Learning Graph Neural Networks using Exact Compression

Peer-reviewed author version

BOLLEN, Jeroen; STEEGMANS, Jasper; VAN DEN BUSSCHE, Jan &
VANSUMMEREN, Stijn (2023) Learning Graph Neural Networks using Exact Compression.

DOI: 10.1145/3594778.3594878
Handle: http://hdl.handle.net/1942/40622

Learning Graph Neural Networks using Exact Compression

Jeroen Bollen
jeroen.bollen@uhasselt.be
UHasselt, Data Science Institute
Belgium

Jasper Steegmans
jasper.steegmans@uhasselt.be
UHasselt, Data Science Institute
Belgium

Jan Van den Bussche
jan.vandenbussche@uhasselt.be
UHasselt, Data Science Institute
Belgium

Stijn Vansummeren
stijn.vansummeren@uhasselt.be
UHasselt, Data Science Institute
Belgium

Abstract

Graph Neural Networks (GNNs) are a form of deep learning that enable a wide range of machine learning applications on graph-structured data. The learning of GNNs, however, is known to pose challenges for memory-constrained devices such as GPUs. In this paper, we study *exact compression* as a way to reduce the memory requirements of learning GNNs on large graphs. In particular, we adopt a formal approach to compression and propose a methodology that transforms GNN learning problems into provably equivalent compressed GNN learning problems. In a preliminary experimental evaluation, we give insights into the compression ratios that can be obtained on real-world graphs and apply our methodology to an existing GNN benchmark.

CCS Concepts

• **Information systems → Graph-based database models**; • **Computing methodologies → Neural networks**.

Keywords

Graph neural networks, color refinement, compression

ACM Reference Format:
Jeroen Bollen, Jasper Steegmans, Jan Van den Bussche, and Stijn Vansummeren. 2023. Learning Graph Neural Networks using Exact Compression. In *Joint Workshop on Graph Data Management Experiences & Systems (GRADES) and Network Data Analytics (NDA) (GRADES & NDA '23), June 18, 2023, Seattle, WA, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3594778.3594878

1 Introduction

Graph Neural Networks (GNNs for short) are a form of deep learning architectures that enable a wide range of ML applications on graph-structured data, such as molecule classification, knowledge graph completion, and web-scale recommendations [3, 10, 12, 13, 26]. At their core, GNNs allow to embed graph nodes into vector space. Crucially, the obtained vectors can capture graph structure, which is essential for the ML applications already cited.

While GNNs are hence an attractive mechanism for ML on graphs, learning GNNs is known to be resource-demanding which limits their scalability [24, 29]. In particular, for large graphs it becomes difficult to encode all the required information into the limited memory of hardware accelerators like GPUs. For this reason, scalable methods for learning GNNs on large graphs are an active subject of research (e.g., [6, 7, 9, 13, 15, 18, 19, 21–23, 27, 28]). Broadly speaking, we can identify three different principles for obtaining scalability in the literature: (1) distributing computation across multiple machines or GPUs [9, 18, 19, 21, 23, 27, 28]; (2) learning on a sample of the input graph instead of the entire graph [6, 13, 15]; and (3) compression [7, 9, 17, 22]. Compression-based approaches limit the memory requirements of learning GNNs by reducing the input graph into a smaller graph and then learn on this smaller, reduced graph instead. In this paper, we are concerned with compression.

Compression methods are based on collapsing multiple input nodes into a single reduced node in the compressed graph. Methods vary, however, in how they collapse nodes. For example, Deng et al. [7] use spectral analysis for this purpose; Liang et al. [17] use variants of multi-level graph partitioning; and Generale et al. [9], who specifically consider knowledge graphs, use general heuristics (such as two nodes having equal set of attributes) or bisimulation. While these methods give intuitive reasons to argue that the structure of the obtained compressed graph should be similar to that of the original graph, no formal guarantee is ever given that learning on the compressed graph is in any way equivalent to learning on the original graph. Furthermore, the methods are usually devised and tested for a specific GNN architecture (such as Graph Convolutional Networks, GCN). It is therefore unclear how they fare on other GNN architectures. Inherently, these methods are hence heuristics. At best the compressed graphs that they generate *approximate* the original graph structure, and it is difficult to predict for which GNN architectures this approximation is good enough, and for which architectures it poses a problem.

Towards a more principled study of learning GNNs on compressed graphs, we propose to take a formal approach and study *exact compression* instead. We make the following contributions.

(1.) We formally define when two learning problems involving graph neural networks are equivalent. Based on this definition, our goal is to transform a given problem into a smaller, equivalent problem based on compression. (Section 2.)

GRADES & NDA '23, June 18, 2023, Seattle, WA, USA

ACM ISBN 979-8-4007-0201-3/23/06…$15.00
https://doi.org/10.1145/3594778.3594878

(2.) We develop a compression methodology that is guaranteed to always yield an equivalent learning problem and that is applicable to a wide class of GNN architectures known as *aggregate-combine GNNs* [2, 11, 12]. This class includes all Graph Convolutional Networks [12]. Our methodology is based on recent insights into the expressiveness of aggregate-combine GNNs [20, 25]. These results imply that if the local neighborhoods of two nodes v, w in input graph G are equal, then any GNN will treat v and w identically. We may intuitively exploit this property for compression: if v and w are treated identically there is no need for them both to be present during learning; having one of them suffices. We fully develop this intuition in Section 3, where we also consider a more relaxed notion of "local neighborhood" that is applicable only to specific kinds of aggregate-combine GNNs.

(3.) We empirically evaluate the effectiveness of our methodology in Section 4. In particular, we give insights into the compression ratios that can be obtained on real-world graphs. While we find that these ratios are diverse, from compressing extremely well to compressing only marginally, a preliminary experiment on an existing GNN benchmark shows positive impact on learning efficiency even with marginal compression.

We start with preliminaries in Section 2 and conclude in Section 5. Interested readers may find the proofs of formal statements in the extended version of this paper [4].

2 Preliminaries

Background. We denote by $\mathbb{R}$ the set of real numbers, by $\mathbb{N}$ the set of natural numbers, and by $\mathbb{N}_\infty$ the set $\mathbb{N} \cup \{\infty\}$ of natural numbers extended with infinity. We will use double curly braces $\{\!\{\ldots\}\!\}$ to denote multisets and multiset comprehension. Formally, we view a multiset over a domain of elements S as a function $M\colon S \to \mathbb{N}$ that associates a multiplicity $M(x)$ to each element $x \in S$. As such, in the multiset $M = \{\!\{a, a, b\}\!\}$, we have that $M(a) = 2$ and $M(b) = 1$. If $M(x) = 0$ then x is not present in M. We denote by $supp(M)$ the set of all elements present in M, $supp(M) := \{x \in S \mid M(x) > 0\}$. Note that if every element has multiplicity at most one, then M is a set. If M is a multiset and $c \in \mathbb{N}_\infty$ then we denote by $M|_{\leq c}$ the multiset obtained from M by restricting the multiplicity of all elements to be at most c, i.e., $M|_{\leq c}(x) = \min(M(x), c)$, for all elements x. Note in particular that $M|_{\leq +\infty} = M$ and that $M|_{\leq 1}$ converts M into a set.

