TU WIEN Informatics

Do we need to Improve Message Passing?

Improving Graph Neural Networks with Graph Transformations

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Logic and Computation

eingereicht von

Fabian Jogl, BSc
Matrikelnummer 01615015

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ.Prof. Dr. Thomas Gärtner
Mitwirkung: Univ.Ass. Maximilian Thiessen, MSc

Wien, 30. Juni 2022

Fabian Jogl　　　　Thomas Gärtner

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

TU WIEN Informatics

Do we need to Improve Message Passing?

Improving Graph Neural Networks with Graph Transformations

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Logic and Computation

by

Fabian Jogl, BSc
Registration Number 01615015

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.Prof. Dr. Thomas Gärtner
Assistance: Univ.Ass. Maximilian Thiessen, MSc

Vienna, 30th June, 2022

Fabian Jogl   Thomas Gärtner

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Erklärung zur Verfassung der Arbeit

Fabian Jogl, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 30. Juni 2022

Fabian Jogl

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Acknowledgements

To start out, I would like to thank my advisors Prof. Thomas Gärtner and Maximilian Thiessen for their countless hours of work and for believing in my ideas.

I would also like to acknowledge David Penz, Tamara Drucks and Patrick Indri for helping me with my poster by providing ample feedback - it was definitely needed and any remaining bad design decisions are my own fault.

Special thanks to Prof. Jiehua Chen and Manuel Sorge for introducing me to research and showing me how to work on unsolved problems.

I would also like to thank my friends for their support and encouragement. Special thanks to my friend Adrian for helping me throughout my journey at TU Wien.

Most importantly, this thesis would not have been possible without the support and love from my family. Without them I would not be where I am now.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Kurzfassung

Wir untersuchen Graph Neuronale Netzwerke (GNNs) mit geändertem Message Passing und schlagen neuartige Graphtransformationen vor, mit denen Standard Message Passing hochmoderne Expressivität und Vorhersageleistungen erreichen kann.

Es ist bekannt, dass Graph Neuronale Netzwerke mit Message Passing (MPNNs) bei der Unterscheidung von Graphen eine begrenzte Expressivität haben. Das bedeutet, dass Paare von Graphen existieren, für die jedes MPNN das gleiche Ergebnis liefert. Dies schränkt die Funktionen ein, die MPNNs ausdrücken können. Außerdem können MPNNs viele Graphenstrukturen nicht erkennen: ein Beispiel dafür sind zyklische Subgraphen deren Anzahl MPNNs nicht zählen können. Allerdings können diese Strukturen wichtige Informationen beinhalten, zum Beispiel ist die Existenz von Zyklen in Molekülstrukturen wichtig, um Moleküleigenschaften vorherzusagen. Deshalb ist es sinnvoll GNNs zu entwicklen die expressiver als MPNNs sind. Wir definieren, dass ein Algorithmus A mindestens so expressiv wie Algorithmus B ist, wenn A jedes Paar von Graphen unterscheiden kann, das B unterscheiden kann. Es besteht ein Zusammenhang zwischen Universalität und Expressivität: wenn ein Algorithmus alle Graphen eines bestimmten Typs unterscheiden kann, dann ergibt die Kombination dieses Algorithmus mit einem Multilayer Perceptrons automatisch einen universellen Funktionsapproximator auf diesen Graphen. Um MPNNs expressiver zu machen ist es notwendig, eine verbesserte Form des Message Passings zu verwenden oder die Graphen zu modifizieren. Das Ändern von Message Passing ist sehr mächtig, erfordert aber erhebliche Änderungen an bestehenden Implementierungen und kann nicht ohne weiteres mit anderen Ansätzen kombiniert werden. Die Modifizierung der Graphen erfordert keine Änderungen am Lernalgorithmus und funktioniert direkt mit Standardimplementierungen. In dieser Arbeit untersuchen wir vier Varianten von GNNs. Drei GNNs die verbessertes Message Passing verwenden: (I) CW-Networks, (II) Equivariant Subgraph Aggregation Networks und (III) (local) δ-k-dimensional WL. Ein GNN welches als Anwendung einer Graphentransformation gesehen werden kann: (IV) MPNNs mit Homomorphismenzählung.

Wir schlagen neuartige Graphentransformationen für (I), (II), (III) vor und vergleichen diese mit dem aktuellen Stand der Wissenschaft. Wir beweisen, dass die Kombination von Graphentransformation mit einem hinreichend expressiven Algorithmus zumindest so expressiv wie der dementsprechende GNN mit verbessertem Message Passing ist. Besonders interessant ist die Kombination mit einem MPNN das gleich expressiv wie der Weisfeiler-Leman Test ist: in diesem Fall erhält man ein GNN das zumindest so expressiv wie ein GNN mit verbessertem Message Passing ist, welches aber nur Standard Message Passing verwendet. Darüber hinaus zeigen wir empirisch, dass diese Transformationen konkurrenzfähige Ergebnisse auf Molekül Datensätzen liefern. Schließlich untersuchen wir Ähnlichkeiten zwischen (I) und (IV) in einer häufigen Situation im Zusammenhang mit Cliquen. Wir beweisen, dass in diesem Fall (I) mindestens so expressiv im Unterscheiden von Graphen wie (IV) ist.

TU Bibliothek WIEN Your knowledge hub
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Abstract

We investigate graph neural networks (GNNs) with modified message passing and propose novel graph transformations that allow standard message passing to achieve state-of-the-art expressiveness and predictive performance.

Message passing graph neural networks (MPNNs) are known to have limited expressiveness in distinguishing graphs. This means that there exist pairs of graphs for which any MPNN will return the same result. This restricts the functions MPNNs can express and implies that MPNNs are unable to detect many graph structures, for example cyclical subgraphs. However, these structures can contain important information: consider the task of learning properties of molecules, there the cycles in the molecular structure encode important information. Thus, MPNNs are not sufficiently expressive. Formally, we define an algorithm A to be at least as expressive as algorithm B if A can distinguish every pair of graphs that B can distinguish. There is a connection between universality and expressiveness: if an algorithm is able to distinguish all graphs of a certain type, then combining that algorithm with a multilayer perceptron automatically gives us a universal function approximator for these graphs. To make MPNNs more expressive, one can either use an improved variant of message passing or modify the graphs. Changing message passing is powerful but requires significant changes to existing implementations and cannot easily be combined with other approaches. Modifying the graphs requires no changes to the learning algorithm and works directly with off-the-shelf implementations. In this thesis, we investigate four graph neural networks that are based on MPNNs. Three GNNs that can be seen as modifying message passing: (I) CW Networks, (II) Equivariant Subgraph Aggregation Networks and (III) (local) δ-k-dimensional WL. One GNN that can be seen as applying a graph transformation: (IV) MPNNs with homomorphism counts.

We propose novel graph transformations for (I), (II), (III) and compare them to the state-of-the-art. We prove that combining each of the graph transformations with a suitably expressive algorithm is at least as expressive as the corresponding GNN with modified message passing. By combining the graph transformation with an MPNN that is as expressive as the Weisfeiler-Leman test, one obtains a GNN that is as expressive as a GNN with modified message passing but *without* modifying message passing. Furthermore, we empirically demonstrate that these transformations lead to competitive results on molecular graph datasets. Finally, we investigate similarities between (I) and (IV) in a common situation relating to cliques. We prove that in this case (I) is at least as expressive as (IV) when it comes to distinguishing graphs.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Contents

Kurzfassung ix

Abstract xi

Contents xiii

1 Introduction 1

2 Preliminaries 5
2.1 Graphs and Graph Isomorphism 5
2.2 Supervised Machine Learning on Graphs 6
2.3 Training of Deep Neural Networks 7
2.4 Message Passing Graph Neural Networks 12
2.5 How Powerful are Message Passing Graph Neural Networks? 13

3 Related Work 17
3.1 Graph Kernels 17
3.2 Graph Neural Networks 18
3.3 Expressiveness of Graph Neural Networks 19

4 More Expressive GNNs 21
4.1 MPNNs with Homomorphism Counts 21
4.2 CW Networks 22
4.3 Equivariant Subgraph Aggregation Networks 24
4.4 δ-k-dimensional Weisfeiler-Leman 26

5 Cliques: Counting Homomorphisms vs Lifting 27
5.1 CWL and Counting Homomorphisms 27
5.2 Complete Graphs and Clique Complex Lifting 29

6 Reducing Learning on Cell Complexes To Graphs 33
6.1 Cell Encoding 33
6.2 Cellular Ring Encoding 35
6.3 Experiments 37

7 Simplifying Augmented Message Passing 41
7.1 Guidelines for graph transformations 41
7.2 Equivariant Subgraph Aggregation Networks 42

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

7.3 δ-k-dimensional Weisfeiler-Leman 45
7.4 Experiments 48

8 Conclusion **51**

List of Figures **53**

List of Tables **55**

Bibliography **57**

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

CHAPTER 1

Introduction

It has been proven that message passing graph neural networks (MPNNs) have limited expressiveness with respect to distinguishing graphs. A common approach of making graph neural networks (GNNs) more expressive modifies the message passing algorithm that underlies MPNNs. While this approach can be used to build more expressive GNNs, it has the downside of needing custom message passing implementations that are incompatible with other algorithms and implementations. A different approach makes GNNs more expressive by using graph transformations to transform the input graphs. This has the advantage that graph transformations are easy to combine with existing implementations, as it only requires changes to the data preprocessing. We investigate the following question: Is it necessary to improve message passing or can we use graph transformations to achieve the same expressiveness? For this we study three GNNs with improved message passing and propose a novel graph transformations for each GNN. We show that that combining one of our graph transformations with an MPNN is at least as expressive as the corresponding GNN with improved message passing, as long as the MPNN is at least as expressive as the Weisfeiler-Leman test. This demonstrates that improving message passing is not necessary to achieve the expressiveness of the studied algorithms. Additionally, we demonstrate that these transformations lead to competitive results on molecular graph datasets.

Given how useful graphs are in many different domains (see Figure 1.1), it makes sense to develop methods that enable us to automatically learn predictive functions based on graph data. For instance, to develop new antibiotics it is necessary to test large libraries of molecules in a lab [Stokes et al., 2020]. Unfortunately, this is both time consuming and expensive. It would be useful if we could train a graph neural network (GNN) on all well understood molecules to predict whether a given molecule has antibiotic properties. Afterwards, this GNN can predict whether new molecules have potential to be used as an antibiotic. With this we can reduce the number of molecules to be tested in a lab to only those which the GNN predicts to have potential and reduce the costs of finding new antibiotics. A similar approach has been used by Stokes et al. [2020] to find the molecule Halicin which has been shown to help against bacterial infections in mice. GNNs are a very common approach to learning on graph data by applying neural networks to graphs. A common variant of GNNs, also used by Stokes et al. [2020] is that of message passing graph neural networks (MPNNs). MPNNs are constructed

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

in multiple layers. In each layer, a representation is computed for each node, depending on the nodes' previous representations and the representations of its neighbors. After the final layer, node representations get aggregated to obtain a single vector for the entire graph. This vector is then passed through a multilayer perceptron which returns the prediction for the entire graph. MPNNs are called *message passing* as updating the representations based on neighborhoods can be seen as passing messages between neighboring nodes.

In this thesis we investigate GNNs that are based on MPNNs but are set apart by either their use of graph transformations or an improved variant of message passing. These GNNs were developed because MPNNs have been proven to have limited expressiveness when it comes to distinguishing graphs [Xu et al., 2019, Morris et al., 2019]. This means that there exist pairs of graphs which no MPNN can distinguish. Additionally, MPNNs struggle to detect structures as can be seen by their inability to count induced subgraphs for *any* connected pattern with at least three nodes [Chen et al., 2020]. This also means that they are unable to count cycles of any length. Given that cycles are important to the structure of molecules this motivates the need for more expressive MPNNs, especially for the domain of biology and medicine.

To make MPNNs more expressive and allow them to distinguish more pairs of graphs we can either change the input graphs via a graph transformation or improve the message passing algorithm. Improving expressiveness via a graph transformation has the advantage that only the graphs need to be transformed before running the learning algorithm. This allows simple integration into previous implementations and can be easily combined with other existing models, for example other graph neural networks. However, changing message passing is a much more general approach and allows for a lot of flexibility when it comes to designing more expressive algorithms. In contrast, improving MPNNs via graph transformations seems very limited as the resulting algorithm will still make use of classical message passing. Thus, intuitively it seems that improving message passing should allow us to build more expressive algorithms than by applying just a graph transformation combined with message passing. In this thesis we show that this is not necessarily the case. We investigate the question: Is it necessary to improve message passing or can we use graph transformations to achieve the same expressiveness? For this, we study four GNNs that are more expressive than MPNNs: (I) CW Networks [Bodnar et al., 2021a], (II) Equivariant Subgraph Aggregation Networks [Bevilacqua et al., 2022], (III) (local) δ-k-dimensional WL [Morris et al., 2020b], and (IV) MPNNs with homomorphism counts [Barceló et al., 2021]. These GNNs can be seen as first applying a transformation to the graphs. For three of them this transformation is to a different structure such as (I) regular cell complexes, (II) multisets of subgraphs or (III) tuples of nodes. For (IV) this transformation is to another graph in which case we call this a *graph transformation*. Then they either apply the usual message passing in case of (IV) or some improved form of message passing in case of (I), (II), (III).

Our contributions are as follows. We propose reducing algorithms with improved message passing to a graph transformation combined with message passing. We demonstrate the application of this idea by proving that three methods with improved message passing (I), (II), (III) can be reduced to graph transformations. For all three methods we propose novel graph transformations and prove that combining our graph transformations with a sufficiently powerful GNN or WL is at least as expressive as the original algorithm. As a consequence, this also means that the combination of our graph transformation together with an MPNN is at least as expressive as the original algorithms, if the MPNNs architecture fulfills some

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

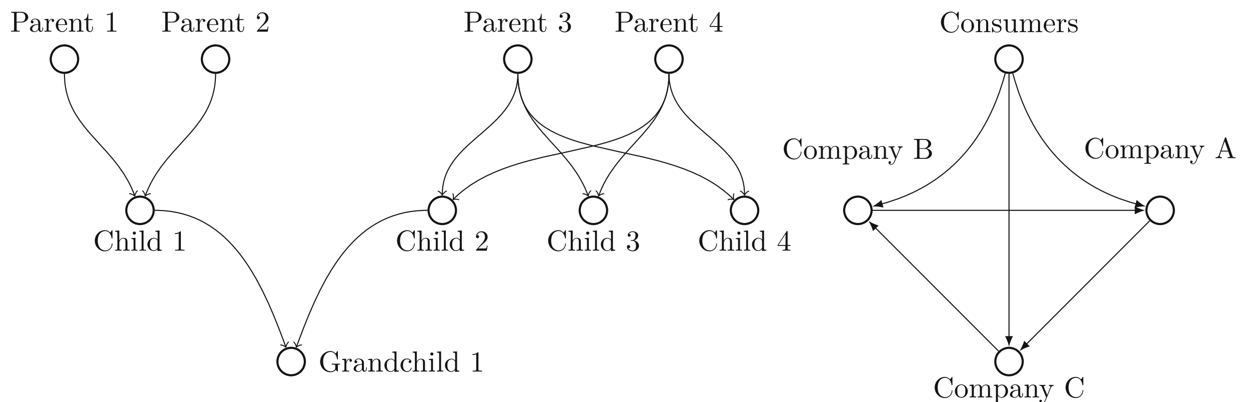

Figure 1.1: Left: a social hierarchy can be represented as a graph. An edge from a vertex X to a vertex Y means that X is a parent of Y. Right: a Leontif directed graph representing an economy. Edges represent funds flowing from the consumers to a company or from a company to a company. (Figure based on Foulds [2012])

requirements like a sufficient number of layers. Additionally, we empirically demonstrate that our methods achieve competitive results on graph datasets. Finally, we also investigate a connection between a graph transformation that counts homomorphisms (IV) and CW Networks (I). We show that for the case of adding clique homomorphisms counts to graphs (IV) and CW Networks (I) that transform cliques into their own entities (I) is at least as expressive as (IV).

We start by defining the basic concepts of graphs, machine learning and (message passing) graph neural networks in Section 2. Then, in Section 3 we give an overview of related work starting with graph kernels and ending with recent works on the expressiveness of message passing graph neural networks. In Section 4 we introduce the GNNs which we will further investigate in the following sections. Next, in Section 5 we investigate a connection between MPNNs with homomorphism counts and CW Networks when it comes to patterns of cliques. We introduce the idea of simplifying or reducing algorithms with improved message passing to a graph transformation combined with message passing in Section 6. For this, we prove that our novel graph transformation *cell encoding* can give MPNNs the expressiveness of CW Networks [Bodnar et al., 2021a] and demonstrate that our algorithm achieves competitive performance empirically. Then, we extend this idea by introducing guidelines that such graph transformations should follow in Section 7. To show the generality of our approach, we reduce two additional algorithms to graph transformations: Equivariant Subgraph Aggregation Networks [Bevilacqua et al., 2022] and (local) δ-k-dimensional WL [Morris et al., 2020b]. This section concludes with experiments that show that our algorithm achieves state-of-the-art results competitive with their original algorithms. Finally, we conclude this work with Section 8 by giving an overview of our results and suggesting future work.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Parts of this thesis have been published as:

- Jogl Fabian, Thiessen Maximilian, Gärtner Thomas. *Reducing Learning on Cell Complexes to Graphs.* Workshop on Geometrical and Topological Representation Learning at ICLR, 2022
- Jogl Fabian, Thiessen Maximilian, Gärtner Thomas. *Weisfeiler and Leman Return with Graph Transformations.* under review, 2022

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

CHAPTER 2

Preliminaries

In this section, we introduce basic concepts needed to understand current work on the expressiveness of graph neural networks. For this, we start by introducing graphs (Section 2.1) and supervised machine learning (Section 2.2). Then, we give a short primer on training deep neural networks (Section 2.3). Finally, we introduce message passing graph neural networks (Section 2.4) and their shortcomings (Section 2.5).

2.1 Graphs and Graph Isomorphism

Figure 2.1: Left: a complete graph with three vertices (K_3), called a triangle or a cycle of length 3 (C_3); Right: a complete graph with four vertices (K_4)

We start with the definition of a graph and basic concepts related to graphs. For this, we take definitions from Diestel [2005]. However, we make some changes in notation for simplification or staying more aligned with the relevant literature.

We define a graph to be a ordered pair $G = (V(G), E(G))$ of sets called the vertex set $V(G)$ and edge set $E(G)$. For our purposes, we require $V(G)$ to be non-empty and finite. This work focuses on *undirected* graphs, thus the set $E(G)$ contains unordered pairs of different vertices $E(G) \subseteq \{\{v, w\} \mid v, w \in V(G), v \neq w\}$. Each edge $\{v, w\} \in E(G)$ joins two *different* vertices v and w; we say that v and w are adjacent, or neighbors. Additionally, we say that v and w are incident with edge $\{v, w\}$. If the graph we talk about is clear then we will denote a graph by just $G = (V, E)$. We use $\mathcal{N}_G(v)$ to denote the *neighborhood* of v, that is all vertices that are adjacent to v.

An adjacency matrix $A \in \mathbb{R}^{n \times n}$ can be used to represent the edges of a graph with n vertices.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

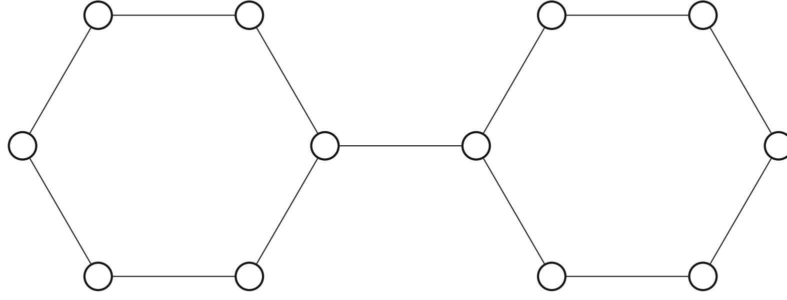

Figure 2.2: A graph modeling a biphenyl molecule

Suppose we have an ordering on our vertices $V = \{v_1, \ldots, v_n\}$. Then, A is defined as

$$A_{ij} = \begin{cases} 1 & \text{if } \{v_i, v_j\} \in E \\ 0 & \text{else} \end{cases} .$$

A graph is called *complete* if all its vertices are pairwise adjacent; we call a complete graph with n vertices a K_n and a K_3 a triangle. Figure 2.1 shows a K_3 and a K_4.

A *path* is a non-empty graph $P = (V, E)$ with $V = \{x_0, x_1, \ldots, x_k\}$ and $E = \{\{x_0, x_1\}, \{x_1, x_2\}, \ldots, \{x_{k-1}, x_k\}\}$. We define the length k of a path to be equivalent to the number of edges in the path. Note that all vertices in V are distinct. To simplify notation, we refer to P as $x_0x_1 \ldots x_k$ and call it a path *from* x_0 *to* x_k. Similarly to paths, we define a walk as an ordered sequence of vertices connected by edges. Note, that unlike in a path, the vertices in a walk are not distinct and can appear multiple times. Let $P = x_0 \ldots x_{k-1}$ be a path with $k \geq 3$. Then, adding the edge $\{x_{k-1}, x_0\}$ to P yields a graph called a *cycle*. We denote cycles by $x_0 \ldots x_{k-1}x_0$. The length of a cycle is its number of edges or vertices. A cycle with k edges is called a k-cycle or a C_k. Figure 2.1 depicts a C_3.

Let $G = (V, E)$ and $G' = (V', E')$ be two graphs. We say that G and G' are *isomorphic* if there exists a bijection $\alpha : V \to V'$ with $\{x, y\} \in E$ if and only if $\{\alpha(x), \alpha(y)\} \in E'$. Intuitively, if two vertices from V are neighbors, then applying α to them yields two neighbors from V' and vice versa. The *graph isomorphism* problem as defined in Babai [2016] is to decide whether two given graphs are isomorphic. There are no known algorithms that solve this problem and run in polytime. However, the graph isomorphism problem has not been proven to be NP-hard. Thus, it is unclear if there exists an efficient algorithm that solves this problem. However, a quasi-polynomial algorithm is known to exist [Babai, 2016].

