SpeqNets: Sparsity-aware Permutation-equivariant Graph Networks

Christopher Morris [1 2 3] **Gaurav Rattan** [1] **Sandra Kiefer** [4] **Siamak Ravanbkash** [2 3]

Abstract

While message-passing graph neural networks have clear limitations in approximating permutation-equivariant functions over graphs or general relational data, more expressive, higher-order graph neural networks do not scale to large graphs. They either operate on k-order tensors or consider all k-node subgraphs, implying an exponential dependence on k in memory requirements, and do not adapt to the sparsity of the graph. By introducing new heuristics for the graph isomorphism problem, we devise a class of universal, permutation-equivariant graph networks, which, unlike previous architectures, offer a fine-grained control between expressivity and scalability and adapt to the sparsity of the graph. These architectures lead to vastly reduced computation times compared to standard higher-order graph networks in the supervised node- and graph-level classification and regression regime while significantly improving standard graph neural network and graph kernel architectures in terms of predictive performance.

1. Introduction

Graph-structured data is ubiquitous across application domains ranging from chemo- and bioinformatics (Barabasi & Oltvai, 2004; Jumper et al., 2021; Stokes et al., 2020) to image (Simonovsky & Komodakis, 2017) and social-network analysis (Easley & Kleinberg, 2010). To develop successful machine-learning models in these domains, we need techniques that exploit the rich information inherent in the graph structure and the feature information within nodes and edges. In recent years, numerous approaches have been proposed for machine learning with graphs—most notably, approaches based on *graph kernels* (Borgwardt et al., 2020; Kriege et al., 2020) or using *graph neural networks* (GNNs) (Chami et al., 2020; Gilmer et al., 2017; Grohe, 2021; Morris et al., 2021). Here, graph kernels based on the 1-*dimensional Weisfeiler–Leman algorithm* (1-WL) (Weisfeiler & Leman, 1968), a simple heuristic for the graph isomorphism problem, and corresponding GNNs (Morris et al., 2019; Xu et al., 2019) have recently advanced the state-of-the-art in supervised node- and graph-level learning. However, the 1-WL operates via simple neighborhood aggregation, and the purely local nature of the related approaches misses important patterns in the given data. Moreover, they are only applicable to binary structures and therefore cannot deal with general structures containing relations of higher arity, e.g., hypergraphs. A more powerful algorithm for graph isomorphism testing is the k-*dimensional Weisfeiler–Leman algorithm* (k-WL) (Babai, 1979; Cai et al., 1992).[1] The algorithm captures more global, higher-order patterns by iteratively computing a coloring or labeling for k-tuples defined over the set of nodes of a given graph based on a certain notion of adjacency between tuples. See (Kiefer, 2020b) for a survey and more background. However, since the algorithm considers all n^k many k-tuples of an n-node graph, it does not scale to large real-world graphs. Moreover, the cardinality of the considered neighborhood is always $k \cdot n$. Hence, a potential *sparsity* of the input graph does not reduce the running time.

New neural architectures that possess the same power as the k-WL in terms of separating non-isomorphic graphs (Azizian & Lelarge, 2020; Geerts, 2020; Maron et al., 2019b) suffer from the same drawbacks, i.e., their memory requirement is lower-bounded by n^k for an n-node graph, and they have to resort to dense matrix multiplication. Recently, Morris et al. (2020b) introduced the local variant (δ-k-LWL) of the k-WL considering only a subset of the neighborhoods in k-WL. However, like the original algorithm, the local variant operates on the set of all possible k-tuples, again resulting in the same (exponential) memory requirements, rendering the algorithm not practical for large, real-world graphs.

[*]Equal contribution [1]Department of Computer Science, RWTH Aachen University, Aachen, Germany [2]Department of Computer Science, McGill University, Montreal, Canada [3]Mila, Quebec AI Institute [4]Max Planck Institute for Software Systems, Saarland Informatics Campus, Germany. Correspondence to: Christopher Morris <chris@christophermorris.info>.

Proceedings of the 39^{th} International Conference on Machine Learning, Baltimore, Maryland, USA, PMLR 162, 2022.

[1]In (Babai, 2016), László Babai mentions that he introduced the algorithm in 1979 together with Rudolf Mathon.

Present work To address the described drawbacks, we introduce a new set of heuristics for the graph isomorphism problem, denoted (k, s)-LWL, which only considers a subset of all k-tuples, namely those *inducing subgraphs with at most s connected components*. We study the effect of k and s on the expressive power of the heuristics. Specifically, we show that the $(k, 1)$-LWL induces a hierarchy of provably expressive heuristics for the graph isomorphism problem, i.e., with increasing k, the algorithm becomes strictly more expressive. Additionally, we prove that the $(k, 2)$-LWL is strictly more expressive than the $(k, 1)$-LWL. Further, we separate the $(k, 2)$-LWL and (k, k)-LWL by showing that the (k, k)-LWL is strictly more expressive than the $(k, 2)$-LWL. Building on these combinatorial insights, we derive corresponding provably expressive, permutation-equivariant neural architectures, denoted (k, s)-SpeqNets, which offer a more fine-grained trade-off between scalability and expressivity compared to previous architectures based on the k-WL, see Figure 1 for a high-level overview of the theoretical results. Empirically, we show how our architectures offer vastly reduced computation times while beating baseline GNNs and other higher-order graph networks in terms of predictive performance on well-known node- and graph-level prediction benchmark datasets.

1.1. Related work

In the following, we review related work from graph kernels, GNNs and graph theory, see Appendix A for an extended discussion.

Graph kernels Historically, kernel methods—which implicitly or explicitly map graphs to elements of a Hilbert space—have been the dominant approach for supervised learning on graphs. Important early work in this area includes random-walk based kernels (Gärtner et al., 2003; Kashima et al., 2003; Kriege et al., 2017) and kernels based on shortest paths (Borgwardt & Kriegel, 2005). More recently, developments in the field have emphasized scalability, focusing on techniques that bypass expensive Gram matrix computations by using explicit feature maps, see, e.g., (Shervashidze et al., 2011). Morris et al. (2017) devised a local, set-based variant of the k-WL and a corresponding kernel. However, the approach is (provably) weaker than the tuple-based algorithm. Further, Morris et al. (2020a) proposed kernels based on the δ-k-LWL.

Yanardag & Vishwanathan (2015a) successfully employed Graphlet (Shervashidze et al., 2009), and Weisfeiler–Leman kernels within frameworks for smoothed (Yanardag & Vishwanathan, 2015a) and deep graph kernels (Yanardag & Vishwanathan, 2015b). Other recent work focuses on assignment-based (Johansson & Dubhashi, 2015; Kriege et al., 2016; Nikolentzos et al., 2017), spectral (Kondor & Pan, 2016; Verma & Zhang, 2017), graph decomposition (Nikolentzos et al., 2018), randomized binning approaches (Heimann et al., 2019), and the extension of kernels based on the 1-WL (Rieck et al., 2019; Togninalli et al., 2019). For a theoretical investigation of graph kernels, see (Kriege et al., 2018), and for a thorough survey of graph kernels, see (Borgwardt et al., 2020; Kriege et al., 2020).

GNNs Recently, GNNs (Gilmer et al., 2017; Scarselli et al., 2009) emerged as an alternative to graph kernels. Notable instances of this architecture include, e.g., (Duvenaud et al., 2015; Hamilton et al., 2017; Veličković et al., 2018), which can be subsumed under the message-passing framework introduced in (Gilmer et al., 2017). Also, approaches based on spectral information were introduced in, e.g., (Defferrard et al., 2016; Bruna et al., 2014; Kipf & Welling, 2017; Monti et al., 2017)—all of which descend from early work in (Kireev, 1995; Baskin et al., 1997; Micheli & Sestito, 2005; Merkwirth & Lengauer, 2005; Micheli, 2009; Sperduti & Starita, 1997; Scarselli et al., 2009).

Limits of GNNs and more expressive architectures Recently, connections of GNNs to Weisfeiler–Leman type algorithms have been shown (Azizian & Lelarge, 2020; Barceló et al., 2020; Chen et al., 2019b; Geerts et al., 2020; Geerts, 2020; Maehara & NT, 2019; Maron et al., 2019a; Morris et al., 2019; Xu et al., 2019). Specifically, (Morris et al., 2019; Xu et al., 2019) showed that the expressive power of any possible GNN architecture is limited by the 1-WL in terms of distinguishing non-isomorphic graphs.

Triggered by the above results, a large set of papers proposed architectures to overcome the expressivity limitations of the 1-WL. Morris et al. (2019) introduced *k-dimensional* GNNs (k-GNN) which rely on a message-passing scheme between subgraphs of cardinality k. Similar to (Morris et al., 2017), the paper employed a local, set-based (neural) variant of the k-WL. Later, this was refined in (Maron et al., 2019a; Azizian & Lelarge, 2020) by introducing *k-order folklore graph neural networks* (k-FGNN), which are equivalent to the folklore or oblivious variant of the k-WL (Grohe, 2021; Morris et al., 2021) in terms of distinguishing non-isomorphic graphs. Subsequently, Morris et al. (2020b) introduced neural architectures based on the δ-k-LWL, which only considers a subset of the neighborhood from the k-WL, taking sparsity of the underlying graph (to some extent) into account. Although more scalable, the algorithm reaches computational exhaustion on large-scale graphs since it considers all n^k tuples of size k. Chen et al. (2019b) connected the theory of universal approximations of permutation-invariant functions and the graph isomorphism viewpoint and introduced a variation of the 2-WL. See (Morris et al., 2021) for an in-depth survey on this topic.

Recent works have extended the expressive power of GNNs,

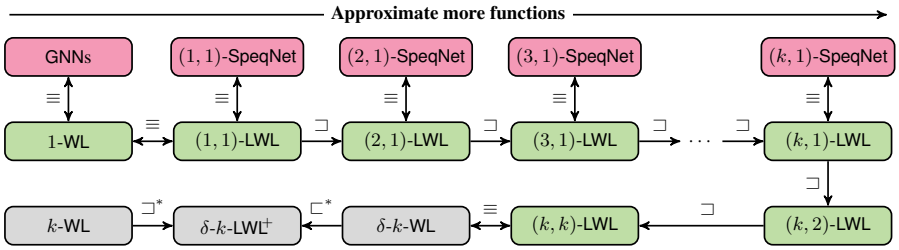

Figure 1. Overview of the expressive power of the proposed algorithms and neural architectures. The green and red nodes represent algorithms proposed in the present work. Forward arrows point to more powerful algorithms or neural architectures. *—Proven in (Morris et al., 2020b). $A \sqsubset B$ ($A \equiv B$): algorithm A is strictly more powerful than (equally powerful as) B.

e.g., by encoding node identifiers (Murphy et al., 2019; Vignac et al., 2020), leveraging random features (Abboud et al., 2020; Dasoulas et al., 2020; Sato et al., 2020), subgraph information (Bevilacqua et al., 2021; Bouritsas et al., 2020; Cotta et al., 2021; Papp et al., 2021; Thiede et al., 2021; You et al., 2021; Zhang & Li, 2021; Zhao et al., 2021), homomorphism counts (Barceló et al., 2021; NT & Maehara, 2020), spectral information (Balcilar et al., 2021), simplicial and cellular complexes (Bodnar et al., 2021b;a), random walks (Tönshoff et al., 2021), graph decompositions (Talak et al., 2021), distance (Li et al., 2020) and directional information (Beaini et al., 2020).

However, all of the above approaches mentioned in the previous paragraph only overcome limitations of the 1-WL, 2-WL, or 3-WL, and do not induce a hierarchy of provably powerful, permutation-equivariant neural architectures aligned with the k-WL hierarchy.

See Appendix A for an overview of the properties of k-WL.

2. Preliminaries

We briefly describe the Weisfeiler–Leman algorithm and, along the way, introduce our notation, see Appendix B for details. We let $[n] \coloneqq \{1, \dots, n\} \subset \mathbb{N}$ for $n \geq 1$, and use $\{\!\{ \dots \}\!\}$ to denote multisets. We also use standard concepts from graph theory (such as graphs, directed graphs, neighbors, trees, and so on). The vertex and the edge set of a graph G are denoted by $V(G)$ and $E(G)$, respectively. The *neighborhood* of v in $V(G)$ is $\delta(v) = \{u \in V(G) \mid (v, u) \in E(G)\}$. We say that two graphs G and H are *isomorphic* ($G \simeq H$) if there exists a bijection $\varphi \colon V(G) \to V(H)$ preserving the adjacency relation, i.e., (u, v) is in $E(G)$ if and only if $(\varphi(u), \varphi(v))$ is in $E(H)$, call φ an *isomorphism* from G to H. If the graphs have vertex or edges labels, the isomorphism is additionally required to match these labels. Let $\mathbf{v}$ be a *tuple* in $V(G)^k$ for $k > 0$, then $G[\mathbf{v}]$ is the subgraph induced by the elements of $\mathbf{v}$, where the nodes are labeled with integers from $\{1, \dots, k\}$ corresponding to their positions in $\mathbf{v}$. A *connected component* of a graph G is an inclusion-wise maximal subgraph of G in which every two nodes are connected by paths.

2.1. Node-refinement algorithms

In the following, we review the Weisfeiler–Leman algorithm and related variants (Morris et al., 2020b). Let k be a fixed positive integer and let $V(G)^k$ denote the set of k-tuples of nodes of the graph G. A *coloring* of $V(G)^k$ is a mapping $C \colon V(G)^k \to \mathbb{N}$, i.e., we assign a number (color) to every tuple in $V(G)^k$. The *initial coloring* C_0 of $V(G)^k$ is specified by the atomic types of the tuples, i.e., two tuples $\mathbf{v}$ and $\mathbf{w}$ in $V(G)^k$ have the same initial color iff the mapping $v_i \mapsto w_i$ induces an isomorphism between the labeled subgraphs $G[\mathbf{v}]$ and $G[\mathbf{w}]$. Note that, given a tuple $\mathbf{v}$ in $V(G)^k$, we can upper-bound the running time of the computation of this initial coloring for $\mathbf{v}$ by $\mathcal{O}(k^2)$. A *color class* corresponding to a color c is the set of all tuples colored c, i.e., the set $C^{-1}(c)$.

For j in $[k]$ and w in $V(G)$, let $\phi_j(\mathbf{v}, w)$ be the k-tuple obtained by replacing the jth component of $\mathbf{v}$ with the node w. That is, $\phi_j(\mathbf{v}, w) = (v_1, \dots, v_{j-1}, w, v_{j+1}, \dots, v_k)$. If $\mathbf{w} = \phi_j(\mathbf{v}, w)$ for some w in $V(G)$, call $\mathbf{w}$ a *j-neighbor* of $\mathbf{v}$. The *neighborhood* of $\mathbf{v}$ is the set of all $\mathbf{w}$ such that $\mathbf{w} = \phi_j(\mathbf{v}, w)$ holds for some j in $[k]$ and a w in $V(G)$.

The *refinement* of a coloring $C \colon V(G)^k \to \mathbb{N}$, denoted by $\widehat{C}$, is a coloring $\widehat{C} \colon V(G)^k \to \mathbb{N}$ defined as follows. For each j in $[k]$, collect the colors of the j-neighbors of $\mathbf{v}$ in a multiset $S_j = \{\!\{C(\phi_j(\mathbf{v}, w)) \mid w \text{ in } V(G)\}\!\}$. Then, for a tuple $\mathbf{v}$, define

$$\widehat{C}(\mathbf{v}) \coloneqq (C(\mathbf{v}), M(\mathbf{v})),$$

where $M(\mathbf{v})$ is the k-tuple $(S_1, \dots, S_k)$. For consistency, the strings $\widehat{C}(\mathbf{v})$ thus obtained are lexicographically sorted and renamed as fresh integers, i.e., ones that have not been used in previous iterations. Observe that the new color $\widehat{C}(\mathbf{v})$ of $\mathbf{v}$ is solely dictated by the color histogram of the neighborhood of $\mathbf{v}$. In general, a different mapping $M(\cdot)$ could be used, depending on the neighborhood information that we would like to aggregate. We will refer to such $M(\cdot)$ as *aggregation maps*.

k-dimensional Weisfeiler–Leman For $k \geq 2$, the k-WL computes a coloring $C_\infty \colon V(G)^k \to \mathbb{N}$ of a given graph

G, as follows.[2] To begin with, the initial coloring C_0 is computed. Then, starting with C_0, successive refinements $C_{i+1} = \widehat{C_i}$ are computed until convergence. That is,

$$C_{i+1}(\mathbf{v}) = (C_i(\mathbf{v}), M_i(\mathbf{v})),$$

where

$$M_i(\mathbf{v}) \coloneqq \big(\{\!\{C_i(\phi_1(\mathbf{v}, w)) \mid w \in V(G)\}\!\}, \ldots, \{\!\{C_i(\phi_k(\mathbf{v}, w)) \mid w \in V(G)\}\!\}\big).$$

The successive refinement steps are also called *rounds* or *iterations*. Since the color classes form a partition of $V(G)^k$, there must exist a finite $\ell \leq |V(G)|^k$ such that $C_\ell = \widehat{C_\ell}$, i.e., the partition induced by C_ℓ is not refined further. The k-WL outputs C_ℓ as the *stable coloring* C_∞.