Graphs. We work with directed node-colored multigraphs. Formally, our graphs are hence triples $G = (V, E, g)$ where V is a finite set of nodes; E is a *multiset* of edges over $V \times V$; and g is a function, called the *coloring* of G, that maps every node $v \in V$ to a *color* $g(v)$. (The term "color" is just an intuitive way to specify that g has some unspecified range.) If Y is the co-domain of g, i.e., g is of the form $g\colon V \to Y$ then we also call g a *Y-coloring* and say that G is a *Y-colored graph*, or simply a Y-graph. When $Y = \mathbb{R}^n$ we also call g an n-dimensional *feature map*. To ease notation we write $v \in G$ to indicate that $v \in V$. Furthermore, we write $G(v)$ instead of $g(v)$ to denote the color of v in G, and we write $G(v \to w)$ instead of $E(v \to w)$ to denote the multiplicity of edge $v \to w$ in G. When E is a set, i.e., when every edge has multiplicity at most one, then we also call G a *simple* graph. We write $\mathsf{in}_G(v)$ for the multiset $\{\!\{w \in G \mid w \to v \in E\}\!\}$ of all incoming neighbors of v. So, if the edge $w \to v$ has multiplicity 5 in E then w also has multiplicity 5 in $\mathsf{in}_G(v)$. We drop subscripts when the graph G is clear from the context. The *size* of a graph G is the number of nodes $|V|$ plus the number of simple edges $|supp(E)|$. This is a reasonable definition of the size of a multigraph, since for each edge it suffices to simply store its multiplicity as a number, and storing a number takes unit cost in the RAM model of computation.

Color transformers. If C is a function that maps X-colored graphs $G = (V, E, g)$ into Y-colored graphs $G' = (V', E', g')$ that leaves nodes and edges untouched and only changes the coloring, i.e., $V = V'$ and $E = E'$ then we call C a *coloring transformer*. In particular, if $X = \mathbb{R}^p$ and $Y = \mathbb{R}^q$ for some dimensions p and q then C is a *feature map transformer*.

Graph Neural Networks. Graph Neural Networks (GNNs) are a popular form of neural networks that enable deep learning on graphs. Many different forms of GNNs have been proposed in the literature. We refer the reader to the overview by Hamilton [12]. In this paper we focus on a standard form of GNNs that is known under the name of *aggregate-combine GNNs* [2], also called *message-passing GNNs*. These are defined as follows [8, 11].

A *GNN layer* of input dimension p and output dimension q is a pair $(\textsc{Agg}, \textsc{Comb})$ of functions where (1) $\textsc{Agg}$ is an *aggregation function* that maps finite multisets of vectors in $\mathbb{R}^p$ to vectors in $\mathbb{R}^h$ for some dimension h and (2) $\textsc{Comb}$ is a *combination function* $\textsc{Comb}\colon \mathbb{R}^p \times \mathbb{R}^h \to \mathbb{R}^q$. In practice, $\textsc{Agg}$ is usually taken to compute the arithmetic mean, sum, or maximum of the vectors in the multiset, while $\textsc{Comb}$ is computed by means of a feedforward neural network whose parameters can be learned.

A *GNN* is a sequence $\overline{L} = (L_1, \ldots, L_k)$ of GNN layers, where the output dimension of L_i equals the input dimension of L_{i+1}, for $1 \leq i < k$. The *input and output dimensions* of the GNN are the input dimension of L_1, and the output dimension of L_k respectively. In what follows, we write $\overline{L}\colon p, q$ to denote that p is the input dimension of $\overline{L}$ and q is the output dimension.

Semantically, GNN layers and GNNs are feature map transformers [8] In particular, when GNN layer $L = (\textsc{Agg}, \textsc{Comb})$ of input dimension p and output dimension q is executed on $\mathbb{R}^p$-colored graph $G = (V, E, g)$ it returns the $\mathbb{R}^q$-colored graph $G' = (V, E, g')$ with g' the q-dimensional feature map defined by

$$g'\colon v \mapsto \textsc{Comb}\big(g(v), \textsc{Agg}\,\{\!\{g(w) \mid w \in \mathsf{in}_G(v)\}\!\}\big).$$

As such, for each node v, $\overline{L}$ aggregates the (multiplicity-weighted) $\mathbb{R}^p$ colors of v's neighbors, and combines this with v's own color to compute the $\mathbb{R}^q$ output.

A GNN $\overline{L}\colon p, q$ simply composes the transformations defined by its layers: given $\mathbb{R}^p$-colored graph G it returns the $\mathbb{R}^q$-colored graph $(L_k \circ L_{k-1} \circ \cdots \circ L_1)(G)$.

Discussion. It is important to stress that in the literature GNNs are defined to operate on *simple graphs*, whereas we have generalized their semantics above to also work on *multigraphs*. We did so because, as we will see in Section 3, the result of compressing a simple graph for the purpose of learning naturally yields a multigraph.

Learning problems. GNNs are used for a wide range of supervised learning tasks on graphs. For example, for a node v, the $\mathbb{R}^q$-vector $\overline{L}(G)(v)$ computed for v by GNN $\overline{L}$ can be interpreted, after normalisation, as a probability distribution over q new labels (for

node classification), or as predicted values (for node regression). Similarly, an edge prediction for nodes v and w can be made based on the pair $(\overline{L}(G)(v), \overline{L}(G)(w))$. Finally, by aggregating $\overline{L}(G)(v)$ over all nodes $v \in G$, one obtains graph embeddings that can be used for graph classification, regression and clustering [12].

In this work, we focus on the tasks of node classification and regression. Our methodology is equally applicable to the other tasks, however.

In order to make precise what we mean by learning GNNs on compressed graphs for node classification, we propose the following formal definition.

Definition 2.1. A *learning problem* of input dimension p and output dimension q is a tuple $\mathcal{P} = (G, T, \text{Loss}, \mathcal{S})$ where

- G is the $\mathbb{R}^p$-colored graph on which we wish to learn;
- T is a subset of G's nodes, representing the training set;
- $\text{Loss}\colon T \times \mathbb{R}^q \to \mathbb{R}$ is a loss function that allows to quantify, for each node $v \in T$ the dissimilarity $\text{Loss}(v, c)$ of the $\mathbb{R}^q$-color c that is predicted for v by a GNN and the desired $\mathbb{R}^q$-color for v as specified in the training set;
- $\mathcal{S}$ is the *hypothesis space*, a (possibly infinite) collection of GNNs of input dimension p and output dimension q.

Given a learning problem $\mathcal{P}$, a learning algorithm produces a "learned" GNN in $\mathcal{S}$ by traversing the search space $\mathcal{S}$. For each currently considered GNN $\overline{L} \in \mathcal{S}$, the observed loss of $\overline{L}$ on G w.r.t. T is computed as

$$\text{Loss}(\overline{L}(G), T) := \sum_{v \in T} \text{Loss}\left(v, \overline{L}(G)(v)\right).$$

The learning algorithm aims to minimize this loss, but possibly returns an $\overline{L}$ for which this is only a local minimum.

In practice, $\mathcal{S}$ is usually a collection of GNNs with the same topology: they all have the same number of layers (with each layer d having the same input and output dimensions accross GNNs in $\mathcal{S}$) and are parametrized by the same number of learnable parameters. Each concrete parametrization constitutes a concrete GNN in $\mathcal{S}$ in our framework. Commonly, the learned GNN $\overline{L}$ is then found by means of gradient descent, which updates the learnable parameters of the GNNs in $\mathcal{S}$ to minimize the observed error.

No matter which concrete learning algorithm is used to solve a learning problem, the intent is that the returned $\overline{L}$ generalizes well: it makes predictions on G for the nodes not in T, and can also be applied to other, new $\mathbb{R}^p$-colored graphs to predict $\mathbb{R}^q$-vectors for each node.

Our research question in this paper is the following.

