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 [29] and link prediction [53]. 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 [12, 51, 56]. 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 [12] (as summarized in Table 3). This can be prohibitively expensive for large real-world graphs.

$$\begin{tikzcd} G \arrow[r, "\mathcal{C}"] \arrow[d, "M"] & G_c \arrow[d, "M"] \\ M(G) \arrow[r, phantom, "\cong"] & M((G_c)) \end{tikzcd}$$

Figure 1: Graph Compression for GNN Inference

(2) *Real-time Inference.* GNN models have been developed for *e.g.,* traffic analysis [40], social recommendation [16], energy forecasting [2, 17, 28], cybersecurity [58], computer vision [44], and edge devices [56]. Such scenarios often require real-time response at *e.g.,* milliseconds [56]. 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 [23, 41, 46, 55, 57]. 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 [39], (learning) to optimize inference process [45], or data sampling [13]. 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, 21], 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 $\mathcal{M}(G)$.

The compression scheme is illustrated in Fig. 1. 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 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 $\mathcal{M}$ [43] as a node classifier that assigns role labels {supplier, doctor, nurse, patient} in a social healthcare network G_1 (illustrated in Fig. 2). 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 p), 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.

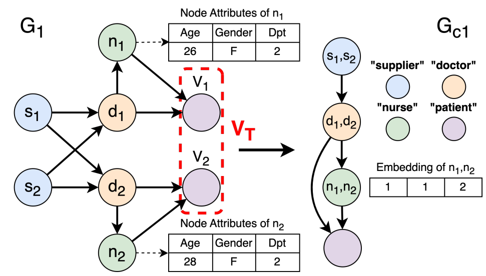
Note that feature equivalence does not necessarily mean that x_1 and x_2 have the same attribute values. For example, while n_1 and n_2 refer to a 26 years old and a 28 years old nurse, respectively, their ages, gender, and profession fall in the same group via (categorical or one-hot) encoding, hence 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:

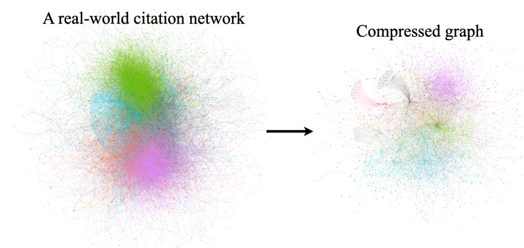
$$X_{v_1}^k = X_{v_2}^k = 2 \times X_{[v]}^k = 2 \times \sigma(\Theta \cdot X_{[n]}^{k-1})$$

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. □



(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]).



(b) Visualization of a fraction of real-world citation network [26] ($|G| = 1,335,586$) 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 2: Compression Scheme to scale node classification

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. 2 (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

Methods	Category	LB	MA	IFGS	Compression Cost
Dspar [35]	S	×	✓	✓	$O(\frac{\|V\|log\|V\|}{\epsilon^2})$
AdaptiveGCN [33]	S	✓	×	×	n/a
NeuralSparse [54]	S	✓	✓	×	$O(q\|E\|)$
SCAL [27]	C	✓	✓	×	n/a
FGC [30]	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**. .**

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 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.

2 RELATED WORK

Several approaches have been developed to accelerate GNN inference in large graphs [13, 19, 39, 56]. There are two main categories: model optimization and graph reduction.

Model optimization. Model optimization strategies refines GNN architecture to reduce model parameters and improve message passing to accelerate inference and training [39]. There are three general strategies: *model pruning* [13, 31, 53], which prunes k least important model parameters to reduce inference cost; *sparse training* [39], which directly drops model parameters with values that are close to 0 at training time, hence reducing the overall cost; and *model quantization* (*e.g.,* Degree-Quant [45]), which reduce precision in terms of decimal places of the model weights to decrease the computational and memory cost of inference.

These approaches are "model-specific" and require known model parameter values (weights) and architectures, and incur additional overhead for learning and fine-tuning to reduce inference costs. In contrast, our approach takes a model-agnostic approach. It performs a "once-for-all" compression to produce a small graph that can be directly processed by any GNN that adopts the same inference process in low PTIME cost, without additional learning overhead.

Graph reduction. Closer to our approach is graph reduction, which simplifies graphs to reduce inference cost at a small sacrifice of model performance [24, 34]. There are two strategies.

