

# Learning to Aggregate on Structured Data

---

Clemens Damke

*March 4, 2020*





**PADERBORN UNIVERSITY**  
*The University for the Information Society*

Department of Electrical Engineering,  
Computer Science and Mathematics  
Warburger Straße 100  
33098 Paderborn



Intelligent Systems and  
Machine Learning Group (ISG)

Master Thesis

## **Learning to Aggregate on Structured Data**

Clemens Damke

- |             |   |
|-------------|---|
| 1. Reviewer | <b>Prof. Dr. Eyke Hüllermeier</b><br>Intelligent Systems and Machine Learning Group (ISG)<br>Paderborn University |
| 2. Reviewer | <b>Prof. Dr. Axel-Cyrille Ngonga Ngomo</b><br>Data Science Group (DICE)<br>Paderborn University                   |

March 4, 2020

**Clemens Damke**

*Learning to Aggregate on Structured Data*

Master Thesis, March 4, 2020

Reviewers: Prof. Dr. Eyke Hüllermeier and Prof. Dr. Axel-Cyrille Ngonga Ngomo

Supervisor: Vitalik Melnikov

**Paderborn University**

*Intelligent Systems and Machine Learning Group (ISG)*

Heinz Nixdorf Institute

Department of Electrical Engineering, Computer Science and Mathematics

Warburger Straße 100

33098 Paderborn





# Abstract



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Motivation . . . . .	1
1.2	Goals . . . . .	2
1.3	Structure . . . . .	2
<b>2</b>	<b>Related Work</b>	<b>3</b>
2.1	Learning to Aggregate . . . . .	3
2.1.1	Uninorm-Aggregation . . . . .	4
2.1.2	OWA-Aggregation . . . . .	5
2.2	Graph Characterization . . . . .	6
2.2.1	The Graph Isomorphism Problem . . . . .	6
2.2.2	Weisfeiler-Lehman Graph Colorings . . . . .	7
2.2.3	Spectral Graph Theory . . . . .	13
2.3	Graph Classification and Regression . . . . .	15
2.3.1	Explicit Graph Embeddings . . . . .	16
2.3.2	Graph Kernels . . . . .	18
2.3.3	Graph Neural Networks . . . . .	20
<b>3</b>	<b>Learning to Aggregate on Graphs</b>	<b>23</b>
3.1	Formalization of LTA Characteristics . . . . .	23
3.2	An LTA Interpretation of GC/GR Methods . . . . .	23
3.3	LTA on Dynamically Decomposed Graphs . . . . .	23
<b>4</b>	<b>Evaluation</b>	<b>25</b>
<b>5</b>	<b>Conclusion</b>	<b>27</b>
5.1	Review . . . . .	27
5.2	Future Directions . . . . .	27
<b>A</b>	<b>Appendix</b>	<b>31</b>
	<b>Bibliography</b>	<b>33</b>







# Introduction

## 1.1 Motivation

The field of *machine learning* (ML) on graph-structured data has applications in many domains due to the general expressive power of graphs. The three most common types of graph ML problems are

1. **Link prediction:** A graph with an incomplete edge set is given and the missing edges have to be predicted. The suggestion of potential friends in a social network is a typical example for this.
2. **Vertex classification & regression:** Here a class or a score has to be predicted for each vertex of a graph. In social graphs this corresponds to the prediction of properties of individuals, e.g. personal preferences or gender. Another example is the prediction of the amount of traffic at the intersections of a street network.
3. **Graph classification & regression:** In this final problem type a single global class or continuous value has to be predicted for an input graph. The canonical example for this is the prediction of properties of molecule graphs, e.g. the toxicity or solubility of a chemical.

In this thesis we will focus on the last problem type, *graph classification and regression* (GC/GR). An ML method for this problem has to accept graphs of varying size and should be permutation invariant wrt. the order in which graph vertices are provided. Those requirements are not met by the commonly used learners that only accept fixed-size feature vectors as their input, e.g. *logistic regression models* (LRMs), *support vector machines* (SVMs) or multilayer perceptrons.

A GC/GR method has to account for two central aspects of the problem: 1. Local structural analysis and 2. global aggregation. The first aspect is about the extraction of relevant features of substructures of the input graph. The latter is about the way in which the local features are combined into a final class or regression value. The existing GC/GR methods are mostly motivated by local structural graph analysis. The aspect of global aggregation on the other hand is less emphasized by those methods.

There is however a separate branch of research that specifically looks at the problem of learning aggregation functions, called *learning to aggregate* (LTA). Current LTA

approaches explicitly learn an aggregation functions for sets which can be interpreted as graphs without edges. The motivation for this thesis is to generalize LTA from sets to arbitrary graphs. The overall goal is to combine the aggregation learning perspective with existing GC/GR methods.

## 1.2 Goals

To extend LTA to graphs, three goals have to be achieved:

1. **Formalization of LTA:** Before LTA can be extended, its essential characteristics have to be defined. Those characteristics should provide the terminology to formally capture the differences and similarities between LTA and existing GC/GR methods.
2. **Give an LTA interpretation of GC/GR methods:** Using the LTA formalization, representative GC/GR approaches should be restated as LTA instances. Currently there is no comprehensive formulation of the relation between both fields of research; this is addressed by the the second goal.
3. **Define an LTA method for graphs:** Using the LTA perspective on GC/GR, hidden assumptions of the existing approaches should become clear and in which way they share the assumptions of LTA. The last goal is to use those insights to formulate an LTA-GC/GR method that combines ideas from the existing approaches with the LTA assumptions.

## 1.3 Structure

**Chapter 2: Related Work**

**Chapter 3: Learning to Aggregate on Graphs**

**Chapter 4: Evaluation**

**Chapter 5: Conclusion**

# Related Work

Before combining LTA and GC/GR as described in section 1.2, we first give an overview of the state-of-the-art in both fields of research. This is done in three steps:

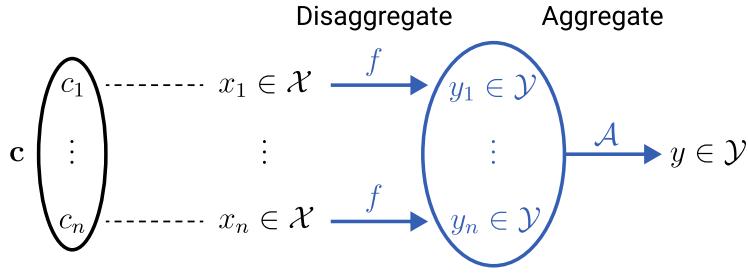
1. We begin with an overview of the existing LTA methods for unstructured inputs.
2. Then we look at the domain of structured inputs. To solve the GC/GR problem, relevant graphs characteristics have to be defined in order to determine the similarity and dissimilarity of graphs. We will look at two common approaches for graph characterization: The Weisfeiler-Lehman algorithm and the notion of graph spectra.
3. Using the described graph characterization approaches, a brief overview of current GC/GR methods will then be given.

## 2.1 Learning to Aggregate

The class of LTA problems was first described by Melnikov and Hüllermeier [MH16]. There an input instance is understood as a composition  $c = \{c_1, \dots, c_n\}$  of so-called constituents, i.e. as a variable-size multiset (denoted as  $\{\cdot\}$ ). The assumption in LTA problems is that for all constituents  $c_i \in c$  a local score  $y_i \in \mathcal{Y}$  is either given or computable. The set of those local scores should be indicative of the overall score  $y \in \mathcal{Y}$  of the composition  $c$ . LTA problems typically require two subproblems to be solved:

1. **Aggregation:** A variadic aggregation function  $\mathcal{A} : \mathcal{Y}^* \rightarrow \mathcal{Y}$  that estimates composite scores has to be learned, i.e.  $y_i \approx \hat{y} = \mathcal{A}(y_1, \dots, y_n)$ . Typically the aggregation function  $\mathcal{A}$  should be associative and commutative to fit with the multiset-structure of compositions.
2. **Disaggregation:** In case the constituent scores  $y_i$  are not given, they have to be derived from a constituent representation, e.g. a vector  $x_i \in \mathcal{X}$ . To learn this derivation function  $f : \mathcal{X} \rightarrow \mathcal{Y}$ , only the constituent vectors  $\{x_i\}_{i=1}^n$  and the composite score  $y$  is given. Thus the constituent scores  $y_i$  need to be *disaggregated* from  $y$  in order to learn  $f$ .

Overall LTA can be understood as the joint problem of learning the aggregation function  $\mathcal{A}$  and the local score derivation function  $f$ . Two main approaches to



**Figure 2.1.** Overview of the structure of LTA for multiset compositions.

represent the aggregation function in LTA problems have been explored.

### 2.1.1 Uninorm-Aggregation

The first approach uses *uninorms* [MH16] to do so. There the basic idea is to express composite scores as fuzzy truth assignments  $y \in [0, 1]$ . Such a composite assignment  $y$  is modeled as the result of a parameterized logical expression of constituent assignments  $y_i \in [0, 1]$ . As the logical expression that thus effectively aggregates the constituents, a uninorm  $U_\lambda$  is used. Depending on the parameter  $\lambda \in [0, 1]$ ,  $U_\lambda$  combines a t-norm  $T$  and a t-conorm  $S$  which are continuous generalizations of logical conjunction and disjunction respectively. One popular choice of norms are the so-called Łukasiewicz norms:

$$\begin{aligned} \text{t-norm } T(a, b) &:= \max\{0, a + b - 1\}, \quad \text{t-conorm } S(a, b) := \min\{a + b, 1\}, \\ \text{uninorm } U_\lambda(a, b) &:= \begin{cases} \lambda T\left(\frac{a}{\lambda}, \frac{b}{\lambda}\right) & \text{if } a, b \in [0, \lambda] \\ \lambda + (1 - \lambda)S\left(\frac{a-\lambda}{1-\lambda}, \frac{b-\lambda}{1-\lambda}\right) & \text{if } a, b \in [\lambda, 1] \\ \lambda \min\{a, b\} & \text{else} \end{cases} \end{aligned} \quad (2.1)$$

At the extreme points  $(0, 0)$ ,  $(0, 1)$ ,  $(1, 0)$  and  $(1, 1)$ ,  $T$  and  $S$  coincide with the Boolean operators  $\wedge$  and  $\vee$ ; the values at all other points are interpolated as shown in fig. 2.2. The uninorm  $U_\lambda$  uses the conjunctive t-norm  $T$  for values below the threshold  $\lambda$  and the disjunctive t-conorm  $S$  for values above the threshold.  $U_\lambda$  therefore smoothly interpolates between a conjunctive and disjunctive operator with the extreme points  $U_1 = T$  and  $U_0 = S$ .

