Weisfeiler and Leman go sparse: Towards scalable higher-order graph embeddings

Christopher Morris [1] **Gaurav Rattan** [2] **Petra Mutzel** [3]

Abstract

The 1-dimensional Weisfeiler-Leman (WL) algorithm has recently emerged as a powerful tool for analysis and design of kernels and neural architectures for (supervised) learning with graphs. However, due to the purely local nature of the algorithms, they might miss essential patterns. Here, the k-dimensional WL accounts for the higher-order interactions between vertices by defining a suitable notion of adjacency between k-tuples of vertices. However, it does not scale and may suffer from overfitting when used in a machine learning setting. We circumvent these issues by proposing new *local* variants of the k-WL and corresponding neural architectures, which consider a subset of the original neighborhood. The expressive power of (one of) our algorithms is strictly higher than the original k-WL with no losses in scalability. In fact, for sparse graphs, the scalability improves by several orders of magnitude. The kernel version establishes a new state-of-the-art for graph classification on a wide range of benchmark datasets, while the neural version shows promising performance on large-scale molecular regression tasks.

1. Introduction

Graph-structured data is ubiquitous across application domains ranging from chemo- and bioinformatics (Barabasi & Oltvai, 2004; Stokes et al., 2020) to image (Simonovsky & Komodakis, 2017) and social network analysis (Easley & Kleinberg, 2010). In recent years, numerous approaches have been proposed for machine learning with graphs—most notably, approaches based on *graph kernels* (Kriege et al., 2020) or using *graph neural networks* (GNNs) (Chami et al., 2020; Gilmer et al., 2017; Grohe, 2020). Here, graph kernels based on the 1-*dimensional Weisfeiler-Leman algorithm* (1-WL) (Weisfeiler & Leman., 1968; Grohe, 2017), and GNNs (Morris et al., 2019; Xu et al., 2019) have recently advanced the state-of-the-art in supervised node and graph classification. Since the 1-WL operates via simple neighborhood aggregation, the purely local nature of these approaches can miss important patterns in the given data. Moreover, they are only applicable to binary structures, and therefore cannot deal with general t-ary structures, e.g., hypergraphs (Zhou et al., 2006), in a straight-forward way.

A provably more powerful algorithm (for graph isomorphism testing) is the k-*dimensional Weisfeiler-Leman algorithm* (k-WL) (Cai et al., 1992; Grohe, 2017; Maron et al., 2019a). The algorithm can capture more global, higher-order patterns by iteratively computing a coloring (or discrete labeling) for k-tuples, instead of single vertices, based on an appropriately defined notion of adjacency between two k-tuples. However, it fixes the cardinality of this neighborhood to $k \cdot n$, where n denotes the number of vertices of a given graph. Hence, the running time of each iteration does not take the *sparsity* of a given graph into account. Further, new neural architectures (Maron et al., 2019a;b) that possess the same power as the k-WL in terms of separating non-isomorphic graphs suffer from the same drawbacks, i.e., they have to resort to dense matrix multiplications. Moreover, when used in a machine learning setting with real-world graphs, the k-WL may capture the isomorphism type, which is the complete structural information inherent in a graph, after only a couple of iterations, which may lead to overfitting, see (Morris et al., 2017). Hence, the task of adapting k-WL for designing practical graph learning algorithms has obvious significance. See Appendix A for an expanded discussion on related work.

Present work. We propose a *local* version of the k-WL, the *local* δ-k-*dimensional Weisfeiler-Leman algorithm* (δ-k-LWL), which considers a subset of the original neighborhood (of a vertex tuple) in each iteration. The cardinality of the *local neighborhood* depends on the sparsity of the graph, i.e., the degrees of the vertices of a given k-tuple. On the theoretical side, we prove that our algorithm is strictly more powerful in distinguishing non-isomorphic graphs compared to the k-WL. On the neural side, we devise a higher-order graph neural

[1]CERC in Data Science for Real-Time Decision-Making, Polytechnique Montréal [2]Department of Computer Science, RWTH Aachen University [3]Department of Computer Science, University of Bonn. Correspondence to: Christopher Morris <christopher.morris@tu-dortmund.de>.