(1) Graph *sparsification* (learns 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 [54] learns supervised DNNs to remove task-irrelevant edges. [13] 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 [35] induces smaller subgraphs by removing edges that have similar "importance" (quantified by approximating a resistance measure as in circuits) to preserve graph spectral information.

(2) Graph *coarsening* groups and amalgamates nodes into groups, without removing nodes. For example, SCAL [27] proposed the use of off-the-shelf coarsening methods LV[37] 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. [30] 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 [37]. GRAPE [49] 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] studies compression to accelerate GNN learning and reduce memory cost, 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-based compression.

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.

3 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]

Inference. We take a query language perspective [6, 21] 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

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

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 $N(v) \subseteq \{u|(u, v) \in E$ 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 [12, 56] (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) [29] 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.

4 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

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.

- $\mathcal{C}$ 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 $v \sim_M^L$. 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* $k \in [1, L]$. □

[1] A feature vector X_v of a node v can be a word embedding or one-hot encoding [20] of $T(v)$. A is often normalized as $\hat{A} = A + I$, where I is the identity matrix.

GNNs Classes	Node Update Function (general form)	Training Cost	Inference Cost
Vanilla [43]	$X_v^k = \sigma(\Theta \cdot \text{AGG}(X_u^{k-1}, \forall u \in \mathcal{N}(v)))$	$O(L\lvert E\rvert + L\lvert V\rvert)$	$O(L\lvert E\rvert + L\lvert V\rvert)$
GCN [12, 29]	$X_v^k = \sigma(\Theta^k(\sum_{u \in \mathcal{N}(v)} \frac{1}{\sqrt{d_u d_v}} x_u^{k-1}))$	$O(L\lvert E\rvert F + L\lvert V\rvert F^2)$	$O(L\lvert E\rvert F + L\lvert V\rvert F^2)$
GAT [9, 47]	$X_v^k = \sigma(\sum_{u \in \mathcal{N}(v)} \alpha_{uv} \Theta^k X_u^{k-1}))$	$O(L\lvert E\rvert dF^2 + L\lvert V\rvert F^2)$	$O(L\lvert E\rvert dF^2 + L\lvert V\rvert F^2)$
GraphSAGE [12, 22]	$X_v^k = \sigma(\Theta^k \cdot (X_v^{k-1} \Vert \text{AGG}(X_u^{k-1}, \forall u \in \mathcal{N}(v))))$	$O(L\lvert V\rvert dF + L\lvert E\rvert dF^2)$	$O(L\lvert V\rvert dF + L\lvert E\rvert dF^2)$
GIN [9, 50]	$X_v^k = \sigma(\text{MLP}((1+\gamma)x_v^{k-1} + \sum_{u \in \mathcal{N}(v)} x_u^{k-1})$	$O(L\lvert E\rvert F + L\lvert V\rvert F^2)$	$O(L\lvert E\rvert F + L\lvert V\rvert F^2)$

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.**

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 than one, *i.e.,* $|G_c| < |G|$. As function $\mathcal{P}$ does not introduce new node or edge to G_c, we have $|C(G))| = |G_c| < |\mathcal{P}(G_c)| < |G|$. (2) To see $\mathcal{M}(G) = \mathcal{M}(\mathcal{P}(C(G)))$, *i.e.,* G_c preserves inference result, it suffices to show that for every node $v \in G$, $\mathcal{M}(G, \{v\}) = M(\mathcal{P}(C(G)), \{[v]\})$. 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 $(C, \mathcal{P})$ is an IFGC. □

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

5 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 [36], and simulation equivalence of Kripke structures in model checking [4, 15]. By enforcing equivalence on embeddings and neighborhood connectivity, it ensures an IFGC to accelerate GNN inference *without* decompression.

