On the Extension of the Weisfeiler-Lehman Hierarchy by WL Tests for Arbitrary Graphs

Silvia Beddar-Wiesing[1,3], Guiseppe Alessio D'Inverno[2,3], Caterina Graziani[2,3], Veronica Lachi[2,3], Alice Moallemy-Oureh[1,3], and Franco Scarselli[2]

[1] University of Kassel, Germany
{s.beddarwiesing,amoallemy}@uni-kassel.de
http://www.gain-group.de
[2] University of Siena, Italy
{giuseppealessio.d,caterina.graziani,veronica.lachi}@student.unisi.it,
franco@diism.unisi.it
[3] Equal Contribution

Abstract. Graph isomorphism (GI) has occupied both theoreticians and applied scientists since the early 1950s. Over the years, several approaches and algorithms with which an isomorphism between two graphs can be tested have been developed. A general approach is the Weisfeiler–Lehman (WL) test, which is based on a coloring algorithm and provides a necessary criterion for graph isomorphism. However, the WL test is restricted to examining graphs with only node attributes. Therefore, this paper presents two extensions of the WL algorithm to allow for testing on arbitrary graphs. One considers edge attributes, and the other tests the isomorphism on dynamic graphs. Additionally, we extend the WL-hierarchy by the attributed and dynamic WL tests and show that it is a partial order, which induces a lattice. In the future, this may allow for practical implications coming from lattice theory.

Keywords: Weisfeiler–Lehman test · Weisfeiler–Lehman Hierarchy · Attributed Graphs · Dynamic Graphs · Graph Isomorphism

1 Introduction

The graph isomorphism (GI) problem lays in the class NP and is thus of particular theoretical interest. Furthermore, the identification of two isomorphic graphs appears to be highly relevant in a broad range of applications, from the distinction of molecular graphs in chemistry to computation graphs in computer science. The history of approaches to solve the problem approximately reaches back to the 1970s, and the so-called Weisfeiler–Lehman (WL) method was considered as one possible candidate to solve the GI problem in polynomial time. It iteratively generates a graph coloring based on the neighborhood structure of the nodes and compares the used color sets for two graphs after termination. Recently, the interest in WL tests has rapidly grown in machine learning since they can also be used to describe the expressive power of Graph Neural Networks (GNNs).

It has been proven that common GNNs can recognize the same class of graphs as 1-WL [1] [2]. However, there exist classes of graphs that are not distinguishable by the 1-WL test while being not isomorphic; therefore, the 1-WL equivalence is a necessary condition for two graphs to be isomorphic. In [3], a great number of non-isomorphic graph pairs are presented that are not distinguishable by the WL test. This problem motivated the development of more powerful k-WL tests [4] for $k \in \mathbb{N}$ following the WL-hierarchy :

$$1 - \mathrm{WL} \subseteq 2 - \mathrm{WL} \subset \ldots \subset k - \mathrm{WL} \subset \ldots$$

In particular, this result shows that if there are two graphs that are not distinguishable by a $k-$WL test, there must be an $l-$WL test with $l > k$ that can distinguish them.

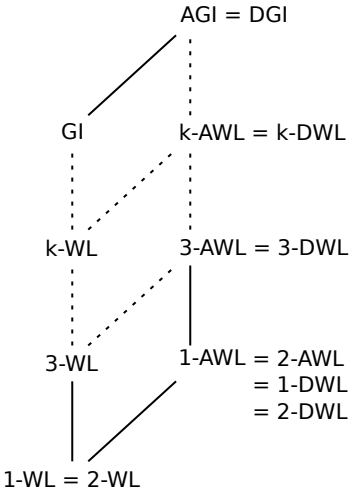

Fig. 1. The WL / AWL complete lattice. Its minimum is the 1-WL test, equivalent to the 2-WL test, and its maximum is the Attributed Graph Isomorphism (AGI) and Dynamic Graph Isomorphism (DGI) test.

In practice, however, the $k-$WL tests by definition are just applicable to simple node-attributed graphs. Since in many theoretical as well as technical applications the used graphs are more complex, as, e.g., heterogeneous graphs, multigraphs, hypergraphs or even dynamic graphs, the usual $k-$WL test cannot be used to distinguish them. For this reason, in this paper we introduce the attributed and dynamic extensions of the $k-$WL tests, namely $k-$AWL and $k-$DWL in Sec. 3.

Attributed graphs with edge and node labels and dynamic graphs are particularly interesting for practical and theoretical reasons. Indeed, attributed graphs with edge and node labels and dynamic graphs are interesting since it has been proved that they can act as a generic form for other graph types, as most common other types of graphs can be transformed into attributed graphs ([5]). Dynamic graphs are interesting because their theoretical analysis differs from static graphs.

It is worth mentioning that the k-AWL and k-DWL, which we will discuss, have been designed as the counterparts of GNNs. Even if the proof cannot be included for space restrictions, there exist generic GNN models that operate on attributed and dynamic graphs and whose expressive power is described by the k-AWL and k-DWL models presented in the paper, respectively.

