A Theoretical and Empirical Investigation into the Equivalence of Graph Neural Networks and the Weisfeiler-Leman Algorithm

From the faculty of Mathematics, Physics, and Computer Science for the purpose of obtaining the academic degree of Bachelor of Sciences.

Eric Tillmann Bill

Supervision:
Prof. Dr. rer. nat. Christopher Morris

Informatik 6
RWTH Aachen University

Contents

1 Introduction

Graphs are ubiquitous in various fields of life. Despite not always being explicitly identified as such, the graph data model's flexibility and simplicity make it an effective tool for modeling a diverse range of data. This includes unexpected instances, such as modeling text or images as a graph, as well as more complex instances like chemical molecules, citation networks, or connectivity encodings of the world wide web (Morris et al. [2020], Scarselli et al. [2009]). Although machine learning has achieved remarkable success with images and text in the last decade, the lack of a significant breakthrough in machine learning for graphs can be attributed to the model's inherent flexibility and simplicity. While nodes in simple applications may be organized sequentially, as in textfiles, or in a grid-like fashion, as in images, information in graphs can be arranged in more complex ways such as trees, acyclic graphs, or cyclic graphs. This complexity presents a significant challenge in developing a general machine-learning framework capable of accommodating all forms of graph inputs.

In recent years, there has been a significant amount of research conducted to investigate Graph Neural Networks (short GNNs). Among the most promising approaches are those utilizing the message-passing architecture, which was introduced by Gilmer et al. [2017] and Scarselli et al. [2009]. Empirically, this framework has demonstrated strong performance across various applications (Kipf and Welling [2017], Hamilton et al. [2017], Xu et al. [2018]). However, its expressiveness is limited, as has been proved by the works of Morris et al. [2018], as well as Xu et al. [2018]. These works establish a connection to the commonly used Weisfeiler-Leman[1] algorithm (short 1-WL), originally proposed by Weisfeiler and Leman [1968] as a simple heuristic for the graph isomorphism problem. In particular, it has been proven that a GNN based on the message-passing architecture can at most be as good as the 1-WL algorithm in distinguishing graphs. Furthermore, the 1-WL method demonstrates numerous similarities with the fundamental workings of the GNN architecture. It is therefore commonly believed that both methods are to some extent equivalent in their capacity.

In this work, we introduce a novel framework, which we coined "1-WL+NN," which involves applying the 1-WL algorithm to an input graph, followed by further processing of the resulting information using a feed-forward neural network. Thereby, we obtain a trainable framework that is suited for all kinds of graph-related tasks, such as graph classification, node regression, and more.

In this thesis, we will conduct an extensive analysis of the newly proposed framework "1-WL+NN" and its connection to GNNs, to be able to establish an understanding of what structural information GNNs can learn and how they leverage this information. For this, we divide this work into two main parts.

Firstly, we will establish the equivalence of both frameworks in terms of their expressiveness. Specifically, we want to show that for every function computed by a GNN, there exists a suitable model based on the "1-WL+NN" framework with optimal parameters that compute the same function.

Secondly, we will undertake an empirical analysis of the performance of our proposed framework, by testing various configurations and comparing them to state-of-the-art GNN models, that are based on the message-passing architecture. With these results, we will start a deeper analysis of how GNNs learn graph representations. In particular, if GNNs can learn and

[1]Based on https://www.iti.zcu.cz/wl2018/pdf/leman.pdf, we will use the spelling "Leman" here, as requested by A. Leman, the co-inventor of the algorithm.

detect relevant graph structures better or are similar in detecting them.

Letzter Satz so wäääähh

2 Related Work

2.1 Graph Neural Networks

In recent years, machine learning models have experienced a surge in popularity due to their significant performance advantages over conventional methods and their ability to autonomously adapt to their tasks. However, a closer examination of the applications where these models are been used typically, reveals that they are highly specialized for the specific input of each application. For instance, modern convolutional neural networks (short CNNs) are designed to take in fixed-sized, grid-like data structures such as images, while modern language models process sequential data like textfiles.