5.1 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 $(C, \mathcal{P})$ where (1) 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. 3, 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, $d_2 \nsim_M^L d_1$, and $d_2 \nsim_M^L d_3$ 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 [15], and Automorphism [49], 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

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

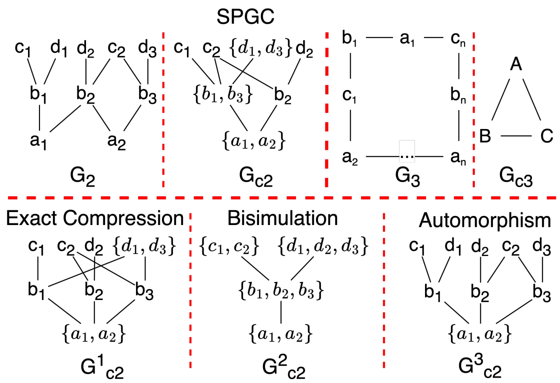


Figure 3: Compressing graphs with SPGC **and a comparison with exact compression [10] , bisimulation [15], and automorphism (adapted from [49]). Exact compression uses node features for initial coloring and $d = 2$ for color refinement.**

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 [15] ignores embedding equivalence and groups 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 [49] 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 5.3). □

5.2 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. □

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 $\mathcal{T}$;

set $R^S := \emptyset$; set $EC := \{V\}$; set $\mathcal{T} := \emptyset$; graph $G_c := \emptyset$;
$R^S :=$ DPP(G);
$R^S := R^S \setminus \{(v, v') \mid X^0_v \neq X'^0_v\}$;
$EC := V/R^S$; /* induce partition EC from refined R^S*/
$(G_c, \mathcal{T}) :=$ CompressG $(\mathcal{T}, G_c, EC, G, M)$;
return G_c and $\mathcal{T}$;

Figure 4: Algorithm SPGC

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, 21].

Discriminative set of GNNs [21]. 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 [42], 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.,* $\mathcal{G}_c = \{G_c | G_c = C(G); G \in \mathcal{G}\}$. 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 discriminaiveness 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].

5.3 Compression Algorithm

We next present a compression algorithm (function 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. 4, 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 [15] to compute the maximum structural

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

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

Figure 5: Procedure CompressG

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 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. Given the induced equivalence classes EC and an encoding of the node update function M_v, procedure CompressG (illustrated in Fig. 5), 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 5.4). 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 $[v]_T(v, [u])$ 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. Such information can be readily obtained by tagging the input GNN class $\mathbb{M}^L$ or encoded as rules. It then updates the entry $[v]_T(v, [u])$ accordingly (lines 6-12).

Example 4: Consider a GNNs class GIN, and graph G_4 shown in Fig. 6. (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 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

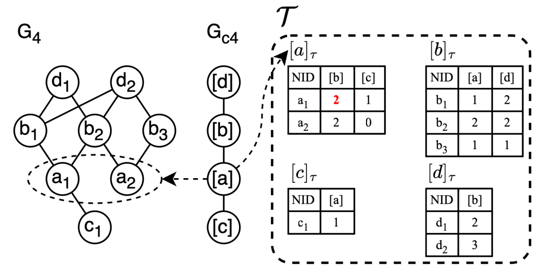


Figure 6: Run-time generation of Memoization structure $\mathcal{T}$.

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.

5.4 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. 6). 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.

6 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

GNNs	Node Update Function M_v	equivalent rewriting $M_{[v]}$; scaling factors are marked in red	notes
Vanilla [43]	$X_v^k = \sigma(\Theta \cdot \text{AGG}(X_u^{k-1}, \forall u \in \mathcal{N}(v)))$	$X_v^k = \sigma(\Theta \cdot \text{AGG}(\mathbf{[v]_T(v,[u])}X_{[u]}^{k-1}, \forall [u] \in \mathcal{N}([v])))$	AGG: $\sum$ or AVG; for AVG, need to multiple by RF_v
GCN [29]	$X_v^k = \sigma(\Theta^k(\sum_{u \in \mathcal{N}(v)} \frac{1}{\sqrt{deg_u deg_v}} x_u^{k-1}))$	$X_v^k = \sigma(\Theta^k(\sum_{[u] \in \mathcal{N}([v])} \frac{1}{\sqrt{\mathbf{deg_v}}} \mathbf{[v]_T(v,[u])} x_{[u]}^{k-1}))$	deg_v: degree of node v in G, topology sensitive
GAT [47]	$X_v^k = \sigma(\sum_{u \in \mathcal{N}(v)} \alpha_{uv}\Theta^k X_u^{k-1}))$	$X_v^k = \sigma(\sum_{[u] \in \mathcal{N}([v])} \mathbf{[v]_T(v,[u])}\Theta^k X_{[u]}^{k-1}))$	weight sensitive
GraphSAGE [22]	$X_v^k = \sigma(\Theta^k \cdot (X_v^{k-1} \Vert \text{AGG}(X_u^{k-1}, \forall u \in \mathcal{N}(v))))$	$X_v^k = \sigma(\Theta^k \cdot (X_{[v]}^{k-1} \Vert \text{AGG}(\mathbf{RF_v} \times \mathbf{[v]_T(v,[u])}X_{[u]}^{k-1}, \forall [u] \in \mathcal{N}([v]))))$	$\Vert$: concatenation; AGG: AVG
GIN [50]	$X_v^k = \sigma(\text{MLP}((1+\gamma)x_v^{k-1} + \sum_{u \in \mathcal{N}(v)} x_u^{k-1}))$	$X_v^k = \sigma(\text{MLP}((1+\gamma)x_{[v]}^{k-1} + \sum_{[u] \in \mathcal{N}([v])} \mathbf{[v]_T(v,[u])} x_{[u]}^{k-1}))$	