Extending the original WL-hierarchy in Sec. 4 changes its total order to a partial order, where the $3-$WL and the attributed $1-$WL are not comparable (cf. Thm. 4.05). Additionally, in Thm. 4.06, we prove that $1-$DWL$= 1-$AWL. A very promising additional result is that the new extended version of the WL-hierachy induces a complete and distributive lattice with the partial order defined

in Cor. 4.07 by: $A \leq B$ iff the partition $\mathcal{P}_A$ induced by the test A is equal or coarser than the partition $\mathcal{P}_B$ induced by the test B. The minimum element of this lattice is the 1-WL test, equivalent to the 2-WL test, while the maximum element is the Attributed Graph Isomorphism (AGI) test and Dynamic Graph Isomorphism (DGI) test defined in Sec.2.

In the future, the goal is to find useful implications from lattice theory for the GI problem with this result. Some possible research questions regarding that would be, e.g.,

1. How big is the **difference** $|\mathcal{P}_B| - |\mathcal{P}_A|$ of the partitions $\mathcal{P}_A, \mathcal{P}_B$ if $A \leq B$?
2. Which approaches from **lattice theory** can be applied to the WL hierarchy to infer beneficial consequences from it?
3. Is it possible for two graphs to find the **minimal WL-test** capable of distinguishing the graphs?
4. What are **minimal requirements** to a subset of WL-tests such that it remains a **lattice**, or that we obtain a **semilattice**?

2 Preliminaries

The following definitions and preliminaries are listed to ensure that this paper is self-contained and enables effortless understanding.

2.1 Graphs

In this paper, only finite graphs are under consideration.

Definition 2.1.01 (Different Graph Types)**.** The elementary graphs are defined in [5] as follows.

1. A **directed graph (digraph)** is a tuple $G = (\mathcal{V}, \mathcal{E})$ containing a set of nodes $\mathcal{V} \subsetneq \mathbb{N}$ and a set of directed edges given as tuples $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$.
2. A **(generalized) directed hypergraph** is a tuple $G = (\mathcal{V}, \mathcal{E})$ with nodes $\mathcal{V} \subsetneq \mathbb{N}$ and hyperedges $\mathcal{E} \subseteq \{(\mathsf{x}, f_i)_i \mid \mathsf{x} \subseteq \mathcal{V},\ f_i : \mathsf{x} \to \mathbb{N}_0\}$ that include a numbering map f_i for the i-th edge $(\mathsf{x}, f)_i$ which indicates the order of the nodes in the (generalized) directed hyperedge.

An elementary graph $G = (\mathcal{V}, \mathcal{E})$ is called

1. **undirected** if the directions of the edges are irrelevant, i.e.,
   - for directed graphs: if $(u, v) \in \mathcal{E}$ whenever $(v, u) \in \mathcal{E}$ for $u, v \in \mathcal{V}$.
   - for directed hypergraphs: if $f_i : \mathsf{x} \to 0$ for all $(\mathsf{x}, f_i)_i \in \mathcal{E}$[4].
2. **multigraph** if the node or edge sets are multisets.
3. **heterogeneous** if the nodes or edges can have different types. I.e., the node set is determined by $\mathcal{V} \subseteq \mathbb{N} \times \mathcal{S}$ with a node type set $\mathcal{S}$ and thus, a node $(v, s) \in \mathcal{V}$ is given by the node v itself and its type s. The edges can be extended by a set $\mathcal{R}$ that describes their types, to $(e, r)\ \forall\, e \in \mathcal{E}$ of edge type $r \in \mathcal{R}$. Otherwise, the graph is called **homogeneous**.

[4] $f_i(\mathsf{x}) = 0$ encodes that x is an undirected hyperedge

4. **attributed** if the nodes $\mathcal{V}$ or edges $\mathcal{E}$ are equipped with node or edge attributes[5], formally given by a node and edge attribute function, respectively, i.e. $\alpha : \mathcal{V} \to \mathcal{A}$ and $\omega : \mathcal{E} \to \mathcal{W}$, where $\mathcal{A}$ and $\mathcal{W}$ are arbitrary attribute sets.

Definition 2.1.02 (Dynamic Graph)**.** Let $T = \{t_i\}_{i \in I} \subsetneq \mathbb{R}$ with $I = [0, \dots, l] \subsetneq \mathbb{N}$ be a set of timestamps. Each timestamp t_i can be bijectively identified by its index i. W.l.o.g., from now on we refer to the set of timestamps as I. Then a **(discrete) dynamic graph** can be considered as a vector of static graph snapshots, i.e., $G = (G_t)_{t \in I}$, where $G_t = (\mathcal{V}_t, \mathcal{E}_t, \alpha_t, \omega_t)\ \forall t \in I$.

Furthermore,

$$\alpha_v(t) := \begin{cases} \alpha(v,t), & v \in \mathcal{V}_t \\ \perp, & \text{otherwise} \end{cases} \quad \text{and} \quad \omega_{\{u,v\}}(t) := \begin{cases} \omega(\{u,v\},t), & \{u,v\} \in \mathcal{E}_t \\ \perp, & \text{otherwise} \end{cases},$$

where $\alpha : \mathcal{V} \times I \to \mathcal{A} \cup \perp$ and $\omega : \mathcal{E} \times I \to \mathcal{B} \cup \perp$ define the vector of **dynamic node/edge attributes**. In particular, $\alpha_v(t) = \perp$ means that at the timestamp t the node v does not exist. Analogoulsy, $\omega_{\{u,v\}}(t) = \perp$ encodes the absence of the edge at that time. Moreover, let

$$\Omega_{ne_v}(t) = \Big(\omega_{\{v,x_1\}}(t), \dots, \omega_{\{v,x_{|ne_v(t)|}\}}(t) \Big)_{t \in I}$$

be the sequence of **dynamic edge attributes of the neighborhood** corresponding node at each timestamp.