2.2 Supervised Machine Learning on Graphs

We introduce basic concepts of supervised machine learning on graphs. For this, we follow Jegelka [2022]. We extend the concept of a graph from an ordered pair $G = (V, E)$ to a quadtuple $G = (V, E, X, W)$. Here, $X = \mathbb{R}^{d \times |V|}$ is a matrix that contains a vector of dimensions d for each node and $W \in \mathbb{R}^{k \times |E|}$ is a matrix that contains a vector of dimension k for each edge. We say that X and W are the vertex and edge features, respectively. For a vertex v and an edge $\{p, q\}$ we use $X(v)$ and $W(p, q)$ to denote the features of v and $\{p, q\}$, respectively.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

As an example, consider the graph in Figure 2.2. This graph encodes the molecule biphenyl [Lide, 2003]. Each vertex in the graph encodes one of the atoms making up biphenyl. The edges encode bonds between atoms. Accordingly, vertex features contain information about the atom such as its type, and edge features contain information about the bounds.

Let $\mathcal{Y}$ be a set called a label set and $\mathcal{G}$ be a set of graphs. Let $\mathcal{F}$ be a class of models, where any $F \in \mathcal{F}$ is a function $F : \mathcal{G} \to \mathcal{Y}$. For example, $\mathcal{F}$ could be the class of all GNNs. For this work we consider tasks where we are given N independent and identically distributed (i.i.d.) samples $\mathcal{D} = (G_i, y_i)_{i=1}^N$ drawn from an fixed but unknown distribution $\mathcal{P}$ on $\mathcal{G} \times \mathcal{Y}$. Each y_i is called the label of G_i. We are given a loss function $l : \mathcal{G} \times \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}^+$ that measures the quality of a prediction $F(G_i)$ by a model F on any sample G_i. Our goal is to find a model F from a model class $\mathcal{F}$ that minimizes the expected loss

$$\min_{f \in \mathcal{F}} \mathbb{E}_{(G,y) \quad \mathcal{P}}[l(G, y, F(G))].$$

Note, that while the assumption that our samples are independent and identically distributed is part of our problem setting, this work does not make explicit use of this assumption. Instead, we just assume we are given a set of graphs together with labels and we are trying to find a model that can predict the label of a graph accurately and generalizes outside our given samples. As an example consider prediciting the solubility of a molecule. For this let $\mathcal{Y} = \mathbb{R}^+$. Thus our goal is to predict a positive real number associated with each graph, this means we want to perform graph regression. Then, the loss l needs to be capable of measuring the difference between our prediction $\hat{y}$ and the label y. A common loss for this is the L_1 loss defined by $|\hat{y} - y|$.

For real world problems, we do not have access to the distribution $\mathcal{P}$ directly. Instead, we are given a training and a test set. Both sets contain graphs with labels. We select our model based on the training set and then evaluate the model on the test set. This allows us to estimate how good our model performs on unseen data from the distriution $\mathcal{P}$.

To summarize, we are given a list of graphs together with a label for each graph. Our goal is to use this data to find a model that minimizes the expected loss. For this work, we will focus on graph level tasks. That is, we have to predict a property for the entire graph. For example, if the graphs represent molecules then the task could be to predict whether a given molecule inhibits the replication of HIV [Hu et al., 2020] (graph classification), or to predict how soluble a molecule is (graph regression). Other tasks on graphs are node classification or edge prediction, that is, predicting whether there is a missing edge between two given vertices in the graph.

2.3 Training of Deep Neural Networks

In this section we describe how to train *deep neural networks* (DNNs). This will serve as a basis to understanding message passing graph neural networks. For most of this section we use the deep learning books of Goodfellow et al. [2016] and Zhang et al. [2021] as our sources. Instead of graphs, we assume that we are given euclidean vectors of data, this means that our training set has the shape $(x_i, y_i)_{i=1}^N$ where $x_i \in \mathbb{R}^n, y_i \in \mathcal{Y}$. We start by introducing the multilayer perceptron (MLP) as one of the most basic DNNs. Then we describe how to train DNNs. Finally, we explain why MLPs struggle with learning functions on graphs.

TU Bibliothek WIEN Your knowledge hub
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Multilayer perceptron. One of the most important DNNs is the *multilayer perceptron* (MLP). Suppose our goal is to approximate a function $y = f^*(x)$, that is the function that generates the label y for every sample x. For this we want to find an MLP $f(x;\theta)$ that approximates f^* and depends on the parameters θ. During training, the goal is to find good values for θ. MLPs and most neural networks are organized into *layers*. These layers combine linear operations together with non-linear *activation functions*. A k layer MLP can be defined via [Goodfellow et al., 2016, 168-177]:

$$h^{(1)} = g^{(1)}\left(W^{(1)T}x + b^{(1)}\right),$$
$$h^{(j)} = g^{(j)}\left(W^{(j)T}h^{(j-1)} + b^{(j)}\right), \qquad \forall j \in \{2, \ldots, k\}.$$

Where for every i in $\{1, \ldots, k\}$ in this definition, $W^{(i)} \in \mathbb{R}^{n_{i-1} \times n_i}$ is a matrix, $b^{(i)} \in \mathbb{R}^{n_i}$ is a vector and $g^{(i)}$ is a non-linear function called an activation function. The parameters θ correspond to $W^{(i)}, b^{(i)}$, for i in $\{1, \ldots, k\}$. We can see that first a linear function defined by $W^{(1)T}x + b^{(1)}$ is applied to the input x and then combined with a non-linear function $g^{(1)}$. The result of this calculation is then fed into the next layer, where the same type of computation is done again. After the final layer, we get an output $z = h^{(k)} \in \mathbb{R}^{n_k}$. [Goodfellow et al., 2016, 168-177]

A common choice for the activation functions $g^{(i)}$ is the *rectified linear unit* (ReLu) defined coordinate-wise by [Goodfellow et al., 2016, 174]:

$$\mathrm{ReLU}(z)_i = \max(0, z_i) \qquad \forall i \in \{1, \ldots, n_k\}.$$

A vector of real numbers as output might suffice for certain tasks. For example, when predicting the price of a house we want an output z from $\mathbb{R}$. Similarly, if we want to predict coordinates in 3d space, then we want an output z from $\mathbb{R}^3$. To achieve such an output we can just remove the activation function from the final layer. However, for a classification task we want to predict the class of the input. Suppose we have m classes and want to predict exactly one. Then, the final layer should output a vector z from $\mathbb{R}^m$. The value of z_i predicts the score of class i, the bigger z_i the more likely the model predicts that i is true. To turn this into values from the intervall $[0, 1]$, one can use the softmax function [Goodfellow et al., 2016, 181-187]:

$$\mathrm{softmax}(z)_i = \frac{\exp(z_i)}{\sum_{j=1}^{n_k} \exp(z_j)}. \tag{2.1}$$

After applying a softmax, each z_i is between 0 and 1, and sum up to 1 in total. In the case of predicting only one of two classes (*binary classification*) we can also just predict the probability of only one of the classes. Then, the final layer outputs just $z \in R$ instead of $z \in R^2$. To turn this z into a valid probability, it must lie in the interval [0,1]. This can be ensured with the *logistic sigmoid* function [Goodfellow et al., 2016, 181-187, 68]:

$$\sigma(z) = \frac{1}{1 + \exp(-z)}. \tag{2.2}$$

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Losses. The loss measures difference between our models predictions $\hat{y}$ and the ground truth y. A common loss for regression tasks is the L_1 *loss* [1]:

$$L_1(\hat{y}, y) = |\hat{y} - y|. \tag{2.3}$$

Let us now consider classification tasks. Suppose x belongs to one of q classes and the model outputs the prediction in the shape of a vector $\hat{y} \in \mathbb{R}^q$ whose elements $\hat{y}_i$ are the probability of x belonging to class i. Additionally, y *one-hot* encodes the label of x, that is, y_i is 1 if x belongs to class i and 0 otherwise. For multi-class classification the *cross entropy loss* is commonly used. Cross entropy is defined as [Zhang et al., 2021, 110-111]:

$$\mathrm{CE}(\hat{y}, y) = -\sum_{i=0}^{q-1} y_i \log(\hat{y}_i). \tag{2.4}$$

For binary classification tasks where the model only predicts a single probability one can use the *binary cross entropy loss*. Note that both $\hat{y}$ and y are one dimensional. Binary cross entropy loss is defined as[2]:

$$\mathrm{BCE}(\hat{y}, y) = -y \log(\hat{y}) + (1 - y) \cdot \log(1 - \hat{y}). \tag{2.5}$$

Gradient Descent. At this point, we have seen how we can use the MLP to define a model and how to use loss functions to evaluate its performance. Now we will introduce a way of setting the models parameters, that is we will *train* the model. This is most commonly done with *stochastic minibatch gradient descent*. The goal is to set the parameters θ of a model f to minimize a given loss function l. In each iteration of this algorithm, we sample a minibatch $\mathcal{B}$ of a fixed number of random samples from our training set. We update our weights with the gradient of the average loss with respect to the model parameters θ in each time step t [Zhang et al., 2021, 465-468, 90]:

$$\theta_{t+1} \leftarrow \theta_t - \frac{\eta_t}{|\mathcal{B}|} \sum_{(x,y)\in\mathcal{B}} \nabla_{\theta_t} l(f(x;\theta_t), y). \tag{2.6}$$

The number of samples per batch $|\mathcal{B}|$ is called the *batch size* and $\eta_t \geq 0$ is the *learning rate*. These parameters influence our model, but are not tuned during the training. They are called *hyperparameters*. [Zhang et al., 2021, 90] Other hyperparameters are the number of layeres in our MLP, or the width n_i of a given layer i.

In practice, the sampling is done by permuting the training set, so that all samples will be drawn equally often. Running gradient descent over the entire training set once is called an *epoch*. After a given number of *epochs* (another hyperparameter) or after some stopping parameter is met, the training stops [Zhang et al., 2021, 90]. There exist more advanced variants of gradient descent such as Adam [Kingma and Ba, 2015] and AdamW [Loshchilov and Hutter, 2019] which regularly used in practice. However, we will not introduce them here as understanding their inner workings is not relevant to our work.

[1] See the PyTorch definition of L1 loss `pytorch.org/docs/stable/generated/torch.nn.L1Loss.html#torch.nn.L1Loss`

[2] See the PyTorch definition of binary cross entropy loss with logits `pytorch.org/docs/stable/generated/torch.nn.BCEWithLogitsLoss.html`

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Schedulers. The learning rate has a big influence on the training. Setting it too low means that the training will take a long time. Setting it too high means that the model might not learn anything at all. However, even if a good learning rate has been selected it might not be optimal during the entire training procedure. At the start of training a higher learning rate might be more beneficial as the model has to learn the big picture. Later, a smaller learning rate might be beneficial so the model does not change too much and overwrites things previously learned [Zhang et al., 2021, 501-505]. For this a *scheduler* can be used.

Here we introduce two different schedulers. The first is *StepLR*[3], this reduces the learning rate by a fixed factor every time a given number of epochs have passed. The second is *ReduceLROnPlateau*[4], this scheduler reduces the learning rate by a fixed factor every time a metric (such as the loss on the validation set) has not improved for a given number of epochs.

Dataset splitting. To ensure that we have an accurate prediction of how our model performs on unseen data, we split our dataset into three parts: *training* dataset, *validation* dataset, and *test* dataset. The training dataset is used to optimize the parameters θ of our model via a variant of gradient descent. The validation dataset, is used to tune our hyperparameters, that is, to find hyperparameters that allow us to train a well performing model. The test set, is used to measure how good the final model performs on *unseen* data. For this, it is very important that the test set is not used during the training. [Zhang et al., 2021, 145-146]

An alternative to the above described training, validation and test split is *k-fold cross validation*. For this the dataset is split into k non-overlapping subsets (folds), that is, k sets of equal size or almost equal size that do not have any element in common and whose union returns the original dataset. To obtain a test error, one takes the average over k trials. In each trial, a different subset is regarded as the test set and the other subsets as the training set. This works better than a train, validation and test split on small datasets, as a small test set makes it difficult to statistically accurately estimate the test error. By averaging over k different test sets we get more significant results. However, this comes at the cost of potentially significant runtime increases due to needing k trials. [Goodfellow et al., 2016, 121]

For classification tasks, it can be beneficial to have similar class distributions in all folds. This is called *stratified* k-fold cross validation. [Kohavi, 1995]

Hyperparameter tuning. A simple way of finding good hyperparameters is *random search*. For each hyperparameter, a marginal distribution of possible values is chosen. Then a set of hyperparameters is sample from these distributions and a model is trained with these hyperparameters on the training set and evaluated on the validation set. This process gets repeated for a fixed number of times, and the model with the best validation performance gets selected as the final model. An alternative to random search is *Bayesian optimization*. For this, a Bayesian regression model predicts the validation error for hyperparameters choices on the basis of how previous hyperparameters have performed. This allows the model to ignore hyperparameters that have had little importance in previous experiments and thus to find good hyperparameters more quickly. [Goodfellow et al., 2016, 432-436]

[3] See the PyTorch documentation `pytorch.org/docs/stable/generated/torch.optim.lr_scheduler.StepLR.html`

[4] See See the PyTorch documentation `pytorch.org/docs/stable/generated/torch.optim.lr_scheduler.ReduceLROnPlateau.html`

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Wien Bibliothek Your knowledge hub

Overfitting and Underfitting. After training a model we need to evaluate its performance. For this we only consider the errors on the training and validation set, as the test set is purely for evaluating our final model. There are two common problems that can arise. The first is that training and validation errors are both similar and high. This means that our model is *underfitting*, that is, it is unable to fit the training data. This usually means that we have to increase the model complexity by adding new layers, or increasing the width of the layers. The second is that the training error is significantly lower than the validation error. This means that our model fits well to the training data, but struggles to generalize to unseen data. This is called *overfitting*. Common ways to remedy overfitting are decreasing the model complexity, adding dropout to the model, or using early stopping. [Zhang et al., 2021, 146-147]

Dropout. *Dropout* is a special layer to combat overfitting in neural networks. It is usually applied after the activiation function. During inference, a dropuout layer returns the input unchanged. During training, it sets each element of its input to zero with probability p and scales up all other elements of the input by $1/(1-p)$. This ensures that the expected value of each layer does not change. Setting some values to zero injects noise into the training process and enforces smoothness on the input-output mapping. [Zhang et al., 2021, 159-165]

Early stopping. *Early stopping* allows us to avoid overfitting on the training set, by stopping the training once overfitting starts. This means that we get a model that is better at generalization and also saves computation time. For this we monitor the validation error during training. Every time the validation error decreases, we store a copy of our model. If the maximum number of epochs is reached or the validation error has not decreased for a certain number of epochs, we return the stored model with the lowest validation error. [Goodfellow et al., 2016, 246-247]

Universality. When thinking about models that should approximate functions it is interesting to consider what kind of functions a model like an MLP can approximate. After all, if our model is incapable of expressing the target function then more data or a better training procedure will not help much. Ideally, we would want our model to be able to fit an arbitrary function. Indeed, this holds for MLPs. It can be shown that MLPs with just a single hidden layer are *universal approximators*, that is, they can approximate any function [Hornik et al., 1989]. Unfortunately, this does not mean that we will always learn *any* specific desired function from the given data, it just means that there exists a specific MLP that can represent this chosen function. [Goodfellow et al., 2016, 198-199]

Why MLPs do not work on graphs. Given the universality of MLPs, it seems tempting to apply them to graphs. Unfortunately, this does not work. An MLP takes a fixed size vector as an input. However, graphs can have an arbitrary size, meaning that there is no natural way to input them into an MLP. More importantly, permuting the vertices in a graph does not change the graph. Thus if we permute the vertices of a graph, we want our models output to be invariant to the permutation. This means that we want our model to represent *permutation invariant* functions. [Jegelka, 2022]

Definition 1 (Jegelka [2022])**.** *Let $f : \mathcal{G} \to \mathbb{R}^n$ be a function. Let G be a graph with adjacency matrix $A \in \mathbb{R}^{n\times n}$ and node features $X \in \mathbb{R}^{n\times d}$. Let $P \in \mathbb{R}^{n\times n}$ be a permutation matrix of G,*

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

that is a matrix that permutes the vertices of G. Then f is permutation invariant if for all graphs G and all permutation matrices P of G it holds that $f(PAP^\top, PX) = f(A, X)$.

The functions learned by an MLP are not necessarily permutation invariant. This motivates the need for another type of neural network that can operate on graphs and learn only permutation invariant functions. One such neural network is the message passing graph neural network. [Jegelka, 2022]

2.4 Message Passing Graph Neural Networks

In this section we define *message passing graph neural networks* (MPNNs) [Gilmer et al., 2017], a specific type of neural network that is permutation invariant and adapted to graphs. In this section we use the definitions from Jegelka [2022]. We use $\{\{\cdot\}\}$ to denote a multiset.

MPNNs consist of up to $T \geq 1$ message passing layers. We call T the depth of the MPNN. In each layer t a representation (embedding) vector $h_v^{(t)} \in \mathbb{R}^{d_t}$ is computed for each vertex $v \in V$. The depth of the MPNN influences how far information will be aggregated from the graph: $h_v^{(t)}$ encodes the t-hop neighborhood of v, that is, the subgraph consisting of all vertices that can be reached from v with a path of length at most t. In each layer t, the embedding of a vector v gets updated from its previous embedding $h_v^{(t-1)}$, the embeddings of its neighbors and possible edge features:

$$
\begin{aligned}
h_v^{(0)} &= X(v), && v \in V \\
m_v^{(t)} &= f_{\text{Agg}}^{(t)}\left(h_v^{(t-1)}, \left\{\left\{h_u^{(t-1)}, W(u,v) \mid u \in N(v)\right\}\right\}\right), && 1 \leq t \leq T \\
h_v^{(t)} &= f_{\text{Up}}\left(h_v^{(t)}, m_v^{(t)}\right), && 1 \leq t \leq T.
\end{aligned}
$$

Here, $f_{\text{Agg}}^{(t)}$ is called the aggregation function and $f_{\text{Up}}^{(t)}$ the update function. The aggregation function is a non-linear function of the form

$$
\begin{aligned}
&f_{\text{Agg}}^{(t)}\left(h_v^{(t-1)}, \left\{\left\{h_u^{(t-1)}, W(u,v) \mid u \in \mathcal{N}(v)\right\}\right\}\right) = \\
&\phi_1^{(t)}\left(\Omega\left(\left\{\left\{\phi_2^{(t)}\left(h_u^{(t-1)}, h_v^{(t-1)}, W(u,v)\right) \mid u \in \mathcal{N}(v)\right\}\right\}\right)\right).
\end{aligned}
$$

Where Ω is a function that operates on multisets. Usually Ω is chosen to be the sum, but one can also use an average, a maximum or a degree-normalized sum. [Xu et al., 2019, Jegelka, 2022]

The functions $\phi_1^{(t)}, \phi_2^{(t)}$ are defined as MLPs. The update function is usually defined as:

$$
f_{Up}\left(h_v^{(t)}, m_v^{(t)}\right) = \sigma\left(W_1^{(t)} h_v^{(t)} + W_2^{(t)} m_v^{(t)}\right) \tag{2.7}
$$

In this equation, σ is a coordinate-wise nonlinear activations, for example the logistic sigmoid function, and $W_1^{(t)}, W_2^{(t)}$ are learnable weight matrices.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Since we focus on graph-level tasks, we want a single representation for the entire graph, for this we aggregate all node representations (pooling) with a permutation invariant *readout* function

$$F(G) = f_{\text{Read}}\left(\left\{\!\!\left\{h_v^{(T)} \mid v \in V\right\}\!\!\right\}\right). \tag{2.8}$$

For our purposes, the readout function computes a sum or an average. After the readout function, one can apply another MLP and possibly a softmax function if the task is graph classification.

To avoid losing information by aggregation over too many layers on can use jumping knowledge [Xu et al., 2018]. For this, we concatenate the output of the different layers for a node before pooling. Then the final representation of node v is equivalent to $[h_v^{(1)}, \dots, h_v^{(T)}]$.

2.5 How Powerful are Message Passing Graph Neural Networks?

In 2019, Xu et al. [2019] and Morris et al. [2019] proposed to study the expressive power of MPNNs through the lens of the graph isomorphism problem. Recall that the graph isomorphism problem is to decide whether two given graphs are isomorphic [Babai, 2016]. MPNNs are permutation invariant as all functions that involve multiple nodes treat these either as sets or multisets, thus MPNNs ignore any ordering of nodes. This means that an MPNN will necessarily map two isomorphic graphs to the same output. However, for two non isomorphic graphs the output can be different. Thus, if an MPNN maps two graphs to different outputs, we know that they are not isomorphic. If they get mapped to the same output, then nothing can be said about their isomorphism. [Xu et al., 2019]

With this we can define the expressiveness of algorithms with respect to distinguishing graphs. To shorten our notation we will use the term *expressiveness* without explicitly stating *with respect to distinguishing graphs* if it is clear from context. We say that an algorithm A is at *least as expressive* as algorithm B if A can distinguish every pair of graphs that B can distinguish. Additionally, if A is at least as expressive as B and can distinguish at least one more pair then is is *more expressive* than B. Finally, A and B are equally expressive if A is at least as expressive as B and vice-versa.

Weisfeiler-Leman algorithm. A standard algorithm to compare MPNNs to is the *Weisfeiler-Leman* (WL) algorithm [Weisfeiler and Leman, 1968]. Before we can define WL, we need to define labeled graphs and node colorings. A *labeled* graph assigns to each node a color from an alphabet Σ. Formally, a *node coloring* is a function $c : V(G) \to \Sigma$ for a sufficiently large alphabet Σ. We use c_v to denote the color assigned to vertex v by c. The WL algorithm is a *color refinement algorithm.* Given a labeled graph with node coloring l an iteration of WL will compute a new node coloring l'. Two vertices are assigned the same colors by l' only if they are assigned the same color by l. This means that with each iteration fewer vertices are assigned the same color. We formally define WL. In each iteration $t > 0$ the algorithm computes a color $c_v^{(t)}$ for node $v \in V(G)$ [Jegelka, 2022]:

1. Let X be the possibly empty node features of G. If the graph has node features then the initial color of vertex v is defined as $c_v^{(0)} = X(v)$, otherwise all vertices are assigned the same initial color.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

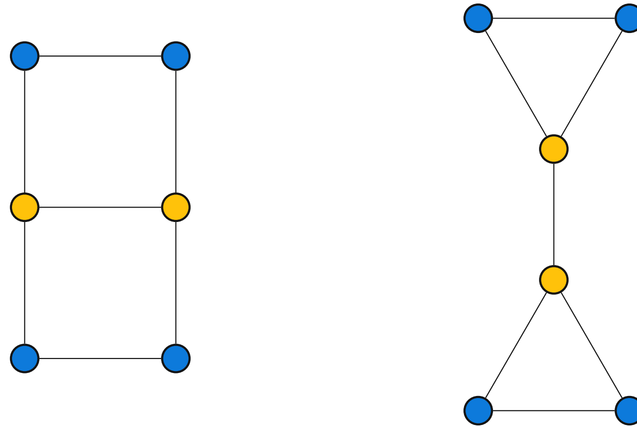

Figure 2.3: Two graphs that cannot be distinguished be the WL algorithm or any MPNN. Node colors represent colors assigned by the WL algorithm.

