Graph Compression for Graph Neural Network Inference at Scale

ABSTRACT

Graph Neural Networks (GNNs) have demonstrated promising performance in graph analysis. Nevertheless, the inference process of GNNs remains to be costly, hindering their applications for large graphs. This paper proposes *inference-preserving graph compression* (IPGC), a graph data compression scheme to scale GNNs inference. Given a graph G and a GNN M, an IPGC computes a small compressed graph G_c from G to best preserve the inference results of M over G, such that the result can be directly computed by accessing G_c with no or little decompression cost. (1) We characterize IPGC with a class of inference equivalence relation. The relation captures node pairs in G that are not distinguishable by the inference process of M. We introduce a sufficient condition for the existence of IPGC. (2) Based on the condition, we introduce three practical specifications of IPGC for representative GNN classes, notably, structural preserving compression (SPGC), which computes G_c that can be directly processed by GNN inference without decompression; and two configurable variants: (α, r)-compression, that allows for better compression ratio, and anchored compression, that preserves inference output for specific nodes of interests. For each compression scheme, we introduce compression and inference algorithms with provable guarantees on the efficiency and quality of the inference results. We conduct extensive experiments on a diverse sets of large-scale graphs, verifying the effectiveness and efficiency of our inference-preserving 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 [54]. In a nutshell, a GNN $\mathcal{M}$ can be treated as a function that converts a feature representation of a graph G (a pair (X, A) of feature matrix X and an adjacency matrix A), to a desired vector representation ("embeddings") via multiple layers. For each node, each layer applies a same "update function" to 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 the model parameters ("weights") and obtain a proper update function that best fits training data. The *inference* of $\mathcal{M}$ applies the learned update functions to compute the embeddings, followed by a post-processing to generate the results (*e.g.,* labels) for downstream tasks (*e.g.,* node classification).

Despite their ability to improve the accuracy, GNNs incur expensive inference process when G is large [12, 52, 56]. The emerging need for large-scale testing, fine-tuning, and benchmarking of graph models, require fast inferences of (pre-trained) GNNs under various configurations. Consider the following scenarios.

(1) *Inference in large networks.* Large real-world graphs may involve billions of nodes and edges. 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 1). This can be prohibitively expensive.

GNNs	Training cost	Inference cost
Vanilla	$O(L\|E\| + L\|V\|)$	$O(L\|E\| + L\|V\|)$
GCN [12]	$O(L\|E\|F + L\|V\|F^2)$	$O(L\|E\|F + L\|V\|F^2)$
GAT [10]	$O(L\|E\|dF^2 + L\|V\|F^2)$	$O(L\|E\|dF^2 + L\|V\|F^2)$
GraphSAGE [12]	$O(\|V\|s_nF + \|E\|s_nF^2)$	$O(\|V\|s_nF + \|V\|s_nF^2)$
GIN [10]	$O(L\|E\|F + L\|V\|F^2)$	$O(L\|E\|F + L\|V\|F^2)$

Table 1: Training & Inference costs of representative GNNs. **L, $|E|$, $|V|$, F, s_n and d denote size of layers, edges, nodes, features per node, sampled neighbors per node, and maximum node degree of G, respectively. Here** GIN **also adopts MLP layers.**

(2) *Real-time Inference.* GNN models have been developed for *e.g.,* traffic analysis [41, 51], social recommendation [17], energy forecasting [2, 3, 18, 28], cybersecurity [58], computer vision [32, 45], 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 m and n are large).

(3) *GNNs in domain sciences.* GNNs have been adopted to support data-driven scientific workflows for *e.g.,* material science, biology, medicine, social science, and geosciences [42, 47, 57]. The need for scaling GNN inference over large scientific knowledge graphs is evident for experiment optimization and validation.

Several approaches have been developed to accelerate GNN inference, by simplifying model architectures [40], optimizing inference process [46], or data sampling [13]. These "model-specific" methods typically require internal GNN layers and parameter values, and incur considerable training overhead.

Compressing graphs for GNN **inference**. Unlike prior approaches, we advocate a model-agnostic, "once-for-all" graph compression scheme (Fig. 1). Our design is inspired by two observations, abstracting model inference as a *query evaluation* process in G:

GNNs *as queries.* Inspired by the query language nature of GNN inference [7, 22], a GNN $\mathcal{M}$ can be viewed as an "inference query" that requests outputs embeddings of the nodes in an input graph G. The inference process of a GNN $\mathcal{M}$ evaluates a inference query (simply denoted as M), by recursively apply a node update function.

Query-preserving compression [16] compress G to a smaller counterpart G_c, such that G_c can be directly queried to obtain the original query result, $(Q(G) = Q(G_c))$, for any query Q from a query class.

In a nutshell, one may use a compression function $\mathcal{C}$ to compress a large graph G to a small counterparts G_c "once-for-all", such that (1) G_c preserve inference output for a set of GNNs $\mathcal{M}$ with consistent inference process, and (2) the inference process can be directly applied on and only refers to G_c, with no or small decompression overhead (by necessary post-processing function $\mathcal{P}$), to obtain $M(G)$, with guaranteed reduced inference cost.

Is such a compression scheme doable? We illustrate an intuition with the following example.

Example 1: Consider a 3-layer GNN $\mathcal{M}$ for a node classification task that assigns role labels { supplier, doctor, nurse, patient} in a

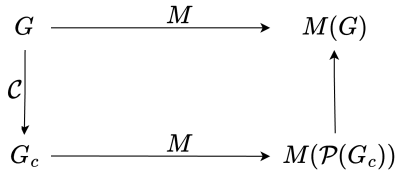


Figure 1: Graph Compression for GNN Inference

social healthcare network. Each node has attributes such as roles, age group, department, etc. To infer the roles of unlabeled nodes $V_T = \{v_1, v_2\}$, an inference process starts by propagating a node feature matrix X of size 8×8 with a node update function M. Via a 3-layer forward message passing, the embeddings of v_1 and v_2 are computed, quantifying the likelihood of p_1 and p_2 being assigned to one of the four labels. As the probability of assigning "patient" are highest, it correctly infers the labels of both as "patient".

An encoding function may assert that the initial features of the nodes n_1 and n_2 are the same after featurization, due to that they have similar attribute values. For example, while n_1 and n_2 refer to a 26-years old female nurse and a 28-years old female nurse, respectively, their ages, gender, and profession fall in the same group via (categorical or one-hot) encoding. In this case, both n_1 and n_2 have the same initial embedding.

