Let's Agree to Degree: Comparing Graph Convolutional Networks in the Message-Passing Framework

Floris Geerts [1] **Filip Mazowiecki** [2] **Guillermo A. Pérez** [1 3]

Abstract

We cast neural networks defined on graphs as message-passing neural networks (MPNNs) to study the distinguishing power of different classes of such models. We are interested in when certain architectures are able to tell vertices apart based on the feature labels given as input with the graph. We consider two variants of MPNNS: anonymous MPNNs whose message functions depend only on the labels of vertices involved; and degree-aware MPNNs whose message functions can additionally use information regarding the degree of vertices. The former class covers popular graph neural network (GNN) formalisms for which the distinguished power is known. The latter covers graph convolutional networks (GCNs), introduced by Kipf and Welling, for which the distinguishing power was unknown. We obtain lower and upper bounds on the distinguishing power of (anonymous and degree-aware) MPNNs in terms of the distinguishing power of the Weisfeiler-Lehman (WL) algorithm. Our main results imply that (i) the distinguishing power of GCNs is bounded by the WL algorithm, but they may be one step ahead; (ii) the WL algorithm cannot be simulated by "plain vanilla" GCNs but the addition of a trade-off parameter between features of the vertex and those of its neighbours (as proposed by Kipf and Welling) resolves this problem.

1. Introduction

A standard approach to learning tasks on graph-structured data, such as vertex classification, edge prediction, and graph classification, consists of the construction of a representation in some metric space of vertices and graphs that captures their structural information. Graph Neural Networks (GNNs) are currently considered the state-of-the art approach for learning such representations. Many variants of GNNs exist but they all follow a similar strategy. Each vertex is initially associated with a feature vector. This is followed by an iterative neighbourhood-aggregation scheme where each vertex aggregates feature vectors of its neighbours, possibly combines this with its own current feature vector, to finally obtain its new feature vector. After a number of iterations, each vertex is then represented by the resulting feature vector. We refer to Zhou et al. (2018) and Wu et al. (2019b) for extensive surveys on GNNs.

The adequacy of GNNs for graph learning tasks is directly related to their distinguishing power which refers to the ability of GNNs to distinguish vertices and graphs in terms of the computed representation. That is, when two vertices are represented by the same feature vector, they are considered the same with regards to any subsequent feature-based task.

Only recently a study of the distinguishing power of GNN variants has been initiated. In Xu et al. (2019) and Morris et al. (2019) the distinguishing power of GNNs is linked to that of the classical Weisfeiler-Lehman (WL) algorithm. The WL algorithm starts from an initial vertex colouring of the graph. Then, similarly as GNNs, the WL algorithm recursively aggregates the colouring of neighbouring vertices. In each iterative step, a vertex colouring is obtained that refines the previous one. The WL algorithm stops when no further refinement is obtained. The distinguishing power of the WL algorithm is well understood (Arvind et al., 2017).

More precisely, in Xu et al. (2019) and Morris et al. (2019) it is shown that for any input graph if vertices can be distinguished by a GNN then they can be distinguished by the WL algorithm. Conversely, Graph Isomorphism Networks (GINs) were proposed in Xu et al. (2019) that can match the distinguishing power of the WL algorithm, on any graph. The construction of GINs relies on multi-layer perceptrons and their ability to approximate arbitrary functions. Morris et al. (2019) show that the distinguishing power of the WL algorithm can also be matched using the simpler GraphSAGE architecture (Hamilton et al., 2017), provided that the input graph is fixed. We refer to Sato (2020) for a survey on the expressive power of GNNs.

[*]Equal contribution [1]University of Antwerp, Belgium [2]Max Planck Institute for Software Systems, Germany [3]Flanders Make, Belgium. Correspondence to: Floris Geerts <floris.geerts@uantwerpen.be>.

Proceedings of the 38^{th} International Conference on Machine Learning, PMLR 139, 2021.

In this paper we start from the observation that *many popular GNNs fall outside* of the classes of GNNs considered in Xu et al. (2019) and Morris et al. (2019). Prominent examples of such GNNs are the Graph Convolutional Networks (GCNs) from Kipf & Welling (2017). Such GCNs still deploy an iterative neighbourhood aggregation of features but they additionally take into account vertex-degree information. Given the popularity of GCNs, natural questions are: "How do GCNs relate to the WL algorithm?" and "What is the distinguishing power of GCNs?" We will answer these questions in this paper.

To do so, we study the distinguishing power of Message-Passing Neural Network (MPNNs), introduced by Gilmer et al. (2017), which are known to encompass GCNs and many other GNN formalisms. As we will see, formalising GNN architectures in the common MPNN framework allows for a fair comparison of their expressive power. More precisely, in this paper we consider two general classes of MPNNs depending on what information is used during message-passing: *anonymous* MPNNs that do not use degree information, and *degree-aware* MPNNs that do use degree information. The former class covers the GNNs studied in Xu et al. (2019) and Morris et al. (2019), the latter class covers GCNs (Kipf & Welling, 2017), among others.

To understand the distinguishing power of MPNNs we use the WL algorithm as a yardstick. We particularly pay attention to the correspondence between the number of iterations of the WL algorithm and the number of rounds of computation[1] of MPNNs. Since it is known that too many rounds of computation results in over-smoothing and prediction degradation (Wu et al., 2019a; Li et al., 2018; Oono & Suzuki, 2020), in practice a small number (often 3) of rounds is used. It is therefore important to understand the distinguishing power of MPNNs – and hence also of GCNs – in terms of their number of rounds.

Contributions. First, we focus on the power of general MPNN architectures and the number of rounds required to distinguish vertices. Slightly generalising Xu et al. (2019) and Morris et al. (2019), we show that for *anonymous* MPNNs the number of rounds coincides with the number of iterations for the WL algorithm (Proposition 8). By contrast, to simulate *degree-aware* MPNNs an extra iteration in the WL algorithm is needed (Proposition 14).

Second, we focus on variants of particular MPNN architectures that can be found in the literature. As an example of anonymous MPNNs we consider the GNNs from Morris et al. (2019) (Theorem 11). We show that for this architecture the number of rounds still coincides with the number of iterations in the WL algorithm. This result refines the result in Morris et al. (2019). In their proof, to simulate t iterations of the WL algorithm $2t$ rounds of computation are required. *We only need t rounds of computation and we need considerably less parameters.*

As an example of degree-aware MPNNs we consider the GCNs of Kipf & Welling (2017). We show that the WL algorithm *cannot* be simulated by the GCNs from Kipf & Welling (2017) (Proposition 18). This observation is somewhat contradictory to the belief that GCNs can be seen as a "continuous generalisation" of the WL algorithm. However, by introducing a *learnable trade-off parameter* between features of the vertex and those of its neighbours, the simulation of the WL algorithm can be achieved by GCNs (Proposition 19). This minor relaxation of GCNs was already suggested in Kipf & Welling (2017) based on empirical results. Our simulation result thus provides a theoretical justification of this parameter.

We complement these results with some experiments. From our theoretical results we extracted two interesting features.

1. Recall that in practice the number of rounds is limited (e.g. 3) and that degree-aware MPNNs are succinct by one round of computation. Will adding vertex degree information improve anonymous architectures?
2. In the example of anonymous MPNN architectures, to simulate the WL algorithm we obtained more compact architectures. How do these compact architectures compare to previous proposals in terms of accuracy?

We experimentally verify that these two features have a positive impact on the accuracy. Thus as a corollary of our theoretical results we provide experimental justification to take these features into consideration.

Detailed proofs of our results, experiments and additional information can be found in the supplementary material.

2. Preliminaries

Let $\mathbb{A}$ denote the set of all algebraic numbers[2]; $\mathbb{Q}$, the set of all rational numbers; $\mathbb{Z}$, the set of all integer numbers; $\mathbb{N}$, the set of all natural numbers including zero, i.e., $\mathbb{N} = \{0, 1, 2, \dots\}$. We write $\mathbb{S}^+$ to denote the subset of numbers from $\mathbb{S}$ which are strictly positive, e.g., $\mathbb{N}^+ = \mathbb{N} \setminus \{0\}$. We use $\{\}$ and $\{\!\{\}\!\}$ to indicate sets and multisets, respectively.