In [5] it has been shown that every graph type is transformable into a **static attributed undirected homogeneous graph (SAUHG)**. Thus, it in the following, only the SAUHG is considered when talking about attributed graphs. W.l.o.g., the attribute spaces are set to the same $\mathbb{R}^k$ vector space. Furthermore, there exists a bijective transformation from any arbitrary graph type defined in Def. 2.1.01 to the SAUHG. They result from concatenating transformations for single graph properties from [5] and are sketched in the following.

Remark 2.1.03 (Graph Type Transformations)**.** Given a graph of arbitrary type, the concatenation of suitable transformations from the following list lead to the SAUHG type.

1. **Hypergraph → Simple Graph:** Transform undirected hyperedges to fully connected subgraphs and directed edges to chained directed bipartite subgraphs given the hyperedge direction.
2. **Multigraph → Simple Graph:** Encode multiple nodes or edges in an additional counter each and, i.a., concatenate the corresponding attributes in a vector.
3. **Dynamic → Static:** Encode the dynamical behaviour of the graph into a timeseries of node and edge attributes accordingly.
4. **Unattributed → Attributed:** Add empty attributes to each node or edge.
5. **Directed → Undirected:** Replace all directed edges by undirected ones and add the direction as encoded additional information into the edge attributes.

[5] In some literature attributes are also called features.

6. **Heterogenous → Homogeneous:** Encode the different node or edge types in an additional dimension of the node or edge attribute, respectively.

The graph isomorphism (GI) in comparison distinguishes graphs by their structure [6]. To substantiate the usage of the WL test on attributed and dynamic graphs, the GI is extended, respectively, as follows.

Definition 2.1.04 (Graph Isomorphism)**.** Let G_1 and G_2 be two static graphs, then $G_1 = (\mathcal{V}_1, \mathcal{E}_1)$ and $G_2 = (\mathcal{V}_2, \mathcal{E}_2)$ are isomorphic to each other $G_1 \approx G_2$, iff there exists a bijective function $\phi : \mathcal{V}_1 \to \mathcal{V}_2$, with
1. $v_1 \in \mathcal{V}_1 \iff \phi(v_1) \in \mathcal{V}_2 \quad \forall\, v_1 \in \mathcal{V}_1$,
2. $\{v_1, v_2\} \in \mathcal{E}_1 \iff \{\phi(v_1), \phi(v_2)\} \in \mathcal{E}_2 \quad \forall\, \{v_1, v_2\} \in \mathcal{E}_1$

In case both graphs are **attributed**, i.e., $G_1 = (\mathcal{V}_1, \mathcal{E}_1, \alpha_1, \omega_1)$ and $G_2 = (\mathcal{V}_2, \mathcal{E}_2, \alpha_2, \omega_2)$, then $G_1 \approx G_2$ iff additionally there exist bijective functions $\varphi_\alpha : A_1 \to A_2$ and $\varphi_\omega : B_1 \to B_2$ with $A_i := \mathrm{im}(\alpha_i)$, $B_i := \mathrm{im}(\omega_i)$ for $i = 1, 2$, and
1. $\varphi_\alpha(v_1) = \alpha_2(\phi(v_1)) \quad \forall\, v_1 \in \mathcal{V}_1$,
2. $\varphi_\omega(\{u_1, v_1\}) = \omega_2(\phi(u_1), \phi(v_1)) \quad \forall\, \{u_1, v_1\} \in \mathcal{E}_1$.

Furthermore, two **dynamic** graphs are isomorphic if all the static graph snapshots in each timestemp are isomorphic.

2.2 Weisfeiler-Lehman Algorithms

To analyze if two given graphs are isomorphic, in general, is a non-trivial problem that lays in the class NP [6]. However, the **Weisfeiler-Lehman (WL)** test can be used to at least distinguish non-isomorphic graphs. If the WL test outputs two graphs as isomorphic the isomorphism is likely but not given for sure.

The 1-WL test is based on a graph coloring algorithm that is applied in parallel on two input graphs. They are detected as non-isomorphic if the number of colors used in both colorings after termination do not coincide. When the numbers are equal, the graphs are only possibly isomorphic.