Since t-norms and t-conorms are commutative and associative they can also be applied to non-empty sets of arbitrary size, i.e.  $T(\{y_1, \dots, y_n\}) = T(y_1, T(\{y_2, \dots, y_n\}))$  with fixpoint  $T(\{y\}) = y$ . Using this extension, a uninorm  $U_\lambda$  can be applied to sets which turns it into a parameterized aggregation function  $A_\lambda : [0, 1]^* \rightarrow [0, 1]$ . In this simple model the LTA aggregation problem boils down to the optimization of  $\lambda$ . The LTA disaggregation problem is solved by jointly optimizing a *logistic regression model* (LRM), i.e. the constituent scores  $\{y_i \in [0, 1]\}_{c_i \in c}$  are described by  $y_i = (1 + \exp(-\theta^\top x_i))^{-1}$ . Overall an LTA model is therefore described by the



**Figure 2.2.** The Łukasiewicz norms and the corresponding uninorm for  $\lambda = 0.5$ .

uninorm parameter  $\lambda$  and the regression coefficients  $\theta$ .

## 2.1.2 OWA-Aggregation

Recently Melnikov and Hüllermeier [MH19] have looked at an alternative class of aggregation functions. Instead of using fuzzy logic to describe score aggregation, *ordered weighted average* (OWA) operators were used. OWA aggregators work by sorting the input scores and then weighting them based on their sort position, i.e.

$$\mathcal{A}_\lambda(y_1, \dots, y_n) := \sum_{i=1}^n \lambda_i y_{\pi(i)}, \quad (2.2)$$

where  $\lambda \in \mathbb{R}^n$  is a weight vector with  $\|\lambda\|_1 = 1$  and  $\pi : [n] \rightarrow [n]$  is a sorting permutation of the input scores with  $[n] := \{1, \dots, n\}$  and  $y_i < y_j \Rightarrow \pi(i) < \pi(j)$ . Depending on the choice of the vector  $\lambda$ , the OWA function  $\mathcal{A}_\lambda$  can express common aggregation functions like min (if  $\lambda = (1, 0, \dots, 0)$ ), max (if  $\lambda = (0, \dots, 0, 1)$ ) or the arithmetic mean (if  $\lambda = (\frac{1}{n}, \dots, \frac{1}{n})$ ).

To deal with varying composition sizes  $n$ , the weights  $\lambda_1, \dots, \lambda_n$  can however not be statically assigned. Instead they are interpolated using a so-called *basic unit interval monotone* (BUM) function  $q : [0, 1] \rightarrow [0, 1]$ . It takes constituent positions that are normalized to the unit interval, i.e.  $\frac{i}{n} \in [0, 1]$ . The BUM function  $q$  is then used to interpolate a weight for any normalized sort position via  $\lambda_i := q\left(\frac{i}{n}\right) - q\left(\frac{i-1}{n}\right)$ . Because  $q$  is monotone with  $q(0) = 0$  and  $q(1) = 1$ , it always holds that  $\|\lambda\|_1 = q(1) - q(0) = 1$ . Using this model, the aggregation problem boils down to optimizing the shape of  $q$ .

In the OWA approach the BUM function  $q$  is modeled as a piecewise linear spline. This spline is described by  $m + 1$  points  $\left\{\left(\frac{j}{m}, a_j\right)\right\}_{j=0}^m$ , the so-called knots of the spline. The curve of  $q$  is obtained by linearly interpolating between neighboring knots as shown in fig. 2.3. If  $0 = a_0 \leq a_1 \leq \dots \leq a_m = 1$ ,  $q$  is a BUM function. The LTA aggregation problem is therefore solved by optimizing  $a \in \mathbb{R}^{m+1}$  under this constraint. The disaggregation problem is tackled by adding the scores  $y_1, \dots, y_M \in \mathbb{R}$  to the learnable parameters of the model where  $M$  is assumed to be the finite number of



**Figure 2.3.** Illustration of how  $a$  describes  $q$  and its relation to  $\lambda$  ( $n = 4, m = 7$ ).

constituents. Currently the OWA approach requires all possible constituents to be part of the training dataset since it does not consider constituent features  $x_i \in \mathcal{X}$  to predict the scores of previously unseen constituents.

## 2.2 Graph Characterization

**Definition 2.1.** A graph  $G := (\mathcal{V}_G, \mathcal{E}_G)$  consists of a finite set of vertices  $v_i \in \mathcal{V}_G$  and a set of edges  $e_{ij} = (v_i, v_j) \in \mathcal{E}_G \subseteq \mathcal{V}_G^2$ . Optionally discrete vertex labels  $l_G[v_i] \in L_V$  or edge labels  $l_G[e_{ij}] \in L_E$  may be associated with all vertices  $v_i \in \mathcal{V}_G$  and edges  $e_{ij} \in \mathcal{E}_G$  respectively. Also continuous feature vectors  $x_G[v_i] \in \mathcal{X}_V, x_G[e_{ij}] \in \mathcal{X}_E$  may be given. If  $\mathcal{X}_E = \mathbb{R}$ ,  $x_G[e_{ij}]$  can be interpreted as an edge weight of  $e_{ij}$ .

In this thesis all graphs  $G$  are assumed to be undirected if not explicitly stated otherwise, i.e.  $e_{ij} \in \mathcal{E}_G \leftrightarrow e_{ji} \in \mathcal{E}_G \wedge l_G[e_{ij}] = l_G[e_{ji}] \wedge x_G[e_{ij}] = x_G[e_{ji}]$ . We denote the set of all undirected graphs as  $\mathcal{G}$ . Additionally we denote  $G[S] := (S, \mathcal{E}_G \cap S^2)$  as the subgraph of  $G$  induced by  $S \subseteq \mathcal{V}_G$ .

To classify or score a graph, it first needs to be characterized by a set of relevant properties. The most strict characterization of a graph is its so-called *isomorphism class*. It uniquely identifies a graph but lacks any notion of similarity between non-identical graphs. We will begin with a brief definition of this strict isomorphism-based graph characterization. Then two less strict characterization approaches are described; the so called Weisfeiler-Lehman coloring and the notion of graph spectra. They are the theoretical foundation of many current GC/GR methods.

### 2.2.1 The Graph Isomorphism Problem

In order to process a graph  $G$  of size  $n = |\mathcal{V}_G|$ , one is forced to choose some encoding, e.g. an adjacency matrix  $A_G \in \{0, 1\}^{n \times n}$ . Such an encoding introduces a vertex ordering  $v_1, \dots, v_n$  that does not carry any semantic meaning. Consequently there are  $n!$  equivalent encodings of  $G$ . To represent those encodings we introduce the

notion of ordered induced subgraphs.

**Definition 2.2.** For all vertex  $k$ -tuples  $v = (v_1, \dots, v_k) \in \mathcal{V}_G^k$  let  $\hat{v} = \{v_i \mid i \in [k]\}$ . Then  $G[v] := (\hat{v}, \mathcal{E}_G \cap \hat{v}^2, v)$  is called the ordered subgraph of  $G$  induced by  $v$ . If  $\hat{v} = \mathcal{V}_G$  and  $k = |\mathcal{V}_G|$ , we call  $v$  an ordering of  $G$  and  $G[v]$  an encoding of  $G$ .

**Definition 2.3.** Two ordered induced subgraphs  $G[v]$  and  $H[w]$  with  $(v_1, \dots, v_k) \in \mathcal{V}_G^k$  and  $(w_1, \dots, w_k) \in \mathcal{V}_H^k$  are *equivalent* ( $G[v] \equiv H[w]$ ) iff.

$$\begin{aligned} & \forall i, j \in [k] : (v_i, v_j) \in \mathcal{E}_G \leftrightarrow (w_i, w_j) \in \mathcal{E}_H \\ & \wedge \forall i, j \in [k] : l_G[v_i, v_j] = l_H[w_i, w_j] \quad \wedge \forall i \in [k] : l_G[v_i] = l_H[w_i] \\ & \wedge \forall i, j \in [k] : x_G[v_i, v_j] = x_H[w_i, w_j] \quad \wedge \forall i \in [k] : x_G[v_i] = x_H[w_i]. \end{aligned}$$

Using this notion of equivalence, we call  $[G[v]] := \{G[w] \mid G[w] \equiv G[v] \wedge w \in \mathcal{V}_G^*\}$  the ordered subgraph equivalence class of  $G[v]$ .

**Definition 2.4.** Two graphs  $G$  and  $H$  are *isomorphic* ( $G \simeq H$ ) iff. there are equivalent encodings  $G[v] \equiv H[w]$  of them. Consequently  $[G] := \{H \mid H \simeq G\}$  is called the isomorphism class of  $G$ .

The goal of the *graph isomorphism* (GI) problem is to check whether  $G \simeq H$  for two arbitrary graphs. Even though there is no known universal algorithm in  $P$  that solves GI, Babai et al. [Bab+80] showed that almost all graphs can be trivially distinguished in linear time. More recently Babai [Bab15] also presented a quasipolynomial upper time bound for the remaining hard GI instances. For all practical purposes, GI can therefore be solved efficiently; e.g. via the nauty program [MP][MP13]. One important subroutine in most GI checkers is the Weisfeiler-Lehman algorithm which will be described next.

## 2.2.2 Weisfeiler-Lehman Graph Colorings

The *Weisfeiler-Lehman* (WL) algorithm [WL68][Cai+92] characterizes a graph  $G$  by assigning discrete labels  $c \in \mathcal{C}$ , called *colors*, to vertex  $k$ -tuples  $(v_1, \dots, v_k) \in \mathcal{V}_G^k$ , where  $k \in \mathbb{N}$  is the freely choosable *WL-dimension*. A mapping  $\chi_{G,k} : \mathcal{V}_G^k \rightarrow \mathcal{C}$  is called a *k-coloring* of  $G$ .

**Definition 2.5.** A coloring  $\chi'$  *refines*  $\chi$  ( $\chi' \preceq \chi$ ) iff.  $\forall a, b \in \mathcal{V}_G^k : \chi(a) \neq \chi(b) \rightarrow \chi'(a) \neq \chi'(b)$ , i.e.  $\chi'$  distinguishes at least those tuples that are distinguished by  $\chi$ .