The k-WL *distinguishes* two graphs G and H if, upon running the k-WL on their disjoint union $G \dot\cup H$, there exists a color c in $\mathbb{N}$ in the stable coloring such that the corresponding color class S_c satisfies

$$|V(G)^k \cap S_c| \neq |V(H)^k \cap S_c|,$$

i.e., the numbers of c-colored tuples in $V(G)^k$ and $V(H)^k$ differ. Two graphs distinguished by the k-WL must be non-isomorphic, because the algorithm is defined in an isomorphism-invariant way.

Local δ-k-dimensional Weisfeiler–Leman algorithm Morris et al. (2020b) introduced a more efficient modification of the k-WL, namely the *local δ-k-dimensional Weisfeiler–Leman algorithm* (δ-k-LWL). In contrast to the k-WL, the δ-k-LWL considers only a subset of the entire neighborhood of a node tuple. Let the tuple $\mathbf{w} = \phi_j(\mathbf{v}, w)$ be a j-neighbor of $\mathbf{v}$. We say that $\mathbf{w}$ is a *local* j-neighbor of $\mathbf{v}$ if w is adjacent to the replaced node v_j. Otherwise, the tuple $\mathbf{w}$ is a *global* j-neighbor of $\mathbf{v}$. The δ-k-LWL considers only local neighbors during the neighborhood aggregation process, and discards any information about the global neighbors. Formally, the δ-k-LWL algorithm refines a coloring $C_i^{k,\delta}$, obtained after i rounds of δ-k-LWL, via the aggregation map,

$$M_i^\delta(\mathbf{v}) \coloneqq \big(\{\!\{C_i^{k,\delta}(\phi_1(\mathbf{v}, w)) \mid w \in \delta(v_1)\}\!\}, \ldots, \{\!\{C_i^{k,\delta}(\phi_k(\mathbf{v}, w)) \mid w \in \delta(v_k)\}\!\}\big),$$

hence considering only the local j-neighbors of the tuple $\mathbf{v}$ in each iteration. The coloring function for the δ-k-LWL is then defined by

$$C_{i+1}^{k,\delta}(\mathbf{v}) \coloneqq (C_i^{k,\delta}(\mathbf{v}), M_i^\delta(\mathbf{v})).$$

[2] We define the 1-WL in the next subsection.

We define the 1-WL to be the δ-1-LWL, which is commonly known as Color Refinement or Naive Node Classification.[3] Hence, we can equivalently define

$$C_{i+1}^{1,\delta}(v) = (C_i^{1,\delta}(v), \{\!\{C_i^{1,\delta}(w) \mid w \in \delta(v)\}\!\}).$$

for a node v in $V(G)$. Morris et al. (2020b) also defined the δ-k-LWL^+, a minor variation of the δ-k-LWL. Formally, the δ-k-LWL^+ refines a coloring C_i (obtained after i rounds) via the aggregation function $M^{\delta,+}(\mathbf{v}) =$

$$\big(\{\!\{(C_i^{k,\delta}(\phi_1(\mathbf{v}, w)), \#_i^1(\mathbf{v}, \phi_1(\mathbf{v}, w))) \mid w \in \delta(v_1)\}\!\}, \ldots, \{\!\{(C_i^{k,\delta}(\phi_k(\mathbf{v}, w)), \#_i^k(\mathbf{v}, \phi_k(\mathbf{v}, w))) \mid w \in \delta(v_k)\}\!\}\big),$$

instead of the δ-k-LWL aggregation defined in Equation (4). Here, we set

$$\#_i^j(\mathbf{v}, \mathbf{x}) \coloneqq \big|\{\mathbf{w} \colon \mathbf{w} \sim_j \mathbf{v},\, C_i^{k,\delta}(\mathbf{w}) = C_i^{k,\delta}(\mathbf{x})\}\big|,$$

where $\mathbf{w} \sim_j \mathbf{v}$ denotes that $\mathbf{w}$ is a j-neighbor of $\mathbf{v}$, for j in $[k]$. Essentially, $\#_i^j(\mathbf{v}, \mathbf{x})$ counts the number of (local or global) j-neighbors of $\mathbf{v}$ which have the same color as $\mathbf{x}$ under the coloring C_i (i.e., after i rounds). Morris et al. (2020b) showed that the δ-k-LWL^+ is slightly more powerful than the k-WL in distinguishing non-isomorphic graphs.

Comparing k-WL variants Let A_1 and A_2 denote two node-refinement algorithms. We write $A_1 \sqsubseteq A_2$ if A_1 distinguishes between all non-isomorphic pairs that A_2 distinguishes, and $A_1 \equiv A_2$ if both $A_1 \sqsubseteq A_2$ and $A_2 \sqsubseteq A_1$ hold. The corresponding strict relation is denoted by $\sqsubset$. For example, following Morris et al. (2020b), it holds that δ-k-$\mathsf{LWL}^+ \sqsubset k$-$\mathsf{WL}$.

The Weisfeiler–Leman hierarchy and permutation-invariant function approximation The Weisfeiler–Leman hierarchy is a purely combinatorial algorithm for testing graph isomorphism. However, the graph isomorphism function, mapping non-isomorphic graphs to different values, is the hardest to approximate permutation-invariant function. Hence, the Weisfeiler–Leman hierarchy has strong ties to GNNs' capabilities to approximate permutation-invariant or equivariant functions over graphs. For example, Morris et al. (2019); Xu et al. (2019) showed that the expressive power of any possible GNN architecture is limited by 1-WL in terms of distinguishing non-isomorphic graphs. Azizian & Lelarge (2020) refined these results by showing that if an architecture is capable of simulating k-WL and allows the application of universal

[3] Strictly speaking, the 1-WL and Color Refinement are two different algorithms. That is, the 1-WL considers neighbors and non-neighbors to update the coloring, resulting in a slightly higher expressivity when distinguishing nodes in a given graph, see (Grohe, 2021) for details. For brevity, we consider both algorithms to be equivalent.

neural networks on vertex features, it will be able to approximate any permutation-equivariant function below the expressive power of k-WL; see also (Chen et al., 2019b). Hence, if one shows that one architecture distinguishes more graphs than another, it follows that the corresponding GNN can approximate more functions.

Kernels based on node-refinement algorithms After running the k-WL (and the other node-refinement algorithms), the concatenation of the histogram of colors in each iteration can be used as a feature vector in a kernel computation. Specifically, in the histogram, for every color c in $\mathbb{N}$, an entry contains the number of nodes or k-tuples colored c.

3. The (k, s)-LWL algorithm

Since both k-WL and its local variant δ-k-LWL consider all k-tuples of a graph, they do not scale to large graphs for larger k. Specifically, for an n-node graph, the memory requirement is in $\Omega(n^k)$. Further, since the k-WL considers the graph structure only at initialization, it does not adapt to its sparsity, i.e., it does not run faster on sparser graphs. To address this issue, we introduce the (k, s)-LWL. The algorithm offers more fine-grained control over the trade-off between expressivity and scalability by only considering a subset of all k-tuples, namely those inducing subgraphs with at most s *connected components*. This combinatorial algorithm will be the basis of the permutation-equivariant neural architectures of Section 4.

Let G be a graph. Then $\#\mathsf{com}(G)$ denotes the number of (connected) components of G. Further, let $k \geq 1$ and $1 \leq s \leq k$, then

$$V(G)^k_s \coloneqq \{\mathbf{v} \in V(G)^k \mid \#\mathsf{com}(G[\mathbf{v}]) \leq s\}$$

is the set of (k, s)-*tuples* of nodes, i.e, k-tuples which induce (sub-)graphs with at most s (connected) components.

In contrast to the algorithms of Appendix B.1, the (k, s)-LWL colors tuples from $V(G)^k_s$ instead of the entire $V(G)^k$. Hence, analogously to Appendix B.1, a coloring of $V(G)^k_s$ is a mapping $C^{k,s}\colon V(G)^k_s \to \mathbb{N}$, assigning a number (color) to every tuple in $V(G)^k_s$. The initial coloring $C^{k,s}_0$ of $V(G)^k_s$ is defined in the same way as before, i.e., specified by the isomorphism types of the tuples, see Appendix B.1. Subsequently, the coloring is updated using the δ-k-LWL aggregation map, see Equation (4). Hence, the (k, s)-LWL is a variant of the δ-k-LWL considering only (k, s)-tuples, i.e., Equation (4) is replaced with $M^{\delta,k,s}_i(\mathbf{v}) \coloneqq$

$$\big(\{\!\{C^{k,s}_i(\phi_1(\mathbf{v}, w)) \mid w \in \delta(v_1) \text{ and } \phi_1(\mathbf{v}, w) \in V(G)^k_s\}\!\},$$
$$\ldots, \{\!\{C^{k,s}_i(\phi_k(\mathbf{v}, w)) \mid w \in \delta(v_k) \text{ and } \phi_k(\mathbf{v}, w) \in V(G)^k_s\}\!\}\big),$$

i.e., $M^{\delta}_i(\mathbf{v})$ restricted to colors of (k, s)-tuples. The stable coloring $C^{k,s}_\infty$ is defined analogously to the stable coloring C^k_∞. In the following two subsections, we investigate the properties of the algorithm in detail. Analogously to the δ-k-LWL$^+$, we also define the (k, s)-LWL$^+$, see Appendix B for details.

3.1. Expressivity

Here, we investigate the expressivity of the (k, s)-LWL, i.e., its ability to distinguish non-isomorphic graphs, for different choices of k and s. In Section 4, we will leverage these results to devise universal, permutation-equivariant graph networks. We start off with the following simple observation. Since the (k, k)-LWL colors all k-tuples, it is equal to the δ-k-LWL.

Observation 1. Let $k \geq 1$, then

$$(k, k)\text{-LWL} \equiv \delta\text{-}k\text{-LWL}, \text{ and}$$
$$(1, 1)\text{-LWL} \equiv \delta\text{-}k\text{-LWL} \equiv 1\text{-WL}.$$

The following result shows that the $(k, 1)$-LWL form a *hierarchy*, i.e., the algorithm becomes more expressive as k increases.

Theorem 1. Let $k \geq 1$, then

$$(k + 1, 1)\text{-LWL} \sqsubset (k, 1)\text{-LWL}.$$

Moreover, we also show that the $(k, 2)$-LWL is more expressive than the $(k, 1)$-LWL.

Proposition 2. For $k \geq 2$, it holds that

$$(k, 2)\text{-LWL} \sqsubset (k, 1)\text{-LWL}.$$

Further, the following theorem yields that increasing the parameter s results in higher expressivity. Formally, we show that the (k, k)-LWL is strictly more expressive than the $(k, 2)$-LWL.

Theorem 3. For $k \geq 2$, it holds that

$$(k, k)\text{-LWL} \sqsubset (k, 2)\text{-LWL}.$$

See Appendix C.1 for an analysis of the asymptotic running time of the (k, s)-LWL, showing that it only depends on k, s, and the sparsity of the graph. In particular, the running time of the (k, s)-LWL on an n-vertex graph of bounded degree is $\tilde{\mathcal{O}}(n^s)$ instead of the usual $\tilde{\mathcal{O}}(n^k)$ for the k-WL, for fixed k and s.

4. SpeqNets

We can now leverage the above combinatorial insights to derive sparsity-aware, permutation-equivariant graph networks, denoted (k, s)-SpeqNets. Given a node-labeled

graph G, let each (k,s)-tuple $\mathbf{v}$ in $V(G)^k_s$ be annotated with an initial feature $f^{(0)}(\mathbf{v})$ determined by its (labeled) isomorphism type, e.g., a one-hot encoding. Alternatively, we can also use some application-specific, real-valued feature. In each layer $t > 0$, we compute a new feature $f^{(t)}(\mathbf{v})$ as

$$f^{W_1}_{\text{mrg}}\Big(f^{(t-1)}(\mathbf{v}), f^{W_2}_{\text{agg}}\Big(\big(\{\!\{f^{(t-1)}(\phi_i(\mathbf{v},w)) \mid w \in \delta(v_i) \text{ and } \phi_i(\mathbf{v},w) \in V(G)^k_s\}\!\}\big)_{i\in[k]}\Big)\Big), \tag{1}$$

in $\mathbb{R}^{1\times e}$, where $W^{(t)}_1$ and $W^{(t)}_2$ are learnable parameter matrices from $\mathbb{R}^{d\times e}$.[4] Here, $f^{W_2}_{\text{mrg}}$ and $f^{W_1}_{\text{agg}}$ are arbitrary differentiable functions, responsible for merging and aggregating the relevant feature information, respectively. Note that we can naturally handle discrete node and edge labels as well as directed graphs. The following result demonstrates the expressive power of (k,s)-SpeqNets, in terms of distinguishing non-isomorphic graphs.

Theorem 4. Let (V,E,ℓ) be a labeled graph, and let $k \geq 1$ and s in $[k]$. Then, for all $t \geq 0$, there exists a sequence of weights $\mathbf{W}^{(t)}$ such that

$$C^{k,s}_t(\mathbf{v}) = C^{k,s}_t(\mathbf{w}) \iff f^{(t)}(\mathbf{v}) = f^{(t)}(\mathbf{w}).$$

Hence, for all $k \geq 1$, the following holds:

$$(k,s)\text{-SpeqNet} \equiv (k,s)\text{-LWL}.$$

Note that it is not possible to come up with an architecture and weight assignments of $f^{W_1}_{\text{mrg}}$ and $f^{W_2}_{\text{agg}}$, such that it becomes more powerful than the (k,s)-LWL, see (Morris et al., 2019). However, all results from the previous section can be lifted to the neural setting. Analogously to GNNs, the above architecture can naturally handle continuous node and edge labels. By using the tools developed in (Azizian & Lelarge, 2020), it is straightforward to show that the above architecture is universal, i.e., it can approximate any continuous, bounded, permutation-invariant function over graphs up to an arbitrarily small additive error.

4.1. Node-, edge-, and subgraph-level learning tasks

The above architecture computes representations for (k,s)-tuples, making it mostly suitable for graph-level learning tasks, e.g., graph classification or regression. However, it is also possible to derive neural architectures based on the (k,s)-LWL for node- and edge-level learning tasks, e.g., node or link prediction. Given a graph G, to learn a node feature for node v, we can simply pool over the features learned for (k,s)-tuples containing the node v as a component. That is, let $t > 0$, then we consider the multisets

$$m^t(v)_i = \{\!\{f^{(t-1)}(\mathbf{t}) \mid \mathbf{t} \in V(G)^k_s \text{ and } t_i = v\}\!\} \tag{2}$$

for i in $[k]$. Hence, to compute a vectorial representation of the node v, we compute a vectorial representation of $m^t(v)_i$ for i in $[k]$, e.g., using a neural architecture for multi-sets, see (Wagstaff et al., 2021), followed by learning a joint vectorial representation for the node v. Again, by (Azizian & Lelarge, 2020), it is straightforward to show that the above architecture is universal, i.e., it can approximate any continuous, bounded, permutation-equivariant function over graphs up to an arbitrarily small additive error. Note that the above approach can be directly generalized to learn subgraph representations on an arbitrary number of vertices.

5. Experimental evaluation

Here, we aim to empirically investigate the learning performance of the kernel, see Appendix B.1, and neural architectures, see Section 4, based on the (k,s)-LWL, compared with standard kernel and (higher-order) GNN baselines. Concretely, we aim to answer the following questions.