Definition 2.2.01 (1-WL test)**.** Let HASH be a bijective function that encodes every possible node feature with a color and $G = (\mathcal{V}, \mathcal{E}, \alpha)$. Then, the 1-WL test is recursively defined on the graph nodes by

– At iteration $i = 0$ the color is set to the hashed node feature:

$$o_v^{(0)} = \mathrm{HASH}(\alpha_v)$$

– For any other iteration $i > 0$ it holds

$$o_v^{(i)} = \mathrm{HASH}\Big(\big(o_v^{(i-1)}, o_{ne_v}^{(i-1)}\big)\Big).$$

The algorithm terminates if the number of colors between two iterations does not change, i.e., if

$$|\{o_v^i \mid v \in \mathcal{V}\}| = |\{o_v^{i+1} \mid v \in \mathcal{V}\}|.$$

So the termination in Def. 2.2.01 corresponds to a stability of the colors of each node after an iteration. Therefore, it is reasonable to define two graphs as equivalent if their node colors result in the same stable position which leads to the following expression of WL-equivalence.

Definition 2.2.02 (WL-equivalence)**.** Let $G_1 = (\mathcal{V}_1, \mathcal{E}_1, \alpha_1)$ and $G_2 = (\mathcal{V}_2, \mathcal{E}_2, \alpha_2)$ be two graphs. Then G_1 and G_2 are WL-equivalent, noted by $G_1 \sim_{WL} G_2$, iff for all nodes $v_1 \in \mathcal{V}_1$ there exists a corresponding node $v_2 \in \mathcal{V}_2$ with $o_{v_1} = o_{v_2}$.

Definition 2.2.03 (k-WL Test)**.** Let

$$N_{s,j} = \{(s_1, ..., s_{j-1}, r, s_{j+1}, ..., s_k) \,|r \in \mathcal{V}\}$$

be the j-th neighborhood of a k-tuple $s = (s_1, ..., s_k) \in \mathcal{V}^k$. It is obtained by replacing the j-th component of s by every node from $\mathcal{V}$. In iteration 0, the algorithm initializes each k-tuple with its *atomic type*, i.e., two k-tuples $s, s' \in \mathcal{V}^k$ get the same color if the assignment $s_i \mapsto s'_i$ induces an isomorphism between the subgraphs induced from the nodes from s and s', respectively.

For iteration $i > 0$, we define

$$O^{(i)}_{s,j} = \text{HASH}\left((o^{(i-1)}_{s'} | s' \in N_j(s))\right),$$

and set

$$o^{(i)}_s = \text{HASH}\left(o^{(i-1)}_s, (O^{(i)}_{s,1}, ..., O^{(i)}_{s,k})\right)$$

Hence, two tuples s and s' with $o^{(i-1)}_s = o^{(i-1)}_{s'}$ get different colors in iteration i if there exists $j \leq k$ such that the number of $j-$neighbors of s and s', respectively, colored with a certain color is different. The algorithm then proceeds analogously to the 1-WL. By increasing k, the algorithm gets more powerful in terms of distinguishing non-isomorphic graphs, i.e., for each $k \geq 2$, there are non-isomorphic graphs which can be distinguished by the $(k+1)$-WL but not by the k-WL [7].

2.3 Lattice

The mathematical structure of a lattice can be fully characterized using algebraic terms as well by order properties. The partial order coming from the WL-hierarchy extension, including the $k-$dimensional attributed and dynamic WL test results in a lattice of all the considered WL tests (cf. Sec. 4).

Definition 2.3.01 (Semilattice)**.** Let $\mathbf{L} =< L, \cdot >$ be a commutative semi-group. If $\forall a \in L$ holds $a \cdot a = a$ (i.e. every element in A is idempotent) then $\mathbf{L}$ is a **semilattice**.

Definition 2.3.02 (Lattice)**.** Let $\mathbf{L} =< L, \vee, \wedge >$ be an algebra such that $< L, \vee >$ and $< L, \wedge >$ are semilattices.
$\mathbf{L}$ is a **lattice** if the *absorption laws* hold, i.e. if

$$a \vee (a \wedge b) = a \qquad \textit{and} \qquad a \wedge (a \vee b) = a \qquad \forall a, b \in L.$$

The lattice operations $\vee$ and $\wedge$ are called *join* and *meet*. It is particularly important to underline the relation between lattices and partially ordered sets (posets). Let L be a set, $\leq$ a partial order on L (i.e., $\leq$ is reflexive, anti-symmetric and transitive) and $X \subseteq L$. Then, an element $a \in L$ is an upper bound of X in L, such that $x \leq a \ \forall x \in X$ and, if $x \leq b \ \forall x \in X$ then $a \leq b$, if it exists. In a similar fashion the lower bound can be defined. Given $X \subseteq L$, we will indicate with $\vee X$ and $\wedge$ the upper bound and the lower bound respectively (if they exist).

Theorem 2.3.03. Let L be a poset such that any $X \subseteq L$, $|X| < \infty$, admits upper and lower bounds. Then, $< L, \vee, \wedge >$ is a lattice where, $\forall a, b \in L$

$$a \vee b = \bigvee\{a, b\} \qquad and \qquad a \wedge b = \bigwedge\{a, b\}.$$

Vice versa, if $< L, \vee, \wedge >$ is a lattice, then the relation $a \leq b$ iff $a \vee b = b$ iff $a \wedge b = a$ is a partial order.
Every finite subset $X \subseteq L$, $X = \{a_1, \ldots, a_n\}$ admits upper and lower bounds defined by

$$\bigvee X = a_1 \vee \cdots \vee a_n \qquad \bigwedge X = a_1 \wedge \cdots \wedge a_n.$$

Definition 2.3.04. Let **L** be a lattice. Then $a, b \in L$ are **comparable** if $a \leq b$ or $b \leq a$. Otherwise, they are **incomparable**.