Take a closer look at the node update function M of the GNN:

$$X_i^k = \sigma(\Theta \cdot \mathrm{AGG}(X_j^{k-1}, \forall j \in \mathcal{N}(i)))$$

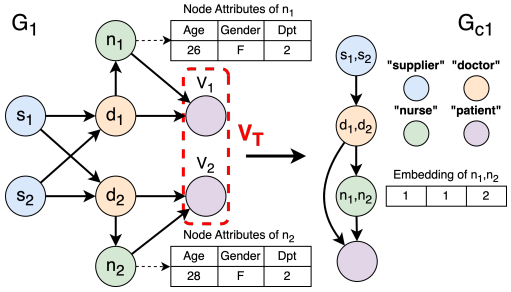
where X_i^{k-1} (resp. X_i^k) is the embedding of node i at the $(k-1)$-th (resp. k-th) layer; AGG is an aggregation operator (*e.g.,* sum $\sum$ or concatenation CONCAT), σ is an activation function, and Θ represents the learnable weight matrix (same across all layers in the GNN). If the initial featurized representation of x_1 and x_2 remains to be the same (with x ranges over s, d, n and p), then the embedding of x_1 and x_2 will remain the same during the inference computation. That is x_1 and x_2 remain to be *indistinguishable* for the inference process using node update function M, for any 3-layered GNN.

By "recursively" merging all such node with the same initial embedding that are also connected to indistinguishable pairs (which include (s_1, s_2), (d_1, d_2), (n_1, n_2), and (p_1, p_2)), a smaller graph G_c can be obtained. Applying the same inference process on G_c shall yield the same embedding and result labels for the test nodes, yet incurs a smaller time cost due to reduced input size. □

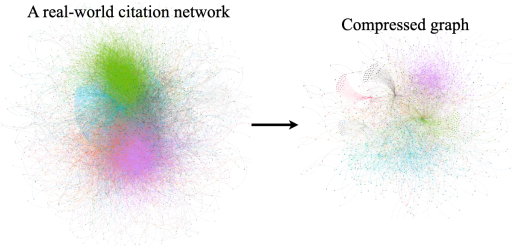
The above example verifies the possibility of a graph compression scheme that finds and compresses nodes that are "equally indistinguishable" for the inference process of GNNs, and by merging them to obtain small structures, which may *preserve* the inferred results of their original counterpart over G. Our study verifies that for representative GNNs, highly compressed structures can be computed with inferred results that either preserve or approximate their original counterparts in G, with no or small loss of accuracy, and a significant reduction in inference cost (see Fig. 2).

Contributions. Our main contributions are as follows.

(1) We formally introduce *inference-preserving graph compression scheme* (IPGC), a general scheme to scale GNN inference to large graphs. Given a graph G, IPGC computes a smaller graph G_c from



(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

G, such that for *any* GNN model that applies a same inference process, G_c preserves the inference result of M over G. The scheme executes a compression function $\mathcal{C}$ to compute G_c "once-for-all", and obtain inference results by only referring to G_c instead of G.

We characterize IPGC with an *inference equivalence* relation, which captures the nodes with learnable 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 IPGC, which specifies G_c as the quotient graph of G induced by the inference equivalence relation.

(2) We specify IPGC for representative GNNs. We first introduce a graph compression scheme, SPGC, that specifies inference equivalence by enforcing equivalence on node embeddings and their neighborhood connectivity. This yields 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 justify SPGC with a uniqueness and minimality property: we show it can produce a unique, smallest G_c up to graph isomorphism among compressed graphs. This establishes a theoretical upper bound for the speed-up it can achieve for inference cost. We also show that SPGC preserves the discriminability of the given GNNs.

(3) We further introduce two *configurable* variants of IPGC to allow flexible trade-off between compression ratio and the quality of inference output. (a) The (α, r)-SPGC adopts an agglomerative node clustering strategy that 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

Methods	Category	LB	MA	IP	Compression Cost
Dspar [36]	S	×	✓	✓	$O(\frac{\lvert V\rvert log\lvert V\rvert}{\epsilon^2})$
AdaptiveGCN [34]	S	✓	×	×	n/a
NeuralSparse [55]	S	✓	✓	×	$O(q\lvert E\rvert)$
SCAL [27]	C	✓	✓	×	n/a
FGC [30]	C	✓	✓	×	$O(\lvert V\rvert^2\lvert V_c\rvert)$
SPGC (Ours)	C	×	✓	✓	$O(\lvert V\rvert + \lvert E\rvert)$
(α, r)-SPGC (Ours)	C	×	✓	✓	$O(\lvert V\rvert\lvert N_r\rvert + \lvert E\rvert)$
ASPGC (Ours)	C	×	✓	✓	$O(\lvert G_L\rvert)$

Table 2: Comparison of graph compression methods. S: Sparsification, C: Coarsening, LB: Learning-Based, MA: Model-Agnostic, IP: Inference-Preserving guarantees, 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.

the inference results for such nodes only, rather than the entire node set. For both variants, we introduce efficient compression and inference algorithms.

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

Related work. Several approaches have been developed to accelerate GNN inference in large graphs [13, 20, 40, 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 [40]). There are three general strategies: *model pruning* [13, 31, 54], which prunes k least important model parameters to reduce inference cost; *sparse training* [40], 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 [46]), which reduce precision in terms of decimal places of the model weights to decrease the computational and memory cost of inference.

These approaches are often "model-specific" and requires known model parameter values (weights) and architectures, and incur additional overhead for learning and fine-tuning to reduce inference cost. 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. The compression and inference remains in low PTIME without additional learning overhead.

Graph reduction. Closer to our approach is graph reduction, which simplifies graphs to reduce inference cost at small sacrifice of model performance [24, 35]. 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 [34] learns an edge predictor to determine and remove task-irrelevant edges to accelerate GNN inference on CPU/GPU clusters. NeuralSparse [55] learns supervised DNNs to remove task-irrelevant edges. [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 [36] 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 super nodes using an aggregation algorithm without removing nodes. For example, SCAL [27] proposed the use of off-the-shelf coarsening methods LV[38] for scaling up GNN training and theoretically proved that coarsening can be considered a type of regularization and may improve the generalization as well as reduce the number of nodes by up to a factor of ten without causing a noticeable downgrade in classification accuracy. [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 [38].

Our work differs from existing graph reduction approaches (summarized in Table. 2) in the following. (1) Our methods remain "model-agnostic" and apply to any GNN that conforms to the same inference process, without requiring model parameters and avoid additional learning overhead. (2) We specify IPGC with variants that preserve or approximate inferred results with provable guarantees. We also establish invariant properties such as uniqueness and minimality of compressed graphs, as well as theoretical bounds for speed up. 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.

Notation	Description
$G = (X, A)$	X: feature matrix, A: adjacency matrix
$\mathcal{M}$, $\mathcal{M}(\cdot)$	a GNNs model, inference process of $\mathcal{M}$
$\mathcal{C}, \mathcal{P}$	compression, post-processing function
M_v^k, x_v^k	update function/embedding for node v at layer k
AGG	aggregation function in GNN (*e.g.,* $\sum$, Avg, ||)
G, G_c	graph, compressed graph of G
IPGC, SPGC	inference/structural preserving graph compression
$\alpha, r, \mathcal{T}$	similarity threshold, num. hops, memoization structure
V_T, V_A	set of test nodes, set of anchored nodes
$R^S, R^{(\alpha,r)}, R_L^A$	structural equivalence, (α, r), anchored relation
EC	induced partition of nodes by relation

Table 3: Summary of Notations.