> *Given a GNN learning problem $\mathcal{P} = (G, T, \text{Loss}, \mathcal{S})$, is it possible to transform this into a new problem $\mathcal{P}' = (G', T', \text{Loss}', \mathcal{S})$ that is obtained by* compressing *G, T, and Loss into a smaller graph G', training set T', and loss function Loss$'$ such that instead of learning a GNN on $\mathcal{P}$ we could equivalently learn a GNN on $\mathcal{P}'$ instead?*

Here "equivalently" means that ideally, no matter which learning algorithm is used, we would like the learned GNN to be identical in both cases. Of course, this is not possible in practice because the learning process is itself non-deterministic, e.g., because the learning algorithm makes random starts; because of stochasticity in stochastic gradient descent; or because of non-deterministic dropout that is applied between layers. Nevertheless, we expect the GNN obtained by learning on the compressed problem would perform "as good" as the GNN obtained by the learning on the uncompressed problem, in the sense that it generalizes to unseen nodes and unseen colored graphs equally well.

To ensure that we may hope any learning algorithm to perform equally well on $\mathcal{P}'$ as on $\mathcal{P}$, we formally define:

Definition 2.2. Two learning problems $\mathcal{P}$ and $\mathcal{P}'$ are *equivalent*, written $\mathcal{P} \equiv \mathcal{P}'$, if they share the same hypothesis space of GNNs $\mathcal{S}$ and, for every $\overline{L} \in \mathcal{S}$ we have $\text{Loss}(\overline{L}(G), T) = \text{Loss}'(\overline{L}(G'), T')$.

In other words, when traversing the hypothesis space for a GNN to return, no learning algorithm can distinguish between $\mathcal{P}$ and $\mathcal{P}'$. All other things being equal, if the learning algorithm then returns a GNN $\overline{L}$ when run on $\mathcal{P}$, it will return $\overline{L}$ on $\mathcal{P}'$ with the same probability.

Note that, while the hypothesis space $\mathcal{S}$ remains unchanged in this definition, it is possible (and, as we will see, actually required) to adapt the loss function Loss into a modified loss function Loss$'$ during compression.

The benefit of a positive answer to our research question, if compression is efficient, is computational efficiency: learning on smaller graphs is faster than learning on larger graphs and requires less memory.

3 Methodology

To compress one learning problem into an equivalent, hopefully smaller, problem we will exploit recent insights into the expressiveness of GNNs [2, 20, 25]. In particular, it is known that if the local neighborhoods of two nodes v, w in input graph G are equal, then any GNN will treat v and w identically. In particular, it will assign the same output colors to v and w. We may intuitively exploit this property for compression: since v and w are treated identically there is no need for them both to be present during learning; having one of them suffices. So, we could compress by removing nodes that are redundant in this sense. We must take care, however, that by removing one, we do not change the structure (and hence, possibly, the predicted color) of the remaining node. Also, of course, we need to make sure that by removing nodes we do not lose training information. I.e., if T specifies a training color for v but not w then if we decide to remove v, we somehow need to "fix" T, as well as the loss function.

This section is devoted to developing this intuitive idea. In Section 3.1 we first study under which conditions GNNs treat nodes identically. Next, in Section 3.2 we develop compression of colored graphs based on collapsing identically-treated nodes, allowing to remove redundant nodes while retaining the structure of the remaining nodes. Finally, in Section 3.3, we discuss compression of the training set and loss function. Together, these three ingredients allow us to compress a learning problem into an equivalent problem, cf. Definition 2.2.

We close this section by proposing an alternative definition of compression that works only for a limited class of learning problems. It is nevertheless interesting as it may allow better compression, as we will show in Section 4.

3.1 Indistinguishability

The following definition formalizes when two nodes, not necessarily in the same graph, are treated identically by a class of GNNs.

Definition 3.1. Let $\mathcal{S}$ be a class of GNNs, let G and H be two $\mathbb{R}^p$-colored graphs for some p, and let $v \in G, w \in H$ be two nodes in these graphs. We say that (G, v) is *$\mathcal{S}$-indistinguishable* from (H, w), denoted $(G, v) \sim_{\mathcal{S}} (H, w)$, if for every GNN $\overline{L} \in \mathcal{S}$ of input dimension p it holds that $\overline{L}(G)(v) = \overline{L}(H)(w)$.

In other words, two nodes are indistinguishable by a class of GNNs $\mathcal{S}$ if no *GNN* in $\mathcal{S}$ can ever assign a different output color to these nodes, when started on G respectively H. We call (G, v) and (H, w) *$\mathcal{S}$-distinguishable* otherwise.

For the purpose of compression, we are in search of sets of nodes in the input graph G that are pairwise $\mathcal{S}$-indistinguishable, with $\mathcal{S}$ the hypothesis space of the input learning problem. It is these nodes that we can potentially collapse in the input learning problem. Formally, let $[G, v]_{\mathcal{S}}$ denote the set of all nodes in G that are S-indistinguishable from v,

$$[G, v]_{\mathcal{S}} := \{w \in G \mid (G, v) \sim_{\mathcal{S}} (G, w)\}.$$

We aim to calculate $[G, v]_{\mathcal{S}}$ and subsequently compress G by removing all but one node in $[G, v]_{\mathcal{S}}$ from G.

Color refinement. To calculate $[G, v]_{\mathcal{S}}$, we build on the work of Morris et al. [20] and Xu et al. [25]. They proved independently that a GNN can distinguish two nodes if an only if the so-called *color refinement* algorithm assigns different colors to these nodes. Color refinement is equivalent to the one-dimensional Weisfeiler-Leman (WL) algorithm [11], and works as follows.

Definition 3.2. The (one-step) *color-refinement* of colored graph $G = (V, E, g)$, denoted $\mathsf{cr}(G)$, is the colored graph $G' = (V, E, g')$ where g' maps every node $v \in G$ to a pair, consisting of v's original color and the multiset of colors of its incoming neighbors:

$$g' \colon v \mapsto \big(G(v), \{\!\{G(w) \mid w \in \mathsf{in}_G(v)\}\!\}\big).$$

As such, we can think of $\mathsf{cr}(G)(v)$ as representing the immediate neighborhood of v (including v), for any node v.

We denote by $\mathsf{cr}^d(G)$ the result of applying d color refinement steps on G, so $\mathsf{cr}^0(G) = G$ and $\mathsf{cr}^{d+1}(G) = \mathsf{cr}(\mathsf{cr}^d(G))$. Using this notation, we can think of $\mathsf{cr}^d(G)(v)$ as representing the local neighborhood of v "up to radius d".

To illustrate, Figure 1 shows a colored graph G and two steps of color refinement.

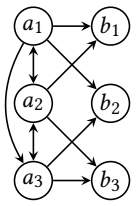
The following property was observed by Morris et al. [20] and Xu et al [25] for GNNs operating on *simple* graphs. We here extend it to multigraphs.

Proposition 3.3. *Let $\overline{L}$ be a GNN composed of $d \in \mathbb{N}$ layers, $d \geq 1$. If $\mathsf{cr}^d(G)(v) = \mathsf{cr}^d(H)(w)$ then $\overline{L}(G)(v) = \overline{L}(H)(w)$. As a consequence, if $\mathcal{S}$ is a hypothesis space consisting of GNNs of at most d layers and $\mathsf{cr}^d(G)(v) = \mathsf{cr}^d(H)(w)$ then $(G, v) \sim_{\mathcal{S}} (H, w)$.*

In other words, d-layer GNNs cannot distinguish nodes that are assigned the same color by d steps of color refinement.