If the elements of L are pairwise comparable, the lattice is fully ordered and it is called a **chain**. We say that a **covers** b if $\{c : a \leq c \leq b\} = \{a, b\}$.

Definition 2.3.05 (Complete Lattice)**.** A lattice **L** is **complete** if each subset of L admits upper and lower bounds.

Definition 2.3.06 (Distributive Lattice)**.** A lattice **L** is **distributive** if $\forall a, b, c \in \mathbf{L}$

$$a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c).$$

There are two 'minimal' counterexamples to distributivity, namely the non-distributive lattices M_3 and N_5, depicted in Figure 2.

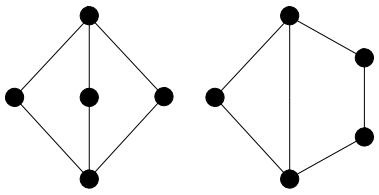

Fig. 2. The lattices M_3 (left) and $N_5(right)$.

The following theorem [8] characterizes distributive lattices in terms of 'forbidden substructures'.

Theorem 2.3.07. Let **L** be a lattice. Then **L** is distributive iff, **L** does not contain an unbounded sublattice which is isomorphic to M_3 or N_5.

3 WL Test Extensions

To exclude the isomorphism for arbitrary graph types, the WL test is preseneted here. In the attributed extension, the coloring update is dependent on the edge attributes and the dynamic version is aligned to the sequential definition of dynamic graphs.

3.1 WL Tests for Attributed Graphs

To cover all graph types introduced in Def. 2.1.01 a more powerful WL test is needed, that can handle attributed edges.

Definition 3.1.01 (Attributed 1-WL test)**.** Let HASH be a bijective function that codes every possible node feature with a color and $G = (\mathcal{V}, \mathcal{E}, \alpha, \omega)$. The **attributed 1-WL (1-AWL) test** is defined recursively through the following.

- At iteration $i = 0$ the color is set to the hashed node attribute:

$$c_v^{(0)} = \text{HASH}(\alpha_v)$$

- At iteration $i > 0$ the HASH function is extended to the edge weights:

$$c_v^{(i)} = \text{HASH}\Big(\big(c_v^{(i-1)}, \Omega_{ne_v}, c_{ne_v}^{(i-1)}\big)\Big)$$

Definition 3.1.02 (Attributed 1-WL equivalence)**.** Let $G_1 = (\mathcal{V}_1, \mathcal{E}_1, \alpha_1, \omega_1)$ and $G_2 = (\mathcal{V}_2, \mathcal{E}_2, \alpha_2, \omega_2)$ be two SAUHGs. Then G_1 and G_2 are **attributed WL-equivalent**, noted by $G_1 \sim_{AWL} G_2$, if and only if for all nodes $v_1 \in \mathcal{V}_1$ there exists a corresponding node $v_2 \in \mathcal{V}_2$ with $c_{v_1} = c_{v_2}$. Analogously, two nodes $u \in V_1, v \in \mathcal{V}_2$ are attributed WL-equivalent, noted by $u \sim_{AWL} v$ if and only if $c_u = c_v$.

Remark 3.1.03. Due to the fact that the coloring of node pairs after the 1-AWL are equal, two graphs are 1-AWL equivalent iff the structure is determined as possibly isomorphic analogously to the 1-WL test, and the attributes of the node pairs are equal.

Remark 3.1.04. Having a closer look at the 1-AWL as defined in Def. 3.1.01 one might think that the extension is quite trivial. We can even observe later in Sec. 4 that the extension is just slightly more powerful than the initial 1-WL test. Indeed, there are many possible ways to define an extension for the attributed case, that can be even more powerful than the 1-AWL. One example could be,

$$c_v^{(i)} = \text{HASH}\Big(c_v^{(i-1)}, \{(c_u^{(i-1)}, \omega_{\{v,u\}}) | u \in ne_v\}\Big).$$

Definition 3.1.05 (Attributed k-WL Test)**.** Given a k-tuple $s = (s_1, ..., s_k) \in \mathcal{V}^k$, the j-th neighborhood of s is defined as in 2.2.03. It is an inductive procedure: in iteration $i = 0$, the algorithm labels each k-tuple with its *atomic type*, i.e.,

two k-tuples s and s' in $\mathcal{V}^k$ get the same color if the map $s_i \mapsto s'_i$ induces an attributed isomorphism between the sub–graphs induced from the nodes from s and s', respectively. For iteration $i > 0$, we define the list of weights of the j-th neighborhood of s as:

$$\Omega_{s,j} = \Big(\omega_{\{x,y\}} | (x,y) \in \mathcal{E} \text{ and x,y are in the same tuple in } N_{s,j}\Big).$$

Then we set

$$C^{(i)}_{s,j} = \text{HASH}\left((c^{(i-1)}_{s'} | s' \in N_j(s))\right),$$

and

$$c^{(i)}_s = \text{HASH}\left(c^{(i-1)}_s, (\Omega_{s,1}, ..., \Omega_{s,k}), (C^{(i)}_{s,1}, ..., C^{(i)}_{s,k})\right).$$

Analoguously to the k-WL test, the k-AWL proceeds as the 1-WL test and two tuples s and s' with $o^{(i-1)}_s = o^{(i-1)}_{s'}$ get different colors in iteration i if there exists $j \leq k$ such that the number of $j-$neighbors of s and s', respectively, colored with a certain color is different. Additionally, an increasing k implies a more powerful k-AWL test.