2 PRELIMINARIES

2.1 Graphs and Graph Neural Networks

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

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

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

GNNs Classes	Node Update Function (general form)
Vanilla [44]	$X_v^k = \sigma(\Theta \cdot \text{AGG}(X_u^{k-1}, \forall u \in \mathcal{N}(v)))$
GCN [29]	$X_v^k = \sigma(\Theta^k(\sum_{u\in\mathcal{N}(v)} \frac{1}{\sqrt{d_u d_v}} x_u^{k-1}))$
GAT [48]	$X_v^k = \sigma(\sum_{u\in\mathcal{N}(v)} \alpha_{uv}\Theta^k X_u^{k-1}))$
GraphSAGE [23]	$X_v^k = \sigma(\Theta^k \cdot (X_v^{k-1}\|\|\text{AGG}(X_u^{k-1}, \forall u \in \mathcal{N}(v))))$
GIN [50]	$X_v^k = \sigma(\text{MLP}((1+\gamma)x_v^{k-1} + \sum_{u\in\mathcal{N}(v)} x_u^{k-1})$

Table 4: Node Update Functions used by representative GNNs. σ: an activation function *e.g.,* ReLU or LeakyReLU. AGG: aggregation function; can be *e.g.,* sum ($\sum$), average (Avg), or concatenation ($\|\|$).

Inference. We take a query language perspective [7, 22] to characterize the inference process of GNNs. A GNN inference process is specified as a composition of *node update functions*.

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

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

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 (denoted as $N^k(v)$). When k=1, X_v^0=$X_v \in X$, *i.e.,* the input feature vector.

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 apply the node update function. We clarify two settings below.

Fixed and Deterministic Inference. (1) 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. (2) 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.

Directed or Undirected Inference. Message-passing based GNNs may or may not follow edge directions. For simplicity, we assume that a proper $N(v)$ is specified in node update function M_v. For a directed G, $N(v)$ may include the nodes connected to v via an incoming or an outgoing edge, *i.e.,* $N(v) \subseteq \{u | (u, v) \in E \text{ or } (v, u) \in E\}$. Note that the inference process does not require G to be connected.

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 4 summarizes several node update functions in their general forms, for mainstream GNNs. All are PTIME computable.

Example 2: Graph Convolution Networks (GCNs) [29] adopt a common node update function as $X_v^k = \sigma(\Theta^k(\sum_{u\in\mathcal{N}(v)} \frac{1}{\sqrt{d_u d_v}} x_u^{k-1}))$. 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 a set of 3-layered GCNs that adopts a node update function that conforms to this general form. □

Remarks. Our characterization *does not* enforce equality of model parameters. Two GNNs in a same class may have different Θ and output different embeddings, given the same input.

Inference query. Given a fixed, deterministic GNN $\mathcal{M}$, a graph G with a set of test nodes V_T, an *inference query* of $\mathcal{M}$ computes the embeddings of the nodes in V_T (denoted as $M(G, V_T)$). By default, $V_T = V$, and the output is simply denoted as $M(G)$.

The embeddings can be post-processed to desired form for downstream tasks. For example, for GNN-based node classification, the output $M(G, V_T)$ is converted to likelihood of labels to be assigned to V_T by a classifier. In general, the inference queries of representative GNNs can be evaluated in PTIME [12, 56] (see Table 1).

3 INFERENCE-PRESERVING COMPRESSION

3.1 Graph Compression

Compression Scheme. 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, such that $V = \bigcup_{[v]\in V_c} [v]$; 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 $(v, v') \in E$.

Given a graph G, a *graph compression scheme* is a pair $(\mathcal{C}, \mathcal{P})$, where (1) $\mathcal{C}$ is a compression function that computes a compressed graph G_c of G, *i.e.,* $G_c = \mathcal{C}(G)$; and (2) $\mathcal{P}$ is (optionally) a post processing function to necessarily decompress G_c by "restoring" information of nodes in G_c to their original counterparts as needed.

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

- $|\mathcal{P}(\mathcal{C}(G))| < |G|$, and
- $M(G) = M(\mathcal{P}(\mathcal{C}(G)))$, for *any* GNN $\mathcal{M} \in \mathbb{M}$.

That is, an IPGC ensures to compute a graph $\mathcal{P}(\mathcal{C}(G))$ with smaller size, such that the result from the inference of *any* GNN in the set $\mathbb{M}$ is preserved by a same inference process over the smaller graph $\mathcal{P}(\mathcal{C}(G))$, at a smaller inference cost.

3.2 A Sufficient Condition

We next introduce a sufficient condition for the existence of IPGC *w.r.t.* a given GNN class. To this end, we first introduce a notion of *inference-equivalent* relation.

Inference equivalence. Given a class of GNN $\mathbb{M}^L$ and a graph G, we say 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 given GNN $M \in \mathbb{M}^L$, $X_v^k = X_{v'}^k$ for any $k \in [0, L]$. The result below tells us that the inference equivalence of nodes ensure that they are "indistinguishable" for the inference process of any specific GNN from a class.

Lemma 1: *Given $\mathbb{M}^L$ and G, for any nodes v and v' in G, and for every $\mathcal{M} \in \mathbb{M}^L$, $M(v, G) = M(v', G)$, if and only if $v \sim_M^L v'$.* □

Proof sketch: The **If** condition can be verified by an induction on the number of layers $k \in [1, L]$. As $v \sim_M^L v'$, $X_v^L = X_{v'}^L$ when k=L; hence $M(v, G) = X_v^L = X_{v'}^L = M(v', G)$. We show the **Only If** condition by contradiction. Assume $M(v, G) = M(v', G)$, yet $v \not\sim_M^L v'$. There there exists a layer $k \in [1, L]$ such that $X_v^L = M_v^L \circ (M_v^{L-1}(\circ(\cdots M_v^{L-k}(X_v^k)) \cdots)) = c^k \cdot X_v^k$; and similarly, $X_{v'}^L = c^k \cdot X_{v'}^k$. Hence $M(v, G) = X_v^L = c^k \cdot X_v^k \neq c^k \cdot X_{v'}^k = X_{v'}^L = M(v', G)$. This contradicts to the assumption that $M(v, G) = M(v', G)$. □

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 2: *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 $(v, v') \in E$.

We now present a *sufficient* condition for the existence of IPGC.