Graph Representation Learning and Beyond (GRL+), ICML 2020 Workshop.

network architecture, the δ-k-LGNN, and show that is has the same expressive power as the δ-k-LWL. Moreover, recent advancements in learning theory for GNNs (Garg et al., 2020) imply that the δ-k-LWL architecture has better generalization abilities compared to dense architectures based on the k-WL. Experimentally, we apply the discrete algorithms (or kernels) and the (local) neural architecture to supervised graph learning, and verify that both are several orders of magnitude faster than the global, discrete algorithms or dense, neural architectures, and prevent overfitting. The discrete algorithms establish a new state-of-the-art for graph classification on a wide range of small- and medium-scale classical datasets. The neural version shows promising performance on large-scale molecular regression tasks.

2. Weisfeiler-Leman: classic and δ-version

Local/global neighbors. Given a k-tuple $\mathbf{v}$ of vertices of a graph G, let $\phi_j(\mathbf{v}, w)$ be the k-tuple obtained by replacing the j^{th}component of $\mathbf{v}$ with the vertex 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 the tuple $\mathbf{w}$ a *j-neighbor* of the tuple $\mathbf{v}$ (and vice-versa). Furthermore, call $\mathbf{w}$ a *local* j-neighbor of $\mathbf{v}$ if w is adjacent to v_j, otherwise call it a *global* neighbor of $\mathbf{v}$.

The k-WL. Given a graph G and an integer $k \geq 0$, the k-WL computes a stable coloring (a mapping $C_\infty \colon V(G)^k \to \mathbb{N}$) for G, via an iterative procedure as follows. The *initial coloring* C_0 of $V(G)^k$ is specified by the isomorphism types of the tuples, i.e., two tuples $\mathbf{v}$ and $\mathbf{w}$ in $V(G)^k$ get a common color iff the mapping $v_i \mapsto w_i$ induces an isomorphism between the labeled subgraphs $G[\mathbf{v}]$ and $G[\mathbf{w}]$. Starting with C_0, successive refinements $C_{i+1} = \widehat{C_i}$ are computed until convergence, i.e., $C_{i+1}(\mathbf{v}) = (C_i(\mathbf{v}), M_i(\mathbf{v}))$, where,

$$M_i(\mathbf{v}) = \big(\{\!\{C_i(\phi_1(\mathbf{v}, w)) \mid w \in V(G)\}\!\}, \ldots, \{\!\{C_i(\phi_k(\mathbf{v}, w)) \mid w \in V(G)\}\!\}\big), \tag{1}$$

is called the *aggregation map*. The k-WL *distinguishes* two graphs G and H if running k-WL on their disjoint union yields disparate colors, hence, certifying their non-isomorphism (see, e.g., (Grohe, 2017) for a detailed treatment).

The δ-k-WL. The δ-k-dimensional *Weisfeiler-Leman algorithm*, denoted by δ-k-WL, is a variant of the classic k-WL which *differentiates* between the local and the global neighbors during neighborhood aggregation (Malkin, 2014). Essentially, δ-k-WL employs the aggregation function $M_i^{\delta,\overline{\delta}}(\cdot)$ instead of $M_i(\cdot)$ above. It replaces a multiset term $\{\!\{C_i(\phi_j(\mathbf{v}, w)) \mid w \in V(G)\}\!\}$ in Equation (1) above by

$$\{\!\{(C_i(\phi_j(\mathbf{v}, w), \mathrm{adj}(\mathbf{v}, \phi_j(\mathbf{v}, w))) \mid w \in V(G)\}\!\},$$

where, the indicator function $\mathrm{adj}(\mathbf{v}, \mathbf{w})$ evaluates to L or G, depending on whether $\mathbf{w}$ is a local or global neighbor of $\mathbf{v}$.

3. Weisfeiler and Leman go sparse

We propose the new *local* δ-k-dimensional Weisfeiler-Leman algorithm (δ-k-LWL). This variant of δ-k-WL considers only local neighbors during the neighborhood aggregation process, and discards any information about the global neighbors. The aggregation function used by the δ-k-WL is

$$M_i^{\delta}(\mathbf{v}) = \big(\{\!\{C_i(\phi_1(\mathbf{v}, w)) \mid w \in N(v_1)\}\!\}, \ldots, \{\!\{C_i(\phi_k(\mathbf{v}, w)) \mid w \in N(v_k)\}\!\}\big), \tag{2}$$

hence considering only the local j-neighbors of the tuple $\mathbf{v}$ in each iteration.