3.2 WL Tests for Dynamic Graphs

For the dynamic WL test an extended version of the attributed 1-WL test (Def. 3.1.01) is defined. Note that it is not equal to an attributed extension of the r-dimensional WL-test defined in [4] since here a special subset of r nodes is used. In the literature, there are several architectures built to deal with dynamic graphs. The 1-DWL test is defined as follows since it mirrors a DGNN [9], a particular GNN architecture that can learn on dynamic graphs. Based on the modified WL test, the corresponding dynamic WL equivalence is then defined.

Definition 3.2.01 (Dynamic 1-WL test)**.** Let $G = (G_t)_{t \in I}$ with $G_t = (\mathcal{V}_t, \mathcal{E}_t, \alpha_t, \omega_t)$ a dynamic graph. Let HASH_t be a bijective function that encodes every possible node feature of G_t with a color. Note that, for each timestamp t, the color set could change. The dynamic 1-WL (1-DWL) test generates a vector of color sets one for each timestamp $t \in I$ by:

- At iteration $i = 0$ the color is set to the hashed node attributes:

$$c^{(0)}_v(t) = \text{HASH}_t\left(\alpha_v(t)\right)$$

- At iteration $i > 0$ the HASH_t function is extended to the edge weights:

$$c^{(i)}_v(t) = \text{HASH}_t\Big(\big(c^{(i-1)}_v(t), \Omega_{ne_v(t)}, c^{(i-1)}_{ne_v(t)}(t)\big)\Big)$$

The algorithm again terminates when the node colorings for both dynamic graphs stagnate.

Definition 3.2.02 (Dynamic 1-WL equivalence)**.** Let $G_1 = (G_t^{(1)})_{t\in I}$ and $G_2 = (G_t^{(2)})_{t\in I}$ be dynamic graphs. Then G_1 and G_2 are **dynamic WL equivalent**, noted by $G_1 \sim_{DWL} G_2$, if and only if for all nodes $v_1 \in \mathcal{V}_{(t)}^{(1)}$ there exists a corresponding node $v_2 \in \mathcal{V}_{(t)}^{(2)}$ with $c_{v_1}(t) = c_{v_2}(t)$ for all $t \in I$. Analogously, two nodes are said to be **dynamic WL equivalent** if their colors resulting from the WL test are equal for all timestamps.

Remark 3.2.03. As stated in Rem. 3.1.04, the 1-AWL test presented in this paper is not the most powerful extension one could define. The same reasons hold for 1-DWL test. Indeed, a more powerful approach could be to additionally consider an aggregation over the past colours of each node.

Definition 3.2.04 (dynamic k-WL Test)**.** Given a k-tuple $s = (s_1, ..., s_k) \in \mathcal{V}_t^k$, the j-th neighborhood of s at time t is obtained by replacing the j-th component of s by every node from $\mathcal{V}_t$:

$$N_{s,j}(t) = \{(s_1, ..., s_{j-1}, r, s_{j+1}, ..., s_k) \,|r \in \mathcal{V}_t\}$$

At iteration $i = 0$, the algorithm labels each k-tuple with its *atomic type* analoguously to the k-AWL, but for any $t \in I$. For iteration $i > 0$, we define the list of weights of the j-th neighborhood of s at time t as:

$$\Omega_{s,j} = \Big(\omega_{\{x,y\}} | (x, y) \in \mathcal{E} \text{ and x,y are in the same tuple in } N_{s,j}(t)\Big).$$

Then we set

$$C_{s,j}^{(i)}(t) = \mathrm{HASH}_t\left((c_{s'}^{(i-1)}(t) | s' \in N_{s,j}(t))\right),$$

and

$$c_s^{(i)}(t) = \mathrm{HASH}_t\left(c_s^{(i-1)}(t), (\Omega_{s,1}(t), ..., \Omega_{s,k}(t)), (C_{s,1}^{(i)}(t), ..., C_{s,k}^{(i)}(t))\right).$$

Note that the HASH_t function can be different in each timestamp.

4 Weisfeiler-Lehman Hierarchy

In this section the attributed and dynamic WL tests are positioned into the Weisfeiler-Lehman Hierarchy.

Theorem 4.01. 1-WL test $\subsetneq$ 1-AWL test

Proof. First, we show that 1-WL test is a subset of the 1-AWL test and then we give a counter example for them being equal.

- $\subset$ Let $g_1 := (\mathcal{V}_1, \mathcal{E}_1),\ g_2 := (\mathcal{V}_2, \mathcal{E}_2)$ be two graphs with $g_1 \sim_{1-WL} g_2$. Introducing empty attribute functions $\alpha_i : \mathcal{V}_i \to \emptyset,\ \omega_i : \mathcal{E}_i \to \emptyset$ for $i = 1, 2$ leads to the the same initial coloring of the nodes and unattributed edges, and thus to the performance of the 1-WL test. Therefore, it follows $g_1 \sim_{1-AWL} g_2$.