Theorem 3: *Given a class of* GNNs $\mathbb{M}^L$ *and a graph G, there exists a* IPGC $(C, \mathcal{P})$ w.r.t. $\mathbb{M}^L$ *and G, if (1) there is a nontrivial inference equivalent relation R_M^L* w.r.t. $\mathbb{M}^L$ *and G, (2) $C(G)$ correctly computes a quotient graph G_c induced by R_M^L, and (3) $|\mathcal{P}(G_c)| < |G|$.* □

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 2, 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, as ensured by $|\mathcal{P}(G_c)| < |G|$, we have $|C(G))| = |G_c| < |\mathcal{P}(G_c)| < |G|$. (2) To prove that G_c preserves inference result, *i.e.,* $M(G) = M(\mathcal{P}(C(G)))$, it suffices to show that for every node $v \in G$, $M(G, v) = M(\mathcal{P}(C(G)), [v])$. We show this with a proof by induction on the layers of GNNs, *i.e.,* for any $k \in [0, L-1]$, $X_v^{k+1} = M_v(\Theta^k, \text{AGG}, N(v), M_v^k) = M_{[v]}(\Theta^k, \text{AGG}, N([v]), M_{[v]}^k) = X_{[v]}^{k+1}$. This is ensured by two constructions: (a) the fixed deterministic inference process of $M \in \mathcal{M}^k$ that applies the same model parameters Θ and AGG, and (b) set $\mathcal{P}$ as a look-up function to retrieve, at inference time, any auxiliary information which are pre-stored by function C for each node $[v]$ whenever needed (see Table 5 for examples), in constant time, hence losslessly restore the embedding for v. 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 IPGC for $\mathcal{M}^L$ and G. □

We present the detailed proof in anonymous full version [1].

Theorem 3 establishes a *principled strategy* to implement IPGC:

- specify a inference-preserving relation R_M^L;
- set C to compute a quotient graph of G induced by R_M^L, and pre-store auxiliary neigbhorhood information; and
- design a matching function $\mathcal{P}$ to look-up auxiliary neighborhood information as needed, and perform the preserved embeddings as needed by referring to G_c only.

Following this principle, we next specify practical IPGC to scale GNN inference for representative GNN classes, with efficient compression and inference algorithms.

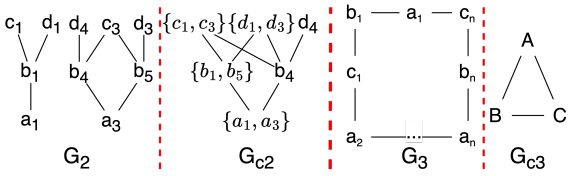


Figure 3: Compressing graphs with SPGC

4 STRUCTURAL-PRESERVING COMPRESSION

We introduce a first IPGC for GNN inference. We specify R_M^L as an extended version of *structural equivalence*. The latter has origins in role equivalence in social science [37], and simulation equivalence of Kripke structures in model checking [5, 15]. By enforcing equivalence on embeddings and neighborhood connectivity, it ensures an IPGC to accelerate GNN inference *without* decompression.

4.1 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 the following holds:

- $X_v^0 = X_{v'}^0$;
- 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 $(u'', u''') \in R^S$.

We show the following result. Intuitively, it tells us that any two nodes that are structural equivalent are "indistinguishable" for GNN inference process.

Theorem 4: *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-preserving relation by definition. □

Structural-preserving Compression. Given $\mathbb{M}^L$ and G, a graph compression scheme $(C, \mathcal{P})$ is a *structural-preserving compression*, denoted as SPGC *w.r.t.* $\mathbb{M}^L$, if

- C computes a non-empty structural equivalence relation R^S in G, and induces the quotient graph G_c of R^S; and
- $\mathcal{P}$ is not needed (or simply an identity function).

Example 3: Consider graphs G_2 and G_3 in Fig. 3.

(1) G_2 has two connected components. The nodes having the same labels 'a', 'c', 'd' also have the same input features, respectively. For example, $X_{a_1}^0 = X_{a_3}^0$, and $X_{d_1}^0 = X_{d_3}^0 = X_{d_5}^0$. For nodes labeled with 'b', $X_{b_1}^0 = X_{b_5}^0 \neq X_{b_4}^0$. One can verify that $R^S = \{(a_1, a_3), (b_1, b_5), (c_1, c_3), (d_1, d_3)\}$. A compressed graph G_{c_2} is illustrated accordingly with only 6 nodes and 7 edges.

Observe that despite d_1, d_4 and d_5 have the same input features, $d_4 \not\sim_M^L d_1$, and $d_4 \not\sim_M^L d_3$ for GNNs with $L \geq 1$. Indeed, d_4 has a neighbor b_4 that has no counterpart in the neighbors of d_1 or d_3 that share the same embedding; hence the output embedding of d_4 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 chain of length-3 paths, where $X^0_{a_i} = X^0_{a_j}$, $X^0_{b_i} = X^0_{b_j}$, and $X^0_{c_i} = X^0_{c_j}$, 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 compressed graph G_{c_3} is illustrated, which has three nodes (equivalent classes) with $A = \{a_i\}$, $B = \{b_i\}$, $C = \{c_i\}$ for $i \in [1, n]$, regardless of how large n is. □

4.2 Properties and Guarantees

We next justify SPGC by investigating "*To what extend can* SPGC *improve* GNN *inference?*" We answer this question by showing a *minimality and uniqueness* property. This in turn provide an optimality guarantees for compression ratio, and an upper bound for speed up of inference cost.

Uniqueness and Minimality. We first show that an SPGC can generate a smallest G_c, which is "unique" up to graph isomorphism. The latter means that if there is another smallest compressed graph G'_c generated by an SPGC, then there exists a graph isomorphism mapping between G^*_c and G'_c.

Lemma 5: *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 $v \sim_M^R v'$ but are not in $[v]$, or $v \not\sim_M^R v'$, but are included in $[v]$. Either leads to contradiction. □

To see how this property benefits speed up of inference cost and compression, we consider a simple indicator, *compression ratio* (denoted as cr), which is defined as $\frac{|G|}{|G_c|}$.

Corollary 6: *Given* $\mathbb{M}^L$ *and* G*,* SPGC *achieves (1) an optimal compression ratio* $\frac{|G|}{|G^*_c|}$*, and (2) a speed-up for the inference of* $\mathbb{M}^L$ *at most* $\frac{d|G|}{|G_c|}$*; where* d *is the maximum node degree of* G*.* □

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 [7, 22].

Discriminative set of GNNs [22]. Given a set of graphs $\mathcal{G}$, the *discriminative set* of a GNN $\mathcal{M}$, denoted as $\mathcal{G}_\mathcal{M}$, refers to the maximum set of pairs $\{(G, G')\}$, where $G, G' \in \mathcal{G}$, such that $M(G) = M(G')$. In the case of equivariant GNNs [43], the strongest discriminativeness can be achieved, for which the set contains all pairs (G, G') such that G and G' are isomorphic [6]. 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.

Algorithm 1 : SPGC

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

Figure 4: Algorithm SPGC

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 7: *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 of Theorems 5-7 and Corollary 6 in [1].

4.3 Compression Algorithm

We next present an algorithm to implement SPGC.

