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 **??**. 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 text files, 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. As of today, there are several solutions to this challenge, one of which we will examine in this thesis in detail.

In recent years, there has been a significant amount of research conducted to investigate Graph Neural Networks (GNNs). Among the most promising approaches are those utilizing the message-passing architecture, which was introduced by **?** and **?**. Empirically, this framework has demonstrated strong performance across various applications **???**. However, its expressiveness is limited, as has been proved by the works of **?**, as well as **?**. These works establish a connection to the commonly used Weisfeiler-Leman[1] algorithm (1-WL), originally proposed by **?** 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 non-isomorphic 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 to capture information in a graph.

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.

The main contribution of this thesis will be to establish a deeper understanding of the representations a GNN learns. Specifically, we aim to address the question of whether GNNs learned representations are only compromised of structural information of graphs that are identifiable by the 1-WL algorithm or if they encapsulate more intricate information. For this, we will conduct an extensive analysis of the newly proposed framework "1-WL+NN" and its connection to GNNs, specifically proving its equivalence to GNNs and using empirical results of an in-depth analysis as a foundation to answer the main question.

2 Related Work

In this section, we will briefly introduce the foundation of our research by explaining the origins of each method, mentioning recent important advances, and providing a brief overview of

[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.

connections between them.

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 (e.g. linear regression, support vector machines) and their ability to generalize to many problem statements. However, a closer examination of the applications where these models are being used typically, reveals that they are highly specialized for the specific input of each application. For instance, modern convolutional neural networks (CNNs) are designed to take in fixed-sized, grid-like data structures such as images, while modern language models process sequential data like text files.

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 text files, 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 (**?**), the need for a general model is of great interest.

From the work of **?**, as well as **?**, the so-called message-passing architecture emerged for GNNs, which are also sometimes referred to as message-passing neural networks (MPNNs). This architecture uses the input graph as its basis of computation by never changing the graph structurally and only modifying the node's features in each layer simultaneously. 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 using for example a feedforward neural network. Later in **??**, 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-Leman Algorithm

The (1-dimensional) Weisfeiler-Leman algorithm (1-WL), proposed by **?**, 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 count of each color of their direct neighbors is 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 is applied 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 **??**.

Since the *graph isomorphism problem* is difficult to solve due to the best known complete algorithm only running in deterministic quasipolynomial time (**?**), the 1-WL algorithm, running in polynomial deterministic time, cannot solve the problem completely. Moreover, **?** constructed counterexamples of non-isomorphic graphs that the heuristic fails to distinguish, e.g. see **??**. 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 as the number of nodes tends to infinity.

To overcome the limited expressiveness of the 1-WL algorithm, it has been generalized to the k-dimensional Weisfeiler-Leman algorithm (k-WL) by **??**, as well as **?**[2]. This version works with k-tuples over the k-ary Cartesian product of 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 (**?**).

2.3 Connections between GNNs and the 1-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. **?**, 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, **?**, as well as **?**, showed that the proposed hierarchy over $k \in \mathbb{N}$ is equivalent to the k-WL hierarchy in terms of their capacity 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 $\{1, \dots, n\} \subset \mathbb{N}$ for any $n \in \mathbb{N}$ and with $\{\!\{\dots\}\!\}$ 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 Graphs

A graph G 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

[2] In **?** on page 27, László Babai explains that he, together with Rudolf Mathon, first introduced this algorithm in 1979. He adds, that the work of **?** introduced this algorithm independently of him.