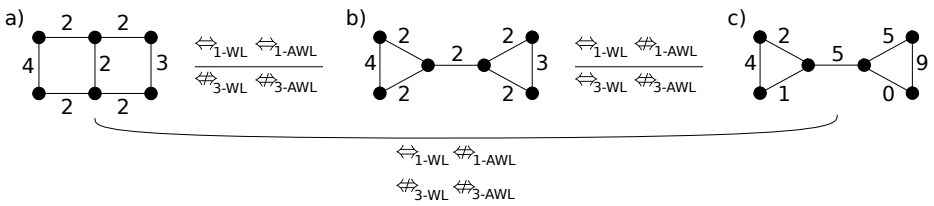

Fig. 3. The graphs a) and b) are falsely recognized as 1-WL and 1-AWL isomorphic; same holds for the 3-WL and 3-AWL. The graphs b) and c) are falsely recognized isomprohic just by the 1-WL/3-WL test while the 1-AWL/3-AWL test correctly recognizes both graphs as non isomorphic. The same result holds for the graphs b) and c) for 1-WL and 1-AWL while 3-WL and 3-AWL can distinguish both. The 1-WL and 3-WL equivalences on the graphs from a) and b) and a) and c) without attributes are taken from [7, §3.1]
.

$\neq$ In Fig. 3 one can see that the 1-AWL test can distinguish the first and last graph (graphs a and c), while the 1-WL test fails.

Corollary 4.02. 2-WL test $\subsetneq$ 1-AWL test

Proof. As stated in [10, §3.5, Cor. 3.5.8], 1-WL test and 2-WL test are equivalent.

Corollary 4.03. k-WL test $\subsetneq$ k-AWL test.

Proof. Follows analogously from the proof of Thm. 4.01.

Corollary 4.04. The k-DWL test $\subseteq$ (k+1)-DWL test.

Proof. Follows immediately from the static Weisfeiler-Lehmann Hierarchy.

Theorem 4.05. The 3-WL test and the 1-AWL test cannot be compared regarding to the Weisfeiler Lehmann Hierarchy.

Proof. In Fig. 3 one can see that the 3-WL test can distinguish the graphs a) and b) but the 1-AWL test cannot. However, the 3-WL test cannot distinguish the graphs b) and c) while the 1-AWL test can.

Theorem 4.06. 1-DWL test = 1-AWL test

Proof. Let $G = (G_t)_{t\in I}$ be a dynamic graph and $\text{STATIC}(G) =: G' = (\mathcal{V}', \mathcal{E}', \alpha', \omega')$ the SAUHG resulting from a bijective graph type transformation (cf. Rem. 2.1.03). Furthermore, let $\mathcal{V} := \bigcup_{t\in I} \mathcal{V}_t$ be the set of all nodes appearing in the graph sequence of G and $\tilde{\alpha} : \mathcal{V} \times I \to \mathcal{A} \times \perp$ with $\tilde{\alpha}_v(t) := (\alpha_v(t), \rho)$ be the extended attribute function for all nodes including a flag $\rho \in \{0, 1\}$ for the existence of a node at time t. The theorem follows immediately from

$$\left(c_u^{(i)}(t)\right)_{t\in I} = \left(c_v^{(i)}(t)\right)_{t\in I} \Leftrightarrow c_u^{(i)} = c_v^{(i)}.$$

for all iterations i and $u, v \in \mathcal{V}$. By induction it is:

$i = 0$:

$$\left(c_u^{(0)}(t)\right)_{t\in I} = \left(c_v^{(0)}(t)\right)_{t\in I} \overset{\text{Def. 3.2.01}}{\Leftrightarrow} \left(\text{HASH}\left(\tilde{\alpha}_u(t)\right)\right)_{t\in I} = \left(\text{HASH}\left(\tilde{\alpha}_v(t)\right)\right)_{t\in I}$$
$$\Leftrightarrow \tilde{\alpha}_u(t) = \tilde{\alpha}_v(t)\ \forall\, t \in I \overset{\text{by constr.}}{\Leftrightarrow} \alpha'_u = \alpha'_v \Leftrightarrow \text{HASH}\left(\alpha'_u\right) = \text{HASH}\left(\alpha'_v\right)$$
$$\Leftrightarrow c_u^{(0)} = c_v^{(0)}$$

$i > 0$: Assume the induction hypothesis (IH) is true for $i-1$ and show the assumption is also true for i.