Labelled graphs. Let $G = (V, E)$ be an undirected graph consisting of $n \in \mathbb{N}$ vertices. Without loss of generality we assume that $V = \{1, 2, \dots, n\}$. Given a vertex $v \in V$,

[1] Roughly speaking, the number of rounds of an MPNN corresponds to the number of (hidden) layers in a GNN.

[2] We use algebraic numbers instead of real or rational numbers because algebraic numbers form a countable set and are closed under taking square roots. These two properties are needed to formally see the WL algorithm as an anonymous MPNN and to model GCNs, respectively.

we denote by $N_G(v)$ its set of neighbours, i.e., $N_G(v) := \{u \mid \{u,v\} \in E\}$. Furthermore, the degree of a vertex v, denoted by d_v, is the number of vertices in $N_G(v)$. With a labelled graph $(G, \boldsymbol{\nu})$ we mean a graph $G = (V, E)$ whose vertices are labelled using a function $\boldsymbol{\nu} : V \to \Sigma$ for some set Σ of labels. We denote by $\boldsymbol{\nu}_v$ the label of $v \in V$.

Henceforth we fix a labelled graph $(G, \boldsymbol{\nu})$ with $G = (V, E)$ and denote by $\mathbf{A}$ the adjacency matrix (of G). We denote by $\mathbf{D}$ the diagonal matrix such that $\mathbf{D}_{vv} = d_v$ for each $v \in V$. We will assume that G does not have isolated vertices and that there are no self-loops. For a matrix $\mathbf{B}$ we denote by $\mathbf{B}_i$ the i-th row of $\mathbf{B}$. If $\mathbf{B}$ is a matrix of dimension $n \times m$, we represent the rows of $\mathbf{B}$ by $\mathbf{B}_v$, for $v \in V$.

We will identify Σ with elements (row vectors) in $\mathbb{A}^s$ for some $s \in \mathbb{N}^+$. In this way, a labelling $\boldsymbol{\ell} : V \to \Sigma$ can be regarded as a matrix in $\mathbb{A}^{n \times s}$ and $\boldsymbol{\ell}_v$ corresponds to the v-th row in that matrix. Conversely, a matrix $\mathbf{L} \in \mathbb{A}^{n \times s}$ can be regarded as the vertex labelling that labels v with the row vector $\mathbf{L}_v$. We use these two interpretations interchangeably.

Given a matrix $\mathbf{L} \in \mathbb{A}^{n \times s}$ and a matrix $\mathbf{L}' \in \mathbb{A}^{n \times s'}$ we say that the vertex labelling $\mathbf{L}'$ is coarser than the vertex labelling $\mathbf{L}$, denoted by $\mathbf{L} \sqsubseteq \mathbf{L}'$, if for all $v, w \in V$, $\mathbf{L}_v = \mathbf{L}_w \Rightarrow \mathbf{L}'_v = \mathbf{L}'_w$. The vertex labellings $\mathbf{L}$ and $\mathbf{L}'$ are equivalent, denoted by $\mathbf{L} \equiv \mathbf{L}'$, if $\mathbf{L} \sqsubseteq \mathbf{L}'$ and $\mathbf{L}' \sqsubseteq \mathbf{L}$ hold. In other words, $\mathbf{L} \equiv \mathbf{L}'$ if and only if for all $v, w \in V$, $\mathbf{L}_v = \mathbf{L}_w \Leftrightarrow \mathbf{L}'_v = \mathbf{L}'_w$.

Weisfeiler-Lehman labelling. Of particular importance is the labelling obtained by colour refinement, also known as the Weisfeiler-Lehman algorithm (or WL algorithm, for short) (Weisfeiler & Lehman, 1968; Grohe, 2017). The WL algorithm constructs a labelling, in an iterative fashion, based on neighbourhood information and the initial vertex labelling. More specifically, given $(G, \boldsymbol{\nu})$, the WL algorithm initially sets $\boldsymbol{\ell}^{(0)} := \boldsymbol{\nu}$. Then, the WL algorithm computes a labelling $\boldsymbol{\ell}^{(t)}$, for $t > 0$, as follows: $\boldsymbol{\ell}_v^{(t)} := \textsc{Hash}((\boldsymbol{\ell}_v^{(t-1)}, \{\!\{\boldsymbol{\ell}_u^{(t-1)} \mid u \in N_G(v)\}\!\}))$, where HASH bijectively maps the above pair, consisting of (i) the previous label $\boldsymbol{\ell}_v^{(t-1)}$ of v; and (ii) the multiset $\{\!\{\boldsymbol{\ell}_u^{(t-1)} \mid u \in N_G(v)\}\!\}$ of labels of the neighbours of v, to a label in Σ which has not been used in previous iterations. When the number of distinct labels in $\boldsymbol{\ell}^{(t)}$ and $\boldsymbol{\ell}^{(t-1)}$ is the same, the WL algorithm terminates. Termination is guaranteed in at most n steps (Immerman & Lander, 1990). By appropriately generalising the WL algorithm (Grohe, 2020) our results carry over to directed edge-labelled graphs. For ease of presentation, we do not detail this further in this paper.

3. Message Passing Neural Networks

We start by describing message passing neural networks (MPNNs) for deep learning on graphs, introduced by Gilmer et al. (2017). Roughly speaking, in MPNNs, vertex labels are propagated through a graph according to its connectivity structure: Given a labelled graph $(G, \boldsymbol{\nu})$ and a computable function $f : V \to \mathbb{A}$ an MPNN computes a vertex labelling $\boldsymbol{\ell} : V \to \mathbb{A}^s$, for some $s \in \mathbb{N}^+$. We introduce the function f to *explicitly describe additional vertex parameters in the message functions*.[3] The vertex labelling computed by an MPNN is computed in a finite number of rounds T. After round $0 \leq t \leq T$ the labelling is denoted by $\boldsymbol{\ell}^{(t)}$. We next detail how $\boldsymbol{\ell}^{(t)}$ is computed. For $t = 0$, we let $\boldsymbol{\ell}^{(0)} := \boldsymbol{\nu}$. Then, for every round $t = 1, 2, \ldots, T$, we define $\boldsymbol{\ell}^{(t)} : V \to \mathbb{A}^{s_t}$, as follows:

$$\boldsymbol{\ell}_v^{(t)} := \textsc{Upd}^{(t)}\Big(\boldsymbol{\ell}_v^{(t-1)}, \sum_{u \in N_G(v)} \textsc{Msg}^{(t)}(\boldsymbol{\ell}_v^{(t-1)}, \boldsymbol{\ell}_u^{(t-1)}, f(v), f(u))\Big) \in \mathbb{A}^{s_t}.$$

That is, each vertex $v \in V$ receives messages from its neighbours which are subsequently aggregated. Formally, the function $\textsc{Msg}^{(t)}$ receives as input f applied to two vertices v and u, and the corresponding labels of these vertices from the previous iteration $\boldsymbol{\ell}_v^{(t-1)}$ and $\boldsymbol{\ell}_u^{(t-1)}$, and outputs a label.[4] Then, for every vertex v, we aggregate by summing all such labels for every neighbour u. Finally, the function $\textsc{Upd}^{(t)}$ updates the result using also the current label $\boldsymbol{\ell}_v^{(t-1)}$. After round T, we define the final labelling $\boldsymbol{\ell} : V \to \mathbb{A}^s$ as $\boldsymbol{\ell}_v := \boldsymbol{\ell}_v^{(T)}$ for every $v \in V$.