The relevance of graphs to these examples lies in the fact that graphs can be used to model various types of inputs across many applications, and they provide a more general framework for modeling data. To illustrate, an image can be modeled as a graph for a CNN, where each pixel corresponds to a node in the graph holding the brightness value for each color value, and each node is connected to its neighboring pixels. Similarly, for sequential data like textfiles, one can encode a directed graph where each word in this file is represented as a node with the word as a feature, and it is connected outgoingly to the next following word. With these examples, we wanted to highlight the flexibility of how graphs can model data, however, this is also problematic, as this makes it particularly hard to construct a general machine-learning model on graphs. Levering any constraints on the format or size of the input can significantly limit the model's generality, and since graphs sizes and formats can vary within their applications, e.g. classification of molecules (PubChem Morris et al. [2020]), the need for a general model is of great interest.

From the work of Gilmer et al. [2017], as well as Scarselli et al. [2009], the so-called message-passing architecture emerged for Graph Neural Networks. This can be understood as a framework that never changes its input graph structurally and only modifies the node's features in each layer. In more detail, in each layer, a GNN based on the message-passing architecture, computes for each node a new feature, based on its current feature and the features of its neighbors. Later in section 3.5, we will give a more formal definition of this architecture. Throughout this thesis, we will use the term GNN and message-passing architecture interchangeably.

2.2 Weisfeiler and Leman Algorithm

The (1-dimensional) Weisfeiler-Leman algorithm (short 1-WL), proposed by Weisfeiler and Leman [1968], was initially designed as a simple heuristic for the *graph isomorphism problem*, but due to its interesting properties, its simplicity, and its good performance, the 1-WL algorithm gained a lot of attention from researchers across many fields. One of the most noticeable of these properties is, that the algorithm color codes the nodes of the input graph in such a way, that in each iteration, each color encodes a learned local substructure.

It works by coloring all nodes in each iteration the same color that fulfills two properties: 1. the nodes already share the same color and 2. the frequencies of the colors of their neighbors are equal. The algorithm continues as long as the number of colors changes in each iteration. For determining whether two graphs are non-isomorphic, the heuristic applies the algorithm to both graphs simultaneously and concludes that the graphs are non-isomorphic as soon as the number of occurrences of a color is different between the graphs. We present a more formal definition of the algorithm later in the section 3.3.

Since the *graph isomorphism problem* is difficult to solve due to it being NP-*complete*, the 1-WL algorithm, running in polynomial deterministic time, cannot solve the problem completely. Moreover, Cai et al. [1992] constructed counterexamples of non-isomorphic graphs that the heuristic fails to distinguish, e.g. figure 1. However, following the work of Babai and Kucera [1979], this simple heuristic is still quite powerful and has a very low probability of failing to distinguish non-isomorphic graphs when both graphs are uniformly chosen at random.

To overcome the limited expressiveness of the 1-WL algorithm, it was generalized to the k-dimensional Weisfeiler-Leman algorithm (short k-WL), which works with k-tuples over the set of nodes. Interestingly, this created a hierarchy for the expressiveness of determining non-isomorphism, such that for all $k \in \mathbb{N}$ there exists a pair of non-isomorphic graphs that can be distinguished by the $(k+1)$-WL but not by the k-WL (Cai et al. [1992]).

2.3 Connections between GNNs and the WL algorithm

A connection between GNNs based on the message-passing architecture and the 1-WL algorithm seems quite natural since both share similar properties in terms of how they process graph data. Most noticeably, both methods never change the graph structurally, since they only compute new node features in each iteration. And additionally, both methods use a 1-hop neighborhood aggregation as their basis for the computation of the new node feature. Following this intuition of both methods being very similar, many authors showed a theoretical connection between these methods. Morris et al. [2018], as well as Xu et al. [2018], showed that GNN's expressiveness power is upper bounded by the 1-WL in terms of distinguishing non-isomorphic graphs. In addition, Morris et al. [2018] also proposed a new k-GNN architecture that works over the set of subgraphs of size k. Interestingly, the authors showed that the proposed hierarchy over $k \in \mathbb{N}$ is equivalent to the k-WL hierarchy in terms of their expressive in distinguishing non-isomorphic graphs, meaning if there exists a k-GNN that can distinguish two non-isomorphic graphs than it is equivalent to say that the k-WL algorithm can distinguish these graphs as well.