General idea. The algorithm, simply denoted as SPGC, follows the principle strategy (Section 3.2) and Lemma 5 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 5 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 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

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

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

Figure 5: Procedure CompressG

EC, CompressG initializes a node in G_c, (lines 1-3). For each edge $(u, v) \in E$, CompressG adds an edge $([u], [v])$ (lines 4-5).

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

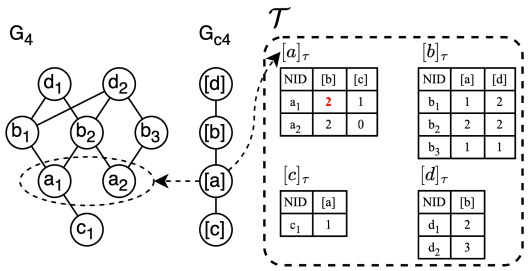


Figure 6: Run-time generation of Memoization structure $\mathcal{T}$.

4.4 Inference Process

Inference algorithm. We outline an algorithm that directly obtains $M(G)$ by referring to G_c^* only, without decompression. Our strategy rewrites the node update function M_v to an equivalent counterpart $M_{[v]}$, that takes as input $[v]$ and the corresponding tuple $[v]_{\mathcal{T}}(v)$ in $\mathcal{T}$, to "scale" the embedding computation with the memorized edge weights. The algorithm performs inference directly in G_c^* with $M_{[v]}$, and simply "re-scale" 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 $[v]_{\mathcal{T}}(v)$. Table 5 illustrates the auxiliary neighborhood information, rewriting of node update functions and scaling factors for representative GNN classes.

Example 5: We continue with Example. 4. Consider inferencing on node a_1, we first look up the values from entries $[a]_T(a_1, [b])$ and $[a]_T(a_1, [c])$ which are equal to 2 and 1 separately as shown in Fig. 6. We next assign these values as scaling factors in Table. 5 to restore message passing to node a_1 as it is in original G_4. □

Inference query cost. As SPGC requires no decompression, an inference query can be directly applied to G_c without incurring additional decompression overhead. The overall inference query 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 bound of inference costs for mainstream GNNs in Table 1. For other and more complex GNNs variants, the inference costs can be derived similarly by scaling down from their counterparts over G.

5 CONFIGURABLE GRAPH COMPRESSION

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

In response, we next introduce two variants of SPGC: (α, r)-SPGC (Section 5), and anchored SPGC (Section 6), respectively.

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

GNNs	Node Update Function M_v	equivalent rewriting $M_{[v]}$; scaling factors are marked in red	notes
Vanilla [44]	$X_v^k = \sigma(\Theta \cdot \text{AGG}(X_u^{k-1}, \forall u \in \mathcal{N}(v)))$	$X_v^k = \sigma(\Theta \cdot \text{AGG}(\mathbf{[v]_T(v,[u])} X_{[u]}^{k-1}, \forall [u] \in \mathcal{N}([v])))$	AGG: $\sum$ or AVG; for AVG, need to multiple by RF_v
GCN [29]	$X_v^k = \sigma(\Theta^k(\sum_{u\in\mathcal{N}(v)} \frac{1}{\sqrt{deg_u deg_v}} x_u^{k-1}))$	$X_v^k = \sigma(\Theta^k(\sum_{[u]\in\mathcal{N}([v])} \frac{1}{\sqrt{\text{deg}_\text{v}}} \mathbf{[v]_T(v,[u])} x_{[u]}^{k-1}))$	deg_v: degree of node v in G, topology sensitive
GAT [48]	$X_v^k = \sigma(\sum_{u\in\mathcal{N}(v)} \alpha_{uv}\Theta^k X_u^{k-1}))$	$X_v^k = \sigma(\sum_{[u]\in\mathcal{N}([v])} \mathbf{[v]_T(v,[u])}\Theta^k X_{[u]}^{k-1}))$	weight sensitive
GraphSAGE [23]	$X_v^k = \sigma(\Theta^k \cdot (X_v^{k-1} \Vert \text{AGG}(X_u^{k-1}, \forall u \in \mathcal{N}(v))))$	$X_v^k = \sigma(\Theta^k \cdot (X_{[v]}^{k-1} \Vert \text{AGG}(\mathbf{RF_v \times [v]_T(v,[u])} X_{[u]}^{k-1}, \forall [u] \in \mathcal{N}([v]))))$	$\Vert$: 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 5: Equivalent rewriting of node update functions for mainstream GNN classes.

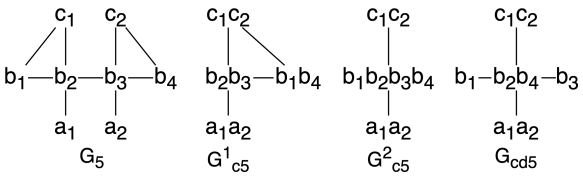


Figure 7: Compressing G_5 with $(0.5, 2)$-SPGC.

(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 $\mathcal{C}$ 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 8: *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-preserving relation, it is no longer an IPGC, 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 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, 33], 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.

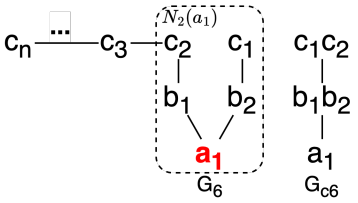


Figure 8: Compressing graph G_6 with anchored SPGC: for 2-layered GNNs, with anchored set $V_A = \{a_1\}$.

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 8.

6 ANCHORED GRAPH COMPRESSION

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

We present our main result below.

Theorem 9: *Given $\mathcal{M}^L$ and G with a set of targeted nodes V_A, there exists an* IPGC *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 IPGC as an anchored SPGC.

Relative inference preserving. 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 $(\mathcal{C}, \mathcal{P})$ is an IPGC relative to V_A, if (1) $|\mathcal{P}(\mathcal{C}(G))| < |G|$; and (2) for any GNN $M \in \mathcal{M}^L$, and any $v \in V_A$, $M(G, v) = M(\mathcal{P}(\mathcal{C}(G), v)$.

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

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

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_6^L with 2-hop neighbors of a_1. It then follows a SPGC counterpart to compute the compressed graph G_{c_6}, with only three nodes. Observe that the compressed graph G_{c_6} does not guarantee to preserve the embedding of other nodes, but only a_1. For inference over node $b_1 \notin V_A$, since G_6^L do not cover within 2-hop neighbors of b_1, G_{c_6} cannot preserve its embedding. One can further verify that (1) the node pair $(c_1, c_2) \in R_L^A$ but $\notin R^S$, and (2) the anchored compression does not need to consider nodes beyond L-hop of anchored nodes (such as the chain from c_3 to c_n), as it best exploits the data locality of GNN inference process "centered" at V_A. □

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
ogbn-products	2.4M	61.9M	47	100

Table 6: Summary of Datasets.

Anchored SPGC. Given graph G and an anchor set V_A, an anchored SPGC, denoted as ASPGC, is a graph compression where $\mathcal{C}$ computes a compressed graph that is the quotient graph of R_L^A.

