On The Uniform Expressivity of Graph Neural Networks

Von der Fakultät für Informatik der RWTH Aachen University
zur Erlangung des akademischen Grades
eines Doktors der Naturwissenschaften

genehmigte Dissertation

vorgelegt von

Eran Rosenbluth, M.Sc.

Berichter:

Prof. Dr. rer. nat. Martin Grohe

Professor Floris Geerts

Tag der mündlichen Prüfung: 15.10.2025

Acknowledgements

"Only some of a person's praise should be said in his presence..." (Yirmiah Ben Elazar, Talmud Bavli, Eruvin 18:B). Prof. Martin Grohe has been my guide in my doctorate for over three years, and we have collaborated on several papers. His involvement in my doctorate has been very influential. I enjoyed working with him, and learned from our discussions as well as from inspiring examples he set, in different aspects of the scientific work. I am grateful for his dedication and commitment.

Four years ago I have decided to move away from my born-and-raised country, Israel, increasing the travel distance to my parents and sisters from 2km to 3000km. Nevertheless, they have always been supportive and helpful, the ones I know I can always count on. I deeply cherish that.

For over three years I have been part of the Informatik 7 chair at RWTH Aachen, and spent much time there. I would like to express my appreciation for the good atmosphere created by members of that chair throughout that time, which contributed to making my stay enjoyable.

My choices of theoretical questions to research have been guided, among other factors, by relevance to practice. Many fruitful discussions with Jan Toenshoff, my colleague at the Informatik 7 chair, have provided an insightful and important view of the practical world. Jan has also been a collaborator I enjoyed working with, on 3 papers, and 2 desserts considered by some as two of the most underrated fine-dining dishes cooked between the Seine and the Rhine. For these and more, I much appreciate sharing a corridor with him for three years.

A call-for-candidates email from the spokesperson of the UnRAVeL Research Training Group (RTG), Prof. Joost-Pieter Katoen, initiated my engagement with the group and the RWTH many years ago. Although having little day-to-day interaction with Joost-Pieter, we have met in several most-significant crossroads through the course of my doctorate, where the path I would take depended also on his decisions. On all occasions, his approach was pragmatic, open-minded, and friendly, and I appreciate it a lot. It allowed me to choose my own way and arrive at this occasion, content with my academic achievements.

For four years I have been part of UnRAVeL RTG. The meetings, from one-hour talks to several-days workshops, were an enriching and fun experience, thanks to the administrative staff; the PIs; and the fellow student members. Moreover, the group's structure enabled me to find a guiding professor and a topic that matched my preferences. A special note of appreciation goes to the welcoming and helpful administrative support provided by Ms. Helene Bolke-Hermanns and Ms. Birgit Willms.

Finally, I would like to thank the following programs for seeing great value in doctoral research of Computer Science, and providing me with proper funding during my doctorate.

- The German Research Council (DFG). RTG 2236 (UnRAVeL).
- The European Union. ERC, SymSim, 101054974. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them.

Contents

Introduction

The main subject of this work are certain parameterized algorithms whose input is a graph. Specifically, we analyze what functions on graphs these algorithms can compute.

1.1 Context and Motivation

Machine Learning

A learning task is a task of finding a function from a certain domain to a certain range, given some form of guiding information, for example: Finding a function from a 1-day period, 10-minute resolution, sequence of weather data such as temperature; barometric pressure; etc. to the amount of precipitation in the next day, given historical 1-year data.

In most cases, it is unfeasible to consider the entire space of potential functions. Hence, a subspace for which an efficient learning algorithm exists must be chosen, typically in the form of a parameterized algorithm referred to as an *architecture*: Each configuration of the architecture's parameters determines a concrete algorithm, referred to as a *model*, which computes a specific function, hence the set of all possible configurations determines a function space. Denote that function space by H, the *expressivity* of the architecture is H and its limits, allowing for additive approximation. The learning algorithm searches through the parameters' space, seeking a model that approximates the target function. Success of the learning algorithm then depends on two conditions:

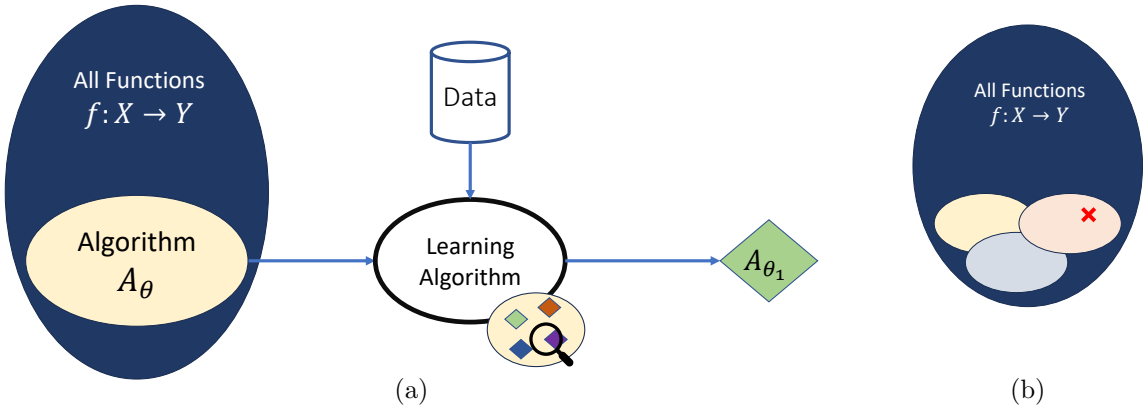
(1) H contains the target function, or a sequence of functions that converge to it.

(2) Given that (1) holds, the learning algorithm can find that target function.

As the decider of (1), an architecture's expressivity is an important fundamental property of it. See Figure 1.1 for an illustration of the learning process and meaning of expressivity.

Machine Learning On Graphs

Graph learning tasks are learning tasks in which the input domain consists of graphs with featured vertices, and the output range consists of either new values for the vertices or one value for the whole graph. The function range may be a finite set, in which case the learning task is referred to as *classification*, or an infinite one e.g. the rational numbers, in which case the task is a *regression* task. Note that in the latter case, it is usually required only to approximate the function output up to a given epsilon.



(a) (b)

Figure 1.1: (a) On the left, an algorithm parameterized by θ (over some space Θ) is chosen, the learning algorithm searches for an optimal setting of θ considering the data, and produces a model where θ is set to $\theta_1 \in \Theta$.
(b) On the right, different algorithms with different expressivity. The red X represents the function that is searched for. If the left or bottom algorithms are chosen, the learned model will never be a good approximation, as their expressivity does not contain the target function.

A task for example would be: Find a function that maps a graph-representation of corporate holdings' structure, where vertices represent companies; they are connected if one has a holding in the other; and a vertex feature is its company's financial data, to a 'Yes/No' answer of whether the corporate will file for bankruptcy in the next 3 years.

1.2 Architectures

Multilayer Perceptron

A *multilayer perceptron* (MLP) is a general form of a Feed Forward Neural Network (FNN). An MLP is defined by a finite sequence of pairs of a weight matrix and a bias vector $(w_1, b_1), \ldots, (w_m, b_m)$ also referred to as its *layers*, and a non-linear *activation function* $\sigma : \mathbb{R} \to \mathbb{R}$. In this paper, whenever we refer to an MLP we mean an MLP with activation function relu $:= \mathsf{max}(0, x)$.

An MLP has a fixed input dimension and output dimensions - number of columns of w_1 and rows of w_m, and computes the composition of affine functions defined by its layers, with the activation function being applied to the output of each of the intermediate layers. As detailed later, MLPs serve as building blocks in the architectures that are the main subject of this paper - GNNs.

While being universal approximators for continuous functions on compact domains [HSW89], the expressivity of MLPs on unbounded domains is quite limited e.g. they cannot approximate $f(x) := x^2$.

Message Passing Graph Neural Networks

The term Graph Neural Network was introduced in [Sca+08]. Message Passing Graph Neural Networks (MP-GNNs, here GNNs for short) [KW17b; Gil+17a] are a class of architectures for graphs, often used in graph learning tasks i.e. their inputs are graphs

with nodes featured by a vector of certain dimension. In the broadest theoretical setting, the vector is over the real numbers, however, most expressivity-analysis studies consider more restrictive domains: Most consider a finite domain, some consider a compact subset of the real numbers, and some consider integers. As for directionality, we consider only undirected graphs. While there are learning tasks that involve directed graphs, and GNNs definitions can be adapted for such cases, we choose to focus on the classical and pure setting of no-directionality.

A GNN is defined by a finite sequence of node-level computations $L_1, \ldots, L_m$ referred to as *layers*, executed by all vertices in parallel and in sync i.e. all vertices execute layer 1, then all execute layer 2, and so on. Each layer L_t comprises a message; aggregation; and combination function, $\mathsf{msg}_t; \mathsf{agg}_t; \mathsf{comb}_t$. Denote by $v^{(t)}$ the value of a vertex v in a graph G after the first t layers, with $v^{(0)}$ being its initial feature, then layer t+1 operates on v as follows:

a. First, the message $m_v w^{(t+1)} \coloneqq \mathsf{msg}_{t+1}(v^{(t)}, w^{(t)})$ is computed for every neighbor $w \in N_G(v)$. Common message functions are the identity function (of the neighbor's value), and an MLP.

b. Then, the aggregation of messages, $\mathsf{agg}_v^{(t+1)} \coloneqq \mathsf{agg}_{t+1}(\{\{m_v w^{(t+1)} | w \in N_G(v)\}\})$, is computed. The aggregation function is typically dimension-wise sum; mean; or max, but can also be other order-invariant functions with a fixed output-dimension. It is common for GNNs to have the same aggregation in all layers e.g. $\forall t\ \mathsf{agg}_t = \text{sum}$.

c. Finally, the new value $v^{(t+1)} \coloneqq \mathsf{comb}_{t+1}(v^{(t)}, \mathsf{agg}_v^{(t+1)})$ is the combination of the current value of v and the aggregation value. The implementation of comb is always by an MLP, hence the name Graph Neural Networks.

In graph-level tasks, the GNN applies a final readout algorithm $R = (F, \mathsf{agg})$ comprising an aggregation followed by an MLP, and the final result is $F(\mathsf{agg}\big(\{\{v^{(m)} \mid v \in V(G)\}\}\big))$.

The node-centric, node-indifferent, nature of the computation means every GNN can technically be applied to graphs of all sizes, and GNNs are isomorphism-invariant. The MLP parts of the layers give GNNs their learnability qualities.

One way to extend the message; aggregate; combine scheme is to add a global computation to each layer, whose value is given as an additional input to the combine function. Common examples of such architectures are GNNs with *global self attention*, known as 'Graph Transformers', and GNNs with *virtual nodes*.

Perhaps less common, but not less useful and successful, *recurrent* GNNs [Sca+08; GM10] are an extension of GNNs in which the sequence of layers can be reiterated a number of times that depends on the input graph, with the output of one iteration being the input of the next.

Another way to extend the basic framework is by enhancing the vertices' features with precomputed local high-order structural information, see for example [Bar+21]. This and several other extensions are out of the scope of this work.

1.3 Expressivity of GNNs

Being used in solving learning tasks, understanding the expressivity of GNNs is of great importance. One should ask what do we mean by "understanding". Clearly, an architec-

ture's expressivity is precisely described by its own definition. For example, a regression architecture $A(x) \coloneqq ax + b$ - with a, b being the parameters, can express precisely all functions of the form $f(x) \coloneqq ax + b$. However, for more complex architectures, their description is in different terms than those we use to describe interesting function classes. The latter may be for example a useful logic; an important complexity class; or an informative description of specific functions, such as: The difference between a node's degree and the average degree of its neighbors, formally $|N_G(v)| - \frac{\Sigma_{w \in N_G(v)} |N_G(w)|}{|N_G(v)|}$. Understanding the expressivity then means knowing what the algorithmic description of an architecture entails in meaningful functional terms. Another type of understanding is knowing how the expressivity of two architectures relate i.e. is one contained in the other or are they partially disjoint. Note that almost always there are trivial functions that both can express, hence two architectures are almost never completely disjoint.

Analyzing their definition, the boundaries of GNNs expressivity can be attributed to three factors:

1. The message-passing scheme. A main consequence of this is a distinguishing power that cannot exceed that of the Color Refinement algorithm or 1-dimensional *Weisfeiler-Leman* (1-WL) as it is sometimes referred to [Mor+19a; Xu+19a].
2. The fixed number of message passes. An obvious effect of that is a fixed information-radius for each node's computation: If there are m message passes then the result for each node cannot be affected by the value of nodes that are more distant. Another effect is inability to compensate for the expressivity limitations of the layers' functions (see next point) by means of repetition.
3. The combination and aggregation functions that make GNNs' layers. For reasons of learnability and runtime performance, these are of specific classes as described in Section 1.2. We have already mentioned MLPs limited expressivity, and as for the aggregation, all described options potentially lose information about the multiset of neighbors' values.

Different studies have used different definitions of expressivity.

The weakest notion is *non-uniform* expressivity. There, a GNN architecture is said to express a function f if and only if for every graph size n there exists a model M_n of that architecture, that approximates f on all graphs **of that size**. That is, the expressing model - and also its size - may depend on n. Combining the results of [Xu+19a; Mor+19a] and of [Che+19] provides that GNNs can be non-uniformly as expressive as message-passing can be, when considering compact initial-feature domain. Using a technique known as Random Node Initialization, where each node is augmented with a random feature, GNNs can probabilistically non-uniformly express every computable function [Abb+21b; SYK21].

A stronger notion is *non-uniform polynomial* expressivity. It is similar to non-uniform, but with the additional requirement that a GNN's size is polynomial in the size of the graphs it is good for. In [Gro23] it is shown that the non-uniform expressivity of bounded-depth polynomial-size GNNs is equivalent to that of a certain logic and circuit complexity class, when considering a finite initial-feature domain.

In summary, under the non-uniform notion, factors (2);(3) above can be overcome. However, that notion is a weak guarantee whose relevance to what GNNs can compute

in practice is very limited: Non-uniform non-polynomial expressive GNNs are too large, before anything else. Non-uniform polynomial expressive GNNs may succeed at inference time only when the sizes of graphs do not exceed those of the graphs at training time, otherwise they fail miserably - as implied by the theory and as we have also witnessed in our experiments.

The strongest and more meaningful notion of expressivity, both to theory and practice, and the one that we use in this work, is *uniform* expressivity. Here, a GNN architecture is said to express a function f if and only if there exists a model M of that architecture, that approximates f on graphs **of all sizes**. In informal terms, a uniform model computes an algorithm - accepting any input size, while a non-uniform one computes a sort of lookup table for the finitely many graphs of a specific size. From a practical standpoint, uniform expressivity means it may be possible to train a GNN model on graphs of smaller sizes and this model will be useful in approximating the target function also on graphs of larger sizes. We are not aware of any meaningful tight bounds shown for the uniform expressivity of (non-recurrent) GNNs. A notable lower-bound is [Bar+20a]. Various recurrent architectures have been tightly characterized, varying in their computational power and resemblance to practical recurrent GNNs. A detailed overview of several of them is provided in Section 6.1.

1.4 Contributions Of This Work

In this work, we shed light on the uniform expressivity of GNNs, which from here on we may also refer to simply as 'expressivity'. Our results are theoretical, accompanied in part with experiments that demonstrate their manifestation in practice.

For non-recurrent GNNs, it is not clear how to meaningfully and tightly bound their expressivity. However, other important questions remain to be answered. In particular, questions concerning the effect that GNNs' prime hyper-parameters have on expressivity. These hyper-parameters are the message and aggregation functions, and the global algorithm in case such an algorithm is added. In each of the chapters 3; 4; and 5, we focus on a different prime hyper-parameter and examine how common choices for its argument, i.e. different architectures, compare in their expressivity: Are they equivalent; does one subsume the other; or are they partially disjoint. Moreover, target functions that we prove to separate different architectures may provide an intuition as to what tasks they should and should not be used for.

Our comparative results for non-recurrent GNNs not only tell us about the strengths and weaknesses of the various specific architectures (e.g. sum-aggregation; identity-msg GNNs) but also demonstrate that the non-recurrent scheme is fundamentally limited. This is when we turn to study recurrent GNNs, in Chapter 6. There, we prove that computable recurrent GNNs can express all message-passing algorithms, with only a polynomial time and space overhead.

The following is an overview of the main results in this work, see Figure 1.2 for a unified view of several results from different chapters:

- In chapter 3 we compare the uniform expressivity of GNNs with different aggregation functions. Key takeaway: sum-aggregation GNNs (Sum-GNNs) do not subsume all other GNNs, unlike in the non-uniform setting.

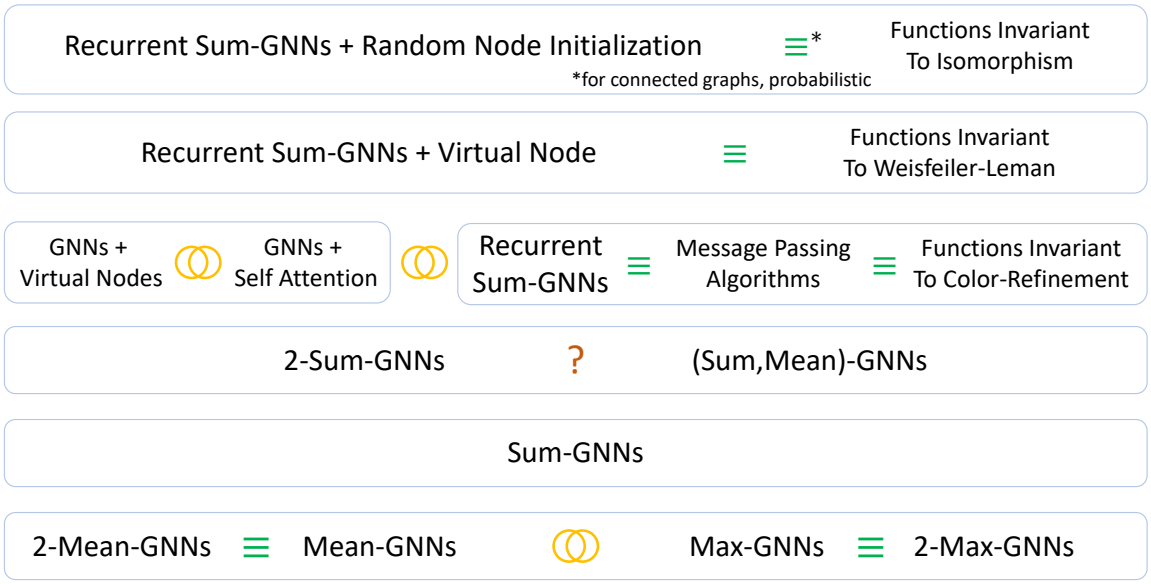


Figure 1.2: Uniform expressivity hierarchy of node-level functions, in a **finite initial-features domain** setting. A unified view of results from different chapters. Architectures in an upper level strictly subsume architectures in a lower level. A double-ring represents partially-disjoint relation. The question mark represents an unknown relation.
Levels 1-3: The relations between (1-side) GNNs with different aggregations are proven in Chapter 3. The relations between 1-side and 2-side GNNs are proven in Chapter 4. **Level 4L**: The relation on the left part of level four is an extension of the main result in Chapter 5. There, we consider combining the global computations of virtual nodes or self-attention, with specific GNNs, as these are the hybrid architectures common in practice. However, the proofs of the no-subsumption statements are such that the GNN - message passing - part of the architecture is irrelevant, hence we can refer to the augmentation of any class of GNNs, and in particular the class of all GNNs. **Level 4M**: The relation between the two parts of level four is not formally proven in this work but is not difficult to prove. **Level 4R**: The relations on the right part of level four are proven in Chapter 6. **Levels 5-6**: The relations on levels five and six are additional results of Chapter 6.

- In chapter 4 we compare the uniform expressivity of GNNs with different message functions. Key takeaway: sum-aggregation GNNs with 2-sided message functions (2-Sum-GNNs) are strictly stronger than those with 1-sided ones (Sum-GNNs), however this is not the case for mean and max aggregation GNNs.

- In chapter 5 we compare the uniform expressivity of GNNs augmented with different kinds of global algorithms. Key takeaway: GNNs augmented with global self-attention, a.k.a. GPS Transformer, do not subsume GNNs augmented with sum-aggregation virtual node.

- In chapter 6 we tightly characterize the uniform expressivity of computable recurrent GNNs. Key takeaway: Finite-precision parameters; single-layer; sum-aggregation; identity-message, recurrent GNNs, can compute any message-passing algorithm, and any computable function that is invariant to the Color Refinement representation of a vertex, with only a polynomial time and space overhead.

Statement Of Personal Contribution

The results in this work are my own work - possibly following fruitful discussions with my collaborators and colleagues, except for the following:

- Lemma 3.4.2 and its proof idea are Martin Grohe's ideas. I worked out the final technical proof-details.
- The implementation of the experiments in Section 3.7 i.e. the complete code-development, and execution of the learning process, was conducted by Jan Toenshoff.
- Section 5.3, most importantly Proposition 5.3.1, is the work of Martin Grohe.
- The implementation of the experiments in Section 5.5 i.e. the complete code-development, and execution of the learning process, was conducted by Jan Toenshoff and Berke Kisin.
- Section 6.3, most importantly, the DAG representation of a color and Lemma 6.3.3, is the work of Martin Grohe.
- Theorem 6.5.1 and Corollary 6.5.3 are the work of Martin Grohe.

Preliminaries

By $\mathbb{N};\mathbb{Q};\mathbb{R}$ we denote the natural; rational; and real numbers respectively. We define $\mathbb{Q}_{[0,1]} \coloneqq \{q : q \in \mathbb{Q}, 0 \leq q \leq 1\}$. Let $\mathbb{R}^d \ni v = v_1, \ldots, v_d$ be a d-dimension vector, we define $v(i) \coloneqq v_i$ the value at position i, and $v[a,b] \coloneqq v_a, \ldots, v_b$ the sub-vector from position a to b. Let $c \in \mathbb{R}$, we define $v + c \coloneqq v_1 + c, \ldots, v_d + c$. Let $Real^d \ni u = (u_1, \ldots, u_d)$, we define $\mathbb{R}^d \ni v + u \coloneqq (v_1 + u_1), \ldots, (v_d + u_d)$. Let $\mathbb{R}^m \ni v = (v_1, \ldots, v_m)$, $\mathbb{R}^n \ni u = (u_1, \ldots, u_n)$, we define $v, u \coloneqq (v_1, \ldots, v_m, u_1, \ldots, u_n) \in \mathbb{R}^{m+n}$ the concatenation of the two. For a vector $v \in \mathbb{R}^d$ we define $\dim(v) \coloneqq d$, and for a matrix $W \in \mathbb{R}^{d_1 \times d_2}$ we define $\dim(W) \coloneqq (d_1, d_2)$. For a set S, we denote the set of all finite multisets with elements from S by $\left(\!\binom{S}{*}\!\right)$. When referring to an aggregation operation on scalars $f : \left(\!\binom{\mathbb{R}}{*}\!\right) \to \mathbb{R}$ as an aggregation of vectors $g : \left(\!\binom{\mathbb{R}^d}{*}\!\right) \to \mathbb{R}^d$ for some $d \in \mathbb{N}$, we mean the dimension-wise operation e.g. let $\{\{x_1, \ldots, x_n\}\}, x_i \in \mathbb{R}^d$ then $\mathsf{max}(x_1, \ldots, x_n) \coloneqq \mathsf{max}\big(x_1(1), \ldots, x_n(1)\big), \ldots, \mathsf{max}\big(x_1(d), \ldots, x_n(d)\big)$. For $r \in \mathbb{R}$ we define $\text{sign}(r) \in \{-1, 0, 1\}$ to be the usual sign function.

We call a parameterized algorithm, such as MLP and GNN (see below), an *architecture*. We call an architecture whose parameters are already set, a *model* of the architecture.

We define $\forall x \in \mathbb{R}\ \text{relu}(x) \coloneqq \mathsf{max}(0, x)$, and $\forall x \in \mathbb{R}\ \text{lsig}(x) \coloneqq \min(1, \text{relu}(x))$.

2.1 Graph

An *undirected graph* $G = \langle V(G), E(G) \rangle$ is a pair, $V(G)$ being a set of vertices and $E(G) \subseteq \{\{u, v\} \mid u, v \in V(G)\}$ being a set of undirected edges. For a vertex $v \in V(G)$ we denote by $N_G(v) \coloneqq \{w \in V(G) \mid \{w, v\} \in E(G)\}$ the *neighbourhood* of v in G, and we denote the size of it by $n_v \coloneqq |N_G(v)|$. When clear from the context, we may omit the G subscript.

Let S be a set, a (vertex) *featured graph* $G = \langle V(G), E(G), S^d, Z(G) \rangle$ is a 4-tuple being a graph with a *feature map* $Z(G) : V(G) \to S^d$, mapping each vertex to a d-tuple over S. We denote the set of graphs featured over S^d by $\mathcal{G}_{S^d}$, we define $\mathcal{G}_S \coloneqq \bigcup_{d \in \mathbb{N}} \mathcal{G}_{S^d}$, and we denote the set of all featured graphs by $\mathcal{G}_*$. The special set of graphs featured over $\{1\}$ is denoted $\mathcal{G}_1$. We denote the set of all feature maps that map to S^d by $\mathcal{Z}_{S^d}$, we denote $\bigcup_{d \in \mathbb{N}} \mathcal{Z}_{S^d}$ by $\mathcal{Z}_S$, and we denote the set of all feature maps by $\mathcal{Z}_*$. Let $D \subseteq \mathcal{G}_*$ be a featured-graph domain, we call a mapping $f : \mathcal{G}_D \to \mathcal{Z}_*$ to new feature maps a *feature transformation*.

Our main focus is on expressing feature transformations - assigning new values to the vertices of a featured graph, however, several of our results consider the usage of GNNs to assign one value-vector (of a fixed dimension) to the whole graph. Let $p, q \in \mathbb{N}$, we call an order-invariant mapping $f : \mathcal{Z}_{\mathbb{R}^p} \to \mathbb{R}^q$ from feature maps to q-tuples, a *readout*

function. Both sum and mean are commonly used to aggregate feature maps, possibly followed by an MLP that maps the aggregation value to the final readout. We call a mapping $f : \mathcal{G}_{S^p} \to \mathbb{R}^q$, from featured graphs to q-tuples, a *graph embedding*. Let F be a set of feature transformations of the form $f : \mathcal{G}_{S^p} \to \mathcal{Z}_{\mathbb{R}^q}$, and let $r : \mathcal{Z}_{\mathbb{R}^q} \to \mathbb{R}^w$ be a readout, we notate the set of graph embeddings $\{r \circ f : f \in F\}$ by $r \circ F$.

Let G, G' be two graphs, and let $f : V(G) \to V(G')$ be a bijective mapping. We say that f is an *isomorphism* if and only if it preserves the features and the edges, that is,

$$\forall v \in V(G)\ Z(G)(v) = Z(G')(f(v)) \bigwedge \forall v, w \in V(G)\ vw \in E(G) \Leftrightarrow f(v)f(w) \in E(G')$$

We say that two graphs G, G' are *isomorphic*, notated $G \cong G'$, if and only if there exists an isomorphism for them. Let G, G' be two graphs and let $v \in V(G), v' \in V(G')$, we say that v, v' are *corresponding*, notated $v \cong v'$, if and only if there exists an isomorphism f such that $v' = f(v)$. We say that a graph embedding F is *isomorphism-invariant* if and only if $\forall G, G'\ G \cong G' \Rightarrow F(G) = F(G')$. We say that a feature transformation F is *equivariant* if and only if $\forall G, G'\ \forall v \in V(G), \forall v' \in V(G')\ \ v \cong v' \Rightarrow F(G)(v) = F(G')(v')$. Although 'equivariant' is the more common term, we may use the term 'isomorphism-invariant' also with respect to feature transformations.

2.2 Multi-Layer Perceptron

For $x \in \mathbb{R}$ we define $\text{relu}(x) \coloneqq \mathsf{max}(0, x)$, and refer to it as an *activation function*. A relu-activated Multi-Layer Perceptron (MLP) $F = (l_1, \ldots, l_m), l_i = (w_i, b_i)$ of depth m and I/O dimensions $d_{in}; d_{out}$, is a sequence of pairs of rational matrix w_i and bias vector b_i such that

$$\dim(w_1)(2) = d_{in}, \dim(w_m)(1) = d_{out}, \qquad \forall i > 1 \dim(w_i)(2) = \dim(w_{i-1})(1),$$

$$\forall i \in [m] \dim(b_i) = \dim(w_i)(1)$$

It defines a function

$$f_F(x) \coloneqq \text{relu}(w_m(... \text{relu}(w_2(\text{relu}(w_1(x) + b_1)) + b_2)...) + b_m)$$

When clear from the context, we may denote $f_F(x)$ by $F(x)$. Let F be an MLP, we define its size to be $|F| \coloneqq \Sigma_{i \in [m]} \dim(w_i)(1) \cdot (\dim(w_i)(2) + 1)$. Throughout this work, when we refer to MLPs we mean relu-activated MLPs unless stated otherwise. Note that these subsume every finitely-many-pieces piecewise-linear activated MLPs, thus, every inexpressivity result for the former class applies to the latter.

MLPs are Lipschitz-Continuous, that is, for every MLP F there exists a minimal $a_F \in \mathbb{R}_{\geq 0}$ such that for every input and output coordinates (i, j), for every specific input arguments $x_1, \ldots, x_n$, and for every $\delta > 0$, it holds that

$$\left| f_F(x_1, \ldots, x_n)_j - f_F(x_1, \ldots x_{i-1}, x_i + \delta, \ldots, x_n)_j \right| / \delta \leq a_F$$

We name a_F the *Lipschitz-Constant* of F.

2.3 Message Passing Graph Neural Network

A *Message Passing Graph Neural Network* (MP-GNN, GNN for short) N of depth m; input dimension p; vertices-output dimension q; and optional graph-output dimension r, is a finite sequence of triplets, referred to as *layers*, plus an optional *readout* pair (for graph output)

$$N = ((\mathsf{comb}_1, \mathsf{agg}_1, \mathsf{msg}_1), \ldots, (\mathsf{comb}_m, \mathsf{agg}_m, \mathsf{msg}_m), (\mathsf{mlp}, \mathsf{agg}))$$

Define $r_0 \coloneqq p$ and $r_m \coloneqq q$. Then, for $i > 1$ layer i of dimensions $q_i; r_i$ comprises a message algorithm $\mathsf{msg}_i : \mathbb{R}^{r_{i-1}} \times \mathbb{R}^{r_{i-1}} \to \mathbb{R}^{q_i}$, an aggregation function $\mathsf{agg}_i : \binom{\mathbb{R}^{q_i}}{*} \to \mathbb{R}^{q_i}$, and a combination algorithm $\mathsf{comb}_i : \mathbb{R}^{r_{i-1}} \times \mathbb{R}^{q_i} \to \mathbb{R}^{r_i}$. The combination algorithm is always an MLP. The message algorithm is usually either the identity of the second argument or an MLP. The aggregation function is typically dimension-wise sum; mean; or max, but can also be other order-invariant functions with a fixed output-dimension.

The sequence of layers defines a feature transformation $f_N : \mathcal{G}_{\mathbb{R}^p} \to \mathcal{Z}_{\mathbb{R}^q}$ as follows: Let $G \in \mathcal{G}_{\mathbb{R}^p}$ and $v \in V(G)$, then we define:

1. $N^{(0)}(G)(v) \coloneqq Z(G)(v)$ i.e. the initial feature map.
2. $\forall t \in [m] \quad N^{(t)}(G)(v) \coloneqq \mathsf{comb}_t\Big(v_N^{(t-1)}, \mathsf{agg}_t\big(\{\!\{\mathsf{msg}_t(v_N^{(t-1)}, w_N^{(t-1)}) \mid w \in N_G(v)\}\!\}\big)\Big)$ i.e. the feature map after applying the first t layers of N.
3. $N(G, v) \coloneqq N^{(m)}(G, v)$ i.e. the final feature map - the output of N.

The optional pair $(\mathsf{mlp}, \mathsf{agg})$, if provided, comprises a global aggregation algorithm $\mathsf{agg} : \binom{\mathbb{R}^q}{*} \to \mathbb{R}^q$, and a final MLP $\mathsf{mlp} : \mathbb{R}^q \to \mathbb{R}^r$, which defines a graph embedding for every $G \in \mathcal{G}_{\mathbb{R}^p}$:

$$N(G) \coloneqq \mathsf{mlp}(\mathsf{agg}(\{\!\{N(G, v) \mid v \in V(G)\}\!\})$$

It is common for GNNs to have the same aggregation in all layers e.g. $\forall i \in [m]$ $\mathsf{agg}_i = \mathsf{sum}$. For an aggregation function agg, we denote by agg-GNNs the class of GNNs for which $\forall i \in [m]$ $\mathsf{agg}_i = \mathsf{agg}$, for example Sum-GNNs are those with $\mathsf{agg} = \mathsf{sum}$. For aggregation functions $\mathsf{agg}_a, \mathsf{agg}_b$, we denote by $(\mathsf{agg}_a, \mathsf{agg}_b)$-GNNs the class of GNNs where $\forall i \in [m]$ $\mathsf{agg}_i = \mathsf{agg}_a, \mathsf{agg}_b$. That is, the input of $\mathsf{comb}_t, t \in [m]$ is

$$\Big(v_N^{(t-1)}, \mathsf{agg}_a\big(\{\!\{\mathsf{msg}_t(v_N^{(t-1)}, w_N^{(t-1)}) \mid w \in N_G(v)\}\!\}\big), \mathsf{agg}_b\big(\{\!\{\mathsf{msg}_t(v_N^{(t-1)}, w_N^{(t-1)}) \mid w \in N_G(v)\}\!\}\big)\Big)$$

Unless mentioned otherwise, agg-GNNs; $(\mathsf{agg}_a, \mathsf{agg}_b)$-GNNs; etc. refer specifically to those in which the message function is trivial i.e. it outputs the neighbor's value.

We define the size of N to be the sum of its underlying MLPs' sizes, that is,

$$\big|N\big| \coloneqq |\mathsf{mlp}| + \Sigma_{i \in [m]}(\big|\mathsf{comb}_i\big| + \big|\mathsf{msg}_i\big|)$$

The operand $\big|\mathsf{msg}_i\big|$ refers to the number of parameters of the message algorithm, for example when the algorithm is an MLP then $\big|\mathsf{msg}_i\big|$ is the size of the MLP.

Note that outside this work the term GNN may refer to a class of architectures that strictly contains MP-GNNs, however in this work we use it as short for MP-GNNs.

2.4 Expressivity

Let $p, q \in \mathbb{N}$, and a set S. Let $F \subseteq \{f \mid f : \mathcal{G}_{S^p} \to \mathcal{Z}_{\mathbb{R}^q}\}$ be a set of feature transformations from $\mathcal{G}_{S^p}$ to $\mathcal{Z}_{\mathbb{R}^q}$, and let $h : \mathcal{G}_{S^p} \to \mathcal{Z}_{\mathbb{R}^q}$ be a feature transformation. We say that F *uniformly expresses* h, notated $F \approx h$, if and only if

$$\forall \varepsilon > 0 \; \exists f \in F : \forall G \in \mathcal{G}_{S^p} \forall v \in V(G) \; \left| f(G)(v) - h(G)(v) \right| \leq \varepsilon$$

That is, for every approximation gap there is a function in the set that additively-approximates the target function by that gap. The uniformity part is that the function approximates the target function on graphs **of all sizes**. We use the terms 'approximates' and 'expresses' interchangeably.

Let F, H be two sets of feature transformations of the form $f : \mathcal{G}_{S^p} \to \mathcal{Z}_{\mathbb{R}^q}$, we say that F *subsumes* H, notated $F \geq H$ if and only if for every $h : \mathcal{G}_{S^p} \to \mathcal{Z}_{\mathbb{R}^q}$ it holds that $H \approx h \Rightarrow F \approx h$. If the subsumption holds only for graphs featured with a subset $T^p \subset S^p$ we notate it by $F \geq^T H$.

Our results concern GNN architectures, and as a GNN architecture determines a set of feature transformations (considering all parameters-configurations) we may use the above terms and notations with respect to these architectures - implicitly referring to the sets they determine.

We use the expressivity terms and notations defined for feature transformations, for graph embeddings as well.

Some Might Say All You Need Is Sum

"...there is no thing that has not its place."

Simeon ben Azzai, Pirkei Avot 4:3

3.1 Intro

This chapter is based on [RTG23]. It focuses on GNNs in which all layers have the same aggregation function and the trivial message algorithm - passing the neighbor's value.

Several works have explored the non-uniform expressivity of Sum-GNNs, characterizing it in various ways, for example [Xu+19a; Mor+19a; Gro23]. The results of all of them, each in its own setting, imply that in the non-uniform notion Sum-GNNs subsume all other GNNs. In the uniform notion though, no tight characterizations and no comparative results (with other classes) have been made for Sum-GNNs, leaving it an open question whether it is sum-supreme also there.

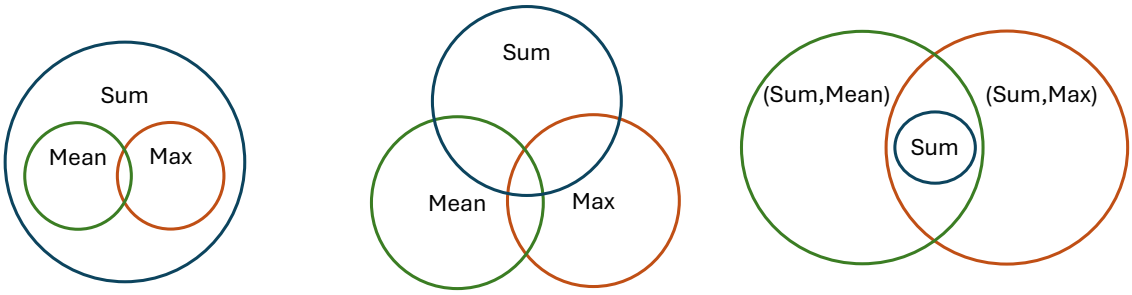
In this chapter, we examine the role of the aggregation function in the uniform expressivity of GNNs. In particular, we ask whether Sum-GNNs subsume Mean-GNNs and Max-GNNs, considering different settings of initial-feature domains.

3.1.1 New Results

Mainly, we prove that Sum-GNNs do not subsume Mean-GNNs nor Max-GNNs (and vice versa), in terms of vertices-embedding expressivity as well as graph-embedding expressivity. Note that while our inexpressivity statements consider additive approximation - as in our definition of expressivity, they hold true also for multiplicative approximation, and the respective proofs require not much additional argumentation to show that. The results progression is as follows:

1. Advantage Sum: For the sake of completeness, in Section 3.3 we prove that even with single-value input features, the neighbors-sum function which can be trivially exactly computed by a Sum-GNN cannot be approximated by any Mean-GNN or Max-GNN.

2. Sum subsumes: Moreover, in Section 3.4 we prove that if the input features are bounded, Sum-GNNs can approximate all Mean-GNNs or Max-GNNs, though not without an increase in size which depends polynomially on the required accuracy, and exponentially on the depth of the approximated Mean-GNNs or Max-GNNs.



(a) Bounded domain **(b)** Unbounded domain **(c)** Finite domain

Figure 3.1: Expressivity relations between GNNs with different aggregations, in different initial-feature domain settings.

3. Advantage Mean and Max: In Section 3.5.1 we show that if we allow unbounded input features then functions that are exactly computable by Mean-GNNs ; Max-GNNs; and others, cannot be approximated by Sum-GNNs.

4. Essential also with finite initial-features domain: In Section 3.5.2 we prove that even with just single-value input features, there are functions that can be exactly computed by a (Sum,Mean)-GNN (a GNN that uses both sum-aggregation and Mean-aggregation) or by a (Sum,Max)-GNN, but cannot be approximated by Sum-GNNs.

5. The world is not enough: In Section 3.6, we examine GNNs with any finite combination of sum; mean; max and other aggregations, and prove upper bounds on their expressivity already in the single-value input features setting.

Results 2,3,4 are illustrated in Figure 3.1.

In Section 3.7 we experiment with synthetic data and observe that what we proved to be expressible is to an extent also learnable, and that in practice inexpressivity is manifested in a significantly higher error than implied by the theory.

3.2 Terms and Concepts

For a featured graph G and a vertex $v \in V(G)$ we define $\mathrm{sum}(v) \coloneqq \Sigma_{w \in N(v)} Z(G)(w)$, $\mathsf{mean}(v) \coloneqq \frac{1}{n_v}\mathrm{sum}(v)$, and $\mathsf{max}(v) \coloneqq \mathsf{max}(Z(G)(w) : w \in N(v))$.

3.3 Mean and Max Do Not Subsume

It has already been stated that Sum-GNNs can express functions that Mean-GNNs and Max-GNNs cannot [Xu+19b]. For the sake of completeness we provide formal proofs that Mean-GNNs and Max-GNNs subsume neither Sum-GNNs nor each other.

3.3.1 Mean and Max Do Not Subsume Sum

Neither Mean-GNNs nor Max-GNNs subsume Sum-GNNs, even when the initial-feature domain is a single value.

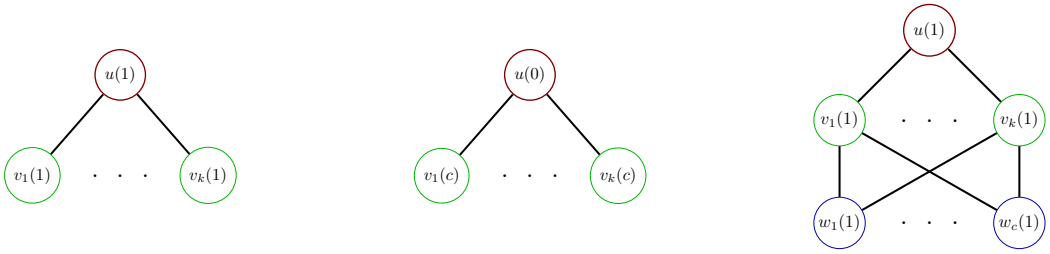


Figure 3.2: A star graph with k leaves, featured over a single-value initial-feature domain (left); a star graph with k leaves, featured over $\mathbb{N}_{>0}$ (middle); a tripartite graph, with k intermediates fully connected to c leaves, featured over a single-value initial-feature domain (right).

We define a featured star graph with (a parameter) k leaves, G_k (see Figure 3.2): For every $k \in \mathbb{N}_{>0}$:

- $V(G_k) = \{u\} \cup \{v_1, \ldots, v_k\}$
- $E(G_k) = \bigcup_{i\in[k]}\{\{u, v_i\}\}$
- $Z(G_k) = \{(u, 1)\} \bigcup_{i\in[k]}\{(v_i, 1)\}$

Let N be an m-layer GNN. We define $u_k^{(t)} \coloneqq N^{(t)}(G_k, u)$, the feature of $u \in V(G_k)$ after operating the first t layers of N. Note that $u_k^{(m)} = N(G_k, u)$.

Lemma 3.3.1. *Assume N is a* Mean*-GNN or a* Max*-GNN . Let the maximum input dimension of any layer be d, and let the maximum Lipschitz-Constant of any MLP of N be a. Then, for every k it holds that $\left|u_k^{(m)}\right| \leq (da)^m$.*

Proof. For every $i, j \in [k]$ there is an automorphism of G_k that maps v_i to v_j, thus they receive the same feature throughout the computation. We define $v_k^{(t)} \coloneqq N^{(t)}(G_k, v_i)$ for every $i \in [k]$. We view $u_k^{(t)}, v_k^{(t)}$ as functions of k. First, assume N is a Mean-GNN . We show by induction that for any $t \in [m]$ it holds that $\left|v_k^{(t)}\right| \leq (2da)^t, \left|u_k^{(t)}\right| \leq (2da)^t$. For $t = 1$, $v_k^{(t)} = f_1(1, 1)$ for some MLP f_1 whose Lipschitz-Constant is at most a, hence $\left|f_1(1, \frac{1}{1})\right| \leq 2a$. Also, $u_k^{(t)} = f_1(1, \frac{k}{k}) = f_1(1, 1) \leq 2a$. Assume correctness for $t = n$. For $t = n + 1$ we have $v_k^{(n+1)} = f_{n+1}(v_k^{(n)}, u_k^{(n)})$ for some MLP f_{n+1} whose Lipschitz-Constant is at most a. Hence, $v_k^{(n+1)} \leq 2da(2da)^n = (2da)^{n+1}$. Also, $u_k^{(n+1)} = f_{n+1}(u_k^{(n)}, \frac{kv_k^{(n)}}{k}) = f_{n+1}(u_k^{(n)}, v_k^{(n)}) \leq 2da(2da)^n = (2da)^{n+1}$.

Next, assume N is a Max-GNN . Notice that for every $t \in [0..(m-1)]$ it holds that $\frac{u_k^{(t)}}{1} = \mathsf{max}(u_k^{(t)})$ and $\frac{kv_k^{(t)}}{k} = \mathsf{max}(v_k^{(t)}, \ldots, v_k^{(t)})$. Hence, the proof idea for a Mean-GNN applies also for a Max-GNN . □

Theorem 3.3.2. *Let $f : \mathcal{G}_1 \to \mathcal{Z}_{\mathbb{R}}$ a feature transformation such that for every k it holds that $f(G_k)(u) = k$. Then,* Mean*-GNNs $\not\approx f$ and* Max*-GNNs $\not\approx f$.*

Proof. Choose any $\varepsilon > 0$. Let N be either Mean-GNN or Max-GNN . Let the maximum input dimension of any layer be d, and let the maximum Lipschitz-Constant of any MLP of N be a. Choose $k = (2da)^m + \varepsilon$, then by Lemma 3.3.1 we have that $\left|N(G_k, u) - f(G_k)(u)\right| \geq \varepsilon$. □

Note that by Theorem 3.3.2, a function such as neighbors-count is inexpressible by Mean-GNNs and Max-GNNs .

Corollary 3.3.3. *We have that* Mean-*GNNs* $\not\geq^{\{1\}}$ Sum-*GNNs,* Max-*GNNs* $\not\geq^{\{1\}}$ Sum-*GNNs.*

Proof. Clearly, there is a Sum-GNN that computes f exactly. By Theorem 3.3.2, there is no Mean-GNN or Max-GNN that approximates f. □

3.3.2 Mean and Max Do Not Subsume Each Other

Mean-GNNs and Max-GNNs do not subsume each other, even in a finite initial-feature domain setting. We define a parameterized graph in which, depending on the parameters' arguments, the average of the center's neighbors is in $[0, \frac{1}{2}]$ while their maximum can be either 0 or 1. For every $k \in \mathbb{N}$ and $b \in \{0, 1\}$:

- $V(G_{k,b}) = \{u\} \cup \{v_1, \ldots, v_k\} \cup \{w\}$
- $E(G_{k,b}) = \bigcup_{i\in[k]}\{\{u, v_i\}\} \cup \{\{u, w\}\}$
- $Z(G_{k,b}) = \{(u, 0)\}\bigcup_{i\in[k]}\{(v_i, 0)\} \cup \{(w, b)\}$

Theorem 3.3.4. *Let* $f : \mathcal{G}_{\{0,1\}} \to \mathcal{Z}_{\mathbb{R}}$ *a feature transformation such that for every* k *it holds that* $f(G_{k,b})(u) = \frac{b}{k+1}$. *Then,* Max-*GNNs* $\not\approx f$.

Proof. Let N be an m-layer Max-GNNs. It is not difficult to see that by induction on m for every $i > 0, j > 0$ it holds that $N(G_{i,1}, u) = N(G_{j,1}, u)$, in particular $\forall k\ N(G_{1,1}, u) = N(G_{k,1}, u)$. Then, $N(G_{1,1}, u) \leq 0.25 \Rightarrow \left|f(G_{1,1})(u) - N(G_{1,1}, u)\right| > 0.24$ and $N(G_{1,1}, u) > 0.25 \Rightarrow \left|f(G_{100,1})(u) - N(G_{100,1}, u)\right| > 0.24$. That is, in both complementary cases there exists a graph for which the network's error is greater than 0.24. □

Theorem 3.3.5. *Let* $f : \mathcal{G}_{\{0,1\}} \to \mathcal{Z}_{\mathbb{R}}$ *a feature transformation such that for every* k *it holds that* $f(G_{k,b})(u) = b$. *Then,* Mean-*GNNs* $\not\approx f$.