The role of the function f in this paper is to distinguish between two classes of MPNNs[5]: those whose message functions only depend on the labels of the vertices involved, in which case we set f to the zero function $f(v) = 0$, for all $v \in V$; and those whose message functions depend on the labels and on the degrees of the vertices involved, in which case we set f to the degree function $f(v) = d_v$, for all $v \in V$. We will refer to the former class as *anonymous* MPNNs and to the latter as *degree-aware* MPNNs. These classes are denoted by $\mathcal{M}_{anon}$ and $\mathcal{M}_{deg}$, respectively. Notice that the anonymous class does not use degree information explicitly (i.e. in the message function). However, we do not impose any semantic restriction disallowing anonymous MPNNs to learn or compute the degree information, and thus use degree information in an implicit way.

3.1. Examples

Example 1 (GNN architectures)**.** We first consider the graph neural network architectures (Hamilton et al., 2017;

[3] In the formalisation by Gilmer et al. (2017) it is not always clear on which parameters the message functions may depend on.

[4] Notice that this is a feedforward model. The layers are ordered and every layer depends only on the previous one.

[5] In general, one could consider any function f.

Morris et al., 2019) defined by:

$$\mathbf{L}^{(t)} := \sigma\left(\mathbf{L}^{(t-1)}\mathbf{W}_1^{(t)} + \mathbf{A}\mathbf{L}^{(t-1)}\mathbf{W}_2^{(t)} + \mathbf{B}^{(t)}\right), \quad (1)$$

where $\mathbf{L}^{(t)}$ is the matrix in $\mathbb{A}^{n\times s_t}$ consisting of the n rows $\boldsymbol{\ell}_v^{(t)} \in \mathbb{A}^{s_t}$, for $v \in V$, $\mathbf{A} \in \mathbb{A}^{n\times n}$ is the adjacency matrix of G, $\mathbf{W}_1^{(t)}$ and $\mathbf{W}_2^{(t)}$ are (learnable) weight matrices in $\mathbb{A}^{s_{t-1}\times s_t}$, $\mathbf{B}^{(t)}$ is a bias matrix in $\mathbb{A}^{n\times s_t}$ consisting of n copies of the same row $\mathbf{b}^{(t)} \in \mathbb{A}^{s_t}$, and σ is a non-linear activation function. We can regard this architecture as an MPNN. Indeed, (1) can be equivalently phrased as the architecture which computes, in round t, for each vertex $v \in V$ the label defined by:

$$\boldsymbol{\ell}_v^{(t)} := \sigma\left(\boldsymbol{\ell}_v^{(t-1)}\mathbf{W}_1^{(t)} + \sum_{u\in N_G(v)} \boldsymbol{\ell}_u^{(t-1)}\mathbf{W}_2^{(t)} + \mathbf{b}^{(t)}\right),$$

where we identified the labellings with their images, i.e., a row vector in $\mathbb{A}^{s_{t-1}}$ or $\mathbb{A}^{s_t}$. To phrase this as an MPNN, it suffices to define for each $\mathbf{x}$ and $\mathbf{y}$ in $\mathbb{A}^{s_{t-1}}$, each $v \in V$ and $u \in N_G(v)$, and each $t \geq 1$:

$$\begin{aligned} \textsc{Msg}^{(t)}(\mathbf{x},\mathbf{y},-,-) &:= \mathbf{y}\mathbf{W}_2^{(t)}, \\ \textsc{Upd}^{(t)}(\mathbf{x},\mathbf{m}) &:= \sigma(\mathbf{x}\mathbf{W}_1^{(t)} + \mathbf{m} + \mathbf{b}^{(t)}) \end{aligned} \quad (2)$$

We write $-$ instead of 0 to emphasise that the message functions use the zero function $f(v) = 0$, for all $v \in V$, and hence do not depend on $f(v)$ and $f(u)$. In other words, the MPNN constructed is an anonymous MPNN. Without loss of generality we will assume that anonymous MPNNs do not use $f(v)$ and $f(u)$ in the messages. If they do then one can replace them with 0. This way it is easy to see that classes of MPNNs that use different functions f in the messages contain the class of anonymous MPNNs. We denote the class of anonymous MPNNs of the form (2) by $\mathcal{M}^{\sigma}_{GNN}$ for activation function σ. □

Another example of an anonymous MPNN originates from the Weisfeiler-Lehman algorithm described in the preliminaries.

Example 2 (Weisfeiler-Lehman)**.** We recall that WL computes, in round $t \geq 1$, for each vertex $v \in V$ the label:

$$\boldsymbol{\ell}_v^{(t)} := \textsc{Hash}\left(\boldsymbol{\ell}_v^{(t-1)}, \{\!\{\boldsymbol{\ell}_u^{(t-1)} \mid u \in N_G(v)\}\!\}\right).$$

Let us assume that the set Σ of labels is $\mathbb{A}^s$ for some fixed $s \in \mathbb{N}^+$. We cast the WL algorithm as an anonymous MPNN by using an injection $h : \mathbb{A}^s \to \mathbb{Q}$. What follows is in fact an adaptation of Lemma 5 from Xu et al. (2019) itself based on Zaheer et al. (2017, Theorem 2). We crucially rely on the fact that the set $\mathbb{A}$ of algebraic numbers is countable (see e.g., Theorem 2.2 Jarvis, 2014). As a consequence, also $\mathbb{A}^s$ is countable.

Let $\tau : \mathbb{A}^s \to \mathbb{N}^+$ be a computable injective function witnessing the countability of $\mathbb{A}^s$. For instance, since elements of $\mathbb{A}$ are encoded as a polynomial $a_0 + a_1x^1 + a_2x^2 + \cdots + a_kx^k \in \mathbb{Z}[x]$ and a pair $n_1/d_1, n_2/d_2$ of rationals, τ can be taken to be the composition of the injection $\alpha : \mathbb{A} \to \mathbb{N}^+$, applied point-wise, and the Cantor tuple function, where

$$\alpha(a_0 + a_1x^1 + a_2x^2 + \cdots + a_kx^k, n_1/d_1, n_2/d_2) \mapsto$$
$$p(1,n_1)p(2,n_2)p(3,d_1)p(4,d_2)\prod_{i=0}^{k} p(i+5,a_i)$$

with π_i being the i-th prime number in

$$p(i,z) = \begin{cases} \pi_{2i}^{z} & \text{if } z \geq 0 \\ \pi_{2i+1}^{-z} & \text{if } z < 0. \end{cases}$$

We next define $h : \mathbb{A}^s \to \mathbb{Q}^+$ as the mapping $\mathbf{x} \mapsto (n+1)^{-\tau(\mathbf{x})}$. Note that h is injective and $h(\mathbf{x})$ can be seen as a number whose $(n+1)$-ary representation has a single nonzero digit. We next observe that the multiplicity of every element in $S := \{\!\{\boldsymbol{\ell}_u^{(t-1)} \mid u \in N_G(v)\}\!\}$ is bounded by the number of all vertices n — and this for all $t \geq 1$. It follows that the function ϕ mapping any such S to $\sum_{\mathbf{x}\in S} h(\mathbf{x})$ is an injection from $\mathbb{A}^s$ to $\mathbb{Q}$. Therefore, the summands can be recovered by looking at the $(n+1)$-ary representation of the sum, and thus the inverse of ϕ is computable on its image. To conclude, we define the message function

$$\textsc{Msg}^{(t)}(\mathbf{x},\mathbf{y},-,-) := h(\mathbf{y}) \in \mathbb{A}.$$

The update function is defined by

$$\textsc{Upd}^{(t)}(\mathbf{x},y) := \textsc{Hash}(\mathbf{x}, \phi^{-1}(y)),$$

where $y \in \mathbb{A}$ since it corresponds to a sum of messages, themselves algebraic numbers. As before, we write $-$ instead of 0 to emphasise that the message functions use the zero function $f(v) = 0$, for all $v \in V$, and hence do not depend on $f(v)$ and $f(u)$. □

We conclude with an example of a degree-aware MPNN. We study degree-aware MPNNs in Section 6.