Let $[G, v]_d$ denote the set of all nodes in G that receive the same color as v after d steps of color refinement,

$$[G, v]_d := \{w \in G \mid \mathsf{cr}^d(G)(v) = \mathsf{cr}^d(G)(w)\}.$$



	cr^0	cr^1	cr^2
a_1	a	① = $(a, \{\!\{a\}\!\})$	$(①, \{\!\{②\}\!\})$
a_2	a	② = $(a, \{\!\{a, a\}\!\})$	$(②, \{\!\{①, ②\}\!\})$
a_3	a	② = $(a, \{\!\{a, a\}\!\})$	$(②, \{\!\{①, ②\}\!\})$
b_1	b	③ = $(b, \{\!\{a, a\}\!\})$	$(③, \{\!\{①, ②\}\!\})$
b_2	b	③ = $(b, \{\!\{a, a\}\!\})$	$(③, \{\!\{①, ②\}\!\})$
b_3	b	③ = $(b, \{\!\{a, a\}\!\})$	$(③, \{\!\{②, ②\}\!\})$

Figure 1: Example of color refinement. Nodes a_1, a_2, a_3 have the same color a; nodes b_1, b_2, b_3 have the same color b. All edges have multiplicity 1.

Then it follows from Proposition 3.3 that $[G, v]_d$ is a *refinement* of $[G, v]_{\mathcal{S}}$ in the sense that $[G, v]_d \subseteq [G, v]_{\mathcal{S}}$, for all $v \in G$. Morris et al. [20] and Xu et al. [25] have also shown that for every graph G and every depth d there exists a GNN $\overline{L}$ of d layers such that $[G, v]_d = [G, v]_{\{\overline{L}\}}$, for every node $v \in G$. Consequently, if, in addition to containing only GNNs with at most d layers, $\mathcal{S}$ includes *all possible* d-layer GNNs, then $[G, v]_d = [G, v]_{\mathcal{S}}$ coincide, for all $v \in G$. Hence, for such $\mathcal{S}$ we may calculate $[G, v]_{\mathcal{S}}$ by calculating $[G, v]_d$ instead. When $\mathcal{S}$ does not include all d-layer GNNs we simply use $[G, v]_d$ as a proxy for $[G, v]_{\mathcal{S}}$. This is certainly safe: since $[G, v]_d \subseteq [G, v]_{\mathcal{S}}$ no GNN in $\mathcal{S}$ will be able to distinguish the nodes in $[G, v]_d$ and we may hence collapse nodes in $[G, v]_d$ for the purpose of compression. In this case, however, we risk that $[G, v]_d$ contains too few nodes compared to $[G, v]_{\mathcal{S}}$, and therefore may not provide enough opportunity for compression. We will return to this issue in Section 3.4.

What happens if there is no bound on the number of layers of GNNs in $\mathcal{S}$? In that case we can still use color refinement to compute $[G, v]_{\mathcal{S}}$ as follows. It is known that after a finite number of color refinements steps we reach a value d such that for all nodes $v \in G$ we have $[G, v]_d = [G, v]_{d+1}$. The smallest value d for which this holds is called the *stable coloring number* of G, and we denote the colored graph obtained by this value of d by $\mathsf{cr}^\infty(G)$ in what follows. Similarly we denote the equivalence classes at this value of d by $[G, v]_\infty$. From Proposition 3.3 it readily follows:

Corollary 3.4. *For any class $\mathcal{S}$ of GNNs, if $\mathsf{cr}^\infty(G)(v) = \mathsf{cr}^\infty(H)(w)$ then $(G, v) \sim_{\mathcal{S}} (H, w)$.*

We note that it is very efficient to compute the set $\{[G, v]_d \mid v \in G\}$ of all color refinement classes: this can be done in time $O((n + m) \log n)$ with n the number of vertices and m the number of edges of the input graph [5].

Example 3.5. To illustrate, consider the colored graph from Figure 1, as well as the color refinement steps illustrated there. (Recall that nodes a_1, a_2, a_3 share the same color, as do b_1, b_2, b_3.) Then after one step of color refinement we have

$$[G, a_1]_1 = \{a_1\}$$
$$[G, a_2]_1 = [G, a_3]_1 = \{a_2, a_3\}$$
$$[G, b_1]_1 = [G, b_2]_1 = [G, b_3]_1 = \{b_1, b_2, b_3\},$$

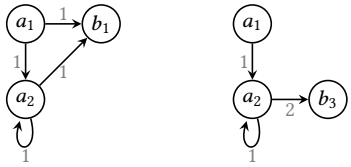


Figure 2: Reduction of the graph of Figure 1 by the 1-substitutions ρ_1 and ρ_2 from Example 3.7.

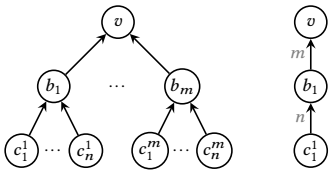


Figure 3: Reduction of a tree-shaped simple graph (left) into a small multigraph (right).

while after two steps of color refinement we obtain the following color refinement classes:

$$[G,a_1]_2 = \{a_1\} \qquad [G,b_1]_2 = [G,b_2]_2 = \{b_1,b_2\}$$
$$[G,a_2]_2 = [G,a_3]_2 = \{a_2,a_3\} \qquad [G,b_3]_2 = \{b_3\}.$$

We invite the reader to check that for every node v in this graph, $[G,v]_3 = [G,v]_2$. As such, the stable coloring is obtained when $d = 2$ and $[G,v]_2 = [G,v]_\infty$.

3.2 Graph reduction

Having established a way to compute redundant nodes, we now turn our attention to compression. Assume that we have already computed the color refinement classes $\{[G,v]_d \mid v \in G\}$ for $d \in \mathbb{N}_\infty$. For each $v \in G$, we wish to "collapse" all nodes in $[G,v]_d$ into a single node. To that end, define a *d-substitution* on a graph G to be a function that maps each color refinement class in $\{[G,v]_d \mid v \in G\}$ to a node $\rho([G,v]_d) \in [G,v]_d$. Intuitively, $\rho([G,v]_d)$ is the node that we wish to keep; all other nodes in $[G,v]_d$ will be removed. In what follows we extend ρ to also operate on nodes in G by setting $\rho(v) = \rho([G,v]_d)$.

Definition 3.6. The *reduction of graph G by d-substitution ρ on G* is the graph $H = (V,E,h)$ where

- $V = \{\rho(v) \mid v \in G\}$
- For all $v,w \in V$ we have
$$E(v \to w) = \sum_{v' \in [G,v]_d} G(v' \to w)$$
In particular, if there is no edge into w in G, there will be no edge into w in H, as this sum is then zero.
- nodes retain colors: for each node $v \in H$ we have $h(v) = G(v)$.

In what follows, we denote by G/ρ the reduction of G by ρ. A graph obtained by reducing G according to some d-substitution ρ is called a *d-reduct of G*.

Example 3.7. Reconsider the colored graph G of Figure 1. Let ρ_1 and ρ_2 be the following $d = 1$-substitutions:

$$\rho_1\colon \{a_1\} \mapsto a_1 \qquad \{a_2,a_3\} \mapsto a_2 \qquad \{b_1,b_2,b_3\} \mapsto b_1$$
$$\rho_2\colon \{a_1\} \mapsto a_1 \qquad \{a_2,a_3\} \mapsto a_2 \qquad \{b_1,b_2,b_3\} \mapsto b_3$$

Then G/ρ_1 and G/ρ_2 are illustrated in Figure 2.

The following proposition is an essential property of our compression methodology.

Proposition 3.8. *For every graph G, every d-substitution ρ of G with $d \in \mathbb{N}_\infty$, and every node $v \in G$ we have $\mathrm{cr}^d(G)(v) = \mathrm{cr}^d(G/\rho)(\rho(v))$.*

It hence follows from Proposition 3.3 and Corollary 3.4 that if $\mathcal{S}$ consists of GNNs of at most $d \in \mathbb{N}_\infty$ layers, then $(G,v) \sim_{\mathcal{S}} (G/\rho, \rho(v))$.

