Graph Neural Networks and Weisfeiler-Lehman

Diogo Braga
University of Technology
Vienna, Austria
e12007525@student.tuwien.ac.at

Maximilian Thiessen
University of Technology
Vienna, Austria
maximilian.thiessen@tuwien.ac.at

Abstract

Real-world data requires a representation of their graph structure, such as molecules, social and financial networks. Therefore, learning with graph-structured data can allow advances related to graph analysis. Graph neural networks are a useful framework for learning graph representation and tackle problems like node classification, link prediction and graph classification. However, there are a few foundations related to the capabilities of these networks. A common problem is learning to represent and distinguish between graph structures. The most promising approach is related to the Weisfeiler-Lehman graph isomorphism test, a heuristic for deciding whether two graphs are isomorphic considering the graphs' properties. Therefore, in this report, we will present foundations of the Graph Isomorphism Problem and explore Graph Neural Networks, explaining them and referencing related approaches.

1 Introduction

Data represents events and entities worldwide and have gained more importance in general, whether in research or a business context. Many of these data have relationships between each other in a given collection and, in order to store this information, graphs are used as a representation structure. Molecules and social networks are some examples in which the application of this type of representation can be beneficial. A graph can represent a molecule, where the nodes represent the atoms and the edges represent covalent bonds. Based on this representation, it is possible, for example, to investigate combinations of atoms that fight an existing virus in society. A graph can also represent a social network, where nodes represent people and edges represent friendships or another type of connection between them. From this point, it is possible to recommend new connections both at a personal and professional level. These are two examples of what graphs can accomplish in the context of Machine Learning. In general, the main tasks are based on: node classification, where our goal is to determine the labelling of nodes using information from their neighbours; link prediction, where we try to predict the existence of a relationship between two nodes; and graph classification, where our goal is to make predictions in one graph by learning from other graphs.

Bearing in mind that this report focuses on representing and distinguishing graph structures, the starting point we will take here is to obtain information about the graph, which can be statistics at node and graph-level.

Regarding the node-level, it is possible to obtain: the degree of a node, in which the number of edges that affect the node is counted; the node centrality, which measures the importance of a node in a graph, taking into account, for example, the probability of a node being visited on a random walk; and the clustering coefficient, which measures the cluster size relative to the neighbours of a node. Although these features transmit useful information, graph-level features have more potential because they have a global view of the graph, as opposed to the node-level that is not as efficient for more demanding tasks.

For extracting graph-level features, *graph kernel methods* are widely used. These can be used, for example, to extract feature representations and find similarities between graphs using kernel functions [Gärtner et al., 2003]. Within these parameters, the Weisfeiler-Lehman (WL) kernel is one well-known approach where all the graph is taken into account. This algorithm's application is based on the WL test, where the basic idea of this heuristic, used to support several algorithms, is the count of node labels at different iterations to test whether two graphs are isomorphic. In the *graph kernel methods* context, the application of the WL test goes in the direction of computing by measuring the difference between the resultant label sets for two graphs, where the kernel uses the count of node labels as the feature vector of the graph [Shervashidze et al., 2011, Morris et al., 2017]. The reference to the WL test is made at this point to contextualize the reader in existing methods for extracting information at a global level. In the new concept of networks introduced in this report, the application mode is different, and therefore the WL test will be better specified below.

Despite very informative, the statistics described before are limited, as they cannot adapt through a learning process. Therefore, instead of extract hand-engineered features, we will go through the graph, collect its structural information and learn representations that can encode it [Hamilton, 2020].

In this report, we approach a current concept of representation at the graph level, the Graph Neural Network, outlining its basic functioning. We present the Graph Isomorphism Problem, a problem that can be solved for a set of certain graphs using the Weisfeiler-Lehman algorithm and in which these networks have also been potentiating new solutions. After that, we cover some approaches that aim to improve the performance of this combination.