Example 3 (GCNs by Kipf and Welling)**.** We consider the GCN architecture by Kipf & Welling (2017), which in round $t \geq 1$ computes

$$\mathbf{L}^{(t)} := \sigma\left((\mathbf{D}+\mathbf{I})^{-1/2}(\mathbf{A}+\mathbf{I})(\mathbf{D}+\mathbf{I})^{-1/2}\mathbf{L}^{(t-1)}\mathbf{W}^{(t)}\right),$$

where we use the same notation as in Example 1 but now with a single (learnable) weight matrix $\mathbf{W}^{(t)}$ in $\mathbb{A}^{s_{t-1}\times s_t}$. This means that, in round t, for each vertex $v \in V$ it computes the label $\boldsymbol{\ell}_v^{(t)} := \sigma(a)$ where

$$a = \left(\frac{1}{1+d_v}\right)\boldsymbol{\ell}_v^{(t-1)}\mathbf{W}^{(t)} + \sum_{u\in N_G(v)} \left(\frac{1}{\sqrt{1+d_v}}\right)\left(\frac{1}{\sqrt{1+d_u}}\right)\boldsymbol{\ell}_u^{(t-1)}\mathbf{W}^{(t)}. \quad (3)$$

We can regard this architecture again as an MPNN. Indeed, it suffices to define for each $\mathbf{x}$ and $\mathbf{y}$ in $\mathbb{A}^{s_{t-1}}$, each $v \in V$ and $u \in N_G(v)$, and each $t \geq 1$:

$$\text{MSG}^{(t)}(\mathbf{x}, \mathbf{y}, d_v, d_u) := \frac{1}{d_v}\left(\frac{1}{1+d_v}\right)\mathbf{x}\mathbf{W}^{(t)} + \left(\frac{1}{\sqrt{1+d_v}}\right)\left(\frac{1}{\sqrt{1+d_u}}\right)\mathbf{y}\mathbf{W}^{(t)}$$

$$\text{UPD}^{(t)}(\mathbf{x}, \mathbf{y}) := \sigma(\mathbf{y}).$$

We remark that the initial factor $1/d_v$ in the message functions is introduced for renormalisation purposes. We indeed observe that the message functions depend only on $\boldsymbol{\ell}_v^{(t-1)}$, $\boldsymbol{\ell}_u^{(t-1)}$, and the degrees d_v and d_u of the vertices v and u, respectively. We note that the use of algebraic numbers allows one to consider square roots, needed to view GCNs as degree-aware MPNNs. □

4. Comparing Classes of MPNNs

The distinguishing power of MPNNs relates to their ability to distinguish vertices based on the labellings that they compute. We are interested in comparing the distinguishing power of classes of MPNNs, taking the number of rounds into account. For a given labelled graph $(G, \boldsymbol{\nu})$ and MPNN M, we denote by $\boldsymbol{\ell}_M^{(t)}$ the vertex labelling computed by M after t rounds. Formally, $\boldsymbol{\ell}_M^{(t)}$ depends on the graph $(G, \boldsymbol{\nu})$ but we drop this additional dependency from the notation for readability.

Definition 4. Consider MPNNs M_1 and M_2 with the same number of rounds T. Let $\boldsymbol{\ell}_{M_1}^{(t)}$ and $\boldsymbol{\ell}_{M_2}^{(t)}$ be their labellings on an input graph $(G, \boldsymbol{\nu})$ obtained after t rounds of computation for every $0 \leq t \leq T$. Then M_1 is said to be $(G, \boldsymbol{\nu})$-*weaker* than M_2, denoted by $M_1 \preceq M_2$, if M_1 cannot distinguish more vertices than M_2 in every round of computation. More formally, $M_1 \preceq M_2$ if $\boldsymbol{\ell}_{M_2}^{(t)} \sqsubseteq \boldsymbol{\ell}_{M_1}^{(t)}$ for every $t \geq 0$. In this case we also say that M_2 is $(G, \boldsymbol{\nu})$-*stronger* than M_1. When $(G, \boldsymbol{\nu})$ is clear from the context we will write weaker and stronger for simplicity.

Notice that two MPNNs can compare differently for different graphs. For a comparison that does not depend on graphs we lift this notion to classes $\mathcal{M}_1$ and $\mathcal{M}_2$ of MPNNs (i.e. sets of MPNNs).

Definition 5. Consider two classes $\mathcal{M}_1$ and $\mathcal{M}_2$ of MPNNs. Then, $\mathcal{M}_1$ is said to be *weaker* than $\mathcal{M}_2$, denoted by $\mathcal{M}_1 \preceq \mathcal{M}_2$, if for all $M_1 \in \mathcal{M}_1$ and for all labelled graphs $(G, \boldsymbol{\nu})$ there exists an $M_2 \in \mathcal{M}_2$ which is $(G, \boldsymbol{\nu})$-stronger than M_1.

Note that we use the words stronger and weaker with a non-strict meaning. In particular if a class is both stronger and weaker compared to another one we say that they are equally strong. Formally, $\mathcal{M}_1$ and $\mathcal{M}_2$ are *equally strong*, denoted by $\mathcal{M}_1 \equiv \mathcal{M}_2$, if $\mathcal{M}_1 \preceq \mathcal{M}_2$ and $\mathcal{M}_2 \preceq \mathcal{M}_1$ hold.

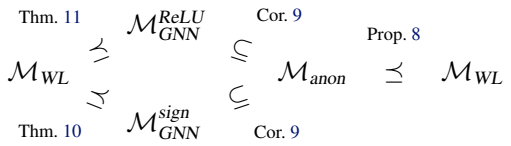

Figure 1. Summary of relationships amongst major anonymous MPNN classes considered in Section 5.

We will also need a generalisation of the previous definitions in which we compare labellings computed by MPNNs at different rounds. This is formalised as follows.

Definition 6. Consider MPNNs M_1 and M_2 with T_1 and T_2 rounds, respectively. Let $\boldsymbol{\ell}_{M_1}^{(t)}$ and $\boldsymbol{\ell}_{M_2}^{(t)}$ be their labellings on an input graph $(G, \boldsymbol{\nu})$ obtained after t rounds of computation. Let $g : \mathbb{N} \to \mathbb{N}$ be a monotonic function such that $g(T_1) = T_2$ (thus $T_1 \leq T_2$). We write $M_1 \preceq_g M_2$, if $\boldsymbol{\ell}_{M_2}^{g(t)} \sqsubseteq \boldsymbol{\ell}_{M_1}^{(t)}$ for every $0 \leq t \leq T_1$.

If $M_1 \preceq_g M_2$ and $g(t) = t + 1$ then we say that M_1 is weaker than M_2 *with one step ahead* and write $M_1 \preceq_{+1} M_2$; if $g(t) = 2t$, M_1 is weaker than M_2 *with a factor of* 2, and write $M_1 \preceq_{\times 2} M_2$. We lift these definitions to classes of MPNNs, just like in Definition 5.

5. The Power of Anonymous MPNNs

In this section we compare classes of anonymous MPNNs (aMPNNs for short) in terms of their distinguishing power using Definition 5. Let $(G, \boldsymbol{\nu})$ be a labelled graph. We will consider the following classes of aMPNNs. We denote by $\mathcal{M}_{WL}$ the class of aMPNNs consisting of an aMPNN M_{WL}^T, for each $T \in \mathbb{N}$, originating from the WL algorithm being ran for T rounds (see Example 2 in Section 3.1). We write M_{WL} when T is clear from the context. Recall that the class of anonymous MPNNs is denoted $\mathcal{M}_{anon}$ and recall the architectures $\mathcal{M}_{GNN}^{ReLU}$ and $\mathcal{M}_{GNN}^{sign}$ (Example 1 in Section 3.1).

The following is the main result in this section.

Theorem 7. *The classes $\mathcal{M}_{WL}$, $\mathcal{M}_{GNN}^{ReLU}$, $\mathcal{M}_{GNN}^{sign}$ and $\mathcal{M}_{anon}$ are all equally strong.*

We prove this theorem in the following subsections by providing the relationships that are summarised in Figure 1.