We also propose the δ-k-LWL$^+$, a minor variation of the δ-k-LWL above, which preserves certain global information in order to achieve strong theoretical guarantees with asymptotically identical scalability. Essentially, we use a term $(C_i(\phi_1(\mathbf{v}, w)), \#_i^1(\mathbf{v}, \phi_1(\mathbf{v}, w)))$ instead of $C_i(\phi_j(\mathbf{v}, w))$, where the counter

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

counts the number of j-neighbors $\mathbf{w}$ of $\mathbf{v}$ such that $\mathbf{w}$ has the same color as $\mathbf{x}$ after i rounds of the δ-k-LWL$^+$. Here, $\mathbf{w} \sim_j \mathbf{v}$ denotes that $\mathbf{w}$ is j-neighbor of $\mathbf{v}$, for j in $[k]$.

Theoretical guarantees. Let A_1 and A_2 denote two k-WL-variants. We write $A_1 \sqsubseteq A_2$ (A_1 is as *powerful* as A_2) if A_1 distinguishes between all non-isomorphic pairs A_2 does, and $A_1 \equiv A_2$ if both directions hold. The corresponding strict relation is denoted by $\sqsubset$. The following theorem, which is our main theoretical result, shows that the δ-k-LWL$^+$ is equivalent in power to the δ-k-WL.

Theorem 1. For the class of connected graphs, the following holds for all $k \geq 2$:

$$\delta\text{-}k\text{-LWL}^+ \equiv \delta\text{-}k\text{-WL}.$$

We also prove that δ-k-WL $\sqsubset$ k-WL, which proves the desired result, i.e., δ-k-LWL$^+$ $\sqsubset$ k-WL (for connected graphs). We remark that, in general, δ-k-LWL$^+$ takes a larger number of rounds to converge. This possibly slower convergence is the key to tackling the overfitting problem associated with the classic k-WL. See Appendix F for a discussion on practicality and remaining challenges, and Figure 2 for an overview of the theoretical results.

4. Higher-order graph kernels and neural networks

Kernels. After running the δ-k-LWL (or δ-k-LWL$^+$), 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 σ in Σ there is

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	NCI1	NCI109	PTC_FM	PROTEINS	REDDIT-BINARY
Baseline	GR	29.8 ±1.0	59.5 ±0.4	40.6 ±0.5	66.3 ±0.2	66.7 ±0.2	62.3 ±0.9	71.6 ±0.2	60.0 ±0.2
	SP	42.3 ±1.3	59.2 ±0.3	39.6 ±0.4	74.5 ±0.3	73.4 ±0.1	63.2 ±0.6	76.4 ±0.4	84.7±0.2
	1-WL	53.4 ±1.4	72.4 ±0.5	50.6 ±0.6	85.1 ±0.2	85.2 ±0.2	62.9 ±1.6	73.7 ±0.5	75.3 ±0.3
	WLOA	59.7 ±1.2	73.1 ±0.7	50.3 ±0.6	85.6 ±0.2	86.0 ±0.3	63.7 ±0.7	73.7 0.5±	88.7 ±0.2
Neural	Gin-0	39.6 ±1.3	72.8 ±0.9	50.0 ±0.1	78.5 ±0.5	77.1 ±0.6	58.0 ±1.4	71.7 ±0.9	90.7 ±0.9
	Gin-ε	38.7 ±2.1	73.0 ±1.0	49.8 ±0.6	78.8 ±0.3	77.2 ±0.3	58.7 ±1.7	70.4 ±1.2	89.4 ±1.2
Global	2-WL	38.9 ±0.8	69.2 ±0.6	48.0 ±0.5	67.5 ±0.3	68.3 ±0.2	64.3 ±0.6	75.3 ±0.3	OOM
	3-WL	45.9 ±0.8	69.2 ±0.4	47.9 ±0.7	OOT	OOT	64.4 ±0.6	OOM	OOM
	δ-2-WL	39.1 ±1.1	69.4 ±0.7	48.0 ±0.4	67.4 ±0.3	68.3 ±0.3	**64.5** ±0.4	75.2 ±0.5	OOM
	δ-3-WL	45.9 ±0.9	69.1 ±0.6	47.9 ±0.8	OOT	OOT	64.4 ±0.6	OOM	OOM
Local	δ-2-LWL	57.7 ±1.0	73.3 ±0.7	50.9 ±0.6	85.4 ±0.2	84.8 ±0.2	62.7 ±1.3	74.5 ±0.6	90.0 ±0.2
	δ-2-LWL$^+$	57.0 ±0.8	78.9 ±0.6	**64.0** ±0.4	**91.8** ±0.2	**90.8** ±0.2	62.7 ±1.4	**82.6** ±0.4	**91.5** ±0.2
	δ-3-LWL	**60.4** ±0.8	73.5 ±0.5	49.6 ±0.7	84.0 ±0.3	83.0 ±0.3	62.6 ±1.2	OOM	OOM
	δ-3-LWL$^+$	58.9 ±1.1	**80.6** ±0.5	60.3 ±0.4	83.9 ±0.3	82.9 ±0.3	62.4 ±1.2	OOM	OOM