Q1 Do the (k,s)-LWL-based algorithms, both kernel and neural architectures, lead to improved classification and regression scores on real-world, graph-level benchmark datasets compared with dense algorithms and standard baselines?

Q2 How does the (k,s)-SpeqNet architecture compare to standard GNN baselines on node-classification tasks?

Q3 To what extent does the (k,s)-LWL reduce computation times compared with architectures induced by the k-WL?

Q4 What is the effect of k and s with respect to computation times and predictive performance?

The source code of all methods and evaluation procedures is available at `https://www.github.com/chrsmrrs/speqnets`.

Datasets To compare the (k,s)-LWL-based kernels, we used the well-known graph-classification benchmark datasets from (Morris et al., 2020a), see Table 3 for dataset statistics and properties.[5] To compare the (k,s)-SpeqNet architecture with GNN baselines, we used the ALCHEMY (Chen et al., 2019a) and the QM9 (Ramakrishnan et al., 2014; Wu et al., 2018) graph regression datasets, again see Table 1 for dataset statistics and properties. Following (Morris et al., 2020b), we opted for not using the 3D-coordinates of the ALCHEMY dataset to solely show the benefits of the (sparse) higher-order structures concerning graph structure and discrete labels. To investigate the performance of the architecture for node classification,

[4] For clarity of presentation, we omit biases.

[5] All datasets are publicly available at `www.graphlearning.io`.

we used the WEBKB datasets (Pei et al., 2020), see Table 4 for dataset statistics and properties.

Kernels We implemented the (k, s)-LWL and (k, s)-LWL$^+$ for k in $\{2, 3\}$ and s in $\{1, 2\}$. We compared our kernels to the Weisfeiler–Leman subtree kernel (1-WL) (Shervashidze et al., 2011), the Weisfeiler–Leman Optimal Assignment kernel (WLOA) (Kriege et al., 2016), the graphlet kernel (GR) (Shervashidze et al., 2009), and the shortest-path kernel (Borgwardt & Kriegel, 2005) (SP). Further, we implemented the higher-order kernels δ-k-LWL, δ-k-LWL$^+$, δ-k-WL, and k-WL kernel for k in $\{2, 3\}$ as outlined in (Morris et al., 2020b). All kernels were (re-)implemented in C++11. For the graphlet kernel, we counted (labeled) connected subgraphs of size 3. We followed the evaluation guidelines outlined in (Morris et al., 2020a).

Neural architectures We used the GIN-ε and Gin-ε-JK architectures (Xu et al., 2019) as neural baselines. For data with (continuous) edge features, we used a 2-layer MLP to map them to the same number of components as the node features and combined them using summation (GINE and GINE-ε). For the evaluation of the (k, s)-SpeqNet neural architectures of Section 4, we implemented them using PYTORCH GEOMETRIC (Fey & Lenssen, 2019), using a Python-wrapped C++11 preprocessing routine to compute the computational graphs for the higher-order GNNs. We used the GIN-ε layer to express $f^{W_1}_{\text{mrg}}$ and $f^{W_2}_{\text{agg}}$ of Equation (1). For the GNN baseline for the QM9 dataset, following (Gilmer et al., 2017), we used a complete graph, computed pairwise ℓ_2 distances based on the 3D coordinates, and concatenated them to the edge features. We note here that our intent is not the beat state-of-the-art, physical knowledge-incorporating architectures, e.g., DimeNet (J. Klicpera, 2020) or Cormorant (Anderson et al., 2019), but to solely show the benefits of the local, sparse higher-order architectures compared to the corresponding (1-dimensional) GNN. For the (k, s)-SpeqNet architectures, in the case of the QM9 dataset, to compute the initial features, for each (k, s)-tuple, we concatenated the node and edge features, computed pairwise ℓ_2 distances based on the 3D coordinates, and a one-hot encoding of the (labeled) isomorphism type. Finally, we used a 2-layer MLP to learn a joint, initial vectorial representation. For the node-classification experiments, we used mean pooling to implement Equation (2) and a standard GCN or GIN layer for all experiments, including the (k, s)-SpeqNet architectures. Further, we used the architectures (SDRF) outlined in (Topping et al., 2021) as baselines.

For the kernel experiments, we computed the (cosine) normalized Gram matrix for each kernel. We computed the classification accuracies using the C-SVM implementation of LIBSVM (Chang & Lin, 2011), using 10-fold cross-validation. We repeated each 10-fold cross-validation ten times with different random folds and report average accuracies and standard deviations.

Following the evaluation method proposed in (Morris et al., 2020a), the C-parameter was selected from $\{10^{-3}, 10^{-2}, \ldots, 10^{2}, 10^{3}\}$ using a validation set sampled uniformly at random from the training fold (using 10% of the training fold). Similarly, the numbers of iterations of the (k, s)-LWL, (k, s)-LWL$^+$, 1-WL, WLOA, δ-k-LWL, δ-k-LWL$^+$, and k-WL were selected from $\{0, \ldots, 5\}$ using the validation set. Moreover, for the (k, s)-LWL$^+$ and δ-k-LWL$^+$, we only added the label function $\#$ on the last iteration to prevent overfitting. We report computation times for the (k, s)-LWL, (k, s)-LWL$^+$, WLOA, δ-k-LWL, δ-k-LWL$^+$, and k-WL with five refinement steps.

All kernel experiments were conducted on a workstation with 791GB of RAM using a single core. Moreover, we used the GNU C++ Compiler 4.8.5 with the flag `-O2`.

For comparing the kernel approaches to GNN baselines, we used 10-fold cross-validation and again used the approach outlined in (Morris et al., 2020a). The number of components of the (hidden) node features in $\{32, 64, 128\}$ and the number of layers in $\{1, 2, 3, 4, 5\}$ of the GIN and GIN-ε layer were again selected using a validation set sampled uniformly at random from the training fold (using 10% of the training fold). We used mean pooling to pool the learned node embeddings to a graph embedding and used a 2-layer MLP for the final classification, using a dropout layer with $p = 0.5$ after the first layer of the MLP. We repeated each 10-fold cross-validation ten times with different random folds and report the average accuracies and standard deviations. Due to the different training methods, we do not provide computation times for the GNN baselines.

For the larger molecular regression tasks ALCHEMY and QM9, we closely followed the hyperparameters found in (Chen et al., 2019a) and (Gilmer et al., 2017), respectively, for the GINE-ε layers. That is, we used six layers with 64 (hidden) node features and a set2seq layer (Vinyals et al., 2016) for graph-level pooling, followed by a 2-layer MLP for the joint regression of the twelve targets. We used the same architecture details and hyperparameters for the (k, s)-SpeqNet. For the ALCHEMY, we used the subset of 12 000 graphs from (Morris et al., 2020b). For both datasets, we uniformly and at random sampled 80% of the graphs for training, and 10% for validation and testing, respectively. Moreover, following (Chen et al., 2019a; Gilmer et al., 2017), we normalized the targets of the training split to zero mean and unit variance. We used a single model to predict all targets. Following (J. Klicpera, 2020, Appendix C), we report mean standardized MAE and mean standardized logMAE. We repeated each experiment five times and report average scores and standard deviations. We used

Table 1. Classification accuracies in percent and standard deviations, OOT— Computation did not finish within one day, OOM— Out of memory.

	Method	Dataset							
		ENZYMES	IMDB-BINARY	IMDB-MULTI	MUTAG	NCI1	PROTEINS	PTC_MR	REDDIT-BINARY
Baseline	GR	29.9 ±0.8	59.3 ±0.9	39.2 ±0.6	72.5 ±1.7	66.2 ±0.2	71.5 ±0.5	56.6 ±1.3	59.7 ±0.5
	SP	40.3 ±0.9	58.7 ±0.6	39.7 ±0.3	81.7 ±1.5	74.1 ±0.2	75.8 ±0.7	59.6 ±1.5	84.5 ±0.2
	1-WL	50.6 ±1.2	72.5 ±0.8	50.0 ±0.8	75.9 ±2.0	84.4 ±0.3	73.1 ±0.6	59.3 ±2.1	73.4 ±0.9
	WLOA	57.1 ±0.8	73.2 ±0.4	49.8 ±0.4	83.4 ±1.2	85.2 ±0.2	73.0 ±0.9	60.3 ±1.9	88.3 ±0.4
GNN	Gin-ε	38.7 ±1.5	72.9 ±0.7	49.7 ±0.7	84.1 ±1.4	77.7 ±0.8	72.2 ±0.6	55.2 ±1.7	89.8 ±0.4
	Gin-ε-JK	39.3 ±1.6	73.0 ±1.1	49.6 ±0.7	83.4 ±2.0	78.3 ±0.3	72.2 ±0.7	56.0 1.3±	90.4 ±0.4
k-WL	2-WL	37.0 ±1.0	68.1 ±1.7	47.5 ±0.7	85.7 ±1.6	66.9 ±0.3	75.2 ±0.4	60.5 ±1.1	OOM
	3-WL	42.3 ±1.1	67.1 ±1.5	46.8 ±0.8	85.4 ±1.5	OOT	OOT	59.0 ±2.0	OOM
	δ-2-LWL	55.9 ±1.0	73.0 ±0.7	50.1 ±0.9	85.6 ±1.4	84.6 ±0.3	75.1 ±0.5	61.7 ±2.4	89.4 ±0.6
	δ-2-LWL$^+$	53.9 ±1.4	75.6 ±1.0	62.7 ±1.4	84.1 ±2.1	**91.3** ±0.3	79.2 ±1.2	61.6 ±1.3	91.4 ±0.4
	δ-3-LWL	**58.2** ±1.2	72.6 ±0.9	49.0 ±1.2	84.1 ±1.6	83.2 ±0.4	OOM	60.7 ±2.2	OOM
	δ-3-LWL$^+$	56.5 ±1.4	76.1 ±1.2	64.3 ±1.2	85.4 ±1.8	82.7 ±0.4	OOM	61.5 ±1.8	OOM
(k,s)-LWL	$(2,1)$-LWL	53.7 ±1.7	73.5 ±0.8	50.8 ±0.7	84.2 ±1.7	82.8 ±0.3	73.2 ±0.6	55.9 ±2.4	76.9 ±0.6
	$(2,1)$-LWL$^+$	51.6 ±1.8	73.7 ±1.1	55.4 ±0.9	79.6 ±3.4	81.9 ±0.3	76.0 ±0.9	60.2 ±2.1	**94.7** ±0.3
	$(3,1)$-LWL	53.4 ±1.4	74.6 ±1.0	51.3 ±0.6	85.3 ±2.4	81.4 ±0.5	72.9 ±1.1	60.2 ±1.7	OOM
	$(3,1)$-LWL$^+$	57.0 ±1.9	**87.1** ±0.6	**67.1** ±1.1	79.2 ±1.5	89.8 ±0.4	**81.2** ±0.8	59.2 ±2.0	OOM
	$(3,2)$-LWL	56.4 ±0.7	73.5 ±0.5	49.7 ±0.6	**86.4** ±2.6	84.9 ±0.4	75.1 ±0.9	61.9 ±2.4	OOM
	$(3,2)$-LWL$^+$	55.8 ±1.7	78.1 ±1.4	59.5 ±1.0	84.5 ±1.9	89.4 ±0.3	78.8 ±0.6	**62.3** ±3.3	OOM

the provided ten training, validation, and test splits for the node-classification datasets. All neural experiments were conducted on a workstation with one GPU card with 32GB of GPU memory.

To compare training and testing times between the $(2,1)$-SpeqNet, $(2,2)$-SpeqNet, GINE-ε architectures, we trained all three models on ALCHEMY (10K) and QM9 to convergence, divided by the number of epochs, and calculated the ratio with respect to the average epoch computation time of the $(2,1)$-SpeqNet (average computation time of dense or baseline layer divided by average computation time of the $(2,1)$-SpeqNet). Contrary to the kernel timing experiments, we did not take into account the time of the preprocessing routine to compute the computational graphs to focus purely on the neural component of the architecture. Clearly, the time for the preprocessing of (k,s)-SpeqNet with small s is much smaller than that of, e.g., the δ-k-WL.

5.1. Results and discussion

In the following, we answer questions **Q1** to **Q4**.

A1 *Kernels* See Table 1. The (k,s)-LWL for k,s in $\{2,3\}$ significantly improves the classification accuracy compared to the k-WL and the δ-k-WL, and the other kernel baselines, while being on par with or better than the δ-2-LWL and δ-3-LWL. The (k,s)-LWL and (k,s)-LWL$^+$ achieve a new state-of-the-art on five out of eight datasets. Our algorithms also perform vastly better than the neural baselines.

Neural architectures See Table 2. On both datasets, all (k,s)-SpeqNet architectures beat the GNN baseline. On the ALCHEMY dataset, the $(2,2)$-SpeqNet and $(3,1)$-SpeqNet perform best, while on the QM9 dataset, the $(2,2)$-SpeqNet performs best by a large margin.

A2 See Table 2. Over all three datasets, the (k,s)-SpeqNet architectures improve over the GCN baseline. Specifically, over all datasets, the $(2,1)$-SpeqNet and the $(2,2)$-SpeqNet lead to an increase of at least 7% in accuracy. For example, both architectures beat the GCN and GIN baseline by at least 17% on the WISCONSIN dataset. Further, the (k,s)-SpeqNet architectures lead to better accuracies compared to the SDRF architecture, e.g., improving on it by more than 10% on the CORNELL dataset. Hence, node-level tasks also benefit from higher-order information. However, increasing k more does not result in increased accuracies.

A3 *Kernels* See Table 5. Clearly, for the same k and $s < k$, the (k,s)-LWL improves over the k-WL and its (local) variants. For example, on the ENZYMES dataset, the $(2,1)$-LWL is more than 20 times faster in terms of computation times compared to the δ-2-LWL. The speed up is even more significant for the non-local 2-WL. This speed-up factor increases more as k increases, e.g., the $(3,1)$-LWL is more than 1 700 times faster compared to the 3-WL, whereas the $(3,2)$-LWL is still more than 87 times faster, while giving better accuracies. Similar speed-up factors can be observed over all datasets.

Neural architectures See Table 6. The $(2,1)$-SpeqNet

Method	Dataset	
	ALCHEMY (10K)	QM9
GINE-ε	0.180 $\pm$0.006 -1.958 $\pm$0.047	0.079 $\pm$0.003 -3.430 $\pm$0.080
$(2,1)$-SpeqNet	0.169 $\pm$0.005 -2.010 $\pm$0.056	0.078 $\pm$0.007 -2.947 $\pm$0.171
$(2,2)$-SpeqNet	**0.115** $\pm$0.001 -2.722 $\pm$0.054	**0.029** $\pm$0.001 -4.081 $\pm$0.058
$(3,1)$-SpeqNet	0.180 $\pm$0.011 -1.914 $\pm$0.097	0.068 $\pm$0.003 -3.397 $\pm$0.086
$(3,2)$-SpeqNet	**0.115** $\pm$0.002 -2.767 $\pm$0.079	OOT

(a) Mean MAE (mean std. MAE, logMAE) on large-scale (multi-target) molecular regression tasks.

Method	Dataset		
	CORNELL	TEXAS	WISCONSIN
GCN	56.5 $\pm$0.9	58.2 $\pm$0.8	50.9 $\pm$0.7
GIN	51.9 $\pm$1.1	55.3 $\pm$2.7	48.4 $\pm$1.6
SDRF + Undirected	57.5 $\pm$0.3	**70.4** $\pm$0.6	61.6 $\pm$0.9
$(2,1)$-SpeqNet	63.9 $\pm$1.7	66.8 $\pm$0.9	67.7 $\pm$2.2
$(2,2)$-SpeqNet	**67.9** $\pm$1.7	67.3 $\pm$2.0	**68.4** $\pm$2.2
$(3,1)$-SpeqNet	61.8 $\pm$3.3	68.3 $\pm$1.3	60.4 $\pm$2.8

(b) Classification accuracies and standard deviations for node classification.

Table 2. Additional experimental results for graph regression and node classification.

severely speeds up the computation time across both datasets. Specifically, on the ALCHEMY dataset, the $(2,1)$-SpeqNet is 1.3 times faster compared to the $(2,2)$-SpeqNet, while requiring twice the computation time of the GINE-ε but achieving a lower MAE. More interestingly, on the QM9 dataset, the $(2,1)$-SpeqNet is 3.4 times faster compared to the $(2,2)$-SpeqNet, while also being 1.3 times faster compared to the GINE-ε. The speed-up over GINE-ε is most likely due to the latter considering the complete graph to compute all pairwise ℓ_2 distances, whereas the $(2,1)$-SpeqNet only considers connected node pairs.