2. For a node v in iteration t its color is defined by

$$c_v^{(t)} = \text{HASH}\left(c_v^{(t-1)}, \left\{\left\{c_w^{(t-1)} \mid w \in \mathcal{N}_G(v)\right\}\right\}\right),$$

   where HASH is an injective hash function that maps to Σ.

3. The algorithm stops if the coloring is *stable*. That is, if at iteration $t-1$ and t the number of colors is the same $\left|\left\{c_v^{(t-1)} \mid v \in V(G)\right\}\right| = \left|\left\{c_v^{(t)} \mid v \in V(G)\right\}\right|$.

WL is guaranteed to terminate within at most $|V(G)|$ iterations [Jegelka, 2022]. Note that WL is usually not defined with node features. We use node features as they are used in many real-world datasets, and allow for a great flexibility in modeling problems with graphs.

To decide whether two graphs G, G' are non isormorphic, the WL algorithm is run on each of them. If at some iteration t their multisets of colors diverge $\left\{\left\{c_v^{(t)} \mid v \in V(G)\right\}\right\} \neq \left\{\left\{c_w^{(t)} \mid w \in V(G')\right\}\right\}$ then the graphs are non isomorphic. Otherwise, the result is not conclusive. [Jegelka, 2022]

Let us now return to MPNNs. Xu et al. [2019] and Morris et al. [2019] proved that any MPNN is at most as expressive as the WL algorithm.

Theorem 1 (Xu et al. [2019] and Morris et al. [2019])**.** *MPNNs are at most as expressive as the WL algorithm.*

This has far reaching consequences for MPNNs. Let F be any MPNN. Theorem 1 implies that there exist non-isomorphic pairs of graphs G, G' such that *any* MPNN F will output the same result meaning $F(G) = F(G')$. Two such graphs are shown in Figure 2.3. This result is independent on the architecture of the MPNN, or on what kind of data it was trained on. This shows that MPNNs are limited, which serves as a motivation to find more expressive GNNs.

As a more concrete examples consider tasks on molecules. Obviously, the structure of a molecule has an impact on its behavior and effects. This is especially true for *cycles* in its structure. Thus, if we want to design GNNs that work well with molecules they might need

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Wien Bibliothek Your knowledge hub

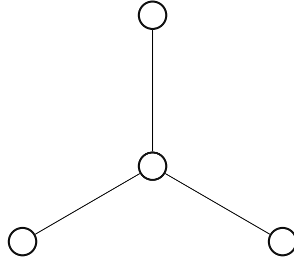

Figure 2.4: A star shaped graph with four vertices.

to be able to count patterns in a graph, that is count the number of (induced) subgraphs that are isomorphic to some given pattern (such as a cycle). However, MPNNs are unable to count the number of induced subgraph patterns of any pattern with at least three nodes. Additionally, MPNNs can only count the number of subgraph patterns of star shaped patterns or pairs of disjoint edges [Chen et al., 2020]. A star shaped graph with $k+1$ vertices is defined as $G = (\{v_0, v_1, \ldots, v_k\}, \{\{v_0, v_1\}, \{v_0, v_2\}, \ldots, \{v_0, v_k\}\})$ [Diestel, 2005], an example of a star shaped graph with 4 vertices can be found in Figure 2.4. This shows that MPNNs are too weak for many graph based problems that we want to solve.

GIN. To show the upper bound of expressivness for MPNNs, Xu et al. [2019] introduced the *graph isomorphism network* (GIN). GIN is provably the most expressive MPNN and equally expressive as the WL algorithm. GIN computes a node embedding in layer t with the function

$$h_v^{(t)} = \phi^{(t)} \left(\left(1 + \epsilon^{(t)}\right) \cdot h_v^{(t-1)} + \sum_{u \in \mathcal{N}_G(v)} h_u^{(t-1)} \right). \tag{2.9}$$

In this equation, $\phi^{(t)}$ is a multilayer perceptron and $\epsilon^{(t)}$ is scalar that is either learnable or fixed. In this section, we have used $x^{(t)}$ to denote x in the t-th iteration or layer. To simplify notation, following sections will denote this without brackets as x^t.

All proofs of expressiveness guarantees for GNNs always put requirements on the GNN. In the case of GIN, the expressiveness proof only holds if it has *sufficiently* many layers. For more general GNNs, there are also requirements such as injectivity on the aggregation function f_{Agg} and on the readout function. [Xu et al., 2019]

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 3

Related Work

In this chapter we given an overview of the related literature. We begin by focusing on machine learning on graphs through graph kernels (Section 3.1). Then, we move on to graph neural networks (Section 3.2) and finally on works that focus on the expressiveness of GNNs (Section 3.3).

3.1 Graph Kernels

In this first part we focus on graph classification with graph kernels. As graph kernels are not the focus of this thesis, this section will only focus on the key concepts of this area. We base this section on the survey Kriege et al. [2020]. A *graph kernel* is a function that measures similarity between graphs. Combining a graph kernel with a support vector machine or another kernel machine allows us to learn graphs structured data and solve tasks like graph classification. Formally, let $\mathcal{K}$ be a non-empty domain, for example the $\mathbb{R}^d$ or the set of graphs. Let $k : \mathcal{K} \times \mathcal{K} \to \mathbb{R}$ be a function. We say that k is a *kernel* on $\mathcal{K}$, if there exists a Hilbert space $\mathcal{H}_k$ with inner product $\langle \cdot, \cdot \rangle$ and a feature map $\phi : \mathcal{K} \to \mathcal{H}_k$ such that for any $x, y \in \mathcal{K}$ it holds that $k(x, y) = \langle \phi(x), \phi(y) \rangle$. As an example, consider $\mathcal{K} = \mathbb{R}^d$ and $\phi(x) = x$. Then, the kernel is equal to the dot product $k(x, y) = x^\top y$, where $x^\top$ is the transpose of x. We say that a kernel $k : \mathcal{K} \times \mathcal{K} \to \mathbb{R}$ is a graph kernel, if $\mathcal{K}$ is a non-empty set of graphs. [Kriege et al., 2020]

As graph kernels measure similarity between graphs, it is important that they are permutation invariant. Methods similar to graph kernels were first developed in the field of chemoinformatics to study the chemical properties of molecules as early as in 1973 [Adamson and Bush, 1973]. In this field, *fingerprints* where used to represent molecules by feature vectors. The term *graph kernel* was first used in 2003 by Gärtner et al. [2003] and Kashima et al. [2003] who both proposed random walk kernels. This type of kernel counts the number of walks that two graphs have in common. The random walk kernel considers all possible walks, other works focused on kernels with walks of a fixed (maximum) length (Borgwardt et al. [2005], Harchaoui and Bach [2007], Kriege et al. [2014, 2019]). Another type of graph kernel is the shortest path kernel which considers the shortest paths between all pairs of vertices in two graphs [Borgwardt and Kriegel, 2005]. In 2015, Hermansson et al. [2015] improved the shortest path

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

kernel, by additionally counting the *number* of shortest paths between any pairs of vertices. As the number of shortest paths is a byproduct of using Dijkstra's algorithm to compute the shortest paths, incorporating it does not cause much additional computation cost [Hermansson et al., 2015]. [Kriege et al., 2020]

The vertex label kernel compares labels of all vertex pairs from two different graphs completely ignoring the graph structure. An alternative approach is that of counting subgraph patterns. The graphlet kernel [Sherashidze et al., 2009] counts the number subgraphs that are isomorphic to an isomorphism type up to some fixed size (graphlets). [Kriege et al., 2020]

Another type of graph kernels uses neighborhood aggregation, for this, we iteratively assign labels to vertices based on the labels of their neighbors. In Section 2.5 we have seen the Weisfeiler-Leman algorithm which uses a neighborhood aggregation algorithm. Long before the connection between message passing graph neural networks and WL was known, WL was was used by Shervashidze et al. [2011] to create graph kernels. The Weisfeiler-Leman subtree kernel [Shervashidze et al., 2011] concatenates the counts of all colors obtained by the WL algorithm within a fixed number of iterations to obtain a vector representation $\phi_{\mathrm{WL}}(G)$ of a graph G. The kernel $k_{\mathrm{WL}}(G, H)$ then corresponds to the inner product between the vector representations of two different graphs $k_{\mathrm{WL}}(G, H) = \langle \phi_{\mathrm{WL}}(G), \phi_{\mathrm{WL}}(H) \rangle$. The Weisfeiler-Leman edge kernel [Shervashidze et al., 2011] counts the colors of the two vertices incident to an edge, instead of the color of vertices. The Weisfeiler-Leman shortest-path kernel [Shervashidze et al., 2011] sums shortest-path kernels on graphs labeled in different iterations of WL. The k-dimensional variant of the WL algorithm can also be used to create graph kernels [Morris et al., 2017]. In Section 6.3 we combine the WL shortest path and WL subtree kernel with one of our novel graph transformations and compare them to state-of-the-art GNNs. This experiment shows that WL kernels can still achieve results competitive to GNNs. [Kriege et al., 2020]

Graph kernels can also be defined by using matchings between the vertices of two graphs, based on maximizing the similarity between the matched pairs of vertices computed via another kernel. An example of this is the optimal assignment kernel [Fröhlich et al., 2005]. However, this approach only yields a valid graph kernel for very specific choices of kernel used to compute the similarity between vertices. Kernels for which this works are strong base kernels [Kriege et al., 2016]. One such base kernel with good performance on graph classification tasks is the Weisfeiler-Leman optimal assignement kernel [Kriege et al., 2016] which was obtained from WL. [Kriege et al., 2020]

3.2 Graph Neural Networks

With graph kernels, we have seen a classical approach of machine learning on graphs. We will now focus on graph neural networks. We base this section on the survey Wu et al. [2021]. Recently, GNNs have exploded in popularity[1]. Neural networks were first applied to graphs in 1997 [Sperduti and Starita, 1997]. This paper introduced the generalized recursive neuron and applied it to logical classification problems represented as directed acyclic graphs. Other early works on GNNs were by Gori et al. [2005], Scarselli et al. [2009], Gallicchio and Micheli [2010].

[1]See for example the recent ICLR 2022 conference. According to `github.com/fedebotu/ICLR2022-OpenReviewData` "graph neural network" was the fourth most commonly used keyword. Additionally, multiple papers about GNNs were accepted for oral presentations such as Geerts and Reutter [2022] which won an "outstanding paper" award and Topping et al. [2022] which got a honorable mention.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

All of these works use recurrent neural networks which are known to be computationally expensive [Wu et al., 2021]. For each node, a representation that is dependent on the neighboring nodes gets computed by the neural network. This representation gets recursively fed into the same neural network until the representations converge [Gori et al., 2005, Wu et al., 2021].

The success of convolutional neural networks in the field of computer vision motivated the use of *convolutions* for graph data. This lead to convolutional GNNs (ConvGNNs). ConvGNNs are built in layers. In each layer a representation for each node is computed that depends on the previous layer. Contrarily to early GNNs, a graph is only passed once through all the layers instead of recursively using the same layer until convergence. There are two categories of ConvGNNs: *spectral-based* and *spacial-based*. Spectral-based GNNs combine learned filters with the *graph Fourier transform* to update representations. However, spectral approaches such as the Spectral Convolutional Neural Network [Bruna et al., 2013] generally need $O(|V|^3)$ time complexity. Other methods, such as Chebyshev Spectral CNN [Defferrard et al., 2016] and the Graph Convolutional Network [Kipf and Welling, 2017] approximate the filters to reduce the runtime complexity. Spacial-based GNNs update the representation of a node using the representation of its neighbors in the graph, early works in this direction were by Scarselli et al. [2009] and Micheli [2009]. There have also been probabilistic methods built on this idea such as the *Contextual Graph Markov Model* [Bacciu et al., 2018], Diffusion-Convolutional Neural Networks [Atwood and Towsley, 2016] and other variants of diffusion on graphs. Message passing graph neural networks (MPNNs) [Gilmer et al., 2017] are a generalized framework of spatial-based GNNs. On a high level, they model graph convolutions as messages passed between neighbors in the graph. More details about MPNNs can be found in Section 2.4. [Wu et al., 2021]

Graph Attention Networks (GAT) [Veličković et al., 2018] combine the *attention mechanism* with graph neural networks. This allows the neural network to reweigh the importance of neighboring nodes. Four years later, Brody et al. [2022] investigated the attention mechanism in GAT. They found that GAT's attention mechanism was very limited as it only computes *static attention* making it in some cases it even difficult to fit the training data. They proved that by simply switching the order of two operations it is possible to obtain the more flexible *dynamic attention*.

Finally, some methods such as GraphSage [Hamilton et al., 2017] or CRAWL [Toenshoff et al., 2021] sample neighbors or paths in the graph allowing them to operate on big graphs efficiently.

3.3 Expressiveness of Graph Neural Networks

After Xu et al. [2019] and Morris et al. [2019] showed the limited expressive power of MPNNs there has been considerable effort of developing more expressive MPNNs and GNNs. As we propose the idea of using graph transformations to simplify algorithms that improve message passing, it makes sense to analyze previous works through the lens of this idea. A similar high-level idea has been proposed independently in parallel to this work, in a positional paper by Veličković [2022]. We start with improved message passing algorithms. These methods usually transform the graph to a different structure and then perform message passing on that structure. We consider this transformation as the first step of the improved message passing. In the following, we introduce three different types of such methods. The first operates on

k-tuples of nodes, the second on simplical complexes or regular cell complexes, and the third on subgraphs.

The first type of method exchanges messages between k-tuples of nodes. Higher dimensional WL (k-WL) is a generalization of WL and forms a sequence of algorithms that are stronger than WL [Immerman and Lander, 1990]. Increasing k increases the expressiveness at the cost of the runtime. Morris et al. [2019] introduced k-dimensional GNNs which extend the concept of k-WL to GNNs. For a sufficiently powerful neural network, k-GNNs have equal expressiveness as k-WL [Morris et al., 2019]. However, the runtime of k-GNNs increases exponentially with k. To combat this, Morris et al. [2020b] introduced (local) δ-k dimensional WL and GNNs. These algorithms use the sparsity of graphs to improve the runtime and expressiveness. More information about the (local) δ-k dimensional WL can be found in Section 4.4. Note that there exists another variant of k-WL called *folklore* k-WL (k-FWL). This algorithm works similar to k-WL but swaps the order of aggregation steps [Jegelka, 2022]. Unfortunately, many papers confusingly refer to the k-FWL algorithm as k-WL with just a short remark that they mean k-FWL. Thus, when reading papers it is important to keep track which variant of WL the authors are using. This thesis does *not* use k-FWL.

The second type of method operates on simplical complexes or regular cell complexes. The two most prominent examples of this idea are Simplical Networks [Bodnar et al., 2021b] and CW Networks [Bodnar et al., 2021a] (see Section 4.2). Other algorithms that work on these structures are Simplical Neural Networks [Ebli et al., 2020], Dist2Cycle [Keros et al., 2022], and Cell Complex Neural Networks [Hajij et al., 2020]. The third type of method decomposes the graph into subgraph, such as Automorphism-based Neural Networks (Autobahn) [Thiede et al., 2021] or Equivariant Subgraph Aggregation Networks (ESAN) [Bevilacqua et al., 2022] (see Section 4.3).

Next, we consider algorithms that can be seen as applying a graph transformation. It has been proven that adding random features to vertices improves the expressiveness of GNNs [Abboud et al., 2021, Dasoulas et al., 2020, Sato et al., 2021]. GNNs with random features are universal, meaning that they can learn any function defined on a graph [Abboud et al., 2021]. However, to do this they sacrifice permutation invariance and equivariance. This means that permuting the graph will change the result of these algorithms. Many permutation invariant approaches extend the graph features by counting patterns. Barceló et al. [2021] extend GNNs with rooted homomorphism counts of a set of patterns (see Section 4.1). They prove multiple theorems relating their GNN and the choice of patterns to the k-WL test. Graph Structural Networks (GSN) [Bouritsas et al., 2020] introduce a new graph convolution layer which extends messages with subgraph isomorphism counts. While this cannot be directly seen as transforming the graph, it is very similar to adding these subgraph isomorphism counts to the vertex features.

Recently, Geerts and Reutter [2022] have developed a tensor language to model GNNs. By expressing a known GNN in this language, one obtains upper bounds in expressiveness with respect to k-WL.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

CHAPTER 4

More Expressive GNNs

In this chapter we introduce four graph neural networks (GNNs) which are based on message passing graph neural networks (MPNNs) but are more expressive than MPNNs and the Weisfeiler-Leman algorithm (WL). We introduce MPNNs with homomorphism counts (Section 4.1), CW Networks (Section 4.2), Equivariant Subgraph Aggregation Networks (Section 4.3), and (local) δ-k-dimensional WL (Section 4.4). Note that all of this methods do not make use of edge features.

The usual first step of developing GNNs that are more expressive than WL, is to introduce a stronger variant of WL. This variant is used to compare the expressiveness to WL and thus also to MPNNs, as MPNNs are at most as expressive as WL [Xu et al., 2019]. Then one usually defines the GNN and proves that its expressiveness is upper bounded by the WL variant. This makes it possible to relate the expressiveness of the GNN to WL and MPNNs. For example, for MPNNs with homomorphism counts $\mathcal{F}$-WL is a variant of WL and $\mathcal{F}$-MPNN is the neural network equivalent, where $\mathcal{F}$ is a set of rooted graphs. $\mathcal{F}$-MPNN is at most as expressive as $\mathcal{F}$-WL, and depending on the parameter $\mathcal{F}$ either at least as expressive as WL or strictly more expressive. Additionally, one can construct an $\mathcal{F}$-MPNN that is equally expressive as the $\mathcal{F}$-WL. Thus, for a suitable $\mathcal{F}$ it holds that $\mathcal{F}$-MPNN is strictly more expressive than WL and MPNNs.

As all our theorems focus on to the expressiveness of these algorithms, we will only introduce the WL variants in detail.

4.1 MPNNs with Homomorphism Counts

We start by introducing rooted graphs and homomorphisms between rooted graphs. A *rooted graph* G^v is a graph where one vertex $v \in V_G$ is defined to be its root. Let $G = (V_G, E_G, X_G)$ and $H = (V_G, E_G, X_H)$ be two graphs. A homomorphism is a function $h : V_G \to V_H$ such that for any $\{u, v\} \in E_G$ it holds that $\{h(u), h(v)\} \in E_H$, and for any $v \in V$ it holds that $X_G(v) = X_H(h(v)))$. For rooted graphs G^v and H^w, an homomorphism h must also map v to w. We use $\hom(G, H)$ and $\hom(G^v, H^w)$ to denote the number of homomorphisms from G to H and from G^v to H^w, respectively. Figure 4.1 shows an example of homomorphisms on rooted graphs. [Barceló et al., 2021]

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

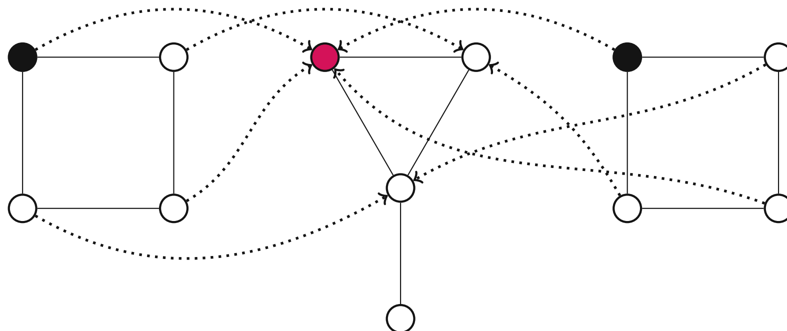

Figure 4.1: A graph (center) with all homomorphisms from a rooted 4-cycle with the black root to the red vertex in the graph.

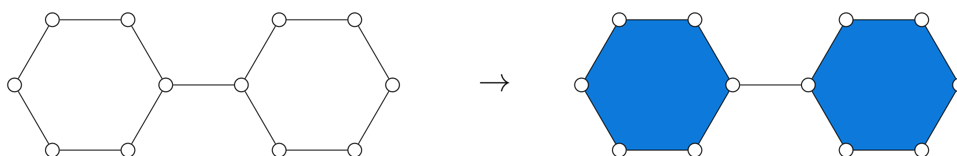

Figure 4.2: A graph (left) turned into a regular cell complex (right). Vertices and edges in the graph are 0 and 1-dimensional cells, respectively. Induced cycles in the graph are 2-dimensional cells drawn in blue. (This image is based on a similar image by Bodnar et al. [2021a])

In this section, we introduce an MPNN variant that extends the vertex features with (rooted) homomorphism counts. Let $\mathcal{F} = \{P_1^r, \dots, P_l^r\}$ be a set of (rooted) graphs. These graphs are also called *patterns*. To obtain the WL variant $\mathcal{F}$-WL, only the first step of the WL algorithm is changed:

1. Let X be the possibly empty node features of G. The initial color of vertex v is defined as $c_v^{(0)} = (X(v), \hom(P_1^r, G^v), \dots, \hom(P_l^r, G^v))$.

Similarly, for $\mathcal{F}$-MPNNs only the definition of initial node embedding gets changed:

$$h_v^{(0)} = (X(v), \hom(P_1^r, G^v), \dots, \hom(P_l^r, G^v)), \; v \in V. \tag{4.1}$$

The set of patterns $\mathcal{F}$ is a hyperparameter that has to be chosen manually. For graphs based on molecules, Barceló et al. [2021] used sets $\{C_3, \dots C_k\}$ of cycles up to size k. For graphs based on social or collaborative data they used sets $\{K_3, \dots, K_k\}$ of cliques up to size k.