Proof. Let N be an m-layer Mean-GNNs. It is not difficult to see that by induction on m, due to the Lipschitz-Continuity of MLPs and the aggregation being mean, we have that $N(G_{k,b}, u)$ is Lipschitz-Continuous with respect to $u's$ initial aggregation input i.e. $\frac{b}{k+1}$. That is, $\exists a \in \mathbb{R} : \forall k_1, k_2, b_1, b_2\ \left|N(G_{k_1,b_1}, u) - N(G_{k_2,b_2}, u)\right| \leq a\left|\frac{b_1}{k_1+1} - \frac{b_2}{k_2+1}\right|$. In particular, $\forall k, b_1, b_2\ \left|N(G_{k,b_1}, u) - N(G_{k,b_2}, u)\right| \leq a\left|\frac{b_1}{k+1} - \frac{b_2}{k+1}\right|$. Hence $\lim_{k\to\infty}\left|N(G_{k,0}, u) - N(G_{k,1}, u)\right| \leq \lim_{k\to\infty} a\left|\frac{b_1}{k+1} - \frac{b_2}{k+1}\right| = 0$. Let $\varepsilon > 0$ and let k, which exists by the above, such that $\left|N(G_{k,0}, u) - N(G_{k,1}, u)\right| < \varepsilon$ then necessarily either $\left|N(G_{k,0}, u) - f(G_{k,0})\right| \geq (0.5 - \varepsilon)$ or $\left|N(G_{k,1}, u) - f(G_{k,1})\right| \geq (0.5 - \varepsilon)$. □

Corollary 3.3.6. *We have that* Mean-*GNNs* $\not\geq^{\{0,1\}}$ Max-*GNNs* , Max-*GNNs* $\not\geq^{\{0,1\}}$ Mean-*GNNs* .

Proof. Clearly, there is a Mean-GNN that computes f of Theorem 3.3.4 exactly, and by Theorem 3.3.4 there is no Max-GNN that approximates f. Clearly, there is a Max-GNN that computes f of Theorem 3.3.5 exactly, and by Theorem 3.3.5 there is no Mean-GNN that approximates f. □

3.4 Sometimes Sum Subsumes

In a bounded initial-feature domain setting, Sum-GNNs can express every function that Mean-GNNs and Max-GNNs can. The bounded initial-feature domain means a bounded range when using mean or max aggregation, a fact which can be exploited to approximate the target GNN with a Sum-GNN. The approximating Sum-GNNs that we construct come at a size cost. While not affecting the asymptotic size with respect to the graph size, it is significant with respect to the approximated-network's size, hence meaningful in practice. We leave the question whether a more efficient construction exists as an open problem.

Every reference in Lemma 3.4.1 (and its proof) to a vertex-related value-vector is element-wise: for every vertex v and a value-function $f(v)$ of output dimension d we use the notation $f(v)$ to represent $f(v)_i$ for all $i \in [d]$.

Lemma 3.4.1. *Let $d \in \mathbb{N}_{>0}$, let $s \in [0,1]$, and let $0 < a \leq s$. Then, there is a* Sum*-GNN N such that for every featured graph $G \in \mathcal{G}_{[0,1]^d}$ and every vertex $v \in V(G)$ it holds that*

$$N(G,v) = \begin{cases} 0 & s \leq \mathsf{mean}(v) \\ \frac{n_v(s-\mathsf{mean}(v))}{a} & s - \frac{a}{n_v} < \mathsf{mean}(v) < s \\ 1 & s - a \leq \mathsf{mean}(v) \leq s - \frac{a}{n_v} \\ 1 - \frac{n_v(s-\mathsf{mean}(v)-a)}{a} & s - a - \frac{a}{n_v} < \\ & \quad \mathsf{mean}(v) < s - a \\ 0 & \mathsf{mean}(v) \leq s - a - \frac{a}{n_v} \end{cases}$$

Proof. Please refer to Figure 3.3 for an illustration of the construction. Let $v^{(t)}$ be the value of a vertex v after layer t and let $g_v^{(t)} = \mathrm{sum}_{w \in N(v)} w^{(t)}$ the sum of $v's$ neighbors' values after layer t. We denote the function computed in layer t of N by f_t, that is, $v^{(t)} = f_t(v^{(t-1)}, g_v^{(t-1)})$. First, we map the value of a vertex (and the sum of its neighbors) to a 2-tuple with the first coordinate being 1 and the second being the vertex' value. That is, we define $f_1 : \mathbb{R}^2 \to \mathbb{R}^2$ to be $f_1(x,y) = (1,x)$. Then, we define $f_2 : \mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{R}$ to be $f_2(x,y) = \mathrm{relu}(\frac{sy_1 - y_2}{a}) - \mathrm{relu}(\frac{sy_1-y_2}{a} - 1) + \mathrm{relu}(\frac{sy_1-y_2}{a} - n_v - 1) - \mathrm{relu}(\frac{sy_1-y_2}{a} - n_v)$. That is, $v^{(2)} = \mathrm{relu}(\frac{n_v(s-\mathsf{mean}(v))}{a}) - \mathrm{relu}(\frac{n_v(s-\mathsf{mean}(v))}{a} - 1) + \mathrm{relu}(\frac{n_v(s-\mathsf{mean}(v))}{a} - n_v - 1) - \mathrm{relu}(\frac{n_v(s-\mathsf{mean}(v))}{a} - n_v)$. To see why $v^{(2)}$ fulfills the requirements, we describe the values of each of the three components for the different ranges of $\mathsf{mean}(v)$.

- $s \leq \mathsf{mean}(v) \Rightarrow \frac{n_v(s-\mathsf{mean}(v))}{a} \leq 0 \Rightarrow \mathrm{relu}(\frac{n_v(s-\mathsf{mean}(v))}{a}) = \mathrm{relu}(\frac{n_v(s-\mathsf{mean}(v))}{a} - 1) = \mathrm{relu}(\frac{n_v(s-\mathsf{mean}(v))}{a} - n_v) = \mathrm{relu}(\frac{n_v(s-\mathsf{mean}(v))}{a} - n_v) = 0 \Rightarrow v^{(2)} = 0$
- $s - \frac{a}{n_v} < \mathsf{mean}(v) < s \Rightarrow 0 < \frac{n_v(s-\mathsf{mean}(v))}{a} < 1 \Rightarrow \mathrm{relu}(\frac{n_v(s-\mathsf{mean}(v))}{a}) - \mathrm{relu}(\frac{n_v(s-\mathsf{mean}(v))}{a} - 1) = \frac{n_v(s-\mathsf{mean}(v))}{a}$; $\mathrm{relu}(\frac{n_v(s-\mathsf{mean}(v))}{a} - n_v - 1) = \mathrm{relu}(\frac{n_v(s-\mathsf{mean}(v))}{a} - n_v) = 0 \Rightarrow v^{(2)} = \frac{n_v(s-\mathsf{mean}(v))}{a}$
- $s - a \leq \mathsf{mean}(v) \leq s - \frac{a}{n_v} \Rightarrow 1 \leq \frac{n_v(s-\mathsf{mean}(v))}{a} \leq n_v \Rightarrow \mathrm{relu}(\frac{n_v(s-\mathsf{mean}(v))}{a}) - \mathrm{relu}(\frac{n_v(s-\mathsf{mean}(v))}{a} - 1) = 1$; $\mathrm{relu}(\frac{n_v(s-\mathsf{mean}(v))}{a} - n_v - 1) = \mathrm{relu}(\frac{n_v(s-\mathsf{mean}(v))}{a} - n_v) = 0 \Rightarrow v_v^{(2)} = 1$

- $s - a - \frac{a}{n_v} < \mathsf{mean}(v) < s - a \Rightarrow n_v < \frac{n_v(s-\mathsf{mean}(v))}{a} < n_v + 1 \Rightarrow \mathrm{relu}(\frac{n_v(s-\mathsf{mean}(v))}{a}) - \mathrm{relu}(\frac{n_v(s-\mathsf{mean}(v))}{a} - 1) = 1$; $\mathrm{relu}(\frac{n_v(s-\mathsf{mean}(v))}{a} - n_v - 1) = 0$; $\mathrm{relu}(\frac{n_v(s-\mathsf{mean}(v))}{a} - n_v) = \frac{n_v(s-\mathsf{mean}(v)-a)}{a} \Rightarrow h_v^{(2)} = 1 - \frac{n_v(s-\mathsf{mean}(v)-a)}{a}$
- $\mathsf{mean}(v) \leq s - a - \frac{a}{n_v} \Rightarrow n_v + 1 \leq \frac{n_v(s-\mathsf{mean}(v))}{a} \Rightarrow \mathrm{relu}(\frac{n_v(s-\mathsf{mean}(v))}{a}) - \mathrm{relu}(\frac{n_v(s-\mathsf{mean}(v))}{a} - 1) = 1; \mathrm{relu}(\frac{n_v(s-\mathsf{mean}(v))}{a} - n_v - 1) + \mathrm{relu}(\frac{n_v(s-\mathsf{mean}(v))}{a} - n_v) = 1 \Rightarrow h_v^{(2)} = 0$

□

3.4.1 Mean by Sum

Sum-GNNs subsume Mean-GNNs in a bounded initial-feature domain setting.

Lemma 3.4.2. *For every $\varepsilon > 0$ and $d \in \mathbb{N}_{>0}$, there exists a* Sum*-GNN N of size $O(d\frac{1}{\varepsilon})$ such that for every featured graph $G \in \mathcal{G}_{[0,1]\subset\mathbb{R}^d}$ it holds that $\forall v \in V(G)\ \left|N(G,v) - \mathsf{mean}(v)\right| \leq \varepsilon$.*

Proof. Please refer to Figure 3.4 for an illustration of the construction. We describe a construction of size $O(\frac{1}{\varepsilon})$ which approximates mean for one coordinate, the extension to d is by a simple duplication. Every reference to a vertex-related value-vector is element-wise: for every vertex v and a value-function $f(v)$ of output dimension d, we use the notation $f(v)$ to represent $f(v)_i$ for all $i \in [d]$.

Let $q \in \mathbb{N}_{>0}$ be the minimal natural such that $\frac{1}{q} < \varepsilon$, and define $a = \frac{1}{q}$. Define $\{s_1 = a, s_2 = 2a, \ldots, s_{q+1} = 1 + a)\}$. The first layer of N is identical to f_1 in the Lemma 3.4.1. The second layer uses a copy of f_2 from the Lemma 3.4.1, for each s_i, multiplied by s_i, and then sums the $q + 1$ outputs. To see why this is correct, assume $s_i - a \leq \mathsf{mean}(v) \leq s_i$. For $j < i$ or $j > i + 1$ we have by Lemma 3.4.1 zero contribution of s_j to the final sum. Next, if $s_i - \frac{a}{n_v} \leq \mathsf{mean}(v)$ then by Lemma 3.4.1 we have a contribution of

$$s_{i+1}\Big(1 - \frac{n_v(s_{i+1} - \mathsf{mean}(v) - a)}{a}\Big) + s_i\Big(\frac{n_v(s_i - \mathsf{mean}(v))}{a}\Big) =$$

$$s_{i+1}\Big(1 - \frac{n_v(s_i - \mathsf{mean}(v))}{a}\Big) + s_i\Big(\frac{n_v(s_i - \mathsf{mean}(v))}{a}\Big)$$

Denoting the last term by x and considering that $s_i - \frac{a}{n_v} \leq \mathsf{mean}(v) \leq s_i$ we have that $\mathsf{mean}(v) \leq x \leq \mathsf{mean}(v) + a$. Finally, if $\mathsf{mean}(v) \leq s - \frac{a}{n_v}$ then by Lemma 3.4.1 we have zero contribution of s_{i+1} and a contribution of $s_i \leq \mathsf{mean}(v) + a$. Overall, we have that $\mathsf{mean}(v) \leq N(G,v) \leq \mathsf{mean}(v) + a$. □

Theorem 3.4.3. *Let N_M be a* Mean*-GNN consisting of m layers, let the maximum input dimension of any layer be d, and let the maximum Lipschitz-Constant of any MLP of N_M be a. Then, for every $\varepsilon > 0$ there exists a* Sum*-GNN N_S such that:*

1. $\forall G \in \mathcal{G}_{[0,1]^d}\ \forall v \in V(G) \quad |N_M(G,v) - N_S(G,v)| \leq \varepsilon$.
2. $\left|N_S\right| \leq O(\left|N_M\right| + \frac{d \cdot m \cdot ad(1-(2ad)^m)}{\varepsilon(1-(2ad))})$.

Proof. Let $N_M = ((f_1, \mathsf{mean}), \ldots, (f_m, \mathsf{mean}))$, that is, $f_1, \ldots, f_m$ are the MLPs constituting N_M's layers. Let $\widehat{\varepsilon} > 0$ and let $N_{\widehat{\varepsilon}} = ((g_1, \mathrm{sum}), (g_2, \mathrm{sum}))$ the GNN constructed in Lemma 3.4.2, with parameter $\widehat{\varepsilon}$. Note that g_1 is indifferent to the aggregation parameter and g_2 is indifferent to the vertex's state parameter, thus, for both parameters an argument of '0' is as good as any other. Define a Sum-GNN with $2m$ layers $N_S = ((\widehat{f}_1, \mathrm{sum}), \ldots, (\widehat{f}_{2m}, \mathrm{sum}))$. For $j = 0 \ldots (m-1)$, each pair of layers $(\widehat{f}_{2j+1}, \mathrm{sum}), (\widehat{f}_{2(j+1)}, \mathrm{sum})$ approximates the operation of (f_{j+1}, mean). For a graph G and a vertex $v \in V(G)$, denote the feature of v after the $(2(j+1))^{th}$ layer of N_S by $\widehat{v}^{(2(j+1))}$, with $\widehat{v}^{(0)} \coloneqq Z(G)(v)$. We define $(\widehat{f}_{2j+1}, \widehat{f}_{2(j+1)})$ as follows.

$$\widehat{f}_{2j+1}(\widehat{v}^{(2j)}, \Sigma_{w \in N(v)} \widehat{w}^{(2j)}) \coloneqq (\widehat{v}^{(2j)}, g_1(\widehat{v}^{(2j)}, 0))$$

$$\widehat{f}_{2(j+1)}((\widehat{v}^{(2j)}, g_1(\widehat{v}^{(2j)}, 0)), \Sigma_{w \in N(v)}(\widehat{w}^{(2j)}, g_1(\widehat{w}^{(2j)}, 0)) \coloneqq$$

$$f_{j+1}(\widehat{v}^{(2j)}, g_2(0, \Sigma_{w \in N(v)} g_1(\widehat{w}^{(2j)}, 0)))$$

For $t \in [m]$ denote the feature of v after the t^{th} layer of N_M by $v^{(t)}$, with $v^{(0)} \coloneqq Z(G)(v)$, and denote by $e_t \coloneqq \left|\widehat{v}_i^{(2t)} - v_i^{(t)}\right|$ the maximum error of any coordinate of the output of the $(2t)^{th}$ layer of N_S. We prove by induction on t that $e_t \leq ad\widehat{\varepsilon}\Sigma_{i \in [t]}(2ad)^{i-1}$. Denote that upper bound by b_t. For $t = 1$, we have

$$e_1 = \left|\widehat{v}^{(2)} - v^{(1)}\right| = f_1(\widehat{v}^{(0)}, g_2(0, \Sigma_{w \in N(v)} g_1(\widehat{w}^{(0)}, 0))) -$$

$$f_1(v^{(0)}, \mathsf{mean}(\{w^{(0)} | w \in N(v)\})|$$

The first d input coordinates to f_1 are identical. For each coordinate i of the last d coordinates, by definition of g_1 and g_2 we have

$$\left|g_2(0, \Sigma_{w \in N(v)} g_1(\widehat{w}^{(0)}, 0))_i - \mathsf{mean}(\{w^{(0)} : w \in N(v)\})_i\right| \leq \widehat{\varepsilon}$$

That difference translates to a difference of at most $a\widehat{\varepsilon}$ in any coordinate of $\left|\widehat{v}^{(2)} - v^{(1)}\right|$. In total, we have $e_1 \leq ad\widehat{\varepsilon}$. Assume correctness for $t = n$. Layer $2(n+1)$ of N_S is, by definition, the operation of f_{n+1} on at most $2 \cdot d$ coordinates. The first d coordinates constitute $\widehat{v}^{(2n)}$ and the last d coordinates constitute $g_2(0, \Sigma_{w \in N(v)} g_1(\widehat{w}^{(2n)}, 0))$. The error of each of the first d coordinates is, by assumption, at most b_n. For each coordinate i of the last d coordinates, we have by assumption

$$\forall w \in N(v) \ \left|\widehat{w}_i^{(2n)} - w_i^{(n)}\right| \leq b_n$$

hence

$$\left|\frac{1}{|N(v)|}\Sigma_{w \in N(v)} \widehat{w}_i^{(2n)} - \frac{1}{|N(v)|}\Sigma_{w \in N(v)} w_i^{(n)}\right| \leq b_n \tag{3.4.1}$$

hence, by definition of g_1 and g_2,

$$\left|g_2(0, \Sigma_{w \in N(v)} g_1(\widehat{w}^{(2n)}, 0))_i - \frac{1}{|N(v)|}\Sigma_{w \in N(v)} w_i^{(n)}\right| \leq b_n + \widehat{\varepsilon} \tag{3.4.2}$$

Combining the error bounds for the two types of input, we have that

$$e_{n+1} = \mathsf{max}(\left|\widehat{v}_i^{(2(n+1))} - v_i^{(n+1)}\right| : i \in [d]) =$$
$$\mathsf{max}(|f_{n+1}(\widehat{v}^{(2n)}, g_2(0, \Sigma_{w\in N(v)} g_1(\widehat{w}^{(2n)}, 0)))_i -$$
$$f_{n+1}(v^{(n)}, \mathsf{mean}(\{w^{(n)} : w \in N(v)\}))_i| : i \in [d]) \leq$$
$$adb_n + ad(b_n + \widehat{\varepsilon}) = 2adb_n + ad\widehat{\varepsilon} =$$
$$ad\widehat{\varepsilon}\Sigma_{i=2}^{n+1}(2ad)^{i-1} + ad\widehat{\varepsilon} =$$
$$ad\widehat{\varepsilon}\Sigma_{i\in[n+1]}(2ad)^{i-1}$$

With the induction proven, we have that

$$b_m = ad\widehat{\varepsilon}\Sigma_{i\in[m]}(2ad)^{i-1} = \widehat{\varepsilon}ad\frac{(1-(2ad)^m)}{1-2ad}$$

Hence, the requirement that $b_m \leq \varepsilon$ can be satisfied by setting

$$\widehat{\varepsilon} = \varepsilon\frac{1-2ad}{ad(1-(2ad)^m)}$$

implying

$$\frac{1}{\widehat{\varepsilon}} = \frac{ad(1-(2ad)^m)}{\varepsilon(1-2ad)}$$

Finally, using Lemma 3.4.2 we have that for each $i \in [m]$ it holds

$$\left|\widehat{f}_{2i-1}\right| + \left|\widehat{f}_{2i}\right| = O\Big(\frac{d \cdot ad(1-(2ad)^m)}{\varepsilon(1-2ad)}\Big) + \left|f_i\right|$$

hence

$$|N_S| = |N_M| + O\Big(\frac{m \cdot d \cdot ad(1-(2ad)^m)}{\varepsilon(1-2ad)}\Big)$$

□

Corollary 3.4.4. Sum-*GNNs* $\geq^{[0,1]}$ Mean-*GNNs.*

3.4.2 Max by Sum

Sum-GNNs subsume Max-GNNs in a bounded initial-feature domain setting.

Lemma 3.4.5. *Let $q \in \mathbb{N}_{>0}$, define $a \coloneqq \frac{1}{q}$, and define a function $f : [0,1] \to \mathbb{R}^q$ such that*

$$f(x)_i \coloneqq \mathsf{max}(0, min(x - a(i-1), a))$$

That is, f is an almost-unary representation of x in units of $\frac{1}{q}$, "almsot" because it may contain a fraction (between 0 and 1) in its last coordinate. For a finite multiset $x = \{x_1, \ldots, x_n\}, x_i \in [0,1]$, define

$$g(x) \coloneqq min((a, \ldots, a), \Sigma_{i\in[n]} f(x_i))$$

a mapping from the multiset to the sum of its elements' representation, coordinate-wise capped at a. Then,

$$\mathsf{max}(x) \leq \Sigma_{i\in[q]} g(x)_i \leq \mathsf{max}(x) + a$$

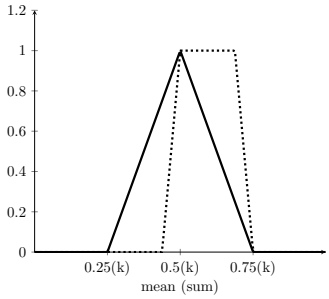


Figure 3.3: A single "position indicator", as constructed in Lemma 3.4.1, for interval $a = 0.25$ and position $s = 0.75$. The full line is for $n_v^- = 1$ and the dotted line is for $n_v^- = 4$. The x-axis represents $\mathsf{mean}(v)$ in the domain $[0, 1]$ and the corresponding $\mathrm{sum}(v)$ in the domain $[0, k]$.

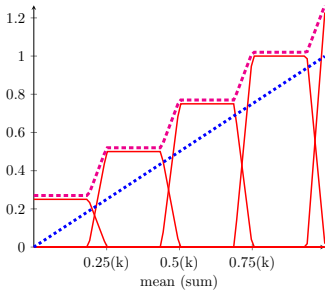


Figure 3.4: An illustration of the construction in Lemma 3.4.2, for $a = 0.25, n_v^- = 4$. The solid red trapezoids are indicators scaled according to the position they are indicating. The dashed magenta steps-line is the sum of the indicators, which is the final function. The dotted blue line is the line to approximate. The x-axis represents $\mathsf{mean}(v)$ in the domain $[0, 1]$ and the corresponding $\mathrm{sum}(v)$ in the domain $[0, k]$.

Proof. w.l.o.g assume $\mathsf{max}(x) = x_1$. For the lower bound, it is not hard to verify that $\forall i \in [q]\ g(x)_i \geq f(x_1)_i$, hence $\Sigma_{i\in[q]} g(x)_i \geq \Sigma_{i\in[q]} f(x_1)_i = x_1$. For the upper bound, assume $j = \mathsf{max}(i : g(x)_i > 0)$, then necessarily $x_1 \geq (j-1)a$ and $\Sigma_{i\in[q]} g(x)_i \leq ja$, hence $\Sigma_{i\in[q]} g(x)_i \leq x_1 + a$. □

Lemma 3.4.6. *For every $\varepsilon > 0$ and $d \in \mathbb{N}_{>0}$, there exists a* Sum*-GNN N of size $O(d\frac{1}{\varepsilon})$ such that for every featured graph $G \in \mathcal{G}_{[0,1]^d}$ and vertex $v \in V(G)$ it holds that $\left|N(G, v) - \mathsf{max}(v)\right| \leq \varepsilon$.*

Proof. We describe a construction of size $O(\frac{1}{\varepsilon})$ that approximates max for one coordinate, the extension to d is by a simple duplication. Every reference to a vertex-related value-vector is element-wise: for every vertex v and a value-function $f(v)$ of output dimension d, we use the notation $f(v)$ to represent $f(v)_i$ for all $i \in [d]$.

Let $q \in \mathbb{N}_{>0}$ be the minimal natural such that $\frac{1}{q} < \varepsilon$ and define $a \coloneqq \frac{1}{q}$. The first GNN layer computes for each vertex v a vector $v^{(1)} \in [0, a]^q$ such that $(v^{(1)})_i = \mathrm{relu}(Z(v) - (i-1)a) - \mathrm{relu}(Z(v) - (i-1)a - a)$. Observe that the computation corresponds to the mapping f in Lemma 3.4.5. The second GNN layer first caps the sum-aggregation of the neighbors' vectors, then sums the coordinates of the capped vector. That is, for a vertex v, let $y_v = \Sigma_{w\in N(v)} w^{(1)}$, then $v^{(2)} = \Sigma_{i\in[q]}(\mathrm{relu}((y_v)_i) - \mathrm{relu}((y_v)_i - a))$. Using Lemma 3.4.5, we get that $\mathsf{max}(v) \leq v^{(2)} \leq \mathsf{max}(v) + a < \mathsf{max}(v) + \varepsilon$. □

Theorem 3.4.7. *Let N_M be a* Max*-GNN consisting of m layers, let the maximum input dimension of any layer be d, and let the maximum Lipschitz-Constant of any MLP of N_M be a. Then, for every $\varepsilon > 0$ there exists a* Sum*-GNN N_S such that:*

1. $\forall G \in \mathcal{G}_{[0,1]^d}\ \forall v \in V(G) \quad |N_M(G, v) - N_S(G, v)| \leq \varepsilon$.
2. $\left|N_S\right| \leq O(\left|N_M\right| + \frac{d \cdot m \cdot ad(1-(2ad)^m)}{\varepsilon(1-(2ad))})$.

Proof. The proof is identical to the Theorem 3.4.3 with the following adaptations:

1. Replacing any mention of 'mean', with 'max'.
2. Replacing any usage of Lemma 3.4.2, with Lemma 3.4.6.
3. Replacing equations (1),(2), with equations (3),(4) hereinafter.

$$\begin{aligned}\Big|\mathsf{max}(\widehat{w}_i^{(2n)} : w \in N(v)) - \\ \mathsf{max}(w_i^{(n)} : w \in N(v))\Big| \leq b_n\end{aligned} \tag{3.4.3}$$

$$\begin{aligned}\Big|g_2(0, \Sigma_{w \in N(v)} g_1(\widehat{w}^{(2n)}, 0))_i - \\ \mathsf{max}(w_i^{(n)} : w \in N(v))\Big| \leq b_n + \widehat{\varepsilon}\end{aligned} \tag{3.4.4}$$

□

Corollary 3.4.8. Sum-*GNNs* $\geq^{[0,1]}$ Max-*GNNs.*

3.5 Mean and Max Have Their Place

In two important settings, mean and max aggregations enable expressing functions that cannot be expressed with sum alone. As in Section 3.3, we define a graph G_θ parameterized by θ over domain Θ. We define a feature transformation f on that graph and prove that it cannot be approximated by Sum-GNNs. The line of proof is as follows:

1. We show that for every Sum-GNN N there exists a finite set P_N of polynomials in θ, those polynomials have a certain property φ, and it holds that:

   $$\forall \theta \in \Theta \ \exists u_\theta \in V(G_\theta) \ \exists p \in P_N : \ N(G_\theta, u_\theta) = p(\theta)$$

   That is, for every choice of θ there is a polynomial in P_N that assumes the same value as the network, for a certain vertex in G_θ.

2. We show that for every finite set P of polynomials (in θ) that obtain φ, it holds that:

   $$\forall \varepsilon > 0 \ \exists \theta \in \Theta : \ \forall p \in P \ \big|p(\theta) - f(G_\theta)(u_\theta)\big| > \varepsilon$$

   That is, for every gap we choose, there is a graph such that for the certain vertex from (1) it holds that all the polynomials of P miss the target function by at least that gap. Combined with (1), we have that N also misses the target function by at least that gap.

Describability

Let P be a set of polynomials in k, c, and let $g(k, c)$ be a function in k, c.

We say that P *weakly-describes* g if and only if:

a. P is finite.

b. $\forall k, c \in \mathbb{N} \;\; \exists p \in P \;:\; p(k, c) = g(k, c)$. That is, for every k, c there is a polynomial in P that assumes the same value as g.

The "weakness" is in view of our goal: We would like P to contain only polynomials of a specific kind, and the above properties do not include that requirement.

We identify a polynomial $p(k, c)$ as being *good* (again, in view of our goal) if and only if $p(k, c) = \Sigma_{i\in[n],j\in[n]} a_{i,j} k^i c^j + \Sigma_{i=0}^{n} b_i k^i$ for some real coefficients $\{a_{i,j}\}, \{b_i\}$ and some maximum degree $n \in \mathbb{N}$. That is, $p(k, c)$ is a polynomial in k, c with maximum degrees n for k, c, and every appearance of c is with multiplication by a polynomials in k of degree at least 1. We say that P is *good* if and only if every polynomial in it is good. That "goodness" is the property we referred to in the proof outline above.

We say that P *describes* g if and only if: P weakly-describes g and P is good. In the following paragraphs, we may combine statements' versions for both describability and weak-describability by mentioning the weak version in parenthesis. We say that g is *describable* (*w-describable*) if and only if there exists a set that (weakly-) describes it.

Let P be a finite set of polynomials in k, c, we denote by $\mathcal{B}(P) \coloneqq \{k^i c^j \;:\; \exists p \in P \quad p = (... + a_{i,j} k^i c^j) \quad a_{i,j} \neq 0\}$ the building blocks of P, that is, the degree combinations that appear in any of the polynomials in P. Let $b \in \{k, c\}$, we define $b\mathcal{B}(P) \coloneqq \{b \cdot k^i c^j : k^i c^j \in \mathcal{B}(P)\}$.

For every $a \in \mathbb{R}$ and a set of functions F of k, c, we define $aF \coloneqq \{af \;:\; f \in F\}$, and $(a + F) \coloneqq \{a + f \;:\; f \in F\}$. For two sets of functions F, H of k, c, we define $F + H \coloneqq \{f + h : f \in F, h \in H\}$.

Lemma 3.5.1. *a. Let $f(k, c)$ be a function (w-)describable by a set P. Let $g(k, c) :=$* relu$(f(k, c))$ *the composition of* relu *over $f(k, c)$, then g is (w-)describable by a set P' such that $\mathcal{B}(P') \subseteq (\mathcal{B}(P) \cup \{k^0 c^0\})$.*

b. Let $f_1(k, c), \ldots, f_l(k, c)$ be functions (w-)describable by $P_1, \ldots, P_l$ respectively. Then, for every real coefficients $\{a_i\}, b$ the affine function $(\Sigma_{i=1}^{n} a_i f_i) + b$ is (w-)describable by a set P such that $\mathcal{B}(P) \subseteq (\{k^0 c^0\} \bigcup_{i\in[l]} \mathcal{B}(P_i))$.

c. Each output of a relu *activated MLP, whose inputs are all (w-)describable by a set P, is (w-)describable by a set P' such that $\mathcal{B}(P') \subseteq (\mathcal{B}(P) \cup \{k^0 c^0\})$.*

d. Let $f(k, c)$ be a function w-describable by a set P, then $kf(k, c)$ is describable by some set P' such that $\mathcal{B}(P') \subseteq k\mathcal{B}(P)$, and $cf(k, c)$ is w-describable by a set P'' such that $\mathcal{B}(P'') \subseteq c\mathcal{B}(P)$.

Proof. a. Let P be a set that (w-)describes the function f. For any k, c either $g(k, c) = f(k, c)$ or $g(k, c) = 0$, hence relu(f) is (w-)describable by $P \cup \{0\}$.
b. It is not hard to verify that if f_i is (w-)describable by P_i then for every $a \in \mathbb{R}$ it holds that af_i is (w-)describable by aP_i, and $f_i + a$ is (w-)describable by $P_i + a$. It is also not hard to verify then that for any $a_i, a_j \in \mathbb{R}$ it holds that $a_i f_i + a_j f_j$ is (w-)describable by

$(a_iP_i)+(a_jP_j)$. A straightforward induction proves that a linear combination of arbitrarily many (w-)describable functions is (w-)describable. Finally, let P a set that (w-)describes the linear combination, then $P + b$ is a set that (w-)describes the affine function.
c. Implied by (a)+(b).
d. It is not hard to verify that if f is w-describable by P then kf is describable by kP. Also, it is not hard to verify that if f is w-describable by P then cf is w-describable by cP. □

Lemma 3.5.2. *Let $\{H_{k,c}\}$ be a set of graphs, parametrized in $k, c \in \mathbb{N}_{>0}$, each having an identified vertex u, such that for every m-layer* Sum-*GNN N it holds that the function $(k, c) \mapsto N(H_{k,c}, u)$ i.e. $N(H_{k,c}, u)$ viewed as a function of $k; c$, is describable. Then, for every* Sum-*GNN N and for every $\varepsilon > 0$ there exist k, c s.t $\left|N(H_{k,c}, u) - c\right| > \varepsilon$.*

Proof. Let P be a finite set of polynomials that describes $N(H_{k,c}, u)$. Fix any specific $c \in \mathbb{N}_{>0}$, and for $K \in \mathbb{N}_{>0}$ denote by $P_{K,c} = \{p \in P : \exists k \geq K : N(H_{k,c}, u) = p(k, c)\}$ only those polynomials in P that intersect with $u^{(m)}_{k,c}$ in the domain $[K, \infty) \times \{c\}$. Denote the polynomials in $P_{K,c}$ that are a constant, by $\widehat{P}_{K,c} = \{p : p \in P_{K,c},\ p \text{ constant}\}$. Let $\varepsilon > 0$ and assume by contradiction that for every $k \in \mathbb{N}_{>0}$ it holds that $\left|N(H_{k,c}, u) - c\right| \leq \varepsilon$. Then, there must exist $K_c \in \mathbb{N}_{>0}$ for which $\widehat{P}_{K_c,c} = P_{K_c,c}$. Otherwise, as P is assumed to describe $N(H_{k,c}, u)$, any appearance of c, in any $p \in P_{k,c}$, is tied to k, and we would have

$$\inf_{p \in (P_{k,c} \setminus \widehat{P}_{k,c})} \left|p(k, c)\right| \xrightarrow[k \to \infty]{} \infty$$

and

$$\sup_{k \in \mathbb{N}} \left|N(H_{k,c}, u) - c\right| = \infty$$

in contradiction to $\left|N(H_{k,c}, u) - c\right| \leq \varepsilon$. By definition, $\widehat{P}_{K,c}$ is a subset of P which is finite, and so $\mathsf{max}(\widehat{P}_{K_c,c}) \leq \mathsf{max}(p \in P : p \text{ constant})$. Denote the last term by max_P. As our reasoning thus far is true for any c, it holds that $\mathsf{max}(\mathsf{max}(\widehat{P}_{K_c,c}) : c \in \mathbb{N}) \leq \mathsf{max}_P$. Finally, for $c = \lceil \mathsf{max}_P + \varepsilon + 1 \rceil$ necessarily for all $k \geq K_c$ it holds that $\left|N(H_{k,c}, u) - c\right| > c - \mathsf{max}_P > \varepsilon$. □

3.5.1 Unbounded, Countable, initial-feature Domain

In an unbounded initial-feature domain setting, Mean; Max; and other GNNs are not subsumed by Sum-GNNs. We define a graph $G_{k,c}$ (see Figure 3.2): For $(k, c) \in \mathbb{N}^2_{>0}$,

- $V(G_{k,c}) = \{u\} \cup \{v_1, \ldots, v_k\}$
- $E(G_{k,c}) = \bigcup_{i \in [k]} \{\{u, v_i\}\}$
- $Z(G_{k,c}) = \{(u, 0)\} \bigcup_{i \in [k]} \{(v_i, c)\}$

Let N be an m-layer Sum-GNN. We define $u^{(t)}_{k,c} \coloneqq N^{(t)}(G_{k,c}, u)$, the feature of $u \in V(G_{k,c})$ after operating the first t layers of N. Note that $u^{(m)}_{k,c} = N(G_{k,c}, u)$. For every $i, j \in [k]$ there is an automorphism of G_k that maps v_i to v_j, thus they receive the same feature throughout the computation. We define $v^{(t)}_{k,c} \coloneqq N^{(t)}(G_{k,c}, v_i)$ for every $i \in [k]$. In our argumentation, we view $u^{(t)}_{k,c}, v^{(t)}_{k,c}$ as functions of k, c.

Lemma 3.5.3. *It holds that $u_{k,c}^{(m)}$ is describable.*

Proof. We show by induction that for every $t \in [m]$ it holds that $v_{k,c}^{(t)}$ is w-describable and that $u_{k,c}^{(t)}$ is describable. For $t = 0$ we have $u_{k,c}^{(t)} = 0, v_{k,c}^{(t)} = c$ and the assumption holds. Assume correctness for $t = n$. By definition, $u_{k,c}^{(n+1)} = f_{n+1}(u_{k,c}^{(n)}, kv_{k,c}^{(n)})$ where f_{n+1} is a relu MLP. By assumption, $v_{k,c}^{(n)}$ is w-describable and so by Lemma 3.5.1 we have that $kv_{k,c}^{(n)}$ is describable. Also, by assumption, $u_{k,c}^{(n)}$ is describable. Hence, by Lemma 3.5.1 we have that $u_{k,c}^{(n+1)}$ is describable. The proof for $v_{k,c}^{(n+1)}$ is in similar fashion. □

Theorem 3.5.4. *Let $f : \mathcal{G}_{\mathbb{N}^1} \to \mathcal{Z}_{\mathbb{R}}$ a feature transformation, such that for every k, c it holds that $f(G_{k,c})(u) = c$. Then,* Sum-*GNNs $\not\approx f$.*

Proof. Immediate from combining Lemma 3.5.3 and Lemma 3.5.2. □

Corollary 3.5.5. *Denote by S the set of all multisets over $\mathbb{N}_{>0}$. Let $g : S \to \mathbb{R}$ be an aggregation such that $\forall a, b \in \mathbb{N}_{>0}\ g(\binom{\{a\}}{b}) = a$, that is, g aggregates every homogeneous multiset to its single unique value. Then,* Sum-*GNNs $\not\geq^{\mathbb{N}}$ g-aggregation GNNs.*

Proof. Let $f : \mathcal{G}_{\mathbb{N}^1} \to \mathcal{Z}_{\mathbb{R}}$ a feature transformation, such that for every featured graph G, and for every vertex $v \in V(G)$, it holds that $f(G)(v) \coloneqq g(N(v))$. Then, by Theorem 3.5.4, Sum-GNNs $\not\approx f$. Clearly, there is a g-aggregation GNN that computes f exactly. □

Corollary 3.5.5 implies a limitation of Sum-GNNs compared to GNNs that use mean; max; or many other aggregations. One may wonder whether the separation exists only with unbounded functions i.e. regression functions, or whether there are also classification functions i.e. with finite output range, that separate the different architectures. In the following, we prove that it is the second case.

Classification

Let $G_{k,c,d}$, for $k, c, d \in \mathbb{N}_{>0}$, be a parameterized graph such that:

- $V(G_{k,c,d}) = \{u_1, \ldots, u_k\} \cup \{v_1, \ldots, v_k\}$
- $E(G_{k,c,d}) = \bigcup_{i\in[k],j\in[k]}\{\{u_i, v_j\}\}$
- $Z(G_{k,c,d}) = \bigcup_{i\in[k]}\{(u_i, d)\} \bigcup_{i\in[k]}\{(v_i, c)\}$

Define $\forall G \in \mathcal{G}_{\mathbb{N}^1}\ \forall x \in V(G)$

$$f(G, x) \coloneqq \begin{cases} 1 & Z(G)(x) > \frac{\Sigma_{w\in N_G(x)} Z(G)(w)}{|N_G(x)|} \\ 0 & \text{otherwise} \end{cases}$$

That is, f maps a node to 1 if its initial feature is greater than the average of its neighbors' initial features, and to 0 otherwise. In particular, $f(G_{k,c,d})(u_i) \coloneqq \mathrm{lsig}(d - c)$. Clearly, Mean-GNNs express f. Let $N = ((F_1, \mathrm{sum}), \ldots, (F_m, \mathrm{sum}))$ be an m-layers Sum-GNN. Define $u_{k,c,d}^{(t)} \coloneqq N^{(t)}(G_{k,c,d,}, u_i)$ the value of N for any of the u_is after applying the first

t layers, and similarly by $v^{(t)}_{k,c,d}$ for any of the v_is. For $k, c, d \in \mathbb{N}_{>0}$, and a polynomial $p(k, c, d)$, we say that p is *great* if and only if

$$p(k, c, d) = \Sigma_{i=0}^{m} k^{2i}(a_i d + q_i) + \Sigma_{i=0}^{m} k^{2i+1}(b_i c + r_i)$$

for some sets of real coefficients $\{a_i\}, \{b_i\}, \{q_i\}, \{r_i\}$. We say it is *great-complement*, notated $\overline{\text{great}}$, if and only if

$$p(k, c, d) = \Sigma_{i=0}^{m} k^{2i+1}(a_i d + q_i) + \Sigma_{i=0}^{m} k^{2i}(b_i c + r_i)$$

Importantly, in both cases c and d multiply different degrees of k. Note that for a great p we have that kp is $\overline{\text{great}}$, and for a $\overline{\text{great}}$ p we have that kp is great.

We claim that N cannot classify the u_is according to the order between c, d. The line of proof is as follows:

1. We show that when c, d are above a certain threshold and then k is above a certain threshold that depends on c, d, the value of the u_is is a great polynomial. This is shown in lemmas 3.5.6 till 3.5.8.

2. We show that great polynomials have the property that there are values of c,d,and k, as large as we wish, such that they do not indicate the order between the values of c and d. That is, for every great polynomial $p(k, c, d)$ it holds that for every $T > 0$ there exist $T < c_0 < d_0 < c_1 < d_1$ such that: For every $K_{min} > 0$ there exist $k_0, k_1, k_2 > K_{min}$ such that

   $$\big(p(k_0, c_0, d_0) \leq p(k_1, c_1, d_0) \leq p(k_2, c_1, d_1)\big) \vee \big(p(k_0, c_0, d_0) \geq p(k_1, c_1, d_0) \geq p(k_2, c_1, d_1)\big)$$

   This is shown in Lemma 3.5.9.

3. Combining (1) and (2) we have that the u_is cannot assume the correct value for all c, d and k.

Lemma 3.5.6. *Let $p(k, c, d)$ be a great or $\overline{\text{great}}$ polynomial , then*

$$\exists s \in \{-1, 1\} \exists T : \forall c, d > T \ \exists K_{c,d} : \forall k > K_{c,d} \ \text{sign}(p(k, c, d)) = s$$

That is, no matter the specific c, d as long as they are above a certain threshold, beyond a certain value of k (which depends on c, d) the sign of p is always the same.

Proof. We prove for a great p, the proof for $\overline{\text{great}}$ is similar. Let

$$p(k, c, d) = \Sigma_{i=0}^{m} k^{2i}(a_i d + q_i) + \Sigma_{i=0}^{m} k^{2i+1}(b_i c + r_i)$$

Define

$$j \coloneqq \begin{cases} \mathsf{max}(i : a_i \neq 0 \vee q_i \neq 0) & 2\mathsf{max}(i : a_i \neq 0 \vee q_i \neq 0) > 2\mathsf{max}(i : b_i \neq 0 \vee r_i \neq 0) + 1 \\ \mathsf{max}(i : b_i \neq 0 \vee r_i \neq 0) & \text{otherwise} \end{cases}$$

That is, j is the index of the coefficients that multiply the highest degree of k, which by definition of *great* can be only one of the two kinds of coefficient pairs. We prove for

$j = \mathsf{max}(i : a_i \neq 0 \vee q_i \neq 0)$, the proof for the complementing case is similar. If $a_j \neq 0$, define $T \coloneqq \left\lceil \left| \frac{q_j}{a_j} \right| \right\rceil + 1$, then it is not difficult to see that

$$\forall d > T \; \forall c \; \exists K_{c,d} : \forall k > K_{c,d} \; \text{sign}(p(k,c,d)) = \text{sign}(a_j)$$

If $a_j = 0$, then it is not difficult to see that

$$\forall c, \forall d \; \exists K_{c,d} : \forall k > K_{c,d} \; \text{sign}(p(k,c,d)) = \text{sign}(q_j)$$

□

Lemma 3.5.7. *Let F be an MLP with inputs $x_1, \ldots, x_l$ and outputs $y_1, \ldots, y_q$ such that $\forall i \in [l]$ x_i is a great ($\overline{\text{great}}$) polynomial. Then, for every $i \in [q]$ there exists a great ($\overline{\text{great}}$) polynomial p_i such that*

$$\exists T : \forall c, d > T \; \exists K_{c,d} : \forall k > K_{c,d} \; F(k,c,d)(i) = p_i(k,c,d)$$

That is, no matter the specific c, d as long as they are above a certain threshold, beyond a certain value of k (which depends on c, d) the value of $F(k,c,d)(i)$ is the value of a certain great ($\overline{\text{great}}$) polynomial. Note that there may be a different thresholds and great ($\overline{\text{great}}$) polynomials corresponding to different output dimensions of F.

Proof. We prove for great input polynomials, the proof for $\overline{\text{great}}$ inputs is similar. Let $p(k,c,d)$ be a great polynomial, then, observe the following:

1. Let p' be another great polynomial, let $a, b, q \in \mathbb{R}$, and let $p'' = ap + bp' + q$. Then, p'' is a great polynomial.
   proof: Straightforward from polynomials arithmetic.

2. Define $g(k,c,d) \coloneqq \text{relu}(p(k,c,d))$, then,

   $$\exists h(k,c,d) \in \{0, p(k,c,d)\} \exists T : \forall c, d > T \; \exists K_{c,d} : \forall k > K_{c,d} \; g(k,c,d)) = h(k,c,d)$$

   proof: By Lemma 3.5.6, for every $c, d > T$, there is some $K_{c,d}$ above which p is either always above zero - in which case $g = p$, or always ≤ 0 - in which case $g = 0$.

Combining observations (1),(2), a straightforward induction on the structure of F proves the claim - defining T to be the maximum over the Ts of each of F's neurons, then for every $c, d > T$ defining $K_{c,d}$ to be the maximum over the $K_{c,d}$s of each of F's neurons. □

Lemma 3.5.8. *There exists a great polynomial p such that*

$$\exists T : \forall c, d > T \; \exists K_{c,d} : \forall k > K_{c,d} \; u^{(m)}_{k,c,d} = p(k,c,d)$$

That is, no matter the specific c, d as long as they are above a certain threshold, beyond a certain value of k (which depends on c, d) the value of $u^{(m)}_{k,c,d}$ is always the same great polynomial.

Proof. For every $t \in [m]$ denote by d_t the output dimension of F_t i.e. $\dim(u^{(t)}_{k,c,d}) = \dim(v^{(t)}_{k,c,d}) = d_t$. We prove by induction on t that for every $t \in [m]$ and for every $i \in [d_t]$:

1. There exists a great polynomial p_i such that

$$\exists T : \forall c, d > T \; \exists K_{c,d} : \forall k > K_{c,d} \; u^{(t)}_{k,c,d}(i) = p_i(k, c, d)$$

2. There exists a $\overline{\text{great}}$ polynomial p'_i such that

$$\exists T : \forall c, d > T \; \exists K_{c,d} : \forall k > K_{c,d} \; v^{(t)}_{k,c,d}(i) = p'_i(k, c, d)$$

For $t = 0$ we have $u^{(0)}_{k,c,d} = d = k^0 d$, $v^{(0)}_{k,c,d} = c = k^0 c$. Assume correctness for $t = n$ we prove for $t = n + 1$:

$$u^{(n+1)}_{k,c,d}(i) = F_{n+1}(u^{(n)}_{k,c,d}, kv^{(n)}_{k,c,d})(i)$$

By the induction assumption, all dimensions of $u^{(n)}_{k,c,d}$ are great polynomials and so are those of $kv^{(n)}_{k,c,d}$, hence the claim for $u^{(n+1)}_{k,c,d}(i)$ follows from Lemma 3.5.7.

$$v^{(n+1)}_{k,c,d}(i) = F_{n+1}(v^{(n)}_{k,c,d}, ku^{(n)}_{k,c,d})(i)$$

By the induction assumption, all dimensions of $v^{(n)}_{k,c,d}$ are $\overline{\text{great}}$ and so are those of $ku^{(n)}_{k,c,d}$, hence the claim for $v^{(n+1)}_{k,c,d}(i)$ follows from Lemma 3.5.7. □

Lemma 3.5.9. *Let $p(k, c, d)$ be a great polynomial, then for every $T > 0$ there exist $T < c_0 < d_0 < c_1 < d_1$ such that: For every $K_{min} > 0$ there exist $k_0, k_1, k_2 > K_{min}$ such that*

$$\big(p(k_0, c_0, d_0) \le p(k_1, c_1, d_0) \le p(k_2, c_1, d_1)\big) \vee \big(p(k_0, c_0, d_0) \ge p(k_1, c_1, d_0) \ge p(k_2, c_1, d_1)\big)$$

Proof. Let $p(k, c, d) = \Sigma_{i=0}^{m} k^{2i}(a_i d + q_i) + \Sigma_{i=0}^{m} k^{2i+1}(b_i c + r_i)$. Define

$$j \coloneqq \begin{cases} \mathsf{max}(i : a_i \neq 0 \vee q_i \neq 0) & 2\mathsf{max}(i : a_i \neq 0 \vee q_i \neq 0) > 2\mathsf{max}(i : b_i \neq 0 \vee r_i \neq 0) + 1 \\ \mathsf{max}(i : b_i \neq 0 \vee r_i \neq 0) & \text{otherwise} \end{cases}$$

We prove for $j = \mathsf{max}(i : a_i \neq 0 \vee q_i \neq 0)$, the proof for the complementing case is similar. If $a_j \neq 0$ define $d_0 \coloneqq \mathsf{max}(T + 2, \left\lceil \left| \frac{q_j}{a_j} \right| \right\rceil + 2), c_0 = d_0 - 1, c_1 = d_0 + 1, d_1 = c_1 + 1$, else define $d_0 = T + 2, c_0 = d_0 - 1, c_1 = d_0 + 1, d_1 = c_1 + 1$. Then,

1) If $j = 0$ then define $k_0 = k_1 = k_2 = K_{min} + 1$. It is not difficult to see that either $p(k_0, c_0, d_0) \le p(k_1, c_1, d_0) \le p(k_2, c_1, d_1)$ or $p(k_0, c_0, d_0) \ge p(k_1, c_1, d_0) \ge p(k_2, c_1, d_1)$.