A4 See Tables 1 and 2. The $(2,1)$-LWL, $(2,1)$-LWL$^+$, and $(3,1)$-LWL$^+$ beat the 1-WL on six out of eight datasets. Going from the $(3,1)$-LWL to the $(3,2)$-LWL often leads to a slight increase in accuracy, e.g., on the ENZYMES and MUTAG datasets, while sometimes leading to a drop in accuracy, e.g., on the IMDB-BINARY dataset. Hence, a better understanding of the model's generalization performance with respect to s needs to be investigated in future work. The effect is less pronounced for the neural architectures; however, all higher-order models beat the GNN baseline. Reducing s leads to a vast reduction in computation time. For example, on the ENZYMES dataset, going from the $(3,2)$-LWL to the $(3,1)$-LWL leads to a speed-up factor of more than 18, while only inducing a small drop in terms of accuracy, whereas the $(3,1)$-LWL$^+$ beats the $(3,2)$-LWL while only increasing the computation time by one second. Similar observations can be made across all datasets.

Increasing s often leads to a better performance on the graph regression tasks. For example, on the ALCHEMY dataset, going from a $(2,1)$-SpeqNet to a $(2,2)$-SpeqNet architecture reduces the MAE by 0.054. Similar effects can be observed for the QM9 dataset, and when going from a $(3,1)$-SpeqNet to a $(3,2)$-SpeqNet architecture. On the node-classification datasets, reducing s leads to a slight drop in accuracy, between 0.5 and 4%, while increasing k beyond 2 often results in a drop in accuracy.

6. Conclusion

To circumvent the exponential running time requirements of k-WL, we introduced a new heuristic for the graph isomorphism problem, namely the (k,s)-LWL. By varying the parameters k and s, the (k,s)-LWL offers a tradeoff between scalability and expressivity and, unlike the k-WL, takes into account the potential sparsity of the graph. Based on these combinatorial insights, we designed provably expressive machine-learning architectures, (k,s)-SpeqNets, suitable for node-, subgraph-, and graph-level prediction tasks. Empirically, we showed that such architectures lead to state-of-the-art results in node- and graph-level classification regimes while obtaining promising results on graph-level regression tasks. We believe that this principled approach paves the way for designing new permutation-equivariant architectures to overcome the limitation of current graph neural networks.

Acknowledgements

Christopher Morris is in part supported by the German Academic Exchange Service (DAAD) through a DAAD IFI postdoctoral scholarship (57515245), a RWTH Junior Principal Investigator Fellowship, and a DFG Emmy Noether grant (468502433). Gaurav Rattan is supported by the DFG Research Grants Program–RA 3242/1-1–411032549. Siamak Ravanbakhsh's research is in part supported by CIFAR AI Chairs program. Computational resources were provided by Mila.

References

Abboud, R., Ceylan, İ. İ., Grohe, M., and Lukasiewicz, T. The surprising power of graph neural networks with random node initialization. *CoRR*, abs/2010.01179, 2020.

Anderson, B. M., Hy, T., and Kondor, R. Cormorant: Covariant molecular neural networks. In *Advances in Neural Information Processing Systems 32*, pp. 14510–14519,

2019.

Arvind, V., Köbler, J., Rattan, G., and Verbitsky, O. On the power of color refinement. In *International Symposium on Fundamentals of Computation Theory*, pp. 339–350, 2015.

Arvind, V., Fuhlbrück, F., Köbler, J., and Verbitsky, O. On Weisfeiler-Leman invariance: Subgraph counts and related graph properties. In *International Symposium on Fundamentals of Computation Theory*, pp. 111–125, 2019.

Atserias, A. and Maneva, E. N. Sherali-adams relaxations and indistinguishability in counting logics. *SIAM Journal on Computing*, 42(1):112–137, 2013.

Atserias, A., Mancinska, L., Roberson, D. E., Sámal, R., Severini, S., and Varvitsiotis, A. Quantum and non-signalling graph isomorphisms. *Journal of Combinatorial Theory, Series B*, 136:289–328, 2019.

Azizian, W. and Lelarge, M. Characterizing the expressive power of invariant and equivariant graph neural networks. *CoRR*, abs/2006.15646, 2020.

Babai, L. Lectures on graph isomorphism. University of Toronto, Department of Computer Science. Mimeographed lecture notes, October 1979, 1979.

Babai, L. Graph isomorphism in quasipolynomial time. In *ACM SIGACT Symposium on Theory of Computing*, pp. 684–697, 2016.

Balcilar, M., Héroux, P., Gaüzère, B., Vasseur, P., Adam, S., and Honeine, P. Breaking the limits of message passing graph neural networks. In *International Conference on Machine Learning*, pp. 599–608, 2021.

Barabasi, A.-L. and Oltvai, Z. N. Network biology: Understanding the cell's functional organization. *Nature Reviews Genetics*, 5(2):101–113, 2004.

Barceló, P., Kostylev, E. V., Monet, M., Pérez, J., Reutter, J. L., and Silva, J. P. The logical expressiveness of graph neural networks. In *International Conference on Learning Representations*, 2020.

Barceló, P., Geerts, F., Reutter, J. L., and Ryschkov, M. Graph neural networks with local graph parameters. *CoRR*, abs/2106.06707, 2021.

Baskin, I. I., Palyulin, V. A., and Zefirov, N. S. A neural device for searching direct correlations between structures and properties of chemical compounds. *Journal of Chemical Information and Computer Sciences*, 37(4): 715–721, 1997.

Beaini, D., Passaro, S., Létourneau, V., Hamilton, W. L., Corso, G., and Liò, P. Directional graph networks. *CoRR*, abs/2010.02863, 2020.

Berkholz, C., Bonsma, P. S., and Grohe, M. Tight lower and upper bounds for the complexity of canonical colour refinement. *Theory of Computing Systems*, 60(4):581–614, 2017.

Bevilacqua, B., Frasca, F., Lim, D., Srinivasan, B., Cai, C., Balamurugan, G., Bronstein, M. M., and Maron, H. Equivariant subgraph aggregation networks. *CoRR*, abs/2110.02910, 2021.

Bodnar, C., Frasca, F., Otter, N., Wang, Y. G., Liò, P., Montúfar, G., and Bronstein, M. M. Weisfeiler and Lehman go cellular: CW networks. *CoRR*, abs/2106.12575, 2021a. URL https://arxiv.org/abs/2106.12575.

Bodnar, C., Frasca, F., Wang, Y., Otter, N., Montufar, G. F., Lió, P., and Bronstein, M. Weisfeiler and Lehman go topological: Message passing simplicial networks. In *International Conference on Machine Learning*, pp. 1026–1037, 2021b.

Borgwardt, K. M. and Kriegel, H.-P. Shortest-path kernels on graphs. In *IEEE International Conference on Data Mining*, pp. 74–81, 2005.

Borgwardt, K. M., Ghisu, M. E., Llinares-López, F., O'Bray, L., and Rieck, B. Graph kernels: State-of-the-art and future challenges. *Foundations and Trends in Machine Learning*, 13(5-6), 2020.

Bouritsas, G., Frasca, F., Zafeiriou, S., and Bronstein, M. M. Improving graph neural network expressivity via subgraph isomorphism counting. *CoRR*, abs/2006.09252, 2020.

Bruna, J., Zaremba, W., Szlam, A., and LeCun, Y. Spectral networks and deep locally connected networks on graphs. In *International Conference on Learning Representation*, 2014.

Busacker, R. G. and Saaty, T. L. *Finite graphs and networks: an introduction with applications*. McGraw-Hill, 1965.

Cai, J., Fürer, M., and Immerman, N. An optimal lower bound on the number of variables for graph identifications. *Combinatorica*, 12(4):389–410, 1992.

Chami, I., Abu-El-Haija, S., Perozzi, B., Ré, C., and Murphy, K. Machine learning on graphs: A model and comprehensive taxonomy. *CoRR*, abs/2005.03675, 2020.

Chang, C.-C. and Lin, C.-J. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. ACM.

Chen, G., Chen, P., Hsieh, C., Lee, C., Liao, B., Liao, R., Liu, W., Qiu, J., Sun, Q., Tang, J., Zemel, R. S., and Zhang, S. Alchemy: A quantum chemistry dataset for benchmarking AI models. *CoRR*, abs/1906.09427, 2019a.

Chen, Z., Villar, S., Chen, L., and Bruna, J. On the equivalence between graph isomorphism testing and function approximation with GNNs. In *Advances in Neural Information Processing Systems*, pp. 15868–15876, 2019b.

Chen, Z., Chen, L., Villar, S., and Bruna, J. Can graph neural networks count substructures? *CoRR*, abs/2002.04025, 2020.

Cotta, L., Morris, C., and Ribeiro, B. Reconstruction for powerful graph representations. In *Advances in Neural Information Processing Systems*, 2021.

Dasoulas, G., Santos, L. D., Scaman, K., and Virmaux, A. Coloring graph neural networks for node disambiguation. In *International Joint Conference on Artificial Intelligence*, pp. 2126–2132, 2020.

Defferrard, M., X., B., and Vandergheynst, P. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems*, pp. 3844–3852, 2016.

Dell, H., Grohe, M., and Rattan, G. Lovász meets Weisfeiler and Leman. In *International Colloquium on Automata, Languages, and Programming*, pp. 40:1–40:14, 2018.

Duvenaud, D. K., Maclaurin, D., Iparraguirre, J., Bombarell, R., Hirzel, T., Aspuru-Guzik, A., and Adams, R. P. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in Neural Information Processing Systems*, pp. 2224–2232, 2015.

Easley, D. and Kleinberg, J. *Networks, Crowds, and Markets: Reasoning About a Highly Connected World*. Cambridge University Press, 2010.

Fey, M. and Lenssen, J. E. Fast graph representation learning with PyTorch Geometric. In *International Conference on Learning Representations, Workshop on Representation Learning on Graphs and Manifolds*, 2019.

Fürer, M. On the combinatorial power of the Weisfeiler-Lehman algorithm. In *International Conference on Algorithms and Complexity*, pp. 260–271, 2017.

Gärtner, T., Flach, P., and Wrobel, S. On graph kernels: Hardness results and efficient alternatives. In *Learning Theory and Kernel Machines*, pp. 129–143. Springer, 2003.

Geerts, F. The expressive power of kth-order invariant graph networks. *CoRR*, abs/2007.12035, 2020.

Geerts, F., Mazowiecki, F., and Pérez, G. A. Let's agree to degree: Comparing graph convolutional networks in the message-passing framework. *CoRR*, abs/2004.02593, 2020.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. Neural message passing for quantum chemistry. In *International Conference on Machine Learning*, 2017.

Grohe, M. *Descriptive Complexity, Canonisation, and Definable Graph Structure Theory*. Cambridge University Press, 2017.

Grohe, M. Word2vec, Node2vec, Graph2vec, X2vec: Towards a theory of vector embeddings of structured data. *CoRR*, abs/2003.12590, 2020.

Grohe, M. The logic of graph neural networks. In *ACM/IEEE Symposium on Logic in Computer Science*, pp. 1–17, 2021.

Grohe, M. and Otto, M. Pebble games and linear equations. *Journal of Symbolic Logic*, 80(3):797–844, 2015.

Grohe, M., Schweitzer, P., and D, W. Deep Weisfeiler Leman. *CoRR*, abs/2003.10935, 2020.

Hamilton, W. L., Ying, R., and Leskovec, J. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pp. 1025–1035, 2017.

Heimann, M., Safavi, T., and Koutra, D. Distribution of node embeddings as multiresolution features for graphs. In *IEEE International Conference on Data Mining*, pp. 289–298, 2019.

Immerman, N. and Lander, E. Describing graphs: A first-order approach to graph canonization. In *Complexity Theory Retrospective: In Honor of Juris Hartmanis on the Occasion of His Sixtieth Birthday, July 5, 1988*, pp. 59–81, 1990.

J. Klicpera, J. Groß, S. G. Directional message passing for molecular graphs. In *International Conference on Learning Representations*, 2020.

Johansson, F. D. and Dubhashi, D. Learning with similarity functions on graphs using matchings of geometric embeddings. In *ACM Knowledge and Data Knowledge Discovery Conference*, pp. 467–476, 2015.

Jumper, J., Evans, R., Pritzel, A., Green, T., Figurnov, M., Ronneberger, O., Tunyasuvunakool, K., Bates, R., Žídek, A., Potapenko, A., Bridgland, A., Meyer, C., Kohl, S. A. A., Ballard, A. J., Cowie, A., Romera-Paredes, B., Nikolov, S., Jain, R., Adler, J., Back, T., Petersen, S.,

Reiman, D., Clancy, E., Zielinski, M., Steinegger, M., Pacholska, M., Berghammer, T., Bodenstein, S., Silver, D., Vinyals, O., Senior, A. W., Kavukcuoglu, K., Kohli, P., and Hassabis, D. Highly accurate protein structure prediction with AlphaFold. *Nature*, 2021.

Kashima, H., Tsuda, K., and Inokuchi, A. Marginalized kernels between labeled graphs. In *International Conference on Machine Learning*, pp. 321–328, 2003.

Kiefer, S. *Power and Limits of the Weisfeiler-Leman Algorithm*. PhD thesis, Department of Computer Science, RWTH Aachen University, 2020a.

Kiefer, S. The Weisfeiler-Leman algorithm: an exploration of its power. *ACM SIGLOG News*, 7(3):5–27, 2020b.

Kiefer, S. and McKay, B. D. The iteration number of Colour Refinement. In *International Colloquium on Automata, Languages, and Programming*, pp. 73:1–73:19, 2020.

Kiefer, S. and Schweitzer, P. Upper bounds on the quantifier depth for graph differentiation in first order logic. In *ACM/IEEE Symposium on Logic in Computer Science*, pp. 287–296, 2016.

Kiefer, S., Schweitzer, P., and Selman, E. Graphs identified by logics with counting. In *International Symposium on Mathematical Foundations of Computer Science*, pp. 319–330, 2015.

Kiefer, S., Ponomarenko, I., and Schweitzer, P. The Weisfeiler-Leman dimension of planar graphs is at most 3. *Journal of the ACM*, 66(6):44:1–44:31, 2019.

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017.

Kireev, D. B. Chemnet: A novel neural network based method for graph/property mapping. *Journal of Chemical Information and Computer Sciences*, 35(2):175–180, 1995. ACS.

Kondor, R. and Pan, H. The multiscale Laplacian graph kernel. In *Advances in Neural Information Processing Systems*, pp. 2982–2990, 2016.

Kriege, N. M., Giscard, P.-L., and Wilson, R. C. On valid optimal assignment kernels and applications to graph classification. In *Advances in Neural Information Processing Systems*, pp. 1615–1623, 2016.

Kriege, N. M., Neumann, M., Morris, C., Kersting, K., and Mutzel, P. A unifying view of explicit and implicit feature maps for structured data: Systematic studies of graph kernels. *CoRR*, abs/1703.00676, 2017. Accepted for publication in *Data Mining and Knowledge Discovery*.

Kriege, N. M., Morris, C., Rey, A., and Sohler, C. A property testing framework for the theoretical expressivity of graph kernels. In *International Joint Conference on Artificial Intelligence*, pp. 2348–2354, 2018.

Kriege, N. M., Johansson, F. D., and Morris, C. A survey on graph kernels. *Applied Network Science*, 5(1):6, 2020.

Li, P., Wang, Y., Wang, H., and Leskovec, J. Distance encoding: Design provably more powerful neural networks for graph representation learning. *Advances in Neural Information Processing Systems*, 2020.

Lichter, M., Ponomarenko, I., and Schweitzer, P. Walk refinement, walk logic, and the iteration number of the Weisfeiler-Leman algorithm. In *ACM/IEEE Symposium on Logic in Computer Science*, pp. 1–13, 2019.

Maehara, T. and NT, H. A simple proof of the universality of invariant/equivariant graph neural networks. *CoRR*, abs/1910.03802, 2019.

Malkin, P. N. Sherali–adams relaxations of graph isomorphism polytopes. *Discrete Optimization*, 12:73–97, 2014.

Maron, H., Ben-Hamu, H., Serviansky, H., and Lipman, Y. Provably powerful graph networks. *CoRR*, abs/1905.11136, 2019a.