Discussion Example 3.7 shows that the choice of d-substitution determines the reduct G/ρ that we obtain. In particular, the example shows that distinct substitutions can yield distinct, *non-isomorphic* reducts. This behavior is unavoidable, unless d is the stable coloring number of G. Indeed, we are able to show:

Proposition 3.9. *There is a single d-reduct of a graph G up to isomorphism if and only if d is greater than or equal to the stable coloring number of G.*

A direct consequence of having different non-isomorphic reducts when d is less than the coloring number is that some of these reducts may be smaller than others. In Example 3.7, G/ρ_1 has 3 nodes and 4 edges while G/ρ_2 has 3 nodes and only 3 edges. We may always obtain a d-reduct of minimal size as follows. For $d \in \mathbb{N}_\infty$, define the *d-incidence* of a node $w \in G$, denoted $\mathsf{incidence}_G^d(w)$, to be the number of color refinement classes in $\{[G,v]_d \mid v \in G\}$ that contain an incoming neighbor of w. That is, the d-incidence of w is the number of distinct classes in $\{[G,u]_d \mid u \in \mathsf{in}_G(w)\}$. The following proposition shows that we obtain a d-reduct of minimal size by by choosing a d-substitution that maps color refinement classes to nodes of minimal d-incidence.

Proposition 3.10. *Let G be a graph, let $d \in \mathbb{N}_\infty$ and let ρ be a d-substitution such that*

$$\mathsf{incidence}_G^d(\rho(v)) = \min_{v' \in [G,v]_d} \mathsf{incidence}_G^d(v'),$$

for every node $v \in G$. Then the size of G/ρ is minimal among all d-reducts of G.

When we report the size of d-reducts in our experiments (Section 4), we always report the minimal size among all d-reducts.

Discussion Example 3.7 illustrates that, depending on the substitution used, the result of reducing a simple graph may be a multigraph. In the literature, however, GNNs and color refinement are normally defined to operate on simple graphs. One may therefore wonder whether it is possible to define a different notion of reduction that always yields a simple graph when executed on simple graphs, instead of a multigraph as we propose here. To answer this question, consider the tree-shaped graph G in Figure 3(left), whose root v has m b-colored children, each having n c-labeled children. It is straightforward to see that when $d \geq 2$, any simple graph H that has a node w such that $\mathrm{cr}^d(G)(v) = \mathrm{cr}^d(H)(w)$ must be such that w has m b-colored children in H, each having n c-labeled children. As such, because H is simple, it must be of size at least as large

as G. Instead, by moving to multigraphs we are able to compress this "regular" structure in G by only three nodes, as show in Figure 3(right). This illustrates that for the purpose of compression we naturally need to move to multigraphs.

3.3 Problem compression

We next combine the insights from Sections 3.1 and 3.2 into a method for compressing learning problems. Let $\mathcal{P} = (G, T, \text{Loss}, \mathcal{S})$ be a learning problem. Let ρ be a d-substitution. We then define the reductions T/ρ and Loss/ρ of T and Loss by ρ as follows:

$$T/\rho = \{\rho(v) \mid v \in T\}$$

$$\text{Loss}/\rho(v, c) = \sum_{w \in T \cap [G,v]_d} \text{Loss}(w, c)$$

We denote by $\mathcal{P}/\rho$ the learning problem $(G/\rho, T/\rho, \text{Loss}/\rho, \mathcal{S})$.

Theorem 3.11. *Let $\mathcal{P}$ be a learning problem with hypothesis space $\mathcal{S}$ and assume that $d \in \mathbb{N}_\infty$ is such that every GNN in $\mathcal{S}$ has at most d layers. (In particular, $d = \infty$ if there is no bound on the number of layers in $\mathcal{S}$.) Then $\mathcal{P} \equiv \mathcal{P}/\rho$ for every d-substitution ρ.*

3.4 Graded Color Refinement

So far, we have focused on compression based on the *depth* of the GNNs present in the hypothesis space, where the depth of a GNN equals its number of layers and the depth of a hypothesis space $\mathcal{S}$ is the maximum depth of any of its GNNs, or ∞ if this maximum is unbounded. In particular, the notion of d-reduct hinges on this parameter d through the calculated color refinement classes $[G, v]_d$.

As we have already observed in Section 3.1, however, when $\mathcal{S}$ does not contain all d-layer GNNs the color refinement classes $[G, v]_d$ that we base compression on may contain too few nodes compared to $[G, v]_{\mathcal{S}}$ and may therefore not provide enough opportunity for compression.

In such cases, it may be beneficial to move to more fine-grained notions of color refinement that better capture $[G, v]_{\mathcal{S}}$. In this section we propose one such fine-grained notion, which takes into account the "width" of $\mathcal{S}$. We say that *GNN $\overline{L}$ has width $c \in \mathbb{N}_\infty$* if for every layer in $\overline{L}$ the aggregation function Agg is such that $\text{Agg}(M) = \text{Agg}(M|_{\leq c})$ for every multiset M. In other words: Agg can only "count" up to c copies of a neighbor's color. When $c = \infty$ there is no limit on the count. The width of $\mathcal{S}$ is then the maximum width of any of its GNNs, or ∞ if this maximum is unbounded. Hypothesis spaces of bounded width clearly do not contain *all* d-layer GNNs, for any depth d.

While we know of no practical GNN architectures that explicitly limit the width, there are influential learning *algorithms* that implicitly limit the width. For example, to speed up learning, GraphSAGE [13] can be parametrized by a hyperparameter c. When $c < \infty$ GraphSAGE does not consider all of a node's neighbors in each layer, but only a random sample of at most c neighbors of that node. Such sampling effectively limits the width of $\mathcal{S}$.

We next define a variant of color refinement, called *graded* color refinement, that takes width into account. It may lead to larger color refinement classes, and therefore potentially also to better compression.

Definition 3.12. Let $c \in \mathbb{N}_\infty$. The (one-step) *c-graded color refinement* of colored graph $G = (V, E, g)$, denoted $\text{cr}_c(G)$, is the colored graph $G' = (V, E, g')$ with

$$g' : v \mapsto \big(G(v), \{\!\{G(w) \mid w \in \text{in}_G(v)\}\!\}|_{\leq c}\big).$$

Note that cr_∞ equals normal, non-graded, color refinement. We also remark that cr_c with $c = 1$ corresponds to standard *bisimulation* on graphs [1]. We denote by $\text{cr}_c^d(G)$ the result of applying d refinement steps of c-graded color refinement, so $\text{cr}_c^0(G) = G$ and $\text{cr}_c^{d+1} = \text{cr}(\text{cr}_c^d(G))$. We denote by $[G, v]_{c,d}$ the set of all nodes in G that receive the same color after d steps of c-graded color refinement. The concept of (c, d)-substitution is then defined analogously to d-substitution, as mappings from the $[G, v]_{c,d}$ color refinement classes to nodes in these classes. The reduction of a graph G by a (c, d) substitution is similar to reductions by d-substitution except that the edge multiplicity $E(v \to w)$ for v, w in the reduction is now limited to c, i.e.,

$$E(v \to w) = \sum_{v' \in [G,v]_{c,d}} G(v' \to w)|_{\leq c}.$$

With these definitions, we can show that for any $\mathcal{P}$ with hypothesis space $\mathcal{S}$ of width c and depth d we have $\mathcal{P} \equiv \mathcal{P}/\rho$ for any (c, d)-reduction ρ. The full development is omitted due to lack of space.