2) If $j > 0$ then by the definition of j and d_0, d_1, we have that $\lim_{k\to\infty} p(k, c_0, d_0) = \lim_{k\to\infty} p(k, c_1, d_0) = \lim_{k\to\infty} p(k, c_1, d_1) \in \{-\infty, \infty\}$. Hence, there exist k_0, k_1, k_2 such that $k_i > K_{min}$ and either $p(k_0, c_0, d_0) \le p(k_1, c_1, d_0) \le p(k_2, c_1, d_1)$ or $p(k_0, c_0, d_0) \ge p(k_1, c_1, d_0) \ge p(k_2, c_1, d_1)$. □

Theorem 3.5.10. *There exist k_0, k_1, k_2 and $c_0 < d_0 < c_1 < d_1$, such that*

$$\big(u^{(m)}_{k_0,c_0,d_0} \le u^{(m)}_{k_1,c_1,d_0} \le u^{(m)}_{k_2,c_1,d_1}\big) \vee \big(u^{(m)}_{k_0,c_0,d_0} \ge u^{(m)}_{k_1,c_1,d_0} \ge u^{(m)}_{k_2,c_1,d_1}\big)$$

Proof. Realize $k_0, k_1, k_2, c_0, d_0, c_1, d_1$ as follows:

1. Let $p(k, c, d)$ and T, which exist by Lemma 3.5.8, such that

$$\forall c, d > T \; \exists K_{c,d} : \forall k > K_{c,d} \; u^{(m)}_{k,c,d} = p(k, c, d)$$

2. Let $c_0, d_0, c_1, d_1 > T$, which exist by Lemma 3.5.9, such that: For every $K_{min} > 0$ there exist $k_0, k_1, k_2 > K_{min}$ such that

$$\big(p(k_0, c_0, d_0) \leq p(k_1, c_1, d_0) \leq p(k_2, c_1, d_1)\big) \vee \big(p(k_0, c_0, d_0) \geq p(k_1, c_1, d_0) \geq p(k_2, c_1, d_1)\big)$$

3. Let $K_{c_0,d_0}, K_{c_1,d_0}, K_{c_1,d_1}$, which exist by Lemma 3.5.8, such that

$$\forall k > K_{c_i,d_j} \; u^{(m)}_{k,c_i,d_j} = p(k, c_i, d_j)$$

4. Define $K_{min} \coloneqq \mathsf{max}(K_{c_0,d_0}, K_{c_1,d_0}, K_{c_1,d_1})$ and let $k_0, k_1, k_2 > K_{min}$, which exist by Lemma 3.5.9, such that

$$\big(p(k_0, c_0, d_0) \leq p(k_1, c_1, d_0) \leq p(k_2, c_1, d_1)\big) \vee \big(p(k_0, c_0, d_0) \geq p(k_1, c_1, d_0) \geq p(k_2, c_1, d_1)\big)$$

Then, by Lemma 3.5.8 we have

$$u^{(m)}_{k_0,c_0,d_0} = p(k_0, c_0, d_0), u^{(m)}_{k_1,c_1,d_0} = p(k_1, c_1, d_0), u^{(m)}_{k_2,c_1,d_1} = p(k_2, c_1, d_1)$$

, by Lemma 3.5.9 we have

$$\big(p(k_0, c_0, d_0) \leq p(k_1, c_1, d_0) \leq p(k_2, c_1, d_1)\big) \vee \big(p(k_0, c_0, d_0) \geq p(k_1, c_1, d_0) \geq p(k_2, c_1, d_1)\big)$$

, and the combination of the two implies the theorem. □

Corollary 3.5.11. *Sum-GNNs cannot classify the u_is according to whether their initial feature is greater than that of the v_is, while Mean-GNNs can.*

Proof. By Theorem 3.5.10 we have that Sum-GNNs $\not\approx f$. It is not difficult to see that for any relu-activated architecture, it can express f if it can express a separating function $g(G, x) \leq a \Leftrightarrow f(G, x) = 0, \; g(G, x) \geq b \Leftrightarrow f(G, x) = 1, \; a, b \in \mathbb{R}, a < b$. Hence, Sum-GNNs cannot express the required classification. □

Graph Embedding

Sum-GNNs are limited compared to Mean; Max; and other GNNs, not only when used to approximate vertices' feature transformations but also when used in combination with a readout function to approximate graph embeddings. Consider another variant of $G_{k,c}$: For $(k, c) \in \mathbb{N}^2_{>0}$,

- $V(G_{k,c}) = \{u_1, \ldots, u_{k^2}\} \cup \{v_1, \ldots, v_k\}$
- $E(G_{k,c}) = \bigcup_{i \in [k^2], j \in [k]} \{\{u_i, v_j\}\}$
- $Z(G_{k,c}) = \bigcup_{i \in [k^2]} \{(u_i, 0)\} \bigcup_{i \in [k]} \{(v_i, c)\}$

Let N be an m-layer Sum-GNN. We use the notations $u_{k,c}^{(t)}$ and $v_{k,c}^{(t)}$ with similar meaning to before, where $u_{k,c}^{(t)}$ now refers to each of the u_i vertices.

Lemma 3.5.12. *It holds that $k^2u_{k,c}^{(m)} + kv_{k,c}^{(m)}$ is describable by a set P such that for every $p \in P$ it holds that p does not contain k^2c (with coefficient $\neq 0$).*

Proof. We prove the correctness of the following statements for every $t \in [m]$, from which the lemma clearly follows.

1. $u_{k,c}^{(t)}$ is describable.

2. $v_{k,c}^{(t)}$ is weakly-describable by a set P such that for every $p \in P$ it holds that p does not contain kc.

Proof is by induction on t. Correctness for $t = 0$ is clear. Assume correctness for $t = n$.

1. By definition, $u_{k,c}^{(n+1)} = f_{n+1}(u_{k,c}^{(n)}, kv_{k,c}^{(n)})$ for some MLP f_{n+1}. By the induction assumption, $u_{k,c}^{(n)}$ is describable and clearly $kv_{k,c}^{(n)}$ is also describable. Hence, by Lemma 3.5.1 we have that $u_{k,c}^{(n+1)}$ is describable.

2. By definition, $v_{k,c}^{(n+1)} = f_{n+1}(v_{k,c}^{(n)}, k^2u_{k,c}^{(n)})$ for some MLP f_{n+1}. By the induction assumption, $v_{k,c}^{(n)}$ obtains the stated property, and clearly so does $k^2u_{k,c}^{(n)}$. By Lemma 3.5.1, we have that the output of operating f_{n+1} on $v_{k,c}^{(n)}, k^2u_{k,c}^{(n)}$ obtains the stated property. □

Lemma 3.5.13. *Let $f : \mathcal{G}_{\mathbb{N}^1} \to \mathbb{R}$ a graph embedding such that $\forall k, c\ f(G_{k,c}) = \frac{kc}{k+1}$. Let F be an MLP, and define a readout $\mathfrak{ro} \coloneqq f_F \circ \mathsf{mean}$. Then, $\mathfrak{ro} \circ$ Sum-GNNs $\not\approx f$.*

Proof. Let N be a Sum-GNN . By definition, $\mathsf{mean} \circ N(G_{k,c}) = \frac{k^2 \cdot u_{k,c}^{(m)} + k \cdot v_{k,c}^{(m)}}{k(k+1)} = \frac{k \cdot u_{k,c}^{(m)} + v_{k,c}^{(m)}}{(k+1)}$. By Lemma 3.5.12, $k \cdot u_{k,c}^{(m)} + v_{k,c}^{(m)}$ is weakly-describable by a set P' such that for every $p \in P'$ it holds that p does not contain kc. Using a similar technique to the one in proof of Lemma 3.5.1, it is not hard to show that $f_F \circ \mathsf{mean} \circ N(G_{k,c})$ is weakly-describable by a set P such that for every $p \in P$ it holds that p does not contain kc. Let $p \in P$ be any polynomial and let $b \in \mathbb{R}$ be the coefficient of k in p. It is not hard to verify that for every c it holds that $\lim_{k\to\infty} \left|\frac{p(k,c)}{k+1}\right| \in \{0, |b|, \infty\}$. The finiteness of P implies that there is a maximal such $|b|$ over all $p \in P$, denote it by b_{max}. The finiteness of P also implies that:

1. Given c and $\delta > 0$ there exists K_0 such that for every $l > K_0$ and every $p \in P$ with a finite limit (as $k \to \infty$) it holds that $\left|\frac{p(l,c)}{l+1} - \lim_{k\to\infty} \frac{p(k,c)}{k+1}\right| < \delta$.

2. Given c and $\delta > 0$ there exists K_0 such that for every $l > K_0$ and every $p \in P$ with an infinite limit (as $k \to \infty$) it holds that $\left|\frac{p(l,c)}{l+1} - c\right| > \delta$.

Finally, for every c it holds that $\lim_{k\to\infty} \frac{kc}{k+1} = c$. Let $\varepsilon > 0$, then for $c = \lceil 2\varepsilon + b_{max} \rceil$ there exists k such that for every $p \in P$ it holds that $\left|\frac{p(k,c)-kc}{k+1}\right| > \varepsilon$. □

Lemma 3.5.14. *Let $f : \mathcal{G}_{\mathbb{N}^1} \to \mathbb{R}$ a graph embedding such that $\forall k, c\ f(G_{k,c}) = \frac{kc}{k+1}$. Let F be an MLP , and define a readout $\mathfrak{ro} \coloneqq f_F \circ sum$. Then, $\mathfrak{ro} \circ$ Sum-GNNs $\not\approx f$.*

Proof. Let $\varepsilon > 0$, then $\text{sum} \circ N(G_{k,c}) = k^2 \cdot u_{k,c}^{(m)} + k \cdot v_{k,c}^{(m)}$. Clearly, $k^2 \cdot u_{k,c}^{(m)} + k \cdot v_{k,c}^{(m)}$ is describable. Hence, by Lemma 3.5.1, it holds that $f_F \circ \text{sum} \circ N(G_{k,c})$ is describable. Let P be a describing set of $k^2 \cdot u_{k,c}^{(m)} + k \cdot v_{k,c}^{(m)}$, let $p \in P$ be any polynomial, and let $b \in \mathbb{R}$ be the coefficient of k^0 in p. It is not hard to verify that for every c it holds that $\lim_{k\to\infty} \left|p(k,c)\right| \in \{0, |b|, \infty\}$. The finiteness of P implies that there is a maximal such $|b|$ over all $p \in P$, denote it by b_{max}. The finiteness of P also implies that:

1. Given c and $\delta > 0$ there exists K_0 such that for every $l > K_0$ and every $p \in P$ with a finite limit (as $k \to \infty$) it holds that $p(l,c) - \lim_{k\to\infty} p(k,c) < \delta$.
2. Given c and $\delta > 0$ there exists K_0 such that for every $l > K_0$ and every $p \in P$ with an infinite limit (as $k \to \infty$) it holds that $\left|p(l,c) - c\right| > \delta$.

Finally, for every c it holds that $\lim_{k\to\infty} \frac{kc}{k+1} = c$. Let $\varepsilon > 0$, then for $c = \lceil 2\varepsilon + b_{max} \rceil$ there exists k such that for every $p \in P$ it holds that $\left|\frac{p(k,c)-kc}{k+1}\right| > \varepsilon$. Let $\varepsilon > 0$, then for $c = \lceil 2\varepsilon + b_{max} \rceil$ there exists k such that for every $p \in P$ it holds that $\left|p(k,c) - \frac{kc}{k+1}\right| > \varepsilon$. □

Theorem 3.5.15. *Let $f : \mathcal{G}_{\mathbb{N}^1} \to \mathbb{R}$ a graph embedding such that $\forall k, c\ f(G_{k,c}) = \frac{kc}{k+1}$. Let $\mathfrak{a} \in \{sum, \mathsf{mean}\}$ be an aggregation and let F be an MLP, and define a readout $\mathfrak{ro} \coloneqq f_F \circ \mathfrak{a}$. Then, $\mathfrak{ro} \circ$ Sum-GNNs $\not\approx f$.*

Proof. Follows from combining Lemma 3.5.13 and Lemma 3.5.14. □

Corollary 3.5.16. *Denote by S the set of all multisets over $\mathbb{N}_{>0}$. Let $g : S \to \mathbb{R}$ an aggregation such that $\forall a, b \in \mathbb{N}_{>0}\ g(\binom{\{a\}}{b}) = a$. Let $\mathfrak{a} \in \{sum, \mathsf{mean}\}$ be an aggregation and let F be an MLP, and define a readout $\mathfrak{ro} \coloneqq f_F \circ \mathfrak{a}$. Then, $\mathfrak{ro} \circ$ Sum-GNNs $\not\geq^{\mathbb{N}}$ mean $\circ$ g-GNNs.*

Proof. Clearly, for a straightforward g-aggregation GNN N_g it holds that $N_g(G_{k,c})(u_i) = c$ and $N_g(G_{k,c})(v_i) = 0$, hence $\mathsf{mean} \circ N_g(G_{k,c}) = \frac{k^2 c}{k^2+k} = \frac{kc}{k+1}$. By Theorem 3.5.15, no composition of $\mathfrak{ro}$ with a Sum-GNN can approximate $f(G) = N_g(G)$. □

We have shown that Sum-GNNs do not subsume Mean and Max (and many other) GNNs. The setting though, consisted of an initial-feature domain $\mathbb{N}_{>0}$, that is, countable unbounded.

3.5.2 Finite initial-feature Domain

mean and max aggregations are essential also when the initial-feature domain is just a single value i.e. when the input is featureless graphs. We define a new graph $G_{k,c}$ (see Figure 3.2): For every $(k,c) \in \mathbb{N}_{>0}^2$,

- $V(G_{k,c}) = \{u\} \cup \{v_1, \ldots, v_k\} \cup \{w_1, \ldots, w_c\}$
- $E(G_{k,c}) = \bigcup_{i\in[k]}\{\{u, v_i\}\} \bigcup_{i\in[k],j\in[c]}\{\{v_i, w_j\}\}$
- $Z(G_{k,c}) = \{(u,1)\} \bigcup_{i\in[k]}\{(v_i,1)\} \bigcup_{i\in[c]}\{(w_i,1)\}$

Let N be an m-layer Sum-GNN. We define $u_{k,c}^{(t)} \coloneqq N^{(t)}(G_{k,c}, u)$, $v_{k,c}^{(t)} \coloneqq N^{(t)}(G_{k,c}, v_i)$, and $w_{k,c}^{(t)} \coloneqq N^{(t)}(G_{k,c}, w_i)$, following a reasoning similar to Section 3.5.1, and view $u_{k,c}^{(t)}, v_{k,c}^{(t)}, w_{k,c}^{(t)}$ as functions of k, c

Lemma 3.5.17. *It holds that $u_{k,c}^{(m)}$ is describable.*

Proof. We show by induction that for every $t \in [m]$ it holds that $v_{k,c}^{(t)}$ is w-describable and that $u_{k,c}^{(t)}, w_{k,c}^{(t)}$ are describable. For $t = 0$ we have $u_{k,c}^{(t)} = v_{k,c}^{(t)} = w_{k,c}^{(t)} = 1$ and the assumption holds. Assume correctness for $t = n$. By definition, $u_{k,c}^{(n+1)} = f_{n+1}(u_{k,c}^{(n)}, kv_{k,c}^{(n)})$ where f_{n+1} is a relu MLP. By assumption, $v_{k,c}^{(n)}$ is w-describable and so by Lemma 3.5.1 we have that $kv_{k,c}^{(n)}$ is describable. Also by assumption, $u_{k,c}^{(n)}$ is describable. Hence, by Lemma 3.5.1 we have that $u_{k,c}^{(n+1)}$ is describable. For $v_{k,c}^{(n+1)}$, by definition, $v_{k,c}^{(n+1)} = f_{n+1}(v_{k,c}^{(n)}, cw_{k,c}^{(n)} + u_{k,c}^{(n)})$, and by assumption $u_{k,c}^{(n)}, v_{k,c}^{(n)}, w_{k,c}^{(n)}$ are w-describable. Hence, by Lemma 3.5.1 we have that $v_{k,c}^{(n+1)}$ is w-describable. The proof for $w_{k,c}^{(n+1)}$ is in similar fashion. □

Theorem 3.5.18. *Let $f : \mathcal{G}_1 \to \mathcal{Z}_{\mathbb{R}}$ a feature transformation, such that for every k, c it holds that $f(G_{k,c})(u) = c$. Then,* Sum-*GNNs $\not\approx f$.*

Proof. Immediate from combining Lemma 3.5.17 and Lemma 3.5.2. □

Corollary 3.5.19. *Denote by S the set of all multisets over $\mathbb{N}_{>0}$, and let $g : S \to \mathbb{R}$ an aggregation such that $\forall a, b \in \mathbb{N}_{>0}\ g(\binom{\{a\}}{b}) = a$. Then,* Sum-*GNNs $\not\geq^{\{1\}}$ (Sum, g)-GNNs.*

Proof. Let $f : \mathcal{G}_{\{1\}} \to \mathcal{Z}_{\mathbb{R}}$ a feature transformation such that for every featured graph G, for every vertex $v \in V(G)$, it holds that $f(G)(v) \coloneqq g(\{\mathrm{sum}(w) : w \in N(v)\})$. Then, by Theorem 3.5.18, Sum-GNNs $\not\approx f$. Clearly, there is a GNN that uses the sum-aggregation dimensions in its first layer and the g-aggregation dimensions in its second layer, that computes f exactly. □

Corollary 3.5.19 implies a limitation of Sum-GNNs compared to stereo aggregation GNNs that combine sum with mean; max; or many other aggregations. The limitation exists even when the initial-feature domain consists of only a single value.

Graph Embedding

Completing the no-subsumption picture, Sum-GNNs are not subsuming, in a 2-values initial-feature domain setting, also when used in combination with a readout function to approximate graph embeddings. We define $G_{k,c}$: For every $(k, c) \in \mathbb{N}_{>0}^2$,

- $V(G_{k,c}) = \{u_1, \ldots, u_{k^2}\} \cup \{v_1, \ldots, v_{k^3}\} \cup \{w_1, \ldots, w_{kc}\}$
- $E(G_{k,c}) = \bigcup_{j \in [k^2], i \in [k^3]} \{\{u_j, v_i\}\} \bigcup_{i \in [k^3], j \in [kc]} \{\{v_i, w_j\}\}$
- $Z(G_{k,c}) = \bigcup_{i \in [k^2]} \{(u_i, 0)\} \bigcup_{i \in [k^3]} \{(v_i, 0)\} \bigcup_{i \in [kc]} \{(w_i, 1)\}$

Let N be an m-layer Sum-GNN. The notations $u_{k,c}^{(t)}$, $v_{k,c}^{(t)}$, and $w_{k,c}^{(t)}$, are used as before.

Lemma 3.5.20. *It holds that $k^2 u_{k,c}^{(m)} + k^3 v_{k,c}^{(m)} + kcw_{k,c}^{(m)}$ is describable by a set P and for every $p \in P$ it holds that p does not contain $k^3 c$ (with coefficient $\neq 0$).*

Proof. We prove the correctness of the following statements, from which the lemma clearly follows.

1. $u_{k,c}^{(t)}$ is weakly-describable by a set P such that for every $p \in P$ it holds that p does not contain kc.

2. $v_{k,c}^{(t)}$ is describable.

3. $w_{k,c}^{(t)}$ is weakly-describable by a set P such that for every $p \in P$ it holds that p does not contain k^2.

Proof is by induction on t. Correctness for $t = 0$ is immediate. Assume correctness for $t = n$.

1. By definition, $u_{k,c}^{(n+1)} = f_{n+1}(u_{k,c}^{(n)}, k^3 v_{k,c}^{(n)})$ for some MLP f_{n+1}. By the induction assumption, $u_{k,c}^{(n)}$ obtains the stated property and the same holds for $k^3 v_{k,c}^{(n)}$. By Lemma 3.5.1, we have that the output of operating f_{n+1} on $u_{k,c}^{(n)}, k^3 v_{k,c}^{(n)}$ obtains the stated property.

2. By definition, $v_{k,c}^{(n+1)} = f_{n+1}(v_{k,c}^{(n)}, k^2 u_{k,c}^{(n)} + kcw_{k,c}^{(n)})$ for some MLP f_{n+1}. By the induction assumption, $v_{k,c}^{(n)}$ obtains the stated property, and clearly so do $k^2 u_{k,c}^{(n)}, kcw_{k,c}^{(n)}$. The rest follows similarly to the end of (1).

3. By definition, $w_{k,c}^{(n+1)} = f_{n+1}(w_{k,c}^{(n)}, k^3 v_{k,c}^{(n)})$ for some MLP f_{n+1}. By the induction assumption, $w_{k,c}^{(n)}$ obtains the stated property, and clearly so does $k^3 v_{k,c}^{(n)}$. The rest follows similarly to the end of (1). □

Lemma 3.5.21. *Let $f : \mathcal{G}_{\{0,1\}^1} \to \mathbb{R}$ a graph embedding such that $\forall k, c\ f(G_{k,c}) = \frac{(k^2+kc)kc}{k^3+k^2+kc}$. Let $\mathfrak{a} \in \{sum, \mathsf{mean}\}$ be an aggregation and let F be an MLP, and define a readout $\mathfrak{ro} := f_F \circ \mathfrak{a}$. Then, $\mathfrak{ro} \circ$ Sum-GNNs $\not\approx f$.*

Proof. Let $\varepsilon > 0$. Define $A := k^2 u_{k,c}^{(m)} + k^3 v_{k,c}^{(m)} + kcw_{k,c}^{(m)}$, then $\mathsf{mean} \circ N(G_{k,c}) = \frac{A}{k^3+k^2+kc}$. By Lemma 3.5.20, A is describable by a set P' such that for every $p \in P'$ it holds that p does not contain $k^3 c$, hence $\mathsf{mean} \circ N(G_{k,c})$ is describable. Hence, by Lemma 3.5.1 $f_F \circ \mathsf{mean} \circ N(G_{k,c})$ is describable. Let P be a describing set. Let $p \in P$ be any polynomial and let $b \in \mathbb{R}$ the coefficient of the component k^3 in p. Then, it is not hard to verify that for every c it holds that $\lim_{k\to\infty} \left|\frac{p(k,c)}{k^3+k^2+kc}\right| \in \{0, |b|, \infty\}$. The finiteness of P implies that there is a maximal such $|b|$ over all $p \in P$, denote it by b_{max}. The finiteness of P also implies that:

1. Given c and $\delta > 0$ there exists K_0 such that for every $l > K_0$ and every $p \in P$ with a finite limit (as $k \to \infty$) it holds that $\left|\frac{p(l,c)}{l^3+l^2+lc} - \lim_{k\to\infty} \frac{p(k,c)}{k^3+k^2+kc}\right| < \delta$.

2. Given c and $\delta > 0$ there exists K_0 such that for every $l > K_0$ and every $p \in P$ with an infinite limit (as $k \to \infty$) it holds that $\frac{p(l,c)}{l^3+l^2+lc} - c > \delta$.

Finally, for every c it holds that $\lim_{k\to\infty} \frac{(k^2+kc)kc}{k^3+k^2+kc} = c$. Hence, for $c = \lceil 2\varepsilon + b_{max} \rceil$ there exists k such that for every $p \in P$ it holds that $\left|\frac{p(k,c)-(k^2+kc)kc}{k^3+k^2+kc}\right| > \varepsilon$, implying $\left|\mathsf{mean} \circ N(G_{k,c}) - f(G_{k,c})\right| > \varepsilon$. □

Lemma 3.5.22. *Let $f : \mathcal{G}_{\mathbb{N}^1} \to \mathbb{R}$ a graph embedding, such that for every k, c it holds that $f(G_{k,c}) = \frac{(k^2+kc)kc}{k^3+k^2+kc}$. Then, sum$\circ$ Sum-GNNs $\not\approx f$.*

Proof. Let $\varepsilon > 0$. Clearly, $k^2 u_{k,c}^{(m)} + k^3 v_{k,c}^{(m)} + kcw_{k,c}^{(m)}$ is describable. Let P a describing set of $k^2 u_{k,c}^{(m)} + k^3 v_{k,c}^{(m)} + kcw_{k,c}^{(m)}$, let $p \in P$ be any polynomial, and let $b \in \mathbb{R}$ be the coefficient of k^0 in p. Then, it is not hard to verify that for every c it holds that $\lim_{k\to\infty} \left|p(k,c)\right| \in \{0, |b|, \infty\}$. The finiteness of P implies that there is a maximal such $|b|$ over all $p \in P$, denote it by b_{max}. The finiteness of P also implies that:

1. Given c and $\delta > 0$ there exists K_0 such that for every $l > K_0$ and every $p \in P$ with a finite limit (as $k \to \infty$) it holds that $\left|p(l,c) - \lim_{k\to\infty} p(k,c)\right| < \delta$.

2. Given c and $\delta > 0$ there exists K_0 such that for every $l > K_0$ and every $p \in P$ with an infinite limit (as $k \to \infty$) it holds that $\left|p(l,c) - c\right| > \delta$.

Finally, for every c it holds that $\lim_{k\to\infty} \frac{(k^2+kc)kc}{k^3+k^2+kc} = c$. Hence, for $c = \lceil 2\varepsilon + \mathsf{max}(0, b_{\mathsf{max}})\rceil$ there exists k such that for every $p \in P$ it holds that $\left|p(k,c) - \frac{(k^2+kc)kc}{k^3+k^2+kc}\right| > \varepsilon$, implying $\left|\text{sum } \circ N(G_{k,c}) - f(G_{k,c})\right| > \varepsilon$. □

Theorem 3.5.23. *Let $f : \mathcal{G}_{\{0,1\}^1} \to \mathbb{R}$ a graph embedding such that $\forall k, c\ f(G_{k,c}) = \frac{(k^2+kc)kc}{k^3+k^2+kc}$. Let $\mathfrak{a} \in \{sum, \mathsf{mean}\}$ be an aggregation and let F be an MLP, and define a readout $\mathfrak{ro} \coloneqq f_F \circ \mathfrak{a}$. Then, $\mathfrak{ro} \circ$ Sum-GNNs $\not\approx f$.*

Proof. Follows from combining Lemma 3.5.21 and Lemma 3.5.22. □

Corollary 3.5.24. *Denote by S the set of all multisets over $\mathbb{N}_{>0}$. Let $g : S \to \mathbb{R}$ an aggregation such that $\forall a, b \in \mathbb{N}_{>0}\ g(\binom{\{a\}}{b}) = a$. Let $\mathfrak{a} \in \{sum, \mathsf{mean}\}$ be an aggregation and let F be an MLP, and define a readout $\mathfrak{ro} \coloneqq f_F \circ \mathfrak{a}$. Then, $\mathfrak{ro} \circ$ Sum-GNNs $\not\geq^{\{0,1\}} \mathsf{mean} \circ$ (Sum, g)-GNNs.*

Proof. Clearly, for a straightforward stereo aggregation (Sum,g)-GNN N_g it holds that $N_g(G_{k,c})(u_i) = kc$, $N_g(G_{k,c})(v_i) = 0$, and $N_g(G_{k,c})(w_i) = kc$, hence $\mathsf{mean} \circ N_g(G_{k,c}) = \frac{(k^2+kc)kc}{k^3+k^2+kc}$. By Theorem 3.5.23, no composition of $\mathfrak{ro}$ with a Sum-GNN can approximate the graph embedding $f(G) \coloneqq \mathsf{mean} \circ N_g(G)$. □

3.6 Sum and More Are Not Enough

In previous sections we showed that Sum-GNNs do not subsume Mean-GNNsand Max-GNNs , by proving that they cannot express specific functions. In this section, rather than comparing different GNNs classes we focus on one broad GNNs class and show that it is limited in its ability to express any one of a certain range of functions.

Denote by S the set of all multisets over $\mathbb{R}$, and let $\mathfrak{a} : S \to \mathbb{R}$ be an aggregation. We say that $\mathfrak{a}$ is a *uniform polynomial aggregation* (UPA) if and only if for every homogeneous multiset $\binom{\{x\}}{b}, x \in \mathbb{R}, b \in \mathbb{N}_{>0}$ it holds that $\mathfrak{a}(\binom{\{x\}}{b})$ is either a polynomial in x or a polynomial in (bx). Note that sum; mean; and max are all UPAs. We say that a GNN $N = (L^{(1)}, \ldots, L^{(m)})$ is an MUPA-GNN (Multiple UPA) if and only if the aggregation input to each of its layers is defined by a sequence of UPAs. That is, $L^{(i)} = (F^{(i)}, (\mathfrak{a}_1^{(i)}, \ldots, \mathfrak{a}_{b_i}^{(i)}))$, for some MLP $F^{(i)}$ and b_i UPAs.

We define a parameterized graph G_k (see Figure 3.2): For every $k \in \mathbb{N}_{>0}$:

- $V(G_k) = \{u\} \cup \{v_1, \ldots, v_k\}$
- $E(G_k) = \bigcup_{i\in[k]}\{\{u, v_i\}\}$
- $Z(G_k) = \{(u, 1)\} \bigcup_{i\in[k]}\{(v_i, 1)\}$

Lemma 3.6.1. *Let A an m-layer* MUPA*-GNN architecture, let l be the maximum depth of any MLP in A, and let d be the maximum in-degree of any node in any MLP in A. Then, there exists $r \in \mathbb{N}$ such that: for every GNN N that is a model of A it holds that $N(G_k, u)$ is piecewise-polynomial (of k) with at most $((d+1)^l)^m$ pieces, and each piece is of degree at most r.*

Proof. Note the following observations:

a. Let f_1, f_2 be piecewise polynomial with p_1, p_2 pieces, then a linear combination of f_1, f_2 has at most $p_1 + p_2$ pieces. This can be seen by considering the set of pieces-joint points of $f_1 + f_2$, and noticing that it is the union of such points of f_1 and such points of f_2. Accordingly, let $f_1, \ldots, f_d$ be piecewise polynomial with at most p pieces each, then a linear combination of $f_1, \ldots, f_d$ has at most $p \cdot d$ pieces.

b. Let f be piecewise polynomial with at most p pieces, then $\mathrm{relu}(f)$ has at most $p+1$ pieces.

c. Let g be an output of a relu MLP of depth l with maximal in-degree d for any node, with inputs which are at most p-pieces polynomial each. Then, by (a)+(b), g is piecewise-polynomial with $(((pd + 1)d + 1)d + 1).. \leq p \cdot (d + 1)^l$ pieces.

d. Let $f(x)$ be piecewise polynomial with at most p pieces, and let $g(x)$ a polynomial, then $g(f(x))$ is piecewise polynomial, with at most p pieces, each of degree at most $deg(f)deg(g)$

e. Let $f(x)$ be piecewise polynomial with at most p pieces, and let $g(y)$ a polynomial, then $g(xf(x))$ is piecewise polynomial, with at most p pieces, each of degree at most $(deg(f) + 1)deg(g)$.

Let N be a a model of A. We define $u_k^{(t)} \coloneqq N^{(t)}(G_k, u)$, the feature of $u \in V(G_k)$ after operating the first t layers of N. Note that $u_k^{(m)} = N(G_k, u)$. For every $i, j \in [k]$ there is an automorphism of G_k that maps v_i to v_j, thus they receive the same feature throughout the computation. We define $v_k^{(t)} \coloneqq N^{(t)}(G_k, v_i)$ for every $i \in [k]$. In our argumentation, we view $u_k^{(t)}, v_k^{(t)}$ as functions of k.

Using observations [a..e] above, we prove by induction on t that $v_k^{(t)}, u_k^{(t)}$, in each coordinate, are piecewise polynomial in k with no more than $((d + 1)^l)^t$ pieces, each of degree at most r_t for some $r_t \in \mathbb{N}$. For $t = 0$ we have that $v_k^{(t)}, u_k^{(t)}$ are constants. Assume correctness for $t = n$. By definition, $u_k^{(n+1)} = f_{n+1}(u_k^{(n)}, \mathfrak{a}_1^{(n+1)}, \ldots, \mathfrak{a}_{b_{n+1}}^{(n+1)})$ where $\mathfrak{a}_j^{(n+1)}$ is a shorthand for the aggregation value $\mathfrak{a}_j^{(n+1)}(\{v_{k,c}^{(n)}\}^k)$. By (d),(e), and the induction assumption, each of the input coordinates to f_{n+1} is piecewise polynomial in k with at most $((d + 1)^l)^n$ pieces, each of degree at most r_{n+1} for some $r_{n+1} \in \mathbb{N}$. Hence, by (c), each coordinate of $u_{k,c}^{(n+1)}$ has at most $((d + 1)^l)^n \cdot (d + 1)^l = ((d + 1)^l)^{n+1}$ pieces, each of degree at most r_{n+1}. By similar reasoning, $v_k^{(n+1)}$ can be shown to have no more than $((d + 1)^l)^{n+1}$ pieces, each of a certain maximal degree. □

Lemma 3.6.1 implies that the architecture bounds (from above) the number of polynomial pieces, and their degrees, that make the function computed by any particular

model of the architecture. With Lemma 3.6.1 at our disposal, we consider any feature transformation that does not converge to a polynomial when applied to $u \in V(G_k)$ and viewed as a function of k. We show that such a function is inexpressible by MUPA-GNNs.

Theorem 3.6.2. *Let $f : \mathcal{G}_1 \to \mathcal{Z}_{\mathbb{R}}$ a feature transformation, and define $g(k) \coloneqq f(G_k)(u)$. Assume that g does not converge to any polynomial, that is, there exists $\varepsilon > 0$ such that for every polynomial p, for every K_0, there exists $k \geq K_0$ such that $\left|g(k) - p(k)\right| \geq \varepsilon$. Then,* MUPA-*GNNs*$\not\approx f$.

Proof. Let ε by which g does not get forever close to any polynomial, and let N be a MUPA-GNN . By Lemma 3.6.1, there is a K_0 such that for every $k \geq K_0$ it holds that $N(G_k, u) = p(k)$ for some polynomial p. By assumption, there exists $k > K_0$ such that $\left|g(k) - p(k)\right| \geq \varepsilon$. Hence, $\left|N(G_k, u) - f(G_k, u)\right| \geq \varepsilon$. □

The last inexpressivity property we prove, concerns a class of functions which we call *PIL* (Polynomial-Intersection Limited). For $n \in \mathbb{N}$ denote by P_n the set of all polynomials in degree $\leq n$. We say that a function $f : \mathbb{N} \to \mathbb{R}$ is PIL if and only if for every $n \in \mathbb{N}$ there exists $k_n \in \mathbb{N}$ such that for every polynomial $p \in P_n$ there exist at most $k_n - 1$ consecutive integer points on which p and f assume the same value. Formally,

$$\sup\left(k : \forall p \in P_n\ \forall x \in \mathbb{N}\ \forall y \in [x..(x+k-1)]\ f(y) = p(y)\right) \in \mathbb{N}$$

We consider every feature transformation f such that for $g(k) \coloneqq f(G_k)(u)$ it holds that g is PIL. This is a different characterization than "no polynomial-convergence" (in Theorem 3.6.2), and neither one implies the other. The result though, is weaker for the current characterization. We show that every MUPA-GNN architecture can approximate such a function only down to a certain $\varepsilon > 0$. That is, every model of the architecture - no matter the specific weights of its MLPs - is far from the function by at least ε (at least in one point). The following lemma is an adaptation of the Polynomial of Best Approximation theorem [May06; Gol62] which is a step in the proof of the Equioscillation theorem attributed to Chebyshev.

Lemma 3.6.3. *For $x, k \in \mathbb{N}$ define $I_{x,k} \coloneqq \{x, x+1, \ldots, x+k-1\}$ the set of consecutive k integers starting at x. Let $f : \mathbb{N} \to \mathbb{R}$ be a PIL, let $n \in \mathbb{N}$, and define $k_n \coloneqq$*

$$1 + \mathsf{max}(k : \forall p \in P_n\ \forall x \in \mathbb{N}\ \forall y \in [x..(x+k-1)]\ f(y) = p(y))$$

Then, for every $x \in \mathbb{N}$ there exists $\varepsilon_{x,k_n} > 0$ such that: for every $p \in P_n$ there exists $y \in I_{x,k_n}$ for which $\left|p(y) - f(y)\right| \geq \varepsilon_{x,k_n}$. That is, for every starting point x there is a bounded interval I_{x,k_n}, and a gap ε_{x,k_n}, such that no polynomial of degree $\leq n$ can approximate f on that interval below that gap.

Proof. Define $I \coloneqq I_{x,k_n}$. For a real-valued function h whose domain contains I, we define $\|h\|_I \coloneqq \mathsf{max}(\left|h(y)\right| : y \in I_{x,k_n})$, the maximum absolute value h attains on I_{x,k_n}. Define $\varepsilon_{x,k_n} \coloneqq \inf(\|f - p\|_I : p \in P_n)$, the distance of f from the closest polynomial of degree $\leq n$, in the segment I_{x,k_n}. We need to show that $\varepsilon_{x,k_n} > 0$. For a vector $a = (a_0, \ldots, a_n) \in \mathbb{R}^{n+1}$ denote by $\|a\|_2$ the Euclidean norm of a. For $a, b \in \mathbb{R}^{n+1}$ we use $d(a, b) \coloneqq \|a - b\|_2$ as the metric in our continuity argumentation. Define $p_a(x) \coloneqq a_0 + \cdots + a_n x^n$ the polynomial determined by a. Note the following:

a) For $a \in \mathbb{R}^{n+1}$, let $g(a) \coloneqq \|p_a\|_I$, then g is continuous.

b) For $a \in \mathbb{R}^{m+1}$, let $g(a) \coloneqq \|f - p_a\|_I$, then g is continuous.

c) There exists $T \in \mathbb{R}$ such that

$$\varepsilon_{x,k_n} = \inf(\|f - p_a\|_I : \|a\|_2 \leq T)$$

Proof: Let $S = \{a \in \mathbb{R}^{n+1} : \|a\|_2 = 1\}$ and define $\delta_S \coloneqq \inf(\|p_a\|_I : a \in S)$. By (a), $\|p_a\|_I$ is continuous, and as S is compact we have that there exists $a^* \in S$ such that $\|p_{a^*}\|_I = \delta_S$. Note that necessarily $k_n \geq n+1$, then by definition of $\|p_{a^*}\|_I$ it must be that either $\delta_S > 0$ or $p_{a^*} = 0$. Since $a^* \in S$, necessarily it is the former that holds. Hence, for every $a \in \mathbb{R}^{n+1}$ we have that $\left\|p_{a/\|a\|_2}\right\|_I \geq \delta_S$, and by $\|p_a\|_I = \|a\|_2 \cdot \left\|p_{a/\|a\|_2}\right\|_I$ we have $\|p_a\|_I \xrightarrow[\|a\|_2 \to \infty]{} \infty$. Finally, note that $\|f - p_a\|_I \geq \|p_a\|_I - \|f\|_I$, and let T such that $\|a\|_2 \geq T \Rightarrow \|p_a\|_I > \varepsilon_{x,k_n} + 1 + \|f\|_I$, then for all $a : \|a\|_2 \geq T$ we have $\|f - p_a\|_I \geq \varepsilon_{x,k_n} + 1 + \|f\|_I - \|f\|_I = \varepsilon_{x,k_n} + 1$. Hence, $\inf(\|f - p\|_I : p \in P_n) = \inf(\|f - p_a\|_I : \|a\|_2 \leq T)$.

By (b) and (c), ε_{x,k_n} is the infimum of a continuous function on a closed ball, hence there exists $a^* \in \mathbb{R}^{n+1}$ such that $\varepsilon_{x,k_n} = \|f - p_{a^*}\|_I$. By the assumption that f is PIL, and the definition of k_n, we have $\|f - p_{a^*}\|_I > 0$. □

Lemma 3.6.4. *For every $q, n \in \mathbb{N}$ there exists a point $T_{q,n} \in \mathbb{N}$ and a gap $\delta_{T_{q,n}} > 0$ such that: for every PIL $f : \mathbb{N} \to \mathbb{R}$, and every piecewise-polynomial g with q many pieces of degree $\leq n$, there exists $y \in \mathbb{N}$, $0 \leq y \leq T_{q,n}$ for which $\left|g(y) - f(y)\right| \geq \delta_{T_{q,n}}$. That is, the number of pieces and the maximum degree of a piecewise-polynomial g determine a guaranteed minimum gap by which g misses f within a guaranteed interval.*

Proof. Define $T_0 = 1$. Using the notation of k_n from Lemma 3.6.3, for every $i \in [q]$ define $T_i \coloneqq (k_n - 1)(i) + 1$, define $I_i \coloneqq I_{T_{i-1},k_n}$, and define $\delta_i \coloneqq \inf(\|f - p\| I_i : p \in P_n)$. Note that $\delta_i > 0$ by Lemma 3.6.3. Finally, define $T_{q,n} \coloneqq T_q$, $\delta_{T_{q,n}} \coloneqq \min(\delta_i : i \in [q])$. Assume by contradiction that g is close to f by less than $\delta_{T_{q,n}}$ for every $y \in [0..T_{q,n}]$, then, necessarily the first polynomial piece of g ends at most at $T_1 - 1$, the second at $T_2 - 1$ and the $q-1$ piece at $T_{q-1} - 1$, then the last polynomial piece starts the latest at T_{q-1} and by $T_{q,n}$ it must have missed at least one point by at least $\delta_{T_{q,n}} > 0$. □

Theorem 3.6.5. *Let $f : \mathcal{G}_1 \to \mathcal{Z}_\mathbb{R}$ a feature transformation, let $g(k) \coloneqq f(G_k)(u)$, and assume that g is PIL. Then, for every* MUPA*-GNN architecture A, there exists $\varepsilon_A > 0$ such that for every N that is a model of A there exists k such that $\left|N(G_k, u) - f(G_k)(u)\right| \geq \varepsilon$.*

Proof. Let the q, r guaranteed by Lemma 3.6.1 for A, and let the $T_{q,r}, \delta_{T_{q,r}}$ guaranteed by Lemma 3.6.4 for q pieces of degree $\leq r$. Then, by Lemma 3.6.4, for $\varepsilon_A \coloneqq \delta_{T_{q,r}}$ and $k \coloneqq T_{q,r}$ the statement holds. □

3.7 Experimentation

We experiment with vertex-level regression tasks. In previous sections we formally proved certain expressivity properties of Sum; Mean; and Max GNNs. Our goal in experimentation is to examine how these properties may affect practical learnability: searching for an approximating GNN using stochastic gradient-descend. With training data ranging over only a small section of the true-distribution range, does the existence of a uniformly-expressing GNN increase the chance that a well-generalizing GNN will be learned?

Specific details concerning training and architecture can be found in the appendix of [RTG23][1].

3.7.1 Data and Setup

For the graphs in the experiments, and with our GNN architecture consisting of two GNN layers (see Appendix B in [RTG23]), mean and max aggregations output the same value for every vertex, up to machine precision. Thus, it is enough to experiment with mean and assume identical results for max.

We conduct experiments with two different datasets, one corresponds to the approximation task in Section 3.5.1, and the other to the task in Section 3.5.2:

1. **U**nbounded **C**ountable Feature Domain (UC): This dataset consists of the star graphs $\{G_{k,c}\}$ from Section 3.5.1, for $k, c \in [1..1000]$. The center's ground truth value is c, and it is the only vertex whose value we want to predict.

2. **S**ingle-**V**alue Feature Domain (SV): This dataset consists of the graphs $\{G_{k,c}\}$ from Section 3.5.2, for $k, c \in [1..1000]$. Again, the center's ground truth value is c, and we do not consider the other vertices' predicted values.

As training data, we vary $k \in [1..100]$ and $c \in [1..100]$. We therefore train on 10K graphs in each experiment. Afterwards, we test each GNN model on larger graphs with $k \in [101..1000\}$ and $c \in [101..1000]$. The increased range of k and c in testing simulates the scenario of unbounded graph sizes and unbounded feature values, allowing us to study the performance in terms of uniform expressivity with unbounded features.

3.7.2 Results

Our primary evaluation metric is the relative error. Formally, if y_{pred} is the prediction of the GNN for the center vertex of an input graph G, with truth label c, we define the relative error as

$$\text{RE}\left(y_{\text{pred}}, c\right) = \frac{|y_{\text{pred}} - c|}{|c|}.$$

A relative error greater or equal to 1 is a strong evidence for inability to approximate, as the assessed approximation is no-better than an always-0 output. It is also reasonable that in practice, when judging the regression of a function whose range vary by a factor of 1000, relative error would be the relevant measure.

[1] code for running the experiments is found at `https://github.com/toenshoff/Uniform_Graph_Learning`

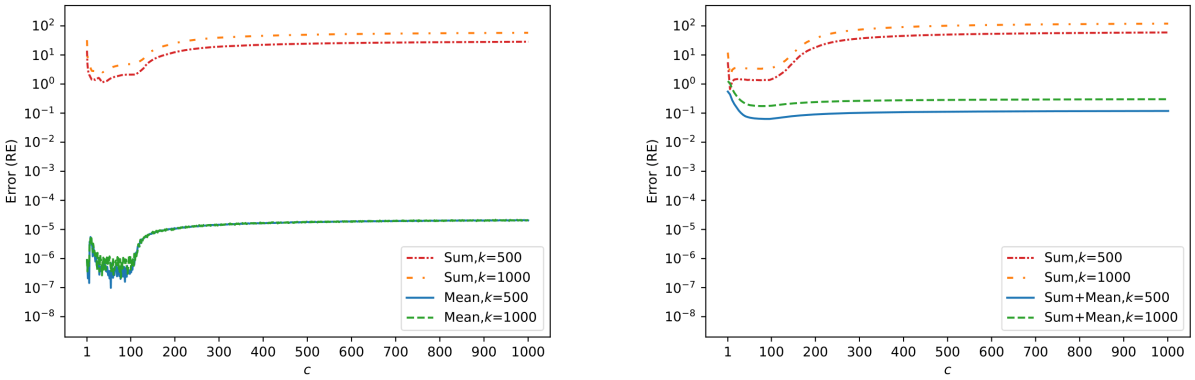


Figure 3.5: Relative Error of different aggregations on Unbounded Countable Features (left) and Single Value Features (right)

Unbounded, Countable, Feature Domain Figure 3.5 provides the test results for UC. Here, we illustrate our results for two representing values of k: $500, 1000$, for all values of c. Note that the error has a logarithmic scale. Mean-GNNs achieve very low relative errors of less than 10^{-4} across all considered combinations of k and c. Their relative error falls to less than 10^{-6} when c is within the range seen during training (≤ 100), Therefore, Mean-GNNs do show some degree of overfitting. Notably, the value of k has virtually no effect on the error of Mean-GNNs . This is expected, since mean aggregation should not be affected by the degree k of a center vertex whose neighbors are identical, up to machine precision. Sum-GNNs yield a substantially higher relative error. For $k = 500$ and $c \leq 100$ the relative error is roughly 1, but this value increases as c grows beyond the training range. Crucially, the relative error of Sum-GNNs also increases with k. For $k = 1000$, the relative error is above 1 even when c is within the range seen during training. Therefore, Sum-GNNs do generalize significantly worse than Mean-GNNs in both parameters k and c. ‘ **Single-Value Feature Domain** Figure 3.5 provides the test results for SV. Again, we plot the relative error against different values of c. Sum-GNNs yield similar relative errors as in the UC experiment. As expected, learned (Sum,Mean)-GNNs do perform significantly better than Sum-GNNs. However, the learning of (Sum,Mean)-GNNs is not as successful as the learning of Mean-GNNs in the UC experiment: relative error is around 10^{-1} for $k = 500$, and slightly larger for $k = 1000$, clearly worse than the UC-experiment performance. In particular, the learned (Sum,Mean)-GNN is sensitive to increases in k. Note that each (Sum,Mean)-GNN layer receives both sum and mean aggregations arguments and needs to choose the right one, thus it is a different learning challenge than in the first experiment.