5.1. General anonymous MPNNs

We focus on the relation between the WL algorithm and anonymous MPNNs in general: they are equally strong. First, note that $\mathcal{M}_{WL}$ is weaker than $\mathcal{M}_{anon}$ since $\mathcal{M}_{WL} \subseteq \mathcal{M}_{anon}$. Second, the fact that $\mathcal{M}_{anon}$ is weaker than $\mathcal{M}_{WL}$

follows from a technical adaptation of the proofs of Lemma 2 in Xu et al. (2019) and Theorem 5 in Morris et al. (2019).

Proposition 8 (Based on Xu et al. (2019); Morris et al. (2019))**.** *The classes $\mathcal{M}_{anon}$ and $\mathcal{M}_{WL}$ are equally strong.*

We remark that we cannot use the results in Xu et al. (2019) and Morris et al. (2019) as a black box because the class $\mathcal{M}_{anon}$ is more general than the class considered in those papers: indeed, we do allow a dependency on label $\boldsymbol{\ell}_v^{(t-1)}$ in the message functions whereas Xu et al. (2019) and Morris et al. (2019) do not. Their MPNNs can, however, also be cast as aMPNNs.

5.2. Graph neural network-based aMPNNs

In this subsection we study the subclasses of aMPNNs arising from the graph neural network architectures of Morris et al. (2019). Let us write $\mathcal{M}_{GNN} := \mathcal{M}_{GNN}^{sign} \cup \mathcal{M}_{GNN}^{ReLU}$.

We start by stating a direct consequence of Proposition 8. It follows by recalling that $\mathcal{M}_{GNN}$ is a subclass of $\mathcal{M}_{anon}$.

Corollary 9. *The class $\mathcal{M}_{GNN}$ is weaker than $\mathcal{M}_{anon}$ and is thus also weaker than $\mathcal{M}_{WL}$.*

More challenging is to show that $\mathcal{M}_{GNN}^{sign}$, $\mathcal{M}_{GNN}^{ReLU}$ and $\mathcal{M}_{WL}$, and thus also $\mathcal{M}_{anon}$, are equally strong. The following results are known.

Theorem 10 (Morris et al., 2019)**.** *(i) The classes $\mathcal{M}_{GNN}^{sign}$ and $\mathcal{M}_{WL}$ are equally strong. (ii) The class $\mathcal{M}_{GNN}^{ReLU}$ is weaker than $\mathcal{M}_{WL}$, and $\mathcal{M}_{WL}$ is weaker than $\mathcal{M}_{GNN}^{ReLU}$, with a factor of two, i.e., $\mathcal{M}_{WL} \preceq_{\times 2} \mathcal{M}_{GNN}^{ReLU}$.*

The factor of two is due to a simulation of the sign function by means of a two-fold application of the ReLU function. We show that this can be avoided.

Theorem 11. *The classes $\mathcal{M}_{GNN}^{ReLU}$ and $\mathcal{M}_{WL}$ are equally strong.*

Proof sketch. We already know that $\mathcal{M}_{GNN}^{ReLU}$ is weaker than $\mathcal{M}_{WL}$ (Theorem 10 and Corollary 9). It remains to show that $\mathcal{M}_{WL}$ is weaker than $\mathcal{M}_{GNN}^{ReLU}$. That is, given an aMPNN and a labelled graph M_{WL}, we need to construct an aMPNN M in $\mathcal{M}_{GNN}^{ReLU}$ such that $\boldsymbol{\ell}_M^{(t)} \sqsubseteq \boldsymbol{\ell}_{M_{WL}}^{(t)}$, for all $t \geq 0$. We observe that since $\boldsymbol{\ell}_{M_{WL}}^{(t)} \sqsubseteq \boldsymbol{\ell}_M^{(t)}$ for any M in $\mathcal{M}_{GNN}^{ReLU}$, this is equivalent to constructing an M such that $\boldsymbol{\ell}_M^{(t)} \equiv \boldsymbol{\ell}_{M_{WL}}^{(t)}$.

The aMPNN M in $\mathcal{M}_{GNN}^{ReLU}$ that we construct uses message and update functions of the form:

$$\textsc{Msg}^{(t)}(\mathbf{x}, \mathbf{y}, -, -) := \mathbf{y}\mathbf{W}^{(t)}$$
$$\textsc{Upd}^{(t)}(\mathbf{x}, \mathbf{y}) := \text{ReLU}(p\mathbf{x}\mathbf{W}^{(t)} + \mathbf{y} + \mathbf{b}^{(t)})$$

for some value $p \in \mathbb{A}$, $0 < p < 1$, weight matrix $\mathbf{W}^{(t)} \in \mathbb{A}^{s_{t-1} \times s_t}$, and bias vector $\mathbf{b}^{(t)} \in \mathbb{A}^{s_t}$.

Note that, in contrast to aMPNNs of the form (2), we only have one weight matrix per round, instead of two, at the cost of introducing an extra parameter $p \in \mathbb{A}$. Moreover, the aMPNN constructed in Morris et al. (2019) uses two distinct weight matrices in $\mathbb{A}^{(s_{t-1}+s_0)\times(s_t+s_0)}$ whereas ours are elements of $\mathbb{A}^{s_{t-1}\times s_t}$ and thus of smaller dimension. Furthermore, we can assume the bias vector $\mathbf{b}^{(t)}$ to be chosen as $q\mathbf{1}$, where $q \in \mathbb{R}$ is a parameter independent of the layer, and $\mathbf{1} \in \mathbb{R}^{1\times s_t}$ is the all ones row vector of appropriate dimension. □

Remark that the factor two, needed for ReLU in Theorem 10, has been eliminated. Phrased in linear algebra, we consider the class $\mathcal{M}_{GNN^-}$ consisting of aMPNNs of the form

$$\mathbf{L}^{(t)} = \sigma((\mathbf{A} + p\mathbf{I})\mathbf{L}^{(t-1)}\mathbf{W}^{(t)} - q\mathbf{J}), \tag{4}$$

and, thus, these suffice to implement the WL algorithm. Here, $\mathbf{J}$ denotes the all ones matrix of appropriate dimension. We thus have obtained a simple class of aMPNNs, $\mathcal{M}_{GNN^-}$, which is equally strong as $\mathcal{M}_{WL}$. We will see in the next section that the parameter p also plays an important role for degree-aware MPNNs.

6. The Power of Degree-Aware MPNNs

In this section we compare various classes of degree-aware MPNNs in terms of their distinguishing power. We recall that degree-aware MPNNs (dMPNNs for short) have message functions that depend on the labels and degrees of vertices. To compare these classes we use Definition 5 and also Definition 6. In the latter definition we will be interested in establishing that classes of dMPNNs are weaker or stronger with 1 step ahead. We will also compare degree-aware MPNNs with anonymous MPNNs. Recall that by Theorem 7 all classes of anonymous MPNNs considered in Section 5 are equivalent for $\equiv$. In particular, they are all equivalent to the class $\mathcal{M}_{WL}$. Therefore, instead of comparing a class $\mathcal{M}$ of dMPNNs with all classes considered in Section 5 it suffices to compare it with $\mathcal{M}_{WL}$.

Quintessential examples of degree-aware MPNNs are the popular graph convolutional networks, as introduced by Kipf & Welling (2017). These are of the form dGNN_4 in Table 1, as already described in Example 3 in Section 3.1. In fact, many commonly used graph neural networks use degree information. We list a couple of such formalisms, taken from the literature, in Table 1. It is easily verified that these can all be cast as dMPNNs along the same lines as graph convolutional networks.