an entry containing the number of nodes or k-tuples that are colored with σ.

Neural architectures. Although the discrete kernels above are quite powerful, they are limited due to their fixed feature construction scheme, hence suffering from poor adaption to the learning task at hand and from the inability to handle continuous node and edge labels in a meaningful way. This motivates our definition of a new neural architecture, called *local* δ-k-GNN (δ-k-LGNN). Given a node labeled graph G, let each tuple $\mathbf{v}$ in $V(G)^k$ be annotated with an initial feature $f^{(0)}(\mathbf{v})$ determined by its isomorphism type. In each layer $t > 0$, we compute a new feature $f^{(t)}(\mathbf{v})$ as

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

in $\mathbb{R}^{1\times e}$ for a tuple $\mathbf{v}$, where $W^{(t)}_1$ and $W^{(t)}_2$ are learnable parameter matrices from $\mathbb{R}^{d\times e}$ and σ denotes a component-wise non-linear function, e.g., a sigmoid or a ReLU.[1]. Moreover, $f^{W_2}_{\text{mrg}}$ and the permutation-invariant $f^{W_1}_{\text{agg}}$ can be arbitrary differentiable functions, responsible for merging and aggregating the relevant feature information, respectively. Initially, we set $f^{(0)}(\mathbf{v})$ to a one-hot encoding of the (labeled) isomorphism type of $G[\mathbf{v}]$. Note that we can naturally handle discrete node and edge labels as well as directed graphs. The following result demonstrates the expressive power of δ-k-GNN, in terms of distinguishing non-isomorphic graphs.

Theorem 2. For all graphs, and all $k \geq 1$:

$$\delta\text{-}k\text{-LGNN} \equiv \delta\text{-}k\text{-LWL}.$$

Moreover, δ-k-GNN inherits the main strength of δ-k-LWL, i.e., it can be implemented using sparse matrix multiplication. Note that it is not possible to come up with an architecture, i.e., instantiations of $f^{W_1}_{\text{mrg}}$ and $f^{W_2}_{\text{agg}}$, such that it becomes more powerful than the δ-k-LWL, see (Morris et al., 2019). However, all results from the previous section can be lifted to the neural setting. That is, one can derive neural architectures based on the δ-k-LWL$^+$, δ-k-WL, and k-WL, called δ-k-LGNN$^+$, δ-k-GNN, and k-WL-GNN, respectively, and prove results analogous to Theorem 2. See Appendix E for a discussion on the generalization capabilities of the (local) neural architecture and incoperating continuous information.

[1]For clarity of presentation we omit biases.

5. Experimental evaluation

We investigate the benefits of the local, sparse algorithms, both kernel and neural architectures, compared to the global, dense algorithms, and standard kernel and GNN baselines, via the following three questions. See Appendix G for an expanded experimental evaluation and details on datasets, hyperparameters, and evaluation protocols. The source code and evaluation scripts will be made available at `www.github.com/chrsmrrs/sparsewl`.

Q1 Do the local algorithms, both kernel and neural architectures, lead to improved classification and regression scores on real-world benchmark datasets compared to global, dense algorithms and standard baselines?

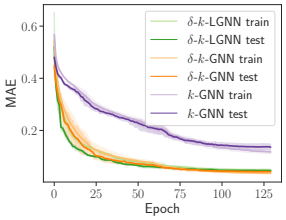
Q2 Does the δ-k-LWL$^+$ lead to improved classification accuracies compared to the δ-k-LWL? Does it lead to higher computation times?