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

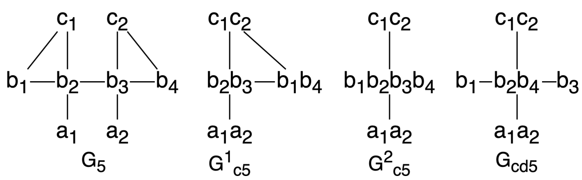


Figure 7: Compressing G_5 with $(0.5, 2)$-SPGC.

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 6.1), and anchored SPGC (Section 6.2), respectively.

6.1 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.

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. 7. 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_c as in SPGC. The difference is that it ad-hocly invokes a decompression procedure decompG (not shown), to reconstruct the neighbors of v, and performs an

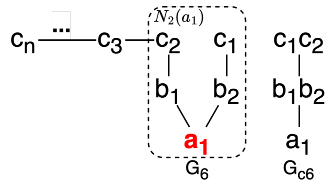


Figure 8: Compressing graph G_6 with anchored SPGC: for 2-layered GNNs, with anchored set $V_A = \{a_1\}$.

inference using original 1-hop neighbors of v to obtain an embedding X_v as close as its original counterpart in G. To minimize decompression cost, decompG extends *Re-Pair*, a reference encoding method [14, 32], for fast decompression from a compact encoding structure. Specifically, it prioritizes the decompression of nodes having most shared neighbors with others in G, to reduce redundant computation, and to "maximize" the chance 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. 7. The decompressed neighbors are kept in G_c 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.

6.2 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, without decompression. Here $|G_L|$ refers to the subgraph of G induced by L-hop of V_A, and $|V_c|$ and $|E_c|$ are both bounded by $|G_L|$.* □

As a constructive proof, we introduce a notion of anchored relation, and construct such an IFGC as an anchored SPGC.

Relative inference friendly. 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^A_L *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^A_L is an equivalence relation over V, and (2) $R^S = R^A_L$ if $V_A = V$, and L is larger than the diameter of G.

Example 7: Consider a class of GNNs $\mathcal{M}^2$, and graph G_6 with $V_A = \{a_1\}$ (shown in Fig. 8). For layer 2 GNNs, ASPGC first induces a subgraph G^L_6 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^L_6 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 $(c_1, c_2) \in R^A_L$ 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. □

Anchored SPGC. Given graph G and an anchor set V_A, an anchored SPGC, denoted as ASPGC, is a graph compression where C computes a compressed graph that is the quotient graph of R^A_L.

Lemma 8: *Given G with a set of anchored nodes V_A, and $\mathcal{M}^L$, the anchored* SPGC *w.r.t. V_A is an* IFGC *that preserves inference results for V_A without decompression.* □

Compression and Inference. The computation of the compressed graph G_c using ASPGC simply follows from its SPGC counterpart. The only difference is that it first induces a subgraph G_L of G using the L-hop neighbors of all the nodes in V_A. It then invokes the compression algorithm of SPGC to derive R^A_L. The inference process over G_c, similarly, follows its SPGC counterpart over G_c.