Barceló et al. [2021] proved that $\mathcal{F}$-WL is more expressive than $\mathcal{F}$-MPNNs. Additionally, an $\mathcal{F}$-MPNN can be designed that is as expressive as $\mathcal{F}$-WL. They also proved that there exists no finite patterns set such that $\mathcal{F}$-WL is more expressive than k-WL, for $k > 2$.

4.2 CW Networks

Bodnar et al. [2021a] generalized the message passing paradigm from graphs to regular cell complexes. Regular cell complexes generalize the simplical complexes used by Bodnar et al. [2021b]. A regular cell complex X is a topological space with a partition of subspaces $\{X_\sigma\}_{\sigma \in P_X}$

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

called cells with an indexing set P_X. The subspace of each cell defines its dimension. The indexing set encodes all topological information about X and can be used to define a boundary relation $\prec$ between cells. Figure 4.2 shows examples of regular cell complexes.

Definition 2 (Bodnar et al. [2021a]). *For cells σ, τ the boundary relation $\sigma \prec \tau$ holds if and only if $X_\sigma \subset X_\tau$ and there is no cell δ such that $X_\sigma \subset X_\delta \subset X_\tau$.*

This boundary relation can then be used to define adjacencies between cells.

Definition 3 (Bodnar et al. [2021a]). *For a regular cell complex X and a cell $\sigma \in P_X$, we define:*

1. *The boundary adjacent cells $\mathcal{B}(\sigma) = \{\tau \mid \tau \prec \sigma\}$. These are the lower-dimensional cells on the boundary of σ. For instance, the boundary cells of an edge are its vertices.*
2. *The co-boundary adjacent cell $\mathcal{C}(\sigma) = \{\tau \mid \sigma \prec \tau\}$. These are the higher-dimensional cells with σ on their boundary. For instance, the co-boundary cells of a vertex are the edges it is part of.*
3. *The upper adjacent cells $\mathcal{N}_\uparrow(\sigma) = \{\tau \mid \exists \delta$ such that $\sigma \prec \delta$ and $\tau \prec \delta\}$. These are the cells of the same dimension as σ that are on the boundary of the same higher-dimensional cell as σ. The typical graph adjacencies between vertices are the canonical example here.*

To generalize the WL algorithm to regular cell complexes, Bodnar et al. define how to collect the colors of neighboring cells.

Definition 4 (Bodnar et al. [2021a]). *For any cells $\sigma, \tau \in P_X$ we define $\mathcal{C}(\sigma, \tau) = \mathcal{C}(\sigma) \cap C(\tau)$.*

Definition 5 (Bodnar et al. [2021a]). *Let c be a celulular coloring of X with c_σ denoting the color assigned to cell $\sigma \in P_X$. We define the following multi-sets of colors:*

1. *The colors of the boundary cells of σ : $c_{\mathcal{B}}(\sigma) = \{\!\{c_\tau \mid \tau \in \mathcal{B}(\sigma)\}\!\}$.*
2. *The upper adjacent colors of σ : $c_\uparrow(\sigma) = \{\!\{(c_\tau, c_\delta) \mid \tau \in \mathcal{N}_\uparrow(\sigma)$ and $\delta \in \mathcal{C}(\sigma, \tau)\}\!\}$.*

Cellular Weisfeiler Leman (CWL) performs message passing on cells. In each iteration of CWL the algorithm computes a coloring for each cell depending on the colors of neighboring cells in the previous iteration. We define the neighborhood of a cell σ via $c_{\mathcal{B}}(\sigma)$ and $c_\uparrow(\sigma)$. Similar to WL two regular cell complexes are not isomorphic if at some iteration the color histograms of the cells are different for the two complexes.

1. Given a regular cell complex X, all cells are initialised with the same color.
2. Given the color c_σ^t of cell σ at iteration t, we compute the color of cell σ at the next iteration c_σ^{t+1} by injectively mapping the multi-sets of colors belonging to the adjacent cells of σ using an injective HASH function:
   $c_\sigma^{t+1} = \text{HASH}\left(c_\sigma^t, c_{\mathcal{B}}^t(\sigma), c_\uparrow^t(\sigma)\right)$.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

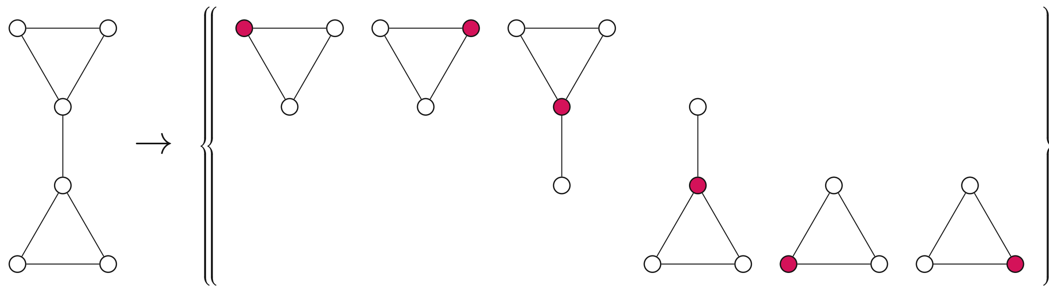

Figure 4.3: A graph (left) transformed into bags of subgraphs (right) by a policy that computes 1-ego-networks. Each subgraph is the induced 1-hop neighborhood of the red colored vertex.

3. The algorithm stops when a stable coloring is reached. Two cell complexes are considered non-isomorphic if their color histograms are different. Otherwise, the test is inconclusive.

This definition is not able to use features assigned to cell complexes. To stay consistent with out definition of WL we will use cell features for the initial coloring of each cell if they exist.

To apply the concept of regular cell complexes to graphs, Bodnar et al. [2021a] define the concept of a *cellular lifting map*, a function f that transforms a graph to a regular cell complex such that two graphs G_1, G_2 are isomorphic if and only if $f(G_1), f(G_2)$ are isomorphic. They prove that lifting maps called *skeleton preserving lifting maps* together with CWL are at least as expressive as WL. Typically, lifting maps create cells out of vertices, together with cells that encode other structures such as induced cycles or cliques. Figure 4.2 shows an example of the lifting map k-IC. This map transforms all vertices, edges and induced cycles of length at most k into a cell. Another lifting map is k-CL, this map transforms all vertices, cells and cliques of size at most k into a cell.

Bodnar et al. [2021a] define CW Networks which combine neural networks with cellular message passing, similar to graph neural networks with message passing. With a sufficiently powerful neural network CW Networks are equally expressive as CWL. Thus, by lifting graphs to cell complexes and then using a CW Network one can obtain algorithms that are strictly more expressive than WL.

4.3 Equivariant Subgraph Aggregation Networks

With CW Networks we have seen an algorithm that performs message passing on cell complexes. Equivariant Subgraph Aggregation Networks [Bevilacqua et al., 2022] (ESAN) perform message passing on a multiset of subgraphs (subgraph bags). To apply this to graphs, a policy first transforms the graphs to subgraph bags. Depending on the policy, this can yield algorithms that are strictly more expressive than WL. As there are no restrictions on the policy, this is a very general method that can be adapted closely to the specific problem. Figure 4.3 shows an example of a policy that computes k-ego-networks, that is, the induced k-hop neighborhood for each node in the graph.

To analyze the theoretical expressiveness of this approach Bevilacqua et al. [2022] present DSS-WL, a stronger variant of WL. DSS-WL takes a graph G and a policy π of computing

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Wien Bibliothek Your knowledge hub

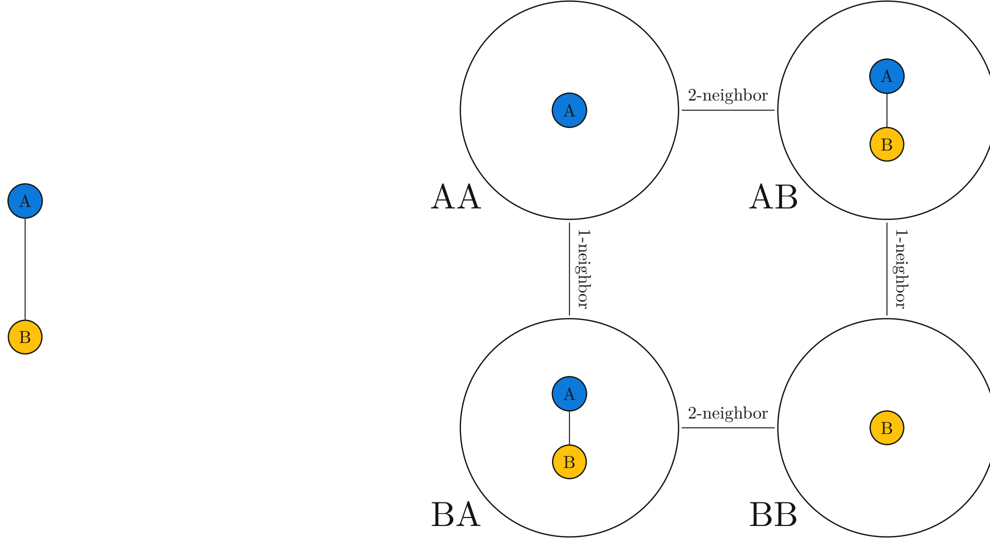

Figure 4.4: A graph (left) and a representation of δ-k-WL for $k = 2$ on this graph (right). Big nodes represent 2-tuples with the relevant subgraphs drawn inside. Edges between big nodes represent adjacencies between tuples, the type of j-neighborhood is written on the edges. Note that all tuples are local neighbors. If we remove the edge $\{A, B\}$ from the graph, then all tuples are global neighbors.

the subgraph bags. In each iteration the algorithm computes a color $c_{v,S}$ for each node v in each subgraph S.

1. The policy π is applied to G to obtain the bag of subgraphs. If G has vertex features then the initial color of a vertex are set to its features. Otherwise, all vertices are assigned the same color.

2. Given the color $c^t_{v,S}$ of vertex v in subgraph $S \in \pi(G)$ in iteration t. The color in the next iteration is defined by $c^{t+1}_{v,S} = \text{HASH}\left(c^t_{v,S}, N^t_{v,S}, C^t_v, M^t_v\right)$. Where $N^t_{v,S} = \left\{\!\!\left\{c^t_{x,S} \mid x \in \mathcal{N}_S(v)\right\}\!\!\right\}$ is the multiset of colors of neighbors of v in subgraph S, $C^t_v = \left\{\!\!\left\{c^t_{v,S'} \mid S' \in \pi(G_i) \text{ and } v \in V(S')\right\}\!\!\right\}$ is the multiset of v's color across the different subgraphs, and $M^t_v = \{\!\{C^t_x \mid x \in \mathcal{N}_{G_i}(v)\}\!\}$ is the multiset of all C^t_x where x is a neighbor of v in the original graph G.

3. After each iteration a color c_S is assigned to each subgraph S that encodes the multiset of node colors in S. Two graphs are non isomorphic if the multiset of these colors for the two graphs diverge. If the color refinement algorithm converges on all subgraphs, then the test is inconclusive.

There exist policies such as the policy that computes k-ego-networks that make DSS-WL strictly more powerful than WL and MPNNs. [Bevilacqua et al., 2022]

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

4.4 δ-k-dimensional Weisfeiler-Leman

The next algorithm, δ-k-dimensional Weisfeiler-Leman (δ-k-WL) [Morris et al., 2020b] performs message passing on k-tuples of vertices, similar to k-WL [Immerman and Lander, 1990, Morris et al., 2019]. However, only δ-k-WL distinguishes between global and local neighborhoods which is denoted by the δ.

The algorithm starts by assigning to each k-tuple a color that encodes its isomorphism class, meaning that two tuples are assigned the same color if and only if there exists an isomorphism between the labeled subgraphs induced by the tuples. In each iteration of δ-k-WL, each k-tuple is assigned a color that depends on its previous color and the colors of its neighbors: Let $\mathbf{v}$ be a k-tuple of vertices from graph G and j be in $\{1, \ldots, n\}$. We use $\phi_j(\mathbf{v}, w)$ to denote the tuple obtained by replacing the j-th vertex in $\mathbf{v}$ by the vertex w. For a tuple $\mathbf{w} = \phi_j(\mathbf{v}, w)$, we say that $\mathbf{v}$ and $\mathbf{w}$ are j-neighbors. We use v_j to denote the j-th vertex in $\mathbf{v}$, that is the vertex that gets replaced by w to obtain $\mathbf{w}$. If v_j and w are neighbors in G, then $\mathbf{v}$ and $\mathbf{w}$ are local j-neighbors, otherwise they are global j-neighbors. Figure 4.4 shows an example of adjacencies between 2-tuples. The function $\mathrm{adj}(\mathbf{v}, \mathbf{w})$ returns 'L' if $\mathbf{v}$ and $\mathbf{w}$ are local j-neighbors and 'G' otherwise. We can define δ-k-WL:

1. Assign to each k-tuple a color that encodes its isomorphism class
2. Given the color $c^t_{\mathbf{v}}$ of tuple $\mathbf{v}$ in iteration t. The color in the next iteration is defined by $c^{t+1}_{\mathbf{v}} = \left(c^t_{\mathbf{v}}, M^t_{\delta,\bar{\delta}}(\mathbf{v})\right)$ where

$$M^t_{\delta,\bar{\delta}}(\mathbf{v}) = \Big(\{\!\!\{ \left(c^n_{\phi_1(\mathbf{v},w)}, \mathrm{adj}(\mathbf{v}, \phi_1(\mathbf{v}, w))\right) \mid w \in V(G) \}\!\!\}, \ldots, \{\!\!\{ \left(c^n_{\phi_k(\mathbf{v},w)}, \mathrm{adj}(\mathbf{v}, \phi_k(\mathbf{v}, w))\right) \mid w \in V(G) \}\!\!\} \Big).$$

3. The algorithm stops when a stable coloring is reached. Two graphs are non-isomorphic if their histogram of colors is different. Otherwise the test is inconclusive.

The δ-k-WL algorithm is strictly more expressive than k-WL. However, as it passes messages between global and local neighbors, it has the same runtime complexity as k-WL. To remedy this, Morris et al. [2019] propose two additional variants of δ-k-WL that make only limited use of global neighborhoods. Local δ-k-Weisfeiler-Leman (δ-k-LWL) only considers local j-neighborhoods. For this $M^t_{\delta,\bar{\delta}}(\mathbf{v})$ gets replaced by

$$M^t_{\delta}(\mathbf{v}) = \Big(\{\!\!\{ c_{\phi_1(\mathbf{v},w)} \mid w \in \mathcal{N}_G(v_1) \}\!\!\}, \ldots, \{\!\!\{ c_{\phi_k(\mathbf{v},w)} \mid w \in \mathcal{N}_G(v_k) \}\!\!\} \Big).$$

The authors proved that δ-k-WL is at least as expressive as δ-k-LWL. Another variant, δ-k-LWL$^+$ uses a similar aggregation function as $M^t_{\delta}(\mathbf{v})$. However, it supplements the color of a local j-neighbor by the *number* of j-neighbors (including global neighbors) that have the same color as the local neighbor. This means that δ-k-LWL$^+$ uses only some global information. Interestingly, δ-k-LWL$^+$ is equally expressive as δ-k-WL. There exist equally expressive neural network variant for all δ-k-WL variants.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

CHAPTER 5

Cliques: Counting Homomorphisms vs Lifting

We investigate how lifting cliques to cell complexes with the lifting map k-CL relates to counting clique homomorphisms. We show that CWL with the lifting map k-CL is at least as expressive as $\{K_3, \dots, K_k\}$-WL. This captures the intuitive notion that by lifting clique complexes we are able to count their homomorphisms. For the sake of a consistent notation with respect to $\mathcal{F}$-WL, we will denote CWL with k-CL lifting as $\{K_3, \dots, K_k\}$-CWL.

First, in Section 5.1 we show that if CWL is able to count homomorphisms then it is at least as expressive as $\mathcal{F}$-WL. This intuitively follows from the fact that counting homomorphisms is the only thing that sets $\mathcal{F}$-WL apart from WL. Then, we show in Section 5.2 that $\{K_3, \dots, K_k\}$-CWL can count cliques. Combining these two lemmas shows that $\{K_3, \dots, K_k\}$-CWL is at least as expressive as $\{K_3, \dots, K_k\}$-WL. This result also generalizes to CW Networks and MPNNs as long as suitably expressive neural network are used.

5.1 CWL and Counting Homomorphisms

We prove a theorem to simplify showing that skeleton preserving CWL with some given lifting map is at least as expressive as $\mathcal{F}$-WL with some set of patterns $\mathcal{F}$.

We start by proving a small lemma stating that for any vertex v the multiset $c^t_{\uparrow}(\{v\})$ contains the labels of all neighbors of v. This means that CWL can use of the same label propagation mechanism as WL and $\mathcal{F}$-WL.

Lemma 2. *For any graph G, vertex $v \in V(G)$ and $t > 0$ it holds that*

$$c^t_{\uparrow}(\{v\}) = \left\{\!\!\left\{ \left(c^{t-1}_{G,\{x\}}, c^{t-1}_{G,\{v,x\}}\right) \mid x \in \mathcal{N}_G(v) \right\}\!\!\right\} \cup \left\{\!\!\left\{ \left(c^{t-1}_{G,\{v\}}, c^{t-1}_{G,\{v,x\}}\right) \mid x \in \mathcal{N}_G(v) \right\}\!\!\right\}.$$

Proof. Let v be an arbitrary vertex from $V(G)$ and $t > 0$.

Since $c^t_{\uparrow}(\{v\})$ was obtained by a skeleton preserving CWL and $\{v\}$ is a 0-dimensional cell, we know that all 1-dimensional cells that have $\{v\}$ on their boundary correspond to edges that

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

are incident to v. Thus $\mathcal{N}_{\uparrow}(\{v\}) = \{x \mid x \in \mathcal{N}_G(v)\} \cup \{v \text{ if } |\mathcal{N}_G(v)| > 0\}$. Plugging $\mathcal{N}_{\uparrow}(\{v\})$ into $c^t_{\uparrow}(\{v\})$ proves the lemma. □

Now we are ready to prove the main theorem of this section. Intuitively, this theorem states that if at some iteration of CWL with some lifting map is able to compute the homomorphisms of a set $\mathcal{F}$ then it is at least as expressive as $\mathcal{F}$-WL.

Lemma 3. *For any two graphs G, H. Let $c^t_{G,\{v\}}, \chi^t_{G,v}$ denote the labels of vertex v in graph G in iteration t of skeleton preserving CWL and $\mathcal{F}$-WL respectively, where $\mathcal{F}$ is some set of patterns and CWL uses an arbitrary lifting operation. Suppose that at some iteration d it holds that $c^d_{G,\{v\}} = c^d_{G,\{w\}} \Rightarrow \chi^0_{G,v} = \chi^0_{H,w}$ for every $v \in V(G)$ and $w \in V(H)$. Then for all $t \geq 0$: $c^{d+t}_{G,\{v\}} = c^{d+t}_{G,\{w\}} \Rightarrow \chi^t_{G,v} = \chi^t_{H,w}$ for every $v \in V(G)$ and $w \in V(H)$.*

Proof. We show this by induction on t. Assume that at some iteration d it holds that $c^d_{G,\{v\}} = c^d_{G,\{w\}} \Rightarrow \chi^0_{G,v} = \chi^0_{H,w}$ for every $v \in V(G)$ and $w \in V(H)$.

Base case: $t = 0$. Let $v \in V(G)$ and $w \in V(H)$ be arbitrary vertices. We need to show that $c^d_{G,\{v\}} = c^d_{G,\{w\}} \Rightarrow \chi^0_{G,v} = \chi^0_{H,w}$. This is equivalent to the initial assumption and thus holds.

Induction hypothesis: We assume $c^{d+t}_{G,\{v\}} = c^{d+t}_{H,\{w\}} \Rightarrow \chi^t_{G,v} = \chi^t_{H,w}$.

Induction step: We assume that for arbitrary vertices $v \in V(G)$ and $w \in V(H)$ it holds that $c^{d+t+1}_{G,\{v\}} = c^{d+t+1}_{H,\{w\}}$. From this we need to show that $\chi^{t+1}_{G,v} = \chi^{t+1}_{H,w}$ or written explicitly we need to show that $(\chi^t_{G,v}, \{\{\chi^t_{G,x} \mid x \in \mathcal{N}_G(v)\}\}) = (\chi^t_{H,w}, \{\{\chi^t_{H,y} \mid y \in \mathcal{N}_H(w)\}\})$.

Since CWL refines the labeling, we know that $c^{d+t+1}_{G,\{v\}} = c^{d+t+1}_{H,\{w\}}$ implies $c^{d+t}_{G,\{v\}} = c^{d+t}_{H,\{w\}}$. Combining this with the induction hypothesis gives us $\chi^t_{G,v} = \chi^t_{H,w}$.

It remains to prove $\{\{\chi^t_{G,x} \mid x \in \mathcal{N}_G(v)\}\} = \{\{\chi^t_{H,y} \mid y \in \mathcal{N}_H(w)\}\}$. From $c^{d+t+1}_{G,\{v\}} = c^{d+t+1}_{H,\{w\}}$ it follows that $c^{d+t+1}_{\uparrow}(\{v\}) = c^{d+t+1}_{\uparrow}(\{w\})$.

From Lemma 2 it follows that

$$\left\{\left\{\left(c^{d+t}_{G,\{x\}}, c^{d+t}_{G,\{v,x\}}\right) \mid x \in \mathcal{N}_G(v)\right\}\right\} \cup \left\{\left\{\left(c^{d+t}_{G,\{v\}}, c^{d+t}_{G,\{v,x\}}\right) \mid x \in \mathcal{N}_G(v)\right\}\right\} \tag{5.1}$$

$$= \left\{\left\{\left(c^{d+t}_{H,\{y\}}, c^{d+t}_{H,\{w,y\}}\right) \mid y \in \mathcal{N}_H(w)\right\}\right\} \cup \left\{\left\{\left(c^{d+t}_{H,\{w\}}, c^{d+t}_{H,\{w,y\}}\right) \mid y \in \mathcal{N}_H(w)\right\}\right\}. \tag{5.2}$$

Thus there exists a bijection $\beta : \mathcal{N}_G(v) \to \mathcal{N}_H(w)$ such that $c^{d+t}_{G,\{x\}} = c^{d+t}_{H,\{\beta(x)\}}$ for any $x \in \mathcal{N}_G(v)$. By invoking the induction hypothesis we obtain that for any $x \in \mathcal{N}_G(v)$ it holds that $\chi^t_{G,x} = \chi^t_{H,\beta(x)}$. This implies

$$\left\{\left\{\chi^t_{G,x} \mid x \in \mathcal{N}_G(v)\right\}\right\} = \left\{\left\{\chi^t_{H,y} \mid y \in \mathcal{N}_H(w)\right\}\right\}.$$

□

This means that if at some iteration the labeling computed by CWL for two given graphs refines the initial labeling of $\mathcal{F}$-WL, then if CWL cannot distinguish those graphs then neither can $\mathcal{F}$-WL. This makes it much easier to prove expressiveness results for different patterns

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Wien Bibliothek Your knowledge hub

and lifting maps: all we need to show is that at some iteration CWL can capture the initial labeling of $\mathcal{F}$-WL. Then we can apply this lemma and have shown that for these patterns and lifting map CWL is at least as expressive as $\mathcal{F}$-WL. Intuitively, this lemma captures the fact that the only difference between $\mathcal{F}$-WL and WL is the initial labeling. As CWL is at least as expressive as WL, if it can compute the initial labeling of $\mathcal{F}$-WL then it can also perform the remaining steps of $\mathcal{F}$-WL.