2 Graph Neural Networks

Following the idea of enabling the graph structural representation, methods are applied that transform each node's information and its respective neighbours into low-dimensional vectors [Hamilton et al., 2018a]. This process creates node-related embeddings from the graph (encoder) and allows the graph to be reestablished later from the node's embeddings (decoder). These approaches learn low-dimensional embeddings of the nodes in a graph and, generally, use shallow approaches. In these cases, the encoder function is usually an embedding lookup based on some simple identification of the node [Hamilton, 2020]. Furthermore, the encoder can use node features and the graph structure around the nodes to generate the embeddings. Based on these ideas, and in order to create a more complex and effective solution to this problem, the concept of Graph Neural Network arises [Scarselli et al., 2009].

2.1 Basics

A graph neural network (GNN) uses the graph structure and deep learning mechanisms to generate representations of the nodes and associated features. The functioning of a GNN is based on the exchange of message vectors between nodes and consequent updating of the neural network, known as *neural message passing* [Gilmer et al., 2017]. For a graph $G = (V, E)$, during each iteration, the information of the neighbourhood ($\mathcal{N}(u)$) of the node $u \in V$ is aggregated, and its *hidden embedding* $\boldsymbol{h}_u^{(k)}$ is updated:

$$\boldsymbol{h}_u^{(k+1)} = \text{UPDATE}^{(k)}\left(\boldsymbol{h}_u^{(k)}, \text{AGGREGATE}^{(k)}(\{\boldsymbol{h}_v^{(k)}, \forall v \in \mathcal{N}(u)\})\right) \tag{1}$$

$$= \text{UPDATE}^{(k)}\left(\boldsymbol{h}_u^{(k)}, \boldsymbol{m}_{\mathcal{N}(u)}^{(k)}\right)$$

where UPDATE and AGGREGATE are differentiable functions and the *message* $\boldsymbol{m}_{\mathcal{N}(u)}$ is the information from the u's graph neighbourhood $\mathcal{N}(u)$. As we can see this process in Figure 1, in each iteration, the nodes aggregate information from their neighbourhood and, as iterations pass, the information from the remaining graph reaches each node. In general, after k iterations, each node embedding contains information from its k-hop neighbourhood, which denotes the neighbourhood considered after performing k iterations.

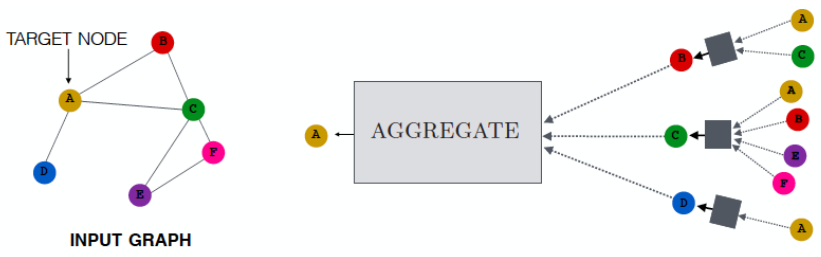


Figure 1: Overview of node aggregation with the neighbourhood [from Hamilton, 2020]

In the final, after K iterations of the GNN message passing, the embeddings for each node are output from the final layer:

$$\boldsymbol{z}_u = \boldsymbol{h}_u^{(K)}, \forall u \in V. \tag{2}$$

These node embeddings have, usually, two types of information: the structure of the graph, which can be the degrees of the nodes associated with the k-hop neighbourhood; and feature-based information that can be, for example, the features in the k-hop neighbourhood. These above equations are written in a logical sense, but if we want to specify the functions present, we can define the basic GNN message passing as

$$\boldsymbol{h}_u^{(k)} = \sigma\big(\boldsymbol{W}_{self}^{(k)} \boldsymbol{h}_u^{(k-1)} + \boldsymbol{W}_{neigh}^{(k)} \sum_{v \in \mathcal{N}(u)} \boldsymbol{h}_v^{(k-1)} + \boldsymbol{b}^{(k)}\big) \tag{3}$$