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 $E \subseteq \{(v, u), (u, v) \mid v, u \in V, v \neq u\}$ such that for every $(v, u) \in E$ also $(u, v) \in E$ 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 define $h_{G,C}$ as the histogram of graph G with coloring C, that maps every color in the image of C under $V(G)$ to the number of occurrences. In detail, $\forall c \in \mathbb{N} : h_{G,C}(c) := |\{v \in V(G) \mid C(v) = c\}| = |C^{-1}(c)|$

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 all $v \in V(G)$ 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. 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:

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

Upon terminating we define $C_\infty := C_i$ as the stable coloring. For an illustration of this coloring algorithm, see **??**. The algorithm always terminates after $n_G := |V(G)|$ iterations (**?**). Moreover, based on the work of **?** about efficient refinement strategies, **?** 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 ($G \ncong H$), 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:

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

Note that this test is only sound and not complete for the *graph isomorphism problem*. Counterexamples where the algorithm fails to distinguish non-isomorphic graphs can be easily constructed, see **??** which was discovered and proven by **?**.

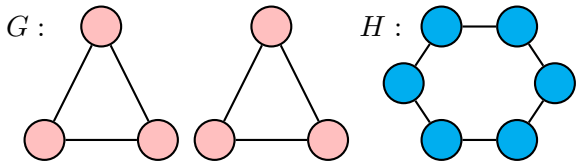

Figure 1: An example of two graphs G and H that are non-isomorphic but cannot be distinguished by the 1-WL.

3.4 1-WL+NN Framework

We define the class 1-WL+NN of functions over $\mathcal{X} \subseteq \mathcal{G}$ as all functions that are comprised of an encoding function $F : \mathcal{X} \to R^k$, followed by a multilayer perceptron that takes vectors over R^C as input. In particular, the encoding function F maps the final coloring C_∞ of a graph G computed by the 1-WL algorithm to a vector in R^K. Note that R can be an arbitrary domain, and $K \in \mathbb{N}_{\geq 1}$ is a constant. A function $\mathcal{B}$ of this class has the following format:

$$\mathcal{B} : \mathcal{X} \to \mathcal{Y},\ G \mapsto \mathrm{MLP} \circ F(G),$$

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 R^K.

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

For $k \in \mathbb{N}$, let $\mathcal{X} = \{G \in \mathcal{G} \mid \forall x \in V(G) \cup E(G) : l_G(x) \leq k\}$ the set of all graphs, where the label alphabet Σ of the respective label function l is bounded with $\Sigma \subseteq [k]$. We define the *counting-encoding* Function $F : \mathcal{X} \to \mathbb{N}^K$ as the function that maps a graph G to a vector

$v \in \mathbb{N}^K$ such that the c.th component of v is equal to the number of occurrences of the color c in the final coloring computed by the 1-WL algorithm when applied to G. More formally, for $G \in \mathcal{X}$ let C_∞ be the final coloring upon the termination of the 1-WL algorithm on G and h_{G,C_∞} the respective color histogram. Then F maps G to a vector $v \in \mathbb{N}^K$, such that for all $c \in [K] : v_c = h_{G,C_\infty}(c)$, where v_c denotes the c.th component of the vector v. Important to note, due to the bounded label alphabet Σ of all graphs $G \in \mathcal{X}$ by the parameter k, there exists a minimal K for the codomain $\mathbb{N}^K$ of F, such that F is well-defined on all graphs $G \in \mathcal{X}$.

The bounded 1-WL+NN class that uses the *counting-encoding* function and is parametrized k, is a special collection of functions which we will refer to as $\mathfrak{B}_k$.

To illustrate how this encoding function works and why we coined it *counting-encoding*, we will quickly introduce an example graph G. In **??**, 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 $i \in [K]$ and sets each i.th component to the number of occurrences in the final coloring. Therefore, the respective color histogram $h_{G,C_\infty} = \{\!\{2, 2, 3, 4\}\!\}$ of G is being mapped to $v \in \mathbb{N}^K$ with $v = (0, 2, 1, 1, 0, \ldots, 0)^T$, since color 2 appears two times, while color 3 and 4 occur only once. All other components of v are set to 0.