3 Preliminaries

We first introduce a couple of notations and definitions that will be used throughout the thesis. With $[n]$, we denote the set $\{1, \ldots, n\} \subset \mathbb{N}$ for any $n \in \mathbb{N}$ and with $\{\!\{\ldots\}\!\}$ we denote a multiset which is formally defined as a 2 tuple (X, m) with X being a set of all unique elements and $m : X \to \mathbb{N}_{\geq 1}$ a mapping that maps every element in X to its number of occurrences in the multiset.

3.1 Graph Framework

A graph is denoted by G and is a 3 tuple $G := (V, E, l)$ that consists of the set of all nodes V, the set of all edges $E \subseteq V \times V$ and a label function $l : M \to \Sigma$ with M being either $V, V \cup E$ or E and $\Sigma \subset \mathbb{N}$ a finite alphabet. Moreover, let $\mathcal{G}$ be the set of all finite graphs. Note, that our definition of the label function allows for graphs with labels either only on the nodes, only on the edges, or on both nodes and edges. Sometimes the values assigned by l are called features, but this is usually only the case when Σ is multidimensional, which we do not cover in this thesis. In addition, although we have defined it this way, the labeling function is optional, and in cases where no labeling function is given, we add the trivial labeling function $f_1 : V(G) \to \{1\}$. Further, G can be either directed or undirected, depending on the definition of E, where $E \subseteq \{(v, u) \mid v, u \in V\}$ defines a directed and $E \subseteq \{(v, u), (u, v) \mid v, u \in V, v \neq u\}$ defines an undirected graph. Additionally, we will use the notation $V(G)$ and $E(G)$ to denote the set of nodes of G and the set of edges of G respectively, as well as l_G to denote the label function of G. With $\mathcal{N}(v)$ for $v \in V(G)$ we denote the set of neighbors of v with $\mathcal{N}(v) := \{u \mid (u, v) \in E(G)\}$.

A coloring of a Graph G is a function $C : V(G) \to \mathbb{N}$ that assigns each node in the graph a color (here a positive integer). Further, a coloring C induces a partition P on the set of nodes, for which we define C^{-1} being the function that maps each color $c \in \mathbb{N}$ to its class of nodes with $C^{-1}(c) = \{v \in V(G) \mid C(v) = c\}$. In addition, we let $h_{G,C} = \{\!\{C(v) \mid v \in V(G)\}\!\}$ be the histogram of graph G with coloring C, that contains for every color in the image of $V(G)$ under C the color and its frequency.

3.2 Permutation-invariance and -equivariance

We use S_n to denote the symmetric group over the elements $[n]$ for any $n > 0$. S_n consists of all permutations over these elements. Let G be a graph with $V(G) = [n]$, applying a permutation $\pi \in S_n$ on G, is defined as $G_\pi := \pi \cdot G$ where $V(G_\pi) = \{\pi(1), \ldots, \pi(n)\}$ and $E(G_\pi) = \{(\pi(v), \pi(u)) \mid (v, u) \in E(G)\}$. We will now introduce two key concepts for classifying functions on graphs. Let $f : \mathcal{G} \to \mathcal{X}$ be an arbitrary function and let $V(G) = [n]$ for some $n \in \mathbb{N}$:

1. The function f is *permutation-invariant* if and only if for all $G \in \mathcal{G}$ where $n_G :=\mid V(G) \mid$ and for every $\pi \in S_{n_G}$: $f(G) = f(\pi \cdot G)$.
2. The function f is *permuation-equivariant* if and only if for all $G \in \mathcal{G}$ where $n_G :=\mid V(G) \mid$ and for every $\pi \in S_{n_G}$: $f(G) = \pi^{-1} \cdot f(\pi \cdot G)$

3.3 Weisfeiler and Leman Algorithm

The Weisfeiler-Leman algorithm consists of two main parts, first the coloring algorithm and second the graph isomorphism test. We will introduce them in this section.