where $\boldsymbol{W}_{self}^{(k)}, \boldsymbol{W}_{neigh}^{(k)}$ are trainable parameter matrices, σ denotes an elementwise non-linearity (e.g., a tanh or ReLU) and $\boldsymbol{b}^{(k)}$ denotes the bias term. In practice, we add the information from the neighbourhood, then join that information with the previous embedding of the node using a linear combination and, in the end, we apply an elementwise non-linearity. We can also define the basic GNN through update and aggregation functions, denoting the aggregate message ($\boldsymbol{m}$) from u's graph neighbourhood as

$$\boldsymbol{m}_{\mathcal{N}(u)} = \sum_{v \in \mathcal{N}(u)} \boldsymbol{h}_v, \tag{4}$$

updating the message through

$$\text{UPDATE}\big(\boldsymbol{h}_u, \boldsymbol{m}_{\mathcal{N}(u)}\big) = \sigma\big(\boldsymbol{W}_{self} \boldsymbol{h}_u + \boldsymbol{W}_{neigh} \boldsymbol{m}_{\mathcal{N}(u)}\big), \tag{5}$$

where we recall that we aggregate the neighbourhood information through

$$\boldsymbol{m}_{\mathcal{N}(u)} = \text{AGGREGATE}^{(k)}\big(\{\boldsymbol{h}_v^{(k)}, \forall v \in \mathcal{N}(u)\}\big), \tag{6}$$

getting this way the message we want to pass. Until this point we define the operations at the node level. On the other hand, GNNs can also be defined using graph-level equations. The basic GNN has the following model definition at the graph level:

$$\boldsymbol{H}^{(t)} = \sigma\big(\boldsymbol{A}\boldsymbol{H}^{(k-1)} \boldsymbol{W}_{neigh}^{(k)} + \boldsymbol{H}^{(k-1)} \boldsymbol{W}_{self}^{(k)}\big) \tag{7}$$

where $\boldsymbol{H}^{(k)}$ denotes the matrix of node representations at layer t in the GNN and $\boldsymbol{A}$ the adjacency matrix used to represent the graph. The representation of a graph can be achieved through pooling, for example, by summing the representation vectors of all nodes. In this report, the context discussed is at the node level as it is where we are in touch with the basic operating processes. However, we refer here to the graph-level equation because it is a possible global way to define GNNs.

It is crucial to recognise two permutation concepts related to this subject since it is key to designing neural networks in graphs. Concisely, neural networks must be permutation invariant or equivariant, thus not depending on the arbitrary order of nodes that we use in the adjacency matrix. A permutation matrix is a square binary matrix with exactly one entry of ones in each row and each column and zeros elsewhere. By definition of Hamilton [2020], any function f that takes an adjacency matrix $\boldsymbol{A}$ as input should ideally satisfy one of the two following properties:

$$\text{Permutation Invariance } \rightarrow f(\boldsymbol{PAP}^T) = f(\boldsymbol{A})$$
$$\text{Permutation Equivariance } \rightarrow f(\boldsymbol{PAP}^T) = \boldsymbol{P}f(\boldsymbol{A})$$

Permutation invariance means that the arbitrary ordering of the rows/columns in the adjacency matrix does not influence the f that takes the adjacency matrix. In contrast, permutation equivariance means that if we permute the adjacency matrix, then the output of f is also permuted consistently.

2.2 Operation Strategies

The basic GNN presented in Equation (3) can be improved in several ways in terms of aggregation and update functions. The basic functions have already been presented in the previous subsection, in this subsection we only refer to methods that can improve the performance of the message passing.