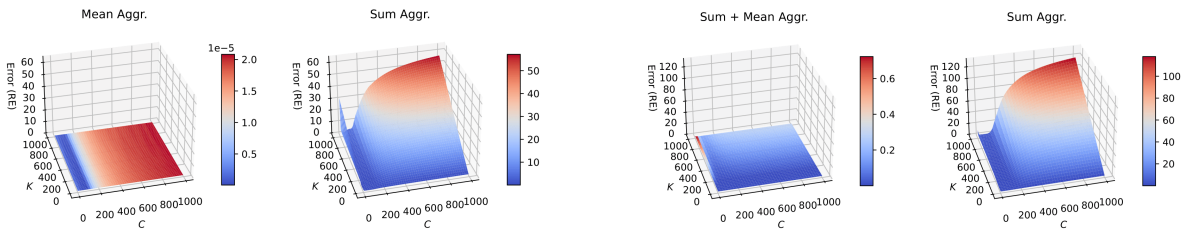
3.7.3 Extended Results

An illustration of the full experimental results can be seen in fig. 3.6. For both datasets, and each tested architecture, we provide the relative error (RE) over the full test range ($k \in [1..1000], c \in [1..1000]$) as a 3D plot. The error is provided on the z-axis, which is linearly scaled. The color map is linear as well and is scaled individually for each subplot to highlight additional details.

The results for the unbounded countable features (UC) experiment are provided in

fig. 3.6a. Note that the color map for the trained Mean-GNN is scaled by 10^{-5}, since the learned function is very close to the ground truth. The trained Sum-GNN performs significantly worse. Relative to itself though, as long as c is in the training range [1..100] it generalizes well along the k axis. Operating the trained Sum-GNN , on c in the training range, resembles the bounded initial-feature domain setting examined in Section 3.4. Hence, the generalization in k, when c is in the training range, resembles the result in Section 3.4: Sum-GNNs can approximate mean when the initial-feature domain is bounded. Once c is beyond the training range, the relative error grows rapidly, both along the k axis (for fixed c) and along the c axis. Interestingly, the error of the trained Sum-GNN also tends upwards at $c < 10$. The learned function therefore lacks robustness even towards the lower end of the training range of c.

The results for the single value features (SV) experiment are provided in fig. 3.6b. Overall, the trained (Sum,Mean)-GNN achieves a significantly lower error than the Sum-GNN. Like in the UC experiment, as long as c is in the training range [1..100] the trained Sum-GNN generalizes relatively well along the k axis, and the performance deteriorates sharply (in both axis) when $c > 100$. We do note though, that the results of the (Sum,Mean)-GNN in this experiment are substantially worse than those of the Mean-GNN in the UC experiment. While there exists a (Sum,Mean)-GNN that computes exactly the SV-experiment function (see proof of Corollary 3.5.19), Stochastic Gradient Descend (SGD) was not able to learn this function in fine detail. To arrive in a good (Sum,Mean)-GNN instance, the first GNN-layer has to learn to ignore the coordinates of the mean-aggregation and to use the coordinates of the sum-aggregation properly, and the second GNN-layer has to learn to ignore the sum and use the mean. These requirements constitute a more challenging learning problem than that of learning a good Mean-GNN for the UC task, and the difference is reflected in the results. Interestingly, the relative error of the (Sum,Mean)-GNN is worst at the lower end of the training range $c < 10$ for high values of k.



(a) Unbounded Countable Features **(b)** Single Value Features

Figure 3.6: Relative Error of different aggregations on UC and SV.

3.8 Summary

We have shown that when comparing single-aggregation common GNNs i.e. Sum-GNNs vs. Mean-GNNs vs. Max-GNNs, there is a certain advantage to the sum-aggregation: When the input features are bounded - a reasonable setting in practice - it is theoretically a sum-supreme case i.e. any Mean or Max GNN can be approximated by a Sum-GNN. We

have also shown that the features are unbounded - a setting which is a good estimation for a scenario in practice where the range at inference time is greater than at training time - sum does not subsume, and the different aggregations' expressivity are partially disjoint. Then we have shown that unlike non-uniform expressivity results, Sum-GNNs do not subsume any other GNNs: A (Sum,Mean)-GNN is strictly stronger already for a trivial initial-features domain. Lastly, we have shown that none of the common aggregations or any finite combination of them is sufficient for expressing even natural functions on a very restricted graph domain.

Our construction of Sum-GNNs that approximate Mean and Max GNNs incurs an inflation in the network size. The uniform setting means that the asymptotic size-complexity with respect to the graph size is not affected by that inflation, as that inflation is independent of the graph size, however it is a substantial inflation with respect to the approximated-network's size, which may deem the emulation impractical. As we have not proven a lower-bound for the construction size, the question whether there is a more efficient construction remains open.

Are Targeted Messages More Effective?

"Everyone is not your customer."

Seth Godin

4.1 Intro

This chapter is based on the sections relating to GNNs, in [GR24]. In Chapter 3, we have assumed a trivial msg algorithm - outputting simply the value of the neighbor, and compared the expressivity of GNNs having different aggregation functions, in different initial-feature-domain settings. Here, we turn to examine the role of the msg algorithm in the uniform expressivity of GNNs. For each of the three common aggregations (sum, mean, max) we compare the 1-sided GNN version i.e. having the neighbor-identity msg algorithm, to the 2-sided version i.e. having an MLP whose inputs are the subject vertex value and its neighbor's value.

Both the 1-sided and the 2-sided versions have been studied before, and both have been used in practice. Instances of 1-GNNs are the graph convolutional networks of [KW17a]. The message passing neural networks of [Gil+17b] are 2-GNNs. Most of the theoretical work on GNNs considers 1-GNNs (e.g. [Bar+20b; Mor+19b; Xu+19a]); this also has to do with the fact that this work is concerned with non-uniform expressivity, for which it is easy to see that the two versions coincide. In practical work, the prevalent perception seems to be that 2-GNNs are superior—this is also our own experience—though an explicit empirical comparison between the two versions has only been carried out recently [Tai+22] (there 1-GNNs are called isotropic and 2-GNNs are called anisotropic).

All our results are in the setting of boolean initial-feature domain and binary range, and we refer to such functions as *queries*. Note that with a binary range (and in fact any finite range) approximation coincides with exact computation: Let $a, b \in \mathbb{R},\ a < b$ be the two target classes, let $f : \mathcal{G}_D \to \mathcal{Z}_{\{a,b\}}$ be a target binary classification defined for some graphs domain $\mathcal{G}_D$, let $\varepsilon < \frac{b-a}{4}$, and let N' be an approximating GNN such that $\forall G \in \mathcal{G}_D\ \forall v \in V(G)\ \left|f(G)(v) - N(G,v)\right| \leq \varepsilon$, then,

$$f(G)(v) = b\,\mathrm{lsig}(2 - \frac{4\left|b - N(G,v)\right|}{b-a}) + a\,\mathrm{lsig}(2 - \frac{4\left|a - N(G,v)\right|}{b-a})$$

Hence, there exists a GNN N - which has the required layers on top of N' - for which

$$\forall G \in \mathcal{G}_D\ \forall v \in V(G)\ \ N(G,v) = f(G)(v)$$

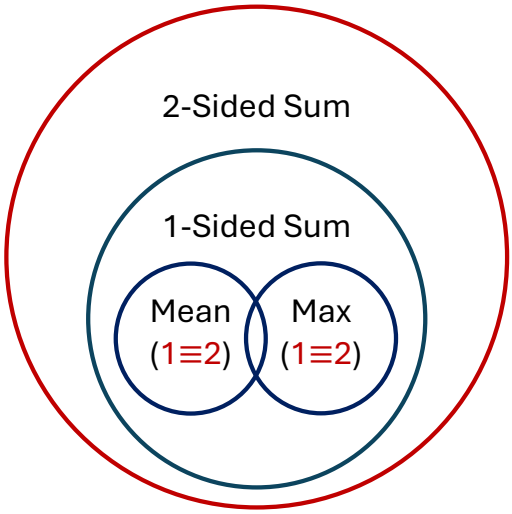


Figure 4.1: The expressivity relations between 1-GNNs and 2-GNNs, for boolean input domain. The strict containment of 1-sided Mean and Max in Sum is already proved in Chapter 3. The strict containment of 1-Sum-GNNs in 2-Sum-GNNs, and the equivalencies for Mean and Max, are proven in this chapter.

4.1.1 New Results

1. For Sum-GNNs, 2 sides is stronger. In Section 4.3 we prove that there is a binary classification function that is trivially expressible by a 2-Sum-GNN but is inexpressible by any 1-Sum-GNN. Moreover, we prove that the strict subsumption holds not only for relu-activated Sum-GNNs but for a whole class of activation functions which we precisely characterize.

2. For Mean-GNNs and Max-GNNs, 2 sides has the same expressivity as 1 side. In Section 4.3.1 we prove that 1-Max-GNNs can approximate 2-MAX-GNNS, and 1-Mean-GNNs can approximate 2-Mean-GNNS.

The results above are illustrated in Figure 4.1.

4.2 Terms and Concepts

We call a GNN with aggregation agg and the trivial msg algorithm, a *1*-agg-*GNN* - for considering only the neighbor's side of the edge. We call a GNN with aggregation agg and an MLP for msg algorithm, a *2*-agg-*GNN* - for considering both sides. For example, a Sum-GNN where the msg algorithm is an MLP is referred to as a 2-Sum-GNN. When clear from the context, we may omit the aggregation name and refer simply to 1-GNNs and 2-GNNs.

4.3 Sum Aggregation

Let $\mathcal{Q}_1$ be the binary classification function, on trivially-featured graphs, that selects all vertices who have a neighbor of larger degree than their own. That is, for every graph

and vertex $G \in \mathcal{G}_1, v \in V(G)$ we let

$$\mathcal{Q}_1(G)(v) \coloneqq \begin{cases} 1 & \exists u \in N_G(v) : \deg_G(v) < \deg_G(u) \\ 0 & \text{otherwise} \end{cases}$$

Theorem 4.3.1. *$\mathcal{Q}_1$ is expressible by a 2-Sum-GNN, but not by a 1-Sum-GNN.*

It is fairly obvious that $\mathcal{Q}_1$ is expressible by a 2-layer 2-GNN. The first layer computes the degree of all vertices by summing the constant message 1 for all neighbours. On the second layer, the message sent along an edge (v, u) is 1 if $\deg(u) < \deg(v)$ and 0 otherwise. The combination function then just needs to check if the sum of all messages is at least 1.

The proof that $\mathcal{Q}_1$ is not expressible by a 1-GNN requires some preparation. We call a polynomial $\mathfrak{p}(X, Y)$ in two variables *nice* if it is of the form

$$\mathfrak{p}(X, Y) = \Sigma_{i=0}^{k} a_i X^{\lfloor i/2 \rfloor} Y^{\lceil i/2 \rceil} \tag{4.3.1}$$

with arbitrary real coefficients a_i. The *leading coefficient* of $\mathfrak{p}$ is a_i for the maximum i such that $a_i \neq 0$, or 0 if all a_i are 0. We call a polynomial $\mathfrak{p}(X, Y)$ *co-nice* if $\mathfrak{p}(Y, X)$ is nice.

Lemma 4.3.2. *Let $\mathfrak{p}(X, Y)$ be a nice or co-nice polynomial with leading coefficient a. Then either $\mathfrak{p}$ is constant and thus $\mathfrak{p}(m, n) = a$ for all $m, n \in \mathbb{N}$, or there is an $n_0 \in \mathbb{N}$ such that for all $n \geq n_0$ we have*

$$\operatorname{sign}\big(\mathfrak{p}(n-1, n)\big) = \operatorname{sign}\big(\mathfrak{p}(n+1, n)\big) = \operatorname{sign}(a) \tag{4.3.2}$$

and

$$|\mathfrak{p}(n-1, n)|, |\mathfrak{p}_1(n+1, n)| \geq \frac{|a|}{2} n. \tag{4.3.3}$$

Proof. There is nothing to prove if $\mathfrak{p}$ is constant, so assume that it is not. Assume that $\mathfrak{p}(X, Y)$ is co-nice. (The argument for nice $\mathfrak{p}$ is similar and even slightly simpler.) Then

$$\mathfrak{p}(X, Y) = \Sigma_{i=0}^{k} a_i X^{\lceil i/2 \rceil} Y^{\lfloor i/2 \rfloor}$$

for some $k \geq 1$ and coefficients a_i with $a_k = a \neq 0$. We write $\mathfrak{p}$ as $\mathfrak{p} = \mathfrak{p}_1 + \mathfrak{p}_2$, where

$$\mathfrak{p}_1(X, Y) = aX^{\lceil k/2 \rceil} Y^{\lfloor k/2 \rfloor}, \quad \mathfrak{p}_2(X, Y) = \Sigma_{i=0}^{k-1} a_i X^{\lceil i/2 \rceil} Y^{\lfloor i/2 \rfloor}.$$

It is easy to see that for sufficiently large n_0 and $m, n \geq n_0$ we have $|\mathfrak{p}_2(m, n)| \leq \frac{1}{4}|\mathfrak{p}_1(m, n)|$, which implies

$$\operatorname{sign}\big(\mathfrak{p}(m, n)\big) = \operatorname{sign}\big(\mathfrak{p}_1(m, n)\big) = \operatorname{sign}(a)$$

and hence (4.3.2). It also implies

$$|\mathfrak{p}(m, n)| \geq \frac{3}{4}\mathfrak{p}_1(m, n) \geq \frac{3}{4}|a|m,$$

where the last inequality holds because $k \geq 1$ and thus $\lceil k/2 \rceil \geq 1$. Since for sufficiently large n we have $n - 1 \geq \frac{2}{3}n$, (4.3.3) follows. □

We say that a function $f : \mathbb{N}^2 \to \mathbb{R}$ is *fast-converging* to a polynomial $\mathfrak{p}(X, Y)$ if for all $r \in \mathbb{R}$ there is an n_0 such that for all $n \geq n_0$ and $m \in \{n-1, n+1\}$ it holds that

$$\big|f(m,n) - \mathfrak{p}(m,n)\big| \leq \frac{1}{n^r}.$$

Note that fast convergence only considers argument pairs $(n-1, n)$ or $(n+1, n)$ for $n \in \mathbb{N}_{>0}$. We let $\mathfrak{C}(\mathfrak{p})$ be the class of all $f : \mathbb{N}^2 \to \mathbb{R}$ that are fast-converging to $\mathfrak{p}$. We let

$$\mathfrak{C} \coloneqq \bigcup_{\mathfrak{p}(X,Y) \text{ nice}} \mathfrak{C}(\mathfrak{p})$$

be the class of all functions that are fast-converging to some nice polynomial. Similarly, for a co-nice polynomial $\mathfrak{p}$ we denote the fast convergence of f to $\mathfrak{p}$ by $\mathcal{FC}^{\mathrm{co}}(f, \mathfrak{p})$, we let $\mathcal{FC}^{\mathrm{co}}(\mathfrak{p})$ be the class of all $f : \mathbb{N}^2 \to \mathbb{R}$ fast-converging to $\mathfrak{p}$, and define $\mathcal{FC}^{\mathrm{co}}$ respectively.

A function $g : \mathbb{R}^k \to \mathbb{R}$ *preserves fast convergence* if for all $f_1, \ldots, f_k \in \mathfrak{C}$ it holds that $g(f_1, \ldots, f_k) \in \mathfrak{C}$ and for all $f_1, \ldots, f_k \in \mathcal{FC}^{\mathrm{co}}$ it holds that $g(f_1, \ldots, f_k) \in \mathcal{FC}^{\mathrm{co}}$.

Here $g(f_1, \ldots, f_k) : \mathbb{N}^2 \to \mathbb{R}$ is the function defined by

$$g(f_1, \ldots, f_k)(x, y) \coloneqq g\big(f_1(x, y), \ldots, f_k(x, y)\big).$$

Lemma 4.3.3. *1. All constant functions $c : \mathbb{N}^2 \to \mathbb{R}$ are in $\mathfrak{C} \cap \mathcal{FC}^{\mathrm{co}}$.*

2. For $f : \mathbb{N}^2 \to \mathbb{R}$, define $f_1, f_2 : \mathbb{N}^2 \to \mathbb{R}$ by $f_1(m, n) \coloneqq m \cdot f(n, m)$ and $f_2(m, n) \coloneqq n \cdot f(m, n)$. Then

$$f \in \mathcal{FC}^{\mathrm{co}} \implies f_1 \in \mathfrak{C};$$
$$f \in \mathfrak{C} \implies f_2 \in \mathcal{FC}^{\mathrm{co}}.$$

3. All linear functions preserve fast convergence.

4. All functions computed by MLPs that only use activation functions preserving fast convergence preserve fast convergence.

Proof. All polynomials of degree 0 (constants) are nice, which proves 1.

To prove 2, suppose that $f \in \mathfrak{C}(\mathfrak{p})$ for the polynomial $\mathfrak{p}$ in (4.3.1). Note that

$$X \cdot \mathfrak{p}(X, Y) = \Sigma_{i=0}^{k} a_i X^{\lfloor i/2 \rfloor + 1} Y^{\lceil i/2 \rceil} = \Sigma_{i=0}^{k} a_i X^{\lceil (i+1)/2 \rceil} Y^{\lfloor (i+1)/2 \rfloor},$$

which is co-nice. To establish the fast convergence, let $r \in \mathbb{R}$. Choose $n_0 \geq 2$ such that for $n \geq n_0$ and $m \in \{n-1, n+1\}$ it holds that $\big|f(m,n) - \mathfrak{p}(m,n)\big| \leq n^{-(r+2)}$. Then

$$\big|m \cdot f(m,n) - m \cdot \mathfrak{p}(m,n)\big| \leq (n+1)\big|f(m,n) - \mathfrak{p}(m,n)\big| \leq \frac{n+1}{n^{r+2}} \leq \frac{1}{n^r}.$$

The second assertion in 2 follows from the first.

Assertion 3 follows from the observation that linear combinations of nice polynomials are nice.

Assertion 4 follows directly from 1 and 3. □

Remark 4.3.4. The proof of Lemma 4.3.32 is the only place where we need "fast" convergence, with a convergence rate bound by an inverse polynomial function in n, instead of just ordinary convergence.

Lemma 4.3.5. *The* relu *function preserves fast convergence.*

Proof. Suppose that $f \in \mathfrak{C}(\mathfrak{p})$ for the polynomial $\mathfrak{p}$ in (4.3.1). We need to show that $\mathrm{relu}(f) \in \mathfrak{C}$. Then the assertion for co-nice polynomials follows by flipping the arguments. Let a be the leading coefficient of $\mathfrak{p}$. If $a = 0$, then $\mathfrak{p} = 0$ is the zero polynomial, and $\mathrm{relu}(f) \in \mathfrak{C}(\mathfrak{p})$ follows from the observation that $|\mathrm{relu}(f(m,n))| \leq |f(m,n)|$ for all m, n. Suppose that $a \neq 0$. By Lemma 4.3.2, there is an n_0 such that for all $n \geq n_0$ and $m \in \{n-1, n+1\}$ we have

$$\mathfrak{p}(m,n) \begin{cases} \geq a & \text{if } a > 0, \\ \leq a & \text{if } a < 0. \end{cases}$$

By fast convergence, we may further assume that $|f(m,n) - \mathfrak{p}(m,n)| \leq \frac{a}{2}$. Thus if $a > 0$, we have $\mathrm{relu}(f(m,n)) = f(m,n)$ and thus $\mathrm{relu}(f) \in \mathfrak{C}(\mathfrak{p})$. If $a < 0$ we have $\mathrm{relu}(f(m,n)) = 0$ and thus $\mathrm{relu}(f) \in \mathfrak{C}(0)$. □

Lemma 4.3.6. *The logistic function preserves fast convergence.*

Proof. Let $f \in \mathfrak{C}(\mathfrak{p})$ for some nice polynomial $\mathfrak{p}$. (The proof for co-nice polynomials is similar.) We shall prove that $g := \mathrm{sig}(f) \in \mathfrak{C}$. Note that $g(m,n) \coloneqq \mathrm{sig}(f(m,n)) = \frac{1}{1+e^{-f(m,n)}}$.

Let a be the leading coefficient of $\mathfrak{p}$.

Case 1: $\mathfrak{p}$ is not constant and $\mathrm{sign}(a) > 0$.
We shall prove that $g \in \mathfrak{C}(1)$.

Let $r \in \mathbb{R}$. Then for suffficiently large n and $m \in \{n-1, n+1\}$ we have $f(m,n) \geq \frac{\mathfrak{p}(m,n)}{2} \geq \frac{a}{4}n$ (since $f \in \mathfrak{C}(\mathfrak{p})$ and by Lemma 4.3.2) and thus

$$\begin{aligned} \big|g(m,n) - 1\big| &= \left|\frac{1}{1+e^{-f(m,n)}} - 1\right| = \frac{e^{-f(m,n)}}{1+e^{-f(m,n)}} \\ &\leq e^{-f(m,n)} \leq e^{-\frac{|a|}{4}n} \leq \frac{1}{n^r}. \end{aligned}$$

Case 2: $\mathfrak{p}$ is not constant and $\mathrm{sign}(a) < 0$.
In this case, we can prove, similarly to Case 1, that $g \in \mathfrak{C}(0)$.

Case 3: $\mathfrak{p}$ is constant.
Then $\mathfrak{p}(m,n) = a$ for all m, n. We shall prove that $g \in \mathfrak{C}(\mathrm{sig}(a))$. For $n \geq 0$, we let

$$\varepsilon_n \coloneqq \mathsf{max}\big\{\big|f(n-1,n) - a\big|, \big|f(n+1,n) - a\big|\big\}.$$

Since $f \in \mathfrak{C}(\mathfrak{p}) = \mathfrak{C}(a)$, for every $r \in \mathbb{R}$ there is an n_0 such that $\varepsilon_n \leq n^{-r}$ for all $n \geq n_0$.

Let $n \geq n_0$ and $m \in \{n-1, n+1\}$. We have $a - \varepsilon_n \leq f(m,n) \leq a + \varepsilon_n$ and thus $\mathrm{sig}(a - \varepsilon_n) \leq g(m,n) \leq \mathrm{sig}(a + \varepsilon_n)$

Since $\frac{d\,\mathrm{sig}}{dx} = \mathrm{sig}(x)(1-\mathrm{sig}(x)) \leq 1$ it follows that

$$\begin{aligned}&\big|g(m,n) - \mathrm{sig}(a)\big|\\ &\leq \mathsf{max}\Big\{\big|\,\mathrm{sig}(a-\varepsilon_n) - \mathrm{sig}(a)\big|, \big|\,\mathrm{sig}(a+\varepsilon_n) - \mathrm{sig}(a)\big|\Big\}\\ &\leq \varepsilon_n \leq n^{-r}.\end{aligned}$$

This proves that $g \in \mathfrak{C}\big(\mathrm{sig}(a)\big)$. □

In view of Lemma 4.3.5, the following theorem immediately implies Theorem 4.3.1.

Theorem 4.3.7. *The query $\mathcal{Q}_1$ is expressible by a 2-GNN with sum-aggregation, but not by a 1-GNN with sum-aggregation and with arbitrary activation functions that preserve fast convergence.*

Proof. Suppose for contradiction that

$$N = ((\mathsf{comb}_1, \mathrm{sum}), \ldots, (\mathsf{comb}_d, \mathrm{sum}))$$

is a 1-GNN that approximates $\mathcal{Q}_1$. By adding a bias of $-1/2$ to the output of the last layer, we may actually assume that for all graphs G and vertices $v \in V(G)$ it holds that

$$N(G)(v) \begin{cases} \geq \frac{1}{4} & \exists u \in N_G(v) :\ \deg_G(v) < \deg_G(u) \\ \leq -\frac{1}{4} & \text{otherwise} \end{cases} \tag{4.3.4}$$

Let $\mathsf{comb}_i : \mathbb{R}^{2p^{(i)}} \to \mathbb{R}^{q^{(i)}}$ be an MLP of I/O dimensions $p^{(i)}, q^{(i)}$. We have $p^{(0)} = 1$, because the input graph is trivial, $q^{(i)} = p^{(i+1)}$ for $0 \leq i < d$, and $q^{(d)} = 1$.

Let $G_{k,l}$ be a parameterized fully-connected bipartite graph, for $k, l \in \mathbb{N}_{>0}, k > 2, l > 2$ where one side has k vertices and the other has l vertices, and all vertices' initial value is 1. Formally:

- $V(G_{k,l}) = \{u_1, \ldots, u_k\} \cup \{v_1, \ldots, v_l\}$
- $E(G_{k,l}) = \bigcup_{i\in[k], j\in[l]} \{\{u_i, v_j\}\}$
- $Z(G_{k,l}) = \bigcup_{i\in[k]} \{(u_i, 1)\} \bigcup_{i\in[l]} \{(v_i, 1)\}$

Clearly, $\forall t \in [d]\ \forall i, j \in [k]\ N^{(t)}(G_{k,l}, u_i) = N^{(t)}(G_{k,l}, u_j)$, then denote by $u_{k,l}^{(t)} \coloneqq N^{(t)}(G_{k,l}, u_i), i \in [k]$ the value of any of the u_is after operating the first t layers of N on $G_{k,l}$. Similarly, $\forall i, j \in [l]\ N^{(t)}(G_{k,l}, v_i) = N^{(t)}(G_{k,l}, v_j)$, then denote by $v_{k,l}^{(t)} \coloneqq N^{(t)}(G_{k,l}, u_i), i \in [k]$.

For $t \in [d], j \in [q^{(t)}]$, let $f_j^{(t)} : \mathbb{N}^2 \to \mathbb{R}$ be the function defined by

$$u_{k,l}^{(t)} = \big(f_1^{(t)}(k,l), \ldots, f_{q^{(t)}}^{(t)}(k,l)\big)$$

That is, $f_j^{(t)}(m,n)$ is the j^{th} entry of the state of every $u_i \in V(G_{k,l})$ after applying the first t layers of N to the graph $G_{k,l}$. Similarly, we let $g_j^{(t)} : \mathbb{N}^2 \to \mathbb{R}$ be the function defined by

$$v_{k,l}^{(t)} = \big(g_1^{(t)}(k,l), \ldots, g_{q^{(t)}}^{(t)}(k,l)\big)$$

Claim 4.3.8. *For all $t \in [d]$, $j \in [q^{(t)}]$ we have*

$$f_j^{(t)} \in \mathfrak{C}, \quad g_j^{(t)} \in \mathcal{FC}^{\mathrm{co}}.$$

Proof. The proof is by induction on t.

For the base step, we observe that $f^{(1)}$ must be constant, because $f^{(0)}$ maps all vertices to the empty tuple, the message function is the identify, the sum of empty tuples is the empty tuples, so the combination function receives the empty tuple as input (at all vertices).

For the inductive step, suppose that $t > 1$ and that $f_j^{(t-1)} \in \mathfrak{C}, g_j^{(t-1)} \in \mathcal{FC}^{\mathrm{co}}$. Let $p \coloneqq p^{(t)} = q^{(t-1)}$, and let $q \coloneqq q^{(t)}$. It will be convenient to think of $\mathsf{comb}_t : \mathbb{R}^{2p} \to \mathbb{R}^q$ as a tuple $(c_1, \dots, c_q)$, where $c_j : \mathbb{R}^{2p} \to \mathbb{R}^q$. Since comb_t is computable by an MLP with activations that preserve fast convergence, each c_j preserves fast convergence. For all $u_i \in V(G_{k,l})$, we have

$$u_{k,l}^{(t)} = \mathsf{comb}_t \left(u_{k,l}^{(t-1)}, \Sigma_{v_j \in V(G_{k,l})} v_{k,l}^{(t-1)} \right)$$

and thus, for $j \in [q]$,

$$f_j^{(t)}(k,l) = c_j \Big(f_1^{(t-1)}(k,l), \dots, f_p^{(t-1)}(k,l), l \cdot g_1^{(t-1)}(k,l), \dots, l \cdot g_p^{(t-1)}(k,l) \Big)$$

By the induction hypothesis, we have $\forall i \in [p]\ f_i^{(t-1)} \in \mathfrak{C}$ and $\forall i \in [p]\ g_i^{(t-1)} \in \mathcal{FC}^{\mathrm{co}}$. The latter implies, by Lemma 4.3.32, that the function $(k,l) \mapsto l \cdot g_i^{(t-1)}(k,l)$ is in $\mathfrak{C}$. Thus $f_j^{(t)} \in \mathfrak{C}$. The argument for $g_j^{(t)}$ is similar. This completes the proof of the claim. ⌟

The function $f_1^{(d)}$ computes the output of N for all vertices $u_i \in V(G_{k,l})$: $f_1^{(d)}(k,l) = u_{k,l}^{(d)} = N(G_{k,l}, u)$. Thus by (4.3.4),

$$f_1^{(d)}(k,l) \begin{cases} \geq \frac{1}{4} & \text{if } k < l, \\ \leq -\frac{1}{4} & \text{if } l \geq k. \end{cases}$$

Thus for all $l \in \mathbb{N}_{>0}$, $f_1^{(d)}(l-1,l) \geq \frac{1}{4}$ and $f_1^{(d)}(l+1,l) \leq -\frac{1}{4}$. Since $f_1^{(d)}(k,l) \in \mathfrak{C}$, this contradicts Lemma 4.3.2. □

Interestingly, there are also somewhat natural functions that, when used as activations, allow 1-GNNs to express the query $\mathcal{Q}_1$.

Example 4.3.9.

1. The query $\mathcal{Q}_1$ can be expressed by a 1-GNN with sum-aggregation and the square root function as activation function.

2. The main results of [Gro23] hold for a class of *piecewise linear approximable* activations. The functions $f, g : \mathbb{R} \to \mathbb{R}$ defined by $f(x) = \min\{1, 1/|x|\}$ and $g(x) = \min\{1, 1/\sqrt{|x|}\}$ (with $f(0) = g(0) = 1$) are piecewise linear approximable. Yet it can be shown that $\mathcal{Q}_1$ can be expressed by a 1-GNN with sum-aggregation that uses f and g as activations.

We leave the proofs of these assertions to the reader.

4.3.1 Mean and Max Aggregations

Maybe surprisingly, the use of sum-aggregation is crucial in Theorem 4.3.1. The corresponding result for mean or max aggregation does not hold. To the contrary, we have the following.

Theorem 4.3.10. *1. Every query computable by a 2-GNN with* max*-aggregation is computable by a 1-GNN with* max*-aggregation.*

2. Every query computable by a 2-GNN with mean*-aggregation is computable by a 1-GNN with* mean*-aggregation.*

To explain the proof, let us first consider max-aggregation. Consider a 2-GNN N with max-aggregation that computes some query $\mathcal{Q}$ on ℓ-labelled graphs $G \in \mathcal{G}_{\{0,1\}^\ell}$, for some fixed ℓ. The initial feature map $Z(G)$ only takes values in the finite set $\{0,1\}^\ell$. It is not hard to prove, by induction on the number of layers, that in a 2-GNN with max-aggregation, the range of the feature map remains finite through all layers of the computation. Now the trick is to use a one-hot encoding of the possible values. Suppose that after i-steps of the computation, the feature map $N^{(i)}(G)$ maps all vertices to vectors in some finite set $S^{(i)} = \{\boldsymbol{s}_1, \dots, \boldsymbol{s}_m\} \subseteq \mathbb{R}^q$. Importantly, this set $S^{(i)}$ is independent of the input graph $G \in \mathcal{G}_{\{0,1\}^\ell}$, it only depends on ℓ and the GNN. In a one-hot encoding of the set $S^{(i)}$ we represent the element $\boldsymbol{s}_j$ by the j-th unit vector $\boldsymbol{e}_j \in \mathbb{R}^m$. We can construct a 1-GNN N' that simulates the computation of N, but represents all states $N^{(i)}(G,u)$ by their one-hot encoding. We only need a 1-GNN with identity messages for this, because aggregating, that is, taking the coordinatewise maximum of, the one-hot encoded states of the neighbours gives a node the full set of states appearing at the neighbours. Since max-aggregation is not sensitive to multiplicities, this is sufficient to reconstruct the messages of the original 2-GNN N and simulate the necessary computations in the combination function.

If we try to prove the assertion for mean-aggregation similarly, we already fail at the first step. The range of the feature maps computed by a Mean-GNN on inputs $G \in \mathcal{G}_{\{0,1\}^\ell}$ is infinite in general. The trick to resolve this is to use finite-valued approximations and then use one-hot encodings of these. The reason that this scheme works is that the functions computed by MLPs are Lipschitz continuous, and that mean-aggregation (as opposed to sum-aggregation) keeps the approximation error bounded.

We introduce additional notations that we use in the proof. For $n \in \mathbb{N}_{>0}$, denote by $\mathbf{1}_{n,i}$ the vector of length n with 1 in the ith position and 0 elsewhere. Let $\mathcal{X} \subseteq \mathbb{R}^p$ be a finite set and assume an enumeration of it $\mathcal{X} = \boldsymbol{x}_1, \dots, \boldsymbol{x}_n$. For all $i \in [n]$ denote by $\mathbf{1}_\mathcal{X}(\boldsymbol{x}_i) \coloneqq \mathbf{1}_{n,i}$ the one-hot representation of $\boldsymbol{x}_i$. Denote by $\mathbf{1}_\mathcal{X}^{-1}$ the inverse function of $\mathbf{1}_\mathcal{X}$, that is, $\mathbf{1}_\mathcal{X}^{-1}(1_{n,i}) \coloneqq \boldsymbol{x}_i$. For a multiset $M \in \left(\!\!\binom{\mathcal{X}}{*}\!\!\right)$ we define $\mathbf{1}_\mathcal{X}(M) \coloneqq \{\!\{\mathbf{1}_\mathcal{X}(\boldsymbol{x}) \mid \boldsymbol{x} \in M\}\!\}$ the corresponding multiset of one-hot representations. For every two vectors $\boldsymbol{u} = (u_1, \dots, u_k), \boldsymbol{v} = (v_1, \dots, v_k) \in \mathbb{R}^k$ we define $\|\boldsymbol{u} - \boldsymbol{v}\| \coloneqq \|\boldsymbol{u} - \boldsymbol{v}\|_\infty = \mathsf{max}_{i\in[k]}|u_i - v_i|$. Moreover, we let $\mathsf{mean}(\boldsymbol{u}) \coloneqq \frac{1}{k}\Sigma_{i\in[k]}u_i$ and $\mathsf{max}(\boldsymbol{u}) = \mathsf{max}_{i\in[k]}u_i$. We define an operator U that removes the duplicates in a multiset, that is, for a multiset M we have $U(M) \coloneqq \{x : x \in M\}$.

Let N be a GNN, for every featured graph G, every $v \in V(G)$, and every $i \in [d]$, we define

$$N_N^{(i)}(v) \coloneqq \{\!\{w_N^{(i)} : w \in N_G(v)\}\!\}$$

the multiset of features of the neighbors of v after operating the first i layers of N.

Proof for MAX.
Let

$$N = (L^{(1)}, \ldots, L^{(d)})$$

be a d-layer 2-GNN where $L^{(i)} = (\mathsf{comb}_i, \mathsf{max}, \mathsf{msg}_i)$, $L^{(i)}$ has input and output dimensions $p^{(i)}, q^{(i)}$, and msg_i has output dimension $r^{(i)}$. We define $D_N^{(0)} \coloneqq \{0,1\}^{p^{(1)}}$ the set of possible input boolean-feature values and for $i \in [d]$ we define

$$D_N^{(i)} \coloneqq \{N^{(i)}(G, v) : G \in \mathcal{G}_{\{0,1\}^{p^{(1)}}}, v \in V(G)\}$$

the set of possible feature values after operating the first i layers of N when the initial initial-feature is boolean. For $i \in [0..d]$ define $d_N^{(i)} \coloneqq \left|D_N^{(i)}\right| \in (\mathbb{N} \cup \{\infty\})$ the size of $D_N^{(i)}$. Assume $i \in [d[$, assume $D_N^{(i)}$ is finite, and assume an enumeration $D_N^{(i)} = x_1, \ldots, x_{d_N^{(i)}}$.

We define $\mathfrak{m}_0 : 1_{D_N^{(i)}}(D_N^{(i)}) \times \{0,1\}^{d_N^{(i)}} \to D_N^{(i)} \times 2^{D_N^{(i)}}$ such that

$$\forall (s,t) \in 1_{D_N^{(i)}}(D_N^{(i)}) \times \{0,1\}^{d_N^{(i)}} \quad \mathfrak{m}_0(s,t) \coloneqq (1^{-1}_{D_N^{(i)}}(s), \{x_i : t(i) = 1\})$$

and note that for every $G \in \mathcal{G}_{\{0,1\}^{p^{(1)}}}, v \in V(G)$

$$\mathfrak{m}_0(1_{D_N^{(i)}}(v_N^{(i)}), \mathsf{max}(1_{D_N^{(i)}}(N_N^{(i)}(v)))) = (v_N^{(i)}, U(N_N^{(i)}(v)))$$

We define $\mathfrak{m}_1 : D_N^{(i)} \times 2^{D_N^{(i)}} \to \mathbb{R}^{r^{(i+1)}}$ such that for every $G \in \mathcal{G}_{\{0,1\}^{p^{(1)}}}, v \in V(G)$

$$\mathfrak{m}_1(v_N^{(i)}, U(N_N^{(i)}(v))) \coloneqq \mathsf{max}\Big(\mathsf{msg}_{i+1}(v_N^{(i)}, x) : x \in U\big(N_N^{(i)}(v)\big)\Big)$$

Finally, we define $\mathfrak{m}_2 : 1_{D_N^{(i)}}(D_N^{(i)}) \times \{0,1\}^{d_N^{(i)}} \to \mathbb{R}^{r^{(i+1)}}$ such that

$$\mathfrak{m}_2 \coloneqq \mathfrak{m}_1 \circ \mathfrak{m}_0$$

and have that for every $G \in \mathcal{G}_{\{0,1\}^{p^{(1)}}}, v \in V(G)$

$$\begin{aligned}&\mathfrak{m}_2(1_{D_N^{(i)}}(v_N^{(i)}), \mathsf{max}(1_{D_N^{(i)}}(N_N^{(i)}(v)))) = \mathsf{max}(\mathsf{msg}_{i+1}(v_N^{(i)}, x) : x \in U(N_N^{(i)}(v)))\\ &= \mathsf{max}(\mathsf{msg}_{i+1}(v_N^{(i)}, x) : x \in N_N^{(i)}(v)))\end{aligned}$$

The last equality is key in determining that the domain of the comb function is finite although there are infinitely many possible multiset-inputs to the max aggregation. By assumption that $D_N^{(i)}$ is finite, the domain of $\mathfrak{m}_2$ is finite and there exists an MLP that implements $\mathfrak{m}_2$. Denote that MLP by $\mathcal{F}$, denote the function it implements by $f_{\mathcal{F}}$, and note that comb_{i+1} is implemented by an MLP, then there exists an MLP that implements

$$\mathsf{comb}'_{i+1} : \{0,1\}^{d_N^{(i)}} \times \{0,1\}^{d_N^{(i)}} \to D_N^{(i+1)}$$

such that for every $G \in \mathcal{G}_{\{0,1\}^{p^{(1)}}}, v \in V(G)$

$$\begin{aligned}&\mathsf{comb}'_{i+1}(1_{D_N^{(i)}}(v_N^{(i)}), \mathsf{max}(1_{D_N^{(i)}}(N_N^{(i)}(v)))) \coloneqq \\ &\mathsf{comb}_{i+1}(v_N^{(i)}, f_{\mathscr{I}}(1_{D_N^{(i)}}(v_N^{(i)}), \mathsf{max}(1_{D_N^{(i)}}(N_N^{(i)}(v))))) = \\ &\mathsf{comb}_{i+1}(v_N^{(i)}, \mathsf{max}(\mathsf{msg}(v_N^{(i)}, x) : x \in N_N^{(i)}(v))) = v_N^{(i+1)}\end{aligned} \tag{4.3.5}$$

Note that (assuming $D_N^{(i)}$ is finite) the domain of comb'_{i+1} is finite, hence $D_N^{(i+1)}$ is finite. Since $D_N^{(0)}$ is finite by definition, we have by induction that $D_N^{(i)}$ is finite $\forall i \in [d+1[$ and there exists comb'_{i+1} as described. For $i \in [d-1[$ we can modify the comb'_{i+1}-implementing MLP into a comb''_{i+1}-implementing MLP that outputs $1_{D_N^{(i+1)}}(v_N^{(i+1)})$ instead of $v_N^{(i+1)}$.

Let comb''_0 be an MLP (which clearly exists) such that for every $G \in \mathcal{G}_{\{0,1\}^{p^{(1)}}}, v \in V(G)$ it holds that $\mathsf{comb}''_0\Big(v_N^{(0)}, \mathsf{max}\big(w_N^{(0)} \mid w \in N_G(v)\big)\Big) = 1_{D_N^{(0)}}(Z(G)(v))$, and define a 1-GNN layer $L'^{(0)} \coloneqq \{\mathsf{comb}''_0, \mathsf{max}\}$. Then, define

$$\forall i \in [d-1] \;\; L'^{(i)} \coloneqq (\mathsf{comb}''_i, \mathsf{max}), \quad L'^{(d)} \coloneqq (\mathsf{comb}'_d, \mathsf{max})$$

Finally, define $N' \coloneqq (L'^{(0)}, \ldots, L'^{(d)})$, then

$$\forall G \in \mathcal{G}_{\{0,1\}^{p^{(1)}}}, v \in V(G) \quad N'(G,v) = N(G,v)$$

□

Proof for MEAN.
We define $\frac{1}{\mathbb{N}_{>0}} \coloneqq \{\frac{1}{a} : a \in \mathbb{N}_{>0}\}$. For every $\delta \in \frac{1}{\mathbb{N}_{>0}}$ and $n \in \mathbb{N}$, we define

$$S_{n,\delta} \coloneqq \{(p_1, \ldots, p_n) : \Sigma p_i = 1, \; \forall i \in [n] \; \exists b \in \mathbb{N} : p_i = b\delta\}$$

the set of $\binom{n+\frac{1}{\delta}-1}{n-1}$ possible elements' proportions in a multiset over a set of size n, s.t. the proportion of each element is a multiple of δ. Let $N = (L^{(1)}, \ldots, L^{(d)})$ be a d-layer 2-GNN where

$$L^{(i)} = (\mathsf{comb}_i, \mathsf{mean}, \mathsf{msg}_i)$$

and $L^{(i)}$ has input and output dimensions $p^{(i)}, q^{(i)}$. Let $\mathcal{X} \in \left(\!\!\binom{\mathbb{R}^{p^{(i)}}}{*}\!\!\right)$ be a finite multiset, and assume an enumeration $\mathcal{X} = x_1, \ldots, x_n$, then for $y \in \mathbb{R}^{p^{(i)}}$ we denote by

$$\mathsf{Msg}_{\mathcal{X}}^{(i)}(y) \coloneqq (\mathsf{msg}_i(y, x_1), \ldots, \mathsf{msg}_i(y, x_n))$$

the vector of messages between y and every element in $\mathcal{X}$. We prove by induction on d that for every $\varepsilon > 0$ there exists a 1-GNN $N' = L'^{(0)}, L'^{(1)}, \ldots, L'^{(d)}$ and a finite set $\mathcal{X}^{(d)}$ such that

$$\forall G \in \mathcal{G}_{\{0,1\}^{p^{(1)}}} \; \forall v \in V(G) \; \exists x_v \in \mathcal{X}^{(d)} : \; \big|N(G,v) - x_v\big| < \frac{\varepsilon}{2} \; , \; \big|N'(G,v) - x_v\big| < \frac{\varepsilon}{2}$$

which implies

$$\big|N(G,v) - N'(G,v)\big| < \varepsilon$$

The layers $L'^{(i)}, i \in [d]$ of the 1-GNN are designed to operate on one-hot representations, and the layer $L'^{(0)}$ translates the initial-feature range to its one-hot representation. The $\mathcal{X}^{(i)}$s are the "make-believe" finite domains by which we will design the layers of the 1-GNN. **Induction Basis** We prove for $d = 1$. Let $\varepsilon > 0$. Define $\mathcal{X} \coloneqq \{0,1\}^{p^{(1)}}$, $n \coloneqq \left|\mathcal{X}\right|$, and assume an enumeration $\mathcal{X} = x_1, \ldots, x_n$. Note that for every $G \in \mathcal{G}_{\{0,1\}^{p^{(1)}}}, v \in V(G)$

$$\mathsf{mean}(\mathsf{msg}_1(v_N^{(0)}, x) : x \in N_N^{(0)}(v)) = \mathsf{Msg}_{\mathcal{X}}^{(1)}(v_N^{(0)}) \cdot \mathsf{mean}(1_{\mathcal{X}}(N_N^{(0)}(v))) \tag{4.3.6}$$

That is, the mean of the messages is equal to the product of the vector of possible messages and the vector of corresponding proportions of the elements (of $\mathcal{X}$) among the neighbors of v. By $\mathcal{X}$ being finite, there exists $g_1 : \mathbb{R} \to \mathbb{R}$, $\lim_{x\to 0} g_1(x) = 0$ such that $\forall G \in \mathcal{G}_{\{0,1\}^{p^{(1)}}}\ \forall v \in V(G)\ \forall \delta > 0\ \forall u \in \mathbb{R}^{r^{(i)}}$

$$\begin{aligned}\left|u - \mathsf{mean}(1_{\mathcal{X}}(N_N^{(0)}(v)))\right| \leq \delta \Rightarrow \\ |\mathsf{Msg}_{\mathcal{X}}^{(1)}(v_N^{(0)}) \cdot \mathsf{mean}(1_{\mathcal{X}}(N_N^{(0)}(v))) - \mathsf{Msg}_{\mathcal{X}}^{(1)}(v_N^{(0)}) \cdot u| < g_1(\delta)\end{aligned}$$

Hence, by comb_1 being Lipschitz-Continuous, and remembering Equation (4.3.6), there exists $g_2 : \mathbb{R} \to \mathbb{R}$, $\lim_{x\to 0} g_2(x) = 0$ such that $\forall G \in \mathcal{G}_{\{0,1\}^{p^{(1)}}}\ \forall v \in V(G)\ \forall \delta > 0\ \forall u \in \mathbb{R}^{r^{(i)}}$

$$\begin{aligned}&\left|u - \mathsf{mean}(1_{\mathcal{X}}(N_N^{(0)}(v)))\right| \leq \delta \Rightarrow \\ &|\mathsf{comb}_1(v_N^{(0)}, \mathsf{mean}(\mathsf{msg}_1(v_N^{(0)}, x) : x \in N_N^{(0)}(v))) - \mathsf{comb}_1(v_N^{(0)}, \mathsf{Msg}_{\mathcal{X}}^{(1)}(v_N^{(0)}) \cdot u)| < g_2(\delta_1)\end{aligned} \tag{4.3.7}$$

Let $\delta_1 \in \frac{1}{\mathbb{N}_{>0}}$ such that $\forall \delta \leq \delta_1\ g_2(\delta) < \frac{\varepsilon}{2}$. Let $G \in \mathcal{G}_{\{0,1\}^{p^{(1)}}}, v \in V(G)$ and let $\boldsymbol{p}_v \in S_{n,\delta_1}$ be proportions - which must exist - such that $\left|\boldsymbol{p}_v - \mathsf{mean}(1_{\mathcal{X}}(N_N^{(0)}(v)))\right| \leq \delta_1$, then,

$$|\mathsf{comb}_1(v_N^{(0)}, \mathsf{mean}(\mathsf{msg}_1(v_N^{(0)}, x) : x \in N_N^{(0)}(v))) - \mathsf{comb}_1(v_N^{(0)}, \mathsf{Msg}_{\mathcal{X}}^{(1)}(v_N^{(0)}) \cdot \boldsymbol{p}_v)| < \frac{\varepsilon}{2} \tag{4.3.8}$$

Note that the first expression in the LHS of Equation (4.3.8) is $N(G, v)$. We proceed to show a similar result for N'. For $\delta_2 \in \frac{1}{\mathbb{N}_{>0}}$ define $\mathsf{comb}'_1 : \{0,1\}^n \times \mathbb{R}^n \to \mathbb{R}^{q^{(1)}}$ such that

$$\forall (x, \boldsymbol{p}) \in 1_{\mathcal{X}}(\mathcal{X}) \times S_{n,\delta_2}\ \ \mathsf{comb}'_1(x, \boldsymbol{p}) \coloneqq \mathsf{comb}_1(1_{\mathcal{X}}^{-1}(x), \mathsf{Msg}_{\mathcal{X}}^{(1)}(1_{\mathcal{X}}^{-1}(x)) \cdot \boldsymbol{p})$$

While the domain on which comb'_1 is applicable is infinite, the domain for which it is explicitly specified is finite, hence comb'_1 can be implemented by an MLP. By argumentation similar to the development of Equation (4.3.7) we have that there exists

$$g_3 : \mathbb{R} \to \mathbb{R}, \lim_{x\to 0} g_3(x) = 0$$

such that $\forall G \in \mathcal{G}_{\{0,1\}^{p^{(1)}}}\ \forall v \in V(G)\ \forall \delta_2 > 0\ \forall u \in \mathbb{R}^{r^{(i)}}$

$$\begin{aligned}&\left|u - \mathsf{mean}(1_{\mathcal{X}}(N_N^{(0)}(v)))\right| \leq \delta_2 \Rightarrow \\ &\quad |\mathsf{comb}'_1(1_{\mathcal{X}}(v_N^{(0)}), \mathsf{mean}(1_{\mathcal{X}}(N_N^{(0)}(v)))) - \mathsf{comb}'_1(1_{\mathcal{X}}(v_N^{(0)}), u)| < g_3(\delta_2)\end{aligned} \tag{4.3.9}$$

Let $\delta_2 \in \frac{1}{\mathbb{N}_{>0}}$ such that $\forall \delta \leq \delta_2 \ g_3(\delta) < \frac{\varepsilon}{2}$. Let $G \in \mathcal{G}_{\{0,1\}^{p^{(1)}}}, v \in V(G)$ and let $\boldsymbol{p}_v \in S_{n,\delta_2}$ be proportions - which must exist - such that $\left|\boldsymbol{p}_v - \mathsf{mean}(1_{\mathcal{X}}(N_N^{(0)}(v)))\right| \leq \delta_2$, then,

$$|\mathsf{comb}'_1(1_{\mathcal{X}}(v_N^{(0)}), \mathsf{mean}(1_{\mathcal{X}}(N_N^{(0)}(v)))) - \mathsf{comb}'_1(1_{\mathcal{X}}(v_N^{(0)}), \boldsymbol{p}_v)| < \frac{\varepsilon}{2} \tag{4.3.10}$$

Let comb'_0 be an MLP (which clearly exists) such that for every $G \in \mathcal{G}_{\{0,1\}^{p^{(1)}}}, v \in V(G)$ it holds that $\mathsf{comb}'_0\Big(v_N^{(0)}, \mathsf{mean}\big(w_N^{(0)} \mid w \in N_G(v)\big)\Big) = 1_{\mathcal{X}}(Z(G)(v))$, and define a 1-GNN layer $L'^{(0)} \coloneqq \{\mathsf{comb}'_0, \mathsf{mean}\}$. Define $L'^{(1)} \coloneqq (\mathsf{comb}'_1, \mathsf{mean})$ and define

$$N' = (L'^{(0)}, L'^{(1)})$$

Then, note that the first expression in the LHS of Equation (4.3.10) is $N'(G, v)$. Define $\delta_3 = min(\delta_1, \delta_2)$, then by Equation (4.3.8) and Equation (4.3.10), for every $G \in \mathcal{G}_{\{0,1\}^{p^{(1)}}}, v \in V(G)$ there exists $\boldsymbol{p}_v \in S_{n,\delta_3}$ such that for

$$x_v \coloneqq \mathsf{comb}_1(v_N^{(0)}, \mathsf{Msg}_{\mathcal{X}}^{(1)}(v_N^{(0)}) \cdot \boldsymbol{p}_v)$$

it holds that

$$|N(G, v) - x_v| < \frac{\varepsilon}{2}, \ \ |N'(G, v) - x_v| < \frac{\varepsilon}{2}$$

Hence, define

$$\mathcal{X}^{(1)} \coloneqq \{\mathsf{comb}_1(y, \mathsf{Msg}_{\mathcal{X}}^{(1)}(y) \cdot \boldsymbol{p}) : (y, \boldsymbol{p}) \in \mathcal{X} \times S_{n,\delta_3}\}$$

, then $\mathcal{X}^{(1)}$ satisfies the requirement.