We note that $[G, v]_d \subseteq [G, v]_{c,d}$, always. Graded color refinement hence potentially leads to better compression, but only applies to problems with hypothesis spaces of width c. We will empirically contrast the compression ratio obtained by (c, d)-reducts to those obtained by d-reducts (i.e., where $c = \infty$) in Section 4. There, we also study the effect of c on learning accuracy for problems whose width is not bounded.

4 Evaluation

In this section, we empirically evaluate the compression methodology described in Section 3. We first give insights into the compression ratios that can be obtained on real-world graphs in Section 4.1. Subsequently, we evaluate the learning on compressed graphs versus learning on the original graphs.

4.1 Compression

We consider the real-world graphs listed in Table 1. The ogbn-* graphs are from the Open Graph Benchmark (OGB), a collection of realistic, large-scale, and diverse benchmark datasets for machine learning on graphs [14], where they belong to the OGB node property prediction benchmark (OGBN). Specifically, ogbn-arxiv is a network of academic papers, where edge $x \to y$ indicates that x cites y. Graph ogbn-arxiv-inv is the inverted version of ogbn-arxiv; it is obtained by reversing edges, so that edge $x \leftarrow y$ indicates that y was cited by x. Graph ogbn-arxiv-undirected is the undirected version of ogbn-arxiv; it is obtained by adding inverse edges to ogbn-arxiv. Next, there is ogbn-products, an undirected graph representing an Amazon product co-purchasing network. The other datasets are from the Stanford Large Network Dataset Collection (SNAP) [16]. Here, snap-roadnet-ca, snap-roadnet-pa, and snap-roadnet-tx are undirected graphs representing road networks, and snap-soc-pokec is a directed graph containing an online social network.

Table 1: Datasets

Graph	Type	#Nodes	#Edges
ogbn-arxiv	directed	169.343	1.166.243
ogbn-arxiv-inv	directed	169.343	1.166.243
ogbn-arxiv-undirected	undirected	169.343	2.315.598
ogbn-products	undirected	2.449.029	61.859.140
snap-roadnet-ca	undirected	1.965.206	5.533.214
snap-roadnet-pa	undirected	1.088.092	3.083.796
snap-roadnet-tx	undirected	1.379.917	3.843.320
snap-soc-pokec	directed	1.632.803	30.622.564

Table 2: Compression at $d = 3$.

Graph	Nodes (%)	Edges (%)
ogbn-arxiv	32.9	56.3
ogbn-arxiv-inv	65.9	92.7
ogbn-arxiv-undirected	93.3	98.9
ogbn-products	88.5	98.6
snap-roadnet-ca	4.4	5.9
snap-roadnet-pa	6.1	7.9
snap-roadnet-tx	3.9	5.2
snap-soc-pokec	87.9	99.1

The input colors used for learning on these graphs typically depend on the application. To get an understanding of the maximum amount of compression that we can obtain independent of the target application, we assign a shared single color c to each node, in all graphs. As such, the color refinement classes $[G, v]_d$ that we obtain are maximal, in the sense that for any other colorored graph G' whose topology equals G we will have $[G', v]_d \subseteq [G, v]_d$. In this setting, we hence reach maximal compression.

We consider three versions of ogbn-arxiv to get an indication of how edge directionality impacts compression. Recall that GNN layers propagate color information following the direction of edges. Hence, because in ogbn-arxiv an edge $x \to y$ indicates that x cites y, color information flows from citing papers to cited papers. In ogbn-arxiv-inv, by contrast, it flows from cited papers to citing papers while in ogbn-arxiv-undirected it flows in both directions. The direction in which the information can flow impacts the number of color refinement classes that we obtain, and hence the compression, as we will see.

Ungraded compression Figures 4(a) and 4(b) shows the fraction of nodes and edges remaining in d-reducts of these graphs, plotted as a function of the number of color refinement rounds d. (Consistent with our definition of the size of a multigraph, the number of edges plotted is the number of unique edges, i.e., ignoring edge multiplicities.) We see that in terms of the number of nodes, compression is effective for $d \leq 2$, obtaining compression ratios of $0.03\% - 0.05\%$ on the road network datasets, to 77% on ogbn-products. For $d = 3$, compression becomes ineffective for ogbn-arxiv-undirected, ogbn-products and snap-soc-pokec as these retain 88% of their nodes or more. Compression on the other datasets is satsifactory for $d = 3$, as shown in Table 2. From $d > 3$ onwards, node compression becomes ineffective for most datasets, with all datasets except ogbn-arxiv retaining at least 86% of their nodes when $d \geq 5$. By contrast, ogbn-arxiv stabilizes at $d = 4$, retaining only 36% of its nodes.

In terms of the number of edges, we see that ogbn-arxiv-inv, ogbn-arxiv-undirected, ogbn-products, and snap-soc-pokec retain almost 100% of their edges from $d \geq 2$ onwards, while ogbn-arxiv retains only 56% of its edges when $d = 3$ and the road networks compress to $5\% - 8\%$ of the edges when $d = 3$.

While we may hence conclude that in general compression becomes ineffective for deeper layer numbers, $d > 4$, we note that in practice most GNN topologies use only $d = 3$ layers. For such GNNs, compression on ogbn-arxiv and the road networks is promising.

One may wonder why the inverted and undirected variants of ogbn-arxiv differ so much in terms of compression. The short answer is that that, their graph topology is completely different. As such, the local neighborhood information that cr^d calculates is also completely different, yielding different reductions. For example, a manual inspection of ogbn-arxiv reveals a number of highly cited papers that have no outgoing edges. While these papers are the "sink nodes" in ogbn-arxiv, they are "source nodes" in ogbn-arxiv-inv. Quite quickly we may then distinguish nodes in ogbn-arxiv-inv based solely on the number of highly cited papers that they cite. This behavior does not occur in ogbn-arxiv, because the highly cited papers are outgoing neighbors, which cr^d ignores.

Graded compression We next evaluate the effect of moving to compression based on c-graded color refinement. Figure 4(c) and (d) shows the fraction of nodes and edges remaining in $(c, 3)$-reducts of our graphs, relative to the number of nodes and edges in the corresponding $d = 3$-reduct, plotted as a function of c. In terms of the number of nodes we see that, as expected, moving to graded compression yields better compression than non-graded compression. In particular, for $c = 1$ all datasets retain $< 1\%$ of their nodes while for $c = 3$, compression is 27% or less for ogbn-arxiv, ogbn-arxiv-undirected, and the road networks. In the latter setting, ogbn-products is at 75% of nodes and snap-soc-pokec at 55%. Compression hence becomes less effective as c increases.

In terms of the number of edges, compression is most effective when $c = 1$ or $c = 2$ in which case it significantly improves over ungraded compression. From $c \geq 3$ onwards, graded compression becomes only marginally better than ungraded compression.

We conclude that graded compression has the potential to yield significantly smaller graphs than ungraded compression, but only for small gradedness values, $c = 1$ (i.e., bisimulation) or $c = 2$.

Conclusion Overall, we see that real-world graphs have diverse non-graded compression ratios: road networks compress extremely well (to 4% of nodes and 5% of edges when $d = 3$); while other graphs such as ogbn-arxiv-inv compress reasonably in terms of number of nodes (65%) but only marginally in terms of edges (93% or more); and graphs such as ogbn-products compress only marginally in both (retaining 90% or more of nodes and edges). This diversity is to be expected: our exact compression methodology is based on exploiting redundancy in local neighborhoods of nodes. For graphs where few nodes have equal local neighborhoods, we cannot expect reduction in size. Moving to graded compression for those graphs improves reduction in size, but will yield approximate compression unless the learning problem hypothesis space has small width.