Aggregation. The most basic neighbourhood aggregation operation is the sum of the neighbour embeddings (Equation 4). If we normalise this based on the degrees of the nodes involved, we can get the average rather than the sum. These follow the notion of neighbourhood normalisation [Kipf and Welling, 2017]. On the other hand, we can make the aggregate operation to be a set function. We map a set of embeddings from a neighbour to a single vector, having this way the neighbourhood information in our message [Zaheer et al., 2017]. In addition to these forms, a common technique for improving the aggregation layer in GNNs is to apply *attention*. The basic idea is to assign importance to each neighbour, and then make different decisions at the moment of aggregation depending on each one's influence. Graph Attention Network (GAT) apply this strategy [Veličković et al., 2018].

Update. The most basic update approach is the linear combination of the node's current embedding with the message coming from its neighbourhood (Equation 5). One common issue with GNNs is *over-smoothing*, where, after several iterations, the representations for the nodes can become very similar to each other. To deal with this problem, we can use an update method where the basic idea is to use vector concatenations or *skip connections*, in which we try to preserve information from previous rounds during the update process [Hamilton et al., 2018a, Pham et al., 2016].

3 GNNs and Graph Isomorphism

The basic functioning of the Graph Neural Networks is connected to the graph isomorphism test. In fact, the basic GNN can be derived from a neural network variation of the Weisfeiler-Lehman isomorphism algorithm, presented in this section. Initially, we approach the general problem that this algorithm solves, the Graph Isomorphism Problem, analysing afterwards the representational power of the GNNs provided by the WL algorithm. Finally, we present an approach that obtains better results than the state-of-art models, explaining why this happens.

3.1 The Graph Isomorphism Problem

The purpose of the graph isomorphism test is to declare whether a pair of $\boldsymbol{G}_1$ and $\boldsymbol{G}_2$ are *isomorphic*, i.e., whether they represent precisely the same graph structure. They can only differ in the order of the nodes in their corresponding adjacency matrices. By definition of Hamilton [2020], if we have two graphs with adjacency matrices $\boldsymbol{A}_1$ and $\boldsymbol{A}_2$ and node features $\boldsymbol{X}_1$ and $\boldsymbol{X}_2$, we say that two graphs are isomorphic if and only if there is a permutation matrix $\boldsymbol{P}$ such that

$$\boldsymbol{PA}_1\boldsymbol{P}^T = \boldsymbol{A}_2 \text{ and } \boldsymbol{PX}_1 = \boldsymbol{X}_2. \tag{8}$$

It is essential to note that the isomorphic graphs are identical in their underlying structure. The order of the nodes in the adjacency matrix has no relation to the underlying graph's structure, being only an arbitrary decision we make when representing a graph using matrices.

The graph isomorphism test is one of the problems in which no polynomial time algorithm is known to solve it. It is known not to be NP-complete, but it is formally identified as NP-indeterminate, i.e., a problem in the complexity class NP but is neither in the class P nor NP-complete. Despite this, some algorithms can test the graph isomorphism in some classes of graphs, including the Weisfeiler-Lehman algorithm.

3.2 The Weisfeiler-Lehman Algorithm

The Weisfeiler-Lehman (WL) test of graph isomorphism is an effective and efficient method for deciding whether two graphs are equal, considering their properties. The simplest version of the WL algorithm (1-WL) [Xu et al., 2019, Hamilton, 2020] follows the next steps:

1. given two graphs, we assign an opening label to each node in both graphs, that can be the degree of the node or another related feature;
2. next, we apply a hash function to the multiset of the current labels related to the neighbourhood's node and the current label of the chosen node, resulting in a new label to each node in both graphs;
3. we repeat step 2 until the labels for all nodes in both graphs converge, i.e., until there are no differences between two consecutive iterations;
4. lastly, we build multisets compiling all the node labels information in both graphs and indicate the graphs as isomorphic if the multisets for both graphs are equal.

Concisely, in each iteration, each node updates its label based on the multiset of labels it has received from its local neighbourhood. After K iterations of this labelling process, each node holds a label that summarizes the structure of its neighbourhood of K-hop. With the combination of these labels, we can characterize the structure of an entire graph or subgraph. The figure 2 illustrates an example of the labelling process.