Induction Step We assume correctness for $d = t$ and every $\varepsilon' > 0$ and prove for $d = t + 1$. Define $\mathcal{X} \coloneqq \mathcal{X}^{(t)}, n \coloneqq \left|\mathcal{X}\right|$, and assume an enumeration $\mathcal{X} = x_1, \ldots, x_n$. Let $N'' = (L''^{(0)}, \ldots, L''^{(t)}), L''^{(i)} = (\mathsf{comb}''_i, \mathrm{avg})$, be a 1-GNN - that exists by the induction assumption - such that

$$\forall G \in \mathcal{G}_{\{0,1\}^{p^{(1)}}} \ \forall v \in V(G) \ \left|v_N^{(t)} - N''(G, v)\right| < \varepsilon'$$

and denote by $v^{\mathcal{X}}$ a value in $\mathcal{X}$ - which exists by the induction assumption - such that $\left|v_N^{(t)} - v^{\mathcal{X}}\right| < \frac{\varepsilon'}{2}, \left|N''(G, v) - v^{\mathcal{X}}\right| < \frac{\varepsilon'}{2}$. Compared to the case of $d = 1$, in the case of $d > 1$ we need to account not only for differences (which can be made as small as we want) between the actual proportions and the proportions that $L'^{(t+1)}$ will be designed for, but also for $\left|v_N^{(t)} - v^{\mathcal{X}}\right|$ i.e. the difference between the actual vertices' values and the values in $\mathcal{X}$ - the domain that $L'^{(t+1)}$ will be designed for. We handle the proportions difference first, using the already walked-through path used in the proof for $d = 1$. For $\delta_1 \in \frac{1}{\mathbb{N}_{>0}}$ define $\mathsf{comb}'_{t+1} : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^{q^{(t+1)}}$ such that

$$\forall (x, \boldsymbol{p}) \in 1_{\mathcal{X}}(\mathcal{X}) \times S_{n,\delta_1} \ \ \mathsf{comb}'_{t+1}(x, \boldsymbol{p}) \coloneqq \mathsf{comb}_{t+1}(1_{\mathcal{X}}^{-1}(x), (\mathsf{Msg}_{\mathcal{X}}^{(t+1)}(1_{\mathcal{X}}^{-1}(x)) \cdot \boldsymbol{p}))$$

As explained in the case of $d = 1$, such comb'_{t+1} can be implemented by an MLP. Also, by the same argumentation used in the case of $d = 1$, there exists $\delta_1 \in \frac{1}{\mathbb{N}_{>0}}$ such that for every $G \in \mathcal{G}_{\{0,1\}^{p^{(1)}}}, v \in V(G)$ there exists $\boldsymbol{p}_v \in S_{n,\delta_1}$ such that for

$$x_v \coloneqq \mathsf{comb}_{t+1}(v^{\mathcal{X}}, \mathsf{Msg}_{\mathcal{X}}^{(t+1)}(v^{\mathcal{X}}) \cdot \boldsymbol{p}_v)$$

it holds that

$$\left|\mathsf{comb}_{t+1}(v^{\mathcal{X}}, \mathsf{mean}(\mathsf{msg}_{t+1}(v^{\mathcal{X}}, w^{\mathcal{X}}) : w \in N_v(G))) - x_v\right| < \frac{\varepsilon}{4} \tag{4.3.11}$$

and

$$\left|\mathsf{comb}'_{t+1}(1_{\mathcal{X}}(v^{\mathcal{X}}), \mathsf{mean}(1_{\mathcal{X}}(w^{\mathcal{X}}) : w \in N_v(G))) - x_v\right| < \frac{\varepsilon}{4} \tag{4.3.12}$$

Next, we turn to the differences between the actual vertices' values (after layer t) and the values in $\mathcal{X}$ - the finite domain according to which $L'^{(t+1)}$ is defined. We start with bounding the gap for the 2-GNN. By msg_{t+1} being Lipschitz-Continuous, there exists $g_1 : \mathbb{R} \to \mathbb{R}$, $\lim_{x\to 0} g_1(x) = 0$ such that $\forall G \in \mathcal{G}_{\{0,1\}^{p^{(1)}}}\ \forall v, w \in V(G)$

$$\left|\mathsf{msg}_{t+1}(v_N^{(t)}, w_N^{(t)}) - \mathsf{msg}_{t+1}(v^{\mathcal{X}}, w^{\mathcal{X}})\right| < g_1(\varepsilon')$$

, and thus, such that $\forall G \in \mathcal{G}_{\{0,1\}^{p^{(1)}}}\ \forall v \in V(G)$

$$|\mathsf{mean}(\mathsf{msg}_{t+1}(v_N^{(t)}, w_N^{(t)}) : w \in N_v(G)) -$$

$$\mathsf{mean}(\mathsf{msg}_{t+1}(v^{\mathcal{X}}, w^{\mathcal{X}}) : w \in N_v(G))| < g_1(\varepsilon')$$

By comb_{t+1} being Lipschitz-Continuous, we have that there exists $g_2 : \mathbb{R} \to \mathbb{R}, \lim_{x\to 0} g_2(x) = 0$ such that

$$\forall G \in \mathcal{G}_{\{0,1\}^{p^{(1)}}}\ \forall v \in V(G)$$

$$|\mathsf{comb}_{t+1}(v_N^{(t)}, \mathsf{mean}(\mathsf{msg}_{t+1}(v_N^{(t)}, w_N^{(t)}) : w \in N_v(G))) -$$
$$\mathsf{comb}_{t+1}(v^{\mathcal{X}}, \mathsf{mean}(\mathsf{msg}_{t+1}(v^{\mathcal{X}}, w^{\mathcal{X}}) : w \in N_v(G)))| < g_2(\varepsilon')$$

Assuming ε' small enough such that $\forall \varepsilon'' < \varepsilon'\ \ g_2(\varepsilon'') < \frac{\varepsilon}{4}$ we have

$$\begin{aligned} &\forall G \in \mathcal{G}_{\{0,1\}^{p^{(1)}}}\ \forall v \in V(G) \\ &\quad |\mathsf{comb}_{t+1}(v_N^{(t)}, \mathsf{mean}(\mathsf{msg}_{t+1}(v_N^{(t)}, w_N^{(t)}) : w \in N_v(G))) - \\ &\qquad \mathsf{comb}_{t+1}(v^{\mathcal{X}}, \mathsf{mean}(\mathsf{msg}_{t+1}(v^{\mathcal{X}}, w^{\mathcal{X}}) : w \in N_v(G)))| < \frac{\varepsilon}{4} \end{aligned} \tag{4.3.13}$$

Next, we bound the gap for the 1-GNN. Define $h : \mathcal{X} \to 1_{\mathcal{X}}(\mathcal{X})$ such that $\forall x \in \mathcal{X}\ \ h(x) = 1_{\mathcal{X}}(x)$, a mapping from the values-domain to its one-hot representation, which can be implemented by an MLP. Define $N' \coloneqq (L'^{(0)}, \ldots, L'^{(t)})$ such that $\forall 0 \leq i \leq (t-1)\ L'^{(i)} \coloneqq L''^{(i)}$, and $L'^{(t)} = \{\mathsf{comb}'_t, \mathrm{avg}\}$ where

$$\mathsf{comb}'_t \coloneqq h \circ \mathsf{comb}''_t$$

By the induction assumption, for every $G \in \mathcal{G}_{\{0,1\}^{p^{(1)}}}, v \in V(G)\ \left|N''(G,v) - v^{\mathcal{X}}\right| < \frac{\varepsilon'}{2}$. Hence, by (every MLP-implementation of) h being Lipschitz-Continuous, there exists $g_3 : \mathbb{R} \to \mathbb{R}$, $\lim_{x\to 0} g_3(x) = 0$ such that for every $G \in \mathcal{G}_{\{0,1\}^{p^{(1)}}}, v \in V(G)$

$$\left|N'(G,v) - 1_{\mathcal{X}}(v^{\mathcal{X}})\right| < g_3(\varepsilon')$$

, and thus, such that for every $G \in \mathcal{G}_{\{0,1\}^{p^{(1)}}}, v \in V(G)$

$$|\mathsf{mean}(N'(G,w) : w \in N_v(G)) - \mathsf{mean}(1_{\mathcal{X}}(w^{\mathcal{X}}) : w \in N_v(G))| < g_3(\varepsilon')$$

Hence, by comb'_{t+1} being Lipschitz-Continuous, there exists

$$g_4 : \mathbb{R} \to \mathbb{R}, \lim_{x \to 0} g_4(x) = 0$$

such that for every $G \in \mathcal{G}_{\{0,1\}^{p^{(1)}}}, v \in V(G)$

$$|\mathsf{comb}'_{t+1}(N'(G,v), \mathsf{mean}(N'(G,w) : w \in N_v(G))) -$$

$$\mathsf{comb}'_{t+1}(1_{\mathcal{X}}(v^{\mathcal{X}}), \mathsf{mean}(1_{\mathcal{X}}(w^{\mathcal{X}}) : w \in N_v(G)))| < g_4(\varepsilon')$$

Assuming ε' small enough such that $\forall \varepsilon'' < \varepsilon' \quad g_4(\varepsilon'') < \frac{\varepsilon}{4}$ we have that for every $G \in \mathcal{G}_{\{0,1\}^{p^{(1)}}}, v \in V(G)$

$$\begin{aligned} &|\mathsf{comb}'_{t+1}(N'(G,v), \mathsf{mean}(N'(G,w) : w \in N_v(G))) - \\ &\mathsf{comb}'_{t+1}(1_{\mathcal{X}}(v^{\mathcal{X}}), \mathsf{mean}(1_{\mathcal{X}}(w^{\mathcal{X}}) : w \in N_v(G)))| < \frac{\varepsilon}{4} \end{aligned} \tag{4.3.14}$$

Define $L'^{(t+1)} \coloneqq (\mathsf{comb}'_{t+1}, \mathsf{mean},)$ and redefine N' to include $L'^{(t+1)}$, that is,

$$N' \coloneqq (L'^{(0)}, \ldots, L'^{(t)}, L'^{(t+1)})$$

Combining Equation (4.3.11) with Equation (4.3.13), and Equation (4.3.12) with Equation (4.3.14) we have that for every $G \in \mathcal{G}_{\{0,1\}^{p^{(1)}}}, v \in V(G)$

$$|N(G,v) - x_v| < \frac{\varepsilon}{2}\ ,\ |N'(G,v) - x_v| < \frac{\varepsilon}{2}$$

Hence, define

$$\mathcal{X}^{(t+1)} \coloneqq \{\mathsf{comb}_{t+1}(y, \mathsf{Msg}_{\mathcal{X}}^{(t+1)}(y) \cdot \boldsymbol{p}) : (y, \boldsymbol{p}) \in \mathcal{X} \times S_{n,\delta_1}\}$$

then $\mathcal{X}^{(t+1)}$ satisfies the requirement. □

4.4 Summary

We study the expressivity of graph neural networks with 1-sided and 2-sided message passing. The picture we see is surprisingly complicated: We show that in contrast to the non-uniform setting, for which it is shown in [GR24] that 1-GNNs and 2-GNNs have the same expressivity, in the uniform setting we can separate 1-GNNs from 2-GNNs, yet only if they use sum-aggregation. Moreover, we introduce the notion of fast convergence, which allows us to extend our result to activation functions other than relu.

Some questions remain open. The query that we use to separate 2-GNNs from 1-GNNs with sum-aggregation turns out to be expressible by 1-GNNs that use both sum and mean. Is there also a query that is expressible by 2-GNNs, but not by 1-GNNs with sum and mean, or even sum, mean, and max aggregation?

Another important question concerns the size inflation in the construction of Mean 1-GNNs and Max 1-GNNs that emulate their 2-GNNs counterparts. The uniform setting means that the asymptotic size complexity is not affected by that inflation as it is independent of the graph size, however it is a substantial inflation depending on the 2-GNNs parameters, that may deem the emulation impractical. As we have not proven a lower-bound for the construction size, the question whether there is a more efficient construction remains open.

Global Affairs

> "But while I believe that optimism is appropriate... I believe optimism must be tempered with prudence."
>
> Ronald Reagan

5.1 Intro

This chapter is based on [Ros+24], excluding Section 5.2 there ('Experiments On Benchmark Datasets'). So far, we have compared different variations of the GNN architecture, first studying the role of the aggregation-function when the message function is trivial, and then studying the role of the message function when considering common aggregation functions. In this chapter, we compare two architectures that augment the message passing scheme - local computations - with global computations, in a finite initial-feature domain setting. At each layer, in parallel to the message passing computation there is a global computation, and the outputs of the two are combined to create the new value of the vertex. The message passing part of each layer consists of a trivial message function - outputting the neighbor's value - and an aggregation that can be either sum or avg.

In the first architecture, which we refer to as *GNN+VN*, the global computation is a *virtual node* [Gil+17b] i.e. an aggregation of all the nodes' values, possibly followed by an MLP. It is a single computation whose output is used by all vertices - and not a vertex-specific computation - hence it runs in linear time for all vertices together. In the second architecture, a specific kind of a *graph transformer* [DB20] known as *GPS* [Ram+22], the global computation is a vertex-specific computation: It is a *self-attention* [Vas+17] between the subject vertex and all other vertices in the graph. Being vertex-specific, it runs in quadratic time for all vertices together. In return, its sophistication and proven success in other machine-learning algorithms - most notably LLMs - suggest that it may be highly expressive, and that GPS possibly subsumes GNN+VN. Independently from the architecture's pure definition, it is common to use it in conjunction with adding *positional encodings* to the vertices' initial features, which may improve expressivity. In our analysis we consider also such combinations.

Previous works have shown that under the non-uniform notion, graph transformers are universal function approximators [Kre+21] when adding positional encodings such as LapPE [DB20]. Although non-uniform expressivity is not the focus of this work, we take the opportunity to formally clarify that such universality is not due to unique properties of graph transformers: More primitive machine-learning algorithms are also universal

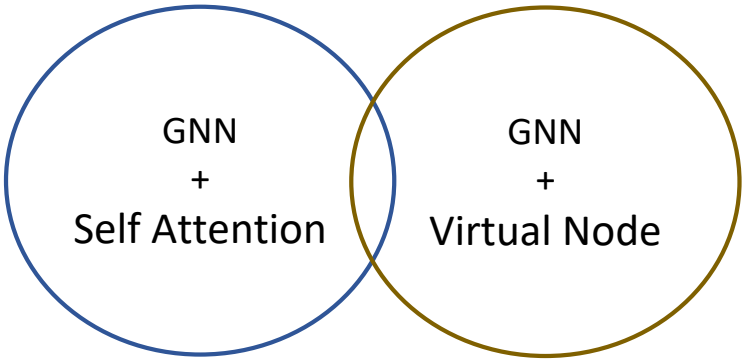


Figure 5.1: The expressivity relation between GNNs + virtual nodes and GNNs + Self-Attention, for finite initial-feature domain.

approximators, in the non-uniform notion, when augmenting the vertices with certain positional encodings.

5.1.1 New Results

All our resultrs are in the finite initial-feature domain setting.

1. Starting with the clarification for the non-uniform setting, we prove that shallow GNNs and even MLPs are universal approximators when augmenting the vertices with certain positional encodings.
2. We prove that, under the uniform notion, universal approximation is impossible even when having positional encodings.
3. We prove that (under the uniform notion) GPS + positional encodings do not subsume GNN+VN. That is, there exists a function that is expressible by the latter and not by the former.
4. We prove that (under the uniform notion) GNNs+VN do not subsume GPS. That is, there exists a function that is expressible by the latter and not by the former.

Results 3,4 are illustrated in Figure 5.1

We complement our theoretical findings with an empirical study on synthetic datasets and observe that a uniformly-expressive model can (sometimes) be learned, and a uniform-inexpressive model does not generalize in terms of graph size.

5.2 Terms and Concepts

Let $G \in \mathcal{G}_*$ be a featured graph. We assume that the vertices in $V(G)$ are named $1, \ldots, |V(G)|$. Then, we denote $Z(G)(i)$ by $Z(G)_i$, and in general for every mapping of the vertices $f(G)$, we denote $f(G)(i)$ by $f(G)_i$. Our notations for the various components of the architectures have a double meaning - depending on the context: They represent the component itself as well as the function that it defines.

We define two algorithm building blocks that are used both in GNN+VN and GPS:

1. A *message passing gadget* (MPG) $M = (\mathsf{comb}, \mathsf{agg})$ of dimension p is a pair where comb is an MLP of I/O dimensions $2p; p$, and $\mathsf{agg} \in \{\mathrm{sum}, \mathrm{avg}, \mathsf{max}\}$ is an aggregation function. It defines a function $M : \mathcal{G}_{\mathbb{R}^p} \to \mathcal{Z}_{\mathbb{R}^p}$ from a featured graph to a new feature map:
$$M(G)_v \coloneqq \mathsf{comb}\big(Z(G)_v, \mathsf{agg}(\{\{Z(G)_u \mid u \in N_G(v)\}\})\big)$$
2. A *readout gadget* $R = (\mathsf{mlp}, \mathsf{agg})$ of dimensions $p; q$, is a pair such that: mlp is an MLP of I/O dimensions $p; q$, and $\mathsf{agg} \in \{\mathrm{sum}, \mathrm{avg}\}$ is an aggregation function. It defines a function $R : \binom{\mathbb{R}^p}{*} \to \mathbb{R}^q$ from a multiset of p-dimensional vectors to a q-dimensional vector:
$$R(X) \coloneqq \mathsf{mlp}(\mathsf{agg}(X))$$

5.2.1 Message Passing + Virtual Node

We start with defining GNN+VN. First, we define a single layer of it. An *MPG+V* $L = (M, R)$ of dimension p is a pair of a message passing gadget of dimension p, and a readout gadget of dimensions $p; p$. It defines a function $L : \mathcal{G}_{\mathbb{R}^p} \to \mathcal{Z}_{\mathbb{R}^p}$ from a featured graph to a new feature map, as follows. Define

$$g(G)_i \coloneqq M(G)_i + Z(G)_i$$

then

$$L(G)_i \coloneqq R(g(G)) + g(G)_i$$

The readout part of the algorithm is what we refer to as the virtual node. Next, we define the complete architecture. A GNN+VN $B = (P, B_1, \ldots, B_m, R)$ of dimensions $p; q; r$ is a tuple, where P is a preparation MLP of I/O dimensions $p; q$, for every $i \in [m]$ it holds that B_i is an MPG+V layer of dimension q, and R is a final readout gadget of dimensions $q; r$. It defines a graph embedding $B : \mathcal{G}_{\mathbb{R}^p} \to \mathbb{R}^r$ as follows: For $G \in \mathcal{G}_{\mathbb{R}^p}$ define

$$G^{(0)} \coloneqq \langle V(G), E(G), P(Z(G)) \rangle$$

where $P(Z(G))$ denotes the application of P in separate to each vertex's initial feature. Then, for $i \in [m]$ define

$$G^{(i)} \coloneqq \langle V(G), E(G), B_i(G^{(i-1)}) \rangle$$

the featured graph which comprises the structure of G and the updated feature map $B_i(G^{(i-1)})$. Finally,

$$B(G) \coloneqq R(B_m(G^{(m-1)}))$$

5.2.2 Message Passing + Self Attention

We now move to define GPS, and we shall start with defining the self-attention mechanism.

An *attention head* $H = (W_Q, W_K, W_V)$ of dimensions $d; d_h$ is a function parameterized by three matrices $W_Q, W_K, W_V \in \mathbb{R}^{d \times d_h}$, where d is the input dimension and d_h is the hidden dimension. For every $n \in \mathbb{N}$ it defines a function $H : \mathbb{R}^{n \times d} \to \mathbb{R}^{n \times d_h}$ such that

$$H(X) \coloneqq \left(\mathrm{softmax}\left(\frac{(XW_Q)(XW_K)^T}{\sqrt{d_h}}\right)\right)XW_V$$

where softmax : $\mathbb{R}^n \to \mathbb{R}^n$ is applied per-row and defined as

$$\text{softmax}(x_1, \ldots, x_n) \coloneqq \frac{e^{x_1}, \ldots, e^{x_n}}{\Sigma_{i\in[n]} e^{x_i}}$$

A *self-attention module* $\text{SA} = (H_1, \ldots, H_k, W_O)$ dimensions $k; d; d_h$ consists of k attention heads $H_1, \ldots, H_k$ of dimensions $d; d_h$, and a weight matrix $W_O \in \mathbb{R}^{kd_h \times d}$. For every $n \in \mathbb{N}$ it defines a function $\text{SA} : \mathbb{R}^{n\times d} \to \mathbb{R}^{n\times d}$ such that

$$\text{SA}(X) \coloneqq [H_1(X), \ldots, H_k(X)]W_O,$$

where $[\cdot]$ denotes row-wise concatenation.

A *Transformer gadget* $\text{TG} = (\text{SA}, \mathsf{mlp})$ of dimension d consists of a self-attention module of dimensions $k; d; d_h$ for some $k, d_h \in \mathbb{N}$, and an MLP of dimensions $d; d$. For every $n \in \mathbb{N}$, and for every $X \in \mathbb{R}^{n\times d}$, define

$$Y(X) \coloneqq \text{norm}_1\left(X + \text{SA}(X)\right)$$

Then, the function $\text{TG} : \mathbb{R}^{n\times d} \to \mathbb{R}^{n\times d}$ is defined as

$$\text{TG}(X) \coloneqq \text{norm}_2\left(Y + \mathsf{mlp}(Y)\right)$$

where mlp is applied per-row, and $\text{norm}_1; \text{norm}_2$ are optional normalization layers such as LayerNorm [BKH16] or BatchNorm [IS15].

We are now ready to define a single layer of GPS. A *GPS layer* $L = (A, M, \mathsf{mlp})$ of dimension p is a tuple, where A is a transformer gadget of dimension p, M is an MPG of dimension p, and mlp is an MLP of dimensions $p; p$. It defines a function $L : \mathcal{G}_{\mathbb{R}^p} \to \mathcal{Z}_{\mathbb{R}^p}$ from a featured graph to a new feature map as follows: Define

$$g(G)_i \coloneqq \left(A(Z(G))_i + Z(G)_i\right) + \left(M(G)_i + Z(G)_i\right)$$

where $Z(G)$ is viewed as a matrix (according to the vertices naming order) when provided as input to A. Then,

$$L(G)_i \coloneqq \mathsf{mlp}(g(X)_i) + g(X)_i$$

Next, we define the complete architecture. A *GPS* $B = (P, B_1, \ldots, B_m, R)$ of dimensions $p; q; r$ is a tuple where P is an MLP of I/O dimensions $p; q$, for every $i \in [m]$ it holds that B_i is a GPS layer of dimension q, and R is final readout gadget of dimensions $q; r$. It defines a graph embedding $B : \mathcal{G}_{\mathbb{R}^p} \to \mathbb{R}^r$ as follows. For $G \in \mathcal{G}_{\mathbb{R}^p}$ define

$$G^{(0)} \coloneqq \langle V(G), E(G), P(Z(G))\rangle$$

where $P(Z(G))$ denotes the application of P in separate to each vertex's initial feature. Then, for $i \in [m]$ define

$$G^{(i)} \coloneqq \langle V(G), E(G), B_i(G^{(i-1)})\rangle$$

the featured graph which comprises the structure of G and the updated feature map $B_i(G^{(i-1)})$. Finally,

$$B(G) \coloneqq R(B_m(G^{(m-1)}))$$

Note that incorporating Batch-Norm in GPS does not increase its expressivity, hence, the lack of it does not affect our results.

5.2.3 Positional Encodings

Graph Transformers typically rely on positional or structural encodings (PE/SE) to inject information about graph structure. [Ram+22] provide an overview over common PEs and SEs while new effective encodings are an active field of research [SQ19; NHK23]. Formally, a *positional encoding* $\pi(G) : V(G) \to \mathbb{R}^k$ is a special feature map assigning a k-dimensional vector to each vertex in a graph based on its local and/or global structure. In practice, PEs are precomputed before training and combined with the graph's original node features through concatenation or addition. Let $M_{\pi,G} = \{\{\pi(G, v) : v \in V(G)\}\}$ be the multiset of vectors that π assigns to the vertices of G. We say that π is *injective up to isomorphism and order* n if for all graphs G, H of order at most n it holds that:

$$G \ncong H \implies M_{\pi,G} \neq M_{\pi,H}. \tag{5.2.1}$$

That is, two graphs only map to the same multiset of vertex embeddings if they are isomorphic.

A common choice is LapPE [Kre+21] which is based on the eigendecomposition (λ, Σ) of the graph Laplacian, thus exploiting spectral graph features. Let $G = (V, E)$ be a graph, and let $L_G \in \mathbb{R}^{V \times V}$ be its Laplacian matrix. Recall that L_G has the vertex degrees as diagonal entries and at off-diagonal position $v \neq w$ entry -1 if v, w are adjacent and entry 0 otherwise. L_G is positive semi-definite and has nonnegative eigenvalues $\lambda_1 \leq \lambda_2 \ldots \leq \lambda_n$. Let $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_n$ be an orthonormal basis of corresponding eigenvectors. These vectors are indexed by vertices of G, that is, $\boldsymbol{u}_i \in \mathbb{R}^V$, and we let u_{iv} be the v-entry of $\boldsymbol{u}_i$. We define a positional encoding π_{Lap} by

$$\pi_{Lap}(G, v) = (\lambda_1, u_{1v}, \lambda_2, u_{2v}, \ldots, \lambda_n, u_{nv})$$

(since $\lambda_1 = 0$ and $\boldsymbol{u}_1 = \mathbf{1}$, we can also omit the first two entries). [1] This positional encoding is injective up to isomorphism, because the graph G can be reconstructed from the eigenvectors $\boldsymbol{u}_i$ and eigenvalues λ_i.[2] To make this positional encoding practical, but still keep it injective, we can round the reals to about n digits for graphs of order n. Note that π_{Lap} not only depends on G, but also on the choice of the vectors $\boldsymbol{u}_i$, which is not canonical even if we normalize the vectors, because the signs cannot be normalized and eigenspaces may have dimension greater than 1. Formally, this means that the mapping π is not equivariant.

[1] In [Kre+21; Ram+22], the eigenvalues λ and eigenvectors Σ are further processed by a learnable algorithm ENC such as a Transformer or DeepSet [Zah+17], that is, $\pi(G, v) = \text{ENC}(\Sigma, \lambda, v)$. In our analysis, we consider a no-ENC/identity-ENC LapPE.

[2] It is necessary to put the eigenvalues as well as the entries of the vectors in the positional encoding. Neither the eigenvalues alone nor the eigenvectors alone are sufficient to make the encoding injective. For the former, just note that there are well-known examples of nonisomorphic co-spectral graphs [see, e.g., GR01, Section 8.1]. For the latter, let G be the graph with two vertices and no edges with Laplacian $L_G = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$, and let H be the graph with two vertices and one edge with Laplacian $L_H = \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}$. The eigenvalues of L_G are $\lambda_1 = \lambda_2 = 0$, and the eigenvalues of L_H are $\lambda_1 = 0, \lambda_2 = 2$. However, for both graphs $\boldsymbol{u}_1 = (1/\sqrt{2}, 1/\sqrt{2})^\top$, $\boldsymbol{u}_2 = (1/\sqrt{2}, -1/\sqrt{2})^\top$ is an orthonormal basis of eigenvectors.

5.3 Non-Uniform Function Approximation on Graphs

In this section, we present universality results that are more or less immediate consequences of MLPs being universal function approximators. Nevertheless, we believe that it is important to discuss these results to clarify that the universality stated in the graph transformer literature is not a unique property of transformers. This has also been briefly discussed by [Mül+23].

A *transformer network* is a less expressive algorithm compared to GPS. Based on the the positional encoding LapPE being injective up to isomorphism and order n, [Kre+21] proved that there exists a transformer network T that maps the positional encodings of graphs G, H of order at most n to the same output if and only if G and H are isomorphic. This is a consequence of the injectivity of the positional encoding π combined with the universality of transformer networks [Yun+20]. It is important to note, though, that the transformer T here will be exponentially large in n; intuitively, it just memorizes an output for each possible multiset $M_{\pi,G}$ for all graphs G of order at most n. Having realized this, it is clear that the existence of a such an "isomorphism network" is not unique to transformer networks, but shared by other sufficiently powerful architectures including GNNs.

Proposition 5.3.1. *2-layer MLPs and 1-layer GNNs are universal function approximators when given access to a positional encoding π that is injective up to isomorphism and order n.*

This fact does not contradict the well known limitation of GNNs to be at most as powerful as the Weisfeiler-Leman algorithm for distinguishing graphs [Mor+19c; Xu+19b]. The idea is that when the positional encoding π is used as an initial coloring for each graph, then Weisfeiler-Leman distinguishes any two non-isomorphic graphs instantly after zero rounds of refinement. The caveat is that the algorithm also distinguishes isomorphic graphs G, H whenever $M_{\pi,G} \neq M_{\pi,H}$.

The full proofs of proposition 5.3.1 can be found in the appendix of [Ros+24]. Note that these proofs as well as the proof of the universality of graph transformers [Kre+21] share the same principle: Essentially they just implement a look-up tables for finitely many inputs. Hence, they do not provide any deep insights into the relative strengths and weaknesses of Graph Transformers and GNNs.

5.4 Uniform Expressivity of GPS and GNN+VNs

In the following sections we prove models' inability to uniformly express various functions. We show that the inexpressivity holds even when the models have reasonable PEs at their disposal e.g. LapPE. Such PEs are commonly used in practice and have the following properties: They are polynomial-time computable and their range is bounded. The former is assumed in Theorem 5.4.1 and the latter in Theorem 5.4.4. Unlike in Section 5.3, in the uniform setting, injectiveness (of PEs) does not guarantee expressivity: The proof idea of memorizing all inputs is not applicable in an infinite input-domain setting, and in

addition, commonly used PEs like LapPE are not injective for input graphs of unbounded size.

Note that throughout the results we consider a GPS model whose transformer modules incorporate neither BatchNorm nor LayerNorm. BatchNorm does not increase GPS expressivity, and Layer-Norm would not help GPS in the tasks we define, hence, the lack of them does not affect our inexpressivity results concerning GPS.

5.4.1 GPS And GNN+VNs Are Not Universal Approximators

First, we prove that in the uniform setting, none of the models we consider is a universal approximator.

Theorem 5.4.1. *Let h be the characteristic function of an NP-hard problem on graphs, for example: $h(G) = 1$ if G is 3-colorable and $h(G) = 0$ otherwise. Let Enc: $\mathcal{G}_* \to \mathcal{Z}_*$ a positional encoding function that is computable in polynomial time. Then, assuming PTIME $\neq$ NPTIME we have that GPS $\not\approx h$ and GNN+VNs $\not\approx h$ even when the input graph is featured with positional encodings computed by Enc.*

Proof. Note that in the uniform setting – having one network to handle graphs of all sizes – the algorithm implied by any (GPS or GNN+VN) network has a runtime polynomial in the size of the input graph. Assume to the contrary that there exists a (GPS or GNN+VN) network N that approximates $h(G)$ every input graph G featured with $\text{Enc}(G)$ as positional encoding. Then, it is possible to approximate $h(G)$ in polynomial time: Compute $\text{Enc}(G)$ in polynomial time and run N on G featured with $\text{Enc}(G)$. This is in contradiction to h being NP-hard and the assumption that PTIME $\neq$ NPTIME. □

Corollary 5.4.2. *GPS and GNN+VNs cannot approximate every computable function on graphs, not even every graph-function in NPTIME, even when the input graph is featured with PTIME-computable positional encodings.*

5.4.2 GPS Do Not Subsume GNN+VNs

Next, we prove that there is a target function that is expressible by a certain GNN+VN and is not expressible by any GPS even when the input graphs to the GPS include bounded PEs.

Let Enc: $\mathcal{G}_* \to \mathcal{Z}_{[-1,1]^d}$ be a positional encoding function with bounded range $[-1,1]^d$ for some $d \in \mathbb{N}$. Define a parameterized, no edges, graph $G_n, n \in \mathbb{N}_{>0}$, featured with PEs according to Enc: $V(G_n) = \{v_1, \ldots, v_n\}$; $E(G_n) = \emptyset$; $\forall v \in V(G_n)\ Z(G_n)(v) \coloneqq \text{Enc}(G_n)(v)$.

Lemma 5.4.3. *Let $L = (A, M, \mathsf{mlp})$ be a p-dimensional GPS layer such that $A = (W_Q, W_K, W_V, W_O)$ is a $(p; q; p)$-dimensional attention module (for some q), $M = (H, \mathsf{agg})$ is a p-dimensional MPG with $\mathsf{agg} \in \{\mathsf{mean}, sum, \mathsf{max}\}$, and mlp is a $p; p$-dimensional MLP. Then, if the input features and the graph degree are bounded, then so are the values of the output feature map. Formally, for every $b \in \mathbb{R}, d \in \mathbb{N}$ there exists $b' \in \mathbb{R}$ such that: For every input graph G of degree at most d and for which $\mathsf{max}(\left|Z(G)_{i,j}\right| : i \in [|G|], j \in [p]) \leq b$, it holds that $\mathsf{max}(\left|A(G)_{i,j}\right| : i \in [|G|], j \in [p]) \leq b'$.*

Proof. 1) The linear transformation defined by W_V is Lipschitz-continuous, and so the bounded input features determine a bounded range for the value vectors coordinates. By definition of the attention mechanism, the output of A is no higher than the maximum over the value vectors. Hence, there exists b'_A such that $\mathsf{max}(\left|A(Z(G))_{i,j}\right| : i \in [|G|], j \in [r]) \leq b'_A$.

2) An MLP is Lipschitz-continuous, hence for $\mathsf{agg} \in \{\mathsf{mean}, \mathsf{max}\}$ we have M to be Lipschitz-continuous, and for a bounded degree input graph we have also for $\mathsf{agg} = sum$ that M is Lipschitz-continuous.

3) The addition and composition of Lipschitz-continuous functions is Lipschitz-continuous and so for a graph-domain that is both degree-bounded and feature-bounded we have that A is bounded. □

Theorem 5.4.4. *Let $f : \mathcal{G}_1 \to \mathbb{R}$ a graph embedding such that for every n it holds that $f(G_n) = n^2$. Then, no GPS can approximate f for all graphs $\{G_n\}$. Formally, GPS $\not\approx f$.*

Proof. The idea is that for every GPS B, if the input graphs are of bounded degree and bounded features, then the value B computes for each vertex is bounded and therefore the output of B for the whole graph is $O(n)$ – no matter the input graph's size, unlike $f(G_n)$. Let a GPS $B = (P, B_1, \ldots, B_m, R)$ of dimension $p; d; q$ where $R = (F_R, AGG_R)$. The degree of any G_n is bounded (to be zero), the features are bounded, and since P is Lipschitz-continuous also $P(Z(G))$ (denoting the application of P in separate to each vertex's initial feature) is bounded. Hence, by Lemma 5.4.3 we have that $B_1(G)$ is bounded. By induction, using Lemma 5.4.3, we have that the output of every layer B_i is bounded, denote the bound on the output of B_m by b_m. Finally, R is Lipschitz-continuous, denote its Lipschitz-constant by a_R, then for $AGG_R \in \{\mathsf{mean}, \mathsf{max}, \mathrm{sum}\}$ we have that B is bounded by $n(p \cdot b_m \cdot d \cdot a_R)$. Then, given $\delta > 0$, we set $n = (\delta + p \cdot b_m \cdot d \cdot a_R + 1)$ and get

$$\left|f(G_n) - B(G_n)\right| \geq n(n - p \cdot b_m \cdot d \cdot a_R) \geq n(\delta + 1) > \delta$$

□

Combining Theorem 5.4.4 with the description of an GNN+VN that computes f exactly, we arrive at our main conclusion.

Corollary 5.4.5. *GPS $\not\geq^{[-1,1]}$ GNN+VNs.*

Proof. By Theorem 5.4.4 no GPS+(bounded)PEs can approximate f. It is not difficult to verify that a GNN+VN with one MPG+V with a sum-readout gadget, and a final sum-readout, can compute f exactly. □

5.4.3 GNN+VNs Do Not Subsume GPS

Lastly, we prove that there is a target function that is expressible by a certain GPS and is inexpressible by any GNN+VN. We define a parameterized, no edges, featured graph $G_{l,r}$, $l, r \in \mathbb{N}_{>0}$, as follows: $V(G_{l,r}) = \{u_1, \ldots, u_l, w_1, \ldots, w_{l \cdot r}\}$; $E(G_{l,r}) = \emptyset$; $Z(G_{l,r}) = \{(u_i, (2, 1))\}_{i \in [l]} \cup \{(w_i, (2, 2))\}_{i \in [l \cdot r]}$.

Note that all u_i are identical and all w_j are identical, hence we can omit the index when referring to a vertex. Let an GNN+VN $B = (P, B_1, \ldots, B_m, R)$ of dimensions $p; d; q$, where $B_i = (M_i, R_i)$ is an MP+V. We first characterize the functions that B can compute for $G_{l,r}$.

Lemma 5.4.6. *Let a $p;q$-dimensional MLP $M = (W^{(1)} \in \mathbb{R}^{d_1 \times p}, b^{(1)} \in \mathbb{R}^{d_1}, \ldots, W^{(m)} \in \mathbb{R}^{q \times d_{m-1}}, b^{(m)} \in \mathbb{R}^q)$, that is, for every input vector $X = (x_1, \ldots, x_p) \in \mathbb{R}^p$ it holds that $M(X) = b^{(m)} + W^{(m)} \operatorname{relu}(b^{(m-1)} + W^{(m-1)} \cdots \operatorname{relu}(b^{(1)} + W^{(1)}X))$. Then, regardless of the input, every output of M is one of finitely many affine functions. Formally, there exists a finite set $F = \{f_1, \ldots, f_k\}$ of affine functions $f_i : \mathbb{R}^p \to \mathbb{R}$, $f_i(X) = c_i + sum_{j=1}^{p}(a_{i,j}X_j)$ such that:*

$$\forall X \in \mathbb{R}^p \; \forall i \in [q] \; \exists f \in F : M(X)_i = f(X)$$

Proof. By induction on the number of layers of M. For $t = 1$ we have that either $\operatorname{relu}(b^{(1)} + W^{(1)}X)_j = 0$ or $\operatorname{relu}(b^{(1)} + W^{(1)}X)_j = b_j^{(1)} + W_{j,\cdot}^{(1)}X$, hence the set $F = \{0, (b_1^{(1)} + W_{1,\cdot}^{(1)}X), \ldots, (b_{d_1}^{(1)} + W_{d_1,\cdot}^{(1)}X)\}$ satisfies the requirement.

Assuming correctness for $t = n-1$, we prove for $t = n$: Let $F^{(n-1)}$ be the function-set that satisfies the lemma up to layer $n-1$. Let $Y \in \mathbb{R}^{d_{n-1}}$ be the output of layer $n-1$, we have that either $\operatorname{relu}(b^{(n)} + W^{(n)}Y)_j = 0$ or $\operatorname{relu}(b^{(n)} + W^{(n)}Y)_j = b_j^{(n)} + W_{j,\cdot}^{(n)}Y$, and by assumption $b_j^{(n)} + W_{j,\cdot}^{(n)}Y = b_j^{(n)} + W_{j,\cdot}^{(n)}(g_1(X), \ldots, g_{d_{n-1}}(X))$ for some $g_i \in F^{(n-1)}$. Hence, $\operatorname{relu}(b^{(n)} + W^{(n)})_j \in \{0\} \bigcup \{b_j^{(n)} + W_{j,\cdot}^{(n)}(g_1(X), \ldots, g_{d-1}(X)) : g_i \in F^{(n-1)}\}$, and since a linear combination of affine functions is an affine function the latter is a finite set of affine functions of X. □

Lemma 5.4.7. *There exists a finite set of functions $F = \{f_1, \ldots, f_k\}$, $f_i : \mathbb{N} \times \mathbb{N} \to \mathbb{R}$, such that:*

1. $\forall f \in F \;\; f(l,r) = sum_{i \in [0..m_1]} sum_{j \in [0..i]} sum_{x \in [0...m+1]} sum_{y \in [0..x]} a_{i,j,x,y}^{(f)} l^i r^j \frac{r^y}{(1+r)^x}$ *for some real coefficients $\{a_{i,j,x,y}^{(f)}\}$*
2. $\forall l \in \mathbb{N} \; \forall r \in \mathbb{N} \; \forall j \in [d] \; \exists f, g \in F : Z(G_{l,r}^{(m)})(u)_j = f(l,r)$ *and* $Z(G_{l,r}^{(m)})(w)_j = g(l,r)$.
3. $\forall l \in \mathbb{N} \; \forall r \in \mathbb{N} \; \forall j \in [q] \; \exists f \in F : B(G_{l,r})_j = f(l,r)$

Proof. Given $t \in [m], i \in [l], j \in [lr]$, we define shorthand notations for the values of any of the u_i and w_j, after the t^{th} MPG+V:

- $u_{l,r}^{(t)} \coloneqq Z(G_{l,r}^{(t)})(u_i), i \in [l]$
- $w_{l,r}^{(t)} \coloneqq Z(G_{l,r}^{(t)})(w_j), j \in [lr]$

By induction on the layer of the GNN+VN:

a) For $t = 0$, the preparation gadget is an MLP, and so its output is one of finitely many affine functions of the initial vertex values $u_{l,r}^{(0)}, w_{l,r}^{(0)}$.

b) Assume correctness for $t = n-1$, we prove for $t = n$. Let an MPG+V $B_n = (M_n, R_n)$ where $M_n = (U_n, AGG_n)$ is an MPG and $R_n = (F_n, RO_n)$ is a readout gadget.

b.1 Since there are no edges, $AGG_n(u_{l,r}^{(n-1)}) = 0$, and so $M_n(G_{l,r}, u^{(n-1)}) = U_n(u_{l,r}^{(n-1)})$. By the induction assumption $U_n(u_{l,r}^{(n-1)}) = U_n(g_1(l,r), \ldots, g_q(l,r))$ where $g_i \in F$ for some finite set F of functions of the form $f(l,r) = \operatorname{sum}_{i=0}^{n} \operatorname{sum}_{j=0}^{i} \operatorname{sum}_{x=0}^{n} \operatorname{sum}_{y=0}^{x} a_{i,j,x,y} l^i r^j \frac{r^y}{(1+r)^x}$. By Lemma 5.4.6, for every sequence $(g_1, \ldots, g_q) \in F^q \; \forall j \in [q] \; U_n(g_1(l,r), \ldots, g_q(l,r))_j$ is one of finitely many affine functions of $(g_1, \ldots, g_q)$ and so overall there is a finite set

F' of functions of the form $f(l,r) = \text{sum}_{i=0}^{n}\text{sum}_{j=0}^{i}\text{sum}_{x=0}^{n}\text{sum}_{y=0}^{x} a_{i,j,x,y}^{(f)} l^i r^j \frac{r^y}{(1+r)^x}$ such that $\forall j \in [q]\ M_n(G_{l,r}, u_{l,r}^{(n-1)})_j \in F'$. Similarly, $M_n(G_{l,r}, w_{l,r}^{(n-1)})_j \in F''$ for some finite F'' of functions of the same form.

b.2 If $RO_n = sum$ then $R_n(M_n(G_{l,r}^{(n-1)})) = F_n(l(M_n(G_{l,r}^{(n-1)}, u_{l,r}^{(n-1)}) + rM_n(G_{l,r}^{(n-1)}, w_{l,r}^{(n-1)})))$. By (b.1), $F_n(l(M_n(G_{l,r}^{(n-1)}, u_{l,r}^{(n-1)}) + rM_n(G_{l,r}^{(n-1)}, w_{l,r}^{(n-1)}))) = F_n(g_1(l,r), \ldots, g_q(l,r))$ where $g_i \in F$ for some finite set F of functions of the form $f(l,r) = \text{sum}_{i=0}^{n+1}\text{sum}_{j=0}^{i}\text{sum}_{x=0}^{n}\text{sum}_{y=0}^{x} a_{i,j,x,y} l^i r^j \frac{r^y}{(1+r)^x}$. By Lemma 5.4.6 then, using a line of proof similar to (b.1), we have that $\forall j \in [q]\ F_n(l(M_n(G_{l,r}^{(n-1)}, u_{l,r}^{(n-1)}) + rM_n(G_{l,r}^{(n-1)}, w_{l,r}^{(n-1)})))_j$ is one of finitely many possible functions of the aforementioned form.

b.3 If $RO_i = avg$ then $R_n(M_n(G_{l,r}^{(n-1)})) = F_n(\frac{l(M_n(G_{l,r}^{(n-1)}, u_{l,r}^{(n-1)}) + rM_n(G_{l,r}^{(n-1)}, w_{l,r}^{(n-1)}))}{l(1+r)}) = F_n(\frac{M_n(G_{l,r}^{(n-1)}, u_{l,r}^{(n-1)}) + rM_n(G_{l,r}^{(n-1)}, w_{l,r}^{(n-1)})}{1+r})$. Using a line of proof similar to (b.2), we have that $\forall j \in [q]\ F_n(\frac{M_n(G_{l,r}^{(n-1)}, u_{l,r}^{(n-1)}) + rM_n(G_{l,r}^{(n-1)}, w_{l,r}^{(n-1)})}{1+r})_j$ is one of finitely many possible functions of the form $f(l,r) = \text{sum}_{i=0}^{n}\text{sum}_{j=0}^{i}\text{sum}_{x=0}^{n+1}\text{sum}_{y=0}^{x} a_{i,j,x,y} l^i r^j \frac{r^y}{(1+r)^x}$.

c) For the final readout R, whether $R = sum$ or $R = avg$, by the induction assumption and using a line of proof similar to (b.3), we have that $\forall j \in [r]\ R(G^{(m)})_j$ is one of finitely many possible functions of the form $\text{sum}_{i=0}^{m+1}\text{sum}_{j=0}^{i}\text{sum}_{x=0}^{m+1}\text{sum}_{y=0}^{x} a_{i,j,x,y} l^i r^j \frac{r^y}{(1+r)^x}$. □

Next, we define a target function and characterize the gap that necessarily exists between it and any function of the kind described in Lemma 5.4.7 (1).