The Weisfeiler-Leman graph coloring algorithm

Let $G = (V, E, l)$ be a graph, then in each iteration i, the 1-WL computes a node coloring $C_i : V(G) \to \mathbb{N}$, which depends on the coloring of the neighbors and the node itself. In iteration $i = 0$, the initial coloring is $C_0 = l$ or if l is non existing $C_0 = c$ for an arbitrary constant $c \in \mathbb{N}$. For $i > 0$, the algorithm assigns a color to $v \in V(G)$ as follows:

$$C_i(v) = \mathsf{RELABEL}((C_{i-1}(v), \{\!\{C_{i-1}(u) \mid u \in \mathcal{N}(v)\}\!\})) \tag{0.1}$$

Where RELABEL injectively maps the above pair to a unique, previously not used, natural number. Although this is not a formal restriction by the inventors, we further require the function to always map to the next minimal natural number. Thereby we can contain the size of the codomain of each coloring for all iterations. The algorithm terminates when the number of colors between two iterations does not change, meaning the algorithm terminates after iteration i if the following condition is satisfied:

$$\forall v, w \in V(G) : C_i(v) = C_i(w) \iff C_{i+1}(v) = C_{i+1}(w) \tag{0.2}$$

Upon terminating we define $C_\infty := C_i$ as the stable coloring. For an illustration of this coloring algorithm, see figure 2. The algorithm always terminates after $n_G := |V(G)|$ iterations (Grohe [2017]). Moreover, based on the work of Paige and Tarjan [1987] about efficient refinement strategies, Cardon and Crochemore [1982] proved that the stable coloring C_∞ can be computed in time $\mathcal{O}(|V(G)| + |E(G)| \cdot log|V(G)|)$.

The Weisfeiler-Leman Graph Isomorphism Test

To determine if two graphs $G, H \in \mathcal{G}$ are non-isomorphic (short $G \not\cong H$), one applies the 1-WL coloring algorithm on both graphs "in parallel" and checks after each iteration if the occurrences of each color are equal, else the algorithm would terminate and conclude non-isomorphic. Formally, the algorithm concludes non-isomorphic in iteration i if there exists a color c such that:

$$|\{v \in V(G) \mid c = C_i(v)\}| \neq |\{v \in V(H) \mid c = C_i(v)\}| \tag{0.3}$$

Note that this test is only sound and not complete for the problem of graph isomorphism. Counterexamples where the algorithm fails to distinguish non-isomorphic graphs can be easily constructed, see Figure 1 which was discovered and proven by Cai et al. [1992].

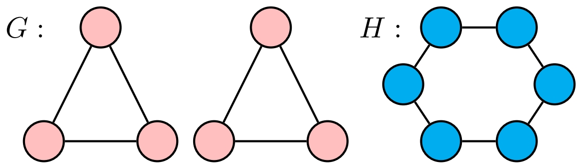

Figure 1: An example of two graphs G and H that are non-isomorphic but cannot be distinguished by the 1-WL

3.4 1-WL+NN Framework

Let $\mathcal{G}_{\leq n} \subset \mathcal{G}$ be the set of all labeled graphs G with n or fewer nodes. We define the class 1-WL+NN of functions over $\mathcal{X} \subseteq \mathcal{G}_{\leq n}$ as all functions that are comprised of an encoding function $F : \mathcal{X} \to R^C$, followed by a multilayer perceptron that takes vectors over R^C as input. In particular, the encoding function F maps the final coloring C^i_∞ of G computed by the 1-WL algorithm to a vector in R^C. Note that R can be an arbitrary domain, and $C \in \mathbb{N}$ is a constant.

As an example of a class following this definition, we present the bounded 1-WL+NN classes that use the *counting-encoding* function and are parametrized by $n, k \in \mathbb{N}$:

For $n, k \in \mathbb{N}$, let $\mathcal{X} = \{G \in \mathcal{G}_{\leq n} \mid \forall x \in V(G) \cup E(G) : l_G(x) \leq k\}$ be the set of all graphs with at most n nodes and their label functions being bounded by k. We define the *counting-encoding* Function $F : \mathcal{X} \to \mathbb{N}^C$ as the function that maps a graph G to a vector $v \in \mathbb{N}^C$ such that the i.th component of v is equal to the occurrence of the color i in the final coloring computed by the 1-WL algorithm when applied to G. More formally, for $G \in \mathcal{X}$ let C^i_∞ be the final coloring upon the termination of the 1-WL algorithm on G and h_G the respective color histogram. Than F maps G to a vector $v \in \mathbb{N}^C$, such that for all $i \in [C] : v_i = h_G(i)$. Important to note that due to the boundedness of the size of graphs by n and its labels by k, there exists a minimal C for the codomain R^C, such that F is well defined on all graphs $G \in \mathcal{X}$. Throughout this thesis, we will always assume that C is set to the minimal value.