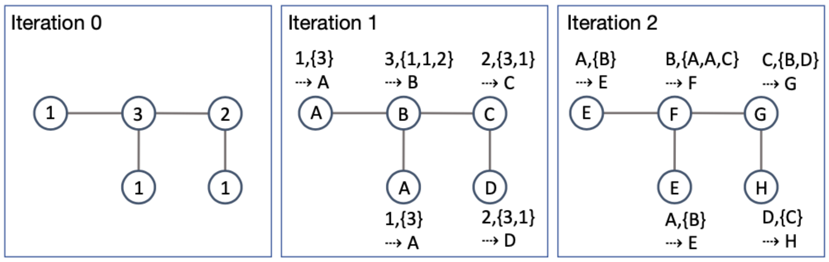


Figure 2: Example of the WL labelling procedure on one graph [from Hamilton, 2020]

There are obvious connections between the WL algorithm and the GNN approach to neural message passing. We update each node's representation based on the aggregated information about the node's neighbourhood in both approaches. The difference between the two approaches is that the WL algorithm aggregates and updates labels using a hash function, while the GNN models aggregate and update node embeddings using neural networks.

The WL algorithm converges in at least $|V|$ iterations and is known to solve the isomorphism test for a broad class of graphs [Babai and Kucera, 1979]. Although successful in general, this algorithm has limitations not working in some classes of graphs. One of them is the class of the regular graphs, where the algorithm cannot even start. This happens because regular graphs have the same degree in each node, and the algorithm cannot partition the node set [Cai et al., 1989]. Another simple example is illustrated in the figure 3 where the test fails. This happens because each node has two degree-2 neighbours, and this information is not sufficient to detect whether a node's neighbours are connected to one another. This limitation stems from the fact that AGGREGATE and UPDATE operations are unable to detect when two nodes share a neighbour. Nevertheless, it is easily visible that the structure of these graphs is different.

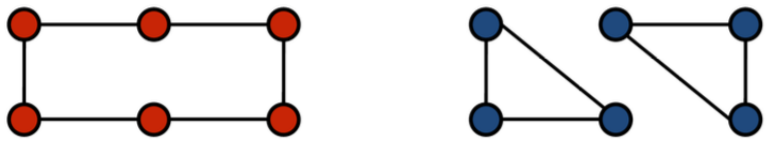


Figure 3: Two graphs (connected 6-cycle and a set of two triangles) that cannot be distinguished by the simplest version of the WL algorithm [from Hamilton, 2020]

3.3 Representational Power of GNNs

The WL algorithm has been widely used for graph representation learning. In particular, to test the representational power in various learning approaches, as is the case with GNNs. Its power can be identified through the results obtained by the representations in the graph isomorphism test. More specifically, with two learned representations $\boldsymbol{z}_{G1}$ and $\boldsymbol{z}_{G2}$ for two graphs, a perfect learning algorithm would generate identical embeddings for two graphs only if those two graphs were also isomorphic:

$$\boldsymbol{z}_{G_1} = \boldsymbol{z}_{G_2} \text{ if and only if } G_1 \text{ is isomorphic to } G_2. \tag{9}$$

What makes the WL algorithm so powerful is its injective aggregation update that maps different neighbourhoods of nodes to different feature vectors. The next theorem from Xu et al. [2019] confirms this.

Theorem 1. *Let $\mathcal{A} : \mathcal{G} \rightarrow \mathbb{R}^d$ be a GNN. With a sufficient number of GNN layers, $\mathcal{A}$ maps any graphs G_1 and G_2 that the Weisfeiler-Lehman test of isomorphism decides as non-isomorphic, to different embeddings if the following conditions hold:*

a) $\mathcal{A}$ aggregates and updates node features iteratively with

$$\boldsymbol{h}_u^{(k)} = \phi\Big(\boldsymbol{h}_u^{(k-1)}, f\Big(\{\boldsymbol{h}_v^{(k-1)} : v \in \mathcal{N}(u)\}\Big)\Big)$$

where the functions f, which operates on multisets, and ϕ are injective.

b) $\mathcal{A}$'s graph-level pooling, which operates on the multiset of node features $\{\boldsymbol{h}_v^{(k)}\}$, is injective.