Analysis. We observe the following. (1) 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. Here G_c can only preserve the inference output for V_A, but not for nodes in $V \setminus V_A$. (2) 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.

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** [38], a citation network where nodes represent documents, and edges are citations among the documents; (2) **Arxiv** [26], an academic collaboration network with nodes representing arXiv papers and edges denoting one paper cites another one; (3) **Yelp** [52] comprises a dense network of user-business interactions, and (4) ***Products*** [26], a product co-purchase network, where nodes represent products sold in Amazon and edges denote products purchased together.

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 [25] (a citation network) with a small world generator [48].

Graph Neural Networks. We have pre-trained three classes of representative GNNs: GCNs[3] [29], GATs[3] [47], and GraphSAGE[3] [22], 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 [35], a graph sparsification method that performs edge down-sampling to preserve graph spectral information, and (2) FGC [30], 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.

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 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 [18] 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].

7.2 Experimental Results

Exp-1: Effectiveness: Accuracy vs. Speed-up. Fig. 9 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. Here a test annotated as "compression method-GNN" refers to the setting that a GNN inference is applied

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

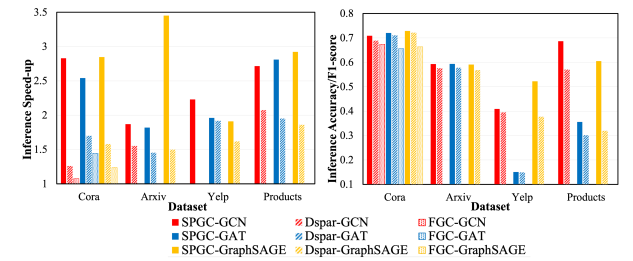
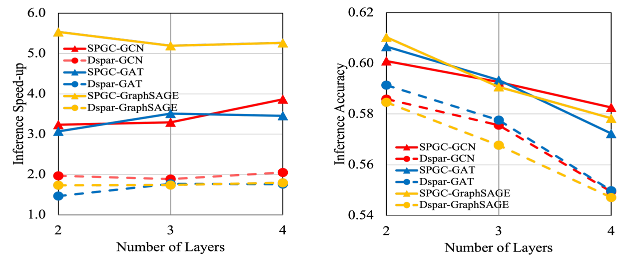


Figure 9: Comparison of $(0.5, 1)$-SPGC with the Baselines in Inference Speed-up and Inference Accuracy/F1-score.



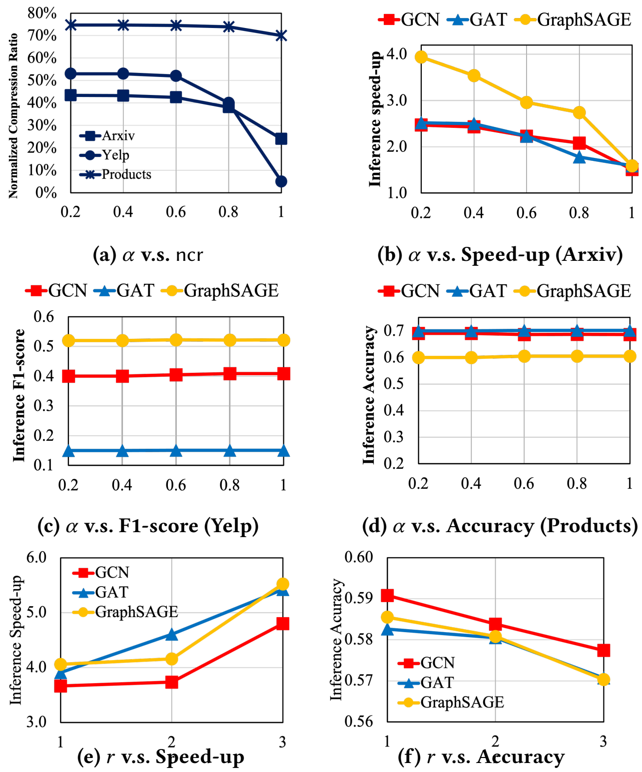
(a) Num. Layers v.s. Speed-up **(b) Num. Layers v.s. Accuracy**

Figure 10: Varying Num. Layers in GNNs (Arxiv).

