# *On the theoretical equivalence of 1-WL+NN and GNNs and an empirical study on their performances*

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 Abstract

## 2 Introduction

- Teaser the reader why graphs are important, how neural networks in general have a significant impact on machine learning up to this day, and conclude that Graph Neural Networks are very interesting as they provide a trainable framework for classification on graphs, similar to ANNs or CNNS.
  Possible applications: Computer progamms, Semantic Web, Chemoinformatics, computer vision and graphics, graph matching, **?**.
- Shortly outline the difficulties graphs are introducing compared to data ANNs or CNNs are working on (like regression, images) and explain how crucial it is to stay general in the approach as we can make no assumptions about the structure of the graphs.
- Introduce Weisfeller and Lehman[1] and their 1-WL algorithm, by only mentioning how it recolors every node in every iteration based on each node's and neighbor's node's colors. Thereby effectively encoding information about the substructure each node of the graph is contained in.
- Teaser how GNNs in their most common architecture (message passing) somehow do the same and that between the 1-WL and GNNs are a couple of similarities, like for example that it has been shown that every GNN is at most as powerful in determining the Isomorphic Problem on graphs as the 1-WL algorithm.
- Introduce the main topic of the thesis, the first more theoretical part of this thesis will try that the 1-WL algorithm combined with Neural Networks are capable of computing every function that is computable by a GNN. Therefore, showing that 1-WL + NN are equivalent to GNN in their expressiveness. The second part will investigate emprically which encoding of the features space for 1-WL works best and more importantly how 1-WL + NN compared to GNNs are generalizing their task.

## 3 Related Work

### 3.1 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 task. However, a closer examination of the applications where these models have been used reveals that they are highly specialized for the specific input of each application. For

[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.

instance, modern convolutional neural networks are designed to take in fixed, grid-like data structures such as images, while modern language models process sequential data like text.

The relevance of graphs to these examples lies in the fact that graphs can be used to model all types of inputs in various 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, one can encode a directed graph where each word 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 models generality, and since graphs sizes and formats can vary within the application, e.g. chemical molecule classifyng, the need for a general model is of great interest.

From the work of **?**, as well **?**, the so called message-passing architecture emerged for Graph Neural Networks (short GNNs). 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 previous feature and the features of its neighbors. Later in section **??**, we will give a more formal definition of this architecture.

TODO: Although there have also been other frameworks proposed like the work of Tönshoff et al which uses random walks, or more powerful modifications like Li et al 2016 that uses a distance information, the message-passing architecture seems to be the most promising, as many researchers focus on it.

Throughout this thesis, I will use the term GNN and message-passing architecture interchangeably.

## 3.2 Weisfeiler and Leman Algorithm

The (1-dimensional) Weisfeiler-Leman algorithm (short 1-WL), proposed by **?**, was initially designed as a simple heuristic for the *graph isomorphism problem*, but due to its interesting properties and simplicity it is now used among other things as a graph kernel in GNNs, for example, **??**. One of these properties is, that the algorithm color codes the nodes of the input graph in such a way, that each color of a node encodes a local substructure the node is contained in, in each iteration.

It works by coloring all nodes in each iteration equally that already are colored the same and the frequencies of the colors of their neighbors are equal. The algorithm continues as long as the number of colors changes until the coloring is stable. In determining whether two graphs are non-isomorphic, the heuristic would apply the algorithm to both graphs simultaneously and conclude 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 **??**.

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, **?** have constructed counterexamples for non-isomorphic graphs that the heuristic fails to distinguish, e.g. figure **??**. However, following the work of **?**, 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 instead 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 (**?**).

### 3.3 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 in more detail, both methods use a 1-hop neighborhood aggregation as their basis for the computation of the new node feature. Following this intuition, many authors showed a theoretical connection between these methods. **?**, as well as **?**, showed that GNN's expressiveness power is upper bounded by the 1-WL in terms of distinguishing non-isomorphic graphs. In Addition, **?** 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.

## 4 Preliminaries

We first introduce a couple of notations that will be used in this thesis. With $[n]$ we denote the set $\{1, \ldots, n\} \subset \mathbb{N}$ for any $n \in \mathbb{N}$ and for $\{\!\!\{ \ldots \}\!\!\}$ 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.

### 4.1 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 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. 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 $f_0 : V(G) \to \{0\}$. 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. With $\mathcal{N}(v)$ for $v \in V(G)$ we denote the set of neighbors of $v$ with $\mathcal{N}(v) := \{u \mid (u, v) \in E(G)\}$

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

TODO: Mention that those definitions are adapted and inspired by **?**.

## 4.3 Weisfeiler and Leman Algorithm

We work here with a combination of the original Weisfeiler and Leman Algorithm and the color refinement algorithm which we call in general 1-dimensional Weisfeiler and Leman algorithm (short 1-WL). TODO: check the statement and add citations.