We consider the following classes of dMPNNs. First, we recall that $\mathcal{M}_{deg}$ is the class of degree-aware MPNNs. Furthermore, for $i \in \{1, 2, \ldots, 6\}$, we define $\mathcal{M}_{dGNN_i}$ as the class of dMPNNs originating from a GNN of the form dGNN_i, from Table 1, by varying the weight matrices $\mathbf{W}^{(t)}$ and,

dGNN_1: $\sigma\left(\mathbf{D}^{-1}\mathbf{A}\mathbf{L}^{(t-1)}\mathbf{W}^{(t)}\right)$
dGNN_2: $\sigma\left(\mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}\mathbf{L}^{(t-1)}\mathbf{W}^{(t)}\right)$
dGNN_3: $\sigma\left((\mathbf{D}+\mathbf{I})^{-1}(\mathbf{A}+\mathbf{I})\mathbf{L}^{(t-1)}\mathbf{W}^{(t)}\right)$
dGNN_4: $\sigma\left((\mathbf{D}+\mathbf{I})^{-1/2}(\mathbf{A}+\mathbf{I})(\mathbf{D}+\mathbf{I})^{-1/2}\mathbf{L}^{(t-1)}\mathbf{W}^{(t)}\right)$
dGNN_5: $\sigma\left((\mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}+\mathbf{I})\mathbf{L}^{(t-1)}\mathbf{W}^{(t)}\right)$
dGNN_6: $\sigma\left((r\mathbf{I}+(1-r)\mathbf{D})^{-1/2}(\mathbf{A}+p\mathbf{I})(r\mathbf{I}+(1-r)\mathbf{D})^{-1/2}\mathbf{L}^{(t-1)}\mathbf{W}^{(t)}\right)$

Table 1. Various graph neural networks taken from Kipf & Welling (2017); Wu et al. (2019a) and Meltzer et al. (2019), which correspond to degree-aware MPNNs. We assume the presence of a bias matrix $\mathbf{B}^{(t)}$ consisting of copies of the same row $\mathbf{b}^{(t)}$.

when applicable, the bias $\mathbf{B}^{(t)}$ and parameters p and r. The following is the main result for this section.

Theorem 12. *For the class of degree-aware MPNNs:*

1. $\mathcal{M}_{WL} \preceq \mathcal{M}_{deg}$ *and* $\mathcal{M}_{deg} \not\preceq \mathcal{M}_{WL}$*;*
2. $\mathcal{M}_{deg} \preceq_{+1} \mathcal{M}_{WL}$*.*

For the architectures from Table 1:

3. $\mathcal{M}_{dGNN_i} \not\preceq \mathcal{M}_{WL}$ *for* $i = 2, 4, 5, 6$ *and* $\mathcal{M}_{dGNN_i} \preceq \mathcal{M}_{WL}$ *for* $i = 1, 3$*;*
4. $\mathcal{M}_{WL} \not\preceq \mathcal{M}_{dGNN_i}$ *for* $1 \leq i \leq 5$ *and* $\mathcal{M}_{WL} \preceq \mathcal{M}_{dGNN_6}$*.*

We prove this theorem in the following subsections by providing the relationships that are summarised in Figure 2.

6.1. General degree-aware MPNNs

We first focus on the relation between the WL algorithm and dMPNNs in general. More specifically, we start with the first item in Theorem 12. As part of the proof we show that $\mathcal{M}_{dGNN_4} \not\preceq \mathcal{M}_{WL}$. We can similarly show that $\mathcal{M}_{dGNN_2}, \mathcal{M}_{dGNN_5}, \mathcal{M}_{dGNN_6} \not\preceq \mathcal{M}_{WL}$, hereby also settling the first part of the third item in Theorem 12.

Proposition 13. *The class* $\mathcal{M}_{WL}$ *is weaker than* $\mathcal{M}_{deg}$*; but the class* $\mathcal{M}_{deg}$ *is not weaker than* $\mathcal{M}_{WL}$*.*

Proof sketch. Notice that $\mathcal{M}_{anon}$ is weaker than $\mathcal{M}_{deg}$, simply because any aMPNN is a dMPNN. The first part of the claim thus follows from Theorem 7.

For the second part it suffices to provide a dMPNN M and a labelled graph $(G, \boldsymbol{\nu})$ such that there is a round $t \geq 0$ for which $\boldsymbol{\ell}^{(t)}_{M_{WL}} \not\sqsubseteq \boldsymbol{\ell}^{(t)}_{M}$ holds.

We construct a dMPNN M from $\mathcal{M}_{dGNN_4}$. Consider the labelled graph $(G, \boldsymbol{\nu})$ with vertex labelling $\boldsymbol{\nu}_{v_1} = \boldsymbol{\nu}_{v_2} = (1,0,0)$, $\boldsymbol{\nu}_{v_3} = \boldsymbol{\nu}_{v_6} = (0,1,0)$ and $\boldsymbol{\nu}_{v_4} = \boldsymbol{\nu}_{v_5} = (0,0,1)$, and edges $\{v_1, v_3\}$, $\{v_2, v_3\}$, $\{v_3, v_4\}$, $\{v_4, v_5\}$, and $\{v_5, v_6\}$. Finally, we define $\mathbf{W}^{(1)} := \left(\begin{smallmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{smallmatrix}\right)$. It can be verified that $(\boldsymbol{\ell}^{(1)}_{M})_{v_4} \neq (\boldsymbol{\ell}^{(1)}_{M})_{v_5}$. We note, however, that $(\boldsymbol{\ell}^{(1)}_{M_{WL}})_{v_4} = \textsc{Hash}((0,0,1), \{\!\{(0,0,1), (0,1,0)\}\!\}) = (\boldsymbol{\ell}^{(1)}_{M_{WL}})_{v_5}$. Hence, $\boldsymbol{\ell}^{(1)}_{M_{WL}} \not\sqsubseteq \boldsymbol{\ell}^{(1)}_{M}$. □

We now focus on the second item in Theorem 12.

Proposition 14. *It holds that* $\mathcal{M}_{deg} \preceq_{+1} \mathcal{M}_{WL}$*.*

We will need the following lemma that states that anonymous MPNNs can compute the degrees of vertices in the first round of computation.

Lemma 15. *Let* $(G, \boldsymbol{\nu})$ *be a labelled graph with* $\boldsymbol{\nu} : V \to \mathbb{A}^s$*. There exists an aMPNN* M_d *such that* $(\boldsymbol{\ell}^{(1)}_{M_d})_v = (\boldsymbol{\nu}_v, d_v) \in \mathbb{A}^{s+1}$ *for every vertex* v *in* V*.*

The proof of Proposition 14 is an application of the lemma.

Proof sketch. Let $(G, \boldsymbol{\nu})$ be a labelled graph with $\boldsymbol{\nu} : V \to \mathbb{A}^{s_0}$. Take an arbitrary dMPNN M_1 such that for every round $t \geq 1$ the message function is $\textsc{Msg}^{(t)}_{M_1}(\mathbf{x}, \mathbf{y}, d_v, d_u) \in \mathbb{A}^{s'_t}$ and $\textsc{Upd}^{(t)}_{M_1}(\mathbf{x}, \mathbf{m})$ is the update function.

We construct an aMPNN M_2 such that $\boldsymbol{\ell}^{(t+1)}_{M_2} \sqsubseteq \boldsymbol{\ell}^{(t)}_{M_1}$ holds. We keep as an invariant **(I1)** stating that for all v if $\mathbf{x}' = (\boldsymbol{\ell}^{(t)}_{M_1})_v \in \mathbb{A}^{s_t}$ then $\mathbf{x} = (\mathbf{x}', d_v) = (\boldsymbol{\ell}^{(t+1)}_{M_2})_v \in \mathbb{A}^{s_t+1}$. For $t = 1$, we let $\textsc{Msg}^{(1)}_{M_2}$ and $\textsc{Upd}^{(1)}_{M_2}$ be the functions defined by Lemma 15. Then, in each round $t \geq 2$, M_2 extracts the degrees from the last entries in the labels and simulates round t of M_1. It is readily verified that $\boldsymbol{\ell}^{(t+1)}_{M_2} \sqsubseteq \boldsymbol{\ell}^{(t)}_{M_1}$ for every t, and that **(I1)** holds. □

In particular it follows that for the dMPNN M constructed in the proof of Proposition 13 it holds that $\boldsymbol{\ell}^{(2)}_{M_{WL}} \sqsubseteq \boldsymbol{\ell}^{(1)}_{M}$.