Lemma 5.4.8. *For $l \in \mathbb{N}_{>0}, r \in \mathbb{N}_{>0}$, define $h(G_{l,r}) \coloneqq l\big(\frac{3+2re^9}{1+re^9} + r\frac{3+2re^{12}}{1+re^{12}}\big)$.*
Let $f(l,r) = sum_{i=0}^{m+1} sum_{j=0}^{i} sum_{x=0}^{m+1} sum_{y=0}^{x} a_{i,j,x,y} l^i r^j \frac{r^y}{(1+r)^x}$ for some coefficients $\{a_{i,j,x,y}\}$, then

$$\exists r_0 : \forall r > r_0 \ \lim_{l\to\infty} \big|f(l,r) - h(G_{l,r})\big| = \infty$$

Proof. 1. If there is $i > 1$ such that $a_{i,j,x,y} \neq 0$ for some j, x, y, then clearly $\lim_{l\to\infty} \big|f(l,r) - h(G_{l,r})\big| = \infty$ for every r, and same if there exist no $i > 0$ and j, x, y for which $a_{i,j,x,y} \neq 0$. This is because the function $h(G_{l,r})$ is linear in l.

2. Otherwise, we can assume that

$$f(l,r) = \text{sum}_{i=0}^{1}\text{sum}_{j=0}^{i}\text{sum}_{x=0}^{m+1}\text{sum}_{y=0}^{x} a_{i,j,x,y} l^i r^j \frac{r^y}{(1+r)^x}$$

Define

$$f_1(l,r) \coloneqq \text{sum}_{x=0}^{m+1}\text{sum}_{y=0}^{x}(\frac{a_{0,0,x,y}}{l}) r^j \frac{r^y}{(1+r)^x},\ f_2(l,r) \coloneqq \text{sum}_{j=0}^{1}\text{sum}_{x=0}^{m+1}\text{sum}_{y=0}^{x} a_{1,j,x,y} r^j \frac{r^y}{(1+r)^x}$$

Then, we have that $f(l,r) = l(f_1(l,r) + f_2(l,r))$ and

$$\big|f(l,r) - h(G_{l,r})\big| = l\big|f_1(l,r) + f_2(l,r) - (\frac{3+2re^9}{1+re^9} + r\frac{3+2re^{12}}{1+re^{12}})\big|$$

Note that $\lim_{l\to\infty} f_1(l,r) = 0$; f_2 is independent of l; and since f_2, $r \mapsto (\frac{3+2re^9}{1+re^9} + r\frac{3+2re^{12}}{1+re^{12}})$ are distinct rational functions of r they can only coincide at finitely many values of r. Hence, $\exists r_0 : \forall r > r_0 \ \lim_{l\to\infty} \big|f(l,r) - h(G_{l,r})\big| = \infty$. □

Combining Lemma 5.4.7 and Lemma 5.4.8, we have that the target function is inexpressible by any GNN+VN. For $l \in \mathbb{N}_{>0}, r \in \mathbb{N}_{>0}$, define $h(G_{l,r}) \coloneqq l\big(\frac{3+2re^9}{1+re^9} + r\frac{3+2re^{12}}{1+re^{12}}\big)$.

Theorem 5.4.9. *No GNN+VN can approximate h for all graphs $\{G_{l,r}\}$. Formally, GNN+VNs $\not\approx h$.*

Proof. By Lemma 5.4.7 we have that no matter the l, r, $B(G_{l,r})$ is necessarily one of finitely many functions $\{f_1, \ldots, f_k\}$ such that, by Lemma 5.4.8, $\forall\ i \in [k]\ \exists r_i : \forall\ r > r_i\ \lim_{l\to\infty} \big|f_i(G_{l,r}) - h(G_{l,r})\big| = \infty$. Hence, $\forall\ r > max(r_i : i \in [k])\ \lim_{l\to\infty} \big|h(G_{l,r}) - B(G_{l,r})\big| = \infty$. □

However, this is not the case for GPS.

Lemma 5.4.10. *There exists a GPS that computes h exactly.*

Proof. Define an attention head $H = (W_K, W_Q, W_V)$ as follows:

- $W_Q = ((1, 1))$
- $W_K = ((2, 3))$
- $W_V = ((2, -1))$

We have that

$$\text{softmax}(Z(G_{l,r})W_Q(Z(G_{l,r})W_K)^T)Z(G_{l,r})W_V = \text{softmax}(\begin{pmatrix} 21\cdots21 & 30\cdots30 \\ 28\cdots28 & 40\cdots40 \end{pmatrix})\ \begin{pmatrix} 3\cdots3 & 2\cdots2 \end{pmatrix}^T =$$

$$\begin{pmatrix} \frac{3+2re^9}{1+re^9}\cdots\frac{3+2re^9}{1+re^9} & \frac{3+2re^{12}}{1+re^{12}}\cdots\frac{3+2re^{12}}{1+re^{12}} \end{pmatrix}^T$$

That is,

$$H(u_i)_{l,r} = \frac{l3e^{21} + lr2e^{30}}{le^{21} + lre^{30}} = \frac{3e^{21} + 2re^{30}}{e^{21} + re^{30}} = \frac{3 + 2re^9}{1 + re^9}$$

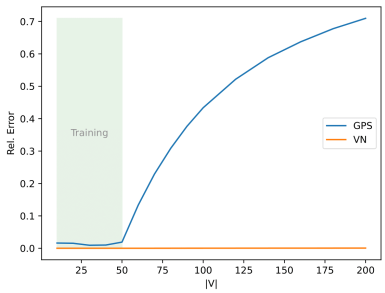
$$H(w_i)_{l,r} = \frac{l3e^{28} + lr2e^{40}}{le^{28} + lre^{40}} = \frac{3e^{28} + 2re^{40}}{e^{28} + re^{40}} = \frac{3 + 2re^{12}}{1 + re^{12}}$$

It is not difficult to verify that there is a GPS B comprising attention H and a final sum readout, such that $B(G_{l,r}) = l(H(u)_{l,r} + rH(w)_{l,r}) = l(\frac{3+2re^9}{1+re^9} + r\frac{3+2re^{12}}{1+re^{12}}) = h(G_{l,r})$. □

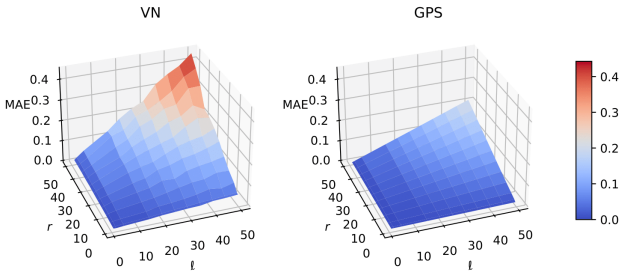
Combining Theorem 5.4.9 and Lemma 5.4.10, we arrive at our main conclusion.

Corollary 5.4.11. *GNN+VNs $\not\geq^{\{1,2\}}$ GPS.*

Proof. By Theorem 5.4.9 no GNN+VN can approximate h, and by Lemma 5.4.10 there exists a GPS that computes h exactly. □



(a) Test performance of GNN+VN and GPS when computing the square function. We report the relative error. The graphs in the training data were restricted to $|V| \leq 50$.



(b) Test performance of GNN+VN and GPS when computing the function $h(G_{l,r})$ from section 5.4.3. The training data is restricted to $l \in [1, 10], r \in [1, 5]$, where l is the number of type-1 vertices and $l \cdot r$ is the number of type-2 vertices.

Figure 5.2: Synthetic experiments confirming the theoretical differences between models in practice

5.5 Experimentation

We empirically verify the difference between GPS and GNN-VN architectures using synthetic datasets based on theorems 5.4.4 and 5.4.9. For further details regarding the data generation please refer to the appendix in [Ros+24]. Specifically, our goal is two fold:

- To test how well a model, whose architecture is uniformly expressive of the target function, and trained by SGD-based optimization, generalizes to larger graph sizes at inference time.
- To test the generalization error of a model whose architecture is uniformly inexpressive of the target function.

GNN+VN $\not\leq$ GPS From corollary 5.4.5 we know that the simple function where the value $|V|^2$ has to be predicted as a graph-level regression target can be uniformly expressed by an GNN with VN but not by a Graph Transformer. In order to verify this empirically, we designed a dataset consisting of 100k random graphs with size $10 \leq |V| \leq 50$. For testing, we use larger graphs of size up to 200 as we are interested in the generalization behavior of the trained models. We train GNN+VN and GPS architectures that are identical in width (256 hidden dimensions) and depth (3 layers). For both models we use a standard GCN layer [KW17c] as a message passing module. To the GPS model, we additionally provide LapPE encodings. fig. 5.2a shows the results on the test data. GNN+VN has learned to perfectly predict the target function even for graphs outside the training distribution. GPS does achieve a low relative error for graphs with size at most 50. However, for sizes unseen during training the performance deteriorates rapidly. Thus, GPS was unable to learn a function that generalizes across graph sizes which is in line with corollary 5.4.5.

GPS $\not\leq$ GNN+VN To empirically study the converse scenario, we generate another synthetic dataset consisting of 100k random graphs $G_{l,r}$ as defined in the beginning of section 5.4.3 with the target function being the graph-level regression target $h(G_l, r)$ from section 5.4.3. During training, we restrict the data distribution to $l \in [1, 10]$ and $r \in [1, 5]$

and increase the range to $l, r \in [1, 50]$ for testing. Again, we train 3-layer GNN+VN and GPS models based on GCN layers of identical width. fig. 5.2b provides the experiment's results. For both models the MAE increases roughly linearly in both l and r. The error of the GNN+VN model increases around twice as quickly as that of the GPS model which generalizes better to values of r and l outside the training distribution which aligns with corollary 5.4.11.

5.6 Summary

In the non-uniform setting we formally proved that strong positional encodings imply universality not only for graph transformers but also for GNNs and even MLPs.

In the uniform setting, we proved that graph transformers and GNNs with virtual nodes express partially-disjoint sets of functions. In particular, we observed that the self-attention mechanism, sophisticated as it may be, cannot compute unbounded aggregation (e.g. counting the nodes), and on the other hand the virtual node scheme cannot approximate the sophistication of self-attention.

Through experiments with synthetic data, we showed that the proven properties in the uniform setting have an effect in practice:

- For the target function expressible by GNN+VN and not by GPS, a trained GNN+VN model generalized well to graphs of sizes even four-times larger than seen during training, while a trained GPS model showed a significant and increasing multiplicative error when applied to graphs even slightly larger than those it was trained on.
- With the target function expressible by GPS and not by GNN+VN, it seems that the proven properties do not have the full effect in practice. A trained GPS model does not generalize very well i.e. an expressive model - which exists - was not learned. However, the GPS model still generalizes significantly better than a trained model of the GNN+VN.

Repetition Makes Perfect

> "Go through it over and over again, for all is therein."
>
> Yochanan ben Bag Bag, Pirkei Avot 5:22

6.1 Intro

This chapter is based on [RG25]. Our comparative results for non-recurrent GNNs not only tell us about the strengths and weaknesses of specific such architectures but also demonstrate that the non-recurrent scheme is fundamentally limited. In this section we study the expressivity of GNNs whose sequence of layers can be repeated - the output of the last layer being the input to the first layer in the next recurrence. We prove that a finite-precision parameters; single-layer; sum-aggregation; identity-msg, recurrent GNN architecture is as expressive as any message-passing algorithm can be, and equivalently - so we prove - as the class of computable functions that are invariant to the Color Refinement representation of a vertex.

Recurrent Graph Neural Networks

Recurrent GNNs [Sca+08; GM10] are similar to GNNs in that a recurrent GNN consists of finitely many layers and those are of the same kind of (non-recurrent) GNN layers, however their sequence can be reiterated a number of times that depends on the input. While perhaps not as common as GNNs, recurrent GNNs have been used successfully in practice [Li+16; Sel+16; BL18; Tön+23] and are considered promising architectures for solving various learning tasks. We remind the reader of the three factors limiting the expressivity of GNNs, described in detail in Section 1.3:

1. The message-passing scheme.
2. The fixed number of message passes.
3. The combination and aggregation algorithms used in the layers.

Recurrent GNNs are still limited by factor (1) as they still consist of local computations, by definition factor (2) does not apply to them, and the question is to what extent recurrence can mitigate factor (3). Recurrent MLPs are proved to be Turing-complete [SS92], implying that a layer's expressivity may be increased by reiterating it. However, it has not been clear thus far if reiteration can recover the information lost by common aggregation functions. The expressivity of a message-passing recurrent architecture, in

a setting of finite initial-features domain and output domain, and where each node's initial feature is augmented with the graph size, has been shown to be as expressing as any message-passing-based architecture can be [PCK24]. However, the combination and aggregation functions there are not limited to be computable and are certainly not implemented by an MLP and one of the common aggregation functions, making the result only an upper bound for computable recurrent GNNs. For a recurrent GNN architecture, it has been proven that there exists a recurrent single layer sum-aggregation GNN that can distinguish any two graphs distinguishable by Color Refinement [BKR24]. However, the proof is existential, the activation function of the GNN must not be relu, and most importantly, the GNN has a parameter whose value must be a real number - of infinite precision - making it incomputable. A tight logical bound for a computable recurrent GNN architecture, named by the authors an 'R-Simple AC-GNN[F]', is proven in [Ahv+]. The architecture is defined to operate only with bounded-length float values, making it limited in one aspect, and to aggregate multisets of such values always in order (e.g. ascending), making it sophisticated in another. All in all, expressivity-wise it is a strictly weaker architecture, and essentially different, than the recurrent GNNs we study. Moreover, it is characterized in different terms than the ones we use to bound our architectures.

Message Passing Limit

By their definition, the uniform expressivity of computable recurrent GNNs is upper-bounded by the expressivity of a general message-passing algorithm with id-less nodes that are aware of the graph size: A single recurring algorithm operating in parallel at each node, whose input at each recurrence is the node's value and a multiset of messages from the node's neighbors, and its output is a new value and a message to send. Importantly, the same message is broadcasted to all neighbors, and messages are received with no identification of the sending node. Recurrent GNNs are then a specific case, where the recurring algorithm is an MLP composed on an aggregation of the messages, rather than a general algorithm operating on the multiset of messages.

A well-studied representation of a node defines another important upper-bound: The class of all algorithms $\mathcal{A}(G, v)$, for a graph and node, that are invariant to the Color Refinement (CR) (or 1-Weisfeiler-leman) representation of the node. The Color Refinement procedure goes back to [Mor65; WL68], see also [Gro21] and Section 6.3.

Our results show that both mentioned upper bounds are actually tight i.e. recurrent GNNs can express every function in the above function-classes.

6.1.1 New Results

We provide tight bounds for the uniform expressivity of computable recurrent GNNs. We focus on finite-precision parameters; single-layer; sum-aggregation; identity-msg, recurrent GNNs, which - as we find - capture the expressivity of all computable recurrent GNNs.

We assume a finite initial-feature domain, and that the original feature is augmented with the graph-size value. In this setting, we prove that the class of functions expressible by recurrent GNNs is equivalent to the following classes of functions, with only polynomial time and space overhead:

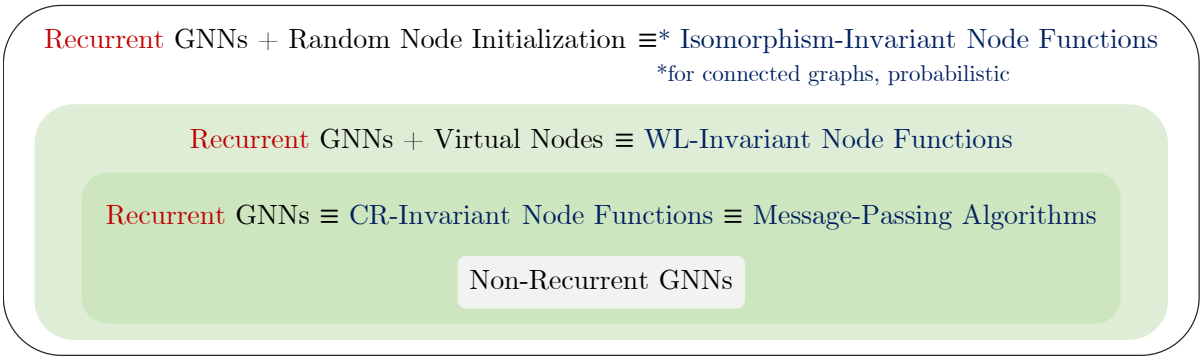


Figure 6.1: Uniform expressivity hierarchy. 'CR' and 'WL' are acronyms for Color Refinement and Weisfeiler-Leman. Main results in this chapter are the equivalencies. 'Node Functions' means computable functions $f(G, v)$ operating on a graph and node.

1. For node-level functions:

   a. All functions computable by a general message-passing algorithm where the nodes are aware of the graph size, that is, a message-passing algorithm where the update function is computable and its input is the multiset of the neighbors' messages.

   b. All computable functions whose input is a featured graph and a vertex of it, that are invariant to the Color Refinement representation of a node.

   c. When the layer's input is augmented with global sum-aggregation, all computable functions whose input is a featured graph and a vertex of it, that are invariant to the Weisfeiler-Leman representation of a node.

   d. When the nodes' initial features are augmented with random features, all computable functions whose input is a featured graph and a vertex of it, that are invariant to isomorphism.

   e. All computable functions whose input is the Color Refinement representation of a node.

2. For graph-level functions: For connected graphs, all computable functions whose input is a featured graph, that are invariant to the Color Refinement representation of a graph.

The main results for node-level functions are illustrated in Figure 6.1.

Roadmap

In Section 6.3 we describe the color representation of a node and its relation to the message passing scheme, and use it to define our upper-bound function classes. In Section 6.4 we define our recurrent-GNN architecture, and intermediate computation models, and describe the reduction from the upper-bound classes through the intermediate models and down to the recurrent GNN. In Section 6.5 we extend our results to graph-level functions and to GNN architectures that go beyond pure message passing.

6.2 Terms and Concepts

We define $\mathcal{B} \coloneqq \{0,1\}^*$ the set of all finite binary strings, and $\mathcal{B}_k \coloneqq \{0,1\}^k$ the set of all binary strings of length k. For $x \in \mathcal{B}_k$ we define $\left|x\right| = k$. For a binary string $\mathcal{B}_k \ni x = b_1, \ldots, b_k$, we define $\mathrm{B2I}(x) \coloneqq \Sigma_{i\in[k]} b_i 2^{i-1}$ the integer represented by the string in base-2. For binary strings $x_1, x_2 \in \mathcal{B}$ we define $x_1 + x_2 \coloneqq \mathrm{B2I}^{-1}(\mathrm{B2I}(x_1) + \mathrm{B2I}(x_2))$. When clear from the context, we may omit the $\mathrm{B2I}^{-1}$ notation and write an integer $x \in \mathbb{N}$ when referring to its base-2-string representation $\mathrm{B2I}^{-1}(x)$.

We consider the graph domain $\mathcal{G}_{\mathcal{B}_k^1}$ for any $k \in \mathbb{N}$ i.e. single-dimension featured graphs where the feature is a bit-string of some length, and in the interest of readability, for the rest of the chapter we omit the superscript and write $\mathcal{G}_{\mathcal{B}_k}$ while still meaning a single dimension. However, our results apply to multi-dimension finite-domain featured graphs as well, as it is possible to extend the construction in our proofs such that the resulting recurrent GNN handles multi-dimension features.

A *message-passing algorithm* [1] is a pair $C = (A, f)$, $A \in \mathcal{B}$, $f : \mathcal{B}^2 \to \mathcal{B}^2$ comprising an initial state and a computable function. It defines a feature transformation $C : \mathcal{G}_{\mathcal{B}} \to \mathcal{Z}_{\mathcal{B}}$ as follows: Define

$$\forall G \in \mathcal{G}_{\mathcal{B}}\ \forall v \in V(G)\ C^{(0)}(G, v) \coloneqq \theta\big(A, |G|, Z(G, v)\big), \emptyset$$

for some encoding θ of the initial state; the graph size; and the initial feature, and $\emptyset$ denoting the empty string. Then, define

$$\forall i > 0 \quad C^{(i+1)}(G, v) \coloneqq f\Big(C^{(i)}(G, v)(1), \mu(\{\{C^{(i)}(G, w)(2) : w \in N_G(v)\}\})\Big)$$

for some multiset encoding μ. Define the first iteration when 'finished' indicator turns 1:

$$I_v \coloneqq \begin{cases} \min(i : C^{(i)}(G, v)(1)(1) = 1) & \text{min exists} \\ \infty & \text{otherwise} \end{cases}$$

Finally, define $C(G, v) \coloneqq C^{(I_v)}(G, v)(2)$, if I_v is defined.

6.3 Message Passing Information

In this section, we give a precise technical description of the limits of message passing algorithms using the *Color Refinement* procedure. It aims to describe the maximal "message passing information" each node can obtain. It will be convenient to refer to this message passing information as the "color" of a node. Note, however, that the message-passing colors are complex objects, nested tuples of multisets, which will later be compactly represented by dags.

Definition 6.3.1. Let $G \in \mathcal{G}_{\mathcal{B}_k}$. For every $t \geq 0$ and $v \in V(G)$, we define the *message passing color of v after t rounds* inductively as follows: The *initial color* of v is just its

[1] Our definition is equivalent to any distributed algorithm where the initial input includes the graph size, and the input from a node's neighbors is a multiset - with no ids or order

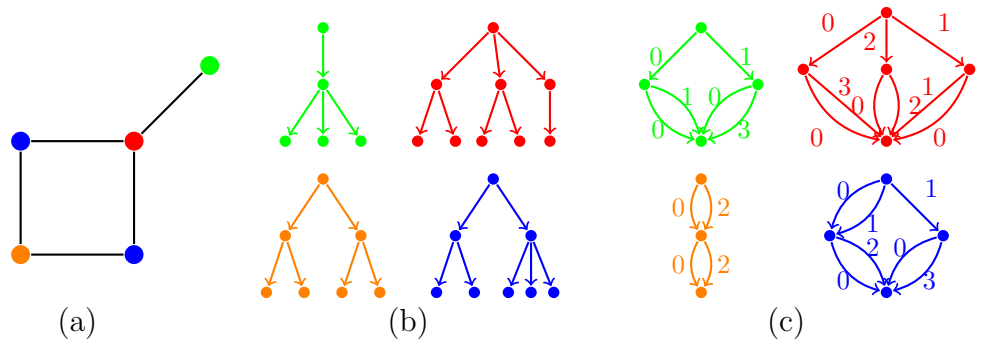


Figure 6.2: Two rounds of Color Refinement on a graph G shown in (a). Colors can be represented as trees (b) or dags (c). The actual colors in the figure illustrate the coloring reached after two steps (they do not indicate initial features of the nodes).

feature in G, that is, $\mathsf{mpc}_G^{(0)}(v) := Z(G,v)$. The color of v after $(t+1)$ rounds is the color of v after t rounds together with the multiset of colours of v's neighbours, that is,

$$\mathsf{mpc}_G^{(t+1)}(v) := \big(\mathsf{mpc}_G^{(t)}(v), \{\{\mathsf{mpc}_G^{(t)}(w) \mid w \in N_G(v)\}\}\big).$$

Moreover, we define the *final color* of v to be $\mathsf{mpc}_G(v) := \mathsf{mpc}_G^{2|G|}(v)$.

For all $t, n, k \in \mathbb{N}$, we let

$$\mathsf{MPC}_{n,k}^{(t)} := \big\{\mathsf{mpc}_G^{(t)}(v) \;\big|\; G \in \mathcal{G}_{\mathcal{B}_k}, |G| = n, v \in V(G)\big\},$$
$$\mathsf{MPC}^{(t)} := \bigcup_{n,k\in\mathbb{N}} \mathsf{MPC}_{n,k}^{(t)}, \qquad \mathsf{MPC} := \bigcup_{t\in\mathbb{N}} \mathsf{MPC}^{(t)}.$$

While most applications of Color Refinement are mainly interested in the partition of the vertices into color classes, for us, the actual colors carrying the message passing information are important. If written as strings in a straightforward manner, the colors will become exponentially large (up to size $\Omega(n^t)$). We may also view the colors by trees, which are still exponentially large (see Figure 6.2(b)).

The Represention Of a Color

We now introduce a succinct representation of polynomial size (see Figure 6.2(c)). Every $c \in \mathsf{MPC}_{n,k}^{(t)}$ will be represented by a directed acyclic graph (dag) $D(c)$ with labelled edges and labelled leaves and possibly with multiple edges between the same nodes. $D(c)$ will have $t+1$ levels (numbered $0, \dots, t$), each with at most n vertices. All edges will go from some level $i \geq 1$ to level $i-1$. Edges will be labelled by natural numbers in the range $0, \dots, n-1$. Every vertex v on a level i will *represent* some color $\gamma(v) \in \mathsf{MPC}_{n,k}^{(i)}$ in such a way that the mapping γ is injective, that is, every $d \in \mathsf{MPC}_{n,k}^{(i)}$ is represented by at most one node.

We say that a color d is an *element* of a color c and write $d \in c$ if $c = (d_0, \{\{d_1, \dots, d_k\}\})$ and $d \in \{d_0, \dots, d_k\}$. We construct the dag $D(c)$ for $c \in \mathsf{MPC}_{n,k}^{(t)}$ inductively as follows.

- Level t consists of a single node u, and we let $\gamma(u) = c$.
- Suppose that for some $i \in [t]$ we have defined level i and that C_i is the set of all colors represented by a node on level i. Note that $|C_i| \leq n$, because at most n distinct colors can appear in a graph of order n.

We let C_{i-1} be the set of all elements of colors in C_i. For every $d \in C_{i-1}$ we introduce a node v_d on level $i-1$ representing d, that is, $\gamma(v_d) := d$.

To define the edges, let v be a node on level i with $\gamma(v) = (d_0, \{\{d_1, \ldots, d_k\}\})$. We add an edge with label 0 from v to the unique node v' on level $i-1$ with $\gamma(v') = d_0$. Moreover, for every $d \in C_{i-1}$ such that the multiplicity of d in the multiset $\{\{d_1, \ldots, d_k\}\}$ is $\ell \geq 1$ we add an edge labelled ℓ from v to the node v'' on level $i-1$ with $\gamma(v'') = d$.

- We still need to define the labels of the leaves, that is, the nodes on level 0. Every node v on level 0 represents an initial color $\gamma(v) \in \mathsf{MPC}^{(0)}_{n,k}$. Such a color is a feature of a graph $G \in \mathcal{G}_{\mathcal{B}_k}$, that is, a bitstring of length k, and we label v by this bitstring.

This completes the description of $D(c)$.

Example 6.3.2. Consider the graph in Figure 2(a). The colors in the figure represent the colors reached after two rounds of Color Refinement. Let us consider one of the colors, say blue, and explain how the (blue) dag representing this color is constructed. The graph has no features, so all nodes get the same initial color, corresponding to the unique bottom node of the dag. Colors reached after the first round of Color Refinement correspond to the degrees of the nodes. In our blue dag, we need two of these colors, "degree 2" and "degree 3". The left node of level 1 of the blue dag represents "degree 2"; it has an edge labeled 2 to the unique node (color) on level 0. The right node of level 1 represents "degree 3"; it has an edge labeled 3 to the unique node on level 0. In addition, both nodes on level 1 have an edge labeled 0 to the unique node on level 0, indicating that in the previous round they both had the color represented by that node. On level 2 we have just one node, representing the color blue. A blue node has one neighbor of degree 2 and one neighbor of degree 3. Hence the top node in the dag has an edge labeled 1 to the node on level 1 representing "degree 2" and an edge labeled 1 to the node on level 1 representing "degree 3". Moreover, a blue node has degree 2 itself. Hence it has an edge labeled 0 to the node on level 1 representing "degree 2".

Observe that $|D(c)| \leq (t+1)n$ and that for $c, d \in \mathsf{MPC}^{(t)}_{n,k}$ we have $D(c) = D(d) \iff c = d$. Furthermore, it is easy to see that given a graph $G \in \mathcal{G}_{\mathcal{B}_k}$ of order n, a node $v \in V(G)$, and a $t \in \mathbb{N}$, we can compute $D(\mathsf{mpc}^{(t)}_G(v))$ in time polynomial in k, n, t. To do this, we construct a dag simultaneously representing all colors appearing in a graph. Once we have this, we can construct a dag only representing a single color by deleting unreachable nodes. We construct the dag level by level in a bottom up fashion, maintaining pointers from the nodes of the graph to the nodes of the dag representing their color at the current level. At each level, we need to sweep through all nodes of the graph and all their incident edges, which overall requires time $O(n+m)$, where m is the number of edges of the graph. Thus overall, to construct t levels of the dag we need time $O(t(n+m))$.

A feature transformation $F : \mathcal{G}_{\mathcal{B}} \to \mathcal{Z}_{\mathcal{B}}$ is *message-passing-invariant* (for short: *mp-invariant*) if for all graphs $G, H \in \mathcal{G}_{\mathcal{B}}$ of the same order $|G| = |H|$ and nodes $v \in V(G), w \in V(H)$, if $\mathsf{mpc}^{(t)}_G(v) = \mathsf{mpc}^{(t)}_H(w)$ for all $t \geq 1$ then $F(G, v) = F(H, w)$. By an induction on the number of message-passing rounds, every feature transformation computed by a message-passing algorithm is mp-invariant. The converse is implied by

the fact that, clearly, $D(\mathsf{mpc}_G(v))$ can be constructed by a message-passing algorithm, together with the following lemma which asserts that, remarkably, $D(\mathsf{mpc}_G(v))$ suffices to compute any mp-invariant feature transformation.

Lemma 6.3.3. *Let $F : \mathcal{G}_{\mathcal{B}} \to \mathcal{Z}_{\mathcal{B}}$ be a computable feature transformation. Then F is mp-invariant if and only if there is an algorithm that computes $F(G, v)$ from $\mathsf{mpc}_G(v)$. More precisely, there is an algorithm that, given $D(\mathsf{mpc}_G(v))$, computes $F(G, v)$, for all graphs $G \in \mathcal{G}_{\mathcal{B}}$ and nodes $v \in V(G)$.*

Furthermore, if F is computable in time $T(n)$ then the algorithm can be constructed to run in time $T(n) + \mathrm{poly}(n)$, and conversely, if the algorithm runs in time $T(n)$ then F is computable in time $T(n) + \mathrm{poly}(n)$.

The crucial step towards proving this lemma is to reconstruct a graph from the message passing color: given $D(\mathsf{mpc}_G(v))$, we can compute, in polynomial time, a graph G' and a node v' such that $\mathsf{mpc}_{G'}(v') = \mathsf{mpc}_G(v)$. This nontrivial result is a variant of a theorem due to [Ott97]. Once we have this reconstruction, Lemma 6.3.3 follows easily. Please refer to [RG25, Proof of Lemma 3.3] for the complete proof details.

6.4 Main Results

We would like to characterize the expressivity of R-GNNs, which operate on rational numbers, in terms of computable functions i.e. algorithms that operate on bit-strings. We use two encodings of the latter representation by the former: Rational Quaternary and Rational Binary, the reasons for which reside in the proof of Lemma 6.4.9. Let

$$\mathrm{RQ} \coloneqq \{\Sigma_{i=1}^{k} a_i 4^{-i} : \forall j \in [k]\ a_j \in \{1, 3\}, k \in \mathbb{N}\}$$

$$\mathrm{RB} \coloneqq \{\Sigma_{i=1}^{k} a_i 2^{-i} : \forall j \in [k]\ a_j \in \{0, 1\}, k \in \mathbb{N}\}$$

, then define the encoding operations

$$\mathfrak{rq} : \mathcal{B} \to \mathrm{RQ},\ \mathfrak{rq}(b_1, \ldots, b_k) \coloneqq \Sigma_{i=1}^{k} (2b_i + 1) 4^{-i}$$

$$\mathfrak{rb} : \mathcal{B} \to \mathrm{RB},\ \mathfrak{rb}(b_1, \ldots, b_k) \coloneqq \Sigma_{i=1}^{k} b_i 2^{-i}$$

For vectors of binary strings we may use the $\mathfrak{rq}, \mathfrak{rb}$ notations to denote the element-wise encoding, that is, $\forall (B_1, \ldots, B_l) \in \mathcal{B}^l$

$$\mathfrak{rq}(B_1, \ldots, B_l) \coloneqq (\mathfrak{rq}(B_1), \ldots, \mathfrak{rq}(B_l)),$$

$$\mathfrak{rb}(B_1, \ldots, B_l) \coloneqq (\mathfrak{rb}(B_1), \ldots, \mathfrak{rb}(B_l))$$

We are now ready to define the recurrent GNN architecture that is the subject of our main result. Part of the definition is the initial input provided to it. We choose to include the maximum feature length (across all nodes) k in that input, as this allows us later to construct a single GNN for all $G \in \mathcal{G}_{\mathcal{B}}$, as stated in Theorem 6.4.3. Alternatively, we could waive having k in the input, make it instead a parameter of the architecture, and restrict the statement in Theorem 6.4.3 to all $G \in \mathcal{G}_{\mathcal{B}_k}$.

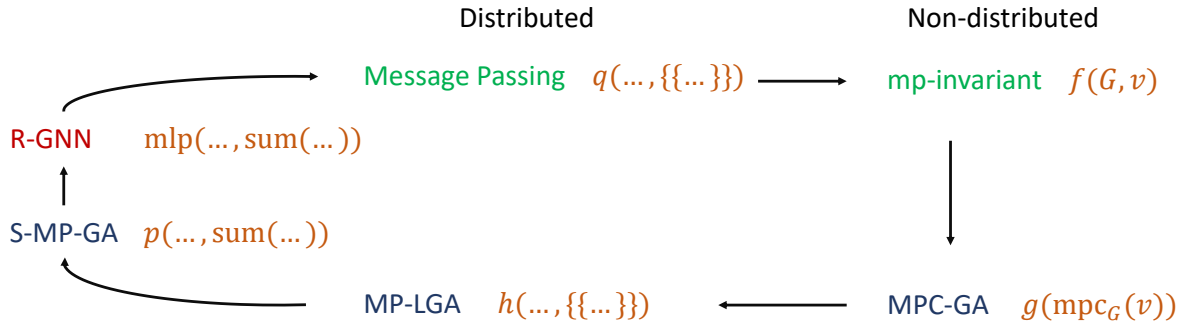


Figure 6.3: An overview of the reduction sequence from message-passing algorithms, and from mp-invariant functions, to R-GNNs. Note that MP-LGA differs from Message Passing by having bounded-length messages. Every recurrent message-passing algorithm is mp-invariant - by induction on the number of message-passing rounds. Then, starting from the mp-invariant class and moving clockwise, the reductions correspond to Lemma 6.3.3; Lemma 6.4.6; Lemma 6.4.8; and Lemma 6.4.9.

Definition 6.4.1. A *Recurrent Sum-GNN* (R-GNN) $N = (A, F)$ of dimension d is a pair comprising a constant initial-state vector $A \in \mathbb{Q}^{d-3}$ and an MLP F of I/O dimensions $2d; d$. Dimension d is a 'computation finished' indicator, and dimension $d-1$ holds the computation result. It defines a feature transformation $N : \mathcal{G}_{\mathcal{B}} \to \mathcal{Z}_{\mathcal{B}}$ as follows: Let $G \in \mathcal{G}_{\mathcal{B}}$, let $k \coloneqq \mathsf{max}(|b| : b \in \mathrm{img}(Z(G)))$ the maximum length over the binary-string features of the vertices in G, and let $v \in V(G)$. Define the initial value of N to be the rational vector comprising the graph size; max feature length; initial feature; and initial state, that is,

$$N^{(0)}(G, v) \coloneqq |G|, k, \mathfrak{rb}(Z(G, v)), A$$

Define the value of N after $t > 0$ iterations to be

$$N^{(t)}(G, v) \coloneqq F(N^{(t-1)}(G, v), \Sigma_{w \in N_G(v)} N^{(t-1)}(G, w))$$

Define the first iteration when 'finished' turns 1

$$I_v \coloneqq \begin{cases} \min(i : N^{(i)}(G, v)(d) = 1) & \text{min exists} \\ \infty & \text{otherwise} \end{cases}$$

Then,

$$N(G, v) \coloneqq \begin{cases} \mathfrak{rb}^{-1}\big(N^{(I_v)}(G, v)(d-1)\big) & I_v \in \mathbb{N} \\ \text{undefined} & \text{otherwise} \end{cases}$$

the binary string represented in rational-binary encoding at position $(d-1)$, when the 'finished' indicator turns 1.

We define a time measure $T_N(G, v) \coloneqq I_v$, and

$$T_N(G) \coloneqq \mathsf{max}(I_v : v \in V(G))$$

We say that an R-GNN N *uses time* $T(n)$, for a function $T : \mathbb{N} \to \mathbb{N}$, if for all graphs G of order at most n it holds that $T_N(G) \leq T(n)$. We define $L_N(G, v)$ to be the largest

bit-length over all parameters' and neurons' values of F, at any point of the computation for v, and we define

$$L_N(G) \coloneqq \Sigma_{v \in V(G)} L_N(G, v)$$

We say that an R-GNN N *uses space* $S(n)$, for a function $S : \mathbb{N} \to \mathbb{N}$, if for all graphs G of order at most n it holds that $L_N(G) \leq S(n)$.

Note that reaching a fixed point is not required for our results, hence it is not part of R-GNNs termination definition. However, the R-GNN we construct in the proof of Lemma 6.4.9 does have that property i.e.

$$I_v \in \mathbb{N} \Rightarrow \forall t \geq I_v \; N^{(t)}(G, v)[d-1, d] = N^{(I_v)}(G, v)[d-1, d]$$

, which may be useful in practice and for relation to logic.

Note that the initial nodes' features in an R-GNN include the size $|G|$ of the input graph G. The following theorem proves that this is a must for maximum expressivity of message passing algorithms. For recurrent GNNs with a global readout mechanism (see Section 6.5), this requirement is removed since such GNNs can compute the size.

Theorem 6.4.2. *There exists a feature transformation $F : \mathcal{G}_{\mathcal{B}} \to \mathcal{Z}_{\mathcal{B}}$ such that for every message-passing algorithm $\mathcal{A}$ where the graph-order input is omitted, there exist $G \in \mathcal{G}_{\mathcal{B}}, v \in V(G)$ for which $\mathcal{A}(G, v) \neq F(G, v)$.*

Proof. Consider the function F defined by $F(G, v) = 1$ if v is contained in a cycle of G and $F(G, v) = 0$ otherwise. It is not hard to see that F is mp-invariant.

Suppose that there is a message-passing algorithm $\mathcal{A}$, where the graph-order input is omitted, that computes F. Consider the computation of $\mathcal{A}$ on a cycle. Regardless of the length of the cycle, the computation will be the same, because for all cycle C, C', all nodes $v \in V(C)$, $v' \in V(C')$, and all $t \in \mathbb{N}$ it holds that $\mathsf{mpc}_C^{(t)}(v) = \mathsf{mpc}_{C'}^{(t)}(v')$. Hence there is an $I \in \mathbb{N}$ such that for all cycles C and nodes $v \in V(C)$ we have $I_v = I$, where $I_v \in \mathbb{N}$ is the finishing iteration of $\mathcal{A}$ on C, v. That is, the computation terminates after I rounds and returns the value $F(C, v) = 1$.

Now consider a long path P of even length $\geq 2I$ and let w be the middle node of this path. As the neighborhood of radius I of w is identical to the neighborhood to a node v on a cycle C of length $\geq 2I$, we have $\mathcal{A}^{(i)}(P, w) = \mathcal{A}^{(i)}(C, v)$ for all $i \leq I$. Hence $I_w = I$ and $\mathcal{A}(P, w) = \mathcal{A}(C, v) = 1$, hence $\mathcal{A}(P, w) \neq F(P, w)$. □

Our main theorem refers to mp-invariant functions. However, since every message-passing algorithm is mp-invariant and since R-GNNs are specific message-passing algorithms, we have that R-GNNs are expressivity-wise equivalent also to message-passing algorithms.

Theorem 6.4.3. *Let $F : \mathcal{G}_{\mathcal{B}} \to \mathcal{Z}_{\mathcal{B}}$ be a computable feature transformation. Then F is mp-invariant if and only if there is an R-GNN N such that*

$$\forall G \in \mathcal{G}_{\mathcal{B}} \; \forall v \in V(G) N(G, v) = F(G)(v)$$

Furthermore, if F is computable in time $T(n)$ and space $S(n)$ then N uses time $O(T(n)) + \mathrm{poly}(n)$ and space $O(S(n)) + \mathrm{poly}(n)$.

6.4.1 Intermediate Reductions

Our proof of Theorem 6.4.3 reduces an mp-invariant function to an R-GNN through a sequence of three intermediate computation models, see Figure 6.3 for an illustration:

$$\text{mp-invariant} \ \leq \text{MPC-GA} \ \leq \text{MP-LGA} \ \leq \text{S-MP-GA} \ \leq \text{R-GNN}$$

The models operate on bit-strings, hence we define bit-encodings for data entities that appear in the models' definitions.

Data Entities Encodings

We define $\delta : \mathsf{MPC} \to \mathcal{B}$ to be an encoding of space complexity $O(n^3 \log n + kn)$ for all $c \in \mathsf{MPC}^{(t)}_{n,k}, t \leq 2n$, such that all required operations can be done in polynomial time. Following the construction description of $D(\mathsf{mpc}_G(v))$ in Section 6.3, it is evident that there exists such an encoding.

We define $\mu : \left(\!\!\binom{\mathcal{B}}{*}\!\!\right) \to \mathcal{B}$ to be a multiset encoding such that the elements can be encoded and decoded in linear time by a Random Access Machine (RAM). We define $\theta : \mathcal{B}^* \to \mathcal{B}$ to be a tuple encoding such that the elements can be encoded and decoded in linear time by a RAM. Clearly, such encodings exist.

For $l, c, k \in \mathbb{N}$ we define $\theta^{(l)}_{k,c} : (\mathcal{B}_k)^l \to \mathcal{B}$ to be a tuple encoding of l bit-strings of length at most k, of space complexity $O(l(\log(c) + k)$, such that the elements can be encoded and decoded in linear time by a RAM, and, most importantly, the separation between the elements is preserved under summation of c such encodings: For all $(x_1, \ldots, x_c), x_i \in (\mathcal{B}_k)^l$ it holds that

$$\Sigma^c_{i=1}(\theta^{(l)}_{k,c}(x_i)) = \theta^{(l)}_{k,c}\Big(\Sigma^c_{i=1}\big(x_i(1)\big), \ldots, \Sigma^c_{i=1}\big(x_i(l)\big)\Big)$$

A straightforward encoding that reserves $\log(c2^k)$ bits for each one of the l parts satisfies the requirements.

Definition 6.4.4. An *MPC Graph Algorithm* (MPC-GA) $C = (M)$ is simply a Turing machine. It defines a feature transformation $C : \mathcal{G}_{\mathcal{B}} \to \mathcal{Z}_{\mathcal{B}}$ as follows:
Let $G \in \mathcal{G}_{\mathcal{B}}, v \in V(G)$, then $C(G, v) \coloneqq M(\delta(\mathsf{mpc}_G(v)))$.

Let F be an mp-invariant function. By Lemma 6.3.3, there exists an MPC-GA that computes F with polynomial time and space overhead. Hence, the first stage in the reduction sequence in Figure 6.3 is already proven. The next step is to translate MPC-GA - whose input is already the color of a vertex - to a distributed algorithm that has to gather that information before applying the core algorithm to it.

Definition 6.4.5. Let $C = (A, f)$, $A \in \mathcal{B}$, $f : \mathcal{B}^2 \to \mathcal{B}^2$ be a pair, comprising an initial state and a computable function. It defines a feature transformation $C : \mathcal{G}_{\mathcal{B}} \to \mathcal{Z}_{\mathcal{B}}$ as follows: Let $G \in \mathcal{G}_{\mathcal{B}}$ and $v \in V(G)$, then define

$$C^{(0)}(G, v) \coloneqq \theta\big(A, |G|, \delta(Z(G, v))\big), \delta\big(Z(G, v)\big)$$

a 2-dimension vector, the first binary string being the tuple encoding of the initial state; graph size; and encoding of the initial feature, and the second being only an encoding of the initial feature. Define $\quad \forall t \geq 0 \ \ C^{(t+1)}(G, v) \coloneqq$

$$f\Big(C^{(t)}(G, v)(1), \mu(\{\{C^{(t)}(G, w)(2) : w \in N_G(v)\}\})\Big)$$

the value after $t + 1$ iterations. Finally, define

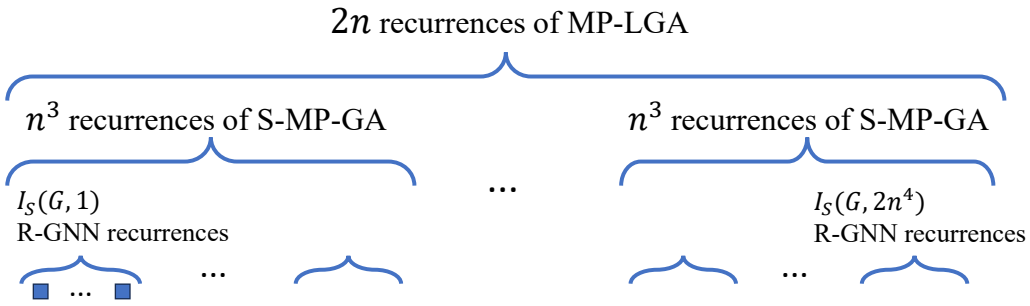


Figure 6.4: Let $G \in \mathcal{G}_{\mathcal{GB}_k}$ and define $n \coloneqq |G|$. An illustration for reducing the construction of $\mathsf{mpc}_G(v)$ to R-GNN. First it is reduced to $O(n)$ iterations of MP-LGA. Then, each iteration of MP-LGA is reduced to $O(n^3)$ iterations of S-MP-GA. Let that S-MP-GA be S, and let $T_S(G,i)$ be the number of Turing machine steps required to compute the i^{th} iteration of S when operating on G. Then, iteration $i, i \in [2n^4]$ is reduced to $O(I_S(G,i))$ recurrences of an R-GNN, where $I_S(G,i) \coloneqq T_S(G,i) + (n^3\log(n) + kn)^2$. The $(n^3\log(n) + kn)^2$ overhead is for translating the sum of messages from RB to RQ.

$$C(G,v) \coloneqq C^{(2|G|+1)}(G,v)(2)$$

That is, the final output is the second output of f after the $2|G|+1$ iteration. We say that $C = (A,f)$ is a *Message Passing Limited Graph Algorithm* (MP-LGA) if

$$\forall G \in \mathcal{G}_{\mathcal{GB}_k}\ \forall\ v \in V(G)\ \forall t \in [2|G|+1]$$

$$|\mu(\{\{C^{(t-1)}(G,w)(2) : w \in N_G(v)\}\})| \leq O(3k|G|^4)$$

, that is, the bit-length of the multiset of neighbors' messages does not exceed $3k|G|^4$.

Lemma 6.4.6. *Let $C = (M)$ be an MPC-GA, then there exist $A \in \mathcal{B}$, $f : \mathcal{B}^2 \to \mathcal{B}^2$ such that $C' = (A,f)$ is an MP-LGA and $\forall G \in \mathcal{G}_{\mathcal{GB}}\ \forall v \in V(G)\ C(G,v) = C'(G,v)$. Furthermore, C' incurs polynomial time and space overhead - in addition to the time and space used by C.*

Proof. The idea is to construct $\mathsf{mpc}_G(v)$ step by step in the first $2|G|$ applications of f. Then, in the $(2|G|+1)$ application, compute the function determined by M, on the constructed $\mathsf{mpc}_G(v)$. The state of the computation remembers the last iteration-number t (up to $2|G|+1$), the graph size, and $\mathsf{mpc}_G^{(t)}(v)$. The required sum-of-messages length limit is implied by the dag-construction description in Section 6.3. For an encoding of a triplet of binary strings $x = \theta(b_1,b_2,b_3), b_i \in \mathcal{B}$, define:

$x.t \coloneqq \mathrm{B2I}(b_1)$, representing the iteration-number part
$x.s \coloneqq \mathrm{B2I}(b_2)$, representing the graph size part
$x.d \coloneqq b_3$, representing the $\delta(\mathsf{mpc}_G^{(x.t)}(v))$ part.