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 (not shown). For example, for ***ogbn-products***, SPGC achieves ncr up to 74.5% while DSpar and FGC achieves 46.2% and 30.7% respectively. (3) SPGC also consistently outperforms DSpar and FGC in terms of its ability to preserve inference results for GNNs, and retain 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. 10(a) (resp. 10(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.



(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 11: Varying α **and** r **in** (α, r)-SPGC

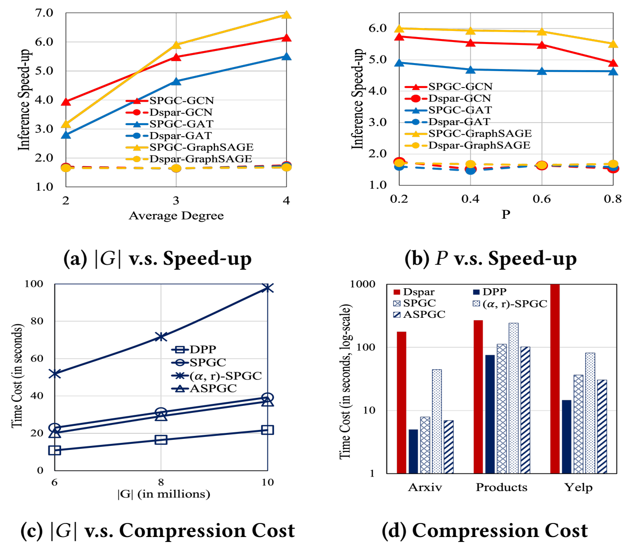
Varying α. Fixing $r = 1$, we varied α from 0.2 to 1, and report the results in Figs. 11(a) to 11(d). It tells us the followings.

(1) As α is increased from 0.2 to 1, ncr drops as illustrated in Fig. 11(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. 11(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.11(c) (resp. 11(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. 11. (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.



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

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

Figure 12: Compression: Scalability and Efficiency.

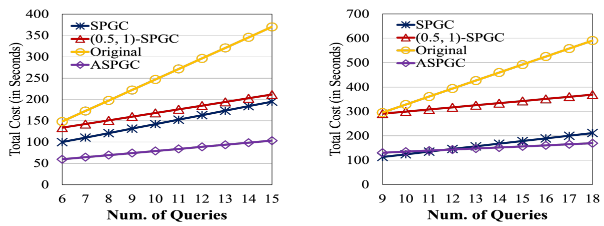
Average degree and "small-world" effect v.s. Inference Speed-up. We simulate **WS-MAG** based on (K, P) Watts-Strogatz algorithm [48] with fixed $|V| = 2M$ from **MAG240M** dataset. K and P represent average degree and re-wiring probability respectively. As P goes up, the less "small-world" (*i.e.,* more random) the graphs become.

Fixing $P = 0.6$, we vary the average degree from 2 to 4. Fig. 12(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 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, 48].

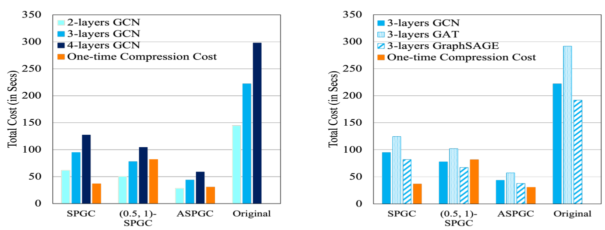
Impact of $|G|$. We increase the size of the simulated graph $|G|$ from $6M$ to $10M$. As shown in Fig. 12(c), (1) SPGC, ASPGC, and DPP all scale well with $|G|$, which is consistent with the cost analysis. (2) (α, r)-SPGC has higher and more sensitive compression cost as $|G|$ increases, as (α, r)-SPGC incurs more time to refine the (α, r)-relation, yet achieves better compression ratio and in turn, more reduction of inference cost.

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. 12(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**



(a) Yelp **(b) Products**

Figure 13: Total Cost: Varying Size of Queries



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

Figure 14: Total Cost: Varying GNN Layers and Class.

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. 13 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 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. 14(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. 14(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. 14(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.

Exp-4: 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 the table below).

	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	<u>0.45</u>	<u>0.52</u>	<u>0.47</u>
Inference Speed-up	SPGC	<u>2.27</u>	<u>2.42</u>	<u>3.94</u>
	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

(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 speed up with model accuracy as needed.