**Definition 2.6.** Two colorings  $\chi$  and  $\chi'$  are *equivalent* ( $\chi \equiv \chi'$ ) iff.  $\chi \preceq \chi' \wedge \chi' \preceq \chi$ , i.e.  $\chi$  is identical to  $\chi'$  up to color substitutions.

The  $k$ -dimensional WL algorithm ( $k$ -WL) works by iteratively refining  $k$ -colorings  $\chi_{G,k}^{(0)} \succeq \chi_{G,k}^{(1)} \succeq \dots$  of a given graph  $G$  until the convergence criterion  $\chi_{G,k}^{(i)} \equiv \chi_{G,k}^{(i+1)}$

is satisfied. We denote the final, maximally refined  $k$ -WL coloring with  $\chi_{G,k}^*$ .

**Definition 2.7.** The color distribution  $dist_{\chi_{G,k}} : \mathcal{C} \rightarrow \mathbb{N}$  of a  $k$ -coloring  $\chi_{G,k}$  counts each color  $c \in \mathcal{C}$  in the coloring, i.e.  $dist_{\chi_{G,k}}(c) := |\{v \in \mathcal{V}_G^k \mid \chi_{G,k}(v) = c\}|$ .

**Definition 2.8.** Two graphs  $G$  and  $H$  are  $k$ -WL *distinguishable* ( $G \not\simeq_k H$ ) iff. there exists a color  $c \in \mathcal{C}$  s.t.  $dist_{\chi_{G,k}^*}(c) \neq dist_{\chi_{H,k}^*}(c)$ .

As we will see, the way in which WL colorings are refined is vertex order invariant; thus any difference in the final coloring of two graphs always implies the non-isomorphism of the colored graphs, i.e.  $G \not\simeq_k H \implies G \not\simeq H$ . The opposite does however not necessarily hold; two  $k$ -WL indistinguishable graphs are not always isomorphic, i.e.  $\exists G, H : G \simeq_k H \wedge G \not\simeq H$ .

In addition to the binary aspect of WL distinguishability and its relation to the GI problem, WL colorings are also useful for more fuzzy graph similarity comparisons as we will see in section 2.3.2 when we look at graph kernels. Before that however, the details of WL color refinement strategy have to be described. We begin with the color refinement algorithm for the most simple case of  $k = 1$ . Then the definitions and intuitions from the 1-dimensional case are extended to its higher-dimensional generalization. Lastly we will discuss the discriminative power of the WL algorithm and its relation to the WL-dimension  $k$ .

## The 1-dimensional WL algorithm

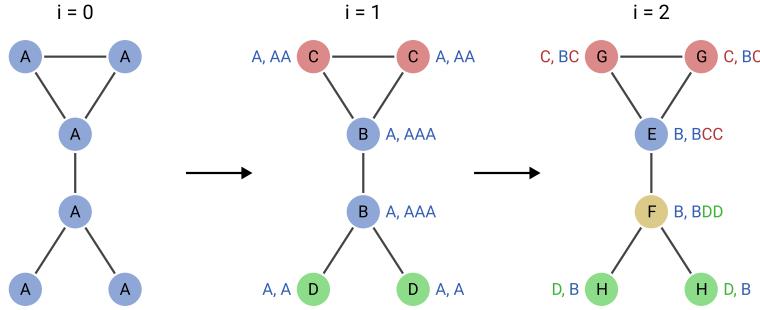
In the 1-dimensional WL algorithm, a color is assigned to each vertex of a graph. If the vertices  $v \in \mathcal{V}_G$  of the input graph  $G$  are labeled, those labels  $l_G[v] \in L_V \subseteq \mathcal{C}$  can be used as the initial graph coloring  $\chi_{G,1}^{(0)}(v) := l_G[v]$ . Since WL colors are inherently discrete, continuous vertex feature vectors  $x_G[v]$  are not considered here. For unlabeled graphs a constant coloring is used, e.g.  $\forall v \in \mathcal{V}_G : \chi_{G,1}^{(0)}(v) = \text{A}$  for some initial color  $\text{A} \in \mathcal{C}$ . In each iteration of the 1-WL color refinement algorithm, the following neighborhood aggregation scheme is used to compute a new color for all vertices:

**Definition 2.9.**  $\chi_{G,1}^{(i+1)}(v) := h \left( \chi_{G,1}^{(i)}(v), \{\chi_{G,1}^{(i)}(u) \mid u \in \Gamma_G(v)\} \right)$ , with  $\Gamma_G(v)$  denoting the set of adjacent vertices of  $v \in \mathcal{V}_G$  and  $h : \mathcal{C}^* \rightarrow \mathcal{C}$  denoting a perfect hash function that assigns a unique color to each finite combination of colors.

In practice the hash function  $h$  is usually defined lazily by using  $\mathcal{C} \subseteq \mathbb{N}$  and enumerating color combinations in whichever order they are hashed s.t. a new color is introduced every time a previously unseen color combination appears at runtime<sup>1</sup>.

---

<sup>1</sup> Note that, even though  $\mathcal{C}$  is formally countably infinite, we will assume that it only contains the finite number of colors occurring in a given finite graph dataset.



**Figure 2.4.** Example 1-WL color refinement steps. After two iterations the coloring stabilizes. Each vertex  $v$  is labeled with its current color and has its previous color and the colors of the hashed neighbors  $\Gamma_G(v)$  written next to it (see definition 2.9).

### The $k$ -dimensional WL algorithm

As we just saw, the 1-WL algorithm iteratively refines colorings of single vertices. While the obtained colorings differ for most non-isomorphic graphs  $G \not\simeq H$ , 1-WL does not generally solve the GI problem as illustrated in fig. 2.5.

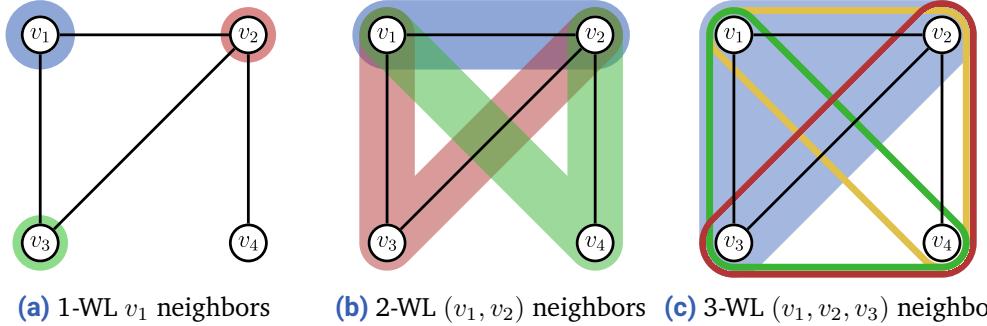


**Figure 2.5.** Two simple non-isomorphic graphs that are indistinguishable by 1-WL.

By extending WL to higher dimensions, such 1-WL indistinguishable cases can however be handled. Analogous to the 1-dimensional definition 2.9, the  $k$ -dimensional color refinement step is defined by

$$\begin{aligned} \text{Definition 2.10. } \chi_{G,k}^{(i+1)}(s) &:= h\left(\chi_{G,k}^{(i)}(s), \{\chi_{G,k}^{(i)}(s[u/1]), \dots, \chi_{G,k}^{(i)}(s[u/k])\} \mid u \in \mathcal{V}_G\right) \\ \text{with } s = (v_1, \dots, v_k) \in \mathcal{V}_G^k, \quad s[u/j] &:= (v_1, \dots, v_{j-1}, u, v_{j+1}, \dots, v_k). \end{aligned}$$

In 1-WL a vertex color is refined by combining the colors of neighboring vertices. In  $k$ -WL the color of a  $k$ -tuple  $s \in \mathcal{V}_G^k$  is refined by combining the colors of its neighborhood which is defined as the set of all  $k$ -tuples in which at most one vertex differs from  $s$ . Note that each vertex  $k$ -tuple has one neighbor for each  $u \in \mathcal{V}_G$ , each of which is a  $k$ -tuple of vertex  $k$ -tuples. This more abstract notion of neighborhood is illustrated in fig. 2.6. For  $k = 2$  this means that each potential edge  $(v, w) \in \mathcal{V}_G^2$  has all possible paths of length 2 from  $v$  to  $w$  as its neighbors (see fig. 2.6b). Also note that, even though  $k$ -WL refines  $k$ -tuple colors, lower-dimensional structures still get their own colors since a tuple does not have to consist of distinct vertices, i.e. in  $k$ -WL the color of a single vertex  $v \in \mathcal{V}_G$  is described by  $\chi_{G,k}^*(s)$  for  $s = (v, \dots, v) \in \mathcal{V}_G^k$ .



**Figure 2.6.** Tuple neighborhoods for different values of  $k$ . The vertices highlighted in blue form the root tuple  $s \in \mathcal{V}_G^k$  whose neighbors are shown; for simplicity neighbors with  $u \in s$  are left out (see definition 2.10). Each neighbor is highlighted with a different color, except for 3-WL where the red, green and yellow triples actually form the single neighbor for  $u = v_4$ .

Let us now look at how the tuple colors are initialized. For this we use the ordered subgraph equivalence classes  $[G[s]]$  (see definition 2.3) which determine the initial color  $\chi_{G,k}^{(0)}(v)$  of each  $k$ -tuple  $s$ . For  $k = 1$  the equivalence class of a single vertex  $v$  directly corresponds to its label  $l_G[v]$ . More generally for  $k > 1$  this means that

$$\chi_{G,k}^{(0)}(s) = \chi_{G,k}^{(0)}(t) \iff G[s] \equiv G[t]. \quad (2.3)$$

Note that there is a fundamental difference in how the adjacency information encoded in  $\mathcal{E}_G$  is used in 1-WL vs.  $k$ -WL: In 1-WL a vertex coloring by itself cannot encode adjacency which is why this information is explicitly incorporated in each refinement step via  $\Gamma_G$  (see definition 2.9). In  $k$ -WL on the other hand each pair of vertices  $(v, u) \in \mathcal{V}_G^2$  appears in at least one  $k$ -tuple (assuming  $k \geq 2$ ) and therefore has at least one color which can implicitly encode the adjacency information. Edges and non-edges are colored differently in the initial coloring since  $G[(v, u)] \neq G[(w, u)]$  if  $(v, u) \in \mathcal{E}_G$  but  $(w, u) \notin \mathcal{E}_G$ ; thus no explicit adjacency information is needed in the  $k$ -WL color refinement step in definition 2.10.