The bounded 1-WL+NN class that uses the *counting-encoding* function and is parametrized by n and k, are a collection of functions of the format $f : \mathcal{X} \to \mathcal{Y}, G \mapsto \text{MLP} \circ F(G)$, where

$\mathcal{Y}$ is a task-specific output set (e.g. labels of a classification task) and MLP is an arbitrary multilayer perceptron that takes in as input, vectors over $\mathbb{N}^C$. We will refer to this collection from now on by $\mathcal{C}_{n,k}$.

To illustrate how this encoding function works and why we coined it *counting-encoding*, we will quickly introduce an example graph G. In figure 2, we give a visual representation of G and its stable coloring after applying the 1-WL algorithm to it. The *counting-encoding* function F counts through all colors $i \in [C]$ and sets each i.th component to the number of occurrences in the final coloring. Therefore the respective color histogram $h_G = \{\!\{2, 2, 3, 4\}\!\}$ of G is being mapped to $v \in \mathbb{N}^C$ with $v = (0, 2, 1, 1, 0, \ldots, 0)^T$.

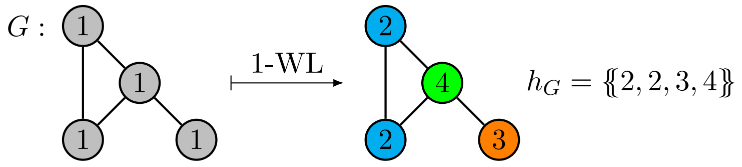

Figure 2: An example of the final coloring computed by applying the 1-WL algorithm on the graph G. The graph G consists of 4 nodes with all their labels being initially 1. Note, that the node label is the coloring of the graph.

3.5 Graph Neural Networks (Message Passing)

Let $G = (V, E, l)$ be an arbitrary graph. A Graph Neural Network (GNN) is a composition of multiple layers where each layer t passes a vector representation of each node v or edge e through $f^{(t)}(v)$ or $f^{(t)}(e)$ respectively and retrieves thereby a new graph that is structurally identical but has new feature information. Note that in the following we will restrict the definition to only consider node features, however, one can easily extend it to also include edge features.

To begin with, we need a function $f^{(0)} : V(G) \to \mathbb{R}^{1\times d}$ that is consistent with l, that translates all labels into a vector representation. Further, for every $t > 0$, f is of the format:

$$f^{(t)}(v) = f_{merge}^{W_{1,t}}(f^{(t-1)}(v),\ f_{agg}^{W_{2,t}}(\{\!\{f^{(t-1)}(w) \mid w \in \mathcal{N}(v)\}\!\})) \tag{0.4}$$

Where $f_{merge}^{W_{1,t}}$ and $f_{agg}^{W_{2,t}}$ are arbitrary differentiable functions with $W_{1,t}$ and $W_{2,t}$ their respective parameters. Additionally, $f_{agg}^{W_{2,t}}$ has to be permuation-invariant. To demonstrate what kind of functions are typically used, we provide the functions used by Hamilton et al. [2017]:

$$f_{merge}^{W_{1,t}}(v) = \sigma(W_{merge} \times \mathsf{concat}(f^{(t-1)}(v),\ f_{agg}^{W_{2,t}}(v))) \tag{0.5}$$

$$f_{agg}^{W_{2,t}}(v) = \max(\{\sigma(W_{pool} \times f^{(t-1)}(u) + b \mid u \in \mathcal{N}(v))\}) \tag{0.6}$$

Where σ is a non-linear elementwise activation function; W_{merge}, W_{pool} are trainable matrices, b a trainable vector and concat the concatenation function.