5.2 Complete Graphs and Clique Complex Lifting

We prove that $\{K_3, \ldots, K_k\}$-CWL is at least as expressive as $\{K_3, \ldots, K_k\}$-WL. For this we show that $\{K_3, \ldots, K_k\}$-CWL can compute a labeling that is equivalent to the initial labeling of $\{K_3, \ldots, K_k\}$-WL. Combining this with Lemma 3 yields desired the expressiveness result. First, we build intuition by proving the simple case of $k = 3$ i.e. the case where we only count triangles. Afterwards, we prove the general case.

5.2.1 Triangles (K_3)

We start by proving the following lemma that allows us to connect the number of homomorphisms to the number of subgraphs.

Lemma 4. *Let G be an undirected loopless graph and v an arbitrary vertex. Then* $\hom(K_3^r, G^v)$ *is equivalent to the two times number of triangles that contain v.*

Proof. Let k be equal to $\hom(K_3^r, G^v)$ and let $f_1, \ldots f_k$ denote all the unique homomorphisms from a triangle K_3^r with root r to G^v. An arbitrary f_j maps the vertices of the triangle to vertices of $V(G)$ at least one of which is v. If f_j maps to only one or two vertices of $V(G)$ then this would mean that G has a loop which leads to a contradiction. If f_j maps to three vertices then these three form a triangle. Observe that for any triangle v, x, y for $x, y \in V(G)$ there exists exactly two homomorphisms from a triangle r, a, b with root r: we map r to v and then either a to x and b to y, or a to y and b to x. □

Next, we show that the label of vertex v after two iterations of $\{K_3\}$-CWL encodes the number of triangles that contain v. In what follows, we assume that each cell corresponding not to a vertex is assigned the initial label 1.

Lemma 5. *Let G be a graph and $v \in V(G)$. Then after two iteration of $\{K_3\}$-CWL the cell $\{v\}$ is assigned the label* $c^2_{\{v\}} = \text{HASH}\left(c^1_{\{v\}}, \emptyset, c^1_{\mathcal{C}}(\{v\}), \emptyset, c^1_{\uparrow}(\{v\})\right)$ *with*

$$c^1_{\mathcal{C}}(\{v\}) = \{\!\{\text{HASH}\left(1, \cdot, \{\!\{1 \mid \textit{for each } K_3 \textit{ containing } \{v, w\}\}\!\}, \cdot, \cdot\right) \mid w \in \mathcal{N}_G(v)\}\!\}\,.$$

Proof. Let $G = (V, E)$ be a graph with its indexing set P_G. We execute the $\{K_3\}$-CWL algorithm on P_G. We denote the initial colors as belonging to the 0th iteration. We use $\tilde{c}^t_\sigma$ to denote the input into the HASH function to get the color c^t_σ, i.e. $c^t_\sigma = \text{HASH}(\tilde{c}^t_\sigma)$.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

First iteration. We compute $\tilde{c}^1_\sigma$ the input into the hash function after the first iteration for any cell σ that corresponds to a vertex or and edge in the graph.

- Any vertex v will be assigned the label:

$$\tilde{c}^1_{\{v\}} = \left(\chi_G(v), \emptyset, \{\!\{1 \mid w \in \mathcal{N}_G(v)\}\!\}, \emptyset, c^0_\uparrow(\{v\})\right).$$

- Any edge $\{v, w\}$ will be assigned the label:

$$\tilde{c}^1_{\{v,w\}} = \left(1, \{\!\{\chi_G(v), \chi_G(w)\}\!\}, \{\!\{1 \mid \text{for each } K_3 \text{ containing } \{v,w\}\}\!\}, c^0_\downarrow(\{v,w\}), c^0_\uparrow(\{v,w\})\right).$$

We can see that the label of an edge cell encodes the number of triangles containing that edge $\{v, w\}$.

Second iteration. We compute $\tilde{c}^2_{\{v\}}$ the input into the hash function of any vertex v after the second iteration:

$$\tilde{c}^2_{\{v\}} = \left(c^1_{\{v\}}, \emptyset, \left\{\!\!\left\{c^1_{\{v,w\}} \mid w \in \mathcal{N}_G(v)\right\}\!\!\right\}, \emptyset, c^1_\uparrow(\{v\})\right).$$

The multiset at the third position of the tuple $\tilde{c}^2_{\{v\}}$ is $c^1_{\mathcal{C}}(\{v\})$. Let us now examine this more closely. Plugging in the result of the first iteration gives us

$$cc^1_{\mathcal{C}}(\{v\}) = \{\!\{\text{HASH}\left(1, \cdot, \{\!\{1 \mid \text{for each } K_3 \text{ containing } \{v,w\}\}\!\}, \cdot, \cdot\right) \mid w \in \mathcal{N}_G(v)\}\!\}.$$

□

Next we show that $\{K_3\}$-CWL can compute compute a labeling that refines the initial labeling of $\{K_3\}$-WL.

Lemma 6. *For any two graphs G, H. Let $c^2_{G,\{v\}}, \chi^2_{G,v}$ denote the labels of vertex v in graph G after two iterations iteration of $\{K_3\}$-CWL and $\{K_3\}$-WL respectively. Then it holds that $c^2_{G,\{v\}} = c^2_{G,\{w\}} \Rightarrow \chi^0_{G,v} = \chi^0_{H,w}$ for any $v \in V(G)$ and $w \in V(H)$.*

Proof. Suppose $c^2_{G,\{v\}} = c^2_{G,\{w\}}$. We want to show that $(\chi_G(v), \hom(K_3, v)) = (\chi_H(w), \hom(K_3, w))$. Each iteration of CWL refines the labels of the previous iteration implying $c^0_{G,\{v\}} = c^0_{G,\{w\}}$ and thus $\chi_G(v) = \chi_H(w)$. Furthermore, $c^2_{G,\{v\}} = c^2_{G,\{w\}}$ implies that $c^1_{\mathcal{C}}(\{v\}) = c^1_{\mathcal{C}}(\{w\})$. By Lemma 5 and the fact that HASH is injective we know that

$$\begin{aligned}&\{\!\{\{\!\{1 \mid \text{for each } K_3 \text{ containing } \{v,x\}\}\!\} \mid x \in \mathcal{N}_G(v)\}\!\} = \\ &\{\!\{\{\!\{1 \mid \text{for each } K_3 \text{ containing } \{w,y\}\}\!\} \mid y \in \mathcal{N}_H(w)\}\!\}.\end{aligned}$$

For any triangle that contains v there are two edges incident two v that are part of the triangle. Hence the number of 1s in the multiset above is equivalent to two times the number of triangles that contain v or w. By Lemma 4 this implies $\hom(K_3, G^v) = \hom(K_3, H^w)$. □

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek, WIEN Your knowledge hub

5.2.2 The General Case $(K_3, \ldots, K_k)$

To begin we show that $\hom(K_k^r, G^v)$ does not depend on homomorphisms from smaller graphs than K_k.

Lemma 7. *Let G be an undirected loopless graph and v an arbitrary vertex. Then for any $k \geq 3$ it holds that $\hom(K_k^r, G^v)$ is equivalent to the $(k-1)!$ times the number of subgraphs that contain v and are isomorphic to a K_k.*

Proof. Let n be equal to $\hom(K_k^r, G^v)$ and let $f_1, \ldots f_n$ denote all the unique homomorphisms from K_k^r to G^v. An arbitrary f_j maps the vertices of K_k^r to vertices of $V(G)$. Note that r always gets mapped to v. Suppose f_j maps to less than k. Then two vertices of K_k^r get mapped to the same vertex. Since K_k^r is a complete graph, this would mean that G has a loop which leads to a contradiction. Thus any f_j maps to k vertices that form a clique.

Now consider any subgraph of G containing v that is isomorphic to a K_k. How many homomorphisms from K_k^r to this subgraph do exist? By the definition of homomorphisms for rooted graphs, we know that r must be mapped to v. However, all other $k-1$ vertices can be freely mapped to any of the remaining $k-1$ vertices from the subgraph. Hence, there exist $(k-1)!$ homomorphisms. □

Now, we show that $\{K_3, \ldots, K_k\}$-CWL can count homomorphisms from cliques of size up to k.

Lemma 8. *Let G, H be a graphs and $v \in V(G), w \in V(H)$. Let $c^j_{G,\{v\}}, c^j_{H,\{w\}}$ be the labels assigned to vertices v, w in the j-th iteration of $\{K_3, \ldots, K_k\}$-CWL with $k \geq j \geq 3$. Then $c^j_{G,\{v\}} = c^j_{H,\{w\}}$ implies that $\hom(K_j^r, G^v) = \hom(K_j^r, H^w)$.*

Proof. First, let the cell $Y = \{y_1, \ldots, y_l\} \subseteq V(G)$ correspond to a set of at least four vertices that form a clique. Consider, $c_{\mathcal{B}}(Y)^t$ this multiset contains the labels of all cells that correspond to a subset of size $l-1$ of Y that also form a clique. In total there are l such subsets.

Let $k \geq j \geq 3$. Let the cell X be a set of vertices $X = \{v, x_1, \ldots, x_{j-1}\}$ that form a clique. Per definition the label of a cell that does not correspond to a vertex is initialized to 1. Consider the multiset $c^t_{\mathcal{B}}(X)$, it contains the label of all lower dimensional cells with X on their boundary. These cells corresponds a subsets of vertices from X that form a clique of size $j-1$, in total there are j such cells. Thus $c^1_X = (1, \{\!\{1 \mid _ \in \{1, \ldots, j\}\}\!\}, \cdot, \cdot, \cdot)$. Now, let us investigate how the label c^1_X gets propagated towards $\{v\}$. As we have we have seen, in every iteration it flows through the set of lower boundaries: First into the $c_{\mathcal{C}}$ multisets of the j cells that correspond to cliques of size $j-1$. Then from each of those it flows to the $c_{\mathcal{C}}$ multisets of the $j-1$ cells that correspond to cliques of size $j-2$ and so on. Thus after j iterations it has flown into the vertex v.

Assume $c^j_{G,\{v\}} = c^j_{H,\{w\}}$, then also $\tilde{c}^j_{G,\{v\}} = \tilde{c}^j_{H,\{w\}}$. Let us now recursively unfold the sets $c_{\mathcal{C}}$ of $\tilde{c}^j_{G,\{v\}}$: that is for we collect all labels in $c^j_{\mathcal{C}}(\{v\})$ and unroll their $c_{\mathcal{C}}$ multisets and so on. We repeat this for the elements in $c_{\mathcal{C}}$ recursively, in total $j-1$ times (including the unraveling of $\tilde{c}^j_{G,\{v\}}$) and call this α_G and α_H for the two graphs. Obviously $\alpha_G = \alpha_H$. This means that for every element in the lowest level of α_G that looks like $(1, \{\!\{1 \mid _ \in \{1, \ldots, j\}\}\!\}, \cdot, \cdot, \cdot)$ there

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

exists the same element in the lowest level of α_H. If we can show that subgraphs that contain v and w are isomorphic to a K_j create these elements, we can apply Lemma 7 to obtain that $\hom(K_j^r, G^v) = \hom(K_j^r, G^w)$.

Since these elements are $j-1$ levels deep they must have been created in either iteration 0 or 1. However, they cannot have been created in iteration 0, because then they would either be 1 or some vertex feature. Thus they must have been created in iteration 1. Then $(1, \{\{1 \mid _ \in \{1, \ldots, j\}\}\}, \cdot, \cdot, \cdot)$ corresponds to $c^0_{\mathcal{B}}(\sigma)$ of some cell σ. This implies that the given cell has j lower dimensional cells on the boundary of σ. All cells correspond to either vertices, edges or cliques. The first two have no or two lower dimensional cells. Because $j \geq 3$ this means that σ corresponds to a clique of size at least 3. Any clique of size l has exactly l lower dimensional cells on its boundary. Thus implying that all elements of shape $(1, \{\{1 \mid _ \in \{1, \ldots, j\}\}\}, \cdot, \cdot, \cdot)$ have been created by a clique of size j. □

The central theorem of this section follows directly from Lemma 3 and Lemma 8.

Theorem 9. *$\{K_3, \ldots, K_k\}$-CWL is at least as expressive as $\{K_3, \ldots, K_k\}$-WL.*

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

CHAPTER 6

Reducing Learning on Cell Complexes To Graphs

In this chapter we show that is possible to reduce an improved message passing algorithm to a graph transformation combined with message passing. For this, we focus on CW Networks [Bodnar et al., 2021a] (see Section 4.2). In Section 6.1, we propose *cell encoding* a novel graph transformation that transforms any regular cell complex to a graph. We prove that cell encoding together with WL is as expressive as cellular WL. This implies that combining a suitably expressive MPNN with cell encoding is at least as expressive as any CW Network. In Section 6.2, we show how this can be used to build more expressive GNNs. Finally, in Section 6.3 we present a set of experiments on graph classification tasks. These experiments show that our method achieves state-of-the-art results that are competitive to the CW Network CIN. Additionally, our results show that cell encoding improves the results of GNNs and graph kernels.

6.1 Cell Encoding

We propose *cell encoding*, a novel algorithm that transforms any regular cell complex X to a graph G_X. A similar construction for a type of regular cell complexes called simplical complexes is already known to the topology community [Grigor'yan et al., 2014]. We show that cell encoding combined with WL is at least as expressive as CWL in distinguishing regular cell complexes. With this, we can perform message passing on graphs instead of cell complexes while keeping the expressiveness guarantees from CWL. However, this approach is not limited to cell complexes obtained with a cellular lifting map. Indeed, any cell complex can be transformed into a graph while ensuring that WL is as expressive least as CWL.

Definition 6 (Cell Encoding)**.** *Given a regular cell complex X with a finite indexing set P_X, cell encoding transforms P_X into a graph $G_X = (V_X, E_X)$ with vertex features. Where*

$$V_X = P_X,$$

$$E_X = \{\{\tau,\delta\} \mid \tau,\delta \in P_X,\ \tau \prec \delta \text{ or } \delta \prec \tau\} \cup \{\{\tau,\delta\} \mid \exists\sigma \in P_X,\ \tau \prec \sigma,\ \delta \prec \sigma\}.$$

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

For unlabeled regular cell complexes we introduce a feature that encodes the dimension of each cell. For labeled regular cell complexes we extend the features correspondingly.

Encoding the dimension of a cell in vertex features can be done via one-hot encoding and we use it to distinguish between cells of different dimensions. An example of cell encoding can be found in Figure 6.1.

Theorem 10. *Cell encoding together with WL is as least as expressive as CWL.*

Proof. Let P_X, P_Y be the indexing sets of two regular cell complexes. Let G_X and H_Y be the graphs obtained by applying cell encoding to P_X and P_Y. We use π to denote the stable coloring obtained by WL to G_X, H_Y and c^t to denote the color obtained by CWL on P_X, P_Y after iteration t. Thus π_σ denotes the colors assigned to *vertex* σ by WL and c^t_σ denotes the color assigned to *cell* σ by CWL. We assume that WL with cell encoding cannot distinguish G_X and H_Y. From this we show that for every iteration $t \geq 0$ of CWL it holds that:

For all $\tau \in V(G_X), \sigma \in V(H_Y)$ with $\pi_\tau = \pi_\sigma$ it holds that $c^t_\tau = c^t_\sigma$.

Note that this is equivalent to showing that if WL with cell encoding cannot distinguish G_X and H_Y then CWL cannot distinguish P_X and P_Y. When WL with cell encoding cannot distinguish G_X and H_Y, then we know that for every vertex in G_X there is a vertex in H_Y that are assigned the same color by WL. The statement then implies that there is a bijective mapping from cells of P_X to P_Y such that they are assigned the same color by CWL which means that the histogram of colors is the same for both graphs. From this it follows that CWL cannot distinguish P_X and P_Y. We show that this statement holds by induction on the iteration t of CWL.

The proof will make use of the fact that π is a stable coloring. This means that any two vertices $p \in V(G_X)$ and $q \in V(H_Y)$ assigned the same color $\pi_p = \pi_q$ would be assigned the same color in any additional iteration of WL. This implies that the multiset of colors of neighbors of p is equivalent to the multiset of colors of neighbors of q. Thus, there exists a bijective function $\alpha : \mathcal{N}_{G_X}(p) \to \mathcal{N}_{H_Y}(q)$ such that for any $x \in \mathcal{N}_{G_X}(p)$ it holds that $\pi_x = \pi_{\alpha(x)}$.

Base case: We show that the statement holds for $t = 0$. CWL initializes all cells to same color or to given cell features. As cell encoding copies the given cell features, all we need to show is that P_X and P_Y have the same number of cells. The number of vertices in G_X and H_Y is equal to the number of cells in P_X and P_Y, respectively. Since WL cannot distinguish G_X and H_Y we know that they must have the same number of vertices.

Induction hypothesis: We assume that the statements holds for n.

Induction step: We show that the statements hold for $n+1$. Let $\tau \in V(G_X), \sigma \in V(H_Y)$ be arbitrary vertices with $\pi_\tau = \pi_\sigma$. We need to show that $c^{n+1}_\tau = c^{n+1}_\sigma$. By the definition of CWL we know that $c^{n+1}_\tau = \text{HASH}\left(c^n_\tau, c^n_{\mathcal{B}}(\tau), c^n_{\uparrow}(\tau)\right)$ and $c^{n+1}_\sigma = \text{HASH}\left(c^n_\sigma, c^n_{\mathcal{B}}(\sigma), c^n_{\uparrow}(\sigma)\right)$. We will show that the inputs into the two hash functions are equal for c^{n+1}_τ and c^{n+1}_σ.

First, we show that $c^n_\tau = c^n_\sigma$. This immediately follows from the assumption $\pi_\tau = \pi_\sigma$ and the induction hypothesis.

Next, we want to show that $c^n_{\mathcal{B}}(\tau) = c^n_{\mathcal{B}}(\sigma)$. The assumption $\pi_\tau = \pi_\sigma$ implies that there exists a bijective function $\alpha : \mathcal{N}_{G_X}(\tau) \to \mathcal{N}_{H_Y}(\sigma)$ such that for any $x \in N_{G_X}(\tau)$ it holds that

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Wien Bibliothek Your knowledge hub

$\pi_x = \pi_{\alpha(x)}$. Since the initial colors encode the dimensions of the cell, this function must also respect the dimension of the cell meaning it only maps vertices to vertices whose cells have the same dimensions. This implies that for every cell μ with $\mu \prec \tau$ we know that there exists a cell $\nu = \alpha(\mu)$ with $\nu \prec \sigma$ such that $\pi_\mu = \pi_\nu$. With the induction hypothesis it follows that $c^n_\mu = c^n_\nu$. Observe, that $\mathcal{B}(\tau)$ contains only cells μ with $\mu \prec \tau$. Analogously, $\mathcal{B}(\sigma)$ contains cells ν with $\nu \prec \sigma$. Thus $c^n_{\mathcal{B}}(\tau) = c^n_{\mathcal{B}}(\sigma)$.

Finally, we need to show that $c^n_\uparrow(\tau) = c^n_\uparrow(\sigma)$. By definition we know that

$$c^n_\uparrow(\tau) = \left\{\!\!\left\{ (c^n_\mu, c^n_\delta) \mid \mu \in \mathcal{N}_\uparrow(\tau) \text{ and } \delta \in \mathcal{C}(\tau, \mu) \right\}\!\!\right\}.$$

We can rewrite this as $c^n_\uparrow(\tau) = \left\{\!\!\left\{ (c^n_\mu, c^n_\delta) \mid \tau \prec \delta \text{ and } \mu \prec \delta \right\}\!\!\right\}$. We will make use of the bijective function α defined in the paragraph above. We know that for any cell δ with $\tau \prec \delta$ there exists a vertex δ adjacent to τ such that $\pi_\delta = \pi_{\alpha(\delta)}$. This implies two things: first by using the induction hypothesis we know that $c^n_\delta = c^n_{\alpha(\delta)}$. Secondly, there exist a bijective function $\beta_\delta : \mathcal{N}_{G_X}(\delta) \to \mathcal{N}_{H_Y}(\alpha(\delta))$ that has the same properties as α. That is, for any $x \in \mathcal{N}_{G_X}(\delta)$ it holds that $\pi_x = \pi_{\beta_\delta(x)}$.

We can now put all of this together. The existence of α means that for any cell δ with $\tau \prec \delta$ there is a cell $\alpha(\delta) \in P_Y$ such that $c^n_\delta = c^n_{\alpha(\delta)}$. Next, with the existence of β_δ it follows that for each cell μ with $\mu \prec \delta$ there exists a cell $\beta_\delta(\mu) \in P_Y$ such that $c^n_\mu = c^n_{\beta_\delta(\mu)}$. With the fact that α and β_δ are bijective, it follows that $c^n_\uparrow(\tau) = c^n_\uparrow(\sigma)$. This proves the induction step and concludes the proof of Theorem 10. □

Note that this theorem does not only apply to WL or suitably expressive GNNs. It works with any algorithm that is at least as expressive as WL, for example WL kernels. This means that as long as an algorithm is at least as expressive as WL with respect to distinguishing graphs, it can use cell encoding to have at least the same expressiveness as CWL or CW Networks. This also means that cell encoding can be used to run any such algorithms on regular cell complexes.

While cell encoding together with WL is as expressive as CWL, this does not mean that it yields exactly the same result. For example, when CWL passes messages via upper adjacent cells, it adds the color of the higher dimensional cell to the message. This is not something covered by our transformation. This does not impact the expressiveness of the method, but still might lead to better results on practical applications.