## Discriminative Power of WL

Now we will look at the types of graphs that can be distinguished by WL in relation to the WL-dimension  $k$ .

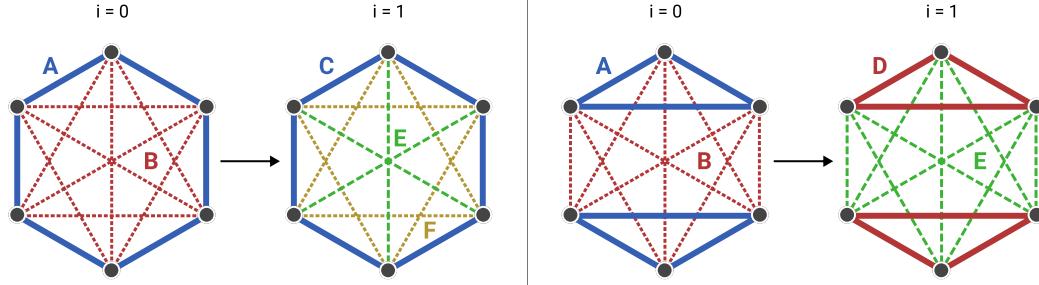
**Lemma 2.11.**  $G \not\sim_k H \implies G \not\sim_{k+1} H$  ( $(k+1)$ -WL is at least as powerful as  $k$ -WL).

*Proof Sketch.* All  $k$ -tuples can be mapped to  $(k+1)$ -tuples via  $\varphi(v_1, \dots, v_k) := (v_1, \dots, v_k, v_k)$ . For each neighbor  $(s[u/1], \dots, s[u/k])$  of  $s \in \mathcal{V}_G^k$ , there is a corresponding neighbor  $(\varphi(s[u/1]), \dots, \varphi(s[u/k]), \varphi(s[u/k]))$  of  $\varphi(s)$ . Using eq. (2.3) (for  $i = 0$ ) and definition 2.10 (for  $i > 0$ ) it follows that  $\forall s, t \in \mathcal{V}_G^k : \chi_{G,k}^{(i)}(s) \neq \chi_{G,k}^{(i)}(t) \rightarrow \chi_{G,k+1}^{(i)}(\varphi(s)) \neq \chi_{G,k+1}^{(i)}(\varphi(t))$ . The lemma then follows by definition 2.8.  $\square$

**Proposition 2.12** (see Immerman and Lander [IL90] for the proof). *For all  $k \in \mathbb{N}$  there are non-isomorphic graphs  $G \not\simeq H$  of size  $\mathcal{O}(k)$  with  $G \simeq_k H$ .*

**Corollary 2.13.** *The discriminative power of  $k$ -WL grows monotonously with  $k$  and never converges, i.e. for all  $k \in \mathbb{N}$  there is an  $l \in \mathbb{N}$  s.t. the set of  $k$ -WL distinguishable graphs is a proper subset of the  $(k + l)$ -WL distinguishable graphs.*

*Proof Sketch.* Note that  $k$ -WL trivially solves the GI problem for all graphs of size  $\leq k$  via the initial coloring (see eq. (2.3)). Using lemma 2.11 and proposition 2.12 the corollary directly follows.  $\square$



**Figure 2.7.** Two non-isomorphic graphs with  $G \simeq_1 H$  and  $G \not\simeq_2 H$ .

Figure 2.7 illustrates corollary 2.13 by showing how 2-WL is able to distinguish the two 1-WL indistinguishable graphs from fig. 2.5. Since the time complexity of  $k$ -WL grows exponentially with  $k$  [IL90, cor. 1.9.7], it does not provide an efficient universal solution to GI. However it turns out that almost all graphs are WL distinguishable even for a small constant  $k$ .

For  $k = 1$  Babai et al. [Bab+80] have shown that two randomly selected non-isomorphic graphs  $G \not\simeq H$  of size  $n$  are 1-WL indistinguishable with probability  $n^{-1/7}$ . Thus 1-WL is already able to distinguish most graphs; it fails however to distinguish any pair of unlabeled  $d$ -regular graphs of the same size [IL90, cor. 1.8.5].

**Definition 2.14.** A graph  $G$  is called  $d$ -regular ( $rg_d(G)$ ) iff.  $\forall v \in \mathcal{V}_G : |\Gamma_G(v)| = d$ .

We have already seen this in fig. 2.5 since the “six-cycle” and the “two-triangles” graphs are in fact both 2-regular and of size 6. This restriction alone is typically not an issue in the context of GC/GR though, as molecular structures or social interaction graphs for example are rarely perfectly regular. A more relevant restriction of 1-WL is the fact that it is unable to detect cycles of length  $m \geq 3$ .

**Definition 2.15.**  $k$ -WL computes a function  $f : \mathcal{G} \rightarrow Y$  iff. that function can be expressed as  $f(G) = g(\text{dist}_{\chi_{G,k}^*})$  via some function  $g : (\mathcal{C} \rightarrow \mathbb{N}) \rightarrow Y$ .

**Definition 2.16.**  $k$ -WL counts a certain subgraph  $S$  iff. it computes  $\text{count}_S(G) := |\{\hat{v} \mid v \in \mathcal{V}_G^* \wedge G[v] \equiv S\}|$  (see definition 2.2). Similarly  $k$ -WL detects a subgraph  $S$  iff. it computes  $\text{contain}_S(G) := \mathbb{1}[\text{count}_S(G) > 0]$ , with  $\mathbb{1}$  denoting the indicator function.

To see why 1-WL is unable to detect  $m$ -cycles in graphs, note that fig. 2.5 already contradicts the positive statement for  $m = 3$ ; this counterexample can be trivially generalized to all larger  $m$  by replacing the “six-cycle” graph with a “ $(2m)$ -cycle-graph” graph and the “two-triangles” graph with a “two- $m$ -cycles” graph which preserves the 2-regularity and therefore the 1-WL indistinguishability.

This cycle-detection restriction of 1-WL is relevant in practice because cycle counts are used in many domains to analyze graphs, e.g. triangle counts are commonly used in social network analysis to find interaction clusters [Mil02][New03][Wel+07] and the detection of 4-, 5- or 6-cycles is required to determine important chemical properties like the aromaticity of a molecule [AB73][Kek66].

If we increase the WL-dimension to  $k = 2$ , the described 1-WL restrictions no longer apply. 2-WL is able to distinguish more than  $1 - \mathcal{O}(1/n)$  of the regular  $n$ -vertex graphs [IL90, cor. 1.8.6] and can even count cycles:

**Proposition 2.17** (see Fürer [Fü17] and Arvind et al. [Arv+19] for the full proof). *2-WL is able to count  $m$ -cycles for  $m \leq 7$  but it cannot detect 8-cycles.*

*Proof Sketch.* Only the idea behind triangle and 4-cycle counting is outlined here to give an intuition for why proposition 2.17 holds. Note that the 2-WL neighbors of each edge  $(v, w) \in \mathcal{E}_G$  correspond to all possible paths  $(v, u, w)$  of length 2, i.e. all possible triangles. By definition 2.10 there must thus be a color subset  $C_j^\triangle \subseteq \mathcal{C}$  representing that an edge is involved in exactly  $j$  triangles after one refinement step. 2-WL then trivially counts triangles by setting  $g(\text{dist}_\chi) = \frac{1}{3} \sum_j j \sum_{c \in C_j^\triangle} \text{dist}_\chi(c)$  to satisfy definition 2.15. Analogously for 4-cycles, let  $C_j^\square \subseteq \mathcal{C}$  be the colors indicating a non-edge  $(v, w) \notin \mathcal{E}_G$  with  $j$  common vertex neighbors  $u \in \Gamma_G(v) \cap \Gamma_G(w)$ . The number of 4-cycles is then determined by the colors of the diagonals through them via  $g(\text{dist}_\chi) = \frac{1}{2} \sum_{j \geq 2} \binom{j}{2} \sum_{c \in C_j^\square} \text{dist}_\chi(c)$ . Using a similar but more involved combinatorial argument requiring multiple color refinement steps, 5-, 6- & 7-cycle counting can be shown.  $\square$

As we just saw, 2-WL is significantly more powerful than 1-WL. Though, by proposition 2.12, there are of course still 2-WL indistinguishable graphs; among others those are the strongly regular graphs  $srg_{n,d,\lambda,\mu}(G)$ .

$$\begin{aligned} \text{Definition 2.18. } srg_{n,d,\lambda,\mu}(G) \iff & |\mathcal{V}_G| = n \quad \wedge \quad \forall v \in \mathcal{V}_G : |\Gamma_G(v)| = d \\ & \wedge \forall (v, w) \in \mathcal{E}_G : |\Gamma_G(v) \cap \Gamma_G(w)| = \lambda \\ & \wedge \forall (v, w) \in \mathcal{V}_G^2 \setminus \mathcal{E}_G : |\Gamma_G(v) \cap \Gamma_G(w)| = \mu \end{aligned}$$

Generally this restriction of 2-WL is not an issue since strongly regular graphs do not appear in typical GC/GR datasets. By going to  $k = 3$ , even some strongly regular graphs as well as all planar graphs can be distinguished though [Kie+17]. Apart from that there are currently few results regarding the classes of distinguishable graphs and computable functions for even higher WL-dimensions.

### 2.2.3 Spectral Graph Theory

Let us now look at an alternative perspective on graph characterization which is provided by *spectral graph theory*. While WL characterizes a graph via its color distribution  $\text{dist}_{\chi_{G,k}^*}$ , the spectral approach uses its so-called *spectrum*  $\lambda_G = (\lambda_{G,1}, \dots, \lambda_{G,n}) \in \mathbb{R}^n$  where  $n = |\mathcal{V}_G|$ . The key idea behind this is to interpret the adjacency matrix  $A_G \in \mathbb{R}^{n \times n}$  of  $G$  not simply as an encoding of the edges ( $A_{G,i,j} = \mathbb{1}[(v_i, v_j) \in \mathcal{E}_G]$ ) but as a linear operator  $A_G : (\mathcal{V}_G \rightarrow \mathbb{R}) \rightarrow (\mathcal{V}_G \rightarrow \mathbb{R})$  acting on the vector space of real-valued functions with a vertex domain. This means that  $A_G$  transforms so-called *graph signals*  $x : \mathcal{V}_G \rightarrow \mathbb{R}$  which are functions assigning *signal strengths* to vertices. By applying  $A_G x$  the signal strength  $x[v_i]$  of each vertex is added to the signal strengths of its neighbors  $\Gamma_G(v_i)$ . Using this functional perspective, one can characterize graphs via the *Fourier transform* (FT). We will now see how this is done; however, since a comprehensive description of spectral graph theory would exceed the scope of this thesis, only a brief overview will be given. For a more detailed introduction to the field we refer to Shuman et al. [Shu+13].