Q3 How much do the local algorithms speed up the computation time compared to the non-local algorithms or dense neural architectures?

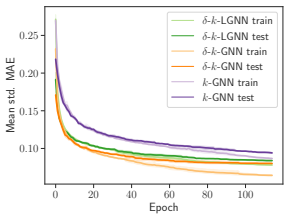
Datasets To evaluate kernels, we use the following, well-known, small-scale ENZYMES, IMDB-BINARY, IMDB-MULTI, NCI1, NCI109, PTC_FM, PROTEINS, and REDDIT-BINARY datasets. To show that our kernels also scale to larger datasets, we additionally used the mid-scale YEAST, YEASTH, UACC257, UACC257H, OVCAR-8, OVCAR-8H datasets. For the neural architectures, we used the large-scale molecular regression datasets ZINC and ALCHEMY. To further compare to the (hierarchical) k-GNN (Morris et al., 2019)

	Method	QM9
Baseline	GINE-ε	0.081 $\pm$0.003
	MPNN	0.034 $\pm$0.001
	1-2-GNN	0.068 $\pm$0.001
	1-3-GNN	0.088 $\pm$0.007
	1-2-3-GNN	0.062 $\pm$0.001
	3-IGN	0.046 $\pm$0.001
	δ-2-LGNN	**0.029** $\pm$0.001

(a) Mean std. MAE on QM9



(b) ZINC



(c) ALCHEMY

Figure 1: Results for neural architectures.

and k-IGN (Maron et al., 2019a), and show the benefits of our architecture in presence of continuous features, we used the QM9 regression dataset.[2] All datasets can be obtained from `www.graphlearning.io`.

Kernels and Networks We implemented the δ-k-LWL, δ-k-LWL$^+$, δ-k-WL, and k-WL kernel for k in $\{2, 3\}$ and compare them to the standard baselines for graph kernels. We also implemented the neural architectures of Section 4, δ-k-LGNN, δ-k-GNN, k-WL-GNN, and used the GIN and GIN-ε architecture (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).[3]

Results and Discussion In the following we answer questions **Q1** to **Q3**.

A1 *Kernels* See Table 1 and Table 5 in the appendix. The local algorithm, for $k = 2$ and 3, severely improves the classification accuracy compared to the k-WL and the δ-k-WL (in some cases, by $> 15\%$).
Neural architectures See Figure 1 and Table 6 in the appendix. On the ZINC and ALCHEMY datasets, the δ-2-LGNN is on par or slightly worse than the δ-2-GNN. Hence, this is in contrast to the kernel variant. We assume that this is due to the δ-2-GNN being more flexible than its kernel variant in weighing the importance of global and local neighbors. This is further highlighted by the worse performance of the 2-WL-GNN, which even performs worse than GINE-ε on the ZINC dataset. On the QM9 dataset, see Figure 1a, the δ-2-LGNN performs better than the higher-order methods from (Maron et al., 2019a; Morris et al., 2019) while being on par with the MPNN architecture. We note here that the MPNN was specifically tuned to the QM9 dataset, which is not the case for the δ-2-LGNN (and the other higher-order architectures).
A2 See Table 1. The δ-2-LWL$^+$ improves over the δ-2-LWL on all datasets excluding ENZYMES. For example, on IMDB-BINARY, IMDB-MULTI, NCI1, NCI109, and PROTEINS the algorithm achieves an improvement over of 4%, respectively, achieving a new state-of-the-art. The computation times are only increased slightly. Similar results can be observed on the medium-scale datasets, see Table 5 in the appendix.

[2]We opted for comparing on the QM9 dataset to ensure a fair comparison concerning hyperparameter selection.

[3]We opted for not implementing the δ-k-LGNN$^+$ as it would involve precomputing #, see Appendix E.

Table 2: Average speed up ratios over all epochs (training and testing) for local, neural architecure.

	Method	Dataset	
		ZINC (10k)	ALCHEMY (10K)
Baseline	GINE-ε	0.2	0.4
	2-WL-GNN	2.2	1.1
	δ-2-GNN	2.5	1.7
	δ-2-LGNN	1.0	1.0

A3 *Kernels:* See Tables 7 and 8 in the appendix. The local algorithm severely speeds up the computation time compared to the δ-k-WL and the k-WL for $k = 2$ and 3, demonstrating the suitability of the local algorithms for practical applications.
Neural architectures: See Table 2. The local algorithm severely speeds up the computation time of training and testing. Especially, on the ZINC dataset, which has larger graphs compared to the ALCHEMY dataset, the δ-2-LGNN achieves a computation time that is more than two times lower compared to the δ-2-GNN and the 2-WL-GNN.