6.2 Cellular Ring Encoding

In this section, we show that cell encoding can be used to build more expressive GNNs. Bodnar et al. [2021a] present cellular lifting maps such as k-IC that when combined with CWL yield algorithms strictly more expressive than WL. We focus on the lifting map k-IC that transforms every vertex, edge and induced cycle of length up to k into a cell. Note that $k \geq 3$ is a hyperparamter that needs to be set separately. Combining k-IC with cell encoding gives us a transformation we call *cellular ring encoding* (CRE). CRE transforms a graph into another graph with vertex features. An example of k-IC and CRE can be seen in Figure 6.1.

Proposition 11. *CRE together with WL is more expressive than WL.*

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

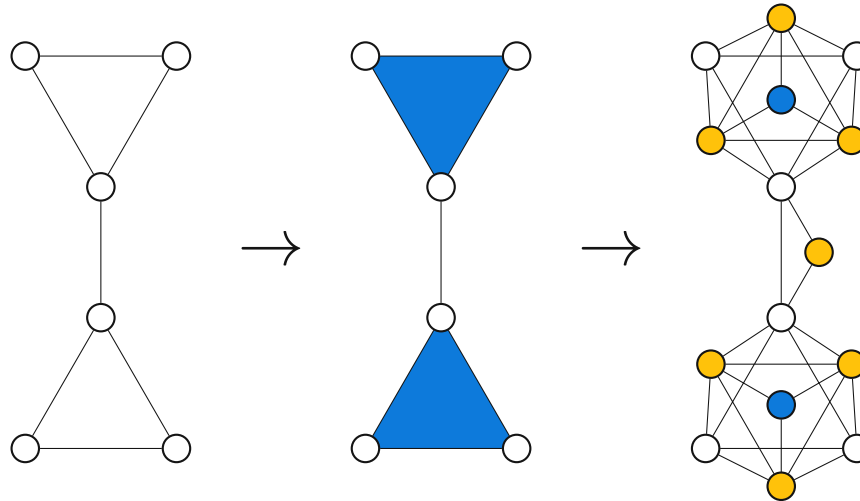

Figure 6.1: Left: a graph. Center: a regular cell complex built from the graph by lifting all induced cycles to 2-dimensional cells (blue) via k-IC. Right: a graph obtained from the cell complex via cell encoding or directly from the original graph via cellular ring encoding. Vertices that existed in the original graph are colored white. Yellow vertices represent edges in the graph and blue vertices represent induced cycles in the graph. (This image is partly based on an image by Bodnar et al. [2021a])

Proof. k-IC has been shown to be strictly more expressive than WL when combined with CWL [Bodnar et al., 2021a]. By Theorem 10 it follows that combining CRE with WL is strictly more expressive than just WL. □

Furthermore, since the graph neural network GIN [Xu et al., 2019] can be made as expressive as WL it follows that combining CRE with GIN is more expressive than WL for suitable parameter choices of GIN.

We will now describe details of our cellular ringe encoding implementation. CRE one-hot encodes the dimension of cells. This means for every vertex that corresponds to a 0-dimensional, 1-dimensional or 2-dimensional cell the vector $(1, 0, 0), (0, 1, 0), (0, 0, 1)$ will be added to the vertex features, respectively. We call this *explicit pattern encoding*. Additionally, we implement the option to collect vertex features into vertices created by CRE. If a vertex is built from a cell that corresponds to an edge or an induced cycle, then this newly created vertex will be assigned the average features of those vertices. For example, if CRE adds a new vertex that corresponds to an edge $\{p, q\}$ in the graph then the newly created vertex will have the average features of p and q. We call this *aggregating vertex features* in the following section. It is only used to aggregate features that were present in the original graph and *not* applied to features created via explicit pattern encoding. Finally, we developed two different ways for CRE to interact with edge features. *Edge features in vertices* appends the average edge feature of a vertex to its features. This allows GNNs that normally cannot use edge features to use them. *Aggregate edge features* gives newly created edges the average edge features of the original edge. For example, if $\{p, q\}$ is an edge then CRE will create a vertex that corresponds to this edge. This new vertex will be adjacent to both p and q. Then aggregate edge features will ensure that these newly created edges have the same features as $\{p, q\}$.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

Table 6.1: Hyperparameters of GIN based methods

Hyperparameters	Gin+CRE PROTEINS, NCI1, NCI109	Gin+CRE Ablation NCI1	Gin Ablation NCI1	Gin+CR ogb-molhiv
Epochs	350	350	350	100
Batch size	16, 32, 64, 128	16, 32, 64, 128	16, 32, 64, 128	32, 64, 128
Learning rate	1e-3, 1e-4, 5e-4, 1e-5	1e-3, 1e-4, 5e-4, 1e-5	1e-3, 1e-4, 5e-4, 1e-5	1e-2, 1e-3, 1-e4, 5e-4, 1e-5
Drop out rate	0, 0.1, 0.2, 0.3, 0.4, 0.5	0, 0.1, 0.2, 0.3, 0.4, 0.5	0, 0.1, 0.2, 0.3, 0.4, 0.5	0, 0.5
Number of layers	2,3,4,5	2,3,4,5	2,3,4,5	2, 3, 4, 5
LR decay steps	5, 10, 20, 30, 40, 50	5, 10, 20, 30, 40, 50	5, 10, 20, 30, 40, 50	50
LR decay rate	0.25, 0.5, 0.9, 0.99	0.25, 0.5, 0.9, 0.99	0.25, 0.5, 0.9, 0.99	0.5
Embedding dimension	64	32, 64	32, 64	32, 64, 128, 300, 512, 1024
Max Ring Size	6, 8, 10	6, 8, 10	N/A	6, 8, 18
Aggr. edge features	N/A	N/A	N/A	True, False
Aggr. vertex features	True, False	True, False	N/A	True, False
Explicit pattern encoding (EPE)	True, False	True, False	N/A	True, False
Edge features in vertices	N/A	N/A	N/A	True, False
Type pooling	Coupled to EPE	Coupled to EPE	N/A	True, False

Table 6.2: Hyperparameters of kernel methods in the ablation on NCI1

Hyperparameters	WL SP (1 iter)	WL SP (2 iter)	WL SP + CRE (1 iter)	WL SP + CRE (2 iter)	WL ST + CRE	WL ST + CRE
WL Iterations	1	1, 2	1	1, 2	1, 2, 3, 4, 5, 10	1, 2, 3, 4, 5, 10
Max Ring Size	N/A	N/A	6, 8, 10	6, 8, 10	N/A	6, 8, 10
Aggr. vertex features	N/A	N/A	True, False	True, False	N/A	True, False
Explicit pattern encoding	N/A	N/A	True, False	True, False	N/A	True, False

6.3 Experiments

In this section, we investigate the performance of Cellular Ring Encoding on graph classification datasets. The experiments[1] are designed to determine whether CRE is on par with the model CIN introduced by Bodnar et al. [2021a] and current GNNs. Furthermore, we also investigate whether CRE generally improves the empirical performance of graph classification methods.

Similar to CIN [Bodnar et al., 2021a], we use GIN [Xu et al., 2019] with Jumping Knowledge [Xu et al., 2018] as our model. CIN performs message passing on cell complexes constructed with the cellular lifting map k-IC. Analogously, we use Cellular Ring Encoding. The size k of the largest induced cycle to lift will be tuned to the given datasets. We use type pooling (see Section 7.4) as our final pooling operation.

[1] Code can be found at `https://github.com/ocatias/CellComplexesToGraphs`

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

Table 6.3: Hyperparameters of CIN in the ablation on NCI1

Hyperparameters	CIN
Epochs	150
Batch size	32, 128
Drop position	lin2, final readout, lin1
Drop rate	0.0, 0.5
Embedding dim.	16, 32, 64
Init method	sum, mean
Learning rate	5e-4, 1e-3, 3e-3, 1e-2
LR decay rate	0.5, 0.9
LR decay steps	50, 20
Use coboundaries	True, False
Number of layers	3 ,4

For experiments on the TUDatasets and the ablation study we use the implementation of GIN with Jumping Knowledge from Bodnar et al. [2021a] that is mostly equivalent to the benchmark implentation from PyTorch Geometric. For experiments on `ogb-molhiv` we take the experimental setup from Hu et al. [2020] including their implementation of GIN with Jumping Knowledge and a virtual node. We extend this setup by adding Cellular Ring Encoding and extending the models with type pooling.

We present all used hyperparameter configurations in Tables 6.1, 6.2 and 6.3. The top part of the table contains model specific hyperparamters and the bottom part contains CRE specific hyperparameters. As the used TUDatasets do not provide edge features the parameters “aggregate edge features” and “edge features in vertices” are irrelevant to those experiments. Additionally, for GIN + CRE on TUDatasets we couple the use of type pooling to the “explicit pattern encoding” parameter meaning type pooling will be used if “explicit pattern encoding” is set to true. For GIN + CRE on `ogb-molhiv` dimensional pooling is a separately tuneable parameter.

TUDataset. We perform two sets of experiments on datasets from the TUDataset collection [Morris et al., 2020a][2], the first follows Xu et al. [2019] and the second follows Errica et al. [2020]. In the first set of experiments we evaluate GIN + CRE on `NCI1`, `NCI109`, and `PROTEINS`. These datasets consist of 1000 - 4000 graphs belonging to one of two classes. These datasets were chosen as they are the largest commonly used molecular datasets in the collection. We perform stratified 10-fold cross-validation and report the average and standard deviation of the epoch with the highest validation accuracy. We use Bayesian optimization to quickly find suitable parameters within less than 20 parameter combinations. Following Xu et al. [2019] we report the result of the parameter configuration with the best validation accuracy. The results can be found in Table 6.4. As expected, CRE improves the accuracy of GIN. Interestingly, GIN + CRE achieves a higher accuracy than CIN on all three datasets.

This evaluation method can overestimate the performance of models. To get a more realistic understanding of the performance of CRE we perform an ablation study by adapting an experiment setup introduced by Errica et al. [2020]. We perform stratified 10-fold cross

[2] `TUDataset` is accessible via pytorch geometric `pytorch-geometric.readthedocs.io/`

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Table 6.4: Accuracy on TUDataset graph classification tasks (bigger is better). Citations denote the source of the result. N/A means that the authors did not evaluate their algorithm on the given dataset.

Method	PROTEINS	NCI1	NCI109
WL Kernel [Shervashidze et al., 2011]	N/A	84.6 ± 0.4	84.5 ± 0.2
GIN [Xu et al., 2019]	76.2 ± 2.8	82.7 ± 1.6	N/A
PPGNs [Maron et al., 2019]	77.2 ± 4.7	83.2 ± 1.1	82.2 ± 1.4
GSN [Bouritsas et al., 2020]	76.6 ± 5.0	83.5 ± 2.0	N/A
CIN [Bodnar et al., 2021a]	77.0 ± 4.3	83.6 ± 1.4	84.0 ± 1.6
GIN + CRE	77.5 ± 3.9	84.0 ± 1.3	84.3 ± 1.5

validation on the NCI1 dataset to obtain tuples of training and test sets. For the GNN methods, we split off 10% of the trainig set as a validation set, while ensuring an equal class distribution. For each fold we tune the parameters on the training and validation set. In total we test 20 randomly selected parameters per fold. Then we train a model with the selected parameters on the training set with early stopping on the validation set and evaluate it on the test set, we do this three times for each fold to smooth out non-deterministic behaviour in the training process.

For the kernel methods we do not split off a validation set. Instead we train on the entire training set and select the parameter configuration with the highest training accuracy, in total we try up to 20 random parameter configurations per fold depending on the size of the parameter grid. We evaluate on the test set and report the average accuracy and standard deviation over all folds. We investigate three neural networks: GIN, GIN with CRE and CIN. We also investigate the Weisfeiler-Leman Shortest Path (WL SP) and Subtree Kernels (WL ST) using an SVM as the learning algorithm, with and without CRE.

The results can be found in Table 6.5. Combining GIN with with CRE improves the accuracy over GIN. Interestingly, CRE does not improve WL ST but improves WL SP when restricting it to a single iteration of the WL algorithm. It is surprising that in this setting CIN performs similarly to GIN even though we would expect it to outperform GIN and achieve similar results as GIN + CRE.

ogbg-molhiv. We evaluate GIN + CRE on the ogb-molhiv dataset [Hu et al., 2020][3]. Results are evaluated with the ROC-AUC score, according to Hu et al. [2020]. ogb-molhiv provides a train, validation and test split, allowing fair comparisons between different methods. Similar to Hu et al. [2020] we extend our setup with a virtual node (VN), that is, a node that is connected to all nodes in the graph. We train GIN + VN + CRE to see how much we can improve upon GIN + VN. We tune the hypereparameters via Bayesian optimization on the validation set, and train a model with the best parameters 10 times without setting a random seed. We report the mean and standard deviation of the test ROC-AUC score in the epoch with the highest validation score in Table 6.6. The experiments show that GIN + VN + CRE substantially improves over just GIN + VN, but does not manage to beat CIN.

[3] ogbg-molhiv is accessible via the ogb python package ogb.stanford.edu

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Table 6.5: Ablation on NCI1 (bigger is better).

Method	Accuracy
WL SP (1 iter)	76.6 ± 2.8
WL SP (2 iter)	78.9 ± 2.3
WL SP (1 iter) + CRE	78.8 ± 2.5
WL SP (2 iter) + CRE	Out of RAM
WL ST	82.3 ± 1.5
WL ST + CRE	82.5 ± 1.5
GIN	81.5 ± 2.2
CIN	81.4 ± 2.2
GIN + CRE	82.4 ± 1.8

Table 6.6: ROC-AUC on ogb-molhiv (bigger is better). Citations denote the source of the result.

Method	ROC-AUC
GIN + VN [Hu et al., 2020]	77.07 ± 1.49
GSN + GIN + VN [Bouritsas et al., 2020]	77.99 ± 1.00
GSN + DGN [Bouritsas et al., 2020]	80.39 ± 0.90
CIN [Bodnar et al., 2021a]	80.94 ± 0.57
GIN + VN + CRE	78.98 ± 1.53

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

CHAPTER 7

Simplifying Augmented Message Passing

In this section we extend the idea of simplifying improved message passing through graph transformations. In Section 7.1 we start by defining properties that such graph transformations should have. Then, we apply this idea to two other graph neural networks: Equivariant Subgraph Aggregation Networks (Section 7.2) and (local) δ-k-dimensional GNNs (Section 7.3). We show that both can be simplified to graph transformations that follow our rules. We identify δ-k-GNNs as an example of our idea, as they have always been implemented via a graph transformation plus message passing. Finally, we evaluate our graph transformations against CW Networks and Equivariant Subgraph Aggregation Networks in Section 7.4. These experiments show that our graph transformations consistently improve the performance of GNNs and achieve results competitive to the original algorithms.

7.1 Guidelines for graph transformations

We provide rules that every graph transformation intended to simplify message passing should follow. Due to the generality of this approach, we are unable to formalize most of these rules. Thus, these rules should be considered more as guidelines than strict rules to follow. In what follows, let `A` be a message passing algorithm that a graph transformation `GT` is intended to simplify. Then `GT` should follow these rules:

Rule 1 `GT` should not execute significant parts of `A`.

Rule 2 Running `GT` should be efficient with respect to its input size and `A`.

Rule 3 Running WL on the graph generated by `GT` should have a similar runtime complexity as running `A` on the original instance.

Rule 4 The result of `GT` should depend only on the input graph and (possibly) some hyperparameters.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Rule 5 `GT` must be permutation invariant.

Rule 1 is intended to stop `GT` from simulating `A`. Otherwise, this yields a simple nonsensensical graph transformation that can simplify any improved message passing algorithm. Assume `GT` simulates `A` and returns a graph with a single node whose features encode the output of `A`. If we combine this graph transformation with WL it is guaranteed to be as expressive as `A`. However, this does not fit into the idea of using message passing and will most likely not work in practice since it compresses a graph the graph to a unique identifier. During training this unique identifier allows the neural network to achieve a low training error by memorizing the labels belonging to each identifier. However, this approach does not work on unseen data as the model achieves a low training error due to memorization and not because it learned a pattern behind the data. As simulating a very small part of `A` should not be a problem, Rule 1 only forbids the simulation of *significant* parts of `A`. Rule 2 and Rule 3 are intended to ensure that the combination of `GT` with WL does not need significantly longer runtime. Thus, it should ensure that `GT` can be used in practice. This is also why Rule 2 only requires *similar* runtime complexity: increasing the runtime complexity linearly might be fine in practice whereas increasing it by a large constant might be not. Rule 4 is designed to stop `GT` from leaking information between graphs. In particular, we can make any dataset of graphs distinguishable by adding a unique identifier to each graph. However, such an approach defeats the purpose of our idea and is unlikely to work on unseen data, for a similar as described in Rule 1. Rule 5 is intended to ensure that the combination of `GT` with WL or a GNN is permumtation invariant. Additionally, Rule 5 makes it impossible to use of random labels as we are mainly interested in deterministic algorithms.

Let us investigate whether cell encoding fulfills these rules. Cell encoding does not execute CWL, thus it fulfills Rule 1. It also fulfills Rule 2 as all operations that are performed by cell encoding also need to be performed if one creates a cell complex. Rule 3 also holds, as the edges created by the graph transformation correspond to a subset of all cell neighborhood relations. In each iteration of WL and CWL they pass a message between all neighbors. This means that every message sent by WL corresponds to at least one message passed by CWL. Additionally, WL might even pass less messages, as described in Section 6.1. Rule 4 also holds as cell encoding only considers the given regular cell complex. Any hyperparameters that might appear come from the way the regular cell complex is constructed, for example the number of induced cycles that should be lifted from a graph. Finally, cell encoding is permutation invariant with respect to any permutation of the cells. Thus Rule 5 holds.

7.2 Equivariant Subgraph Aggregation Networks

We apply our idea of simplifying improved message passing to Equivariant Subgraph Aggregation Networks [Bevilacqua et al., 2022]. We propose *subgraph bag encoding* (SBE), an algorithm that for a given policy π transforms a graph G to G_π. We prove that SBE combined with WL is at least as expressive as DSS-WL (or DSS-GNN) in distinguishing pairs of graphs. Figure 7.1 shows an example of SBE. For each vertex v in a subgraph $S \in \pi(G)$, SBE creates a unique vertex v_S and connects them according to $E(S)$. For any vertex $v \in V(G)$, it creates a vertex v_G and a separator vertex v'. All vertices of the form v_G are connected as defined by $E(G)$. Separator vertices connect vertices in different subgraphs that stem from the same vertex in the original graph.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Wien Bibliothek Your knowledge hub

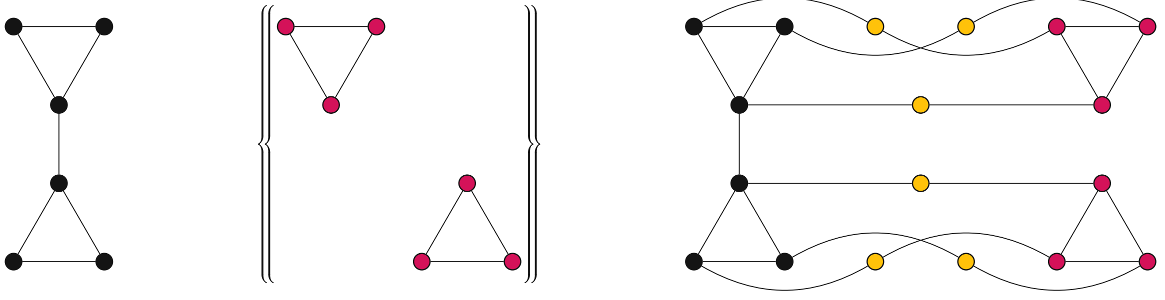

Figure 7.1: A graph (left) transformed into bags of subgraphs (center) by a policy and the result of applying subgraph bag encoding to the graph and bags of subgraphs (right). Black vertices represent the original graph, red vertices the subgraphs and yellow vertices the separator vertices.

Definition 7 (Subgraph Bag Encoding)**.** *Given a graph G and policy π we define $G_\pi = (V_\pi, E_\pi)$ to be the graph obtained by subgraph bag encoding. Where*

$$V_\pi = \{v' \mid v \in V(G)\} \cup \bigcup_{S \in \pi(G) \cup G} V(S),$$

$$E_\pi = \bigcup_{S \in \pi(G) \cup G} E(S) \cup \{\{v', v_S\} \mid S \in \pi(G) \cup G \text{ and } v \in V(S)\}.$$

For unlabeled graphs we introduce a vertex feature that encodes whether it is a separator vertex, was created from a subgraph, or was created from G. For labeled graphs we extend the features correspondingly.

Theorem 12. *SBE together with WL is at least as expressive as DSS-WL.*

Proof. Let G, H be two graphs and π a policy. We use $\tau_{v,S}$ to denote the coloring of vertex v_S obtained by applying WL to G_π, and $\tau_{v'}$ to denote the color of separator vertex v'. We use $c^t_{v,S}$ to denote the color of vertex v in subgraph S obtained in the t-th iteration of DSS-WL. We assume that WL cannot distinguish G_π, H_π. We prove that for any $t \geq 0$ it holds that

$$\forall v \in V(G), w \in V(H), S \in \pi(G), T \in \pi(H) : \tau_{v,S} = \tau_{w,T} \rightarrow c^t_{v,S} = c^t_{w,T}$$

by induction on the iteration t of DSS-WL. Note that proving this statement is equivalent to the theorem: As the graphs cannot be distinguished by WL we know that there exists a bijection $\alpha : V(G_\pi) \rightarrow V(H_\pi)$ that maps vertices of the same colors together. Since the additional features let us distinguish separator vertices v' from v_S, this means that α always maps a vertex v_S to a vertex w_T such that they are assigned the same color. From the statement then follows that the graphs are assigned the same colors by DSS-WL.

Base case: Any vertices $v_S \in V(G_\pi), w_T \in V(H_\pi)$ are assigned the same color by WL as their counterpart $v \in V(S), w \in V(T)$ by DSS-WL. Thus from $\tau_{v,S} = \tau_{w,T}$ it follows that $c^0_{v,S} = c^0_{w,T}$.