Let $\delta(c_1),\ldots,\delta(c_l), \forall i \in [l]\ c_i \in \mathsf{MPC}^{(t)}_{|G|,k}$ be the dag encodings of a vertex v and its neighbors colors after t Color Refinement iterations, for some k,t. We define the operation of combining these encodings into the dag encoding of the next-iteration color of v.

$$\mathfrak{dc}(\delta(c_1),\{\{\delta(c_2),\ldots,\delta(c_l)\}\}) \coloneqq \delta(c_1,\{\{(c_2),\ldots,(c_l)\}\})$$

We define (A, f) as follows: $A = 0$, representing an initial iteration-number of zero. $f(x_1, x_2) \coloneqq$

$$\begin{cases} (\theta(x_1.t+1, x_1.s, \delta(x_1.d, x_2)), \mathfrak{dc}(x_1.d, x_2)) & \\ & x_1.t \leq x_1.s \\ (\theta(x_1.t+1, x_1.s, x_1.d), M(x_1.d)) & \\ & x_1.t = x_1.s + 1 \end{cases}$$

□

Next, we reduce the MP-LGA model to a model where the input from neighbors is **the sum** of the multiset of neighbors messages rather than the multiset itself. This addresses the first main obstacle of the overall reduction: To recover the information lost by the sum-aggregation and reconstruct the multiset of messages.

Definition 6.4.7. Let $C = (A, f)$, $A \in \mathcal{B}$, $f : \mathcal{B}^4 \to \mathcal{B}^4$ be a pair, comprising an initial state and a computable function. It defines a feature transformation $C : \mathcal{G}_{\mathcal{B}_k} \to \mathcal{Z}_{\mathcal{B}}$ as follows: Let $G \in \mathcal{G}_{\mathcal{B}_k}$, $v \in V(G)$, then define

$$C^{(0)}(G, v) \coloneqq \theta\big(|G|, Z(G, v), A\big), \theta^{(2)}_{k,|G|}\big(1, Z(G, v)\big), 0, 0$$

a vector of 4 binary strings, the first being an encoding of the graph size; initial feature; and initial state, the second being an encoding of 1 and the initial feature, and the 3rd and 4th representing the final result and a 'finished' indicator. Define $\forall t \geq 0$ $C^{(t+1)}(G, v) \coloneqq$

$$f\Big(C^{(t)}(G, v)(1),\ \Sigma_{w \in N(v)} C^{(t)}(G, w)(2),\ C^{(t)}(G, v)[3, 4]\Big)$$

the value after $t + 1$ iterations. Define

$$I_v \coloneqq \begin{cases} \min(i : C^{(i)}(G, v)(4) = 1) & \text{min exists} \\ \infty & \text{otherwise} \end{cases}$$

$$C(G, v) \coloneqq \begin{cases} C^{(I_v)}(G, v)(3) & I_v \in \mathbb{N} \\ \text{undefined} & \text{otherwise} \end{cases}$$

That is, the result is the binary string at position 3, when the 'finished' indicator turns 1. We say that $C = (A, f)$ is a *Sum MP Graph Algorithm* (S-MP-GA) if $\forall G \in \mathcal{G}_{\mathcal{B}_k}$ $\forall\, v \in V(G)$ $\forall t \in [2|G|+1]$ it holds that $|\Sigma_{w \in N(v)} C^{(t-1)}(G, w)(2)| \leq O(3k|G|^4)$ i.e. the bit-length of the sum of neighbors' messages is bounded by $3k|G|^4$.

Besides having a sum aggregation, an S-MP-GA differs from an MP-LGA in two technical properties:

1. A message sent by a vertex consist of two parts, 'count' and 'value', rather than one. This is in order to count the number of sending vertices, which is useful later.
2. Two dimensions are used solely to define the final value, to make S-MP-GAs similar in that regard to R-GNNs.

The next lemma is a key step in our sequence of reductions.

Lemma 6.4.8. *Let $C = (A, f)$ be an MP-LGA then there exist A', f' such that $C' = (A', f')$ is an S-MP-GA and $\forall G \in \mathcal{G}_{\mathcal{B}}\ \forall v \in V(G)\ C'(G, v) = C(G, v)$. Furthermore, C' incurs polynomial time and space overhead.*

Proof. Unlike in the proof of Lemma 6.4.6, the emulating function f' is not very concise, hence we define it in pseudo-code style, in Listing 6.1. Each recurrence of C, where a node simply receives the multiset of its neighbors' features, requires $O(|G|^3)$ recurrences of C', where a node receives the sum of whatever its neighbors are sending, in order to extract the neighbors' individual features. The idea of such a phase of $O(|G|^3)$ recurrences is as follows:

1. For the first $|G|^2$ recurrences, vertices propagate max(own value, neighbors' messages average). The neighbors' average can be computed by dividing the sum of values by the sum of '1' each sender sends a dedicated dimension. Note that if there are two vertices with different values, the lower-value one will necessarily perceive an average higher than its own value - even if they are farthest from each other, after at most $|G|$ recurrences. Whenever a vertex perceives a higher average than its own value it temporarily disables itself and propagates the average, until the end of the first $|G|^2$ recurrences. This is implemented in the 'find_max' code. Hence, after $|G|^2$ recurrences, it is guaranteed that the vertices left enabled are exactly those with the maximum value in the whole graph - excluding those already counted for and permanently disabled (see below). The $|G|^2 + 1$ recurrence is used so that each vertex operates according to whether it is one of the max-value vertices. This is implemented in the 'send_if_max' code. Then, in the $|G|^2 + 2$ recurrence each vertex knows that the average it receives is actually the maximum value, and the count is the number of its neighbors with that value, and adds that information to the constructed multiset of neighbors' values. This is implemented in the 'receive_max' code. Then, all max-value vertices permanently disable themselves, all temporarily-disabled vertices re-enable themselves, and another $|G|^2$ recurrences phase starts - to reveal the next maximum-value and counts.
2. Since there are at most $|G|$ distinct values, after $|G|$ iterations of the process above, every vertex has finished constructing the multiset of its neighbors values. All is left to do then is to apply f of the MP-LGA on the (current value and) constructed multiset, and update the vertex's value to be the output of f.

The required message length limit is implied by the definition of the algorithm and the definition of δ. □

6.4.2 Reduction to R-GNN

Finally, we describe the reduction of S-MP-GA to R-GNN. The essential difference between the models is the recurring algorithm: In an S-MP-GA it can be any computable function i.e. a Turing machine, while in an R-GNN it is restricted to be an MLP.

Lemma 6.4.9. *Let $C = (B, h)$ be an S-MP-GA, then there exists an R-GNN $N = (A, F)$ such that $\forall G \in \mathcal{G}_{\mathcal{B}}\ \forall v \in V(G)\ C(G, v) = N(G, v)$. Furthermore, N incurs polynomial time and space overhead.*

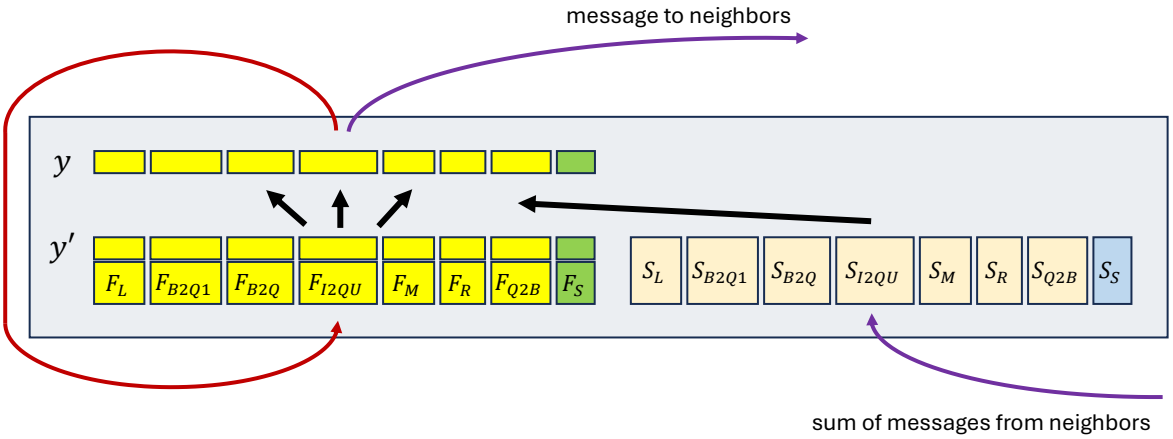


Figure 6.5: The structure of F. In yellow are all the switched sub-networks that pass control and data from one to another. In green is the synchronization sub-network that runs continuously. In beige and blue are the dimensions that assume the sum of neighbors' sub-networks output in the previous recurrence. Layer y' is the outputs of the individual sub-networks. These, together with the neighbors' sum, are inter-routed using layers $[y'+1..y]$, to create the desired interoperability. Finally, the output of layer y is sent to neighbors and also becomes (half) the input for the next recurrence.

Proof. Note that an R-GNN is, in a way, an extension of a recurrent MLP to the sum-aggregation message-passing setting. In [SS92] it is shown that recurrent MLPs are Turing-complete. Let M be a Turing-machine that computes h, we would like to use the result in [SS92] and emulate M using the recurrent MLP in an R-GNN. Yet, this requires overcoming two significant gaps:

1. An encoding gap. In [SS92] the emulation of a Turing machine is done by emulating a two-stack machine where a stack's content is always represented as a value in RQ. Since RQ is not closed under summation, a naive attempt to use the sum of the neighbors' stacks directly - as input to a Turing machine emulation is doomed to fail: The sum may be an invalid input and consequently the output will be wrong. To overcome this, we augment the Turing-machine-emulation recurrent MLP, F_M, with two recurrent MLPs, that compute the translations

   $$\text{RB2RQ} : \text{RB} \to \text{RQ},\ \text{RB2RQ} \coloneqq \mathfrak{rq} \circ \mathfrak{rb}^{-1}, \quad \text{RQ2RB} : \text{RQ} \to \text{RB},\ \text{RQ2RB} \coloneqq \mathfrak{rq} \circ \mathfrak{rb}^{-1}$$

   The former, F_{B2Q}, translates the received sum of messages, and the latter, F_{Q2B}, translates the new message to be sent. The number of recurrences required for each translation is quadratic for RB2RQ and linear for RQ2RB, in the number of translated bits. Existence of such recurrent MLPs is not trivial - the result in [SS92] assumes an input in RQ for a reason.

   The messages that are used in our algorithm have a fixed length, and consist of two parts which the receiver should be able to treat separately. When multiple messages are summed, those two parts can potentially interfere. However, we make sure that the message length is large enough to contain the sums of both parts separately, and assume

that the message's two parts are written in positions that guarantee no interference after messages' summation. The fixed message-length is $3kn^4$, which is more than the maximum-possibly-required $\log(n(2^{L_\delta(n,k)}+2^1))$ - the length of the sum of n complete pairs (mpc, 0/1-indicator). One of the inputs of the RB2RQ MLP is the fixed message-length, which is necessary for the RB2RQ algorithm. Note that k is known before constructing the overall network, and n is given as an initial feature of the vertex, however we provide a stronger result by describing a network that receives also k as an input hence supports all initial-feature lengths. We construct a recurrent MLP, F_L, that computes $3kn^4$ at the very beginning of the overall computation, which can then be used by the MLPs that require this value as one of their inputs.

In the beginning of the run, it is required to translate the initial-feature and graph-size, from a value in RB and an integer to one value in RQ - required by the Turing-machine-emulation F_M. First, another copy of an RB2RQ MLP, F_{B2Q1}, translates the feature, and then F_{I2QU} does both the translation of the graph size and the encoding of the two RQ values as a pair.

Finally, as the two final-output dimensions of an R-GNN - a "finished" indicator and the final feature of the vertex - are defined in Definition 6.4.1 to be a 0/1 value and a value in RB, we construct another translating MLP, F_R, that converts these outputs from RQ when the whole R-GNN computation is about to finish.

2. A synchronization gap. In an S-MP-GA computation, nodes are synchronized by definition: The i$^{\text{th}}$ recurrence's input is the the sum of results of the $i-1$ application of h to each of the neighbors. However, in an emulating R-GNN that is based on [SS92], every recurrence corresponds merely to a Turing machine step. As different nodes may require a different number of steps to complete the computation of a single application of h, when a node finishes its i^{th} emulation of h, in the t^{th} recurrence, its external input in recurrence $t+1$ is not necessarily the sum of its neighbors' i^{th} result, and it is unknown in what recurrence it will be. To overcome this, we augment the recurrent MLP described thus far with a recurrent MLP, F_S, that implements a synchronization algorithm across all nodes. That MLP runs for a the same number of recurrences for all nodes, hence, in itself it is synchronized.

Overall, the recurrent MLP F consists of 8 recurrent sub-networks:
$F_L, F_{B2Q_1}, F_{I2QU}, F_{B2Q}, F_M, F_R, F_{Q2B}, F_S$. For a graph and vertex $G \in \mathcal{G}_{\mathcal{B}_k},\ v \in V(G)$, the computation flow is as follows:

a. F_L computes the length of the messages, $3k|G|^4$, to be used by F_{B2Q}, then passes control to (b).

b. F_{B2Q_1} translates $Z(G,v)$ from RB to RQ, to be used by F_{I2QU}, then passes control to (c).

c. F_{I2QU} translates $|G|$ to its unary representation in RQ - base-4 '3' digits, and concatenates it to the base-4 '1' digit followed by the result of (b). That is, $x \to (\Sigma_{i\in[x]}\frac{3}{4^i}) + \frac{1}{4^{x+1}} + \frac{1}{4^{x+2}}\mathfrak{rq}(\mathfrak{rb}^{-1}\big(Z(G,v)\big))$. The output is to be used by F_M. Then it passes control to (1.1).

1.1. F_{B2Q} translates the sum of messages, from RB to RQ, then passes control to (1.2).

1.2. F_M computes the Turing machine function, then passes control to (1.3).

1.3. F_R Checks whether the overall computation - the function computed by the R-GNN - is finished, and if it is, translates the final output - final feature of the vertex - from RQ to RB. Note that in such case this output is "locked" - it will not change over subsequent recurrences. If the overall computation is not finished, control is passed to (1.4).

1.4. F_{Q2B} translates the new message to send, computed by F_M, from RQ to RB, then stops.

2. In parallel to (1), F_S operates continuously. Every $2n$ recurrences it produces a signal whether all nodes' previous computation is finished already n recurrences before. If the signal is positive, the sequence of (1) is restarted.

We proceed to formalize the structure of the whole R-GNN network, that is, the structure of each sub-network and how theses sub-networks are put together. We describe them not necessarily in their order of operation - described above - but also in order of similarity in functionality.

We would like the sub-networks $F_L, F_{B2Q_1}, F_{I2QU}, F_{B2Q}, F_M, F_R, F_{Q2B}$ to operate in two modes, 'On' - where they compute what they are supposed to (probably over multiple recurrences) and have the results as the values of certain output neurons, and 'Off' - where the results-outputs are remained unchanged. Switching to 'On' mode should be initiated by an external change to an 'On/Off' switch neuron, at which time certain preconditions must hold for the rest of the state neurons of the sub-network in order to assure that it indeed computes what it is supposed to. Switching to 'Off' mode is part of the sub-network's results when finishing its current computation, which in turn causes another sub-network to switch on, and so on and so forth. The goal is that at any recurrence of the whole R-GNN, only one sub-network, excluding F_S, is actually computing its function. We formalize the concept of a switched network, in the following definition.

Definition 6.4.10 (Switched Recurrent MLP)**.** A *switched* recurrent MLP (switched MLP) $F = (((w_1, b_1), \ldots, (w_m, b_m)), D, f)$, of d dimension, is a sequence of rational matrices and bias vectors - similar to a recurrent MLP; A set $D \subseteq \{0,1\} \times \{0,1\} \times \mathbb{Q}^{d_I+d_O} \times \mathbb{Q}^{d-d_O-d_I-2}$;
and a function $f : \mathbb{Q}^{d_I} \to \mathbb{Q}^{d_O}, 2 + d_I + d_O \leq d$. The $d - d_I - d_O$ dimensions are neurons that only remember the state of the computation. The semantics are that the first input dimension is a switch, the second is an indication if the switch has turned off from previous iteration, D is a set of valid initial states, and f is a target function whose input(s) start at the third input dimension, and whose output(s) start after the inputs. More formally:

1. $\forall x \in D \;\; x(1) = 0 \Rightarrow$
$(f_F^{(1)}(x)(1) = 0 \wedge f_F^{(1)}(x)(2) = 0 \wedge f_F^{(1)}(x)[2 + d_I + 1, 2 + d_I + d_O] = x[2 + d_I + 1, 2 + d_I + d_O] \wedge f_F^{(1)}(x) \in D)$
That is, for every valid input such that the first dimension is 0, the MLP remains "switched off": Switch is 'Off', turned-off indicator is 0, target-function outputs maintain their value, and the overall output is a valid initial state - ready for being switched on.

2. For $x \in D$, define $t_x := min\big(t : \forall t' \geq t \; f_F^{(t')}(x) = f_F^{(t)}(x)\big) - 1$. Then,

$$\forall x \in D \;\; x(1) = 1 \Rightarrow$$

$$(f_F^{(t_x)}(x)(1) = 0 \wedge f_F^{(t_x)}(x)(2) = 1 \wedge f_F^{(t_x)}(x)[2{+}d_I{+}1, 2{+}d_I{+}d_O] = f(x[3, 2{+}d_I]) \wedge f_F^{(t_x)}(x) \in D)$$

That is, for every valid input such that the first dimension is 1, the MLP is "switched on" and: After enough recurrences, the first output turns 0 - switching off; the switched-off indicator turns on; the function outputs are the value of the target function applied to the function inputs; and the overall output is a valid initial state. Note that by (1) we have that $f^{(t_x+1)}(x)$ is the same as $f^{(t_x)}(x)$ except for the second dimension which is 0 and not 1, because the switch does not turn off in recurrence $t_x + 1$ - it was already off.

Note that while an external input is required to initiate a computation - set on the 'On/Off' switch and initialize the inputs, it is only a preceding stage to the network's operation and not in the scope of its definition. This is in contrast to Definition 6.4.18 where new external inputs are used in every recurrence throughout the computation and hence are part of the networks' specification.

Constructing F_M

Remember that we wish the R-GNN to compute the same function as a given S-MP-GA $C = (B, h)$. The first sub-network we describe is the main switched sub-network: At the core, it is a recurrent MLP that emulates the Turing-machine that computes h, which exists by [SS92]. Then, that MLP is wrapped so to make it a switched recurrent MLP.

Lemma 6.4.11. *Let $f : \mathcal{B}^4 \to \mathcal{B}^4$ be a computable function, and let $g : RQ^4 \to RQ^4$ such that*

$$\forall x \in \mathcal{B}^4 \;\; g\big(\mathfrak{rq}(x)\big) = \big(\mathfrak{rq}(f(x))\big)$$

Let $S \coloneqq \{y \in RQ^4 \mid \exists x \in \mathcal{B}^4 \; y = \mathfrak{rq}(x)\}$ be the set of valid inputs to g. Define

$$\forall d' \in \mathbb{N} \;\; D_{d'} \coloneqq \{x \mid x \in \{0,1\}^2 \times S \times RQ^4 \times \{0,1\} \times S \times \{0\}^{d'-4}\}$$

Then, for some $d' \in \mathbb{N}$ there exist a $(d'+11)$-dimension switched network

$$F = \Big(\big((w_1, b_1), \ldots, (w_m, b_m)\big), D_{d'}, g\Big)$$

Proof. Let $d' \in \mathbb{N}$ so there exists a d'-dimension recurrent MLP $F' = (l'_1, \ldots, l'_{m'})$ such that

$$\forall x \in S \times \{0\}^{d'-4} \; \exists k \in \mathbb{N} :$$
$$(\forall t < k \; F'^{(t)}(x)(5) = 0), \; F'^{(k)}(x)(5) = 1, \; F'^{(k)}(x)[1, 4] = g(x)$$

Such F' exists by [SS92, Thm 2 and Section 4.3]. We define a switched network

$$F = \big((l_1, \ldots, l_m), D_{d'}, g\big)$$

of dimension $d \coloneqq (d' + 11)$ and depth $m \coloneqq (m' + 5)$, as follows. The idea is to wrap a copy of F' with 2 pre-processing and 3 post-processing layers, and additional dimensions - that relay the information between the pre and post processing parts. Note that later we describe how F_M itself is wrapped when combined into the whole network of the R-GNN. We denote: The d-dimension input vector to the first layer by x; the d-dimension input

vector to the third layer - the layer in which the copy of F' starts, by x'; the output vector of layer $m'+2$, in which the copy of F' ends, by y'; and the d-dimension output of the last layer, which is the input vector for the next recurrence, by y.

The first 11 dimensions represent F's switch; switch-turned-off indicator; function inputs; function outputs; and previous-recurrence switch-value. Initially, $x(1) = x(2) = x(11) = 0$ i.e. switch-related values are 0. When F is swithced on i.e. $x(1) = 1, x(2) = 0, x(11) = 0$, we want the inputs to become F' inputs and we want to reset the rest of F' dimensions to 0, and so we define

$$x'[12,15] \coloneqq \text{lsig}(x[3,6] - \text{lsig}(1 - \text{relu}(x(1) - x(11)))) + \text{lsig}(x[12,15] - \text{lsig}(\text{relu}(x(1) - x(11))))$$

$$x'[16,d] \coloneqq \text{lsig}(x[16,d] - \text{lsig}(\text{relu}(x(1) - x(11))))$$

We also want the switch-turned-off indicator to always remember the switch state at the beginning of the recurrence, and so we define $x'(2) \coloneqq x(1)$. Considering that the activation function of our network is relu, x' can be computed from x using at most 2 layers. These would be l_1, l_2.

On the output end, if the input switch was 0, we want all outputs of F' to be zero - so it is ready for a new run. Otherwise: If F' is finished i.e. $y'(16) = 1$ then we want F' function outputs i.e. $y'[12,15]$ to be the output of F i.e. $y[7,10]$, its switch to turn off, and the switch-turned-off indicator to be 1. If F' is not finished then we want to pass all outputs of F' as they are to next recurrence. The above logic can be implemented as follows:

$$\begin{aligned}
&y(1) = \text{relu}(y'(1) - y'(16)) \text{ (if } F' \text{ finished then turn off)} \\
&y(2) = \text{relu}(y'(2) - \text{relu}(y'(1) - y'(16))) \text{ (if } F' \text{ just turned off)} \\
&y(11) = \text{relu}(y'(1) - y'(16)) \\
&y[7,10] = \text{relu}\left(y'[12,15] - \text{relu}\left(1 - y'(16)\right)\right) + \text{relu}\left(y'[7,10] - y'(16)\right)
\end{aligned}$$

Considering that the activation function of our network is relu, y can be computed from y' using at most 3 layers. These would be $w_{2+m'+1}, w_{2+m'+2}, w_{2+m'+3}$. Layers $[3..m'+2]$ are essentially 2 separate stack-of-layers side-by-side, as follows:

1. The first stack is F's (switch, switch-turned-off indicator, function inputs, function outputs, previous switch), in dimensions [1..11], each simply passing on from layer to layer.
2. The second column is F' as is, set in layers $[3..m'+2]$.

□

For F_M, we did not explicitly describe the core network that implements the required functionality. Instead, we deduced its existence, from the result in [SS92]. However, the mode of operation of a network a-la [SS92] does not match the mode of operation of an R-GNN. As the combined functionality of $F_L, F_{B2Q_1}, F_{I2QU}, F_{B2Q}, F_M, F_R, F_{Q2B}$ is meant to bridge that gap, the descriptions of these sub-networks cannot themselves be based on [SS92] since this would just move the gap to a different location in the algorithm. Hence, we describe each of these sub-networks explicitly: We provide a pseudo-code

that implements the required functionality when ran recurrently, and meets a certain specification which implies implementabilty by an MLP. The specification consists of a structure and an instruction-set, which together define an MLP implementation, as follows.

Programming MLPs

Definition 6.4.12 (MLP Code)**.** An *MLP Code* having d state-variables i.e. variables that should maintain their value from the end of one recurrence to the beginning of the next, has the following properties which together imply a d-dimension MLP implementation.

- The code has two sections:
  An 'Initialize' section. There, the d state-variables, which correspond to the I/O neurons of the MLP, are defined. Their names are prefixed by 's_', and their initial values - before the R-GNN run starts - are defined as well. These must be non-negative.
  A 'RecurrentOp' section. There, is the code that defines a single recurrence of an implementing MLP. It consists solely of statements that conform to one of the following definitions of an *expression*. The invariant of these definitions is that for every expression there exists an MLP sub-structure whose inputs are the statement's operands and whose output is the value of the statement.

  - The state-variables are expressions. They correspond to the input neurons, and to subsequent dedicated neurons in each layer. That is, for each state-variable there is a corresponding neuron in each layer.
  - For the sake of code-readability, intermediate variables may also be used, and are considered expressions. Declared using the keyword 'var', they correspond to a specific neuron whose value is defined by an assignment to the variable.
  - Let $x_1, \ldots, x_n$ be expressions, let $w_1 \ldots, w_n \in \mathbb{Q}, b_1 \ldots, b_n \in \mathbb{Q}$, and define $x \coloneqq \Sigma_{i\in[n]} w_i x_i + b_i$, then:
    1. Both $\text{relu}(x), \text{lsig}(x)$ are expressions, regardless of the sign of x. Note that lsig is expressible by combining two relus.
    2. If $x \geq 0$ then x is an expression.
  - Let v, x be a variable (expression) and an expression, then $v = x$ is an expression. Note that such assignment is implemented by feeding the neuron that represents x into a next-layer neuron that represents v.
  - Let v be a variable, and let $0 \leq x \leq 1, s \in \{0, 1\}$ be expressions representing a change and a state. Then, the conditional addition

$$v.\text{increase\_if}(s, x) \coloneqq \begin{cases} v + x & s = 1 \\ v & s = 0 \end{cases}$$

    is an expression, and if $v - x > 0$ then also the conditional subtraction,

$$v.\text{decrease\_if}(s, x) \coloneqq \begin{cases} v - x & s = 1 \\ v & s = 0 \end{cases}$$

is an expression. Note that due to the restrictions $0 \leq x \leq 1, s \in \{0,1\}$, $v.\text{increase\_if}(s,x) \equiv v+\text{relu}(x-(1-s))$, Similar argument holds for $v.\text{decrease\_if}(s,x)$.

- Let $0 \leq v \leq 1$ be a variable, let $a \in \mathbb{N}$ be a constant, and let $s \in \{0,1\}$ be an expression. Then, the conditional division

$$v.\text{div}a\text{\_if}(s) \coloneqq \begin{cases} \frac{v}{a} & s=1 \\ v & s=0 \end{cases}$$

is an expression. Note that due to the restrictions on the arguments, $v.\text{div}a\text{\_if}(s) \equiv v - \text{relu}((1-\frac{1}{a})v-(1-s))$, which is expressible by a neuron.

- Let $0 \leq v \leq 1$ be a variable, let $0 \leq x \leq 1$ be an expression, and let $s \in \{0,1\}$ be another expression. Then, the conditional assignment

$$v.\text{set\_if}(s,x) \coloneqq \begin{cases} x & s=1 \\ v & s=0 \end{cases}$$

is an expression. Note that due to the restrictions on the arguments, $v.set\_if(s,x) \equiv v.decrease\_if(s,v)+\text{relu}(x-(1-s))$, which is expressible by 2 neurons in one layer and another neuron in the next layer.

Constructing F_L

Lemma 6.4.13. *Let $a \in \mathbb{Q}$, then there exist $d \in \mathbb{N}$ for which there is a d-dimension switched network*

$$F = \big((l_1,\ldots,l_m), \{(1,0),(0,1),(0,0)\} \times \mathbb{N}^2 \times \mathbb{Q}^8, (x_1,x_2) \to ax_1x_2^3\big)$$

That is, there exists a switched network that for every $a \in \mathbb{Q}, x_1, x_2 \in \mathbb{N}$ computes $ax_1x_2^3$.

Proof. Following the code in Listing 6.2, which conforms to Definition 6.4.12, it is not difficult to verify that it defines a 12-dimension switched network

$$F = \Big(\big((w_1,b_1),\ldots,(w_m,b_m)\big), \{(1,0),(0,1),(0,0)\} \times \mathbb{N}^2 \times \mathbb{Q}^8, (x_1,x_2) \to ax_1x_2^3\Big)$$

for some rational matrices and biases (w_i,b_i). □

Constructing F_{B2Q_1}

The sub-network F_{B2Q_1} is identical to F_{B2Q} which is described later. The difference is that F_{B2Q_1} is used only once, in the initialization stage of the whole $R-GNN$ run. There, it converts the input feature, provided by the user in RB encoding, to RQ encoding, which is then used by F_{I2QU}.

Constructing F_{I2QU}

Lemma 6.4.14. *Define* $D \coloneqq \{(1,0),(0,1),(0,0)\} \times \mathbb{N} \times RQ \times \mathbb{Q}_{[0,1]} \times \{0\}\}$*, and define* $\forall x \in \mathbb{N}\ \forall y \in RQ\ \ g(x,y) \coloneqq (\frac{1}{4^{x+1}} + \frac{y}{4^{x+2}} + \Sigma_{i\in[x]}\frac{3}{4^i})$*. Then, there exists a switched network*

$$F = ((l_1, \ldots, l_m), D, g)$$

That is, there exists a switched network that given $x \in \mathbb{N}$ *and* $y \in RQ$ *outputs the translation of* x *to Unary Rational Quaternary concatenated with the digit '1' in RQ concatenated to* y*.*

Proof. Following the code in Listing 6.3, which conforms to Definition 6.4.12, it is not difficult to verify that it defines the required switched network. □

Constructing F_{B2Q}

Lemma 6.4.15. *Define*

$$\forall L \in \mathbb{N}\ \ D_L \coloneqq \{(1,0),(0,1),(0,0)\} \times \{L\} \times \{x : x = \mathfrak{rb}(y), y \in \mathcal{B}_L\} \times \mathbb{Q}_{[0,1]} \times \mathbb{Q}^{12}\}$$

Then, there exists a recurrent MLP $F = (l_1, \ldots, l_m)$ *such that* $\forall L \in \mathbb{N}$ *it holds that* $S_L = (F, D_L, \mathfrak{rq} \circ \mathfrak{rb}^{-1})$ *is a switched network. That is, there exists a recurrent MLP such that given a message length* $L \in \mathbb{N}$ *and a rational binary encoding of a message* $x \in \mathcal{B}_L$*, outputs the message's rational quaternary encoding.*

Proof. Following the code in Listing 6.4, which conforms to Definition 6.4.12, it is not difficult to verify that it defines the required switched network. □

Constructing F_{Q2B}

Unlike F_{B2Q}, F_{Q2B} does not have the message length as input, since the RQ encoding allows us to identify, without receiving the length explicitly, when we have read all digits and the message is empty: The value of an empty message is 0 while the value of any non-empty message is at least $\frac{1}{4}$.

Lemma 6.4.16. *Define* $D \coloneqq \{(1,0),(0,1),(0,0)\} \times RQ \times \mathbb{Q}_{[0,1]} \times \{0\} \times \mathbb{Q}_{[0,1]}\}$*, then there exists a switched network*

$$F = ((l_1, \ldots, l_m), D, \mathfrak{rb} \circ \mathfrak{rq}^{-1})$$

That is, there exists a switched network that for every $x \in \mathcal{B}$ *translates its rational quaternary encoding to its rational binary encoding.*

Proof. Following the code in Listing 6.5, which conforms to Definition 6.4.12, it is not difficult to verify that it defines the required switched network. □

Constructing F_R

Lemma 6.4.17. *Define* $D \coloneqq \{(1,0),(0,1),(0,0)\} \times RQ^2 \times \mathbb{Q}^2_{[0,1]} \times \{0\} \times \mathbb{Q}_{[0,1]}\}$*, then there exists a switched network*

$$F = ((l_1, \ldots, l_m), D, (1, \mathfrak{rb} \circ \mathfrak{rq}^{-1}))$$

That is, there exists a switched network that for every $x \in \mathcal{B}$ *outputs the indication* 1 *and the translation of* x *from its rational quaternary encoding to its rational binary encoding.*

Proof. Following the code in Listing 6.6, which conforms to Definition 6.4.12, it is not difficult to verify that it defines the required switched network. □

The last sub-network we describe, F_S, is not a switched network. It is 'on' in every recurrence, in parallel to whatever switched network is 'on' in that recurrence. Also, part of its input dimensions are external i.e. they do not assume their output-layer value from the previous recurrence, rather, they are set externally at the beginning of each recurrence. We formally define this kind of network as follows.

Definition 6.4.18 (Recurrent MLP With External Inputs)**.** A d-dimension m-depth recurrent MLP with external inputs
$F = ((l_1, \ldots, l_m), z_1, \ldots, z_L)$ is a variation of a d-dimension recurrent MLP, such that the last L dimensions are external input sequences $\{z_i^{(t)}\}_{i \in [L], t \in \mathbb{N}}$. That is,

$$f_F^{(0)}(x) \coloneqq x \in \mathbb{Q}^d, \quad \forall t > 0 \quad f_F^{(t)}(x) \coloneqq f_F\Big(f_F^{(t-1)}(x)[1, d-L], \big(z_1(t), \ldots, z_L(t)\big)\Big)$$

Constructing F_S

The sub-network F_S implements a mechanism that synchronizes the nodes' Turing-machine emulations. The idea is that at specific recurrences, all nodes record and broadcast their status. Then, for the next $|G|$ recurrences they consider their own recording and the messages they receive, and broadcast whether there is an indication of a non-finished node. This means that if and only if there is such node then all nodes will be aware of it at the end of those $|G|$ recurrences - because it takes at most $|G|$ recurrences for the information to propagate. In other words, all nodes have the same global status-snapshot at the end of those $|G|$ recurrences.

If according to the snapshot all nodes' Turing machines are finished, then all nodes start their new computation in the next recurrence - exactly at the same time. In addition, we construct the R-GNN such that when a node's Turing machine is finished the node continues to send the same result-message in subsequent recurrences until starting a new computation. The above scheme assures that all nodes' Turing machines start computation $i+1$ - emulating iteration $i+1$ of the emulated S-MP-GA algorithm - with input which contains the sum of their neighbors' i^{th}-computation results.

Due to technical limitations of computing with MLP, F_S runs in cycles of $2|G|$ recurrences instead of $|G|$: The first $|G|$ recurrences are used to reset its state, and the second $|G|$ recurrences implement the scheme above. It has three external inputs: The graph size; the sum of neighbors' indicator that "there is a node that had not finished yet at the beginning of the cycle"; a 'finished' indicator for the node's current computation, and two

outputs: "should start new computation" - used by F_{B2Q}; and "there is a node that had not finished yet at the beginning of the cycle" indicator - to be sent to neighbors. The first output is 0 throughout the cycle except for the last recurrence where it is 1 in case all nodes are finished, and the second output is 0 throughout the first half of the cycle - where it is basically disabled - and becomes 1 in the second half in case an indication of a non-finished node was received.

The following Lemma formalizes the described behavior.

Lemma 6.4.19. *Let $n \in \mathbb{N}$, then there exist a d-dimension recurrent with external inputs MLP*
$F = ((l_1, \ldots, l_m), z_1, z_2, z_3)$ such that:

$$f_F^{(t)}(1) \coloneqq \begin{cases} 0 & t \mod 2n > 0 \\ 1 - f_F^{(t-1)}(2) & t \mod 2n = 0 \end{cases}$$

$$f_F^{(t)}(2) \coloneqq \begin{cases} 0 & t \bmod 2n \leq n \\ \min(1, \Sigma_{i=t-(t \bmod n)}^{t}(z_2^{(i)} + z_3^{(i)})) & t \bmod 2n > n \end{cases}$$

Proof. Following the code in Listing 6.7, which conforms to Definition 6.4.12, it is not difficult to verify that it defines the required recurrent network with external inputs. □

Constructing The R-GNN $N = (A, F)$

We are now ready to put together all the sub-networks above into one recurrent MLP F. Their I/O dimension, initial values - corresponding to their part in the A vector, and individual functionality, are already defined in their dedicated sections above. Here we describe how they are connected to one another to form the whole recurrent network F. We refer to them by their names in the sections describing their construction. Please refer to Figure 6.5 for an illustration of the construction.

Remark 6.4.20. Note that the positions of the inputs to the various sub-networks, as implied by the sub-networks' definitions and the construction below, do not match their positions according to the definition of $N^{(0)}(G, v)$ in Definition 6.4.1. The reason for the permutation is so both the definition of $N^{(0)}(G, v)$ and the description of the construction are easier to follow. Reversing the permutation e.g. by adding a prelude and postlude layers is trivial and for purpose of focus we do not include it.

For a d-dimension m-depth sub-network F_H we define $d(F_H) \coloneqq d; m(F_H) \coloneqq m$. Note that when referring to the dimensions of a sub-network we mean its I/O dimensions. Whatever are the dimensions and arguments of its hidden layers, they do not make a difference to our construction of F. Assume that F_H is embedded in F such that dimensions $a..(a + d(F_H) - 1)$ in the input layer of F are the input layer of F_H. We define $E(F_H) \coloneqq (a + d(F_H) - 1)$ the last dimension of the part of F_H in F.

Define $m_{\max} \coloneqq max(m(F_H) \mid \text{H is a sub-network})$ the maximum depth over all sub-networks. We assume all sub-networks of lower depth are extended to depth $m_{\max}$ by simply passing on their output layer, hence we can refer to layer $m_{\max}$ of any sub-network. We denote by $y'_{F_H}(i)$ the i^{th} dimension in the output layer of the F_H part of F, that is, the $a + i - 1$ neuron in layer $m_{\max}$ of F. To implement the interoperability between the

sub-networks, we add additional layers after y', up to the final output layer which we denote by y. The interconnections are defined by describing certain dimensions in y as functions of (also) dimensions in other sub-networks' outputs and the aggregation values - in y'. Dimensions in y for which we do not define functions, simply assume their values in y' as is. We denote by $y_{F_H}(i)$ the $a+i-1$ neuron in the last layer of F - whose output in one recurrence is the value of the $a+i-1$ input neuron in the following recurrence. Note that y, y' are not to be confused with the y, y' layers mentioned in the individual description of F_M.

Let H be the unique name of one of the sub-networks e.g. $H = B2Q$. We denote by S_H the dimensions of F that are the sum of the neighbors' F_H. For example, $S_M(1)$ is the sum, over all neighbors, of the first dimension of F_M's output in the previous recurrence, and is in dimension $E(F_S) + E(F_{B2Q}) + 1$ in F. As described later, the sum values are used as inputs, only in the extra layers of F - after the y' layer. That is, they simply pass on from the input layer until the last layer, and some of their neurons are inputs to other parts of F in the extra layers after y'.

And so, the whole flow is established in which at every recurrence of F only specific sub-networks are active, and eventually their outputs become inputs for other sub-networks which in turn become the active ones etc.

Subnetwork F_L, dimensions $[1..d(F_L)]$

Subnetwork F_{B2Q_1}, dimensions $[d(F_L)+1..d(F_L)+d(F_{B2Q_1})]$

$y_{F_{B2Q_1}}(1) \coloneqq \text{relu}\left(y'_{F_{B2Q_1}}(1) + y'_{F_L}(2)\right)$ i.e. F_{B2Q_1} should start after F_L is finished.

$y_{F_{B2Q_1}}(3) \coloneqq \text{relu}\left(y'_{F_L}(3) - \left(1 - y'_{F_L}(2)\right)\right) + \text{relu}\left(y'_{F_{Q2B_1}}(3) - \left(y'_{F_L}(2)\right)\right)$ i.e. F_{Q2B_1} feature length input is set to be the dimension with that data used also by F_L, for the beginning of a new computation, otherwise it maintains its value.

Subnetwork F_{I2UQ}, dimensions $[E(F_{B2Q_1})+1..E(F_{B2Q_1})+d(F_{I2UQ})]$

$y_{F_{I2UQ}}(1) \coloneqq \text{relu}\left(y'_{F_{I2UQ}}(1) + y'_{F_{B2Q_1}}(2)\right)$ i.e. F_{I2UQ} should start after F_{B2Q_1} is finished.

$y_{F_{I2UQ}}(3) \coloneqq \text{relu}\left(y'_{F_L}(4) - \left(1 - y'_{F_{B2Q_1}}(2)\right)\right) + \text{relu}\left(y'_{F_{I2UQ}}(3) - \left(y'_{F_{B2Q_1}}(2)\right)\right)$ i.e. F_{I2UQ} graph-size input is set to be the dimension with that data used also by F_L, for the beginning of a new computation, otherwise it maintains its value.

$y_{F_{I2UQ}}(4) \coloneqq \text{relu}\left(y'_{F_{B2Q_1}}(5) - \left(1 - y'_{F_{B2Q_1}}(2)\right)\right) + \text{relu}\left(y'_{F_{I2UQ}}(4) - \left(y'_{F_{B2Q_1}}(2)\right)\right)$ i.e. F_{I2UQ} num-in-RQ input is set to be the output of F_{B2Q_1}, for the beginning of a new computation, otherwise it maintains its value.

Subnetwork F_{B2Q}, dimensions $[E(F_{I2UQ})+1..E(F_{I2UQ})+d(F_{B2Q})]$

$y_{F_{B2Q}}(1) \coloneqq \text{relu}\left(y'_{F_{B2Q}}(1) + y'_{F_S}(1) - y'_{F_{I2UQ}}(1)\right)$ i.e. F_{B2Q} should start only after F_{I2UQ} has switched off, and from then on it should be on either if it is in the middle of a computation or if it received a signal from F_S that it can start again. Note that a signal from F_S means that $F_M; F_{Q2B}$ are off.

$y_{F_{B2Q}}(3) \coloneqq y'_{F_L}(5)$ i.e. F_{B2Q} should take $L_{n,k}$ from the first output of F_L.

$y_{F_{B2Q}}(4) \coloneqq \text{relu}\left(y'_{S_M}(8) - y'_{F_S}(1) - y'_{F_{I2UQ}}(1)\right)$ i.e. F_{B2Q} should take new input from the sum of neighbors (at position: second output of F_M) when it is signaled to start again.

Subnetwork F_M, dimensions $[E(F_{B2Q})+1..E(F_{B2Q})+d(F_M)]$

$y_{F_M}(1) \coloneqq \text{relu}\left(y'_{F_M}(1) + y'_{F_{B2Q}}(2)\right)$ i.e. F_M should turn on after F_{B2Q} turns off.

$y_{F_M}(3) \coloneqq \text{relu}\left(y'_{F_M}(7) - \left(1 - y'_{F_{B2Q}}(2)\right) + \text{relu}\left(y'_{F_M}(3) - \left(y'_{F_{B2Q}}(2)\right)\right)\right.$ i.e. F_M first input is set to be the first output for the beginning of a new computation, otherwise it

maintains its value.

$y_{F_M}(4) \coloneqq \text{relu}\left(y'_{F_{B2Q}}(5) - \left(1 - y'_{F_{B2Q}}(2)\right)\right) + \text{relu}\left(y'_{F_M}(4) - \left(y'_{F_{B2Q}}(2)\right)\right)$ i.e. F_M second input is set to be the output of F_{B2Q} for the beginning of a new computation, otherwise it maintains its value.

$y_{F_M}(7) \coloneqq \text{relu}\left(y'_{F_{I2UQ}}(5) - \left(1 - y'_{F_{I2UQ}}(2)\right) + \text{relu}\left(y'_{F_M}(7) - \left(y'_{F_{I2UQ}}(2)\right)\right)\right.$ i.e. F_M first output gets the result of F_{I2UQ} and maintains it as long as there is no computation, so when the first computation starts $y_{F_M}(3)$ will read the the result of F_{I2UQ}.

Subnetwork F_R, dimensions $[E(F_M) + 1..E(F_M) + d(F_R)]$

$y_{F_R}(1) \coloneqq \text{relu}\left(y'_{F_R}(1) + y'_{F_M}(2)\right)$ i.e. F_R should turn on after F_M turns off.

$y_{F_R}(3) \coloneqq \text{relu}\left(y'_{F_M}(10) - \left(1 - y'_{F_M}(2)\right)\right) + \text{relu}\left(y'_{F_R}(3) - \left(y'_{F_M}(2)\right)\right)$ i.e. F_R 'allFinished' input is set to be the last output of F_M upon start.

$y_{F_R}(4) \coloneqq \text{relu}\left(y'_{F_M}(9) - \left(1 - y'_{F_M}(2)\right)\right) + \text{relu}\left(y'_{F_R}(4) - \left(y'_{F_M}(2)\right)\right)$ i.e. F_R 'final feature' input is set to be the one before last output of F_M upon start.

Subnetwork F_{Q2B}, dimensions $[E(F_R) + 1..E(F_R) + d(F_{Q2B})]$

$y_{F_{Q2B}}(1) \coloneqq \text{relu}\left(y'_{F_{Q2B}}(1) + y'_{F_R}(2) - y'_{F_R}(5)\right)$ i.e. F_{Q2B} should turn on after F_R turns off but not if overall computation of the R-GNN is finished.

$y_{F_{Q2B}}(3) \coloneqq \text{relu}\left(y'_{F_M}(8) - \left(1 - y'_{F_R}(2)\right)\right) + \text{relu}\left(y'_{F_{Q2B}}(3) - \left(y'_{F_R}(2)\right)\right)$ i.e. F_{Q2B} input is set to be the second output of F_M for the beginning of a new computation, otherwise it maintains its value.

Subnetwork F_S, dimensions $[E(F_{Q2B}) + 1..E(F_{Q2B}) + d(F_S)]$

$y_{F_S}(d(F_S) - 2) \coloneqq y_{F_L}(2)$ i.e. F_S first input is the graph size, which is found constantly in the second dimension of F_L.

$y_{F_S}(d(F_S) - 1) \coloneqq \Big(1 -$

$\left(y'_{F_{B2Q}}(1) + y'_{F_M}(1) + y'_{F_{Q2B}}(1) + y'_{F_{B2Q}}(2) + y'_{F_M}(2) + y'_{F_{Q2B}}(2) + y'_{F_R}(2)\right)\Big)$ i.e. F_S second input should be 1 if the whole computation process is currently off i.e. most recent computation is finished. The reason for sampling both the "switch on" and "switch just turned off" dimensions is to avoid getting into the analysis of corner-case-timing cases.