Depending on the objective, whether the GNN is tasked with a graph or only a node or edge task, the last layer differs. In the case of graph tasks, we add a permutation-invariant aggregation function to the end, here called READOUT, that aggregates over every node and computes a fixed-size output vector for the entire graph, e.g. a label for graph classification. In order to ensure that we can train the GNN in an end-to-end fashion, we require READOUT to be also differentiable.

Let $\mathcal{A}$ be an instance of the described GNN framework. Further, let $K \in \mathbb{N}$ be the number of layers of the GNN, $\mathcal{G}$ the set of all graphs, $\mathcal{Y}$ the task-specific output set (e.g. labels of a classification task), then the overall function computed by $\mathcal{A}$ is:

$$\mathcal{A} : \mathcal{G} \to \mathcal{Y} : x \mapsto f^{(K)} \circ \ldots \circ f^{(0)}(x) \tag{0.7}$$

$$\mathcal{A} : \mathcal{G} \to \mathcal{Y} : x \mapsto \mathsf{READOUT} \circ f^{(K)} \circ \ldots \circ f^{(0)}(x) \tag{0.8}$$

As we require all aggregation functions to be permutation-invariant, the total composition $\mathcal{A}$ is permutation-invariant, and with similar reasoning, it is also differentiable. This enables us to train $\mathcal{A}$ like any other machine learning method in an end-to-end fashion, regardless of the underlying encoding used for graphs. This definition and use of notation are inspired by Morris et al. [2018] and Xu et al. [2018].

4 Main Part

In this section, we will discuss the topic of this thesis. The thesis will mainly establish what makes GNNs empirically so good, by trying to analyze if GNNs only argue about learned graph structures that can be also learned 1-WL algorithm or more. We will argue via a connection between GNNs and 1-WL+NN. The work is divided into first, a theoretical part, where we prove the equivalence between GNNs and 1-WL+NN about their expressive. The second part will then be based on the results of the first part, an empirical analysis of the performance of GNNs and 1-WL+NN in different configurations that lay the basis for further investigations what makes GNNs so special. We will now introduce each part individually.

What makes GNNs so special ist so mhmm :3

4.1 Part I: Theoretical Proof of the Equivalence

In this part, we will start by introducing the framework we coined 1-WL+NN, and demonstrate how a model of this framework can be used and trained in an end-to-end fashion for graph tasks. More importantly, we will build a connection to GNNs by trying to prove the following hypothesis:

> *For every function $\mathcal{A}$ computed by a GNN (definition in section 3.5), there exists a 1-WL+NN model (definition in section 3.4) computing $\mathcal{A}$ as well.*

After proving this hypothesis formally, we can conclude that GNNs and 1-WL+NN models share the same capacity.

To date, there is no research that we are aware of that examines the relationship between these two frameworks. Nonetheless, several works, as outlined in 2.3, offer results that hint that this hypothesis may be valid. Additionally, in the research conducted by Zopf [2022], the author empirically demonstrated how well the 1-WL test is in distinguishing non-isomorphic graphs across various datasets and used these results to give an upper bound on the actual classification task of the datasets when training an individual GNN on each dataset. This demonstrates, how the expressiveness of a graph algorithm on an arbitrary task is somehow limited by its capacity in distinguishing non-isomorphism.

Our approach for showing the validness of the hypothesis is, that the proving direction of "1-WL+NN $\subseteq$ GNN" is quite trivial, as we can easily encode the 1-WL coloring algorithm in each GNN layer. The other direction, however, showing "GNN $\subseteq$ 1-WL+NN", is more challenging, for which we want to take inspiration from the proof presented in section 3.3 by Xu et al. [2018]. In more detail, we will try to prove that for every $n \in \mathbb{N}$ and arbitrary k, the class $\mathcal{C}_{n,k}$ (bounded class of 1-WL+NN using the *counting-encoding* function) is GNN-approximating. For this, we have to prove that there exists no other encoding function that is strictly more expressive than the *counting-encoding* function and we have to define what GNN-approximating means.

4.2 Part II: Empirical Analysis and Comparison

In this part, we will provide a comprehensive analysis of the performance of the 1-WL+NN framework by testing it on well-established benchmark datasets for GNNs. With this analysis we will try to answer the following questions:

Q1) Which encoding of the feature space for 1-WL+NN framework has the best performance in generalizing? Does this result align with the results of research in other fields?