Induction hypothesis: We assume the statement holds for n.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Wien Bibliothek Your knowledge hub

Induction step: Let $v \in V(G), w \in V(H), S \in \pi(G), T \in \pi(H)$. We assume that $\tau_{v,S} = \tau_{w,T}$ holds and want to show $c^{n+1}_{v,S} = c^{n+1}_{w,T}$. For this, we show that the color obtained by the hash function is the same, as in both cases the same values get put into the function. Thus, we need to show that $c^n_{v,S} = c^n_{w,T}$, $N^n_{v,S} = N^n_{w,T}$, $C^n_v = C^n_w$, and $M^n_v = M^n_w$. From the induction hypothesis and $\tau_{v,S} = \tau_{w,T}$ it immediately follows that $c^n_{v,S} = c^n_{w,T}$.

We want to show that $N^n_{v,S} = N^n_{w,T}$. From $\tau_{v,S} = \tau_{w,T}$ and the fact that τ is a stable coloring we know there exists a bijection $\beta : \mathcal{N}_{G_\pi}(v_S) \to \mathcal{N}_{H_\pi}(w_T)$ where for any $x \in \mathcal{N}_{G_\pi}(v_S)$ it holds that $\tau_x = \tau_{\beta(x)}$. Additionally, all neighbors of v_S are either separator vertices or neighbors in the subgraph S and the features allow us to distinguish between those two types. Thus β maps vertices from $V(S)$ to $V(T)$. Note that these are exactly the vertices whose colors are in $N^n_{v,S}, N^n_{w,T}$. Thus, by combining this with the induction hypothesis we obtain that $N^n_{v,S} = N^n_{w,T}$.

We want to show that $C^n_v = C^n_w$. We use the function β defined in the previous paragraph. Due to the additional features, β must map a separator vertex v' to a separator vertex w' meaning that $\tau_{v'} = \tau_{w'}$. As the coloring is stable, this means that there exists a bijective function γ mapping neighbors of v' to neighbors of w' such that they are assigned the same color by τ. As these neighborhoods correspond to $c_{v,S'}, c_{w,T'}$ for all $S' \in \pi(G), T' \in \pi(H)$, we can combine this with the induction hypothesis to obtain that $C^n_v = \left\{\!\!\left\{ c^n_{v,S'} \mid S' \in \pi(G) \text{ and } v \in V(S') \right\}\!\!\right\} = \left\{\!\!\left\{ c^n_{w,T'} \mid T' \in \pi(H) \text{ and } w \in V(T') \right\}\!\!\right\} = C^n_w$. Note that v' also has a neighbor v_G, that is a vertex from the graph G, which was *not* created from a policy. However, as this vertex is assigned a different feature γ does not map such a vertex to a vertex created by the policy.

Finally, we want to show $M^n_v = M^n_w$. Recall that $M^n_v = \{\!\!\{ C^n_x \mid x \in \mathcal{N}_G(v) \}\!\!\}$. In the previous paragraph we have already argued that $\tau_{v'} = \tau_{w'}$. Thus, we know that $\tau_{v,G} = \tau_{w,H}$ as the coloring is stable and they are assigned a feature that is unique among the neighbors of v' and w'. Thus, there exists a bijective function $\sigma : \mathcal{N}_G(v) \to \mathcal{N}_H(w)$ such that for any neighbor $x \in \mathcal{N}_G(v)$ it holds that $\tau_{x,G} = \tau_{\sigma(x),H}$. We want to show that for any such neighbor $x \in N_G(v)$ it holds that $C^n_x = C^n_{\sigma(x)}$. Let x be one such vertex. As $\tau_{x,G} = \tau_{\sigma(x),H}$ it follows that the separator vertices assigned to x and $\sigma(x)$ have the same color: $\tau_x = \tau_{\sigma(x)}$. From here we can use the same argument as in the previous paragraph to obtain that $C^n_x = C^n_{\sigma(x)}$. This concludes the proof of the induction hypothesis and shows that the theorem holds. □

Note that Theorem 12 does not put any requirements on the policy. This allows maximum flexibility when it comes to developing new policies while still being able to profit from subgraph bag encoding.

We will now argue that SBE fulfills all rules from Section 7.1. SBE does not execute any parts of DSS-WL (Rule 1). It runs in linear time with respect to the number of vertices and edges in the input graph and subgraphs (Rule 2). Running WL on the instances generated by SBE does not need significantly more operations than DSS-WL (Rule 3): the only difference is that because of separator vertices passing information between vertices in the subgraph and the original graph requires one additional operation. Additionally, we believe that separator vertices are not necessary for Theorem 12 to hold. However, more research is needed in this direction. The output of SBE only depends on the input and any hyperparameters of the policy (Rule 4). Finally, as required by Rule 5, SBE is permutation invariant if the policy is permutation invariant. Hence, SBE fulfills all rules.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

7.3 δ-k-dimensional Weisfeiler-Leman

We apply our simplification idea to (local) δ-k-dimensional WL and GNNs [Morris et al., 2020b]. For this we present two graph transformations: *δ-k graph transformation* and *local δ-k graph transformation*. Very similar constructions were used by Morris et al. [2020b] to implement neural network variants of δ-k-WL. This shows that our idea has actually been used in practice before. However, it has never been used to simplify established GNNs and never explicitly defined.

The *δ-k graph transformation*, is an algorithm that transforms a graph to another graph such that applying WL or a suitably expressive GNN to the result graph is equally expressive as δ-k-WL.

Definition 8 (δ-k Graph Transformation)**.** *Let $G = (V, E)$ be a graph and $k \geq 1$ an integer. We denote by T_L (T_G) the sets containing every triplet (x, y, j) for which the k-tuples x and y are local (respectively global) j-neighbors. Then applying a δ-k graph transformation to G returns the graph $G_{\delta,k} = (V_{\delta,k}, E_{\delta,k})$ where*

$$V_{\delta,k} = \left\{x \mid x \in V(G)^k\right\} \cup \{l_{xyj} \mid (x, y, j) \in T_L\} \cup \{g_{xyj} \mid (x, y, j) \in T_G\},$$

$$E_{\delta,k} = \{\{x, l_{xyj}\}, \{y, l_{xyj}\} \mid (x, y, j) \in T_L\} \cup \{\{x, g_{xyj}\}, \{y, g_{xyj}\} \mid (x, y, j) \in T_G\}.$$

For any vertex x corresponding to a tuple we add a vertex feature that encodes its isomorphism type and allows us to distinguish it from l and g vertices. To all l_{xyj} (or g_{xyj}) vertices we add a feature that allows us to distinguish it from all g (respectively l) vertices and all vertices corresponding to tuples. Additionally, we add a feature that encodes j.

The l and g vertices and their features allow us to distinguish whether two tuples are global or local j-neighbors, and keep track of the value of j. These nodes are not strictly necessary if one uses edge features. For this one removes all edges, and all g and l vertices. Then one simply adds an edge between two vertices if their corresponding tuples are j-neighbors for any value of j. The edge features then encode j and whether two vertices are global or local neighbors. This works because for any two different tuples there exists at most one value of j for which they are j-neighbors. Thus at most one edge needs to be created between any two vertices.

For the purpose of GNNs, the features must be padded to have the same length for all vertices. Additionally, it might be useful to use one-hot encoding for j and the type of vertex.

We will now argue that the δ-k graph transformation fulfills all rules from Section 7.1. The graph transformation does execute not δ-k-WL (Rule 1). Running the graph transformation has linear complexity with respect to the number of tuples and their neighbors (Rule 2). Let us now consider the number of operations that running an iteration of WL on the generated graph requires (Rule 3). Due to the l and g vertices, each iteration will require a number of additional operations equivalent to the number of adjacencies between all tuples. In practice, this would be implemented with edge features and thus would not impact the runtime. Thus, we argue that Rule 3 holds. The output of δ-k graph transformation only depends on the input graph and the hyperparameter k (Rule 4). Finally, this transformation is permutation invariant (Rule 5).

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Theorem 13. *δ-k graph transformation together with WL is at least as expressive as δ-k-WL.*

Proof. Let G and H be two graphs, we need to prove that if WL cannot distinguish $G_{\delta,k}$ and $H_{\delta,k}$ then δ-k-WL cannot distinguish G and H. We use π_v to denote the color of vertex v encoding a k-tuple obtained by applying WL to $G_{\delta,k}$ or $H_{\delta,k}$. We use $k^t_{\mathbf{v}}$ to denote the color of the corresponding k-tuple $\mathbf{v}$ computed in the t-th iteration of δ-k-WL on G or H. We assume that WL cannot distinguish $G_{\delta,k}, H_{\delta,k}$ and show the stronger statement

$$\forall x \in V(G)^k, y \in V(H)^k : \pi_x = \pi_y \rightarrow c^t_{\mathbf{x}} = c^t_{\mathbf{y}}$$

for all $t \geq 0$ by induction on the iteration t of δ-k-WL.

Base case: Let x and y be arbitrary k-tuples. The statement follows from the fact that both the initial features of x, y and $c^0_{\mathbf{x}}, c^0_{\mathbf{y}}$ encode the isomorphism class of x and y.

Induction hypothesis: We assume the statement holds for n.

Induction step: Let x and y be arbitrary k-tuples. We need to show that $\pi_x = \pi_y \rightarrow c^{n+1}_{\mathbf{x}} = c^{n+1}_{\mathbf{y}}$. We assume that $\pi_x = \pi_y$. From the definition of δ-k-WL it follows that $c^{n+1}_{\mathbf{x}} = \left(c^n_{\mathbf{x}}, M^n_{\delta,\bar{\delta}}(\mathbf{x})\right)$ and $c^{n+1}_{\mathbf{y}} = \left(c^n_{\mathbf{y}}, M^n_{\delta,\bar{\delta}}(\mathbf{y})\right)$. From the assumption that $\pi_x = \pi_y$ and the induction hypothesis it follows that $c^n_{\mathbf{x}} = c^n_{\mathbf{y}}$. Next, we want to show that $M^n_{\delta,\bar{\delta}}(\mathbf{x}) = M^n_{\delta,\bar{\delta}}(\mathbf{y})$. By definition we know that

$$\begin{aligned} M^n_{\delta,\bar{\delta}}(\mathbf{x}) = \Big(& \left\{\!\!\left\{ \left(c^n_{\phi_1(\mathbf{x},w)}, \operatorname{adj}(x, \phi_1(\mathbf{x}, w))\right) \mid w \in V(G) \right\}\!\!\right\}, \dots, \\ & \left\{\!\!\left\{ \left(c^n_{\phi_k(\mathbf{x},w)}, \operatorname{adj}(\mathbf{x}, \phi_k(\mathbf{x}, w))\right) \mid w \in V(G) \right\}\!\!\right\} \Big). \end{aligned}$$

We prove that $M^n_{\delta,\bar{\delta}}(\mathbf{x}) = M^n_{\delta,\bar{\delta}}(\mathbf{y})$ by showing that for any $j \in [1, \dots, k]$ it holds that

$$\begin{aligned} & \left\{\!\!\left\{ \left(c^n_{\phi_j(\mathbf{x},w)}, \operatorname{adj}(\mathbf{x}, \phi_j(\mathbf{x}, w))\right) \mid w \in V(G) \right\}\!\!\right\} \\ = & \left\{\!\!\left\{ \left(c^n_{\phi_j(\mathbf{y},w)}, \operatorname{adj}(\mathbf{y}, \phi_j(\mathbf{y}, w))\right) \mid w \in V(H) \right\}\!\!\right\}. \end{aligned} \tag{7.1}$$

As $\pi_x = \pi_y$ we know that there exists a a bijective function $\alpha : \mathcal{N}_{G_{\delta,k}}(x) \rightarrow \mathcal{N}_{H_{\delta,k}}(y)$ such that for any neighbor p of x it holds that $\pi_p = \pi_{\alpha(p)}$. Recall that all neighbors of x and y are l and g vertices, and that the features of l and g vertices encode their type and the j of the j-neighborhood that connects the tuples. As WL is a color refinement algorithm it follows that α only maps l to l vertices and g to g vertices. Additionally, α only maps one vertex to another if they have the same j. Due to the coloring being stable and x and y having the same color it follows that

$$\begin{aligned} & \left\{\!\!\left\{ \left(\pi_{\phi_j(x,w)}, \operatorname{adj}(x, \phi_j(x, w))\right) \mid w \in V(G) \right\}\!\!\right\} \\ = & \left\{\!\!\left\{ \left(\pi_{\alpha(\phi_j(x,w))}, \operatorname{adj}(x, \alpha(\phi_j(x, w)))\right) \mid w \in V(G) \right\}\!\!\right\} \\ = & \left\{\!\!\left\{ \left(\pi_{\phi_j(y,w)}, \operatorname{adj}(y, \phi_j(y, w))\right) \mid w \in V(H) \right\}\!\!\right\}. \end{aligned}$$

By applying the induction hypothesis we obtain that Equation 7.1 holds. This shows that the induction step is true and concludes the proof of Theorem 3. □

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

As δ-k-WL is equally expressive as δ-k-LWL$^+$ and at least as expressive as k-WL and δ-k-LWL, this result also generalizes to those algorithms. However, the big advantages of δ-k-LWL and δ-k-LWL$^+$ over δ-k-WL is that they only make very restricted use of global j-neighborhoods. This means that δ-k graph transformation is not an efficient alternative as it uses all global neighborhoods. However, the graph transformation can be adapted into *local δ-k graph transformation* which only makes uses of local neighborhoods.

Definition 9 (Local δ-k Graph Transformation)**.** *Let $G = (V, E)$ be a graph and $k \geq 1$ an integer. Then applying a local δ-k graph transformation to G returns the graph $G_{\delta,k} = (V_{\delta,k}, E_{\delta,k})$ where*

$$\begin{aligned} V_{\delta,k} =& \Big\{ x \mid x \in V(G)^k \Big\} \cup \\ & \Big\{ l_{xyj} \mid x, y \in V(G)^k, j \in [1, \ldots, k], x \text{ and } y \text{ are local } j\text{-neighbors} \Big\}, \\ E_{\delta,k} =& \Big\{ \{x, l_{xyj}\}, \{y, l_{xyj}\} \mid x, y \in V(G)^k, j \in [1, \ldots, k], x \text{ and } y \text{ are local } j\text{-neighbors} \Big\}. \end{aligned}$$

For any vertex x corresponding to a tuple we add a vertex feature that encodes its isomorphism type and a feature that allows us to distinguish it from l vertices. To all l_{xyj} vertices we add a feature that allows us to distinguish it from all vertices corresponding to tuples. Additionally, we add a feature that encodes j.

Similar to δ-k graph transformation, we can avoid using l vertices if we use edge features. The rules from Section 7.1 hold and can be argued similarly to the δ-k graph transformation.

Theorem 14. *Local δ-k graph transformation together with WL is at least as expressive as δ-k-LWL.*

Proof. Let G and H be two graphs, we need to prove that if WL cannot distinguish $G_{\delta,k}$ and $H_{\delta,k}$ then δ-k-LWL cannot distinguish G and H. We use π_v to denote the color of vertex v encoding a k-tuple obtained by applying WL to $G_{\delta,k}$ or $H_{\delta,k}$. We use $k^t_{\mathbf{v}}$ to denote the color of the corresponding k-tuple $\mathbf{v}$ computed in the t-th iteration of δ-k-LWL on G or H. We assume that WL cannot distinguish $G_{\delta,k}, H_{\delta,k}$ and show the stronger statement

$$\forall x \in V(G)^k, y \in V(H)^k : \pi_x = \pi_y \rightarrow c^t_{\mathbf{x}} = c^t_{\mathbf{y}}$$

for all $t \geq 0$ by induction on the iteration t of δ-k-LWL.

Base case: Let x and y be arbitrary k-tuples. The statement follows from the fact that both the initial features of x, y and $c^0_{\mathbf{x}}, c^0_{\mathbf{y}}$ encode the isomorphism class of x and y.

Induction hypothesis: We assume the statement holds for $t = n$.

Induction step: Let x and y be arbitrary k-tuples. We need to show that $\pi_x = \pi_y \rightarrow c^{n+1}_{\mathbf{x}} = c^{n+1}_{\mathbf{y}}$. We assume that $\pi_x = \pi_y$. From the definition of δ-k-LWL it follows that $c^{n+1}_{\mathbf{x}} = (c^n_{\mathbf{x}}, M^n_\delta(\mathbf{x}))$ and $c^{n+1}_{\mathbf{y}} = \left(c^n_{\mathbf{y}}, M^n_\delta(\mathbf{y})\right)$. From the assumption that $\pi_x = \pi_y$ and the induction hypothesis it follows that $c^n_{\mathbf{x}} = c^n_{\mathbf{y}}$. Next, we want to show that $M^n_\delta(\mathbf{x}) = M^n_\delta(\mathbf{y})$. By definition we know that

$$\begin{aligned} M^n_\delta(\mathbf{x}) = \Bigg(& \Big\{\!\!\Big\{ \left(c^n_{\phi_1(\mathbf{x},w)} \right) \mid w \in \mathcal{N}_G(x_1) \Big\}\!\!\Big\}, \ldots, \\ & \Big\{\!\!\Big\{ \left(c^n_{\phi_k(\mathbf{x},w)} \right) \mid w \in \mathcal{N}_G(x_k) \Big\}\!\!\Big\} \Bigg). \end{aligned}$$

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

We prove that $M^n_\delta(\mathbf{x}) = M^n_\delta(\mathbf{y})$ by showing that for any $j \in [1, \ldots, k]$ it holds that

$$\begin{aligned} &\{\{ \left(c^n_{\phi_j(\mathbf{x},w)} \right) \mid w \in \mathcal{N}_G(x_j) \}\} \\ =&\{\{ \left(c^n_{\phi_1(\mathbf{y},w)} \right) \mid w \in \mathcal{N}_H(y_j) \}\} . \end{aligned} \tag{7.2}$$

As $\pi_x = \pi_y$ we know that there exists a a bijective function $\alpha : \mathcal{N}_{G_{\delta,k}}(x) \to \mathcal{N}_{G_{\delta,k}}$ such that for any neighbor p of x it holds that $\pi_p = \pi_{\alpha(p)}$. Recall that all neighbors of x and y are l vertices. As the feature of an l vertex encodes j and WL is a color refinement algorithm, we know that α only maps two vertices to another if they have the same j. Due to the coloring being stable and x and y having the same color it follows that

$$\begin{aligned} &\{\{ \left(\pi_{\phi_j(x,w)} \right) \mid w \in \mathcal{N}_G(x_j) \}\} \\ =&\{\{ \left(\pi_{\alpha(\phi_j(x,w))} \right) \mid w \in \mathcal{N}_G(x_j) \}\} \\ =&\{\{ \left(\pi_{\phi_j(y,w)} \right) \mid w \in \mathcal{N}_H(y_j) \}\} . \end{aligned}$$

By applying the induction hypothesis we obtain that Equation 7.2 holds. This shows that the induction step is true and concludes the proof of Theorem 4. □

As δ-k-LWL$^+$ makes use of counting global neighbors with the same color it is not obvious whether it is possible to create a similar algorithm by just combining a graph transformation and WL. This shows one of the limits of our idea: while we are able to create a transformation with the same expressiveness, we are unable to build one that is equally sparse.

7.4 Experiments

We investigate the performance of cell encoding and subgraph bag encoding compared to the original algorithms. We intend to show that our approach of simplifying improved message passing via a graph transformation yields competitive results on real life molecular datasets. Thus, we do not include the (local) δ-k graph transformation in our experiments, as it has always been implemented as a graph transformation together with a GNN.

Models. We compare the empirical performance of cell encoding and subgraph bag encoding against the CW Network CIN [Bodnar et al., 2021a] and the Equivariant Subgraph Aggregation Network DSS-GNN [Bevilacqua et al., 2022]. Similar to Bodnar et al. [2021a], Bevilacqua et al. [2022] we use the Graph Isomorphism Network (GIN) [Xu et al., 2019] as our GNN. As GIN with suitable parameters is equally expressive as WL, this means that GIN with CRE or SBE can be more expressive than WL. We combine GIN with methods obtained via cell encoding and subgraph bag encoding. CIN performs message passing on cell complexes constructed with the cellular lifting map k-IC. Analogously, we use the cell encoding method cellular ring encoding (CRE). For subgraph bag encoding we use a policy that transforms a graph into a set of 3-ego-networks, that is a set containing the induced 3-hop neighborhoods for each vertex. We use the same policy for DSS-GNN. We also include GIN and a multilayer perceptron (MLP) as baselines. The baseline MLP performs a mean pooling operation over the whole graph and then applies a MLP to the resulting feature vector, completely ignoring the graph structure and edge features.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Wien Bibliothek Your knowledge hub

Type pooling. In our novel graph transformations vertices get assigned additional features that encode the type of a vertex. In cell encoding, the features of a vertex encode the dimension of corresponding cell. In subgraph bag encoding, the features encode whether a vertex is a separator vertex, in the original graph, or in a subgraph. Inspired by a pooling operation by Bodnar et al. [2021a] we propose *type pooling*. The idea is that vertices with different types contain different information and have varying importance. Thus instead of pooling all vertices in a graph, we instead pool all vertices of a type. To get a single output for the entire graph, we then apply a multilayer perceptron with a ReLU activation function to each pooled vector and then sum the results. Formally, let T be the set of types and let H_t with $t \in T$ be the set of all representations of vertices of type t, and MLP_t be a multilayer perceptron with the ReLU activation function. Then type pooling computes

$$\text{Type-Pool}(T, H) = \sum_{t \in T} \mathrm{MLP}_t \left(\sum_{h \in H_t} h \right).$$

Datasets. We evaluate on the graph regression dataset `ZINC` [Gómez-Bombarelli et al., 2018, Sterling and Irwin, 2015][1], and on two graph classification datasets `ogbg-molhiv` [Hu et al., 2020] and `ogbg-moltox21` [Hu et al., 2020][2]. For `ZINC` we use the commonly used smaller variant that contains 12000 molecular graphs, it has been used for example by Bevilacqua et al. [2022], Barceló et al. [2021], Bouritsas et al. [2020]. `ogbg-molhiv` contains 41127 graphs and `ogbg-moltox21` contains 7831 graphs. Both `ogbg-molhiv` and `ogbg-moltox21` are binary graph classification problems. However, the prior has only one task while the latter has 12 tasks, meaning that 12 predictions need to be performed simultaneously. As defined in the open graph benchmark (ogb) [Hu et al., 2020] we use ROC-AUC to evaluate the results on `ogbg-molhiv` and `ogbg-moltox21`. As is common, we use mean average error for `ZINC`.