Maron, H., Fetaya, E., Segol, N., and Lipman, Y. On the universality of invariant networks. In *International Conference on Machine Learning*, pp. 4363–4371, 2019b.

Merkwirth, C. and Lengauer, T. Automatic generation of complementary descriptors with molecular graph networks. *Journal of Chemical Information and Modeling*, 45(5):1159–1168, 2005.

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

Micheli, A. and Sestito, A. S. A new neural network model for contextual processing of graphs. In *Italian Workshop on Neural Nets Neural Nets and International Workshop on Natural and Artificial Immune Systems*, volume 3931 of *Lecture Notes in Computer Science*, pp. 10–17. Springer, 2005.

Monti, F., Boscaini, D., Masci, J., Rodolà, E., Svoboda, J., and Bronstein, M. M. Geometric deep learning on graphs and manifolds using mixture model CNNs. In *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5425–5434, 2017.

Morris, C., Kersting, K., and Mutzel, P. Glocalized Weisfeiler-Lehman kernels: Global-local feature maps of graphs. In *IEEE International Conference on Data Mining*, pp. 327–336, 2017.

Morris, C., Ritzert, M., Fey, M., Hamilton, W. L., Lenssen, J. E., Rattan, G., and Grohe, M. Weisfeiler and leman go neural: Higher-order graph neural networks. In *AAAI Conference on Artificial Intelligence*, pp. 4602–4609, 2019.

Morris, C., Kriege, N. M., Bause, F., Kersting, K., Mutzel, P., and Neumann, M. TUDataset: A collection of benchmark datasets for learning with graphs. *CoRR*, abs/2007.08663, 2020a.

Morris, C., Rattan, G., and Mutzel, P. Weisfeiler and Leman go sparse: Towards higher-order graph embeddings. In *Advances in Neural Information Processing Systems*, 2020b.

Morris, C., L., Y., Maron, H., Rieck, B., Kriege, N. M., Grohe, M., Fey, M., and Borgwardt, K. Weisfeiler and Leman go machine learning: The story so far. *CoRR*, abs/2112.09992, 2021.

Murphy, R. L., Srinivasan, B., Rao, V. A., and Ribeiro, B. Relational pooling for graph representations. In *International Conference on Machine Learning*, volume 97, pp. 4663–4673, 2019.

Nikolentzos, G., Meladianos, P., and Vazirgiannis, M. Matching node embeddings for graph similarity. In *AAAI Conference on Artificial Intelligence*, pp. 2429–2435, 2017.

Nikolentzos, G., Meladianos, P., Limnios, S., and Vazirgiannis, M. A degeneracy framework for graph similarity. In *International Joint Conference on Artificial Intelligence*, pp. 2595–2601, 2018.

NT, H. and Maehara, T. Graph homomorphism convolution. *CoRR*, abs/2005.01214, 2020.

Papp, P. A., K. Martinkus, L. F., and Wattenhofer, R. DropGNN: Random dropouts increase the expressiveness of graph neural networks. In *Advances in Neural Information Processing Systems*, 2021.

Pei, H., Wei, B., Chang, K. C., Lei, Y., and Yang, B. Geom-GCN: Geometric graph convolutional networks. In *International Conference on Learning Representations*, 2020.

Ramakrishnan, R., Dral, P., O., Rupp, M., and von Lilienfeld, O. A. Quantum chemistry structures and properties of 134 kilo molecules. *Scientific Data*, 1, 2014. Nature.

Rieck, B., Bock, C., and Borgwardt, K. M. A persistent Weisfeiler-Lehman procedure for graph classification. In *International Conference on Machine Learning*, pp. 5448–5458, 2019.

Sato, R., Yamada, M., and Kashima, H. Random features strengthen graph neural networks. *CoRR*, abs/2002.03155, 2020.

Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., and Monfardini, G. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

Shervashidze, N., Vishwanathan, S. V. N., Petri, T. H., Mehlhorn, K., and Borgwardt, K. M. Efficient graphlet kernels for large graph comparison. In *International Conference on Artificial Intelligence and Statistics*, pp. 488–495, 2009.

Shervashidze, N., Schweitzer, P., van Leeuwen, E. J., Mehlhorn, K., and Borgwardt, K. M. Weisfeiler-Lehman graph kernels. *Journal of Machine Learning Research*, 12:2539–2561, 2011.

Simonovsky, M. and Komodakis, N. Dynamic edge-conditioned filters in convolutional neural networks on graphs. In *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 29–38, 2017.

Sperduti, A. and Starita, A. Supervised neural networks for the classification of structures. *IEEE Transactions on Neural Networks*, 8(2):714–35, 1997. IEEE.

Stokes, J., Yang, K., Swanson, K., Jin, W., Cubillos-Ruiz, A., Donghia, N., MacNair, C., French, S., Carfrae, L., Bloom-Ackerman, Z., Tran, V., Chiappino-Pepe, A., Badran, A., Andrews, I., Chory, E., Church, G., Brown, E., Jaakkola, T., Barzilay, R., and Collins, J. A deep learning approach to antibiotic discovery. *Cell*, 180:688–702.e13, 02 2020.

Talak, R., Hu, S., Peng, L., and Carlone, L. Neural trees for learning on graphs. *CoRR*, abs/2105.07264, 2021.

Thiede, E. H., Zhou, W., and Kondor, R. Autobahn: Automorphism-based graph neural nets. *CoRR*, abs/2103.01710, 2021.

Togninalli, M., Ghisu, M. E., Llinares-López, F., Rieck, B., and Borgwardt, K. M. Wasserstein Weisfeiler-Lehman graph kernels. In *Advances in Neural Information Processing Systems*, pp. 6436–6446, 2019.

Tönshoff, J., Ritzert, M., Wolf, H., and Grohe, M. Graph learning with 1D convolutions on random walks. *CoRR*, abs/2102.08786, 2021.

Topping, J., Giovanni, F. D., Chamberlain, B. P., Dong, X., and Bronstein, M. M. Understanding over-squashing and bottlenecks on graphs via curvature. *CoRR*, abs/2111.14522, 2021.

Valiente, G. *Algorithms on Trees and Graphs*. Springer, 2002.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., L. P., and Bengio, Y. Graph attention networks. In *International Conference on Learning Representations*, 2018.

Verma, S. and Zhang, Z. Hunt for the unique, stable, sparse and fast feature learning on graphs. In *Advances in Neural Information Processing Systems 30*, pp. 88–98, 2017.

Vignac, C., Loukas, A., and Frossard, P. Building powerful and equivariant graph neural networks with structural message-passing. In *Advances in Neural Information Processing Systems*, 2020.

Vinyals, O., Bengio, S., and Kudlur, M. Order matters: Sequence to sequence for sets. In *International Conference on Learning Representations*, 2016.

Wagstaff, E., Fuchs, F. B., Engelcke, M., Osborne, M. A., and Posner, I. Universal approximation of functions on sets. *CoRR*, abs/2107.01959, 2021.

Weisfeiler, B. *On Construction and Identification of Graphs*. Lecture Notes in Mathematics, Vol. 558. Springer, 1976.

Weisfeiler, B. and Leman, A. The reduction of a graph to canonical form and the algebra which appears therein. *Nauchno-Technicheskaya Informatsia*, 2 (9):12–16, 1968. English translation by G. Ryabov is available at https://www.iti.zcu.cz/wl2018/pdf/wl_paper_translation.pdf.

Wu, Z., Ramsundar, B., Feinberg, E. N., Gomes, J., Geniesse, C., Pappu, A. S., Leswing, K., and Pande, V. MoleculeNet: A benchmark for molecular machine learning. *Chemical Science*, 9:513–530, 2018.

Xu, K., Hu, W., Leskovec, J., and Jegelka, S. How powerful are graph neural networks? *International Conference on Machine Learning*, 2019.

Yanardag, P. and Vishwanathan, S. V. N. A structural smoothing framework for robust graph comparison. In *Advances in Neural Information Processing Systems*, pp. 2125–2133, 2015a.

Yanardag, P. and Vishwanathan, S. V. N. Deep graph kernels. In *Knowledge Discovery and Data Mining Conference*, pp. 1365–1374, 2015b.

You, J., Gomes-Selman, J., Ying, R., and Leskovec, J. Identity-aware graph neural networks. *CoRR*, abs/2101.10320, 2021.

Zhang, M. and Li, P. Nested graph neural networks. *CoRR*, abs/2110.13197, 2021.

Zhao, L., Jin, W., Akoglu, L., and Shah, N. From stars to subgraphs: Uplifting any GNN with local structure awareness. *CoRR*, abs/2110.03753, 2021.

Appendix

A. Related work (extended)

In the following, we review related work from graph theory.

Theory The Weisfeiler–Leman algorithm constitutes one of the earliest and most natural approaches to isomorphism testing (Weisfeiler, 1976; Weisfeiler & Leman, 1968), having been heavily investigated by the theory community over the last few decades (Grohe, 2017). Moreover, the fundamental nature of the k-WL is evident from a variety of connections to other fields such as logic, optimization, counting complexity, and quantum computing. The power and limitations of k-WL can be neatly characterized in terms of logic and descriptive complexity (Babai, 1979; Immerman & Lander, 1990), Sherali-Adams relaxations of the natural integer linear program for the graph isomorphism problem (Atserias & Maneva, 2013; Grohe & Otto, 2015; Malkin, 2014), homomorphism counts (Dell et al., 2018), and quantum isomorphism games (Atserias et al., 2019). In their seminal paper, Cai et al. (1992) showed that, for each k, there exists a pair of non-isomorphic graphs of size $\mathcal{O}(k)$ that are not distinguished by the k-WL. (Kiefer, 2020a;b) gives a thorough survey of these results. For $k = 1$, the power of the algorithm has been completely characterized (Arvind et al., 2015; Kiefer et al., 2015). Moreover, upper bounds on the running time (Berkholz et al., 2017) and the number of iterations for $k = 1$ (Kiefer & McKay, 2020) and for the folklore $k = 2$ (Kiefer & Schweitzer, 2016; Lichter et al., 2019) have been shown. For k in $\{1, 2\}$, Arvind et al. (2019) studied the abilities of the (folklore) k-WL to detect and count fixed subgraphs, extending the work of Fürer (2017). The former was refined in (Chen et al., 2020). Kiefer et al. (2019) showed that the folklore 3-WL completely captures the structure of planar graphs. The algorithm (for logarithmic k) plays a prominent role in the recent result of (Babai, 2016) improving the best-known running time for the graph isomorphism problem. Recently, Grohe et al. (2020) introduced the framework of Deep Weisfeiler–Leman algorithms, which allow the design of a more powerful graph isomorphism test than Weisfeiler–Leman type algorithms. Finally, the emerging connections between the Weisfeiler–Leman paradigm and graph learning are described in two recent surveys (Grohe, 2020; Morris et al., 2021).

B. Preliminaries

As usual, let $[n] \coloneqq \{1, \dots, n\} \subset \mathbb{N}$ for $n \geq 1$, and we use $\{\!\{\dots\}\!\}$ to denote multisets, i.e., the generalization of sets allowing for multiple instances for each of its elements.

Graphs A *graph* G is a pair $(V(G), E(G))$ with *finite* sets of *nodes* $V(G)$ and *edges* $E(G) \subseteq \{\{u, v\} \subseteq V \mid u \neq v\}$. If not otherwise stated, we set $n \coloneqq |V(G)|$. For ease of notation, we denote the edge $\{u, v\}$ in $E(G)$ by (u, v) or (v, u). In the case of *directed graphs*, $E \subseteq \{(u, v) \in V \times V \mid u \neq v\}$. A *labeled graph* G is a triple (V, E, ℓ) with a label function $\ell \colon V(G) \cup E(G) \to \mathbb{N}$. Then $\ell(v)$ is a *label* of x for x in $V(G) \cup E(G)$. The *neighborhood* of v in $V(G)$ is denoted by $\delta(v) = \{u \in V(G) \mid (v, u) \in E(G)\}$ and the *degree* of a node v is $|\delta(v)|$. Let $S \subseteq V(G)$ then $G[S] = (S, E_S)$ is the *subgraph induced by* S, where $E_S = \{(u, v) \in E(G) \mid u, v \in S\}$. A *connected component* of a graph G is an inclusion-wise maximal subgraph of G in which every two nodes are connected by paths. A *tree* is a connected graph without cycles. A *rooted tree* is an oriented tree with a designated node called *root*, in which the edges are directed away from the root. Let p be a node in a rooted tree. Then we call its out-neighbors *children* with *parent* p. We denote an undirected *cycle* on k nodes by C_k. Given two graphs G and H with disjoint node sets, we denote their disjoint union by $G \dot{\cup} H$.

Two graphs G and H are *isomorphic* and we write $G \simeq H$ if there exists a bijection $\varphi \colon V(G) \to V(H)$ preserving the adjacency relation, i.e., (u, v) is in $E(G)$ if and only if $(\varphi(u), \varphi(v))$ is in $E(H)$. Then φ is an *isomorphism* between G and H. Moreover, we call the equivalence classes induced by $\simeq$ *isomorphism types*, and denote the isomorphism type of G by τ_G. In the case of labeled graphs, we additionally require that $\ell(v) = \ell(\varphi(v))$ for v in $V(G)$ and $\ell((u, v)) = \ell((\varphi(u), \varphi(v)))$ for (u, v) in $E(G)$. Let $\mathbf{v}$ be a *tuple* in $V(G)^k$ for $k > 0$, then $G[\mathbf{v}]$ is the subgraph induced by the multiset of elements of $\mathbf{v}$, where the nodes are labeled with integers from $\{1, \dots, k\}$ corresponding to their positions in $\mathbf{v}$.

Equivariance For $n > 0$, let S_n denote the set of all permutations of $[n]$, i.e., the set of all bijections from $[n]$ to itself. For σ in S_n and a graph G such that $V(G) = [n]$, let $G_\sigma = \sigma \cdot G$ be the graph such that $V(\sigma \cdot G) = \{v_{\sigma^{-1}(1)}, \dots, v_{\sigma^{-1}(n)}\}$ and $E(G_\sigma) = \{(v_{\sigma^{-1}(i)}, v_{\sigma^{-1}(j)}) \mid (v_i, v_j) \in E(G)\}$. That is, applying the permutation σ reorders the nodes. Hence, for two isomorphic graphs G and H on the same vertex set, i.e., $G \simeq H$, there exists σ in S_n such that $\sigma \cdot G = H$.

Let $\mathcal{G}$ denote the set of all graphs, and let $\mathcal{G}_n$ denote the set of all graphs on n nodes. A function $f\colon \mathcal{G} \to \mathbb{R}$ is *invariant* if for every $n > 0$ and every σ in S_n and graph G, $f(\sigma \cdot G) = f(G)$. A function $f\colon \mathcal{G} \mapsto \mathcal{G}$ is *equivariant* if for every $n > 0$, $f(\mathcal{G}_n) \subseteq \mathcal{G}_n$ and for every σ in S_n, $f(\sigma \cdot G) = \sigma \cdot f(G)$.

Kernels A *kernel* on a non-empty set $\mathcal{X}$ is a symmetric, positive semidefinite function $k\colon \mathcal{X} \times \mathcal{X} \to \mathbb{R}$. Equivalently, a function $k\colon \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a kernel if there is a *feature map* $\phi\colon \mathcal{X} \to \mathcal{H}$ to a Hilbert space $\mathcal{H}$ with inner product $\langle \cdot, \cdot \rangle$, such that $k(x, y) = \langle \phi(x), \phi(y) \rangle$ for all x and y in $\mathcal{X}$. A *graph kernel* is a kernel on the set $\mathcal{G}$ of all graphs.

B.1. Node-refinement algorithms (extended)

In the following, we briefly describe the Weisfeiler–Leman algorithm and related variants (Morris et al., 2020b). Let k be a fixed positive integer. There exist two definitions of the k-WL, the so-called oblivious k-WL and folklore or non-oblivious k-WL, in literature, see, e.g., (Grohe, 2021). There is a subtle difference in how they aggregate neighborhood information. Within the graph learning community, it is customary to abbreviate the oblivious k-WL as k-WL, a convention that we follow in this paper.