Lemma 10: *Given G with a set of anchored nodes V_A, and $\mathcal{M}^L$, the anchored* SPGC w.r.t. *V_A is an* IPGC *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_L^A. 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 9 follows.

7 EXPERIMENTAL STUDIES

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 memoization and decompression techniques.

7.1 Experimental Settings

Datasets. We employ four real-world graph benchmark datasets (summarized in Table. 6): (1) **Cora** [39], 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** [53] comprises a dense network of user-business interactions, and (4) ***ogbn-products*** [26], a product co-purchase network, where nodes represent products sold in Amazon and edges denote products purchased together.

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 [49].

Graph Neural Networks. We have pre-trained three classes of representative GNNs: GCNs[3] [29], GATs[3] [48], and GraphSAGE[3] [23], for each dataset. For a fair comparison, (1) for all the datasets, we consider node classification, and (2) all compression methods are applied for the same set of GNNs.

Compression Methods. We compare SPGC and its variants, (α, r)-SPGC and ASPGC, with two state-of-the-art compression methods: (1) DSpar [36], a graph sparsification method that performs edge down-sampling to preserve graph spectral information, and (2) FGC [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 $(\mathcal{C}, \mathcal{P})$ 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 deployed in Python with PyTorch Geometric [19] and BisPy [4] 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[2].

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 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

[2]https://anonymous.4open.science/r/SPGC

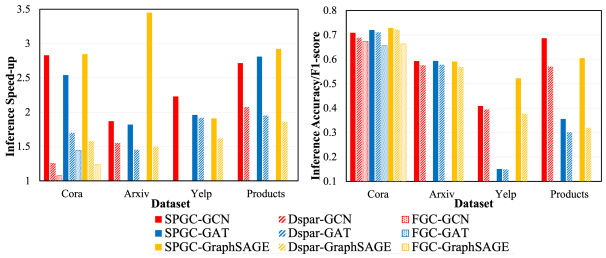
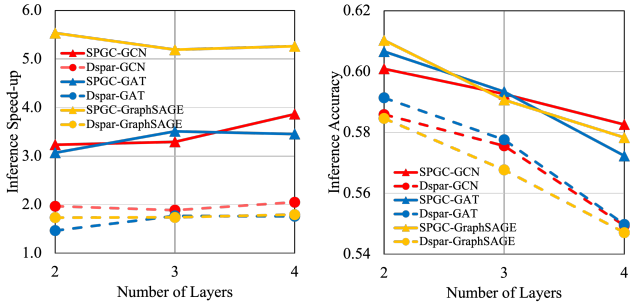


Figure 9: Comparison of $(0.5, 1)$-SPGC with the Baselines in Inference Speed-up and Inference Accuracy/F1-score.



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

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

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.

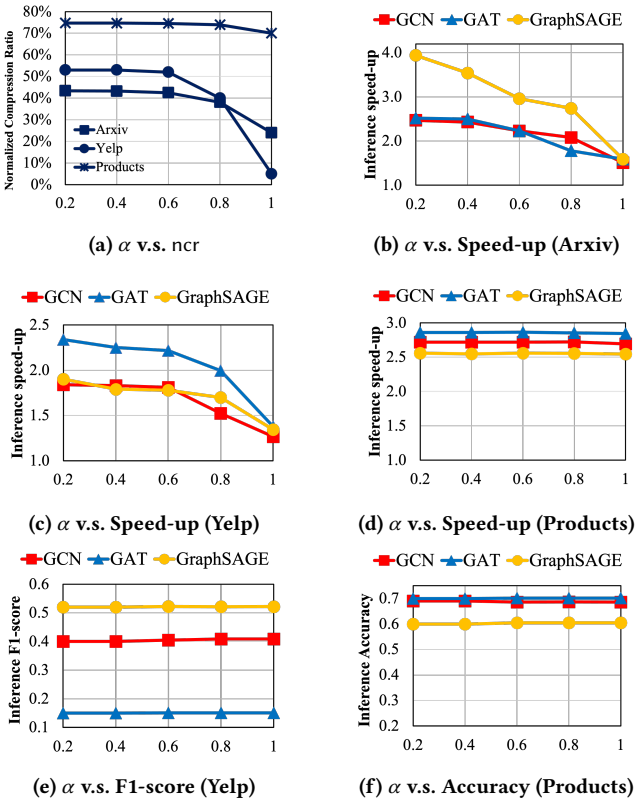
Varying Number of Layers. We varied the number of layers of GNNs from 2 to 4 over ***Arxiv*** and report its impact on inference speed-up (resp. accuracy) in Fig. 10(a) (resp. 10(b)). (1) $(0.5, 1)$-SPGC consistently outperforms all the baselines in both inference speed-up and accuracy, due to that it 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.

Varying α. Fixing $r = 1$, we varied α from 0.2 to 1, and report the results in Fig. 11. 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, as α increases, for all the three GNNs, 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), 11(c), and 11(d).

(3) As α increases, the F1-score (resp. accuracy) of the inference

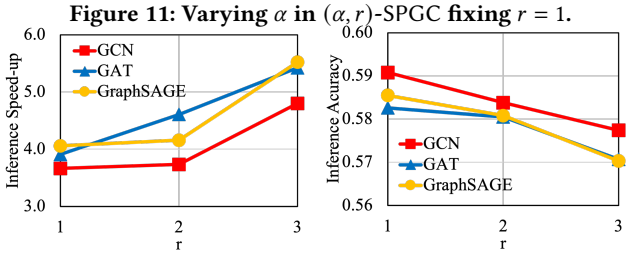


(a) α v.s. ncr (b) α v.s. Speed-up (Arxiv)

(c) α v.s. Speed-up (Yelp) (d) α v.s. Speed-up (Products)

(e) α v.s. F1-score (Yelp) (f) α v.s. Accuracy (Products)

Figure 11: Varying α in (α, r)-SPGC fixing $r = 1$.