A GNN that satisfies this theorem induces the WL test by learning to embed the subtrees into the low-dimensional space. This allows GNNs to map similar graph structures to similar embeddings and separating between different structures. Therefore, a GNN can have as great a discriminative power as the WL algorithm if the GNN aggregation operation can model injective functions, as we will see in the model presented below.

3.4 The Graph Isomorphism Network Approach

The Graph Isomorphism Network (GIN) [Xu et al., 2019] is a simple neural network that generalizes the WL test and in which its representational power is equal to the power of the WL test, the maximum discriminative power among GNNs.

As we conclude above, to be as powerful as the WL algorithm, both AGGREGATE and UPDATE operations need to be injective. This means that in these operators, for each output value, there is one and only one respective input value. This approach defines a basic GNN model, which is as powerful as the WL algorithm. They define this model by the following update:

$$\boldsymbol{h}_u^{(k)} = \text{MLP}^{(k)}\Big((1+\epsilon^{(k)}) \cdot \boldsymbol{h}_u^{(k-1)} + \sum_{v \in \mathcal{N}(u)} \boldsymbol{h}_v^{(k-1)}\Big), \tag{10}$$

where MLP is a multi-layer perceptron and $\epsilon^{(k)}$ is a trainable parameter.

Practically, we aggregate the neighbourhood embeddings through a sum, adding these values to the current node's embedding, which is multiplied by a learning value. After that, we apply the resulting value in an MLP, because MLPs can represent the composition of functions.

3.5 Less Powerful State-of-Art Approaches

To help compare power between networks, Xu et al. [2019] created a framework that uses multisets to represent the set of feature vectors of a given node's neighbours. Then, the neighbour aggregation in GNNs can be thought of as an aggregation over the multiset. Consequently, to have large representational power, a GNN must aggregate different multisets into different representations. The more discriminative the multiset function is, the more powerful the representational power of the GNN. Xu et al. [2019] refer the next lemma which states that sum aggregators can represent injective functions over multisets.

Lemma 1. *Assume $\mathcal{X}$ is countable. There exists a function $f : \mathcal{X} \to \mathbb{R}^n$ so that $h(X) = \sum_{x \in X} f(x)$ is unique for each multiset $X \subset \mathcal{X}$ of bounded size. Moreover, any multiset function g can be decomposed as $g(X) = \phi\big(\sum_{x \in X} f(x)\big)$ for some function ϕ.*

Based on Lemma 1 and Theorem 1, we can see why state-of-art approaches like Graph Convolutions Networks (GCN) [Kipf and Welling, 2017] and GraphSAGE (SAmple and AGgregate) [Hamilton et al., 2018b] are not powerful as the GIN, this way not performing so well in the capacity of distinguish different graph structures.

First, GCN is a well-known GNN variant based on graph convolutions, in which we stack simple graph convolutional layers. The element-wise *mean* pooling is used, and the AGGREGATE and UPDATE steps are integrated as follows:

$$\boldsymbol{h}_v^{(k)} = \text{ReLU}\big(W \cdot \text{MEAN}\big(\{\boldsymbol{h}_u^{(k-1)}, \forall u \in \mathcal{N}(v) \cup \{v\}\}\big)\big), \tag{11}$$

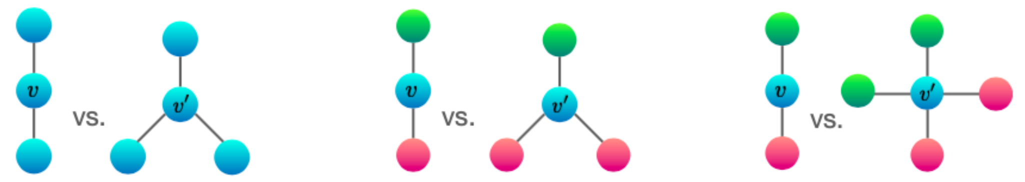
where W is a learnable matrix.