We proceed to the definition of the k-WL. Let $V(G)^k$ denote the set of k-tuples of nodes of the graph G. A *coloring* of $V(G)^k$ is a mapping $C\colon V(G)^k \to \mathbb{N}$, i.e., we assign a number (color) to every tuple in $V(G)^k$. The *initial coloring* C_0 of $V(G)^k$ is specified by the atomic types of the tuples, i.e., two tuples $\mathbf{v}$ and $\mathbf{w}$ in $V(G)^k$ have the same initial color iff mapping $v_i \mapsto w_i$ induces an isomorphism between the labeled subgraphs $G[\mathbf{v}]$ and $G[\mathbf{w}]$. Note that, given a tuple $\mathbf{v}$ in $V(G)^k$, we can upper-bound the running time of the computation of this initial coloring for $\mathbf{v}$ by $\mathcal{O}(k^2)$. A *color class* corresponding to a color c is the set of all tuples colored c, i.e., the set $C^{-1}(c)$.

For j in $[k]$ and w in $V(G)$, let $\phi_j(\mathbf{v}, w)$ be the k-tuple obtained by replacing the jth component of $\mathbf{v}$ with the node w. That is, $\phi_j(\mathbf{v}, w) = (v_1, \ldots, v_{j-1}, w, v_{j+1}, \ldots, v_k)$. If $\mathbf{w} = \phi_j(\mathbf{v}, w)$ for some w in $V(G)$, call $\mathbf{w}$ a *j-neighbor* of $\mathbf{v}$. The *neighborhood* of $\mathbf{v}$ is the set of all $\mathbf{w}$ such that $\mathbf{w} = \phi_j(\mathbf{v}, w)$ for some j in $[k]$ and a $w \in V(G)$.

The *refinement* of a coloring $C\colon V(G)^k \to \mathbb{N}$, denoted by $\widehat{C}$, is a coloring $\widehat{C}\colon V(G)^k \to \mathbb{N}$ defined as follows. For each j in $[k]$, collect the colors of the j-neighbors of $\mathbf{v}$ in a multiset $S_j = \{\!\{C(\phi_j(\mathbf{v}, w)) \mid w \in V(G)\}\!\}$. Then, for a tuple $\mathbf{v}$, define

$$\widehat{C}(\mathbf{v}) \coloneqq (C(\mathbf{v}), M(\mathbf{v})),$$

where $M(\mathbf{v})$ is the k-tuple $(S_1, \ldots, S_k)$. For consistency, the strings $\widehat{C}(\mathbf{v})$ thus obtained are lexicographically sorted and renamed as integers, not used in previous iterations. Observe that the new color $\widehat{C}(\mathbf{v})$ of $\mathbf{v}$ is solely dictated by the color histogram of the neighborhood of $\mathbf{v}$. In general, a different mapping $M(\cdot)$ could be used, depending on the neighborhood information that we would like to aggregate. We will refer to a mapping $M(\cdot)$ as an *aggregation map*.

***k*-dimensional Weisfeiler–Leman** For $k \geq 2$, the k-WL computes a coloring $C_\infty\colon V(G)^k \to \mathbb{N}$ of a given graph G, as follows.[6] To begin with, the initial coloring C_0 is computed. Then, starting with C_0, successive refinements $C_{i+1} = \widehat{C_i}$ are computed until convergence. That is,

$$C_{i+1}(\mathbf{v}) = (C_i(\mathbf{v}), M_i(\mathbf{v})),$$

where

$$M_i(\mathbf{v}) = \big(\{\!\{C_i(\phi_1(\mathbf{v}, w)) \mid w \in V(G)\}\!\}, \ldots, \{\!\{C_i(\phi_k(\mathbf{v}, w)) \mid w \in V(G)\}\!\}\big). \tag{3}$$

The successive refinement steps are also called *rounds* or *iterations*. Since the disjoint union of the color classes form a partition of $V(G)^k$, there must exist a finite $\ell \leq |V(G)|^k$ such that $C_\ell = \widehat{C_\ell}$, i.e., the partition induced by C_ℓ cannot be refined further. The k-WL outputs C_ℓ as the *stable coloring* C_∞.

The k-WL *distinguishes* two graphs G and H if, upon running the k-WL on their disjoint union $G \dot{\cup} H$, there exists a color c in $\mathbb{N}$ in the stable coloring such that the corresponding color class S_c satisfies

$$|V(G)^k \cap S_c| \neq |V(H)^k \cap S_c|,$$

i.e., there the numbers of c-colored tuples in $V(G)^k$ and $V(H)^k$ differ. Two graphs that are distinguished by the k-WL must be non-isomorphic, because the algorithm is defined in an isomorphism-invariant way.

[6] We define the 1-WL in the next subsection.

Finally, the application of different aggregation maps $M(\cdot)$ yield related versions of k-WL. For example, setting $M(\cdot)$ to be

$$M^F(\mathbf{v}) = \{\!\{\big(C(\phi_1(\mathbf{v}, w)), \ldots, C(\phi_k(\mathbf{v}, w))\big) \mid w \in V(G)\}\!\},$$

yields the so-called folklore-version of k-WL (see e.g. (Cai et al., 1992)). It is known that the oblivious version of the k-WL is as powerful as the folklore version of the $(k-1)$-WL (Grohe, 2021).

Local δ-k-dimensional Weisfeiler–Leman algorithm Morris et al. (2020b) introduced a more efficient variant of the k-WL, namely the *local δ-k-dimensional Weisfeiler–Leman algorithm* (δ-k-LWL). In contrast to the k-WL, the δ-k-LWL considers only a subset of the entire neighborhood of a node tuple. Let the tuple $\mathbf{w} = \phi_j(\mathbf{v}, w)$ be a j-neighbor of $\mathbf{v}$. We say that $\mathbf{w}$ is a *local* j-neighbor of $\mathbf{v}$ if w is adjacent to the replaced node v_j. Otherwise, the tuple $\mathbf{w}$ is a *global* j-neighbor of $\mathbf{v}$. The δ-k-LWL considers only local neighbors during the neighborhood aggregation process, and discards any information about the global neighbors. Formally, the δ-k-LWL algorithm refines a coloring $C_i^{k,\delta}$ (obtained after i rounds of δ-k-LWL) via the aggregation function,

$$M_i^\delta(\mathbf{v}) = \big(\{\!\{C_i^{k,\delta}(\phi_1(\mathbf{v}, w)) \mid w \in \delta(v_1)\}\!\}, \ldots, \{\!\{C_i^{k,\delta}(\phi_k(\mathbf{v}, w)) \mid w \in \delta(v_k)\}\!\}\big), \tag{4}$$

hence considering only the local j-neighbors of the tuple $\mathbf{v}$ in each iteration. The coloring function for the δ-k-LWL is then defined by

$$C_{i+1}^{k,\delta}(\mathbf{v}) = (C_i^{k,\delta}(\mathbf{v}), M_i^\delta(\mathbf{v})). \tag{5}$$

We define the 1-WL to be the δ-1-LWL, which is commonly known as Color Refinement or Naive Node Classification.[7] Hence, we can equivalently define

$$C_{i+1}^{1,\delta}(v) = (C_i^{1,\delta}(v), \{\!\{C_i^{1,\delta}(w) \mid w \in \delta(v)\}\!\}). \tag{6}$$

for a node v in $V(G)$.

Morris et al. (2020b) also defined the δ-k-LWL$^+$, a minor variation of the δ-k-LWL. Formally, the δ-k-LWL$^+$ refines a coloring C_i (obtained after i rounds) via the aggregation function

$$\begin{aligned} M^{\delta,+}(\mathbf{v}) = \big(&\{\!\{(C_i^{k,\delta}(\phi_1(\mathbf{v}, w)), \#_i^1(\mathbf{v}, \phi_1(\mathbf{v}, w))) \mid w \in \delta(v_1)\}\!\}, \ldots, \\ &\{\!\{(C_i^{k,\delta}(\phi_k(\mathbf{v}, w)), \#_i^k(\mathbf{v}, \phi_k(\mathbf{v}, w))) \mid w \in \delta(v_k)\}\!\}\big), \end{aligned} \tag{7}$$

instead of the δ-k-LWL aggregation defined in Equation (4). Here, we set

$$\#_i^j(\mathbf{v}, \mathbf{x}) \coloneqq \big|\{\mathbf{w} \colon \mathbf{w} \sim_j \mathbf{v},\, C_i^{k,\delta}(\mathbf{w}) = C_i^{k,\delta}(\mathbf{x})\}\big|, \tag{8}$$

where $\mathbf{w} \sim_j \mathbf{v}$ denotes that $\mathbf{w}$ is a j-neighbor of $\mathbf{v}$, for j in $[k]$. Essentially, $\#_i^j(\mathbf{v}, \mathbf{x})$ counts the number of j-neighbors (local or global) of $\mathbf{v}$ which have the same color as $\mathbf{x}$ under the coloring C_i (i.e., after i rounds). Morris et al. (2020b) showed that the δ-k-LWL$^+$ is slightly more powerful than the k-WL in distinguishing non-isomorphic graphs.

C. The (k, s)-LWL algorithm (extended)

Since both k-WL and its local variant δ-k-LWL consider all k-tuples of a graph, they do not scale to large graphs for larger k. Specifically, for an n-node graph, the memory requirement is $\Omega(n^k)$. Further, since the k-WL considers the graph structure only at initialization, it does not adapt to its sparsity, i.e., it does not run faster for sparser graphs. To address this issue, we introduce the (k, s)-LWL. The algorithm offers more fine-grained control over the trade-off between expressivity and scalability by only considering a subset of all k-tuples, namely those inducing subgraphs with at most s *connected components*. This combinatorial algorithm will be the basis of the permutation-equivariant neural architectures of Section 4.

Formally, let G be a graph, then $\#\mathsf{com}(G)$ denotes the number of (connected) components of G. Further, let $k \geq 1$ and $1 \leq s \leq k$, then

$$V(G)_s^k \coloneqq \{\mathbf{v} \in V(G)^k \mid \#\mathsf{com}(G[\mathbf{v}]) \leq s\}$$

[7] Strictly speaking, the 1-WL and Color Refinement are two different algorithms. That is, the 1-WL considers neighbors and non-neighbors to update the coloring, resulting in a slightly higher expressivity when distinguishing nodes in a given graph, see (Grohe, 2021) for details. For brevity, we consider both algorithms to be equivalent.

is the set of (k,s)-*tuples* of nodes, i.e, k-tuples which induce (sub-)graphs with at most s (connected) components.

In contrast to the algorithms of Appendix B.1, the (k,s)-LWL colors tuples from $V(G)_s^k$ instead of the entire $V(G)^k$. Hence, analogously to Appendix B.1, a coloring of $V(G)_s^k$ is a mapping $C_i^{k,s}\colon V(G)_s^k \to \mathbb{N}$ for $i \geq 0$, assigning a number (color) to every tuple in $V(G)_s^k$. The initial coloring $C_0^{k,s}$ of $V(G)_s^k$ is defined in the same way as before, i.e., specified by the isomorphism types of the tuples. Subsequently, the coloring is updated using the δ-k-LWL aggregation map, see Equation (4). Hence, the (k,s)-LWL is a variant of the δ-k-LWL considering only (k,s)-tuples, i.e., Equation (4) is replaced with

$$\begin{aligned} M_i^{\delta,k,s}(\mathbf{v}) \coloneqq \big(&\{\!\{C_i^{k,s}(\phi_1(\mathbf{v},w)) \mid w \in \delta(v_1) \text{ and } \phi_1(\mathbf{v},w) \in V(G)_s^k\}\!\}, \\ &\ldots, \{\!\{C_i^{k,s}(\phi_k(\mathbf{v},w)) \mid w \in \delta(v_k) \text{ and } \phi_k(\mathbf{v},w) \in V(G)_s^k\}\!\}\big), \end{aligned} \tag{9}$$

i.e., $M_i^{\delta}(\mathbf{v})$ restricted to colors of (k,s)-tuples. The stable coloring $C_\infty^{k,s}$ is defined analogously to the stable coloring C_∞^k. In the following two subsections, we investigate the properties of the algorithm in detail.

Analogously to the δ-k-LWL$^+$, we also define the (k,s)-LWL$^+$ using

$$\begin{aligned} M^{\delta,+}(\mathbf{v}) = \big(&\{\!\{(C_i^{k,s}(\phi_1(\mathbf{v},w)), \#_{i,s}^1(\mathbf{v},\phi_1(\mathbf{v},w))) \mid w \in \delta(v_1) \text{ and } \phi_1(\mathbf{v},w) \in V(G)_s^k\}\!\}, \ldots, \\ &\{\!\{(C_i^{k,s}(\phi_k(\mathbf{v},w)), \#_{i,s}^k(\mathbf{v},\phi_k(\mathbf{v},w))) \mid w \in \delta(v_k) \text{ and } \phi_k(\mathbf{v},w) \in V(G)_s^k\}\!\}\big), \end{aligned}$$

where the function

$$\#_{i,s}^j(\mathbf{v},\mathbf{x}) = \big|\{\mathbf{w}\colon \mathbf{w} \sim_j \mathbf{v},\, C_i^{k,s}(\mathbf{w}) = C_i^{k,s}(\mathbf{x}) \text{ and } \mathbf{w} \in V(G)_s^k\}\big|,$$

restricts $\#_i^j(\mathbf{v},\mathbf{x})$ to (k,s)-tuples.

Here, we investigate the expressivity of the (k,s)-LWL, i.e., its ability to distinguish non-isomorphic graphs, for different choices of k and s. In Section 4, we will leverage these results to devise universal, permutation-equivariant graph networks. We start off with the following simple observation. Since the (k,k)-LWL colors all k-tuples, it is equal to the δ-k-LWL.

Observation 2. Let $k \geq 1$, then

$$(k,k)\text{-LWL} \equiv \delta\text{-}k\text{-LWL} \quad \text{and} \quad (1,1)\text{-LWL} \equiv \delta\text{-}k\text{-LWL} \equiv 1\text{-WL}.$$

The following result shows that the $(k,1)$-LWL form a *hierarchy*, i.e., the algorithm becomes more expressive as k increases.

Theorem 5. Let $k \geq 1$, then

$$(k+1,1)\text{-LWL} \sqsubset (k,1)\text{-LWL}.$$

Moreover, we also show that the $(k,2)$-LWL is more expressive than the $(k,1)$-LWL.

Proposition 6. For $k \geq 2$, it holds that

$$(k,2)\text{-LWL} \sqsubset (k,1)\text{-LWL}.$$

Further, the following theorem yields that increasing the parameter s results in higher expressivity. Formally, we show that the (k,k)-LWL is strictly more expressive than the $(k,2)$-LWL.

Theorem 7. For $k \geq 2$, it holds that

$$(k,k)\text{-LWL} \sqsubset (k,2)\text{-LWL}.$$

See Appendix C.1 for an analysis of the asymptotic running time of the (k,s)-LWL, showing that it only depends on k, s, and the sparsity of the graph. In particular, the running time of the (k,s)-LWL on an n-vertex graph of bounded degree is $\tilde{\mathcal{O}}(n^s)$ instead of the usual $\tilde{\mathcal{O}}(n^k)$ for the k-WL, for fixed k and s.

C.0.1. Proofs of Theorems 1 and 3 and Proposition 2

To prove Theorem 1, we introduce the (k,s)-*tuple graph*. It essentially contains all (k,s)-tuples as nodes, where each node $v_{\mathbf{t}}$ is labeled by the isomorphism type of the (k,s)-tuple $\mathbf{t}$. We join two nodes by an edge, labeled j, if the underlying (k,s)-tuples are j-neighbors. The formal definition of the (k,s)-tuple graph is as follows. Recall that τ denotes an isomorphism type.

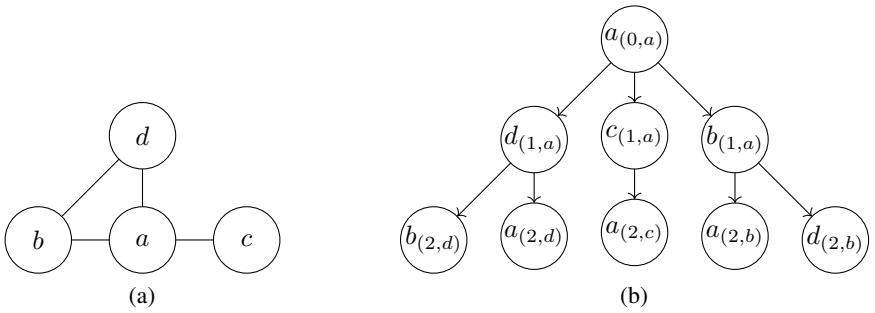

Figure 2. Illustration of the unrolling operation around the node a for $i = 2$.