Q2) Is there a difference in performance between both frameworks, 1-WL+NN and GNNs, in generalizing? And if so, which one generalizes better after fewer training iterations? Is there an explanation for this behavior?

Q3) Is one of the tasks better suited for a specific task setting? For example, is more suitable for graph classification? Why could this be, what is the fundamental difference between both frameworks leading to this result?

For an extensive analysis, we will work out different configurations of functions of the 1-WL+NN framework, that will be tested with different types of encoding functions. However, as the result and insights of part I will influence the choice of configurations and encoding functions, we do not feel comfortable determining them yet and therefore leave their choice open for now. For a baseline for GNNs, we will use three well-tested and state-of-the-art GNNs:

1. Graph Convolutional Network (short GCN) by Kipf and Welling [2017] with and without training
2. GraphSAGE developed by Hamilton et al. [2017] with and without training
3. Graph Isomorphism Network (short GIN) developed by Xu et al. [2018] with and without training

For running the test cases, we will implement the 1-WL+NN with all its different configurations in Python using the open source library PyTorch[2] and the open source extension PyTorch Geometric[3].

We will select the datasets to be used for benchmarking from the TU-Dataset, a curated collection of graph datasets that are highly suitable for training graph-based algorithms. This collection offers data from diverse applications of varying sizes, enabling us to thoroughly evaluate the performance of our frameworks on a wide range of inputs. This dataset is the result of extensive work by Morris et al. [2020].

[2]Open source machine learning framework that was originally developed by Meta AI and does now belong to the Linux Foundation umbrella. https://pytorch.org

[3]Open source library that acts as an extension to PyTorch and allows for easy writing and training of graph neural networks. https://pytorch-geometric.readthedocs.io

Bibliography

[1] L. Babai and L. Kucera. Canonical labelling of graphs in linear average time. In *Symposium on Foundations of Computer Science*, pages 39–46, 1979. 5

[2] J.-Y. Cai, M. Fürer, and N. Immerman. An optimal lower bound on the number of variables for graph identification. *Combinatorica*, 12(4):389–410, Dec 1992. ISSN 1439-6912. doi: 10.1007/BF01305232. URL `https://doi.org/10.1007/BF01305232`. 5, 7

[3] A. Cardon and M. Crochemore. Partitioning a graph in $O(|A| \log_2 |V|)$. *Theoretical Computer Science*, 19(1):85 – 98, 1982. 7

[4] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. Neural message passing for quantum chemistry. In *International Conference on Machine Learning*, 2017. 3, 4

[5] M. Grohe. *Descriptive complexity*, page 40–93. Lecture Notes in Logic. Cambridge University Press, 2017. doi: 10.1017/9781139028868.004. 7

[6] W. L. Hamilton, R. Ying, and J. Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pages 1025–1035, 2017. 3, 8, 10

[7] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017. 3, 10

[8] C. Morris, M. Ritzert, M. Fey, W. L. Hamilton, J. E. Lenssen, G. Rattan, and M. Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks, 2018. URL `https://arxiv.org/abs/1810.02244`. 3, 5, 9

[9] C. Morris, N. M. Kriege, F. Bause, K. Kersting, P. Mutzel, and M. Neumann. TUDataset: A collection of benchmark datasets for learning with graphs. *CoRR*, abs/2007.08663, 2020. 3, 4, 10

[10] R. Paige and R. Tarjan. Three partition refinement algorithms. *SIAM Journal on Computing*, 16(6):973–989, 1987. 7

[11] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009. 3, 4

[12] B. Weisfeiler and A. Leman. The reduction of a graph to canonical form and the algebra which appears therein. *Nauchno-Technicheskaya Informatsia*, 2(9):12–16, 1968. English translation by G. Ryabov is available at `https://www.iti.zcu.cz/wl2018/pdf/wl_paper_translation.pdf`. 3, 4

[13] K. Xu, W. Hu, J. Leskovec, and S. Jegelka. How powerful are graph neural networks?, 2018. URL `https://arxiv.org/abs/1810.00826`. 3, 5, 9, 10

[14] M. Zopf. 1-wl expressiveness is (almost) all you need, 2022. URL `https://arxiv.org/abs/2202.10156`. 9