**The classical Fourier transform** Let us begin by describing the FT for the more common case of functions with real domains. All functions  $f : \mathbb{R} \rightarrow \mathbb{R}$  can be interpreted as infinite-dimensional vectors  $f = \int_{\mathbb{R}} f(t) b_t dt$  with  $b_t : \mathbb{R} \rightarrow \mathbb{R}$  being a standard basis vector/function defined as  $b_t(s) := \begin{cases} 1 & \text{if } s = t \\ 0 & \text{else} \end{cases}$ . The values of  $f$  are then described as its  $b_t$  components  $\langle b_t, f \rangle = f(t)$ , where  $\langle b_t, \cdot \rangle = \delta_t(\cdot)$  denotes the Dirac delta function translated by  $t$ . The Fourier transform  $\hat{f}$  of  $f$  corresponds to a change of basis from the standard basis vectors  $b_t$  to the Fourier basis vectors  $u_{\xi}(t) := e^{2\pi i \xi t}$ , i.e.  $f = \int_{\mathbb{R}} \hat{f}(t) u_{\xi} dt$ . The Fourier basis is characterized by the fact that it is an eigenbasis of the so-called *Laplace operator*  $\Delta$ , i.e.  $\Delta u_{\xi} = \lambda_{\xi} u_{\xi}$  with  $\lambda_{\xi}$  being the eigenvalue corresponding to  $u_{\xi}$ . In the real domain this Laplace operator  $\Delta$  is effectively the same as the second-derivative  $\frac{d^2}{dt^2}$ . Thus it is easy to see that indeed  $\Delta u_{\xi} = -\frac{d^2}{dt^2} u_{\xi} = \lambda_{\xi} u_{\xi}$  for  $\lambda_{\xi} = (2\pi\xi)^2$ .

**The graph Fourier transform** To extend the notion of the FT from real functions  $f : \mathbb{R} \rightarrow \mathbb{R}$  to graph signals  $x : \mathcal{V}_G \rightarrow \mathbb{R}$  we need to define a graph variant of the Laplace operator  $\Delta$ . It turns out that there are multiple possible ways to do so, the most simple being the so-called *combinatorial graph Laplacian*.

**Definition 2.19.** The combinatorial graph Laplacian  $L_G \in \mathbb{R}^{n \times n}$  is defined as

$$L_G := D_G - A_G \quad \text{with the degree matrix } D_G := \begin{pmatrix} d_1 & & \\ & \ddots & \\ & & d_n \end{pmatrix}, d_i := |\Gamma_G(v_i)|.$$

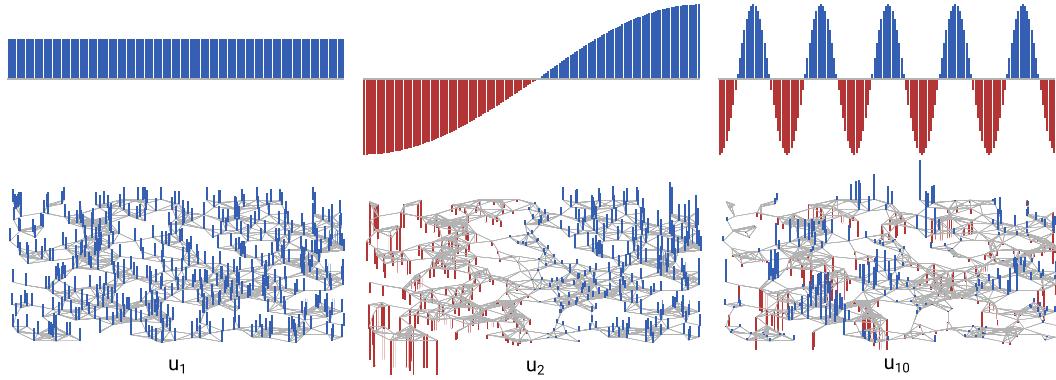
Using this definition, applying  $L_G x$  is analogous to taking the second derivative  $\frac{d^2}{dt^2} f$ .

Putting both Laplacian variants,  $L_G$  and  $\frac{d^2}{dt^2}$ , side-by-side gives an intuition for why this is the case:

$$-\frac{d^2}{dt^2} f(t) = \lim_{h \rightarrow 0} \frac{1}{h^2} (\underbrace{f(t) - f(t-h)}_{\Delta_{t,t-h}} + \underbrace{f(t) - f(t+h)}_{\Delta_{t,t+h}}) \quad \left| \quad L_G x[v] = \sum_{u \in \Gamma_G(v)} \underbrace{(x[v] - x[u])}_{\Delta_{v,u}}$$

The second derivative of a function  $f$  essentially averages the value differences in the neighborhood of a point  $t$ . For real-valued functions this neighborhood only consists of the two infinitesimally close points to the left and to the right of  $t$ , i.e.  $t-h$  and  $t+h$ . The combinatorial graph Laplacian represents the same operation, where each point/vertex  $v$  might however have more than two neighbors  $u \in \Gamma_G(v)$  that need to be averaged.

The FT of a graph signal  $x$  then is  $\hat{x}[i] = \langle u_{G,i}, x \rangle$  with  $u_G = \{u_{G,i}\}_{i=1}^n$  being the eigenvectors of  $L_G$ , s.t.  $x = \sum_{i=1}^n \hat{x}[i] u_{G,i}$ . Note that  $L_G$  is guaranteed to only have real eigenvectors  $u_G$  and eigenvalues  $\lambda_G$  due to the fact that we only consider undirected graphs with symmetric adjacency matrices  $A_G$ . By convention we assume that the eigenvalues are ordered ascendingly:  $\lambda_{G,0} \leq \dots \leq \lambda_{G,n}$ . Those eigenvalues are called the *spectrum of G* and they are a vertex-permutation-invariant graph characterization. Intuitively those eigenvalues describe the connectivity between different parts of the graph. The first eigenvalues represent connectivity at a general, coarse level while the last eigenvalues represent the connectivity of finer substructures. Figure 2.8 illustrates this idea. A concrete example for the relation between a graph's structure and its spectrum is the fact that a graph with  $m$  connected components has exactly  $m$  zero eigenvalues, i.e.  $0 = \lambda_{G,1} = \dots = \lambda_{G,m} < \lambda_{G,m+1}$ . For a more detailed discussion of this relation we refer to Das [Das04].



**Figure 2.8.** Comparison between the basis functions/vectors of the real domain FT and the graph FT. For the eigenfunctions on the upper half only the real cosine components of the complex exponentials are shown.  
BASED ON: [SHU+13]

**Definition 2.20.** Two graphs  $G$  and  $H$  are *cospectral* ( $G \simeq_\lambda H$ ) iff.  $\forall i : \lambda_{G,i} = \lambda_{H,i}$ .

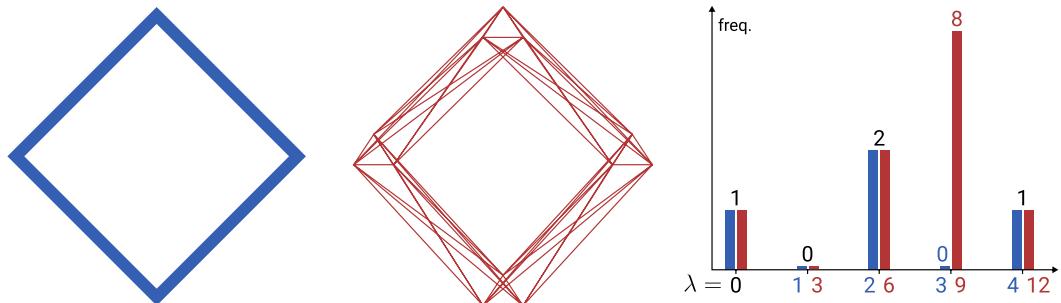
Alzaga et al. [Alz+10] have shown that, while the cospectrality test can be more

powerful than 1-WL, it is always weaker than 2-WL, i.e.  $G \simeq_2 H \implies G \simeq_\lambda H$ . Despite this limit on the discriminative power of the graph spectrum, it is still useful for determining graph similarity, e.g. by defining a distance measure on the vector space of spectra (see Gu et al. [Gu+15]).

As briefly mentioned before, the combinatorial graph Laplacian  $L_G$  is not the only possible choice of graph Laplacian. Note that the eigenvalues of  $L_G$  grow with the number of edges<sup>2</sup>. Consequently a small graph  $G$  will typically have a large spectral distance to a large graph  $H$  (assuming  $|\mathcal{V}_G| \ll |\mathcal{V}_H|$ ) irrespective of their similarity when ignoring the scale difference. In domains where the absolute size of a graph should not influence its spectral characterization, it can therefore be useful to normalize the spectrum. One common way to do so is via the so-called *symmetric normalized Laplacian*  $L_G^{\text{sym}}$ .

**Definition 2.21.**  $L_G^{\text{sym}} := D^{-\frac{1}{2}} L_G D^{-\frac{1}{2}}$  is the symmetric normalized Laplacian of  $G$ .

The eigenvalues  $\lambda_{G,i}^{\text{sym}}$  of this Laplacian all lie in the range  $[0, 2]$ . Figure 2.9 illustrates how this makes it possible to compare the structure of graphs across varying vertex counts. Besides  $L_G^{\text{sym}}$  there are also other ways to normalize the spectrum, e.g. by using the random walk Laplacian  $L_G^{\text{rw}} := D^{-1} L_G$ , which will however not be covered here (see Shuman et al. [Shu+13]).



**Figure 2.9.** Comparison of the unnormalized  $L_G$  spectrum and the normalized  $L_G^{\text{sym}}$  spectrum. Two structurally similar graphs of different size are compared. The **large graph** is derived by replacing each vertex of the **small 4-cycle graph** by a triangle. While their unnormalized spectra have different absolute ranges ( $[0, 4]$  vs.  $[0, 12]$ ), the histogram on the right shows that their normalized eigenvalue distributions align.