Definition 8. Let G be a graph and let $k \geq 1$, and s in $[k]$. Further, let $\mathbf{s}$ and $\mathbf{t}$ be tuples in $V(G)_s^k$. Then the directed, labeled (k, s)-*tuple graph* $T_s^k(G) = (V_T, E_T, \ell_T)$ has node set $V_T = \{v_\mathbf{t} \mid \mathbf{t} \in V(G)_s^k\}$, and

$$(v_\mathbf{s}, v_\mathbf{t}) \in E_T \iff \mathbf{t} = \phi_j(\mathbf{s}, w) \text{ holds for some } j \text{ in } [k] \text{ and some } w \text{ in } V(G). \tag{10}$$

We set $\ell_T((v_\mathbf{s}, v_\mathbf{t})) \coloneqq j$ if $\mathbf{t}$ is a local j-neighbor of $\mathbf{s}$, and let $\ell_T(v_\mathbf{s}) \coloneqq \tau_{G[\mathbf{s}]}$.

Given a graph G and the corresponding (k, s)-tuple graph $T_s^k(G)$, we define a variant of the 1-WL, which takes into account edge labels. Namely, for $v_\mathbf{t}$ in V_T, the new algorithm uses the colorings $C_0^{1,\delta,*}(v_\mathbf{t}) = \tau_{G[\mathbf{t}]}$ and

$$C_{i+1}^{1,\delta,*}(v_\mathbf{t}) = (C_i^{1,\delta,*}(v_\mathbf{t}), \{\!\{(C_i^{1,\delta,*}(v_\mathbf{s}), \ell(v_\mathbf{t}, v_\mathbf{s})) \mid v_\mathbf{s} \in \delta(v_\mathbf{t})\}\!\}) \tag{11}$$

for $i > 0$. Note that the 1-WL, see Equation (6), and the variant defined via Equation (11) have the same asymptotic running time. The following lemma states that the (k, s)-LWL can be simulated on the (k, s)-tuple graph using the above variant of the 1-WL.

Lemma 9. Let G be a graph, $k \geq 1$, and s in $[k]$. Then

$$C_i^{k,s}(\mathbf{t}) = C_i^{k,s}(\mathbf{u}) \iff C_i^{1,\delta,*}(v_\mathbf{t}) = C_i^{1,\delta,*}(v_\mathbf{u}),$$

for all $i \geq 0$, and all (k, s)-tuples $\mathbf{t}$ and $\mathbf{u}$ in $V(G)_s^k$.

Proof sketch. Induction on the number of iterations using Definition 8. □

The *unrolling* of a neighborhood around a node of a given graph to a tree is defined as follows, see Figure 2 for an illustration.

Definition 10. Let $G = (V, E, \ell)$ be a labeled (directed) graph and let v be in V. Then $U_{G,v}^i = (W_i, F_i, l_i)$ for $i \geq 0$ denotes the *unrolled tree* G around v at depth i, where

$$W_i = \begin{cases} \{v_{(0,v)}\} & \text{if } i = 0 \\ W_{i-1} \cup \{u_{(i,w_{(i-1,p)})} \mid u \in \delta(w) \text{ for } w_{(i-1,p)} \in W_{i-1}\} & \text{otherwise,} \end{cases}$$

and

$$F_i = \begin{cases} \emptyset & \text{if } i = 0 \\ F_{i-1} \cup \{(w_{(i-1,p)}, u_{(i,w)}) \mid u \in \delta(w) \text{ for } w_{(i-1,p)} \in W_{i-1}\} & \text{otherwise.} \end{cases}$$

The label function is defined as $l_i(u_{(j,p)}) = \ell(u)$ for u in V, and $l_i(u_{(j,w)}) = \ell((w, u))$. For notational convenience, we usually omit the subscript i.

In the following, we use the unrolled tree for the above defined (k, s)-tuple graph. For $k \geq 2$ and s in $[k]$, we denote the *directed*, unrolled tree of the (k, s)-tuple graph of G around the node $v_\mathbf{t}$ at depth i for the tuple $\mathbf{t}$ in $V(G)_s^k$ by $\mathbf{U}_{T_s^k(G),v_\mathbf{t}}^i$. For notational convenience, we write $\mathbf{U}_{T,v_\mathbf{t}}^i$ for $\mathbf{U}_{T_s^k(G),v_\mathbf{t}}^i$. Further, for two (k, s)-tuples $\mathbf{t}$ and $\mathbf{u}$, we write

$$\mathbf{U}_{T,v_\mathbf{t}}^i \simeq_{v_\mathbf{t} \to v_\mathbf{u}} \mathbf{U}_{T,v_\mathbf{u}}^i \tag{12}$$

(a) (b)

Figure 3. The graphs A_{k+2} and B_{k+2} for $k = 4$.

if there exists a (labeled) isomorphism φ between the two unrolled trees, mapping the (root) node $v_{\mathbf{t}}$ to $v_{\mathbf{u}}$. Moreover, we need the following two results.

Theorem 11 ((Busacker & Saaty, 1965; Valiente, 2002))**.** The 1-WL distinguishes any two directed, labeled non-isomorphic trees.

Using the first result, the second one states that the (k, s)-LWL can be simulated by the variant of the 1-WL of Equation (11) on the unrolled tree of the (k, s)-tuple graph, and hence can be reduced to a tree-isomorphism problem.

Lemma 12. Let G be a *connected* graph, then the (k, s)-LWL colors the tuples $\mathbf{t}$ and $\mathbf{u}$ in $V(G)_s^k$ equally if and only if the corresponding unrolled (k, s)-tuple trees are isomorphic, i.e.,

$$C_i^{k,s}(\mathbf{t}) = C_i^{k,s}(\mathbf{u}) \iff \mathbf{U}^i_{T,v_{\mathbf{t}}} \simeq_{v_{\mathbf{t}} \to v_{\mathbf{u}}} \mathbf{U}^i_{T,v_{\mathbf{u}}},$$

for all $i \geq 0$.

Proof sketch. First, by Lemma 9, we can simulate the (k, s)-LWL for the graph G using the (k, s)-tuple graph $T_s^k(G)$. Secondly, consider a node $v_{\mathbf{t}}$ in the (k, s)-tuple graph $T_s^k(G)$ and a corresponding node in the unrolled tree around $v_{\mathbf{t}}$. Observe that the neighborhoods for both nodes are identical. By definition, this holds for all nodes (excluding the leaves) in the unrolled tree. Hence, by Lemma 9, we can simulate the (k, s)-LWL for each tuple $\mathbf{t}$ by running the 1-WL in the unrolled tree around $v_{\mathbf{t}}$ in the (k, s)-tuple graph. Since the 1-WL decides isomorphism for trees, see Theorem 11, the result follows. □

The following lemma shows that the $(k + 1, 1)$-LWL is strictly more expressive than the $(k, 1)$-LWL for every $k \geq 2$.

Lemma 13. Let $k \geq 2$. Let $G \coloneqq C_{2(k+2)}$ and $H \coloneqq C_{(k+2)} \dot{\cup} C_{(k+2)}$. Then, the graphs G and H are distinguished by $(k + 1, 1)$-LWL, but they are not distinguished by $(k, 1)$-LWL.

Proof. We first show that the $(k+1, 1)$-LWL distinguishes the graphs G and H. Let $\mathbf{v}$ in $V(G)_1^{k+1}$ be a tuple $(v_1, \ldots, v_{k+1})$ such that $v_1, \ldots, v_{k+1}$ is a path of length k in G. Let $\mathbf{w}$ in $V(H)_1^{k+1}$ be a tuple $(w_1, \ldots, w_{k+1})$ such that $w_1, \ldots, w_{k+1}$ is a path of length k in H. By the structure of H, there exists a vertex w_{k+2} in $V(H)$ such that $w_1, \ldots, w_{k+2}$ forms a cycle of length $k + 2$. We claim that $\mathbf{v}$ does not have any local 1-neighbor $\mathbf{x}$ in $V(G)_1^{k+1}$ such that $\mathbf{x}$ is non-repeating, i.e., every vertex in $\mathbf{x}$ is distinct. This holds because replacing the first vertex of $\mathbf{v}$ with any other vertex of G will yield a disconnected tuple. On the other hand, $\mathbf{w}$ admits a non-repeating, local 1-neighbor, obtained by replacing the first vertex w_1 by w_{k+2}. Hence, the $(k + 1, 1)$-LWL distinguishes G and H.

Next, we show that the $(k, 1)$-LWL does not distinguish the graphs G and H. Indeed, for every j in $[k]$, every k-tuple $\mathbf{x}$ in $V(G)_1^k$ or $V(H)_1^k$ has exactly two local j-neighbors, corresponding to the two neighbors y, z of the vertex x_j. The exact number of local j-neighbors of $\mathbf{x}$ which additionally lie in $V(G)_1^k$ (or $V(H)_1^k$) depends only on the atomic type of $\mathbf{x}$, since the length of cycles in G and H is at least $k + 2$. Hence, the $(k, 1)$-LWL neighborhood of every tuple in G or H depends only on its atomic type. This implies that the $(k, 1)$-LWL does not refine the initial coloring for G as well as H, and hence it does not distinguish G and H. □

Although Lemma 13 already implies Theorem 1, the construction hinges on the fact that the graphs G and H are not connected. To address this, for $k \geq 2$, we introduce two connected graphs A_{k+2} and B_{k+2} defined as follows. The graph A_{k+2} has $2(k + 2)$ nodes and $2(k + 2) + 1$ edges, and consists of two disjoint cycles on $k + 2$ nodes connected by a single edge. The graph B_{k+2} also has the same number of nodes and edges, and consists of two cycles on $k + 3$ nodes, each,

sharing exactly two adjacent nodes. See Figure 3 for an illustration of the graphs A_{k+2} and B_{k+2} for $k = 4$. We obtain the following result for the two graphs.

Lemma 14. For $k \geq 2$, the $(k, 1)$-LWL does not distinguish the graphs A_{k+2} and B_{k+2}, while the $(k + 1, 1)$-LWL does.

Proof. We first show the second part, i.e., that the $(k + 1, 1)$-LWL distinguishes the graphs A_{k+2} and B_{k+2}. Without loss of generality, assume that $V(A_{k+2}) = \{a_1, \ldots, a_{2(k+2)}\}$ and that $E(A_{k+2})$ consists of the edges (a_i, a_{i+1}) for $1 \leq i \leq k+1$, $(a_1, a_{(k+2)})$, (a_i, a_{i+1}) for $(k+3) \leq i \leq 2(k+2)-1$, and $(a_{k+3}, a_{2(k+2)})$. The two cycles are connected by the edge $(a_{(k+2)}, a_{(k+3)})$ in $E(A_{k+2})$. Further, assume $V(B_{k+2}) = \{b_1, \ldots, b_{2(k+2)}\}$ where (b_i, b_{i+1}) in $E(B_{k+2})$ for $1 \leq i \leq 2(k+2)-1$ and $(b_1, b_{2(k+2)+2})$ in $E(B_{k+2})$. Finally, the edge $(b_1, b_{k+3}) \in E(B_{k+2})$ is shared by the two cycles of length $k + 3$ each.

Now, let $\mathbf{t} = (a_1, \ldots, a_{k+1})$ in $V(A_{k+2})_1^{k+1}$. Observe that the tuple $(a_1, \ldots, a_{k+2})$ is a $(k + 1)$-neighbor of the tuple $\mathbf{t}$, inducing a graph on $k + 1$ nodes. Further, since the two cycles in the graph B_{k+2} have length $k + 3$, there is no tuple without repeated nodes that has a $(k + 1)$-neighbor without repeated nodes. Hence, the two graphs are distinguished by $(k + 1, 1)$-LWL.

We now show that the $(k, 1)$-LWL does not distinguish the graphs A_{k+2} and B_{k+2}. First, we construct a bijection $\theta \colon V(A_{k+2}) \to V(B_{k+2})$ as induced by the following coloring:

Based on the bijection θ, we define the bijection $\theta^k \colon V(A_{k+2})_1^k \to (A_{B+2})_1^k$, by applying θ component-wise to (k, s)-tuples. Observe that $G[\mathbf{s}] \simeq G[\theta^k(\mathbf{s})]$ for $\mathbf{s}$ in $V(A_{k+2})_1^k$.

Claim 15. Let $\mathbf{s}$ be a tuple in $V(G)_1^k$ and $\mathbf{t} = \theta^k(\mathbf{s})$ in $V(H)_1^k$. Let $N_j(\mathbf{s})$ and $N_j(\mathbf{t})$ be the j-neighbors of the tuple $\mathbf{s}$ and $\mathbf{t}$, respectively, for j in $[k]$. Then θ^k yields a one-to-one correspondence between $N_j(\mathbf{s})$ and $N_j(\mathbf{t})$. Consequently, $G[\mathbf{u}] \simeq G[\theta^k(\mathbf{u})]$ for $\mathbf{u}$ in $N_j(\mathbf{s})$ and $\theta^k(\mathbf{u})$ in $N_j(\mathbf{t})$.

Proof. The desired claim follows by observing that the bijective map $\theta \colon V(A_{k+2}) \to V(B_{k+2})$ preserves neighborhoods, i.e. for every x in $V(A_{k+2})$, $\theta(N_F(x)) = N_K(\theta(x))$. □

We now again leverage the above claim to show that $C_i^{k,s}(\mathbf{s}) = C_i^{k,s}(\theta^k(\mathbf{s}))$ for $i \geq 0$, implying the required result. By a straightforward inductive argument, using Claim 15, we can inductively construct a tree isomorphism between the unrolled trees around the node v_s and v_t in the corresponding (k, s)-tuple graph such that $\mathbf{U}_{T,v_\mathbf{s}}^i \simeq_{v_\mathbf{s} \to v_\mathbf{t}} \mathbf{U}_{T,v_\mathbf{t}}^i$. By Lemma 12, this implies $C_i^{k,s}(\mathbf{s}) = C_i^{k,s}(\theta(\mathbf{t}))$ for $i \geq 0$. This shows that the (k, s)-LWL does not distinguish A_{k+2} and B_{k+2}. □

Hence, Theorem 1 directly follows from Lemma 14. Moreover, we also show that the $(k, 2)$-LWL is more expressive than the $(k, 1)$-LWL.

Proposition 16. Let $k \geq 2$. Then

$$(k, 2)\text{-LWL} \sqsubset (k, 1)\text{-LWL}.$$

Proof. As in Lemma 13, let $G \coloneqq C_{2(k+2)}$ and $H \coloneqq C_{(k+2)} \dot{\cup} C_{(k+2)}$. By Lemma 13, G and H are not distinguished by the $(k, 1)$-LWL for $k \geq 2$. We claim that the $(k, 2)$-LWL distinguishes G and H for $k = 2$ already. Since the $(k, 2)$-LWL is at least as powerful as the $(2, 2)$-LWL, this yields the desired claim.

With respect to the $(2, 2)$-LWL, observe that the $(2, 2)$-tuple graph $T_2^2(H)$ consists of four connected components while the $(2, 2)$-tuple graph $T_2^2(G)$ consists of a single connected component. More precisely, there exist two connected components of $T_2^2(H)$ that consist only of 2-tuples containing two non-adjacent nodes which are in the same connected component of the graph H. Note that none of these 2-tuples is adjacent to any 2-tuples in $V(H)_1^2$. Moreover, there exists no such connected component in $T_2^2(G)$. Also, note that the number of neighbors of each 2-tuple of the graphs is exactly 4, excluding self loops. Hence, the $(2, 2)$-LWL will distinguish the two graphs. □

Moreover, the following result shows that increasing the parameter s results in higher expressivity. Formally, we show that the (k, k)-LWL is strictly more expressive than the $(k, 2)$-LWL. Note that we use vertex-colored graphs (rather than simple undirected graphs) in our proofs.

Theorem 17. Let $k \geq 2$, then

$$(k, k)\text{-LWL} \sqsubset (k, 2)\text{-LWL}.$$

For the proof of Theorem 17, we modify the construction employed in (Morris et al., 2020b), Appendix C.1.1., where they provide an infinite family of graphs $(G_k, H_k)_{k\in\mathbb{N}}$ such that (a) k-WL does not distinguish G_k and H_k, although (b) δ-k-LWL distinguishes G_k and H_k. Since our proof closely follows theirs, let us recall some relevant definitions from their paper.