On the other hand, GraphSAGE is a framework for inductive node embedding where node features are analysed to learn an embedding function that generalises unvisited nodes. In the pooling variant, AGGREGATE has been formulated as

$$\boldsymbol{a}_v^{(k)} = \text{MAX}\big(\{\text{ReLU}\big(W \cdot \boldsymbol{h}_u^{(k-1)}\big), \forall u \in \mathcal{N}(v)\}\big) \tag{12}$$

where MAX represents an element-wise max-pooling. The UPDATE step could be a concatenation followed by a linear mapping $W \cdot [\boldsymbol{h}_v^{(k-1)}, \boldsymbol{a}_v^{(k)}]$.

Mean and max-pooling aggregators are permutation invariant, as they are well-defined multiset functions. However, they are not injective and, consequently, they are not so powerful. Analysing the different possibilities: sum collect the full multiset, mean collect the distribution of elements of a given type, and the max aggregator reduces the multiset to a simple set, ignoring multiplicities. Figure 4 illustrates pairs of structures where the mean and max-pooling aggregators fail to identify differences. Node colours denote different node features. We assume that first, the GNNs aggregate neighbours, and after that, they combine that information with the central node labelled as v and v'.



(a) Mean and Max both fail (b) Max fails (c) Mean and Max both fail

Figure 4: Examples of graph structures that mean and max aggregators fail to distinguish. Between the two graphs, nodes v and v' get the same embedding even though their corresponding graph structures differ [from Xu et al., 2019]

In figure 4(a), every node has the same feature a, and $f(a)$ gives the same result for all nodes, regardless of the function f. When performing neighbourhood aggregation, the mean or maximum over $f(a)$ continues $f(a)$ and we always reach the same node representation everywhere. Therefore, in this case, mean and max-pooling aggregators are unable to obtain any structural information. In contrast, the sum aggregator distinguishes the structures because $2 \cdot f(a)$ and $3 \cdot f(a)$ result in different values. Consider h_{color} (r for red, g for green) to denote node features transformed by f. Figure 4(b) shows that maximum over the neighbourhood of the blue nodes v and v' yields *max (hg, hr)* and *max (hg, hr, hr)*, which coincide in the same representation. Therefore, max-pooling cannot distinguish between them. In contrast, the sum aggregator works well because $\frac{1}{2}$ *(hg+hr)* and $\frac{1}{3}$ *(hg+hr+hr)* are different. Likewise, in figure 4(c), both mean and max fail because $\frac{1}{2}$ *(hg+hr)* is equal to $\frac{1}{4}$ *(hg+hg+hr+hr)*.

3.6 Improvement Approaches

In the previous sections, we concluded that there are no message-passing GNN (MP-GNN) algorithms more powerful than the WL algorithm. However, researching how to make GNNs more powerful than the WL algorithm is an area with many research. Here we cover two approaches with potential [Hamilton, 2020].

Relational Pooling. As we see in figure 3, message passing approaches frequently fail to identify closed triangles in a graph. To address this limitation, Murphy et al. [2019] consider expanding MP-GNNs with unique node ID features. In other words, add a unique identifier for each node. In figure 3, adding unique node IDs would allow a MP-GNN to identify when two nodes share a neighbour, which would make the two graphs distinguishable. However, this idea does not work well because, when adding unique node IDs, we introduce a new problematic issue: the MP-GNN is no longer equivalent to permutation. Standard MP-GNNs are permutation equivariant. If we permute the adjacency matrix and node features, then the resulting node embeddings are permuted equivalently. The problem is that assigning a unique ID to each node establishes a node order for the graph, which breaks the permutation equivariance. Murphy et al. [2019] propose the Relational Pooling approach to mitigate this issue, which involves the marginalization of all possible node permutations. The sum of all possible permutation matrices recovers the permutation invariance and we retain the extra representational power of adding unique node IDs. Murphy et al. [2019] prove that this extension of a MP-GNN can distinguish graphs that are indistinguishable by the WL algorithm.