## 2.3 Graph Classification and Regression

The existing approaches to tackle the GC/GR problem can be categorized into three main families: 1. Explicit graph embeddings, 2. graph kernels and 3. graph neural networks. We will now look at the characteristics of those families and give a brief overview of specific methods.

<sup>2</sup> This is trivially shown by  $\sum_{i=1}^n \lambda_{G,i} = \text{Tr}(L_G) = \text{Tr}(D_G) = |\mathcal{E}_G|$ .

### 2.3.1 Explicit Graph Embeddings

The basic idea of explicit graph embedding approaches is to map a graph  $G \in \mathcal{G}$  to some vector in a finite vector space  $\mathcal{X} = \mathbb{R}^d$ . A function  $f : \mathcal{G} \rightarrow \mathcal{X}$  is called a *graph embedding function*. By embedding a graph into  $\mathcal{X}$ , any classification or regression algorithm that works with vectors can then be applied to solve the GC/GR problem.

The so-called vertex embedding problem is closely related to the graph embedding problem. As the name suggests, a *vertex embedding function*  $f_G$  maps all vertices  $v \in \mathcal{V}_G$  to  $\mathcal{X}$ . The embedding vector  $f_G(v)$  ideally encodes relevant information about a vertex and its structural position in  $G$ . It can be used to solve the vertex classification and regression problem via arbitrary ML methods for vectors. We will now look at two main families of explicit graph and vertex embedding approaches.

#### Fingerprint Embeddings

The first works on graph embeddings were motivated by the study of chemical structures [AB73][WW86]. There a molecule can be interpreted as a labeled graph for which the GC/GR problem corresponds to the prediction of some chemical property, e.g. toxicity or solubility. So-called *fingerprint embeddings* try to match a fixed set of subgraphs  $S_1, \dots, S_d$  to the input graph. The embedding of a graph  $G$  is a binary vector  $f(G) = x \in \{0, 1\}^d$  with  $x_i := \text{contains}_{S_i}(G)$  (see definition 2.16). This simple approach usually requires a careful choice of subgraphs but can still be competitive with the other more recent approaches we will look at in the following sections. Fingerprint embeddings are for example used in multiple state-of-the-art toxicity prediction tools like RASAR [Lue+18][Tox], ProTox [Drw+14][Ban+18b][Ban+18a] or the Toxicity Estimation Software Tools [Tes].

#### Skip-gram inspired Embeddings

Skip-gram embeddings were introduced by Mikolov et al. as part of the well-known word2vec [Mik+13] word embedding method from natural language processing. While a fingerprint embedding explicitly assigns an interpretation to each embedding dimension (i.e. to each standard basis vector), a skip-gram embedding only optimizes the distance between embedding vectors based on the similarity of the embedded instances without providing an interpretation of the embedding dimensions.

**word2vec** Let us first look at the word2vec skip-gram method. It gets a sequence of words  $(w_0, \dots, w_n)$  as input and outputs embedding vectors  $f(w_0), \dots, f(w_n) \in \mathbb{R}^d$ . To do this the context  $C_k(w_i) = \{w_{i-k}, \dots, w_{i+k}\}$  is computed for all words where

$w_i$  is the so-called *context root*. The word contexts are then used to optimize the following log-likelihood objective:

$$\max_{f, f_C} \sum_{i=1}^n \log P(C_k(w_i) | w_i) = \max_{f, f_C} \sum_{i=1}^n \sum_{w_j \in C_k(w_i)} \left[ f(w_i)^\top f_C(w_j) - \log Z_{w_i} \right] \quad (2.4)$$

with  $P(C_k(w_i) | w_i) := \prod_{w_j \in C_k(w_i)} \overbrace{\frac{\exp(f(w_i)^\top f_C(w_j))}{Z_{w_i}}}^{P(w_j | w_i)}$  and  $Z_{w_i} := \sum_{j=1}^n \exp(f(w_i)^\top f_C(w_j))$

word2vec essentially uses an expectation maximization scheme to maximize the probabilities  $P(w_j | w_i)$  of observing the context words  $w_j \in C_k(w_i)$  of all words  $w_i$ . Those probabilities are described by the overlap of the embeddings  $f(w_i)$  of words  $w_i$  and the embeddings  $f_C(w_j)$  of their context words  $w_j$ . Intuitively this means that words with similar contexts will be mapped close to each other in the embedding space. Note that word2vec actually finds two embeddings  $f(w)$  and  $f_C(w)$  for each word of which only the first is returned. The two embeddings represent two different perspectives on words:  $f$  describes a word  $w_i$  as the root of a context  $C_k(w_i)$ ,  $f_C$  on the other hand describes a word  $w_j$  as part of a context  $C_k(w_i) \ni w_j$ .

**Vertex Embeddings** Skip-gram embeddings can be naïvely extended to graphs by realizing that word2vec effectively already is a vertex embedding method for linear graphs in which  $C_k(v)$  is simply the  $k$ -neighborhood of the vertex/word  $v$ . The problem with this naïve extension is that the sizes of  $k$ -neighborhoods in arbitrary graphs can be much larger and often tend to grow exponentially with  $k$ . To deal with this computational problem the so-called DeepWalk [Per+14] and node2vec [GL16] methods perform random walks of fixed length to effectively take samples from the neighborhood of vertices. Both methods only differ in the transition matrix that is used for the random walk. Another difference of DeepWalk and node2vec compared to word2vec is the so-called *feature space symmetry* which states that the context root interpretation ( $f$ ) of a vertex should be symmetric to its context element interpretation ( $f_C$ ), i.e.  $f = f_C$ . The combination of random walk context sampling and the feature space symmetry assumption can be used to compute vertex embeddings even for very large graphs.

**Graph Embeddings** Skip-gram methods can not only be used for vertex embeddings but also to embed entire graphs. One way to do this is via the graph2vec [Nar+17] method. It is inspired by doc2vec [LM14] which in turn is based on word2vec. graph2vec gets a set of graphs  $\mathcal{G} = \{G_1, \dots, G_N\}$  as input and outputs graph embeddings  $f(G_1), \dots, f(G_N)$ . While in word2vec every word can be a context root as well as a context element, graph2vec uses the graphs  $\mathcal{G}$  as context roots. The context

$C_k(G_i)$  of a graph  $G_i$  is defined as

$$C_k(G_i) := \bigcup_{v_j \in \mathcal{V}_{G_i}} \left\{ \chi_{G_i,1}^{(l)}(v_j) \right\}_{l=0}^L \quad \text{with } k \in \mathbb{N} \text{ and } \chi_{G_i,1}^{(l)} \text{ as in definition 2.9.} \quad (2.5)$$

Intuitively this context can be understood as the set of 1-WL-distinguishable subgraphs of  $G_i$  with diameter  $\leq 2L$ . Since WL is used to identify distinct subgraphs, graph2vec can only be applied to graphs with discrete vertex labels. Using the previous definitions, the context root embedding function has the signature  $f : \mathcal{G} \rightarrow \mathbb{R}^d$  while the context element embedding function is of type  $f_C : \bigcup_{G_i \in \mathcal{G}} C_k(G_i) \rightarrow \mathbb{R}^d$ . To find those embeddings the word2vec objective from eq. (2.4) is reused. Analogous to word2vec, graph2vec therefore embeds graphs that share subgraphs close to each other, whereas graphs that do not share substructures tend to be embedded further away from each other.

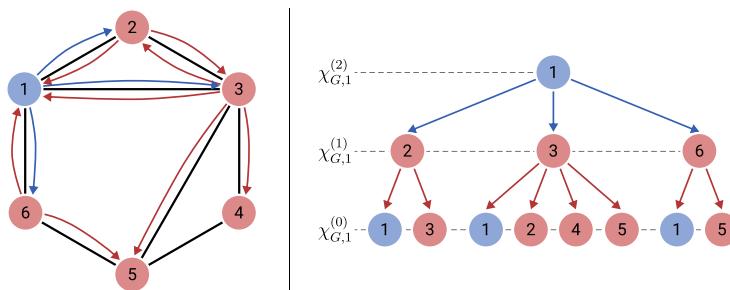
### 2.3.2 Graph Kernels

Instead of mapping a graph into an explicitly defined vector space of fixed dimension, one can also do so implicitly by employing the kernel trick. There is a large variety of so-called *graph kernels* (GKs) to do this. GKs can be used in combination with any kernelized learner, typically SVMs, to solve the GC/GR problem. While there is a large variety of different GKs [Kri+20], we will focus on those that are based on the previously described family of WL algorithms.

**WL subtree kernel** One well-known GK is based directly on the 1-WL coloring algorithm, the so-called *WL subtree kernel* [She+11]. It uses the mapping

$$\varphi_{ST}(G) := \bigoplus_{l=0}^L \left( \text{dist}_{\chi_{G,1}^{(l)}}(c) \right)_{c \in \mathcal{C}} \quad \text{with } \oplus \text{ denoting vector concatenation.} \quad (2.6)$$

$\varphi_{ST}(G)$  encodes the color counts across a fixed number  $L$  of 1-WL refinement steps. Via this mapping, the WL subtree kernel function  $k_{ST} : \mathcal{G} \times \mathcal{G} \rightarrow \mathbb{R}$  can then be simply written as the standard inner vector product  $k_{ST}(G, H) := \langle \varphi_{ST}(G), \varphi_{ST}(H) \rangle$ .



**Figure 2.10.** Illustration of the correspondence between the WL color  $\chi_{G,1}^{(l)}(v)$  and the breadth-first subtree of depth  $l$  rooted at  $v$  (for  $l = 2$  and  $v = \textcircled{1}$ ). BASED ON: [SHE+11]

Figure 2.10 illustrates how this definition relates to subtrees. Since all vertex colors  $c \in \mathcal{C}$  correspond to a subtree isomorphism class,  $k_{\text{ST}}(G, H)$  effectively computes the similarity of  $G$  and  $H$  by comparing their local subtree structures of depth at most  $L$ . Note that the underlying mapping  $\varphi_{\text{ST}}$  cannot be computed independently for each graph  $G \in \mathcal{G}_{\mathcal{D}} \subseteq \mathcal{G}$  in a given training dataset  $\mathcal{D}$  since the set of colors  $c \in \mathcal{C}$  introduced by the hashing function  $h$  is conjointly determined by all graphs  $\mathcal{G}_{\mathcal{D}}$  (see definition 2.9). Consequently the dimensionality  $d$  of  $\varphi_{\text{ST}}(G) \in \mathbb{N}^d$  varies, depending on the entire dataset  $\mathcal{D}$ .