6.2. Graph neural network-based dMPNNs

We next consider the relation between the WL algorithm and dMPNNs that originate from graph neural networks as those listed in Table 1. More specifically, we consider the following general graph neural network architecture

$$\begin{aligned}\mathbf{L}^{(t)} := \sigma(&\mathbf{L}^{(t-1)}\mathbf{W}^{(t)}_1 + \mathbf{B}^{(t)} \\ &+ \text{diag}(\mathbf{g})(\mathbf{A} + p\mathbf{I})\text{diag}(\mathbf{h})\mathbf{L}^{(t-1)}\mathbf{W}^{(t)}_2),\end{aligned} \tag{5}$$

where $p \in \mathbb{A}$ is parameter satisfying $0 \leq p \leq 1$, $\mathbf{W}^{(t)}_1$ and $\mathbf{W}^{(t)}_2$ are learnable weight matrices in $\mathbb{A}^{s_{t-1} \times s_t}$, $\mathbf{B}^{(t)}$ is a bias matrix consisting of n copies of the same row $\mathbf{b}^{(t)}$, and $\text{diag}(\mathbf{g})$ and $\text{diag}(\mathbf{h})$ are positive diagonal matrices in $\mathbb{A}^{n \times n}$ obtained by putting the vectors $\mathbf{g}$ and $\mathbf{h}$ in $\mathbb{A}^n$ on their diagonals, respectively. We only consider vectors $\mathbf{g}$ and $\mathbf{h}$ which are *degree-determined*. That is, when $d_v = d_w$ then $\mathbf{g}_v = \mathbf{g}_w$ and $\mathbf{h}_v = \mathbf{h}_w$ for all vertices v and w. Example vectors that satisfy this are: $\mathbf{g}_v = d_v$; or $\mathbf{g}_v = c$ (i.e. the

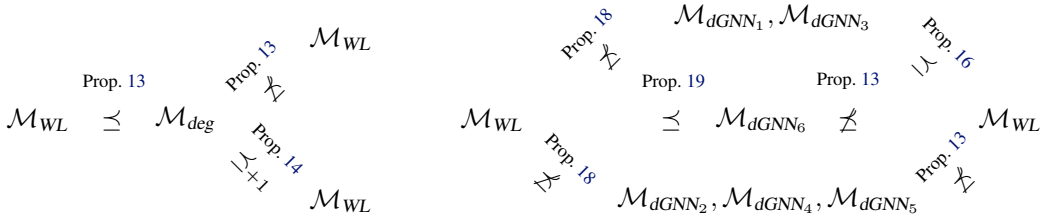

Figure 2. Summary of results comparing degree-aware MPNNs in Theorem 12. We note that Proposition 13 shows only $\mathcal{M}_{dGNN_4} \not\preceq \mathcal{M}_{WL}$, but $\mathcal{M}_{dGNN_2}, \mathcal{M}_{dGNN_5}, \mathcal{M}_{dGNN_6} \not\preceq \mathcal{M}_{WL}$ can be easily inferred from it.

same constant everywhere) for all v. Furthermore, σ is either the sign or the ReLU function.

All graph neural networks in Table 1 can be seen as special cases of (5). Moreover, graph neural networks of the form (5) can be cast as dMPNNs. The reason that one obtains dMPNNs is because of the degree-determinacy assumption. We define the class $\mathcal{M}_{dGNN}$ as the class of dMPNNs with message and update functions of the form:

$$\begin{aligned} \textsc{Msg}^{(t)}(\mathbf{x}, \mathbf{y}, d_v, d_u) &:= g(d_v)h(d_u)\mathbf{y}\mathbf{W}_2^{(t)} \\ &+ {}^1\!/_{d_v}(\mathbf{x}\mathbf{W}_1^{(t)} + pg(d_v)h(d_v)\mathbf{x}\mathbf{W}_2^{(t)}) + \mathbf{b}^{(t)} \end{aligned} \tag{6}$$

$$\textsc{Upd}^{(t)}(\mathbf{x}, \mathbf{m}) := \sigma(\mathbf{m}) \tag{7}$$

for any $\mathbf{x}, \mathbf{y} \in \mathbb{A}^{s_{t-1}}$, $0 \leq p \leq 1$, $\mathbf{m} \in \mathbb{A}^{s_t}$ $\mathbf{W}_1^{(t)} \in \mathbb{A}^{s_{t-1} \times s_t}$,$\mathbf{W}_2^{(t)} \in \mathbb{A}^{s_{t-1} \times s_t}$, bias vector $\mathbf{b}^{(t)} \in \mathbb{A}^{s_t}$, nonlinear activation function σ, and functions $g, h : \mathbb{A} \to \mathbb{A}$. We note that this encoding is just a generalisation of the encoding of GCNs as dMPNNs.

We know from Proposition 14 that the class $\mathcal{M}_{dGNN}$ is weaker than $\mathcal{M}_{WL}$, with 1 step ahead. Indeed, it suffices to note that $\mathcal{M}_{dGNN} \subseteq \mathcal{M}_{deg}$. In particular, the classes $\mathcal{M}_{dGNN_1}$–$\mathcal{M}_{dGNN_6}$ corresponding to the graph neural network architectures from Table 1 are all weaker than $\mathcal{M}_{WL}$, with 1 step ahead. Furthermore, in the proof of Proposition 13 we have shown that the condition that $\mathcal{M}_{WL}$ is 1 step ahead is necessary for $\mathcal{M}_{dGNN_4}$, and thus also for $\mathcal{M}_{dGNN}$. One can provide similar examples for $\mathcal{M}_{dGNN_2}$, $\mathcal{M}_{dGNN_5}$ and $\mathcal{M}_{dGNN_6}$.

In contrast, the two remaining classes, $\mathcal{M}_{dGNN_1}$ and $\mathcal{M}_{dGNN_3}$, are weaker than $\mathcal{M}_{WL}$ (with no step ahead).

Proposition 16. *The classes $\mathcal{M}_{dGNN_1}$ and $\mathcal{M}_{dGNN_3}$ are weaker than $\mathcal{M}_{WL}$.*

The reason is that dMPNNs in these classes are equivalent to dMPNNs that only use degree information *after* neighbour aggregation, and are thus equivalent to anonymous MPNNs. *To boost the expressive power of aMPNNs, it is therefore important to use degree information before aggregating.*

To conclude, we investigate whether $\mathcal{M}_{dGNN}$ and its subclasses $\mathcal{M}_{dGNN_1}$–$\mathcal{M}_{dGNN_6}$ are stronger than $\mathcal{M}_{WL}$. For $\mathcal{M}_{dGNN}$ this follows from Theorem 7, stating in particular that $\mathcal{M}_{GNN} \equiv \mathcal{M}_{WL}$, and from the following remark.

Remark 17. It holds that $\mathcal{M}_{GNN} \preceq \mathcal{M}_{dGNN}$.

Indeed, we observe that $\mathcal{M}_{GNN}$ corresponds to the subclass of $\mathcal{M}_{dGNN}$ in which the functions g and h are the constant one function, i.e., $g(n) = h(n) = 1$ for all $n \in \mathbb{N}^+$, and moreover, $p = 0$. More precisely, for every MPNN M in $\mathcal{M}_{GNN}$ there is an MPNN M' in $\mathcal{M}_{dGNN}$ such that $M \equiv M'$, from which Remark 17 follows.

So, we know already that $\mathcal{M}_{WL} \preceq \mathcal{M}_{dGNN}$. However, the aMPNN M in $\mathcal{M}_{GNN}$ such that $M_{WL} \preceq M$ holds, as constructed for Theorems 10 and 11, does not comply with the forms of MPNNs corresponding to the graph neural networks given in Table 1. We next investigate which classes $\mathcal{M}_{dGNN_i}$ are stronger that $\mathcal{M}_{WL}$.

