	75.09%	43.70%
Train Time	682.63s	273.32s	405.70s
Train Accuracy	0.65	0.63	0.65

Table 7: Comparison of Training Accuracy and Cost of G_c from Original G, $(0.5, 1)$ − SPGC, and SPGC (Arxiv).



(a) α v.s. Speed-up (Yelp) **(b) α v.s. Speed-up (Products)**

Figure 19: Varying α in SPGC.

training accuracy and time comparison. We observe that, compared to training on the original G, SPGC, *i.e.,*: SPGC achieves a 40.57% faster training than training on G while retaining comparable accuracy. On the other hand, $(0.5, 1)$-SPGC trades a mere 3.10% loss in accuracy for a 59.96% faster training time compared to the training on G. Above results experimentally demonstrate that training on the compressed graph G_c from SPGC and (α, r)-SPGC can accelerate the training procedure of GNN while achieving comparable test accuracy compared to the training on original G.

Varying α and inference speed-up. As α increases, it is harder for (α, r)-SPGC to improve their inference efficiency due to larger compressed structure. Here we show two additional results from **Yelp** and **Products** datasets in Fig. 19. We observe that as α increases, inference speed-up on **Yelp** and **Products** exhibit similar trends as observed from **Arxiv** datasets.

Memory Cost Analysis. We analyze memory consumption by comparing the memory costs associated with the original graph G and its compressed counterpart G_c derived using SPGC. Following the definition of ncr, we define Memory Compression Rate (mcr) as mcr = $1 - \frac{|M_c|}{|M|}$. It quantifies the fraction of memory cost that is "reduced": the larger, the better; The detailed results are presented in Table 8. To demonstrate the memory savings achieved by G_c, we conduct the following three tests:

(1) **Graph Memory (GM) Test**: We measure the graph memory usage as the graph size in the PyTorch Geometric Graph Data format [19] including node features, node labels, and edge index tensors. Comparing to G, the GM for G_c is reduced by 38.26%, 21.58%, and 76.01% for ***Arxiv***, **Yelp**, and ***Ogbn-Products*** respectively;

(2) **Memoziation Structure $\mathcal{T}$ Memory (MSM) Test**: We evaluate the memory cost of the memoization structure $\mathcal{T}$. Our findings indicate that: i): On average, the memory cost of $\mathcal{T}$ accounts for only 10.21% of that of G_c and on average across all the three datasets; and ii) The combined memory cost of $G_c + \mathcal{T}$ still remains significantly

Arxiv	|V|+|E|	GM	MSM	PM
G	1,335,586	101.13 MB	-	1.44 GB
G_c	1,078,756	62.44 MB	3.75 MB	1.23 GB
mcr	19.23% (ncr)	38.26%	-	14.58%
Yelp	|V|+|E|	GM	MSM	PM
G	14,671,666	1036.03 MB	-	14.41 GB
G_c	13,711,624	812.50 MB	50.27 MB	13.24 GB
mcr	6.54% (ncr)	21.58%	-	8.09%
Ogbn-Products	|V|+|E|	GM	MSM	PM
G	64,308,169	1887.47 MB	-	30.12 GB
G_c	22,211,452	536.50 MB	83.55 MB	10.49 GB
mcr	65.46% (ncr)	76.01%	-	65.19%

Table 8: Memory Cost comparison between the original graph G and compressed graph G_c using SPGC. Graph memory (GM) is measured in the PyTorch Geometric Graph Data format, memoization structure ($\mathcal{T}$) memory (MSM) is measured directly, and peak memory (PM) represents the peak memory usage during inference with a 3-layer GCN (hidden size = 32).