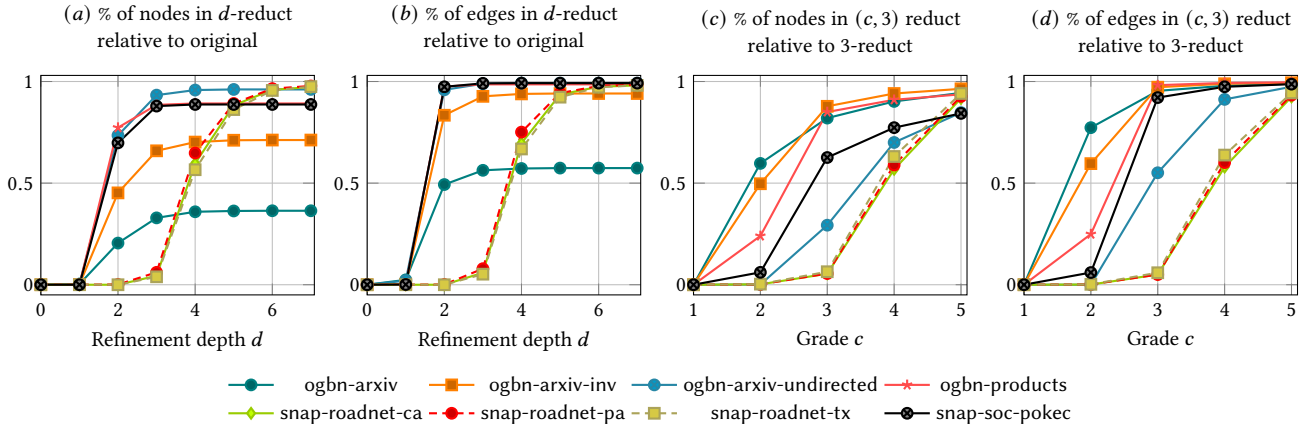


Figure 4: Normalized reduction in nodes and edges using d-reduction (a and b) and $(c, 3)$-reduction (c and d).

Table 3: Comparison of learning on the original uncompressed problem $\mathcal{P}_1$, the uncompressed problem with discretized labels $\mathcal{P}_2$, and $(c, d = 3)$-compressed variants $\mathcal{P}_3^c$.

Problem	Nodes (%)	Edges (%)	Accuracy (%)	Training time (s)	Training mem (GiB)
$\mathcal{P}_1$	100.0	100.0	66.7	61.66	2.14
$\mathcal{P}_2$	100.0	100.0	65.9	58.99	2.02
$\mathcal{P}_3^1$	76.9	95.9	60.9	48.10	1.59
$\mathcal{P}_3^3$	79.2	97.1	64.5	49.71	1.65
$\mathcal{P}_3^5$	79.3	97.2	64.9	49.55	1.65
$\mathcal{P}_3^\infty$	79.4	97.2	63.8	49.16	1.65

4.2 Learning

We next turn to validating empirically the effect of learning on compressed problems according to our methodology. Specifically, we apply our methodology to learning on ogbn-arxiv-inv with $d = 3$. We know from Section 4.1 that compression on this graph is reasonable in terms of nodes, but only marginal in terms of edges. Despite the modest compression, the effect on learning efficiency in terms of learning time and memory consumption is still positive, as we will see.

We hence focus in this section on ogbn-arxiv-inv and the associated learning problem from the OGBN benchmark: predict, for every paper in ogbn-arxiv-inv, its subject area (e.g., cs.AI, cs.LG, and cs.OS, ...). There are 40 possible subject areas. In addition to the citation network, we have available for each paper a 128-dimensional feature vector obtained by averaging the embeddings of words in its title and abstract. Our learning problem $\mathcal{P}_1$ hence consists of the ogbn-arxiv-inv graph in which each node is colored with this feature vector. The training set T consists of 90941 nodes, obtained conform the OGBN benchmark. The hypothesis space $\mathcal{S}$ consists of GNNs that all share the same topology and vary only in their concrete parameters. The topology consists of 3 GNN layers whose AGG function computes the mean of all colors of incoming neighbors. The COMB function applies a linear transformation to $G(v)$, applies a linear transformation to the result of the aggregation, and finally sums these two intermediates together. Each layer, except the final one, is followed by a batch normalisation as well ReLU to introduce non-linearity. The layers have dimensions $(128, 256)$, $(256, 256)$ and $(256, 40)$, respectively. We apply a 50% dropout between layers during training. Softmax is applied after the last layer to turn feature vectors into subject areas, and we use cross-entropy as loss function.

Unfortunately, the initial coloring in $\mathcal{P}_1$ is such that every node has a distinct color. Therefore, every node is in its own unique color refinement class, and compression is not possible. We therefore transform $\mathcal{P}_1$ into a problem $\mathcal{P}_2$ that can be compressed by first converting the 128-dimensional word embedding vectors into estimates of paper areas by means of a multilayer perceptron (MLP) that is trained on the nodes in T *but without having the graph structure available*. This hence yields an initial estimate of the paper area for each node. By learning a GNN on the graph that is colored by one-hot encodings of these initial estimates, the estimates can be refined based on the graph topology. We denote the new problem hence obtained by $\mathcal{P}_2$. Note that $\mathcal{P}_2$ has the same training set, hypothesis space, and loss function as $\mathcal{P}_1$. The MLP has input dimension 128, one hidden layer of dimension 256, and output layer of dimension 40.

We next compress $\mathcal{P}_2$ using (c, d)-reduction with $d = 3$. The resulting compressed problems are denoted $\mathcal{P}_3^c$. The corresponding compressed graph sizes are shown in Table 3. We note that, because we consider labeled graphs here, the compression ratio is worse than the maximum-compression scenario of Section 4.1.

We gauge the generalisation power of the learned GNNs by computing the accurracy on the test set, determined by the OGBN benchmark, comprising 48603 nodes. We learn for 256 epochs with learning rate 0.01 on all problems. All experiments are run on an HP ZBook Fury G8 with Intel Core i9 11950H CPU, 32 GB of RAM and NVIDIA RTX A3000 GPU with 6 GB RAM.

The results are summarized in Table 3. Comparing $\mathcal{P}_1$ with $\mathcal{P}_2$ we see that estimating the paper area through an MLP has marginal effect on the test accurracy, training time and memory consumption. Further comparing $\mathcal{P}_2$ with $\mathcal{P}_3^c$, which are equivalent by Theorem 3.11, we see that training accurracy is indeed comparable

between $\mathcal{P}_2$ and $\mathcal{P}_3^\infty$; we attribute the difference in accurracy to the stochastic nature of learning. There is a larger difference in accurracy between $\mathcal{P}_2$ and $\mathcal{P}_3^c$ when $c = 1$, i.e., when we compress based on bisimulation, than when $c > 1$. This is because c-graded compression is an approximation, as explained in Section 3.4, and because, as we can see, c-graded compression for $c > 1$ is nearly identical in size to ungraded compression. For $c > 1$ we may hence expect there to be only marginal differences w.r.t. ungraded compression. Learning the compressed problems $\mathcal{P}_3^c$ is more efficient than learning on the uncompressed $\mathcal{P}_2$, taking only 81.5–84.3% of the learning time and 78.7–81.6% of the memory respectively which is comparable to the reduction in number of nodes when compressing.

Our evaluation in this section is on a single learning problem; it should hence be interpreted as a preliminary insight that requires further evaluation. Based on these preliminary findings, however, we conclude that compressed learning can yield observable improvements in training time and memory consumption.

5 Conclusion and Future Work

We have proposed a formal methodology for exact compression of GNN-based learning problems. While the attainable exact compression ratio depends on the input graph, our experiment in Section 4.2 nevertheless indicates that observable improvements in learning efficiency are possible, even when the compression in terms of the number of edges is negligible.