$$\left(c_u^{(i)}(t)\right)_{t\in I} = \left(c_v^{(i)}(t)\right)_{t\in I}$$
$$\overset{\text{Def. 3.2.01}}{\Leftrightarrow} \left(\text{HASH}\left(c_u^{(i-1)}(t), \Omega_{ne_u(t)}, c_{ne_u(t)}^{(i-1)}\right)\right)_{t\in I}$$
$$= \left(\text{HASH}\left(c_v^{(i-1)}(t), \Omega_{ne_v(t)}, c_{ne_v(t)}^{(i-1)}\right)\right)_{t\in I}$$
$$\overset{\text{Bij. of HASH}}{\Leftrightarrow} c_u^{(i-1)}(t) = c_v^{(i-1)}(t) \wedge \Omega_{ne_u(t)} = \Omega_{ne_v(t)} \wedge c_{ne_u(t)}^{(i-1)} = c_{ne_v(t)}^{(i-1)}\ \forall\, t \in I$$
$$\overset{\text{IH}}{\Leftrightarrow} c_u^{(i-1)} = c_v^{(i-1)} \wedge \Omega_{ne_u} = \Omega_{ne_v} \wedge c_{ne_u}^{(i-1)} = c_{ne_v}^{(i-1)}$$
$$\overset{\text{Bij. of HASH}}{\Leftrightarrow} \text{HASH}\left(\left(c_u^{(i-1)}, \Omega_{ne_u}, c_{ne_u}^{(i-1)}\right)\right) = \text{HASH}\left(\left(c_v^{(i-1)}, \Omega_{ne_v}, c_{ne_v}^{(i-1)}\right)\right)$$
$$\overset{\text{Def. 3.1.01}}{\Leftrightarrow} c_u^{(i)} = c_v^{(i)}$$

Corollary 4.07 (Weisfeiler-Lehman Lattice)**.** The Weisfeiler-Lehman hierarchy is a infinite, bounded, complete and distributive lattice (as a consequence of theorem 2.3.07). The partial order relation is: $A \leq B$ iff the partition $\mathcal{P}_A$ induced by the test A is equal or coarser than the partition $\mathcal{P}_B$ induced by the test B, i.e. $|\mathcal{P}_A| \leq |\mathcal{P}_B|$ and each subset in $\mathcal{P}_A$ is a union of subsets in $\mathcal{P}_B$, i.e.,

$$\forall\, \mathcal{A} \in \mathcal{P}_A\ \exists\, \{\mathcal{B}_i\} \subset \mathcal{P}_B, \text{ for some indices } i, \text{ s.t. } \mathcal{A} = \bigcup \mathcal{B}.$$

The minimum element of this lattice is the 1-WL test, equivalent to the 2-WL test, while the maximum element is the Attributed Graph Isomorphism test (AGI) and Dynamic Graph Isomorphism test (DGI).

5 Conclusion

Graph isomorphism is still a non-trivial problem laying in the class NP. However, a common practical solution for at least distinguishing non-isomorphic graphs is the Weisfeiler-Lehman (WL) method. Given that the initial WL tests can just handle node attributed graphs, in this paper we extended the WL tests to arbitrary graphs. Particularly, we introduced a $k-$dimensional attributed and dynamic version of the $k-$WL test, respectively. With this WL extensions we also extended the initial WL-hierarchy that induces a partial ordering on the WL-test. This results in a complete and distributive lattice, which paves the way for concepts of lattice theory to be utilized in the context of graph isomorphism and WL tests.

Acknowledgments

This work was partially supported by the Ministry of Education and Research Germany (BMBF, grant number 01IS20047A). We thank Dr. Josephine M. Thomas, Prof. Gerd Stumme and Maximilian Stubbemann (University Kassel), as well as Prof. Paolo Aglianò (University Siena) for the very informative, productive and inspiring discussions.

References

1. AA Leman and B Weisfeiler. A reduction of a graph to a canonical form and an algebra arising during this reduction. *Nauchno-Technicheskaya Informatsiya*, 2(9):12–16, 1968.
2. Shervashidze, N. and Schweitzer, P. and Van Leeuwen, E. J. and Mehlhorn, K. and Borgwardt, K. M. Weisfeiler-Lehman Graph Kernels. *Journal of Machine Learning Research*, 2011.
3. Jin-yi Cai, Martin Fürer, and Neil Immerman. An optimal lower bound on the number of variables for graph identifications. *Comb.*, 12(4):389–410, 1992.
4. Morris, Christopher and Ritzert, Martin and Fey, Matthias and Hamilton, William L and Lenssen, Jan Eric and Rattan, Gaurav and Grohe, Martin. Weisfeiler and Leman Go Neural: Higher-Order Graph Neural Networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 4602–4609, 2019.
5. Josephine Maria Thomas, Silvia Beddar-Wiesing, and Alice Moallemy-Oureh. Graph type expressivity and transformations. *arXiv:2109.10708*, 09 2021.
6. Martin Grohe and Pascal Schweitzer. The graph isomorphism problem. *Commun. ACM*, 63(11):128–134, 2020.
7. R. Sato. A survey on the expressive power of graph neural networks. *ArXiv*, abs/2003.04078, 2020.
8. Mai Gehrke and Sam van Gool. Topological duality for distributive lattices, and applications. *arXiv preprint arXiv:2203.03286*, 2022.
9. Joakim Skardinga, Bogdan Gabrys, and Katarzyna Musial. Foundations and modelling of dynamic networks using dynamic graph neural networks: A survey. *IEEE Access*, 2021.
10. Martin Grohe. *Descriptive complexity, canonisation, and definable graph structure theory*, volume 47. Cambridge University Press, 2017.