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:

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

Where RELABEL injectively maps the above pair to a unique, previously not used, natural number. 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:

$$\forall v, w \in V(G) : C_i(v) = C_i(w) \iff C_{i+1}(v) = C_{i+1}(w) \qquad (0.1)$$

Upon terminating we define $C_\infty := C_i$ as the stable coloring. Note that the algorithm always terminates after $n_G := |V(G)|$ iterations as there are only $n_G$ different cardinalities of the image under any coloring. TODO: check the reasoning, I think $|C_i| <|C_{i-1}|$ but have no recollection of this.

To determine if two graphs $G, H \in \mathcal{G}$ are non-isomorphic (short $G \ncong H$), one would apply the 1-WL algorithm on both graphs "in parallel" and check 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: $|\{v \in V(G) \mid c = C_i(v)\}| \neq |\{v \in V(H) \mid c = C_i(v)\}|$. Note that this method 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 **??** which was discovered and proven by **?**.

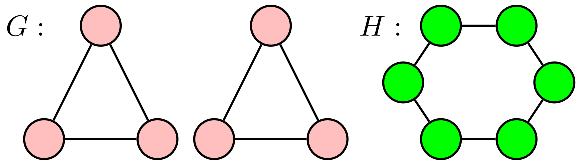

Figure 1: An example of two graphs $G$ and $H$ that are non-isomorphic but cannot be distinguished by the 1-WL

### 4.4 1-WL+NN Framework

Let $\mathcal{Y}$ be the task-specific output set (e.g. set of class labels), $\mathcal{NN}$ a feedforward neural network, enc an encoding function, $\pi_{pool}$ a pooling function. Further, for $G \in \mathcal{G}$, let $(C^i_\infty)_G$ be the final coloring upon termination when applying the 1-WL algorithm on $G$. Than the computed function $\mathcal{A}$ is:

$$\mathcal{A} : \mathcal{X} \rightarrow \Sigma,\ G \mapsto \mathcal{NN} \circ \pi_{pool}(\{\!\!\{(C^i_\infty)_G(v) \mid v \in V(G)\}\!\!\}) \tag{0.2}$$

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

$$f^{(t)}(v) = f^{W_{1,t}}_{merge}(f^{(t-1)}(v),\ f^{W_{2,t}}_{agg}(\{\!\!\{f^{(t-1)}(w) \mid w \in \mathcal{N}(v)\}\!\!\})) \tag{0.3}$$

Where $f^{W_{1,t}}_{merge}$ and $f^{W_{2,t}}_{agg}$ are arbitrary differentiable functions with $W_{1,t}$ and $W_{2,t}$ their respective parameters. Additionally, $f^{W_{2,t}}_{agg}$ has to be permuation-invariant. This definition has been adapted from **?**.

Depending on the objective, whether the GNN is tasked with node classification or graph classification, the last layer differs. In the latter case, we add a permutation-invariant aggregation function to the end, here called READOUT, that aggregates over every node and computes a single label for the entire graph (inspired by **?**). Note, 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:

$$\mathcal{A} : \mathcal{G} \rightarrow \mathcal{Y} : x \mapsto \circ f^{(K)} \circ \ldots \circ f^{(0)}(x) \tag{0.4}$$

$$\mathcal{A} : \mathcal{G} \rightarrow \mathcal{Y} : x \mapsto \mathsf{READOUT} \circ f^{(K)} \circ \ldots \circ f^{(0)}(x) \tag{0.5}$$

As we required all aggregation functions to be permutation-invariant, the total composition $\mathcal{A}$ is permutation-invariant, and similarly, 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.

## 5 Main Part

### 5.1 Theoretical Part

- Proof of the following statement: "Every function computed by a GNN, can also be computed 1-WL+NN. The GNN is defined as in section **??** and the 1-WL+NN as in section **??**."
- Idea for now is that one can connect the functions computable by 1-WL+NN to a class that is universally approximating. From this on I will show that this class is equivalent to all functions computable by a GNN.

## 5.2 Emprical / Practical Part

- Elaborate on the datasets I want to use? Why these, their related fields, and related work already conducted on them and why it is particularly good to use them
- As baseline use the implementation GCN(Kipf  welling 2017), GraphSAGE(Hamilton 2017) and GIN(Xu2018)

### Encoding

- For the different embedding schemas used for the 1-WL algorithm, I want to consider: 1. one-hot encoding 2. Look up Table 3. GNNs
- Explain that I will implement all these solutions by using the state-of-the-art implementations of Pytorch and the opensource extension for graphs Pytorch_geometric
- State a hypothesis that I think that the 2. or 3. option will have a significant difference in performance in comparison to option 1.
- State how I will evaluate these results. Mentioning the statistical tools I am going to use

### 1-WL vs GNNs

- Test all models, including all encoding variants of the previous section on the selected datasets.