Setup. Our setup is based on Dwivedi et al. [2020]. We evaluate each method on each of the three datasets. All datasets supply a train, validation and test split which we use and keep fixed for all algorithms. We tune the hyperparameters on the validation set.

For all methods we tune the number of layers, dropout probability, and embedding dimension (or hidden layer size for MLP). For GIN+CRE and CIN, we tune the size of the largest cycle to lift. For GIN+CRE and GIN+SBE we also tune whether to use type pooling. Additionally, for GIN+CRE we tune whether to aggregate edge and vertex features to higher dimensional cells. The hyperparameters can be found in Table 7.1.

We try 20 parameter combinations per method and dataset. During training we use a learning rate of 10^{-3} and the ReduceLROnPlateau scheduler: whenever the model has not improved on the validation set for 20 epochs we lower the learning rate by 50%. The training stops when the learning rate dips below 10^{-5} or after at most 300 epochs. The only exception is that we train CIN on the `ogbg-molhiv` dataset for only 100 epochs due to how long the training takes. For evaluation, we use the metric obtained in the epoch with the best validation performance and report the average and standard deviation over 10 separate training runs with different random seeds. We use the L1 loss for `ZINC` and the binary cross entropy loss for `ogbg-molhiv` and `ogbg-moltox21`.

[1] `ZINC` is accessible via pytorch geometric `pytorch-geometric.readthedocs.io/`

[2] Both ogb datasets are accessible via the ogb python package `ogb.stanford.edu`

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Table 7.1: Hyperparameter grids for all models

Parameter	Models	Possible Values
Number of layers	All	[2,3,4,5]
Embedding Dimension / Hidden Layer Size (MLP)	All	[32, 64, 128, 256, 300]
Dropout probability	All	[0, 0.5]
Ring size	GIN+CRE, CIN	[6, 8]
Type pooling	GIN+CRE, GIN+SBE	[Yes, No]
Aggregate vertex features	GIN+CRE	[Yes, No]
Aggregate edge features	GIN+CRE	[Yes, No]

Table 7.2: ROC-AUC on `ogbg-molhiv` (bigger is better)

Algorithm	Test ROC-AUC	Validation ROC-AUC
MLP	0.62 ± 0.013	0.651 ± 0.005
GIN	0.766 ± 0.014	0.823 ± 0.007
CIN	0.772 ± 0.014	0.829 ± 0.014
GIN + CRE	0.779 ± 0.017	0.833 ± 0.007
DSS-GNN	0.774 ± 0.017	0.847 ± 0.011
GIN + SBE	0.737 ± 0.013	0.831 ± 0.010

Table 7.3: ROC-AUC on `ogbg-moltox21` (bigger is better)

Algorithm	Test ROC-AUC	Validation ROC-AUC
MLP	0.613 ± 0.003	0.586 ± 0.001
GIN	0.749 ± 0.005	0.788 ± 0.004
CIN	0.754 ± 0.006	0.803 ± 0.004
GIN + CRE	0.750 ± 0.007	0.804 ± 0.006
DSS-GNN	0.771 ± 0.009	0.824 ± 0.005
GIN + SBE	0.762 ± 0.006	0.814 ± 0.004

Table 7.4: Mean average error (MAE) on `ZINC` (smaller is better)

Algorithm	Test MAE	Validation MAE
MLP	1.441 ± 0.001	1.317 ± 0.001
GIN	0.250 ± 0.007	0.264 ± 0.004
CIN	0.083 ± 0.005	0.087 ± 0.005
GIN + CRE	0.090 ± 0.005	0.097 ± 0.005
DSS-GNN	0.101 ± 0.004	0.118 ± 0.006
GIN + SBE	0.103 ± 0.006	0.133 ± 0.006

Results. The results can be found in Table 7.2, 7.3, and 7.4. GIN+CRE and CIN obtain similar results on all three datasets, and outperform GIN on two of the three datasets. On two of the three datasets GIN+SBE performs much better than the GIN and MLP baselines and competitively to DSS-GNN. Only on `ogbg-molhiv` it performs slightly worse than DSS-GNN and GIN, but still much better than the MLP baseline.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

CHAPTER 8

Conclusion

In this thesis, we have investigated graph neural networks (GNNs) that are based message passing graph neural networks (MPNNs) but have higher expressiveness. We have shown that many GNNs with improved message passing can be simplified to a combination of graph transformation and classical message passing. This simplification has many advantages. First, instead of requiring specialized implementations that are able to perform message passing on regular cell complex or sets of subgraphs, graph transformations just require changes to the preprocessing of data. Secondly, these transformations make MPNNs compatible to additional structures such as regular cell complexes. Thirdly, this approach unifies different message passing algorithms and allows for easier comparisons between them.

We have proposed a list of guidelines that such graph transformations should follow. For this, we studied three different variants of graph neural networks: CW Networks, Equivariant Subgraph Aggregation Networks and (local) δ-k dimensional GNNs. We proposed graph transformations for these algorithms and proved that combining the graph transformation with message passing is at least as expressive as the original algorithm. For CW Networks and Equivariant Subgraph Aggregation Networks we demonstrated that our graph transformations achieve competitive performance on molecular graph datasets. We used (local) δ-k dimensional GNNs as an example to argue that our approach of simplifying more expressive message passing algorithms via graph transformations has already been used in the past. Finally, we investigated a connection between MPNNs with homomorphism counts and CW Networks. We showed CW Networks with the lifting map k-CL is at least as expressive as $\{K_3, \ldots, K_k\}$-GNNs.

However, this approach also has its problems. Firstly, the problem of distinguishing graphs and expressiveness is not directly relevant to real-world problems. For example, Zopf [2022] shows that in many datasets pretty much all graphs can be distinguished from each other by WL. Additionally, they showed that WL based methods can theoretically achieve very high accuracies on most datasets. We argue that we want our GNNs to learn simple functions that generalize well to unseen data. For example, an MPNN that tries to count cycles can obtain a 100 % accuracy as long as all graphs in the dataset are WL distinguishable. However, this does not mean that it will generally learn to count cycles and instead will overfit on the unique representation of each graph. Thus, when this MPNN is applied to completely unseen graphs it will perform significantly worse. Secondly, another problem is that many GNNs

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

with higher expressiveness only work on small graphs due to the runtime complexity of the methods involved. For example, building a regular cell complex by lifting cycles or cliques requires us to *count* all of these structures. This makes it difficult to apply these methods to large graphs. As our graph transformations build graphs that are similar to the structures build by the original algorithm, they inherit their inefficiencies. Thirdly, the experiments in Section 7.4 show that our graph transformations do not always yield similar results to the original algorithms. Sometimes they even yield significantly worse results than an MPNN baseline. This shows the importance of always using baseline algorithms, ideally at least an MPNN and an MLP that ignores the graph structure. We argue that due to how simple it is to try multiple algorithms based on graph transformations this is not a huge issue.

We will now give an outlook on future work. Due to the generality of our idea, our guidelines for graph transformations (Section 7.1) are not formal. It would be interesting to see whether these rules can be formalized and whether they have loopholes or problems that require changes. An obvious extension of our idea is applying it to other GNNs such as Automorphism-based Graph Neural Nets Thiede et al. [2021]. In Section 7.3 we have seen that we are unable to build a graph transformation that yields the same expressiveness as δ-k-LWL$^+$ and has the same sparsity in the resulting graph. Thus, it would be interesting to see if there exists a message passing algorithm that *cannot* be simplified by a reasonable graph transformation.

Another interesting direction is investigating non deterministic algorithms used to compute structures in the graph. In the paper that introduced Equivariant Subgraph Aggregation Networks [Bevilacqua et al., 2022], the authors proposed stochastic sampling of subgraphs to reduce the number of subgraphs. They showed that this did not have much impact on the empirical performance. This introduces a source of randomness to the algorithm which means that the algorithm is only permutation invariant in expectation. This is generally an undesired property. However, is has been shown that attaching random labels to vertices is already enough to yield a GNN that can learn any function on a graph [Abboud et al., 2021]. Thus, it would be interesting to see if there exist cases in which the randomness from stochastic sampling is enough to prove universality results.

Finally, we believe that it can be shown that CW Networks with the lifting map k-C is at least as expressive as $\{C_3, \ldots, C_k\}$-GNNs.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

List of Figures

1.1 Left: a social hierarchy can be represented as a graph. An edge from a vertex X to a vertex Y means that X is a parent of Y. Right: a Leontif directed graph representing an economy. Edges represent funds flowing from the consumers to a company or from a company to a company. (Figure based on Foulds [2012]) . 3

2.1 Left: a complete graph with three vertices (K_3), called a triangle or a cycle of length 3 (C_3); Right: a complete graph with four vertices (K_4) 5
2.2 A graph modeling a biphenyl molecule 6
2.3 Two graphs that cannot be distinguished be the WL algorithm or any MPNN. Node colors represent colors assigned by the WL algorithm. 14
2.4 A star shaped graph with four vertices. 15

4.1 A graph (center) with all homomorphisms from a rooted 4-cycle with the black root to the red vertex in the graph. 22
4.2 A graph (left) turned into a regular cell complex (right). Vertices and edges in the graph are 0 and 1-dimensional cells, respectively. Induced cycles in the graph are 2-dimensional cells drawn in blue. (This image is based on a similar image by Bodnar et al. [2021a]) 22
4.3 A graph (left) transformed into bags of subgraphs (right) by a policy that computes 1-ego-networks. Each subgraph is the induced 1-hop neighborhood of the red colored vertex. 24
4.4 A graph (left) and a representation of δ-k-WL for $k = 2$ on this graph (right). Big nodes represent 2-tuples with the relevant subgraphs drawn inside. Edges between big nodes represent adjacencies between tuples, the type of j-neighborhood is written on the edges. Note that all tuples are local neighbors. If we remove the edge $\{A, B\}$ from the graph, then all tuples are global neighbors. 25

6.1 Left: a graph. Center: a regular cell complex built from the graph by lifting all induced cycles to 2-dimensional cells (blue) via k-IC. Right: a graph obtained from the cell complex via cell encoding or directly from the original graph via cellular ring encoding. Vertices that existed in the original graph are colored white. Yellow vertices represent edges in the graph and blue vertices represent induced cycles in the graph. (This image is partly based on an image by Bodnar et al. [2021a]) 36

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Wien Bibliothek Your knowledge hub

7.1 A graph (left) transformed into bags of subgraphs (center) by a policy and the result of applying subgraph bag encoding to the graph and bags of subgraphs (right). Black vertices represent the original graph, red vertices the subgraphs and yellow vertices the separator vertices. 43

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

List of Tables

6.1 Hyperparameters of GIN based methods 37
6.2 Hyperparameters of kernel methods in the ablation on NCI1 37
6.3 Hyperparameters of CIN in the ablation on NCI1 38
6.4 Accuracy on TUDataset graph classification tasks (bigger is better). Citations denote the source of the result. N/A means that the authors did not evaluate their algorithm on the given dataset. 39
6.5 Ablation on `NCI1` (bigger is better). 40
6.6 ROC-AUC on `ogb-molhiv` (bigger is better). Citations denote the source of the result. 40

7.1 Hyperparameter grids for all models 50
7.2 ROC-AUC on `ogbg-molhiv` (bigger is better) 50
7.3 ROC-AUC on `ogbg-moltox21` (bigger is better) 50
7.4 Mean average error (MAE) on `ZINC` (smaller is better) 50

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Bibliography

Ralph Abboud, İsmail İlkan Ceylan, Martin Grohe, and Thomas Lukasiewicz. The surprising power of graph neural networks with random node initialization. In *IJCAI*, 2021.

George W. Adamson and Judith A. Bush. A method for the automatic classification of chemical structures. *Information Storage and Retrieval*, 9(10):561–568, 1973.

James Atwood and Don Towsley. Diffusion-convolutional neural networks. In *NeurIPS*, 2016.

László Babai. Graph isomorphism in quasipolynomial time [extended abstract]. In *Proceedings of the Forty-Eighth Annual ACM Symposium on Theory of Computing; Full paper available as arXiv preprint arXiv:1512.03547*, STOC '16, page 684–697. Association for Computing Machinery, 2016.

Davide Bacciu, Federico Errica, and Alessio Micheli. Contextual graph markov model: A deep and generative approach to graph processing. In *ICML*, 2018.

Pablo Barceló, Floris Geerts, Juan L. Reutter, and Maksimilian Ryschkov. Graph Neural Networks with Local Graph Parameters. In *NeurIPS*, 2021.

Beatrice Bevilacqua, Fabrizio Frasca, Derek Lim, Balasubramaniam Srinivasan, Chen Cai, Gopinath Balamurugan, Michael Bronstein, and Haggai Maron. Equivariant Subgraph Aggregation Networks. In *ICLR*, 2022.

Cristian Bodnar, Fabrizio Frasca, Nina Otter, Yu Guang Wang, Pietro Liò, Guido Montúfar, and Michael Bronstein. Weisfeiler and Lehman go cellular: CW Networks. In *NeurIPS*. 2021a.

Cristian Bodnar, Fabrizio Frasca, Yu Guang Wang, Nina Otter, Guido Montúfar, Pietro Liò, and Michael Bronstein. Weisfeiler and Lehman go topological: Message passing simplicial networks. In *ICML*, 2021b.

Karsten M. Borgwardt and Hans-Peter Kriegel. Shortest-path kernels on graphs. *IEEE International Conference on Data Mining*, page 8 pp., 2005.

Karsten M. Borgwardt, Cheng Soon Ong, Stefan Schönauer, S. V. N. Vishwanathan, Alex J. Smola, and Hans-Peter Kriegel. Protein function prediction via graph kernels. *Bioinformatics*, 21 Suppl 1:i47–56, 01 2005.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Giorgos Bouritsas, Fabrizio Frasca, Stefanos Zafeiriou, and Michael M. Bronstein. Improving graph neural network expressivity via subgraph isomorphism counting. *arXiv preprint arXiv:2006.09252, earlier version appeared in Graph Representation Learning and Beyond (GRL+) Workshop at ICML*, 2020.

Shaked Brody, Uri Alon, and Eran Yahav. How Attentive are Graph Attention Networks? In *ICLR*, 2022.

Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. In *ICLR*, 2013.

Zhengdao Chen, Lei Chen, Soledad Villar, and Joan Bruna. Can graph neural networks count substructures? In *NeurIPS*, 2020.

George Dasoulas, Ludovic Dos Santos, Kevin Scaman, and Aladin Virmaux. Coloring graph neural networks for node disambiguation. In *IJCAI*, 2020.

Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *NeurIPS*, 2016.

Reinhard Diestel. Graph theory. *Springer-Verlag Heidelberg*, 2005.

Vijay Prakash Dwivedi, Chaitanya K Joshi, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Benchmarking graph neural networks. *arXiv preprint arXiv:2003.00982*, 2020.

Stefania Ebli, Michaël Defferrard, and Gard Spreemann. Simplicial neural networks. In *NeurIPS*, 2020.

Federico Errica, Marco Podda, Davide Bacciu, and Alessio Micheli. A fair comparison of graph neural networks for graph classification. In *ICLR*, 2020.

Leslie R Foulds. *Graph theory applications*. Springer Science & Business Media, 2012.

Holger Fröhlich, Jörg K. Wegner, Florian Sieker, and Andreas Zell. Optimal assignment kernels for attributed molecular graphs. In *ICML*, 2005.

Claudio Gallicchio and Alessio Micheli. Graph echo state networks. In *International Joint Conference on Neural Networks (IJCNN)*, 2010.

Thomas Gärtner, Peter Flach, and Stefan Wrobel. On graph kernels: Hardness results and efficient alternatives. In *Learning Theory and Kernel Machines*, pages 129–143. Springer Berlin Heidelberg, 2003.

Floris Geerts and Juan L Reutter. Expressiveness and approximation properties of graph neural networks. In *ICLR*, 2022.

Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In *ICML*, 2017.

Rafael Gómez-Bombarelli, Jennifer N. Wei, David Duvenaud, José Miguel Hernández-Lobato, Benjamín Sánchez-Lengeling, Dennis Sheberla, Jorge Aguilera-Iparraguirre, Timothy D. Hirzel, Ryan P. Adams, and Alán Aspuru-Guzik. Automatic chemical design using a data-driven continuous representation of molecules. *ACS Central Science*, 4(2):268–276, 2018.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep learning. MIT Press, 2016.

M. Gori, G. Monfardini, and F. Scarselli. A new model for learning in graph domains. In *International Joint Conference on Neural Networks (IJCNN)*, 2005.

Alexander Grigor'yan, Yuri Muranov, and Shing-Tung Yau. Graphs associated with simplicial complexes. *Homology, Homotopy and Applications*, 16, 2014.

Mustafa Hajij, Kyle Istvan, and Ghada Zamzmi. Cell complex neural networks. In *Topological Data Analysis and Beyond Workshop at NeurIPS*, 2020.

William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive Representation Learning on Large Graphs. In *NeurIPS*, 2017.

Zaid Harchaoui and Francis Bach. Image classification with segmentation graph kernels. In *CVPR*, 2007.

Linus Hermansson, Fredrik D. Johansson, and Osamu Watanabe. Generalized shortest path kernel on graphs. In *Discovery Science*, pages 78–85. Springer International Publishing, 2015.

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. volume 2, pages 359–366, 1989.

Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open Graph Benchmark: Datasets for machine learning on graphs. In *NeurIPS*, 2020.

Neil Immerman and Eric Lander. Describing graphs: A first-order approach to graph canonization. In *Complexity Theory Retrospective*, pages 59–81. Springer New York, 1990.

Stefanie Jegelka. Theory of graph neural networks: Representation and learning. In *International Congress of Mathematicians; arXiv preprint arXiv:2204.07697*, 2022.

Hisashi Kashima, Koji Tsuda, and Akihiro Inokuchi. Marginalized kernels between labeled graph. In *ICML*, 2003.

Alexandros Dimitrios Keros, Vidit Nanda, and Kartic Subr. Dist2cycle: A simplicial neural network for homology localization. In *AAAI*, 2022.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

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

Ron Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *IJCAI*, 1995.

Nils Kriege, Marion Neumann, Kristian Kersting, and Petra Mutzel. Explicit versus implicit graph feature maps: A computational phase transition for walk kernels. In *ICDM*, 2014.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Nils Kriege, Marion Neumann, Christopher Morris, Kristian Kersting, and Petra Mutzel. A unifying view of explicit and implicit feature maps of graph kernels. *Data Mining and Knowledge Discovery*, 33, 2019.

Nils M. Kriege, Pierre-Louis Giscard, and Richard Wilson. On valid optimal assignment kernels and applications to graph classification. In *NeurIPS*, 2016.

Nils M. Kriege, Fredrik D. Johansson, and Christopher Morris. A survey on graph kernels. *Applied Network Science*, 5:1–42, 2020.

David R. Lide. CRC handbook of chemistry and physics. *CRC Press*, 2003.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *ICLR*, 2019.

Haggai Maron, Heli Ben-Hamu, Hadar Serviansky, and Yaron Lipman. Provably powerful graph networks. In *NeurIPS*, 2019.

Alessio Micheli. Neural network for graphs: A contextual constructive approach. *IEEE Transactions on Neural Networks*, 20(3):498–511, 2009.

Christopher Morris, Kristian Kersting, and Petra Mutzel. Glocalized Weisfeiler-Lehman graph kernels: Global-local feature maps of graphs. In *ICDM*, 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. In *AAAI*, 2019.

Christopher Morris, Nils M. Kriege, Franka Bause, Kristian Kersting, Petra Mutzel, and Marion Neumann. TUDataset: A collection of benchmark datasets for learning with graphs. In *ICML Workshop on Graph Representation Learning and Beyond*, 2020a.

Christopher Morris, Gaurav Rattan, and Petra Mutzel. Weisfeiler and Leman go sparse: Towards scalable higher-order graph embeddings. In *NeurIPS*, 2020b.

R. Sato, Makoto Yamada, and Hisashi Kashima. Random Features Strengthen Graph Neural Networks. In *SDM*, 2021.

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

N. Sherashidze, S.V.N. Vishwanathan, Tobias Petri, Kurt Mehlhorn, and Karsten Borgwardt. Efficient graphlet kernels for large graph comparison. *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 5:488–495, 2009.

Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M. Borgwardt. Weisfeiler-Lehman graph kernels. *JMLR*, 12:2539–2561, 2011.

A. Sperduti and A. Starita. Supervised neural networks for the classification of structures. *IEEE Transactions on Neural Networks*, 8(3):714–735, 1997.

Teague Sterling and John J. Irwin. Zinc 15 – ligand discovery for everyone. *Journal of Chemical Information and Modeling*, 55(11):2324–2337, 2015.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Jonathan Stokes, Kevin Yang, Kyle Swanson, Wengong Jin, Andres Cubillos-Ruiz, Nina Donghia, Craig MacNair, Shawn French, Lindsey Carfrae, Zohar Bloom-Ackerman, Victoria Tran, Anush Chiappino-Pepe, Ahmed Badran, Ian Andrews, Emma Chory, George Church, Eric Brown, Tommi Jaakkola, Regina Barzilay, and James Collins. A Deep Learning Approach to Antibiotic Discovery. *Cell*, 180:688–702.e13, 02 2020.

Erik Thiede, Wenda Zhou, and Risi Kondor. Autobahn: Automorphism-based Graph Neural Nets. In *NeurIPS*, 2021.

Jan Toenshoff, Martin Ritzert, Hinrikus Wolf, and Martin Grohe. Graph learning with 1d convolutions on random walks. *arXiv preprint arXiv:2102.08786*, 2021.

Jake Topping, Francesco Di Giovanni, Benjamin Paul Chamberlain, Xiaowen Dong, and Michael M. Bronstein. Understanding over-squashing and bottlenecks on graphs via curvature. In *ICLR*, 2022.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks. In *ICLR*, 2018.

Petar Veličković. Message passing all the way up. In *ICLR Workshop on Geometrical and Topological Representation Learning*, 2022.

Boris Weisfeiler and Andrei Leman. The reduction of a graph to canonical form and the algebra which appears therein. *Nauchno-Technicheskaya Informatsia*, 9, 1968.

Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1):4–24, 2021.

Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with Jumping Knowledge networks. In *ICML*, 2018.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *ICLR*, 2019.

Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola. Dive into deep learning. *arXiv preprint arXiv:2106.11342*, 2021.

Markus Zopf. 1-wl expressiveness is (almost) all you need. 2022.

TU WIEN Bibliothek Your knowledge hub
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.