**WL shortest path kernel** Extending the idea of the WL subtree kernel, the *WL shortest path kernel* [She+11][BK05] adds global structural information to the local subtree comparisons. The  $i$ -th component of a graph's shortest path vector  $\varphi_{\text{SP}}(G)$  corresponds to the 4-tuple  $(l_i, a_i, b_i, d_i) \in \{0, \dots, L\} \times \mathcal{C} \times \mathcal{C} \times \mathbb{N}$  and its value is described by

$$\varphi_{\text{SP}}(G)_i := \left| \left\{ (v, u) \in \mathcal{V}_G^2 \mid \chi_{G,1}^{(l_i)}(v) = a_i \wedge \chi_{G,1}^{(l_i)}(u) = b_i \wedge d_{\text{SP}}(v, u) = d_i \right\} \right|, \quad (2.7)$$

with  $d_{\text{SP}}(v, u)$  denoting the length of the shortest path from  $v$  to  $u$  in  $G$ . Analogous to  $k_{\text{ST}}$ , the shortest path kernel function is defined as  $k_{\text{SP}}(G, H) := \langle \varphi_{\text{SP}}(G), \varphi_{\text{SP}}(H) \rangle$ . Since  $\varphi_{\text{SP}}(G)$  has one component for each possible 4-tuple  $(l_i, a_i, b_i, d_i)$ , its dimensionality grows quadratically with the total number of introduced 1-WL colors  $\mathcal{C}$  and linearly with the length of the longest shortest path  $\max_{G \in \mathcal{G}_{\mathcal{D}}, (v,u) \in \mathcal{V}_G^2} d_{\text{SP}}(v, u)$ . While this approach is computationally significantly more expensive than the subtree kernel, it also has a higher discriminative power and allows the shortest path kernel to distinguish even some 1-WL indistinguishable graphs. The subtree kernel only compares graphs by the presence of local subtrees; the shortest path kernel additionally checks how well the distances between those subtrees align. Looking back at the 1-WL indistinguishable “six-cycle”/“two-triangles” example from fig. 2.5, we see that this additional information distinguishes the two graphs due to the fact that the longest shortest path in a six-cycle has length 3 while the longest shortest path in a triangle is of length 1.

**Higher dimensional WL kernels** Instead of incorporating shortest path information to increase the power of the WL kernel, one can alternatively just increase the WL-dimension  $k$ . A naïve generalization of the 1-WL subtree kernel would simply use eq. (2.6) but with the  $k$ -WL colorings  $\chi_{G,k}^{(l)}$  instead of  $\chi_{G,1}^{(l)}$ . The problem with this approach is that the runtime of WL increases exponentially with  $k$ ; even for  $k = 2$  the cost often becomes infeasibly high when working with large graphs. To tackle this problem Morris et al. [Mor+17] proposed a combination of two optimizations:

1.  **$k$ -multisets:** The first optimization is to assign colors to vertex  $k$ -multisets instead of vertex  $k$ -tuples, where a  $k$ -multiset is any multiset  $s \subseteq \mathcal{V}_G$  with  $|s| = k$ . This reduces the amount of colors that have to be refined in each

WL iteration by a factor of  $k!$ . Based on definition 2.10, the multiset color refinement step is defined as

$$\chi_{G,k}^{(l+1)}(s) := h \left( \chi_{G,k}^{(l)}(s), \{ \{ \chi_{G,k}^{(l)}(s \setminus \{v\} \cup \{u\}) \mid v \in s \} \mid u \in \mathcal{V}_G \} \right). \quad (2.8)$$

Even though this simplification generally reduces the discriminative power of the kernel, for  $k = 2$  specifically no information and therefore no power is lost by replacing the tuples  $(v, u)$  and  $(u, v)$  with  $\{v, u\}$  since the undirectedness of graphs implies that  $\chi_{G,2}^{(l)}(v, u) = \chi_{G,2}^{(l)}(u, v)$  even when doing tuple-based color refinement.

2. **Neighborhood localization:** The second optimization uses the fact that most real-world graphs tend to be sparse [Chu10], i.e.  $|\mathcal{E}_G| = \mathcal{O}(|\mathcal{V}_G|)$ . The multiset-based  $k$ -WL algorithm refines the color of a  $k$ -multiset  $s$  by hashing the colors of its neighbors as defined in eq. (2.8) where there is one neighbor per vertex  $u \in \mathcal{V}_G$ . By the sparsity assumption, most of those vertices  $u$  are however not connected with any of the vertices in  $s$ , i.e.  $u \notin \Gamma_G(s)$  with  $\Gamma_G(s) := \bigcup_{v \in s} \Gamma_G(v)$ . The refinement runtime can therefore often be reduced significantly by only considering “local” neighbors  $u \in \Gamma_G(s)$  instead of the “global” neighborhood  $u \in \mathcal{V}_G$  in eq. (2.8).

We call the kernel that only uses the first optimization  $k$ -GWL ( $k$ -dim. global WL) and the kernel that uses both optimizations  $k$ -LWL ( $k$ -dim. local WL). Morris et al. [Mor+17] have empirically shown that focusing on local graph structures via LWL often actually performs better than the computationally more expensive GWL kernel.

### 2.3.3 Graph Neural Networks

The last family of GC/GR approaches we will look at is that of graph neural networks. The idea to feed a graph into a *neural network* (NN) was first described by Gori et al. [Gor+05]. Since then many variants and extensions of that idea have been proposed [Wu+19]. We will focus specifically on the so-called *graph convolutional neural networks* (GCNNs) which can be divided into two variants: Spectral and spatial GCNNs.

#### Spectral GCNNs

The class of spectral GCNNs is motivated by spectral graph theory (see section 2.2.3). The spectral perspective on GC/GR expects a graph  $G$  with real vertex feature vectors  $x_G[v_i] \in \mathbb{R}^f$  as its input. We call  $X_G := (x_G[v_1], \dots, x_G[v_n])^\top \in \mathbb{R}^{n \times f}$  the vertex feature matrix of  $G$ . Note that each of the  $f$  columns of  $X_G$  can be interpreted as

a separate graph signal function  $x_{G,j} : \mathcal{V}_G \rightarrow \mathbb{R}$ . The core idea of spectral GCNNs is to learn a graph convolution kernel  $g$ , similar to the grid kernels in conventional convolutional neural networks [LB98]. The problem with this idea is that the convolution operation  $g * x$  requires some notion of distance in order to move the kernel  $g$  over the function  $x$ . Graphs do not generally satisfy this requirement, i.e. the notion of a vertex distance  $|v_i - v_j|$  is not clearly defined. Via the convolution theorem  $\widehat{g * x} = \hat{g} \cdot \hat{x}$  we can however still define a graph convolution operator that works directly in the spectral domain:

$$g * x_{G,j} := U\hat{g}(\Lambda)U^\top x_{G,j} \text{ with the eigendecomposition } L_G = U\Lambda U^\top \quad (2.9)$$

To see the connection to the convolutional theorem, note that the matrix of eigenvector rows  $U^\top$  describes the graph Fourier transform and conversely that  $U$  describes the inverse graph Fourier transform. Bruna et al. [Bru+13] proposed a NN architecture that learns such a spectral filter  $\hat{g}$ .

## Spatial GCNNs

### Graph Pooling



# Learning to Aggregate on Graphs

3.1 Formalization of LTA Characteristics

3.2 An LTA Interpretation of GC/GR Methods

3.3 LTA on Dynamically Decomposed Graphs



# 4

## Evaluation



# Conclusion

## 5.1 Review

## 5.2 Future Directions







## Appendix

A



# Bibliography

- [AB73] George W. Adamson and Judith A. Bush. „A method for the automatic classification of chemical structures“. In: *Information Storage and Retrieval* 9.10 (1973), pp. 561–568 (cit. on pp. 12, 16).
- [Alz+10] Alfredo Alzaga, Rodrigo Iglesias, and Ricardo Pignol. „Spectra of symmetric powers of graphs and the Weisfeiler–Lehman refinements“. In: *Journal of Combinatorial Theory, Series B* 100.6 (2010), pp. 671–682. arXiv: 0801 . 2322v1 [math.SP] (cit. on p. 14).
- [Arv+19] Vikraman Arvind, Frank Fuhlbrück, Johannes Köbler, and Oleg Verbitsky. „On Weisfeiler-Leman Invariance: Subgraph Counts and Related Graph Properties“. In: *Fundamentals of Computation Theory*. Springer International Publishing, 2019, pp. 111–125. arXiv: 1811.04801v3 [cs.DM] (cit. on p. 12).
- [Bab+80] László Babai, Paul Erdős, and Stanley M. Selkow. „Random graph isomorphism“. In: *SIAM Journal on computing* 9.3 (1980), pp. 628–635 (cit. on pp. 7, 11).
- [Bab15] László Babai. *Graph Isomorphism in Quasipolynomial Time*. Dec. 11, 2015. arXiv: 1512.03547v2 [cs.DS] (cit. on p. 7).
- [Ban+18b] Priyanka Banerjee, Andreas O. Eckert, Anna K. Schrey, and Robert Preissner. „ProTox-II: a webserver for the prediction of toxicity of chemicals“. In: *Nucleic Acids Research* 46.W1 (2018), W257–W263 (cit. on p. 16).
- [BK05] K.M. Borgwardt and H. Kriegel. „Shortest-Path Kernels on Graphs“. In: *Fifth IEEE International Conference on Data Mining (ICDM'05)*. IEEE, 2005 (cit. on p. 19).
- [Bru+13] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. *Spectral Networks and Locally Connected Networks on Graphs*. Dec. 21, 2013. arXiv: 1312.6203v3 [cs.LG] (cit. on p. 21).
- [Cai+92] Jin-Yi Cai, Martin Fürer, and Neil Immerman. „An optimal lower bound on the number of variables for graph identification“. In: *Combinatorica* 12.4 (1992), pp. 389–410 (cit. on p. 7).
- [Chu10] Fan Chung. „Graph theory in the information age“. In: *Notices of the American Mathematical Society* 57 (June 2010) (cit. on p. 20).
- [Das04] K.Ch. Das. „The Laplacian spectrum of a graph“. In: *Computers & Mathematics with Applications* 48.5-6 (2004), pp. 715–724 (cit. on p. 14).
- [Drw+14] Małgorzata N. Drwal, Priyanka Banerjee, Mathias Dunkel, Martin R. Wettig, and Robert Preissner. „ProTox: a web server for the in silico prediction of rodent oral toxicity“. In: *Nucleic Acids Research* 42.W1 (2014), W53–W58 (cit. on p. 16).