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 or no decompression overhead, and small loss on inference accuracy. A future topic is to evaluate our compression scheme for more GNN classes and tasks.

REFERENCES

[1] 2024. 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] 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).

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

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

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

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

[17] 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).

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

[19] 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).

[20] 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).

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

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

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

[24] 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).

[25] 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).

[26] 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).

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

[28] 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*.

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

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

[31] Aditya Kusupati, Vivek Ramanujan, Raghav Somani, Mitchell Wortsman, Prateek Jain, Sham Kakade, and Ali Farhadi. 2020. Soft threshold weight reparameterization for learnable sparsity. In *ICML*.

[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] 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).

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

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

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

[39] 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.

[40] 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).

[41] 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).

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

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

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

[45] 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).

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

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

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

[49] 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.

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

[51] 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).

[52] 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).

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

[54] 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*.

[55] 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*.

[56] 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).

[57] 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).

[58] 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

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

Proof. (1) It is easy to verify that R^L_M is reflexive, *i.e.,* $(v, v) \in R^L_M$ for any $v \in V$: for any node v, and any of its neighbor v', an identity mapping verifies the reflexiveness of R^L_M. (2) We show the symmetric by definition. If $(v, v') \in R^L_M$, 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 $(u, h(u)) \in R^L_M$. 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^L_M states that for any pair (v_1, v_2) and (v_2, v_3) from G, if $(v_1, v_2) \in R^L_M$ and $(v_2, v_3) \in R^L_M$, then $(v_1, v_3) \in R^L_M$. Let h be the matching relation that induces R^L_M, 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 $(v_1, v_3) \in R^L_M$ as induced by $h' = h \circ h$. The transitivity of R^L_M 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^L_M *w.r.t.* (1) $C(G)$ computes a quotient graph G_c induced by a nontrivial inference equivalent relation R^L_M *w.r.t.* $\mathbb{M}^L$ and G, and (2) $|\mathcal{P}(G_c)| < |G|$.

Proof. Let R^L_M 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^L_M is a nontrivial equivalence relation. As $R^L_M \neq \emptyset$, 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^L_M equivalently as a "matching" relation h between each node v and its equivalence class $[v]$.

Given any two nodes v and v' such that $(v, v') \in R^L_M$, *i.e.,* $v \sim^L_M v'$, consider an equivalent characterization of R^L_M 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 $M(v_i, G) = X^0_{v_i} = X^0_{v_j}$ =$M^0(v_j)$ = $M^0([v]) = M([v], G_c)$.

(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^i_M, $M^i(v, G) = M^i(v', G) = M^i([v], G_c)$ 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 $X^{i+1}_v = M_v(\Theta^i, \text{AGG}, N(v), M^i_v)$, and $X^{i+1}_{[v]} = M_{[v]}(\Theta^i, \text{AGG}, N([v]), M^i_{[v]})$. 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,
- $M^i_v = M^i_{[v]}$ 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 $X^i_u = X^i_{[u]}$, ensured by R^i_M and the definition of quotient graphs.

We now show that $X^{k+1}_v = M_v(\Theta^k, \text{AGG}, N(v), M^k_v)$, $= M_{[v]}(\Theta^k, \text{AGG}, N([v]), M^k_{[v]}) = X^{k+1}_{[v]}$. 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-preserving 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 5, 6.1 and 6.2, 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. $\square$

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-preserving 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. $\square$

Proof of Corollary ??. 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. $\square$

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 $(G'_c, G_c) \in \mathcal{G}_{c\mathcal{M}}$. $\square$

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

Algorithm 3 Procedure DPP(G)