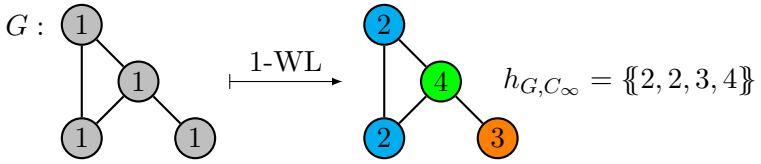

Figure 2: An example of the final coloring computed by applying the 1-WL algorithm on the graph G. The graph G consists of 4 nodes with all their labels being initially set to 1. Note that each label corresponds to a color, which we have also plotted for illustration purposes.

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 $t > 0$, $f^{(t)}$ is of the format:

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

where $f_{\text{merge}}^{W_{1,t}}$ and $f_{\text{agg}}^{W_{2,t}}$ are arbitrary differentiable functions with $W_{1,t}$ and $W_{2,t}$ their respective parameters. Additionally, $f_{\text{agg}}^{W_{2,t}}$ has to be permuation-invariant. To demonstrate what kind of functions are typically used, we provide the functions used by **?**:

$$f_{\text{merge}}^{W_{1,t}}(v) = \sigma(W_{\text{merge}} \cdot \mathsf{concat}(f^{(t-1)}(v),\ f_{\text{agg}}^{W_{2,t}}(v)))$$
$$f_{\text{agg}}^{W_{2,t}}(v) = \max(\{\sigma(W_{\text{pool}} \cdot f^{(t-1)}(u) + b) \mid u \in \mathcal{N}(v)\})$$

where σ is a non-linear element wise 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:

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

if $\mathcal{A}$ is configured for a graph task, otherwise:

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

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 **?** and **?**.

4 Main Part

In this section, we will discuss the topic of this thesis. The main contribution of this thesis will be to establish a deeper understanding of the representations a GNN learns. Specifically, we aim to address the question of whether GNNs learned representations are only compromised of structural information of graphs that are identifiable by the 1-WL algorithm or if they encapsulate more intricate information like potential biases. For this we will introduce a framework we coined 1-WL+NN , prove its equivalence to GNNs and use empirical results of an in-depth analysis as a foundation to answer the main question.

For this, we will divide the work into two parts, with the first being a theoretical part, where we prove the equivalence between GNNs and 1-WL+NN about their expressiveness. 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 into the representations GNNs learn. We will now introduce each part individually.

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 **??**), there exists a 1-WL+NN model (definition in **??**) approximating $\mathcal{A}$ arbitrarily well.*

More formally, we will prove that for every function $\mathcal{A}$ computed by a GNN, and for every $\epsilon \in \mathbb{R}$ with $\epsilon > 0$, there exists a 1-WL+NN model computing function $\mathcal{B}$, such that

$$\|\mathcal{A} - \mathcal{B}\|_\infty := \sup_{G \in \mathcal{G}} \|\mathcal{A}(G) - \mathcal{B}(G)\|_\infty < \epsilon. \tag{0.1}$$

After proving this hypothesis formally, we can conclude that GNNs and 1-WL+NN models are able to approximate the same functions, and are therefore in some sense equal.

To date, there is no research that we are aware of, that examines the relationship between these two or similar frameworks. Nonetheless, several works, as outlined in **??**, offer results that hint that this hypothesis may be valid. Additionally, in the research conducted by **?**, 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.

In order to prove this hypothesis, we will take inspiration from the proof presented in the work of **?** in section 3. First we, will introduce two properties a collection of functions can fulfill, which we will call *1-WL-Discriminating* and *GNN-Approximating*. A collection $\mathcal{C}$ is *1-WL-Discriminating* if for every graph $G_1, G_2 \in \mathcal{X}$ (here $\mathcal{X}$ is a constrained set of graphs we will specify later) that can be distinguished using the 1-WL isomorphism test, there exists a function in the collection $f \in \mathcal{C}$ such that $f(G_1) \neq f(G_2)$. Further, a collection $\mathcal{C}$ is *GNN-Approximating* if for every function $\mathcal{A}$ computed by a GNN model and every $\epsilon > 0$, there exists a function in the collection $f \in \mathcal{C}$ such that $\|\mathcal{A} - f\|_\infty < \epsilon$ (see **??**).