- [Fü17] Martin Fürer. „On the Combinatorial Power of the Weisfeiler-Lehman Algorithm“. In: *Lecture Notes in Computer Science*. Springer International Publishing, 2017, pp. 260–271. arXiv: 1704.01023v1 [cs.DS] (cit. on p. 12).
- [GL16] Aditya Grover and Jure Leskovec. *node2vec: Scalable Feature Learning for Networks*. July 3, 2016. arXiv: 1607.00653v1 [cs.SI] (cit. on p. 17).
- [Gor+05] M. Gori, G. Monfardini, and F. Scarselli. „A new model for learning in graph domains“. In: *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005*. IEEE, 2005 (cit. on p. 20).
- [Gu+15] Jiao Gu, Bobo Hua, and Shiping Liu. „Spectral distances on graphs“. In: *Discrete Applied Mathematics* 190-191 (2015), pp. 56–74. arXiv: 1402.6041v2 [math.SP] (cit. on p. 15).
- [IL90] Neil Immerman and Eric Lander. „Describing graphs: A first-order approach to graph canonization“. In: *Complexity theory retrospective*. Springer, 1990, pp. 59–81 (cit. on pp. 11, 12).
- [Kek66] Aug. Kekulé. „Untersuchungen über aromatische Verbindungen. I. Ueber die Constitution der aromatischen Verbindungen.“ In: *Annalen der Chemie und Pharmacie* 137.2 (1866), pp. 129–196 (cit. on p. 12).
- [Kie+17] Sandra Kiefer, Ilia Ponomarenko, and Pascal Schweitzer. „The Weisfeiler-Leman dimension of planar graphs is at most 3“. In: *2017 32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. IEEE, 2017. arXiv: 1708.07354v1 [cs.DM] (cit. on p. 12).
- [Kri+20] Nils M. Kriege, Fredrik D. Johansson, and Christopher Morris. „A survey on graph kernels“. In: *Applied Network Science* 5.1 (2020). arXiv: 1903.11835v2 [cs.LG] (cit. on p. 18).
- [LB98] Yann LeCun and Yoshua Bengio. „Convolutional Networks for Images, Speech, and Time Series“. In: *The Handbook of Brain Theory and Neural Networks*. Cambridge, MA, USA: MIT Press, 1998, 255?258 (cit. on p. 21).
- [LM14] Quoc Le and Tomas Mikolov. „Distributed Representations of Sentences and Documents“. In: *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*. ICML'14. Beijing, China: JMLR.org, 2014, pp. II–1188–II–1196. arXiv: 1405.4053v2 [cs.CL] (cit. on p. 17).
- [Lue+18] Thomas Luechtefeld, Dan Marsh, Craig Rowlands, and Thomas Hartung. „Machine Learning of Toxicological Big Data Enables Read-Across Structure Activity Relationships (RASAR) Outperforming Animal Test Reproducibility“. In: *Toxicological Sciences* 165.1 (2018), pp. 198–212 (cit. on p. 16).
- [MH16] Vitalik Melnikov and Eyke Hüllermeier. „Learning to Aggregate Using Uninorms“. In: *Machine Learning and Knowledge Discovery in Databases*. Springer International Publishing, 2016, pp. 756–771 (cit. on pp. 3, 4).
- [MH19] Vitalik Melnikov and Eyke Hüllermeier. „Learning to Aggregate: Tackling the Aggregation/Disaggregation Problem for OWA“. In: *Proceedings of The Eleventh Asian Conference on Machine Learning*. Ed. by Wee Sun Lee and Taiji Suzuki. Vol. 101. Proceedings of Machine Learning Research. Nagoya, Japan: PMLR, 2019, pp. 1110–1125 (cit. on p. 5).

- [Mik+13] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. „Distributed Representations of Words and Phrases and Their Compositionality“. In: *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*. NIPS’13. Lake Tahoe, Nevada: Curran Associates Inc., 2013, pp. 3111–3119. arXiv: 1310.4546v1 [cs.CL] (cit. on p. 16).
- [Mil02] R. Milo. „Network Motifs: Simple Building Blocks of Complex Networks“. In: *Science* 298.5594 (2002), pp. 824–827 (cit. on p. 12).
- [Mor+17] Christopher Morris, Kristian Kersting, and Petra Mutzel. „Glocalized Weisfeiler-Lehman Graph Kernels: Global-Local Feature Maps of Graphs“. In: *2017 IEEE International Conference on Data Mining (ICDM)*. IEEE, 2017. arXiv: 1703.02379v3 [cs.LG] (cit. on pp. 19, 20).
- [MP13] Brendan D. McKay and Adolfo Piperno. *Practical graph isomorphism, II*. Jan. 8, 2013. arXiv: 1301.1493v1 [cs.DM] (cit. on p. 7).
- [Nar+17] Annamalai Narayanan, Mahinthan Chandramohan, Rajasekar Venkatesan, et al. *graph2vec: Learning Distributed Representations of Graphs*. July 17, 2017. arXiv: 1707.05005v1 [cs.AI] (cit. on p. 17).
- [New03] M. E. J. Newman. „The Structure and Function of Complex Networks“. In: *SIAM Review* 45.2 (2003), pp. 167–256 (cit. on p. 12).
- [Per+14] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. „DeepWalk: online learning of social representations“. In: *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD ’14*. ACM Press, 2014. arXiv: 1403.6652v2 [cs.SI] (cit. on p. 17).
- [She+11] Nino Shervashidze, Pascal Schweitzer, Erik Jan Van Leeuwen, Kurt Mehlhorn, and Karsten M. Borgwardt. *Weisfeiler-Lehman Graph Kernels*. 2011 (cit. on pp. 18, 19).
- [Shu+13] D. I. Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst. „The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains“. In: *IEEE Signal Processing Magazine* 30.3 (2013), pp. 83–98 (cit. on pp. 13–15).
- [Wel+07] Howard T. Welser, Eric Gleave, Danyel Fisher, and Marc Smith. „Visualizing the signatures of social roles in online discussion groups“. In: *Journal of Social Structure* (2007) (cit. on p. 12).
- [WL68] Boris Weisfeiler and Andrei A. Lehman. „A reduction of a graph to a canonical form and an algebra arising during this reduction“. In: *Nauchno-Technicheskaya Informatsia* 2.9 (1968), pp. 12–16 (cit. on p. 7).
- [Wu+19] Zonghan Wu, Shirui Pan, Fengwen Chen, et al. *A Comprehensive Survey on Graph Neural Networks*. Jan. 3, 2019. arXiv: 1901.00596v4 [cs.LG] (cit. on p. 20).
- [WW86] Peter Willett and Vivienne Winterman. „A Comparison of Some Measures for the Determination of Inter-Molecular Structural Similarity Measures of Inter-Molecular Structural Similarity“. In: *Quantitative Structure-Activity Relationships* 5.1 (1986), pp. 18–25 (cit. on p. 16).

## Websites

- [Ban+18a] Priyanka Banerjee, Robert Preissner, Andreas Eckert, and Anna K. Schrey. *ProTox-II - Prediction Of Toxicity Of Chemicals*. Charité – Universitätsmedizin Berlin. 2018. URL: [http://tox.charite.de/protox\\_II/](http://tox.charite.de/protox_II/) (visited on Nov. 18, 2019) (cit. on p. 16).
- [MP] Brendan McKay and Adolfo Piperno. *nauty and Traces*. URL: <http://pallini.di.uniroma1.it/index.html> (visited on Feb. 21, 2020) (cit. on p. 7).
- [Tes] *Toxicity Estimation Software Tool (TEST)*. EPA. URL: <https://www.epa.gov/chemical-research/toxicity-estimation-software-tool-test> (visited on Nov. 18, 2019) (cit. on p. 16).
- [Tox] *ToxTrack - Cheminformatics Modeling*. ToxTrack Inc. URL: <https://toxtrack.com/> (visited on Nov. 18, 2019) (cit. on p. 16).

# List of Figures

2.1	Overview of the structure of LTA for multiset compositions. . . . .	4
2.2	The Łukasiewicz norms and the corresponding uninorm for $\lambda = 0.5$ . . . . .	5
2.3	Illustration of how a BUM function is described as a linear spline and its relation to the OWA weights. . . . .	6
2.4	Example 1-WL color refinement steps. . . . .	9
2.5	Two simple non-isomorphic graphs that are indistinguishable by 1-WL. . . . .	9
2.6	WL neighborhoods for different values of $k$ . . . . .	10
2.7	Two non-isomorphic graphs with $G \simeq_1 H$ and $G \not\simeq_2 H$ . . . . .	11
2.8	Comparison between the basis vectors of the real domain Fourier transform and the graph Fourier transform. . . . .	14
2.9	Comparison of the unnormalized $L_G$ spectrum and the normalized $L_G^{\text{sym}}$ spectrum. . . . .	15
2.10	Illustration of the correspondence between the WL color $\chi_{G,1}^{(l)}(v)$ and the breadth-first subtree of depth $l$ rooted at $v$ . . . . .	18







# Erklärung zur Masterarbeit

Ich, Clemens Damke (Matrikel-Nr. 7011488), versichere, dass ich die Masterarbeit mit dem Thema *Learning to Aggregate on Structured Data* selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Die Stellen der Arbeit, die ich anderen Werken dem Wortlaut oder dem Sinn nach entnommen habe, wurden in jedem Fall unter Angabe der Quellen der Entlehnung kenntlich gemacht. Das Gleiche gilt auch für Tabellen, Skizzen, Zeichnungen, bildliche Darstellungen usw. Die Masterarbeit habe ich nicht, auch nicht auszugsweise, für eine andere abgeschlossene Prüfung angefertigt. Auf § 63 Abs. 5 HZG wird hingewiesen.

Paderborn, 4. März 2020

---

Clemens Damke