6. Conclusion

We verified that our local, sparse algorithms lead to vastly reduced computation times compared to their global, dense counterparts while establishing new state-of-the-art results on a wide range of benchmark datasets. Moreover, we also showed strong theoretical guarantees on the expressiveness of these algorithms. Future work includes a more fine-grained analysis of the proposed algorithm, e.g., moving away from the restrictive graph isomorphism objective and deriving a deeper understanding of the neural architecture's capabilities when optimized with stochastic gradient descent.

References

Abu-El-Haija, S., Perozzi, B., Kapoor, A., Alipourfard, N., Lerman, K., Harutyunyan, H., Steeg, G. V., and Galstyan, A. Mixhop: Higher-order graph convolutional architectures via sparsified neighborhood mixing. In *International Conference on Machine Learning*, pp. 21–29, 2019.

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.

Arora, S., Du, S. S., Hu, W., Li, Z., Salakhutdinov, R., and Wang, R. On exact computation with an infinitely wide neural net. In *Advances in Neural Information Processing Systems*, pp. 8139–8148, 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.

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

Bai, S., Zhang, F., and Torr, P. H. S. Hypergraph convolution and hypergraph attention. *CoRR*, abs/1901.08150, 2019.

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.

Berkholz, C., Bonsma, P. S., and Grohe, M. Tight lower and upper bounds for the complexity of canonical colour refinement. In *Annual European Symposium on Algorithms*, pp. 145–156. Springer, 2013.

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

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.

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.

Cangea, C., Velickovic, P., Jovanovic, N., Kipf, T., and Liò, P. Towards sparse hierarchical graph classifiers. *CoRR*, abs/1811.01287, 2018.

Cao, S., Lu, W., and Xu, Q. GraRep: Learning graph representations with global structural information. In *ACM International Conference on Information and Knowledge Management*, pp. 891–900, 2015.

Chami, I., Ying, Z., Ré, C., and Leskovec, J. Hyperbolic graph convolutional neural networks. In *Advances in Neural Information Processing Systems*, pp. 4869–4880, 2019.

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.

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, J., Ma, T., and Xiao, C. Fastgcn: Fast learning with graph convolutional networks via importance sampling. In *International Conference on Learning Representation*, 2018a.

Chen, J., Zhu, J., and Song, L. Stochastic training of graph convolutional networks with variance reduction. In *International Conference on Machine Learning*, pp. 941–949, 2018b.

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.

Corso, G., Cavalleri, L., Beaini, D., Liò, P., and Velickovic, P. Principal neighbourhood aggregation for graph nets. *CoRR*, abs/2004.05718, 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.

Dobson, P. D. and Doig, A. J. Distinguishing enzyme structures from non-enzymes without alignments. *Journal of Molecular Biology*, 330(4):771 – 783, 2003.

Du, S. S., Hou, K., Salakhutdinov, R. R., Poczos, B., Wang, R., and Xu, K. Graph Neural Tangent Kernel: Fusing graph neural networks with graph kernels. In *Advances in Neural Information Processing Systems*, pp. 5723–5733, 2019.

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.

Dwivedi, V. P., Joshi, C. K., Laurent, T., Bengio, Y., and Bresson, X. Benchmarking graph neural networks. *CoRR*, abs/2003.00982, 2020.

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

Fan, R.-E., Chang, K.-W., Hsieh, C.-J., Wang, X.-R., and Lin, C.-J. LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874, 2008.

Fey, M. and Lenssen, J. E. Fast graph representation learning with PyTorch Geometric. *CoRR*, abs/1903.02428, 2019.

Fey, M., Lenssen, J. E., Weichert, F., and Müller, H. SplineCNN: Fast geometric deep learning with continuous B-spline kernels. In *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 869–877, 2018.

Flam-Shepherd, D., Wu, T., Friederich, P., and Aspuru-Guzik, A. Neural message passing on high order paths. *CoRR*, abs/2002.10413, 2020.