We will than show that the collection $\mathfrak{B}_k$ (bounded class of 1-WL+NN using the *counting-encoding* function) is *1-WL-Discriminating*. And, we will show that if a collection $\mathcal{C}$ is *1-WL-Discriminating*, than augmenting $\mathcal{C}$ by adding a carefully constructed multilayer perceptron to the end makes it *GNN-Approximating*. Combining both results together, gives us a proof of the overall hypothesis.

Note, that showing the reverse direction that every function $\mathcal{B}$ computed by a 1-WL+NN model, is also computable by a GNN model is quite trivial, as we can easily encode the 1-WL coloring algorithm in a GNN layer and choose a sufficiently large number of such layers for a model to compute $\mathcal{B}$ as well.

4.2 Part II: Empirical Analysis

In this part, we will first provide a comprehensive analysis of the performance of the 1-WL+NN framework in comparison to GNNs by testing it on well-established benchmark datasets of graph-related tasks. These results will lay the foundation for the direction we will take in the end, when investigating the main question.

In order to have confidence in our final results of this analysis. We will first try to find what optimal configurations of 1-WL+NN models provide the best empirical performance. We will therefore investigate typical machine learning related questions like, how prone our model is to overfit, or how fast it can generalize, but also two specific questions for the framework:

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

Q2) Does a reduction of the input size for the MLP by using encoding functions that compress the information yield better performance in generalizing faster?

For a good comparison, we will use three well-tested and state-of-the-art GNNs for our GNN baseline:

1. Graph Convolutional Network (GCN) by **?**
2. GraphSAGE developed by **?**
3. Graph Isomorphism Network (GIN) developed by **?**

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[3] and the open source extension PyTorch Geometric[4]. 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 **?**.

With this analysis, we will lead into the investigation of the actual question. When 1-WL+NN is provable equivalent to GNNs, a 1-WL+NN model acts as a sort of GNN that only uses structural information of the graph to compute a decision. With this in mind, the results of the analysis will give us two possible interpretations:

- If GNNs and 1-WL+NN show similar empirical performance, we assume that GNNs are only about learning local substructures in a graph and leverage only this information in their computation.
- If GNNs show empirically better performance, we would assume that GNNs learn somehow a better representation of their input graphs, which does not only contain structural information.

We assume, that GNNs in comparison will be more performant such that they somehow encode more information into their representations. In particular, we expect phenomena like an inductive bias toward local structures as a possible explanation. There are several hints towards this assumption, for instance, the work of **?** proves that GNNs inevitably suffer from an effect called over-squashing, meaning that the amount of information that can be exchanged between two nodes in a message-passing architecture decreases as the distance between both nodes increases. However, **?** argues that this effect and the resulting bias towards local substructures is one of the main reasons GNNs can achieve such good generalization, especially for problems that require relational reasoning.

If our empirical results support this assumption, we will further investigate what specific phenomena cause this difference in performance, for which we will develop more test cases.

As an example of how we will try to show the existence of a local bias to local structures in GNNs, we will create an artificial graph dataset where each graph is composed of one node marked as source, one node marked as target, and the rest marked as neutral. The task is to classify whether there exists a path between the source node and the target node. We will ensure that the minimum distance between these nodes is never greater than the total number of layers of the tested GNN model, so that we can ensure that information can be exchanged between the source and target nodes during the model's computation. The intuition here is that

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

[4]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

a 1-WL+NN model should achieve near-perfect performance because it uses only structural information as basis of its decision, whereas a GNN model suffers from the over-squashing effect, where the farther apart two nodes are, the less information is exchanged between them, making it challenging for the model to correctly classify if there exists a path between the source and target node.

We will develop more sophisticated methods and approaches as the work progresses, as we can only speculate on the outcome for now. In the end, we hope to achieve and present a thorough understanding of the representations learned by a GNN.