1: **for** $v \in V$ **do**
2: compute rank(n);
3: $\phi := \max\{\text{rank}(n)\}$;
4: node partition Par $:= \{B_i : i = -\infty, 0, \dots, \phi\}$
5: collapse $B_{-\infty}$;
6: **for** $i = -\infty, 0, \dots, \phi$ **do**
7: induce R_i at rank i;
8: **for** $n \in V \cap B_{-\infty}$ **do**
9: **for** $i = 0, \dots, \phi$ **do**
10: $R_i := R_i \setminus \{(v, v') | (v, n) \in E \text{ and } (v', n) \notin E\}$;
11: update B_i;
12: **for** $i = 0, , \dots, \phi$ **do**
13: $D_i := \{X \in \text{Par} : X \subseteq B_i\}$;
14: refine D_i with **Paige-Tarjan**;
15: collapse $X \mid \forall X \in D_i$;
16: **for** $n \in V \cap B_i$ **do**
17: **for** $j = i + 1, \dots, \phi$ **do**
18: $R_j := R_j \setminus \{(v, v') | (v, n) \in E \text{ and } (v', n) \notin E\}$;
19: update B_j;
20: $R := \bigcup_{i \in \{-\infty, 0, \dots, \phi\}} R_i$;
21: **return** R

Figure 15: 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 AL_c and rules;

set $R^{(\alpha,r)} := \emptyset$; set $EC := \{V\}$; set $\mathcal{T}' := \emptyset$; set $AL := \emptyset$; set $\text{AL}_\text{c} := \emptyset$; 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;
$R^{(\alpha,r)} := \text{DPP}(G_r)$;
$R^{(\alpha,r)} := R^{(\alpha,r)} \setminus \{(v, v') \mid \text{xsim}(X_v^0, X_{v'}^0) < \alpha\}$;
$EC := V / R^{(\alpha,r)}$; /* induce partition EC from $R^{(\alpha,r)}$ */
$G_c, \mathcal{T}$:= CompressG ($\mathcal{T}$, G_c, EC, G, M);
AL_c, rules := Re − Pair(AL);
return $G_c, \mathcal{T}, EC, \text{AL}_\text{c}$, rules;

Figure 16: Algorithm (α, r)**-SPGC**

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 $j \in \{i + 1, \dots, \phi\}$ 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. 5.4, 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-preserving 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 preserving. 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 [14, 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) and 2) common r-hops neighbors shared by nodes within in same equivalence relation.

Algorithm 5 : decompG

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

$G_{cd} = G_c$;
sort AL_c by node degrees in G_r;
for $[v] \in G_c.V_c$ **do**
while $\exists$ not visited $v \in [v]$ **do**
$D = \{u | u \in N_r(v) \land \neg$ 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 17: Algorithm decompG

Decompression Algorithm. We next outline decompG shown in Fig. 17 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 8: We continue with Example. 6. Given G_{c5}, EC, AL_c, and rules, decompG sorts AL_c in the descending order of node degrees while still keeping it in the order of equivalent class. decompG next decompresses $N_r([a])$ such that the nodes $b_1 b_2 b_3 c_1$ are decompressed first by a_1 and then $b_4 c_2$ are decompressed by a_2 as illustrated in the Fig. 18. Similarly, $N_r([b])$ and $N_r([c])$ are decompressed accordingly. decompG returns the decompressed graph G_{cd} as shown in the Fig. 7. □

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).

AL

v	$N_{r=2}(v)$
a_1	$b_1b_2b_3c_1$
a_2	$b_2b_3b_4c_2$
b_1	$a_1b_2b_3c_1$
b_2	$a_1a_2b_1b_3b_4c_1c_2$
b_3	$a_1a_2b_1b_2b_4c_1c_2$
b_4	$a_2b_2b_3c_2$
c_1	$a_1b_1b_2b_3$
c_2	$a_2b_2b_3b_4$

Re-Pair

AL_c

v	$N_{r=2}(v)$
a_1	b_1Y
a_2	Xb_4c_2
b_1	Hc_1
b_2	Lb_3Q
b_3	Lb_2Q
b_4	Hc_2
c_1	a_1b_1X
c_2	Hb_4

$Rules = \{X : b2b3, Y : Xc_1, M : a_1a_2, N : c_1c_2, H : a_2X, L : Mb_1, Q : b_4N\}$

decompG

Decompressed

v	D
a_1	$b_1b_2b_3c_1$
a_2	b_4c_2
b_2	$a_1a_2c_1c_2$
b_3	$\emptyset$
b_1	$\emptyset$
b_4	$\emptyset$
c_1	$a_1b_1b_2b_3$
c_2	a_2b_4

Figure 18: Illustration of the compression by Re − Pair **and decompression by** decompG (G_5 **in Fig. 7 as the example).**

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

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 11 line 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.