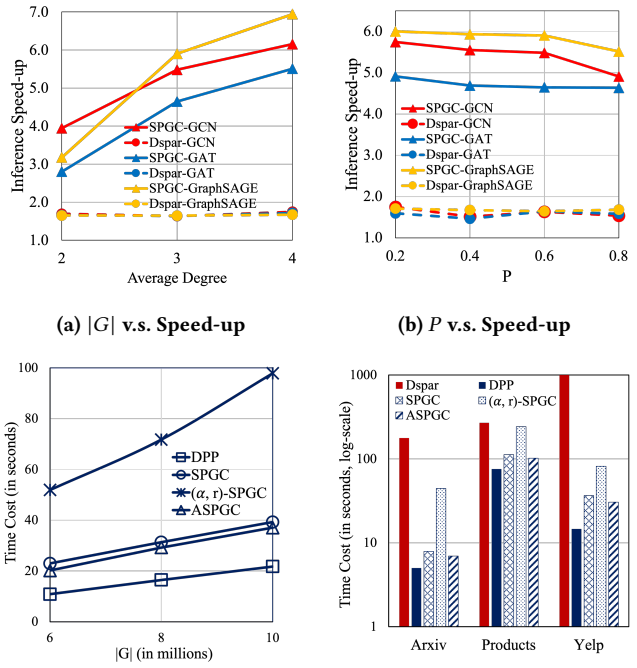
(a) r v.s. Speed-up (b) r v.s. Accuracy

Figure 12: Varying r in (α, r)-SPGC fixing $\alpha = 0.25$ (Arxiv).

results over ***Yelp*** (resp. **ogbn-products**) remains insensitive, as shown in Figs.11(e) (resp. 11(f)). 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 results while can effectively trade accuracy for 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. 12. (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

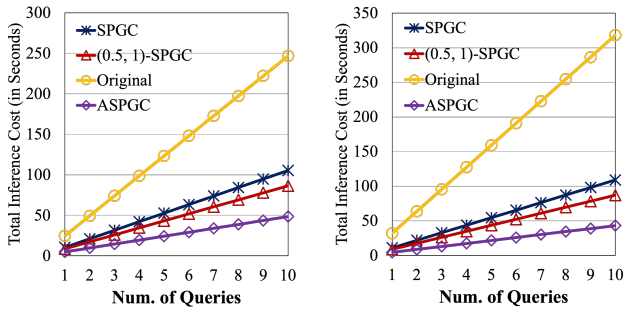
Figure 13: Scalability and Efficiency Test.

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

Fixing $P = 0.6$, we vary the average degree from 2 to 4. Fig. 13(a) shows that the inference speed-up across three GNNs. (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 benefit from larger $|E|$, which makes nodes more likely to be merged, resulting in a smaller G_c and greater speed-up. This is also consistent with Corollary 6 where the maximum inference speed up is bounded by $d \cdot \text{cr}$.

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. Above observation (1) tells us that SPGC may perform better when the graphs exhibit a "small-world" effect, which is a common phenomenon in the graphs from many fields including power grids, road maps, brain networks, and social networks [8, 49].

Impact of $|G|$. We increase the size of the simulated graph $|G|$ from $6M$ to $10M$. As shown in Fig. 13(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.



(a) Yelp **(b) *ogbn-products***

Figure 14: Comparison 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. 13(d). Note that the compression cost of SPGC and its variants include the time cost of DPP. We have the following discoveries. (1) SPGC and its variants are much faster than DSpar on all three real-world datasets. (2) Within the compression cost of SPGC and its variants, we observe the following pattern: T((α, r)-SPGC) > T(SPGC) > T(ASPGC). (α, r)-SPGC is the slowest. Compared to SPGC, ASPGC is faster because it induces a subgraph from V_A instead of compressing from G which leads to less compression cost. In contrast to SPGC and its variants, DSpar is much more costly. For example, for **Yelp**, the compression cost induced by DSpar exceeds three hours (out of range).

Inference cost. We conduct feasibility analysis to compare the inference cost of SPGC, (α, r)-SPGC, ASPGC to the original (inference on G, without compression). Fig. 14 reports the total inference costs induced by every method as the number of inference queries increase from 1 to 10 given fixed size of inference queries defined from V_T on **Yelp** and ***ogbn-products*** datasets. It tells us the followings. (1) As the number of queries increases, the total inference cost grows linearly regardless of the methods. (2) The total inference cost of ASPGC < $(0.5, 1)$-SPGC < SPGC for both datasets as shown in Fig. 14(a) and 14(b). This is expected since the size of compressed G_c from ASPGC, (α, r)-SPGC, and SPGC follow the same order.

Given the one-time compression cost $T(C)$ induced by ASPGC, (α, r)-SPGC, or SPGC, there exists an integer x such that when the **Num. of Queries** $\geq x$, the total time cost (*i.e.,* the one-time compression cost plus total inference cost on x queries) will be less than total time cost of conducting inference on the original G. For example, for V_T is 5% randomly selected from V in **Yelp**, $T(C)$ = 81.78s for $(0.5, 1)$-SPGC, 36.60s for SPGC, 30.52s for ASPGC, the average inference cost on compressed graph T_{MC} = 10.54s for SPGC, 8.64s for $(0.5, 1)$-SPGC, 4.85s for ASPGC, and T_{MG} = 24.68s, we can easily compute that $x = 6$ for $(0.5, 1)$-SPGC (resp. 3 for SPGC; 2 for ASPGC). This means that if the **Num. of Queries** ≥ 6 (resp. 3; 2), we will benefit from implementing $(0.5, 1)$-SPGC (resp. SPGC; ASPGC) to accelerate query loads inference.

Exp-4: Ablation Analysis. We next investigate how aggregation methods in GNNs, memoization structure $\mathcal{T}$, and neighbor recovery affect the inference accuracy and speed-up achieved by SPGC.

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

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

Table 7: Comparison of Inference Accuracy with Different Aggregation Methods in SPGC **(Arxiv).**

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

Table 8: Effeciveness of memoization and decompression.

The impact of memoization structure and decompression. We conduct ablation analysis using ***ogbn-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 (Table. 8). (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. However, such improvement comes at a significant cost of reduced inference speed-up.

8 CONCLUSION

We have proposed IPGC, a graph data compression scheme to generate compressed graphs that can be directly processed by GNN inference process to obtain original counterpart. We have introduced a sufficient condition, and introduced three practical specifications of IPGC, 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 IPGC-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] [n.d.]. Full version. https://anonymous.4open.science/r/SPGC/SPGC_full.pdf.

[2] Arman Ahmed, Sagnik Basumallik, Amir Gholami, Sajan K Sadanandan, Mohammad HN Namaki, Anurag K Srivastava, and Yinghui Wu. 2023. Spatio-Temporal Deep Graph Network for Event Detection, Localization and Classification in Cyber-Physical Electric Distribution System. *IEEE Transactions on Industrial Informatics* (2023).

[3] 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).

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

[5] 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.

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

[7] 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).

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

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

[10] 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).

[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] Wenfei Fan, Jianzhong Li, Xin Wang, and Yinghui Wu. 2012. Query preserving graph compression. In *SIGMOD*.

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

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

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

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

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

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

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

[24] 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] Loic Landrieu and Martin Simonovsky. 2018. Large-scale point cloud semantic segmentation with superpoint graphs. In *CVPR*.

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

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

[35] 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).

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[51] Bing Yu, Haoteng Yin, and Zhanxing Zhu. 2018. Spatio-temporal Graph Convolutional Networks: A Deep Learning Framework for Traffic Forecasting. In *IJCAI*.

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

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

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

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

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

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

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

Given the above analysis, Lemma 1 follows. □

Proof of Lemma 2. Given $\mathbb{M}$ and G, the binary relation R_M^L is an equivalence relation, *i.e.,* it is reflexive, symmetric, and transitive.