We start with some negative results, hereby showing part of the fourth item in Theorem 12.

Proposition 18. *None of the classes $\mathcal{M}_{dGNN_i}$, for $i \in \{1, 2, \ldots, 5\}$, are stronger than $\mathcal{M}_{WL}$.*

The proof consists of a number of counterexamples. In particular, the class $\mathcal{M}_{dGNN_4}$, corresponding to GCNs (Kipf & Welling, 2017), is not stronger than $\mathcal{M}_{WL}$. Indeed, we can show that there are no $\mathbf{W}^{(1)}, \mathbf{B}^{(1)}$ such that $\mathbf{L}^{(1)} \sqsubseteq \boldsymbol{\ell}^{(1)}_{M_{WL}}$ for the labelled graph $(G_2, \boldsymbol{\nu})$ with one edge $\{v_1, v_2\}$ and vertex labelling $\boldsymbol{\nu}_{v_1} = (1, 0)$ and $\boldsymbol{\nu}_{v_2} = (0, 1)$. As such, *"plain vanilla" GCNs fall short in simulating the WL algorithm.* It will follow from our next result that a slight extension suffices to remedy this shortcoming.

Proposition 19. *The class $\mathcal{M}_{dGNN_6}$ is stronger than $\mathcal{M}_{WL}$.*

The claim actually holds for any degree-determined $\mathbf{g}$ and $\mathbf{h}$. Our proof closely follows that of Theorem 11. More specifically, we construct a dMPNN M corresponding to (5) such that $\boldsymbol{\ell}^{(t)}_{M} \sqsubseteq \boldsymbol{\ell}^{(t)}_{M_{WL}}$ for all $t \geq 0$. The crux of the argument is that one can choose $0 < p < 1$ close enough to 1 so that — loosely speaking — the initial label is not obliterated.[6]

[6]The parameter p serves a similar purpose to the epsilon term

We already mentioned that the result holds for any degree-determined $\mathbf{g}$ and $\mathbf{h}$. Particularly, the class of dMPNNs originating from graph neural networks of the form

$$\sigma\left((\mathbf{D}+\mathbf{I})^{-1/2}(\mathbf{A}+p\mathbf{I})(\mathbf{D}+\mathbf{I})^{-1/2}\mathbf{L}^{(t-1)}\mathbf{W}^{(t)}-q\mathbf{J}\right),$$

with $p, q \in \mathbb{A}$, $0 \leq p, q \leq 1$, is stronger than $\mathcal{M}_{WL}$. The *introduction of the parameter* p was already suggested in Kipf & Welling (2017). Proposition 13 shows that it is *necessary to encode the WL algorithm*. Our result thus provides a theoretical justification for including this parameter.

7. Conclusion and Future Work

We investigated the distinguishing power of two classes of MPNNs, anonymous and degree-aware MPNNs, and shown that both classes are equivalent to the WL algorithm when one ignores the number of computation rounds. Taking the computation rounds into consideration, however, reveals that degree information may boost the distinguishing power.

Other activation functions. The class $\mathcal{M}_{GNN}$ considers only ReLU and sign as the activation function. For future work one could consider other nonlinear activation functions, such as softplus or sigmoid. The challenge in extending the constructions from this paper or those in Morris et al. (2019) lies in ensuring that after applying the activation function one gets a non-singular matrix by transforming the input matrix. For ReLU and sign, we can ensure that an upper-triangular matrix with non-zero diagonal elements can be obtained. This requires mapping lower diagonal elements to 0. For smooth functions softplus or sigmod, 0 is obtained as a limit value only, which prevents this technique from working.

Beyond degree-aware MPNNs. There are other models in the literature that are not captured by our general degree-aware MPNNs and which are worth exploring in the MPNN framework. For example, Graph Attention Networks (Velickovic et al., 2018) require the message functions to depend on attention weights which likely results in an increase of expressive power (see also Sato (2020)). More generally, one can envisage MPNNs in which message functions may differ in the parameters that they can use, resulting in different expressive power.

Acknowledgements

We thank the ICML referees for their detailed and helpful comments that improved this paper. This work was supported by the Belgian FWO "SAILor" project (G030020N).

in GINs Xu et al. (2019): to ensure injectivity. Since Xu et al. (2019) work over rational numbers in their proofs it suffices to choose irrational epsilons. By contrast, our result does not require such an approximation, and explicit weight matrices and parameter ranges for p are given that suffice to simulate the WL algorithm.

References

Arvind, V., Köbler, J., Rattan, G., and Verbitsky, O. Graph isomorphism, color refinement, and compactness. *Computational Complexity*, 26(3):627–685, 2017.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017*, pp. 1263–1272, 2017.

Grohe, M. *Descriptive Complexity, Canonisation, and Definable Graph Structure Theory*. Lecture Notes in Logic. Cambridge University Press, 2017.

Grohe, M. word2vec, node2vec, graph2vec, x2vec: Towards a theory of vector embeddings of structured data. In *Proceedings of the 39th Symposium on Principles of Database Systems*, pp. 1–16, 2020.

Hamilton, W. L., Ying, Z., and Leskovec, J. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems 30, NeurIPS 2017*, pp. 1024–1034, 2017.

Immerman, N. and Lander, E. Describing graphs: A first-order approach to graph canonization. In *Complexity Theory Retrospective: In Honor of Juris Hartmanis on the Occasion of His Sixtieth Birthday*, pp. 59–81. Springer, 1990.

Jarvis, F. *Algebraic number theory*. Undergraduate Mathematics Series. Springer, 2014.

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. In *Proceedings of the 5th International Conference on Learning Representations*, 2017.

Li, Q., Han, Z., and Wu, X. Deeper insights into graph convolutional networks for semi-supervised learning. In *Proceedings of the 30th AAAI Conference on Artificial Intelligence*, pp. 3538–3545, 2018.

Meltzer, P., Mallea, M. D. G., and Bentley, P. J. Pinet: A permutation invariant graph neural network for graph classification. *CoRR*, abs/1905.03046, 2019.

Morris, C., Ritzert, M., Fey, M., Hamilton, W. L., Lenssen, J. E., Rattan, G., and Grohe, M. Weisfeiler and Leman go neural: Higher-order graph neural networks. In *Proceedings of The 33rd AAAI Conference on Artificial Intelligence*, pp. 4602–4609, 2019.

Oono, K. and Suzuki, T. Graph neural networks exponentially lose expressive power for node classification.

In *Proceedings of the 8th International Conference on Learning Representations*, 2020.

Sato, R. A survey on the expressive power of graph neural networks. *ArXiv*, abs/2003.04078, 2020.

Velickovic, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., and Bengio, Y. Graph attention networks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*, 2018.

Weisfeiler, B. J. and Lehman, A. A. A reduction of a graph to a canonical form and an algebra arising during this reduction. *Nauchno-Technicheskaya Informatsiya*, 2(9): 12–16, 1968.

Wu, F., Jr., A. H. S., Zhang, T., Fifty, C., Yu, T., and Weinberger, K. Q. Simplifying graph convolutional networks. In *Proceedings of the 36th International Conference on Machine Learning*, pp. 6861–6871, 2019a.

Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., and Yu, P. S. A comprehensive survey on graph neural networks. *CoRR*, abs/1901.00596, 2019b.

Xu, K., Hu, W., Leskovec, J., and Jegelka, S. How powerful are graph neural networks? In *Proceedings of the 7th International Conference on Learning Representations*, 2019.

Zaheer, M., Kottur, S., Ravanbakhsh, S., Póczos, B., Salakhutdinov, R., and Smola, A. J. Deep sets. In *Advances in Neural Information Processing Systems 30*, pp. 3391–3401, 2017.

Zhou, J., Cui, G., Zhang, Z., Yang, C., Liu, Z., and Sun, M. Graph neural networks: A review of methods and applications. *CoRR*, abs/1812.08434, 2018.