K-dimensional GNNs. Several approaches have considered improving GNNs by adapting the generalisations concerning the WL algorithm. The WL algorithm that we introduced in Section 3.2 is the simplest of the k-WL family of algorithms. The WL algorithm that we introduced earlier is identified as the 1-WL algorithm and can be generalised to the k-WL algorithm for k > 1. The main idea behind the k-WL algorithms is that we label k-size subgraphs instead of individual nodes. As with the 1-WL algorithm, the summary multiset generated by the k-WL algorithm can be used to test graph isomorphism by comparing the multisets for two graphs. Morris et al. [2019] developed a k-GNN approach with similar procedures to the k-WL algorithm. k-GNNs learn embeddings associated with subgraphs instead of nodes, and message passing occurs according to the neighbourhoods of the subgraphs.

4 Conclusion

This report's main idea was to provide a survey related to Deep Learning on Graphs, a promising research field with many challenges. Therefore, we approached a framework that has created new paths and enhanced solutions to Graph Theory problems; in this case, Graph Neural Networks. Part of the progress of Graph Neural Networks is related to the famous Weisfeiler-Lehman algorithm to solve the Graph Isomorphism Problem, considering that both work in similar ways. Consequently, we also addressed this issue, referring to limitations and where advances have been, explaining the concepts of why previously popular approaches have limitations on their representative power.

References

L. Babai and L. Kucera. Canonical labelling of graphs in linear average time. In *20th Annual Symposium on Foundations of Computer Science (sfcs 1979)*, pages 39–46, 1979.

J.-Y. Cai, M. Furer, and N. Immerman. An optimal lower bound on the number of variables for graph identification. In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science*, SFCS '89, page 612–617, USA, 1989. IEEE Computer Society.

Thomas Gärtner, Peter Flach, and Stefan Wrobel. On graph kernels: Hardness results and efficient alternatives. *Lecture Notes in Artificial Intelligence (Subseries of Lecture Notes in Computer Science)*, 2777(August):129–143, 2003.

Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. Proceedings of Machine Learning Research, International Convention Centre, 2017. PMLR.

William L. Hamilton. Graph representation learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 14(3):1–159, 2020.

William L. Hamilton, Rex Ying, and Jure Leskovec. Representation learning on graphs: Methods and applications, 2018a.

William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs, 2018b.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks, 2017.

C. Morris, K. Kersting, and P. Mutzel. Glocalized weisfeiler-lehman graph kernels: Global-local feature maps of graphs. In *IEEE International Conference on Data Mining (ICDM)*, pages 327–336, 2017.

Christopher Morris, Martin Ritzert, Matthias Fey, William Hamilton, Jan Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33:4602–4609, 2019.

Ryan Murphy, Balasubramaniam Srinivasan, Vinayak Rao, and Bruno Ribeiro. Relational pooling for graph representations. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 4663–4673. PMLR, 2019.

Trang Pham, Truyen Tran, Dinh Phung, and Svetha Venkatesh. Column networks for collective classification, 2016.

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.

Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M. Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(77): 2539–2561, 2011.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks, 2018.

Keyulu Xu, Stefanie Jegelka, Weihua Hu, and Jure Leskovec. How powerful are graph neural networks? *7th International Conference on Learning Representations, ICLR*, 2019.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabás Póczos, Ruslan Salakhutdinov, and Alexander J. Smola. Deep sets. *CoRR*, 2017.