In terms of future work, first and foremost our preliminary empirical evaluation should be extended to more learning tasks. Second, as we have seen c-graded color refinement offers a principled way of approximating exact compression, which becomes exact for hypothesis spaces of width c. It is an interesting question whether other existing approximate compression proposals [7, 17] can similarly be tied to structural properties of the hypothesis space.

Acknowledgments

We thank Floris Geerts for helpful discussions and the anonymous reviewers for their constructive comments. S. Vansummeren and J. Steegmans were supported by the Bijzonder Onderzoeksfonds (BOF) of Hasselt University under Grants No. BOF20ZAP02 and BOF21D11VDBJ. This work was further supported by the Research Foundation Flanders (FWO) under research project Grant No. G019222N. We acknowledge computing resources and services provided by the VSC (Flemish Supercomputer Center), funded by the Research Foundation – Flanders (FWO) and the Flemish Government.

References

[1] Christel Baier and Joost-Pieter Katoen. 2008. *Principles of Model Checking*. MIT Press.

[2] 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. *ACM SIGMOD Record* 49, 2 (2020), 6–17.

[3] Peter W. Battaglia, Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, Caglar Gulcehre, Francis Song, Andrew Ballard, Justin Gilmer, George Dahl, Ashish Vaswani, Kelsey Allen, Charles Nash, Victoria Langston, Chris Dyer, Nicolas Heess, Daan Wierstra, Pushmeet Kohli, Matt Botvinick, Oriol Vinyals, Yujia Li, and Razvan Pascanu. 2018. Relational Inductive Biases, Deep Learning, and Graph Networks. https://doi.org/10.48550/arXiv.1806.01261 arXiv:arXiv:1806.01261

[4] Jeroen Bollen, Jasper Steegmans, Jan Van den Bussche, and Stijn Vansummeren. 2023. *Learning Graph Neural Networks using Exact Compression (Extended Version)*. Technical Report. http://arxiv.org/abs/2304.14793.

[5] A. Cardon and M. Crochemore. 1982. Partitioning a Graph in $O(|A| \log_2 |V|)$. *Theoretical Computer Science* 19, 1 (July 1982), 85–98.

[6] Jie Chen, Tengfei Ma, and Cao Xiao. 2018. FastGCN: Fast Learning with Graph Convolutional Networks via Importance Sampling. In *ICLR*. OpenReview.net.

[7] Chenhui Deng, Zhiqiang Zhao, Yongyu Wang, Zhiru Zhang, and Zhuo Feng. [n. d.]. GraphZoom: A Multi-level Spectral Approach for Accurate and Scalable Graph Embedding. In *ICLR 2020*. OpenReview.net.

[8] Floris Geerts, Jasper Steegmans, and Jan Van den Bussche. 2022. On the Expressive Power of Message-Passing Neural Networks as Global Feature Map Transformers. In *Foundations of Information and Knowledge Systems (LNCS)*. Springer, 20–34.

[9] Alessandro Generale, Till Blume, and Michael Cochez. 2022. Scaling R-GCN Training with Graph Summarization. In *Companion Proceedings of the Web Conference 2022 (WWW '22)*. ACM, 1073–1082.

[10] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. [n. d.]. Neural Message Passing for Quantum Chemistry. In *ICML 2017*. JMLR.org, 1263–1272.

[11] Martin Grohe. [n. d.]. The Logic of Graph Neural Networks. In *LICS '21*. ACM, 1–17.

[12] William L. Hamilton. 2020. *Graph Representation Learning*. Morgan & Claypool Publishers.

[13] William L. Hamilton, Rex Ying, and Jure Leskovec. [n. d.]. Inductive Representation Learning on Large Graphs. In *NIPS 2017*. Curran Associates Inc., 1025–1035.

[14] 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. In *Advances in Neural Information Processing Systems*, Vol. 33. Curran Associates, Inc., 22118–22133.

[15] Wenbing Huang, Tong Zhang, Yu Rong, and Junzhou Huang. [n. d.]. Adaptive Sampling towards Fast Graph Representation Learning. In *NIPS 2018*. Curran Associates Inc., 4563–4572.

[16] Jure Leskovec and Andrej Krevl. [n. d.]. SNAP Datasets: Stanford Large Network Dataset Collection.

[17] Jiongqian Liang, Saket Gurukar, and Srinivasan Parthasarathy. 2021. MILE: A Multi-Level Framework for Scalable Graph Embedding. *Proceedings of the International AAAI Conference on Web and Social Media* 15 (May 2021), 361–372.

[18] Ningyi Liao, Dingheng Mo, Siqiang Luo, Xiang Li, and Pengcheng Yin. 2022. SCARA: Scalable Graph Neural Networks with Feature-Oriented Optimization. *PVLDB* 15, 11 (2022), 3240–3248.

[19] Haiyang Lin, Mingyu Yan, Xiaocheng Yang, Mo Zou, Wenming Li, Xiaochun Ye, and Dongrui Fan. 2022. Characterizing and Understanding Distributed GNN Training on GPUs. *IEEE Computer Architecture Letters* 21, 1 (2022), 21–24. https://doi.org/10.1109/LCA.2022.3168067

[20] Christopher Morris, Martin Ritzert, Matthias Fey, William L. Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. 2019. Weisfeiler and Leman Go Neural: Higher-Order Graph Neural Networks. *AAAI* 33, 01 (2019), 4602–4609. https://doi.org/10.1609/aaai.v33i01.33014602

[21] Jingshu Peng, Zhao Chen, Yingxia Shao, Yanyan Shen, Lei Chen, and Jiannong Cao. 2022. Sancus: Staleness-Aware Communication-Avoiding Full-Graph Decentralized Training in Large-Scale Graph Neural Networks. *PVLDB* 15, 9 (2022), 1937–1950. https://doi.org/10.14778/3538598.3538614

[22] Guillaume Salha, Romain Hennequin, Viet-Anh Tran, and Michalis Vazirgiannis. 2019. A Degeneracy Framework for Scalable Graph Autoencoders. In *IJCAI*. ijcai.org, 3353–3359.

[23] Qiange Wang, Yanfeng Zhang, Hao Wang, Chaoyi Chen, Xiaodong Zhang, and Ge Yu. [n. d.]. NeutronStar: Distributed GNN Training with Hybrid Dependency Management. In *SIGMOD 2022*. ACM, 1301–1315.

[24] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. 2021. A Comprehensive Survey on Graph Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems* 32, 1 (2021), 4–24.

[25] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. [n. d.]. How Powerful Are Graph Neural Networks?. In *ICLR 2019*. OpenReview.net.

[26] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, and Jure Leskovec. [n. d.]. Graph Convolutional Neural Networks for Web-Scale Recommender Systems. In *KDD 2018*. ACM, 974–983. https://doi.org/10.1145/3219819.3219890

[27] Binhang Yuan, Cameron R. Wolfe, Chen Dun, Yuxin Tang, Anastasios Kyrillidis, and Chris Jermaine. 2022. Distributed Learning of Fully Connected Neural Networks Using Independent Subnet Training. *PVLDB* 15, 8 (2022), 1581–1590.

[28] Chenguang Zheng, Hongzhi Chen, Yuxuan Cheng, Zhezheng Song, Yifan Wu, Changji Li, James Cheng, Hao Yang, and Shuai Zhang. 2022. ByteGNN: Efficient Graph Neural Network Training at Large Scale. *PVLDB* 15, 6 (2022), 1228–1242.

[29] Zulun Zhu, Jiaying Peng, Jintang Li, Liang Chen, Qi Yu, and Siqiang Luo. 2022. Spiking Graph Convolutional Networks. In *IJCAI*, Vol. 3. 2434–2440.