Construction of G_k and H_k. Let K denote the complete graph on $k + 1$ vertices (without any self-loops). The vertices of K are indexed from 0 to k. Let $E(v)$ denote the set of edges incident to v in K: clearly, $|E(v)| = k$ for all v in $V(K)$. We call the elements of $V(K)$ *base vertices*, and the elements of $E(K)$ *base edges*. Define the graph G_k as follows:

1. For the vertex set $V(G_k)$, we add
   (a) (v, S) for each v in $V(K)$ and for each *even* subset S of $E(v)$,
   (b) two vertices e^1, e^0 for each edge e in $E(K)$.
2. For the edge set $E(G_k)$, we add
   (a) an edge $\{e^0, e^1\}$ for each e in $E(K)$,
   (b) an edge between (v, S) and e^1 if v in e and e in S,
   (c) an edge between (v, S) and e^0 if v in e and e not in S,

For every v in K, the set of vertices of the form (v, S) is called the *vertex cloud* for v. Similarly, for every edge e in $E(K)$, the set of vertices of the form $\{e^0, e^1\}$ is called the *edge cloud* for e.

Define a companion graph H_k, in a similar manner to G_k, with the following exception: in Step 1(a), for the vertex 0 in $V(K)$, we choose all *odd* subsets of $E(0)$. Counting vertices, we find that $|V(G)| = |V(H)| = (k + 1) \cdot 2^{k-1} + \binom{k+1}{2} \cdot 2$. This finishes the construction of the graphs G and H. We set $G_k \coloneqq G$ and $H_k \coloneqq H$.

Distance-two-cliques. A set S of vertices is said to form a *distance-two-clique* if the distance between any two vertices in S is exactly 2. The following results were shown in (Morris et al., 2020b).

Lemma 18 ((Morris et al., 2020b))**.** The following holds for the graphs G_k and H_k defined above.

- There exists a distance-two-clique of size $(k + 1)$ inside G_k.
- There does not exist a distance-two-clique of size $(k + 1)$ inside H_k.

Hence, G_k and H_k are non-isomorphic.

Lemma 19 ((Morris et al., 2020b))**.** The δ-k-LWL distinguishes G_k and H_k. On the other hand, k-WL does not distinguish G_k and H_k.

We are ready to present the proof of Theorem 17.

Proof of Theorem 17. Observe that the (k, k)-LWL is the same as the δ-k-LWL. Hence, it suffices to show an infinite family of graphs (X_k, Y_k), k in $\mathbb{N}$, such that (a) $(k, 2)$-LWL does not distinguish X_k and Y_k, although (b) δ-k-LWL distinguishes X_k and Y_k.

Let X_k be the graph obtained from the graph G_k as follows. First, for every base vertex v in $V(K)$, every vertex of $V(G_k)$ in the vertex cloud for v receives a color Red_v. Hence, vertex clouds form color classes, where each such class has a distinct color. Similarly, for every base edge e in $E(K)$, every vertex of $V(G_k)$ in the edge cloud for e receives a color Blue_e. Finally, let $\Delta > 3k$. Then, we replace every edge e in G_k by a path of length Δ, such that every vertex on this path is colored with the color $(\{c, c'\})$, where c and c' are the colors of the endpoints of e in G_k. We call such path vertices *auxiliary* vertices. The graph Y_k is obtained from H_k by an identical construction.

First, we show that the $(k, 2)$-LWL does not distinguish the graphs X_k and Y_k. We use a modified version of the bijective k-pebble game (Grohe, 2017): (a) we enforce the k pebbles to form at most two components at any point during the game,

and (b) when the Spoiler and Duplicator pick the i^{th} pebble from each graph, the Duplicator is required to exhibit a bijection only between the position-i local neighbourhoods of the two pebbling configuration tuples $\boldsymbol{x} \in X_k^k$ and $\boldsymbol{y} \in Y_k^k$, instead of a bijection between the vertex sets of X_k and Y_k. Observe that there can be at most two vertices out of $k+1$ vertices in $V(K)$ such that the corresponding vertex clouds contain a tupled vertex, by our choice of Δ. Hence, in the usual parlance of Cai-Fürer-Immerman games (Cai et al., 1992), the twisted edge can always be hidden among the remaining $(k-1)$ vertices of K. This ensures that for all $i \in [k]$, a partial isomorphism between $\boldsymbol{x} \backslash \boldsymbol{x}_i$ and $\boldsymbol{y} \backslash \boldsymbol{y}_i$ can always be extended to a bijective mapping between the i-local-neighborhoods of $\boldsymbol{x}$ and $\boldsymbol{y}$. Hence, the Duplicator cannot win this pebble game and therefore, (k, 2)-LWL cannot distinguish the graphs X_k and Y_k.

Next, we show that the δ-k-LWL distinguishes the graphs X_k and Y_k. Our proof closely follows the corresponding proof in (Morris et al., 2020b). Instead of showing a discrepancy in the number of distance-two-cliques, we instead use colored-distance-$(2\Delta+1)$-cliques defined as follows. Let S be a set of vertices belonging to the vertex clouds. The set S is said to form a colored-distance-$(2\Delta+1)$-clique if any two vertices in S are connected by a path of exactly $2\Delta+1$ vertices, of which 2Δ are auxiliary vertices and one vertex is a vertex from an edge cloud. Analogously to their proof, it can be shown that (a) there exists a colored-distance-$(2\Delta+1)$-clique of size $(k+1)$ inside X_k, and (b) there does not exist a colored-distance-$(2\Delta+1)$-clique of size $(k+1)$ inside Y_k, and hence, (c) X_k and Y_k are non-isomorphic. Finally, we claim that the δ-k-LWL is powerful enough to detect colored-distance-$(2\Delta+1)$-cliques. The proof is analogous to (Morris et al., 2020b, Appendix C.1.1, Proof of Lemma 9). This yields that the δ-k-LWL distinguishes the graphs X_k and Y_k. □

C.1. Asymptotic running time

In the following, we bound the asymptotic running time of the (k,s)-LWL. Due to Lemma 9, we can upper-bound the running time of the (k,s)-LWL for a given graph by upper-bounding the time to construct the (k,s)-tuple graph and running the 1-WL variant of Equation (11) on top. Proposition 21 establishes an upper bound on the asymptotic running time for constructing the (k,s)-tuple graph from a given graph. Thereto, we assume a *d-bounded-degree graph* G, for $d \geq 1$, i.e., each node has at most d neighbors.

To prove the proposition, we define (k,s)-*multisets*. Let G be a graph, $k \geq 1$, and s in $[k]$, then the set of (k,s)-multisets

$$S(G)_s^k = \{\{\!\{v_1, \ldots, v_k\}\!\} \mid \mathbf{v} \in V(G)_s^k\}$$

contains the set of multisets inducing subgraphs of G on at most k nodes with at most s components. The following results upper-bounds the running time for the construction of $S(G)_s^k$.

Algorithm 1 Generate (k,s)-multisets

Input: Graph G, k, s, and $S(G)_s^s$
Output: (k,s)-multiset $S(G)_s^k$

```
Let R be an empty set data structure
for M ∈ S(G)_s^s do
    Let S be a queue data structure containing only (M, s)
    while S not empty do
        Pop (T, c) from queue S
        if c + 1 ≤ k then
            for t ∈ T do
                for u ∈ δ(t) ∪ {t} do
                    Add (T ∪ {u}, c + 1) to S
        else
            Add T to R
return R
```

Proposition 20. Let G be a d-bounded-degree graph, $k \geq 2$, and s in $[k-1]$. Then Algorithm 1 computes $S_s^k(G)$ in time $\tilde{\mathcal{O}}(n^s \cdot k^{k-s}(d+1)^{k-s})$.

Proof. Let $c < k$ and let T' be an element in $S(G)_s^{c+1}$. By definition of $S(G)_s^c$ and $S(G)_s^{c+1}$, there exists a c-element multiset T in $S(G)_s^c$ such that $T' = T \cup \{v\}$ is in $S(G)_s^{c+1}$ for a node v in $V(G)$. Since s is fixed, v is either in the

neighborhood $\delta(w)$ for w in T or $v = w'$ for w' in T. Hence, lines 7 to 9 in Algorithm 1 generate $S(G)_s^{c+1}$ from $S(G)_s^c$. The set data structure R makes sure that the final solution will not contain duplicates. The running time follows directly when using, e.g., a red–black tree, to represent the set R. □

Based on the above result, we can easily construct $T_s^k(G)$ from $S_s^k(G)$, implying the following result.

Proposition 21. Let G be a d-bounded-degree graph, $k \geq 3$, and s in $[k-1]$. Then we can compute $T_s^k(G)$ in time $\tilde{\mathcal{O}}(n^s \cdot k^{k-s}(d+1)^{k-s+1} \cdot k! \cdot k)$.

Proof. The running time follows directly from Proposition 20. That is, from $S_s^k(G)$, we can generate the set $V_s^k(G)$ by generating all permutations of each element in the former. By iterating over each resulting (k,s)-tuple and each component of such (k,s)-tuple, we can construct the needed adjacency information. □

Hence, unlike for the k-WL, the running time of the (k,s)-LWL does not depend on n^k for an n-node graph and is solely dictated by s, k, and the sparsity of the graph.

Moreover, observe that the upper bound given in Proposition 21, by leveraging Lemma 9, also upper-bounds the asymptotic running time for one iteration of the (k,s)-LWL.

D. SpeqNets: Sparse, permutation-equivariant graph networks

We can now leverage the above combinatorial insights to derive sparsity-aware, permutation-equivariant graph networks, denoted (k,s)-SpeqNet. Given a labeled graph G, let each (k,s)-tuple $\mathbf{v}$ in $V(G)_s^k$ be annotated with an initial feature $f^{(0)}(\mathbf{v})$ determined by its (labeled) isomorphism type, e.g., a one-hot encoding of $\tau_{G[\mathbf{v}]}$. Alternatively, we can also use some application-specific, real-valued feature. In each layer $t > 0$, we compute a new feature $f^{(t)}(\mathbf{v})$ as

$$f_{\text{mrg}}^{W_1}\Big(f^{(t-1)}(\mathbf{v}), f_{\text{agg}}^{W_2}\big(\{\!\{f^{(t-1)}(\phi_1(\mathbf{v},w)) \mid w \in \delta(v_1) \text{ and } \phi_1(\mathbf{v},w) \in V(G)_s^k\}\!\}, \ldots, \{\!\{f^{(t-1)}(\phi_k(\mathbf{v},w)) \mid w \in \delta(v_k) \text{ and } \phi_k(\mathbf{v},w) \in V(G)_s^k\}\!\}\big)\Big),$$

in $\mathbb{R}^{1\times e}$, where $W_1^{(t)}$ and $W_2^{(t)}$ are learnable parameter matrices from $\mathbb{R}^{d\times e}$ for some $d, e > 0$. Here, $f_{\text{mrg}}^{W_2}$ and $f_{\text{agg}}^{W_1}$ are arbitrary differentiable functions, responsible for merging and aggregating the relevant feature information, respectively. Note that we can naturally handle discrete node and edge labels as well as directed graphs. The following result demonstrates the expressive power of the (k,s)-SpeqNet, in terms of distinguishing non-isomorphic graphs.

Theorem 22. Let (V,E,ℓ) be a labeled graph, and let $k \geq 1$ and s in $[k]$. Then for all $t \geq 0$, there exists weights $W_1^{(t)}$ and $W_2^{(t)}$ such that

$$C_t^{k,s}(\mathbf{v}) = C_t^{k,s}(\mathbf{w}) \iff f^{(t)}(\mathbf{v}) = f^{(t)}(\mathbf{w}).$$

Hence, the following holds for all $k \geq 1$:

$$(k,s)\text{-SpeqNet} \equiv (k,s)\text{-LWL}.$$

Proof sketch. First, observe that the (k,s)-LWL can be simulated on an appropriate node- and edge-labeled graph, see Lemma 9. Secondly, following the proof of (Morris et al., 2019, Theorem 2), there exists a parameter matrix $W_2^{(t)}$ such that we can injectively map each multiset in Equation (1), representing the local j-neighbors for j in $[k]$, to a d-dimensional vector. Moreover, we concatenate j to each such vector to distinguish between different neighborhoods. Again, by (Morris et al., 2019, Theorem 2), there exists a parameter matrix $W_1^{(t)}$ such that we can injectively map the set of resulting k vectors to a unique vector representation. Alternatively, one can concatenate the resulting k vectors and use a multi-layer perceptron to learn a joint lower-dimensional representation. □

It is not possible to come up with an architecture, i.e., instantiations of $f_{\text{mrg}}^{W_1}$ and $f_{\text{agg}}^{W_2}$ such that it becomes more powerful than the (k,s)-LWL, see (Morris et al., 2019). *However, all results from the previous section can be lifted to the neural setting,* see also Section 2.1. Analogously to GNNs, the above architecture can naturally handle continuous node and edge labels. By using the tools developed in (Azizian & Lelarge, 2020), it is straightforward to show that the above architecture is universal, i.e., it can approximate any possible permutation-invariant function over graphs up to an arbitrarily small additive error.

E. Additional experimental results

Table 3. Dataset statistics and properties for graph-level prediction tasks, †—Continuous vertex labels following (Gilmer et al., 2017), the last three components encode 3D coordinates.

Dataset	Properties					
	Number of graphs	Number of classes/targets	∅ Number of nodes	∅ Number of edges	Node labels	Edge labels
ENZYMES	600	6	32.6	62.1	✓	✗
IMDB-BINARY	1 000	2	19.8	96.5	✗	✗
IMDB-MULTI	1 500	3	13.0	65.9	✗	✗
MUTAG	188	2	17.9	19.8	✓	✗
NCI1	4 110	2	29.9	32.3	✓	✗
PTC_FM	349	2	14.1	14.5	✓	✗
PROTEINS	1 113	2	39.1	72.8	✓	✗
REDDIT-BINARY	2 000	2	429.6	497.8	✗	✗
ALCHEMY	202 579	12	10.1	10.4	✓	✓
QM9	129 433	12	18.0	18.6	✓(13+3D)†	✓(4)

Table 4. Dataset statistics and properties for node-level prediction tasks.

Dataset	Properties		
	Number of nodes	Number of edges	Number of node features
CORNELL	183	295	1 703
TEXAS	183	309	1 703
WISCONSIN	251	490	1 703

Table 5. Overall computation times for selected datasets in seconds (Number of iterations for WL-based methods: 5), OOT—Computation did not finish within one day (24h), OOM—Out of memory.

	Graph Kernel	Dataset					
		ENZYMES	IMDB-BINARY	IMDB-MULTI	MUTAG	NCI1	PTC_MR
Glob.	1-WL	<1	<1	<1	<1	1.9	<1
	2-WL	225.9	91.2	38.3	4.3	1 127.8	10.7
	3-WL	55 242.7	17 565.2	4 977.1	259.8	OOT	1324.2
Local	δ-2-LWL	25.2	27.3	19.8	<1	82.2	1.1
	δ-2-LWL$^+$	25.6	26.1	18.5	<1	108.3	1.2
	δ-3-LWL	3 519.0	3 560.3	1957.5	36.5	15 207.3	89.9
	δ-3-LWL$^+$	3 674.9	3 636.5	2162.3	43.6	15 945.6	111.1
(k,s)-LWL	$(2,1)$-LWL	1.6	12.0	11.0	<1	5.8	<1
	$(2,1)$-LWL$^+$	1.7	12.6	11.0	<1	6.7	<1
	$(3,1)$-LWL	51.1	1 040.2	1112.2	1.4	111.3	1.9
	$(3,1)$-LWL$^+$	52.1	1 049.4	1238.7	1.6	120.1	2.0
	$(3,2)$-LWL	937.9	2 571.1	2252.6	19.0	3 502.6	29.6
	$(3,2)$-LWL$^+$	1 046.1	2 937.8	2572.2	22.4	3 888.7	34.4

Table 6. Average speed-up ratios over all epochs (training and testing).

Method	Dataset	
	ALCHEMY (10K)	QM9
GINE-ε	0.5	1.3
$(2,1)$-SpeqNet	1.0	1.0
$(2,2)$-SpeqNet	1.3	3.4