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

Proof of Theorem 3. Given a class of GNNs $\mathbb{M}^L$ and graph G, there exists a IPGC $(C, \mathcal{P})$ for $\mathbb{M}^L$ and G, if (1) there is a non-empty inference equivalent relation R_M^L *w.r.t.* $\mathbb{M}^L$ and G, (2) $C(G)$ correctly computes a quotient graph G_c induced by R_M^L, and (3) $|\mathcal{P}(G_c)| < |G|$.

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

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

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

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

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

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

We now show that $X_v^{k+1} = M_v(\Theta^k, \text{AGG}, N(v), M_v^k)$, $= M_{[v]}(\Theta^k, \text{AGG}, N([v]), M_{[v]}^k) = X_{[v]}^{k+1}$. To see this, $M_{[v]}$ only needs to "invoke" a fast look up function $\mathcal{P}$ that retrieves an edge weight (pre-stored during compression C; see "Notes") to adjust its direct aggregation over $N_{[v]}$ in G_c, as needed, without decompression. We illustrate below typical examples of the weighted update for major GNNs in Table 5 augmenting Table. 4; 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 $(\mathcal{C}, \mathcal{P})$ is an IPGC 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 4, 5 and 6, respectively, for IPGC specifications.

Proof of Theorem 4. 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 3, we perform an induction on the number of layers k of GNNs M.

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

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

Proof of Theorem 5. 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 $\mathcal{C}$ 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 1, 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 4. Hence there exists a unique, smallest quotient graph induced by R^{S*} up to graph isomorphism. □

Proof of Corollary 6. 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 4. A maximum speed up for inference cost can be computed as $\frac{O(LmdF^2+LnF^2)}{O(Lm'dF^2+Ln'F^2)} \leq \frac{md+n}{m'+n'} \leq d\frac{m+n}{m'+n'} = d \cdot \text{cr}$.

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

Proof of Theorem 7. 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 $(G'_c, G_c) \in \mathcal{G}_{c\mathcal{M}}$. □

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

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

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

Inference process with decompression. Following the inference algorithm in Sec. 4.4, the users can directly query on V_T from G_c compressed from (α, r)-SPGC without any decompression and benefit from the accelerated inference. However, this may result in dropped inference accuracy since inference equivalence is not directly preserved. A pair (v, v') in an (α, r)-relation $R^{(\alpha,r)}$ is no longer conform to embedding equivalence, thus an (α, r)-SPGC $(C, \_)$ alone is not an IPGC, *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 [9]). We non-trivially extend Re-Pair compression, a reference encoding method [14, 33] 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.

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 $[v]$.

Algorithm 3 Procedure DPP(G)

```
1:  for v ∈ V do
2:     compute rank(n);
3:  φ := max{rank(n)};
4:  node partition Par := {B_i : i = −∞, 0, ..., φ}
5:  collapse B_{−∞};
6:  for i = −∞, 0, ..., φ do
7:     induce R_i at rank i;
8:  for n ∈ V ∩ B_{−∞} do
9:     for i = 0, ..., φ do
10:       R_i := R_i \ {(v, v')|(v, n) ∈ E and (v', n) ∉ E};
11:       update B_i;
12: for i = 0, , ..., φ do
13:    D_i := {X ∈ Par : X ⊆ B_i};
14:    refine D_i with Paige-Tarjan;
15:    collapse X | ∀X ∈ D_i;
16:    for n ∈ V ∩ B_i do
17:       for j = i + 1, ..., φ do
18:          R_j := R_j \ {(v, v')|(v, n) ∈ E and (v', n) ∉ E};
19:          update B_j;
20: R := ⋃_{i∈{−∞,0,...,φ}} 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 $\mathsf{AL_c}$ and rules;

1: set $R^{(\alpha,r)} := \emptyset$; set $EC := \{V\}$; set $\mathcal{T}' := \emptyset$; set $AL := \emptyset$; set $\mathsf{AL_c}$ $:= \emptyset$; dictionary rules $:= \emptyset$; graph G_c, $G_r := \emptyset$;
2: $G_r := (V, E_r) \mid E_r := \{(u, v)\}, v \in V$ and $u \in N_r(v)$;
3: induce adjacency list AL from E_r;
4: $R^{(\alpha,r)} := \mathsf{DPP}(G_r)$;
5: $R^{(\alpha,r)} := R^{(\alpha,r)} \setminus \{(v, v') \mid \mathsf{xsim}(X^0_v, X^0_{v'}) < \alpha\}$;
6: $EC := V/R^{(\alpha,r)}$; /* induce partition EC from $R^{(\alpha,r)}$*/
7: $G_c, \mathcal{T} := \mathsf{CompressG}\ (\mathcal{T}, G_c, EC, G, M)$;
8: $\mathsf{AL_c}$, rules := Re − Pair(AL);
9: **return** $G_c, \mathcal{T}, EC, \mathsf{AL_c}$, rules;

Figure 16: Algorithm (α, r)-SPGC

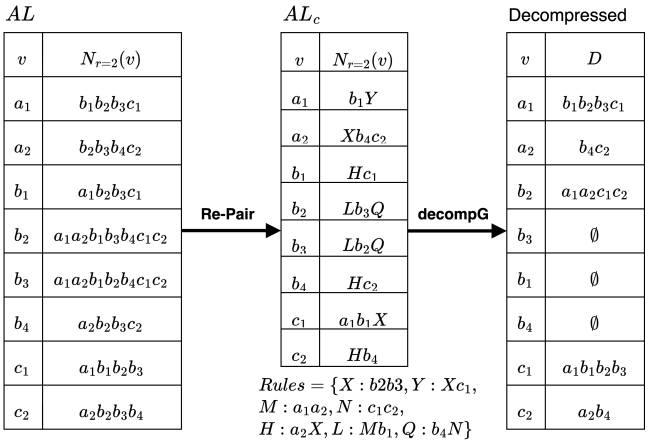
Example 8: We continue with Example. 6. Given G_{c5}, EC, $\mathsf{AL_c}$, and rules, decompG sorts $\mathsf{AL_c}$ in the descending order of node degrees while still keeping it in the order of equivalent class. decompG next decompresses $N_r([a])$ such that the nodes $b_1b_2b_3c_1$ are decompressed first by a_1 and then b_4c_2 are decompressed by a_2 as illustrated in the Fig. 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. □

Algorithm 5 : decompG

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

```
1: G_cd = G_c;
2: sort AL_c by node degrees in G_r;
3: for [v] ∈ G_c.V_c do
4:    while ∃ not visited v ∈ [v] do
5:       D = {u|u ∈ N_r(v) ∧ ¬ decompressed by v ∈ [v]};
6:       for ∀u ∈ D do
7:          V_cd.add(u);
8:          E_cd.add(u, [v]);
9:       mark v as visited;
10: return G_cd;
```

Figure 17: Algorithm decompG



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	$a_2b_2b_3b_4$

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

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).**

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

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

Lemma 11: *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.