$y_{F_S}(d(F_S)) \coloneqq y'_{S_S}(2)$ i.e. F_S third input should be the sum of the neighbors' F_S indication if it has received a "not-finished" signal. □

Listing 6.1: Emulate MP-LGA

```

    Initialize: // impelmentation of C^(0)(G, v)
        dim1.graph_size = graph_size
        dim1.feature = feature
        dim1.disabled = false
        dim1.tmp_disabled = false
        dim1.neighbors_values_and_counts = {}
        dim1.inner_loop_counter = 0
        dim1.outer_loop_counter = 0
        dim1.receive_max = 0
        dim1.MP_GC_dim1 = MP_GC_init_dim1
        dim1.MP_GC_iteration_count = 0
        dim2.count = 1
        dim2.value = feature
        dim3 = 0
        dim4 = 0

    run(prev_dim1, neighbors_dim2_sum, prev_dim3){
        output_dim1 = prev_dim1.copy() // start with a copy, then set what needs to be updated
        output_dim3 = prev_dim3 // used to indicate finishing the overall computation, start with
            ↪ same value as previous
        output_dim4 = prev_dim4 // used to hold the final value when the computation is finished
        if(prev_dim1.MP_GC_iteration_count == prev_dim1.graph_size+1){// finished whole
            ↪ computation, output_dim2.value should have the final output
            output_dim3 = 1
            output_dim4 = prev_dim1.feature
        }
        else if(prev_dim1.outer_loop_counter == prev_dim1.graph_size){// finished collecting multiset
            ↪ of neighbors' features, run MP_GC_func
            (MP_GC_output_dim1, MP_GC_output_dim2) = MP_GC_func(prev_dim1.MP_GC_dim1,
                ↪ prev_dim1.neighbors_values_and_counts)
            output_dim1.MP_GC_dim1 = MP_GC_output_dim1
            output_dim1.feature = MP_GC_output_dim2
            output_dim1.MP_GC_iteration_count = prev_dim1.MP_GC_iteration_count + 1
            output_dim1.neighbors_values_and_counts = {}
            output_dim1.inner_loop_counter = 0
            output_dim1.outer_loop_counter = 0
        }
        else if(prev_dim1.outer_loop_counter < prev_dim1.graph_size){// still collecting
            if(prev_dim1.inner_loop_counter==prev_dim1.graph_size^2){ // finished isolating max
                ↪ among uncollected
                send_if_max(prev_dim1, output_dim1, output_dim2)
                output_dim1.receive_max = 1 // next stage is to read neighbors that sent max
                output_dim1.inner_loop_counter = 0
            }
            else if(prev_dim1.receive_max == 1){
                receive_max(prev_dim1, neighbors_dim2_sum, output_dim1, output_dim2)
                output_dim1.receive_max = 0
                output_dim1.outer_loop_counter +=1
            }
            else{
                find_max(prev_dim1, neighbors_dim2_sum, output_dim1, output_dim2)
                output_dim1.inner_loop_counter +=1
            }
```

```
50          }
51      }
52
53      find_max(prev_dim1, neighbors_dim2_sum, output_dim1, output_dim2){
54          neighbors_avg_value = neighbors_dim2_sum.value / neighbors_dim2_sum.count
55          output_dim=2 = 1
56          // if the vertex is disabled (or tmpDisabled) or its initial feature is lower than the observed
                ↪ value, then propogate the observed value
57          output_dim2.value = neighbors_avg_value
58          if (prev_dim1.feature < neighbors_avg_value)
59          {
60              // vertex value is lower, then tmpDisable vertex so eventually the only non−tmpDisabled
                    ↪ vertices will be those with maximum value among the enabled vertices in the
                    ↪ graph.
61              output_dim1.tmp_disabled = true
62          }
63          else if (!prev_dim1.disabled && !prev_dim1.tmp_disabled)
64          {
65              // if the vertex is enabled and its initial feature higher than the observed value then
                    ↪ propogate its value
66              output_dim2.value = prev_dim1.feature
67          }
68      }
69
70      send_if_max(prev_dim1, output_dim1, output_dim2){
71          if (!prev_dim1.disabled && !prev_dim1.tmp_disabled)
72          { // vertex is one of those with max value among the enabled, send its value to its neighbors,
                ↪ and disable it
73              output_dim1.disabled = true
74              output_dim2.count = 1
75              output_dim2.value = prev_dim1.feature
76          }
77          else
78          { // vertex is either disabled because it already sent its (relatively high) value, or it is
79          // tmpDisabled because of its relatively low value. Then, signal that its shouldn't be counted
80              output_dim2.count = 0
81              output_dim2.value = 0
82          }
83      }
84
85      receive_max(prev_dim1, neighbors_dim2_sum, output_dim1, output_dim2){
86          neighbors_avg_value = neighbors_dim2_sum.value / neighbors_dim2_sum.count
87          // we assume that at this point the vertices that sent a non−zero value, and '1' dim2.count, all
                ↪  share the same value − the maximum value among non−disabled vertices in the
                ↪ graph. Hence, their average is that maximum value.
88          if (neighbors_avg_value > 0)
89          // otherwise the vertex has no neighbors with the max value, since we assume non−zero initial
                ↪  values
90          {
91              output_dim1.neighbors_values_and_counts.add(neighbors_dim2_sum.count,
                    ↪ neighbors_avg_value)
92          }
93          if (!prev_dim1.disabled)
94          // vertex still hasn't got to be a max non−disabled value, hence it continues to try − until all
                ↪ higher values will be recorded.
95          {
```

```
96              output_dim1.tmp_disabled = false
97              output_dim2.count = 1
98              output_dim2.value = prev_dim1.initial_feature
99          }
100         else
101         {
102             output_dim2.count = 0
103             output_dim2.value = 0
104         }
105     }
```

Listing 6.2: Implement Message Length Computation

```
1
2       // Computes 3·s_initFeatLen·s_graphSize^4
3
4       Initialize:
5           [1] s_switchOn = 1
6           [2] s_switchTurnedOff = 0
7           [3] s_initFeatLen = 0
8           [4] s_graphSize = graphSize
9           [5] s_result = 0
10          [6] s_counter1 = 0
11          [7] s_counter2 = 0
12          [8] s_counter3 = 0
13          [9] s_counter4 = 0
14          [10] s_stateReset1 = 0
15          [11] s_stateReset2 = 0
16          [12] s_stateReset3 = 0
17          [13] s_stateAddTo1 = 0
18
19      RecurrentOp:
20          var prevSwitchVal = s_switchOn
21          var edgeCaseOne = Lsig(1 − (s_input1 − 1)) // s_input1 = 1
22          // we add 3 times becaues we want to multiply by 3
23          s_result.icrease_if(lsig(s_switchOn − s_stateReset1−s_stateReset2 −s_stateReset3 ), 1)
24          s_result.icrease_if(lsig(s_switchOn − s_stateReset1−s_stateReset2 −s_stateReset3 ), 1)
25          s_result.icrease_if(lsig(s_switchOn − s_stateReset1−s_stateReset2 −s_stateReset3 ), 1)
26
27          var goodOne = lsig(lsig(s_graphSize − s_counter1) − (1−s_stateAddTo1) − edgeCaseOne) //
                ↪ we want to add and we can
28          s_stateReset1 = lsig(s_stateReset1− lsig(1−s_counter1)) // maintain
29          s_stateReset1 = lsig(s_stateReset1+ lsig(s_stateAddTo1− lsig(s_graphSize−1−s_counter1))) //
                ↪ turn on: we want to add counter reached limit, reset counter
30          s_stateReset1 = lsig(s_stateReset1 −edgeCaseOne)
31          s_stateAddTo1 = \lsig(s_stateAddTo1 − s_stateReset1−edgeCaseOne);
32          s_counter1.decrease_if(s_stateReset1, 1) // resetting
33          s_counter1.increase_if(goodOne, 1)
34
35          var addTo2 = lsig(lsig(s_stateReset1 − lsig(s_counter1))) // reset1 finished
36          var goodTwo = lsig(lsig(s_graphSize − s_counter2) − (1−addTo2)) // we want to add and we
                ↪ can
37          s_stateReset2 = lsig(s_stateReset2 − lsig(1 − (s_counter2))) // maintain
38          s_stateReset2 = \lsig(s_stateReset2 + \lsig(addTo2 − \lsig(s_graphSize − 1 − s_counter2))); //
                ↪  turn on: we want to add but cannot
39          s_counter2.decrease_if(s_stateReset2, 1)
```

```
40          s_counter2.increase_if(goodTwo, 1)
41
42          var addTo3 = lsig(lsig(s_stateReset2 − lsig(s_counter2))) // reset2 finished
43          var goodThree = lsig(lsig(s_graphSize −1−s_counter3) − (1−addTo3)) // we want to add and
                ↪ we can
44          s_stateReset3 = lsig(s_stateReset3 − lsig(1 − (s_counter3))) // maintain
45          s_stateReset3 = lsig(s_stateReset3 + lsig(addTo3 − lsig(s_graphSize −1−s_counter3))) // turn
                ↪ on: we want
46          s_counter3.decrease_if(s_stateReset3, 1)
47          s_counter3.increase_if(goodThree, 1)
48
49          var addTo4 = lsig(lsig(s_stateReset3 − lsig(s_counter3))) // reset3 finished
50          var goodFour = lsig(lsig(s_graphSize −1−s_counter4) − (1−addTo4)) // we want to add and
                ↪ we can
51          s_stateReset4 = lsig(s_stateReset4 − lsig(1 − (s_counter4))) // maintain
52          s_stateReset4 = lsig(s_stateReset4 + lsig(addTo4 − lsig(s_graphSize −1−s_counter4))) // turn
                ↪ on: we want
53          s_counter4.decrease_if(s_stateReset3, 1)
54          s_counter4.increase_if(goodThree, 1)
55
56          var addTo5 = \lsig(\lsig(s_stateReset4 − \lsig(s_counter4))); // reset3 finished
57          var goodFive = lsig(lsig(s_initFeatLen −1−s_counter5) − (1−addTo5)) // we want to add and
                ↪ we can
58          s_counter5.increase_if(goodFive, 1)
59
60          s_switchOn = 1 − lsig(lsig(s_counter5 − s_initFeatLen +2)+addTo5 + lsig(1−goodFive) − 2)
61          s_stateAddTo1 = lsig(1−s_stateReset1 − s_stateReset2 − s_stateReset3− s_stateReset4 − (1−
                ↪ s_switchOn))
62
63          s_switchTurnedOff = relu(prevSwitchVal−s_switchOn)
```

Listing 6.3: Implement I2QU Translation + Concatenation To Other

```
1
2       Initialize:
3           [1] s_switchOn = 0
4           [2] s_switchTurnedOff = 0
5           [3] s_numberLeftToProcess = 0
6           [4] s_otherInRQ = 0
7           [5] s_result = 0
8           [6] s_stateAddToNumber = 0
9
10      RecurrentOp:
11          var prevSwitchVal = s_stateAddToNumber
12          s_stateAddToNumber = lsig(s_numberLeftToProcess + s_switchOn−1))
13          s_switchOn = s_stateAddToNumber
14          s_switchTurnedOff = relu(prevSwitchVal−s_switchOn)
15          var switchTurnedOn = relu(s_switchOn−prevSwitchVal)
16
17          // init procedure, when switch turns on
18          s_result.set_if(switchTurnedOn, s_otherInRQ)
19          s_result.div4_if(switchTurnedOn) // together with next line: insert a separating "0" i.e. 1/4 in
                ↪ RQ
20          s_result.increase_if(switchTurnedOn, 1/4.0)
21
22          // operation in add to number state
```

```
23          s_result.div4_if(s_stateAddToNumber)
24          s_result.increase_if(s_stateAddToNumber, 3/4.0)
25          s_numberLeftToProcess.decrease_if(s_stateAddToNumber, 1)
```

Listing 6.4: Implement B2Q Translation

```
1
2       /* The idea of the algorithm in general lines is as follows:
3        Initially, x = Σ_{i∈[m]} a_i/2^i where x =s_number_in_process, m =s_messageBitLength.
4        All relevant variables are reset to their starting values in the s_stateInit stage. Then,
5        For i=1..m
6            Assume we are left with x = Σ_{j∈[i..m]} a_j/2^j. The s_stateReduce stage implements:
7            x = relu(Σ_{j∈[i+1..m} 1/2^j) // at that point a_i = 1 ⇒ x ≥ 1/2^m and a_i = 0 ⇒ x ≤ 0
8            Then the s_stateShiftLeft stage implements:
9            x = lsig(2^m x) // at that point a_i = 1 ⇒ x = 1 and a_i = 0 ⇒ x = 0
10           Then: the s_addToNumber stage updates the result accordingly − adding 1/4^i or 3/4^i, and so is
                 ↪ the s_numberLeftToProcess.
11      */
12
13      Initialize:
14          [1] s_switch = 0
15          [2] s_switchTurnedOff = 0
16          [3] s_messageBitLength = 0
17          [4] s_numberLeftToProcess = 0
18          [5] s_numberInC4 = 0
19          [6] s_number_in_process = 0
20          [7] s_stateInit = 0
21          [8] s_stateReduce = 0
22          [9] s_stateShiftLeft = 0
23          [10] s_stateAddToNumber = 0
24          [11] s_maxDigitsToTheRight = 0
25          [12] s_digitsToTheLeft = 0
26          [13] s_digitsToTheRight = 0
27          [14] s_nextReduce = 0
28          [16] s_C41 = 0
29          [17] s_C43 = 0
30
31      RecurrentOp:
32          var switchWasOff = lsig(s_stateReduce + s_stateShiftLeft + s_stateAddToNumber +
                 ↪ s_stateInit)
33          var switchTurnedOn = relu(s_switch−switchWasOff)
34          s_stateInit = switchTurnedOn
35          var prevSwitchVal = s_switch
36
37          // determine state of current pass
38          s_stateReduce = lsig(lsig(s_stateReduce− lsig(1−s_digitsToTheRight))+\lsig(s_stateInit − \lsig(
                 ↪ s_messageBitLength − s_maxDigitsToTheRight)) − (1 − s_switch))
39          s_stateShiftLeft= lsig(1−s_stateReduce−s_stateInit−(1 − s_switch))
40          s_stateShiftLeft = lsig(s_stateShiftLeft − lsig(1−s_digitsToTheLeft)−(1−s_switch))
41          s_stateAddToNumber= lsig(lsig(s_maxDigitsToTheRight) − (s_stateReduce+ s_stateShiftLeft+
                 ↪ s_stateInit)−(1−s_switch))
42
43          s_stateInit = lsig(lsig(s_messageBitLength− s_maxDigitsToTheRight) − (1− s_stateInit)−(1−
                 ↪ s_switch))
44          // operation in init mode
45          var change= lsig(s_messageBitLength− s_digitsToTheRight)
```

```
46          s_digitsToTheRight.increase_if(s_stateInit, change)
47          s_maxDigitsToTheRight.increase_if(s_stateInit, 1)
48          s_nextReduce.set_if(s_stateInit, 1 / 4.0)
49          s_C41.set_if(s_stateInit, 1 / 4.0)
50          s_C43.set_if(s_stateInit, 3 / 4.0)
51          s_numberInC4.set_if(s_stateInit, 0)
52
53          // operation in the reduce state
54          s_digitsToTheRight.decrease_if(s_stateReduce, 1)
55          s_digitsToTheLeft.increase_if(s_stateReduce, 1)
56          var tmp = s_number_in_process
57          tmp.decrease_if(s_stateReduce, s_nextReduce)
58          s_number_in_process = lsig(tmp)
59          s_nextReduce.div2_if(s_stateReduce)
60
61          // operation in the shift left state
62          var tmp2 = s_number_in_process
63          tmp2.increase_if(s_stateShiftLeft, s_number_in_process)
64          s_number_in_process = lsig(tmp2)
65          s_digitsToTheLeft.decrease_if(s_stateShiftLeft, 1)
66          s_nextReduce.increase_if(s_stateShiftLeft, s_nextReduce)
67          s_digitsToTheRight.increase_if(s_stateShiftLeft, lsig(s_maxDigitsToTheRight-1-
                ↪ s_digitsToTheRight)
68
69          // operation in add to number state
70          // multiplying s_number_in_process by 2, this is sometimes required to make it ≥ 1 (all when
                ↪ the extracted digit is 1)
71          var tmp3 = s_number_in_process
72          tmp3.increase_if(s_stateAddToNumber, s_number_in_process)
73          s_number_in_process = lsig(tmp3)
74          s_numberInC4.increase_if(s_stateAddToNumber, lsig(s_C41 - s_number_in_process) + lsig(s_C43
                ↪  -(1-s_number_in_process)))
75          s_C41.div4_if(s_stateAddToNumber)
76          s_C43.div4_if(s_stateAddToNumber)
77          s_numberLeftToProcess.decrease_if(s_stateAddToNumber, lsig(0.5 - (1-s_number_in_process)))
78          s_numberLeftToProcess = lsig(s_numberLeftToProcess)
79          s_numberLeftToProcess.increase_if(s_stateAddToNumber, s_numberLeftToProcess)
80          s_number_in_process.decrease_if(s_stateAddToNumber, s_number_in_process)
81          s_number_in_process.increase_if(s_stateAddToNumber, s_numberLeftToProcess)
82
83          s_maxDigitsToTheRight.decrease_if(s_stateAddToNumber, 1)
84          s_stateReduce.increase_if(s_stateAddToNumber, s_stateReduce)
85
86          s_switch = lsig(s_stateReduce + s_stateShiftLeft + s_stateAddToNumber + s_stateInit)
87          s_switchTurnedOff = lsig(prevSwitchVal-s_switch)
```

Listing 6.5: Implement Q2B Translation

```
1
2       Initialize:
3           [1] s_switchOn = 0
4           [2] s_switchTurnedOff = 0
5           [3] s_numberLeftToProcess = 0
6           [4] s_numberInBinary = 0
7           [5] s_stateAddToNumber = 0
8           [6] s_nextPosBinaryValue = 0
```

```

    RecurrentOp:
        var prevSwitchVal = s_stateAddToNumber
        s_stateAddToNumber = \lsig(lsig(8·s_numberLeftToProcess−1) − relu(1 − $s_switchOn$))
        s_switchOn = s_stateAddToNumber
        s_switchTurnedOff = \relu(prevSwitchVal−s_switchOn)
        var switchTurnedOn = \relu(s_switchOn−prevSwitchVal)

        \\ init procedure, when switch turns on
        s_nextPosBinaryValue.set_if(switchTurnedOn, 0.5)
        s_numberInBinary.set_if(switchTurnedOn, 0)

        \\ number translation procedure
        var extractedDigit = lsig(4· s_numberLeftToProcess−2) // first digit is in {1,3} and we
            ↪ translate to {0,1}
        s_numberInBinary.increase_if(s_stateAddToNumber, \lsig(s_nextPosBinaryValue − (1 −
            ↪ extracted)))
        s_numberLeftToProcess.decrease_if(s_stateAddToNumber, (1 + 2· extracted)/4.0)
        // next 2 lines essentially multiply by 4
        s_numberLeftToProcess = ChangeIfState(s_stateAddToNumber, s_numberLeftToProcess) //
            ↪ multiply by 2
        s_numberLeftToProcess = ChangeIfState(s_stateAddToNumber, s_numberLeftToProcess) //
            ↪ multiply by 2
        s_nextPosBinaryValue = div2_if_state(s_stateAddToNumber)
```

Listing 6.6: Implement Translation Of Final Result

```

    Initialize:
        [1] s_switchOn = 0
        [2] s_switchTurnedOff = 0
        [3] s_allFinished = 0
        [4] s_numberLeftToProcess = 0
        [5] s_allFinishedZeroOne = 0
        [6] s_numberInBinary = 0
        [7] s_stateAddToNumber = 0
        [8] s_nextPosBinaryValue = 0
        [9] s_lockResult = 0 // once we have the final result (in dimensions 5,6) we want it to stay no
            ↪ matter what happens in the network.

    RecurrentOp:
        var prevSwitchVal = s_stateAddToNumber
        var allFinishedZeroOne = relu(4· s_allFinished−2) // from RQ to {0,1}
        var switchOnAndAllFinished = s_switchOn+allFinishedZeroOne−1
        s_stateAddToNumber = relu(lsig(lsig(8·s_numberLeftToProcess
            ↪ −1) − (1 − $switchOnAndAllFinished$)) − $s_lockResult$)
        s_switchTurnedOff = relu(prevSwitchVal−s_switchOn +
                                    relu((1−prevSwitchVal) + (s_switchOn−
                                        ↪ switchOnAndAllFinished) − 1))
        s_allFinishedZeroOne.set_if(lsig(relu(s_switchTurnedOff + allFinishedZeroOne−1)+s_lockResult
            ↪ ), 1)
        s_lockResult = \relu(s_lockResult + s_allFinishedZeroOne)
        s_switchOn = s_stateAddToNumber
        var shouldInit = relu(s_switchOn−prevSwitchVal)

        \\ init procedure, when switch turns on
```

```
26          s_nextPosBinaryValue.set_if(shouldInit, 0.5)
27          s_numberInBinary.set_if(shouldInit, 0)
28
29          \\ number translation procedure
30          var extractedDigit = lsig(4· s_numberLeftToProcess−2) // first digit is in {1,3} and we
                ↪ translate to {0,1}
31          s_numberInBinary.increase_if(s_stateAddToNumber, \lsig(s_nextPosBinaryValue − (1 −
                ↪ extracted)))
32          s_numberLeftToProcess.decrease_if(s_stateAddToNumber, (1 + 2· extracted)/4.0)
33          // next 2 lines essentially multiply by 4
34          s_numberLeftToProcess = ChangeIfState(s_stateAddToNumber, s_numberLeftToProcess) //
                ↪ multiply by 2
35          s_numberLeftToProcess = ChangeIfState(s_stateAddToNumber, s_numberLeftToProcess) //
                ↪ multiply by 2
36          s_nextPosBinaryValue = div2_if_state(s_stateAddToNumber)
```

Listing 6.7: Implement Synchronizer (F_S)

```
1
2       Initialize:
3           [1] s_stateReadyForNextAlgoIteration = 0 // signals the initiation of a new computation −
                ↪ starting from B2Q
4           [2] s_foundNotFinished = 0
5           [3] s_syncCountdown = 0
6           [4] s_stateSyncInProgress = 0
7           [5] s_stateCountdownOver = 1
8           [6] s_stateResetCountdown = 0
9           [7] s_syncNumOfCycles = 0 // external input, should be constant
10          [8] s_otherNodesNotFinished = 0 // external input
11          [9] s_curNodeFinished = 0 // external input
12
13      RecurrentOp:
14          s_stateReadyForNextAlgoIteration = 0; // updated during the pass
15          var cannotStart = relu(1−s_syncNumOfCycles) // as long as s_syncNumOfCycles is not
                ↪ received, cannot start
16          var startedResetAndNotFinished = = lsig( s_stateResetCountdown + lsig( s_syncRoundsCount
                ↪ −s_syncCountdown) − 1)
17          s_stateResetCountdown = relu(startedResetAndNotFinished−cannotStart) // do nothing until
                ↪ can start
18
19          var countdownGEzeroNotInReset = lsig(lsig( s_syncCountdown)−s_stateResetCountdown)
20          s_stateSyncInProgress = relu(countdownGEzeroNotInReset−cannotStart) // do nothing until
                ↪ can start
21          s_stateCountdownOver = relu ( (1 − lsig(s_syncCountdown))−cannotStart) ) // ...
22
23          s_foundNotFinished = relu ( lsig(s_foundNotFinished +s_otherNodesNotFinished +(1−
                ↪ s_curNodeFinished)−s_stateResetCountdown)−cannotStart ) // either already had not
                ↪ −finished indication in this cycle, or received indication of a non−finished node, or
                ↪ current node is not finished
24          s_syncCountdown.decrease_if(s_stateSyncInProgress, 1) // if during sync, countdown
25
26          s_stateReadyForNextAlgoIteration = lsig(s_stateCountdownOver+(1−s_foundNotFinished) − 1)
                ↪  // will be 0 while cannotStart
27          s_stateResetCountdown = s_stateReadyForNextAlgoIteration) // reached 0 and all finished,
                ↪ should start reset
```

```
28      s_foundNotFinished = lsig(s_foundNotFinished−s_stateResetCountdown) // reset also resets
            ↪ foundNotFinished
29      s_syncCountdown.increase_if(s_stateResetCountdown, 1) // in reset mode continue to increase
            ↪ countdown
```

6.5 Further Results

Our main theorem, Theorem 6.4.3, has a number of interesting variants and implications. First, it has a version for *graph embeddings* $F : \mathcal{G}_{\mathcal{B}} \to \mathcal{B}$. A graph-embedding R-GNN $N = (R, M)$ comprises an R-GNN R and an MLP M. It applies R, averages the nodes' values, and applies M. Formally, denote by I_v the last iteration of R on a node v, then

$$\forall G \in \mathcal{G}_{\mathcal{B}}\ N(G) \coloneqq \mathfrak{rb}^{-1}\Big(M\big(\tfrac{1}{|G|}\Sigma_{v\in V(G)}R^{(I_v)}(G,v)\big)\Big)$$

We say that two graphs are *mp-indistinguishable*, or indistinguishable by Color Refinement, if the same message-passing colors appear with the same multiplicities, that is,

$$\forall t \geq 0 \quad \{\!\{\mathsf{mpc}_G^{(t)}(v) \mid v \in V(G)\}\!\} = \{\!\{\mathsf{mpc}_H^{(t)}(v) \mid v \in V(H)\}\!\}$$

Note that this implies $|G| = |H|$. A function $F : \mathcal{G}_{\mathcal{B}} \to \mathcal{B}$ is *mp-invariant* if for all mp-indistinguishable graphs G, H we have $F(G) = F(H)$. Clearly, all graph embeddings computable by R-GNNs are mp-invariant. As for the converse, this turns out to be false for disconnected graphs, see [RG25, Appendix C] for a complete proof.

Theorem 6.5.1 ([RG25])**.** *There exists an mp-invariant graph embedding $F : \mathcal{G}_{\mathcal{B}} \to \mathcal{B}$ such that for every R-GNN N there exists a disconnected graph G for which $N(G) \neq F(G)$.*

However, we show that for connected graphs, all mp-invariant graph-level functions are computable by R-GNNs.

Theorem 6.5.2. *Let $\mathcal{CG}_{\mathcal{B}} \subset \mathcal{G}_{\mathcal{B}}$ be the subset of connected graphs in $\mathcal{G}_{\mathcal{B}}$, and let $F : \mathcal{CG}_{\mathcal{B}} \to \mathcal{B}$ be computable. Then F is mp-invariant if and only if there is an R-GNN N such that*

$$\forall G \in \mathcal{CG}_{\mathcal{B}}\ N(G) = F(G)$$

Furthermore, if F is computable in time $T(n)$ and space $S(n)$, then N uses time $O(T(n))+\mathrm{poly}(n)$ and space $O(S(n)) + \mathrm{poly}(n)$.

Proof. Define a feature transformation $F_{\mathrm{T}} : \mathcal{CG}_{\mathcal{B}} \to \mathcal{Z}_{\mathcal{B}}$, $F_{\mathrm{T}}(G)(v) \coloneqq F(G)$. The connectivity of the input, together with [RG25, Lemma A.11], mean that the (graph embedding) mp-invariance of F implies (feature transformation) mp-invariance of F_{T}. By the latter and Theorem 6.4.3 we have that there is an R-GNN N' such that $\forall G \in \mathcal{CG}_{\mathcal{B}}\ \forall v \in V(G)\ N'(G, v) = F_{\mathrm{T}}(G)(v)$. Let $N = (N', x \mapsto x)$ be a graph-embedding R-GNN where the final MLP computes the identity function, then we have that

$$\forall G \in \mathcal{CG}_{\mathcal{B}}\ N(G) =$$

$$\mathfrak{rb}^{-1}\Big(\frac{1}{|G|}\Sigma_{v\in V(G)}\mathfrak{rb}(N'(G,v))\Big) =$$

$$\mathfrak{rb}^{-1}\Big(\mathfrak{rb}\big(F_{\mathrm{T}}(G)(v)\big)\Big) = F_{\mathrm{T}}(G)(v) = F(G)$$

□

So far, R-GNNs can only compute mp-invariant functions. For connected graphs, we can break the invariance by introducing *random node initialization* (RNI) [Abb+21a], that is, augmenting the initial feature of each node by arandom number $r \sim U(0,1)$. GNNs with RNI describe a randomized algorithm, yet this randomized algorithm, or the random variable it computes, still satisfies the usual equivariance condition i.e. its output probability-distribution is identical for every two equivariant vertices (see [Abb+21a]). We say that an R-GNN N with RNI computes a function $F : \mathcal{G}_{\mathcal{GB}} \to \mathcal{Z}_{\mathcal{B}}$ on a graph $G \in \mathcal{G}_{\mathcal{GB}}$ and node $v \in V(G)$ if and only if: $\Pr(N(G,v) = F(G,v)) \geq p$ for some $\frac{1}{2} < p$, and $N(G,v) \neq F(G,v) \Leftrightarrow N(G,v) = \text{null-value}$. By repeatedly running N we can boost $\Pr(N(G,v) = F(G,v))$ arbitrarily close to 1.

Corollary 6.5.3. *Let $\mathcal{C}\mathcal{G}_{\mathcal{GB}} \subset \mathcal{G}_{\mathcal{GB}}$ be the subset of connected graphs in $\mathcal{G}_{\mathcal{GB}}$, and let $F : \mathcal{C}\mathcal{G}_{\mathcal{GB}} \to \mathcal{Z}_{\mathcal{B}}$ be computable in time $T(n)$ and space $S(n)$. Then, there exists an R-GNN N with RNI, such that F is computable by N. Furthermore, N uses time $O(T(n)) + \text{poly}(n)$, space $O(S(n)) + \text{poly}(n)$, and $O(n \log n)$ random bits.*

Proof. We can view the computation of an R-GNN with RNI as a two-stage process: Given a graph $G \in \mathcal{G}_{\mathcal{GB}}, G = \{V(G), E(G), \mathcal{B}, Z(G)\}$, we first extend the initial feature of every node by a random number. This gives us a graph which we denote by $\widetilde{G}$, $\widetilde{G} = \{V(G), E(G), \mathcal{B}, \widetilde{Z}(G)\}$, with the same structure as G but extended features. Then, we run a deterministic R-GNN on $\widetilde{G}$. As the features of an R-GNN are rational numbers - finite precision, we restrict the length of the binary representation of the random numbers to $3 \log n$, that is, we choose a random bitstring of length $3 \log n$ for each node. With probability greater than $2/3$, each node will get a different random number assigned to it, in which case $\widetilde{G}$ is considered *individualized*.

Let $G \in \mathcal{C}\mathcal{G}_{\mathcal{GB}}$ be a connected graph, and let $v \in V(G)$. For $\widetilde{G}, v$ we define

$$\widetilde{F}(\widetilde{G})(v) \coloneqq \begin{cases} F(G)(v) & \widetilde{G} \text{ is individualized} \\ \text{null-value} & \text{otherwise} \end{cases}$$

Note that for any two augmented graphs $\widetilde{G}, \widetilde{H}$ that are connected and individualized, and two vertices $v \in V(\widetilde{G}), v' \in V(\widetilde{H})$, it holds that $\mathsf{mpc}_{\widetilde{G}}(v) = \mathsf{mpc}_{\widetilde{H}}(v') \Rightarrow$ there is an isomorphism $f : V(\widetilde{G}) \to V(\widetilde{H})$ such that $f(v) = f(v')$. Hence, for such $\widetilde{G}, v, \widetilde{H}, v'$ it holds that $\mathsf{mpc}_{\widetilde{G}}(v) = \mathsf{mpc}_{\widetilde{H}}(v') \Rightarrow \widetilde{F}(\widetilde{G})(v) = \widetilde{F}(\widetilde{H})(v')$. Hence, $\widetilde{F}$ is mp-invariant. Hence, by Theorem 6.4.3 there is an R-GNN N' that computes $\widetilde{F}$. Hence, let N be an RNI R-GNN that uses N' as its deterministic R-GNN, then N computes F. □

Another variant of our main result concerns a common extension of GNNs which is the addition of *global sum-aggregation* (a.k.a. *global sum*; *virtual nodes*) i.e. a sum-aggregation of the features of all nodes, as a third input to the combine MLP [Gil+17a; Bar+20b]. Adapting Definition 6.4.1, let $N = (A, F)$ be an R-GNN with global sum, then F has I/O dimensions $3d; d$, and

$$N^{(i+1)}(G,v) \coloneqq F\big(N^{(i)}(G,v),\ \text{sum}_{w \in N_G(v)} N^{(i)}(G,w),\ \text{sum}_{w \in V(G)} N^{(i)}(G,w)\big)$$

Note that here we do not need the size of the graph as an input, since it can easily be computed using global sum. Instead of mp-invariance, R-GNNs with global sum satisfy an

invariance that we call *WL-invariance*, which is finer i.e. mp-invariance ⇒ WL-invariance. It is based on a variant of the Color Refinement algorithm, the 1-dimensional Weisfeiler-Leman algorithm, that captures the additional global information obtained through global sum, see [RG25, appendix A] and [Gro21] for further details.

Theorem 6.5.4. *Let $F : \mathcal{G}_{\mathcal{B}} \to \mathcal{Z}_{\mathcal{B}}$ be a computable feature transformation. Then F is WL-invariant if and only if there is an R-GNN N with global sum such that*

$$\forall G \in \mathcal{G}_{\mathcal{B}}\ \forall v \in V(G)\ N(G, v) = F(G)(v)$$

Furthermore, if F is computable in time $T(n)$ and space $S(n)$, then N uses time $O(T(n))+$ $\mathrm{poly}(n)$ and space $O(S(n)) + \mathrm{poly}(n)$.

Proof. The proof imitates the proof of Theorem 6.4.3, reducing WL-invariant functions to R-GNNs with global aggregation, using adapted versions of the intermediate models MPC-GA; MP-LGA; and S-MP-GA, which we will denote by MPC-GA^w;MP-LGA^w;and S-MP-GA^w:

- MPC-GA^w differs from MPC-GA in that that it receives the dag representing $\mathsf{wl}_G(v)$ instead of $\mathsf{mpc}_G(v)$ (see [RG25, Appendix A, The Weisfeiler Leman Algorithm]).
- MP-LGA^w differs from MP-LGA in that in each iteration it has 3 inputs rather than 2 - the third input being the multiset of values of $V(G) \setminus N_G(v)$.
- S-MP-GA^w differs from S-MP-GA in that in each iteration it has 5 inputs rather than 4 - the additional input being the sum of values of $V(G)$.

The reduction from MPC-GA^w to MP-LGA^w is a straightforward adaptation of the reduction from MPC-GA to MP-LGA. Reducing MP-LGA^w to S-MP-GA^w is similar to reducing MP-LGA to S-MP-GA: It is not difficult to see how to modify the 'receive_max' procedure (see Listing 6.1) to use a new input 'global_sum' and, in addition to the current update of the multiset of neighbors values, update a multiset of the values of $V(G)\setminus N_G(v)$, thus collecting the required information for emulating an MP-LGA^w iteration. Finally, reducing S-MP-GA^w to R-GNN with global sum-aggregation can be achieved by replicating the processing of the neighbors' sum:

- Having a sub-network F_{B2Q_s}, similar to F_{B2Q}, to translate the received global sum from RB to RQ.
- Having dimensions in F_M to accommodate the global sum input, thus being able to emulate a Turing machine that has the same inputs as an S-MP-GA^w.
- Having connections between F_{B2Q_s} and other sub-networks, similar to those of F_{B2Q}.

□

Theorem 6.5.4 also has a version for graph-level functions, analogues to Theorem 6.5.2. Unlike in Theorem 6.5.2, the restriction to connected graphs is not necessary here as global sum provides access to the values of all vertices in the graph - regardless of their connected component.

Corollary 6.5.5. *Let $F : \mathcal{G}_{\mathcal{B}} \to \mathcal{B}$ be computable. Then F is WL-invariant if and only if there is an R-GNN N such that*

$$\forall G \in \mathcal{G}_{\mathcal{B}}\ N(G) = F(G)$$

Furthermore, if F is computable in time $T(n)$ and space $S(n)$, then N uses time $O(T(n))+\mathrm{poly}(n)$ and space $O(S(n)) + \mathrm{poly}(n)$.

6.6 Summary

We prove that recurrent graph neural networks are universal for message-passing algorithms and message-passing-invariant functions, with only a polynomial time and space overhead. Note that our theorem is not an approximation theorem: Our target functions are computable, and our constructed GNNs emulate the Turing machines that compute them, hence the constructed GNNs compute the target functions exactly.

Furthermore, by adding randomization and focusing on connected graphs, we can overcome the limitation to mp-invariant functions and compute any isomorphism-invariant function.

Our complexity analysis for the reduction states "polynomial overhead", and it is not difficult to extract an upper bound of $O(n^{10}k^2)$ from our proofs. It will be useful to know whether our reduction can be improved so to have a lower complexity, and it will be useful to have a lower bound for the complexity of any reduction from computable mp-invariant functions to R-GNNs. Ideally, one would construct a reduction that matches a proved lower bound. These questions remain open.

Concluding Remarks

We have uncovered important parts of the expressivity picture of Graph Neural Networks.

For non-recurrent GNNs, we have proved comparative results concerning the prime hyper-parameters of a GNN architecture: We have shown how the most common combinations for message; aggregation; and global computation algorithm, compare in their expressivity. This was done considering the effect of one hyper-parameter at a time. We have seen that almost every single change to a hyper-parameter changes the expressivity of GNNs, and through our separation proofs we have demonstrated the nature of the strengths and weaknesses of the various choices.

Several of our separation results are complemented with experiments. These show that the expressivity properties of an architecture have a meaningful effect in practice: Trained models of uniformly expressive architectures generalize in graph-size significantly better than models of non-expressive ones. Moreover, the latter always show large errors on graphs even slightly larger than those used for training.

Through our study of non-recurrent GNNs it becomes evident that the non-recurrent GNN scheme is fundamentally limited in its expressivity, far beyond the inherited limitations of a message-passing algorithm with a fixed information-radius. It is then that we turn to study recurrent GNNs, where a sequence of message-passing layers is repeated until certain convergence conditions are met with regards to certain output dimensions. For this scheme, which is similar to common and successful architectures in practice, we prove a profound result: Computable recurrent GNNs can emulate any message passing algorithm and, equivalently, compute any function that is invariant to the color of a node according to the Color Refinement algorithm. In particular, we prove that the emulation can be achieved with finite-precision parameters; relu-activation; sum-aggregation; identity-msg; 1-layer, GNN. Our proof is constructive, and the construction incurs only a polynomial time and space overhead.

All of our positive results are proven by constructing GNNs that express whatever is required. While the constructions are asymptotically efficient with respect to the graph size, either maintaining the same complexity or - in the case of the recurrent GNNs result - incurring a polynomial overhead, we do not prove any non-trivial lower bounds for achieving the required expressivity. Such lower bounds would have meaningful practical implications, clarifying further how to interpret our results when considering also the architecture's size. Hence, finding these bounds is an interesting open problem.

Bibliography

[Gol62] Michael Golomb. *Lectures on theory of approximation*. Argonne National Laboratory, Applied Mathematics Division, 1962.

[Mor65] H.L. Morgan. "The generation of a unique machine description for chemical structures-a technique developed at chemical abstracts service." In: *Journal of Chemical Documentation* 5.2 (1965), pp. 107–113.

[WL68] B.Y. Weisfeiler and A.A. Leman. "The reduction of a graph to canonical form and the algebra which appears therein". In: *NTI, Series 2* (1968). English translation by G. Ryabov available at `https://www.iti.zcu.cz/wl2018/pdf/wl_paper_translation.pdf`.

[HSW89] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. "Multilayer feedforward networks are universal approximators". In: *Neural networks* 2.5 (1989), pp. 359–366.

[SS92] Hava T Siegelmann and Eduardo D Sontag. "On the computational power of neural nets". In: *Proceedings of the fifth annual workshop on Computational learning theory*. 1992, pp. 440–449.

[Ott97] Martin Otto. *Bounded variable logics and counting – A study in finite models*. Vol. 9. Lecture Notes in Logic. Springer Verlag, 1997.

[GR01] C. Godsil and G. Royle. *Algebraic Graph Theory*. Springer, 2001.

[May06] Robert Mayans. *The Polynomial of Best Approximation*. `https://www.maa.org/sites/default/files/images/upload_library/4/vol6/Mayans/Best.html`. Accessed: 2023-06-03. 2006.

[Sca+08] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. "The graph neural network model". In: *IEEE transactions on neural networks* 20.1 (2008), pp. 61–80.

[GM10] C. Gallicchio and A. Micheli. "Graph echo state networks". In: *Proceedings of the IEEE International Joint Conference on Neural Networks*. 2010.

[IS15] Sergey Ioffe and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift". In: *International conference on machine learning*. pmlr. 2015, pp. 448–456.

[BKH16] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. "Layer normalization". In: *arXiv preprint arXiv:1607.06450* (2016).

[Li+16] Yujia Li, Richard Zemel, Marc Brockschmidt, and Daniel Tarlow. "Gated Graph Sequence Neural Networks". In: *Proceedings of ICLR'16*. 2016.

[Sel+16] Daniel Selsam, Matthew Lamm, B Benedikt, Percy Liang, Leonardo de Moura, David L Dill, et al. "Learning a SAT Solver from Single-Bit Supervision". In: *International Conference on Learning Representations*. 2016.

[Gil+17a] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. "Neural message passing for quantum chemistry". In: *International conference on machine learning*. PMLR. 2017, pp. 1263–1272.

[Gil+17b] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. "Neural Message Passing for Quantum Chemistry". In: *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*. Ed. by Doina Precup and Yee Whye Teh. Vol. 70. Proceedings of Machine Learning Research. PMLR, 2017, pp. 1263–1272. URL: http://proceedings.mlr.press/v70/gilmer17a.html.

[KW17a] T. N. Kipf and M. Welling. "Semi-supervised classification with graph convolutional networks". In: *Proceedings of the 5th International Conference on Learning Representations*. 2017.

[KW17b] Thomas N. Kipf and Max Welling. "Semi-Supervised Classification with Graph Convolutional Networks". In: *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL: https://openreview.net/forum?id=SJU4ayYgl.

[KW17c] Thomas N. Kipf and Max Welling. "Semi-Supervised Classification with Graph Convolutional Networks". In: *International Conference on Learning Representations (ICLR)*. 2017.

[Vas+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. "Attention is all you need". In: *Advances in neural information processing systems* 30 (2017).

[Zah+17] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. "Deep sets". In: *Advances in neural information processing systems* 30 (2017).

[BL18] Xavier Bresson and Thomas Laurent. *Residual Gated Graph ConvNets*. 2018. URL: https://openreview.net/forum?id=HyXBcYg0b.

[Che+19] Zhengdao Chen, Soledad Villar, Lei Chen, and Joan Bruna. "On the equivalence between graph isomorphism testing and function approximation with gnns". In: *Advances in neural information processing systems* 32 (2019).

[Mor+19a] Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. "Weisfeiler and leman go neural: Higher-order graph neural networks". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 33. 01. 2019, pp. 4602–4609.

[Mor+19b] Christopher Morris, Martin Ritzert, Matthias Fey, William L. Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. "Weisfeiler and Leman Go Neural: Higher-Order Graph Neural Networks". In: *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*. AAAI Press, 2019, pp. 4602–4609. DOI: 10.1609/AAAI.V33I01.33014602. URL: https://doi.org/10.1609/aaai.v33i01.33014602.

[Mor+19c] Christopher Morris, Martin Ritzert, Matthias Fey, William L. Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. "Weisfeiler and Leman Go Neural: Higher-Order Graph Neural Networks". In: *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence (AAAI)*. 2019, pp. 4602–4609.

[SQ19] Vighnesh Shiv and Chris Quirk. "Novel positional encodings to enable tree-based transformers". In: *Advances in neural information processing systems* 32 (2019).

[Xu+19a] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. "How Powerful are Graph Neural Networks?" In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL: https://openreview.net/forum?id=ryGs6iA5Km.

[Xu+19b] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. "How Powerful are Graph Neural Networks?" In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL: https://openreview.net/forum?id=ryGs6iA5Km.

[Bar+20a] Pablo Barceló, Egor V Kostylev, Mikaël Monet, Jorge Pérez, Juan L Reutter, and Juan-Pablo Silva. "The expressive power of graph neural networks as a query language". In: *ACM SIGMOD Record* 49.2 (2020), pp. 6–17.

[Bar+20b] Pablo Barceló, Egor V. Kostylev, Mikaël Monet, Jorge Pérez, Juan L. Reutter, and Juan Pablo Silva. "The Logical Expressiveness of Graph Neural Networks". In: *8th International Conference on Learning Representations (ICLR 2020)*. OpenReview.net, 2020. URL: https://openreview.net/forum?id=r1lZ7AEKvB.

[DB20] Vijay Prakash Dwivedi and Xavier Bresson. "A generalization of transformer networks to graphs". In: *arXiv preprint arXiv:2012.09699* (2020).

[Yun+20] Chulhee Yun, Srinadh Bhojanapalli, Ankit Singh Rawat, Sashank Reddi, and Sanjiv Kumar. "Are Transformers universal approximators of sequence-to-sequence functions?" In: *International Conference on Learning Representations*. 2020. URL: https://openreview.net/forum?id=ByxRM0Ntvr.

[Abb+21a] Ralph Abboud, Ismail Ilkan Ceylan, Martin Grohe, and Thomas Lukasiewicz. "The Surprising Power of Graph Neural Networks with Random Node Initialization". In: *Proceedings of the 30th International Joint Conference on Artificial Intelligence*. Ed. by Zhi-Hua Zhou. 2021, pp. 2112–2118. DOI: 10.24963/ijcai.2021/291.

[Abb+21b] Ralph Abboud, İsmail İlkan Ceylan, Martin Grohe, and Thomas Lukasiewicz. "The Surprising Power of Graph Neural Networks with Random Node Initialization". In: *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021, Virtual Event / Montreal, Canada, 19-27 August 2021*. Ed. by Zhi-Hua Zhou. ijcai.org, 2021, pp. 2112–2118. DOI: 10.24963/ijcai.2021/291. URL: https://doi.org/10.24963/ijcai.2021/291.

[Bar+21] Pablo Barceló, Floris Geerts, Juan Reutter, and Maksimilian Ryschkov. "Graph neural networks with local graph parameters". In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 25280–25293.

[Gro21] Martin Grohe. "The Logic of Graph Neural Networks". In: *36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy, June 29 - July 2, 2021*. IEEE, 2021, pp. 1–17. DOI: 10.1109/LICS52264.2021.9470677. URL: https://doi.org/10.1109/LICS52264.2021.9470677.

[Kre+21] Devin Kreuzer, Dominique Beaini, Will Hamilton, Vincent Létourneau, and Prudencio Tossou. "Rethinking graph transformers with spectral attention". In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 21618–21629.

[SYK21] Ryoma Sato, Makoto Yamada, and Hisashi Kashima. "Random features strengthen graph neural networks". In: *Proceedings of the 2021 SIAM International Conference on Data Mining (SDM)*. SIAM. 2021, pp. 333–341.

[Ram+22] Ladislav Rampášek, Michael Galkin, Vijay Prakash Dwivedi, Anh Tuan Luu, Guy Wolf, and Dominique Beaini. "Recipe for a general, powerful, scalable graph transformer". In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 14501–14515.

[Tai+22] Shyam A. Tailor, Felix L. Opolka, Pietro Liò, and Nicholas Donald Lane. "Do We Need Anisotropic Graph Neural Networks?" In: *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022. URL: https://openreview.net/forum?id=hl9ePdH04%5C_s.

[Gro23] Martin Grohe. "The Descriptive Complexity of Graph Neural Networks". In: *Proceedings of the 38th Annual ACM/IEEE Symposium on Logic in Computer Science*. 2023. DOI: 10.1109/LICS56636.2023.10175735.

[Mül+23] Luis Müller, Mikhail Galkin, Christopher Morris, and Ladislav Rampášek. "Attending to Graph Transformers". In: *arXiv e-prints*, arXiv:2302.04181 (Feb. 2023), arXiv:2302.04181. DOI: 10.48550/arXiv.2302.04181. arXiv: 2302.04181 [cs.LG].

[NHK23] Nhat Khang Ngo, Truong Son Hy, and Risi Kondor. "Multiresolution graph transformers and wavelet positional encoding for learning long-range and hierarchical structures". In: *The Journal of Chemical Physics* 159.3 (2023).

[RTG23] Eran Rosenbluth, Jan Toenshoff, and Martin Grohe. *Some Might Say All You Need Is Sum*. 2023. arXiv: 2302.11603 [cs.LG]. URL: https://arxiv.org/abs/2302.11603.

[Tön+23] Jan Tönshoff, Berke Kisin, Jakob Lindner, and Martin Grohe. "One model, any CSP: graph neural networks as fast global search heuristics for constraint satisfaction". In: *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence*. 2023, pp. 4280–4288.

[BKR24] César Bravo, Alexander Kozachinskiy, and Cristobal Rojas. "On dimensionality of feature vectors in MPNNs". In: *Forty-first International Conference on Machine Learning*. 2024. URL: https://openreview.net/forum?id=UjDp4Wkq2V.

[GR24] Martin Grohe and Eran Rosenbluth. "Are Targeted Messages More Effective?" In: *ArXiv* arXiv:2403.06817 (2024). DOI: 10.48550/arXiv.2403.06817.

[PCK24] Maximilian Pfluger, David Tena Cucala, and Egor V Kostylev. "Recurrent Graph Neural Networks and Their Connections to Bisimulation and Logic". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 38. 13. 2024, pp. 14608–14616.

[Ros+24] Eran Rosenbluth, Jan Tönshoff, Martin Ritzert, Berke Kisin, and Martin Grohe. *Distinguished In Uniform: Self Attention Vs. Virtual Nodes*. 2024. arXiv: 2405.11951 [cs.LG]. URL: https://arxiv.org/abs/2405.11951.

[RG25] Eran Rosenbluth and Martin Grohe. "Repetition Makes Perfect: Recurrent Sum-GNNs Match Message Passing Limit". In: *arXiv preprint arxiv:2505.00291* (2025).

[Ahv+] Veeti Ahvonen, Damian Heiman, Antti Kuusisto, and Carsten Lutz. "Logical characterizations of recurrent graph neural networks with reals and floats". In: *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.