	Compression Scheme	GCN	GAT	GraphSAGE
Inference Accuracy	SPGC	**0.59**	**0.58**	**0.59**
	SPGC_w/o_$\mathcal{T}$	0.42	0.44	0.37
	SPGC_w/o_$\mathcal{T}$_w_1-hop	0.45	0.52	0.47
Inference Speed-up	SPGC	2.27	2.42	3.94
	SPGC_w/o_$\mathcal{T}$	**2.57**	**2.85**	**4.24**
	SPGC_w/o_$\mathcal{T}$_w_1-hop	2.18	2.23	3.54

Table 9: Ablation Studies of Memoization Structure $\mathcal{T}$ and and Neighbor Decompression w.r.t. Accuracy and Speed-up.

smaller than that of G, with reductions of 34.56%, 16.72%, and 71.58% for ***Arxiv***, **Yelp**, and ***Ogbn-Products*** respectively;

(3) **Peak Memory (PM) Test**: We measure the peak memory usage during the inference process using a 3-layers GCN model, using a hidden size of 32. Compared to the original graph G, the compressed G_c reduces the peak memory by 14.58%, 8.09%, and 65.19% for ***Arxiv***, **Yelp**, and ***Ogbn-Products***. Notably, as the size of graph increases, the reduction in peak memory becomes more pronounced, reflecting the overall decrease in both graph size and memory footprint.

Our memory cost analysis demonstrates the significant efficiency gains achieved after compressing G into G_c using SPGC. Through the three key evaluations - Graph Memory (GM), Memoziation Structure Memory (MSM), and Peak Memory (PM), we observe consistent reductions in memory consumption across all three datasets.

Ablation Analysis. We next investigate how memoization structure $\mathcal{T}$, and neighbor recovery affect the inference accuracy and speed-up achieved by SPGC. We also investigated the impact of aggregation method and improvement of model training cost. We present additional tests with details in [1]. We conduct ablation analysis using **Arxiv** to compare SPGC with its two variants: SPGC _w/o_$\mathcal{T}$: SPGC without $\mathcal{T}$, and SPGC _w/o_$\mathcal{T}$_w_1-hop: SPGC without $\mathcal{T}$, but with 1-hop neighbor decompression. We find the following (as shown in Table 9). (1) Incorporating $\mathcal{T}$ into SPGC results in a noteworthy 43.13% increase in inference accuracy, at the cost of a marginal 12.12% reduction in inference speed-up compared to SPGC _w/o_$\mathcal{T}$. This suggests that the memoization effectively improves inference accuracy while incurring only a small overhead in inference cost. (2) Compared to SPGC _w/o_$\mathcal{T}$, SPGC _w/o_$\mathcal{T}$_w_1-hop demonstrates an average improvement of 16.50% in inference accuracy at a cost of smaller inference speed-up. These verifies the adaptiveness of SPGC in trading inference speed up with model inference accuracy as needed.

9.4 Appendix D: Auxiliary Information

Similarity Merging. SPGC and its variants leverage both **structural similarity** and **embedding similarity** to construct similarity-based merging. They prioritize structural similarity as defined by our proposed structural equivalence followed by embedding similarity based on input node features for fine-tuning (see Fig. 3.) Since the original graph G may contain nodes with distinct feature vectors, we apply a featurization pre-processing step to discretize input features. This ensures that numerical values within same range are categorized together after featurization, improving consistency in similarity measures. As shown in Table. 8, compared to original G, the compressed G_c from SPGC achieves a reduction of the graph size ($|V| + |E|$) by 19.23%, 6.54%, and 65.46% for ***Arxiv***, **Yelp**, and ***Ogbn-Products*** respectively. We define a new metric to directly quantify the percent of similar nodes psn = $1 - \frac{|V_{dis}|}{|V|}$ in SPGC, where $|V_{dis}|$ is the number of stand-alone nodes in R^S such that they are not merged with any nodes in V. For instance, using the equivalence relation defined by SPGC, we found 43.06% nodes in **Arxiv** and 87.42% nodes in ***Ogbn-Products*** are similar by the similarity definition of SPGC. For SPGC, the extent of similarity merging in an arbitrary graph G depends on its inherent structural and embedding similarities. To enhance the performance of compression, our new variant (α, r)-SPGC, introduces configurable parameters α and r, enabling fine-tuned control over the merging process and improving the compression ratio.