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

Gao, H. and Ji, S. Graph U-Nets. In *International Conference on Machine Learning*, pp. 2083–2092, 2019.

Garg, V. K., Jegelka, S., and Jaakkola, T. S. Generalization and representational limits of graph neural networks. *CoRR*, abs/2002.06157, 2020.

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

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

Grohe, M., Kersting, K., Mladenov, M., and Selman, E. Dimension reduction via colour refinement. In *European Symposium on Algorithms*, pp. 505–516, 2014.

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.

Huang, W., Zhang, T., Rong, Y., and Huang, J. Adaptive sampling towards fast graph representation learning. In *Advances in Neural Information Processing Systems*, pp. 4563–4572, 2018.

Immerman, N. and Lander, E. *Describing Graphs: A First-Order Approach to Graph Canonization*, pp. 59–81. Springer, 1990.

Jacot, A., Hongler, C., and Gabriel, F. Neural Tangent kernel: convergence and generalization in neural networks. In *Advances in Neural Information Processing Systems*, pp. 8580–8589, 2018.

Jin, W., Barzilay, R., and Jaakkola, T. S. Junction tree variational autoencoder for molecular graph generation. In *International Conference on Machine Learning*, pp. 2328–2337, 2018.

Jin, Y., Song, G., and Shi, C. GraLSP: Graph neural networks with local structural patterns. *CoRR*, abs/1911.07675, 2019.

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

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

Kiefer, S. and McKay, B. D. The iteration number of colour refinement. *CoRR*, abs/2005.10182, 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.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

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

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

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., 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., Neumann, M., Morris, C., Kersting, K., and Mutzel, P. A unifying view of explicit and implicit feature maps of graph kernels. *Data Minining and Knowledge Discovery*, 33(6):1505–1547, 2019.

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

Lee, J. B., Rossi, R. A., Kong, X., Kim, S., Koh, E., and Rao, A. Graph convolutional networks with motif-based attention. In *28th ACM International Conference on Information*, pp. 499–508, 2019.

Lichter, M., Ponomarenko, I., and Schweitzer, P. Walk refinement, walk logic, and the iteration number of the Weisfeiler-Leman algorithm. In *34th Annual ACM/IEEE Symposium on Logic in Computer Science*, pp. 1–13, 2019.

Loukas, A. What graph neural networks cannot learn: depth vs width. In *International Conference on Learning Representations*, 2020.

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. In *Advances in Neural Information Processing Systems*, pp. 2153–2164, 2019a.

Maron, H., Ben-Hamu, H., Shamir, N., and Lipman, Y. Invariant and equivariant graph networks. In *International Conference on Learning Representations*, 2019b.

Meng, C., Mouli, S. C., Ribeiro, B., and Neville, J. Subgraph pattern neural networks for high-order graph evolution prediction. In *AAAI Conference on Artificial Intelligence*, pp. 3778–3787, 2018.

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.

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. IEEE, 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.

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

Murphy, R. L., Srinivasan, B., Rao, V. A., and Ribeiro, B. Janossy pooling: Learning deep permutation-invariant functions for variable-size inputs. In *International Conference on Learning Representations*, 2019b.

Niepert, M., Ahmed, M., and Kutzkov, K. Learning convolutional neural networks for graphs. In *International Conference on Machine Learning*, pp. 2014–2023, 2016.

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.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, pp. 8024–8035, 2019.

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.

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.

Rong, Y., Huang, W., Xu, T., and Huang, J. DropEdge: Towards deep graph convolutional networks on node classification. In *International Conference on Learning Representations*, 2020.

Sato, R., Yamada, M., and Kashima, H. Approximation ratios of graph neural networks for combinatorial problems. In *Neural Information Processing Systems*, pp. 4083–4092, 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.

Schomburg, I., Chang, A., Ebeling, C., Gremse, M., Heldt, C., Huhn, G., and Schomburg, D. BRENDA, the enzyme database: updates and major new developments. *Nucleic acids research*, 32(Database issue):D431—3, January 2004.

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.

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.

Togninalli, M., Ghisu, 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.

Velickovic, P., Cucurull, G., Casanova, A., Romero, A., Liò, 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*, pp. 88–98, 2017.

Verma, S. and Zhang, Z. Stability and generalization of graph convolutional neural networks. In *ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1539–1548, 2019.

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

Wale, N., Watson, I. A., and Karypis, G. Comparison of descriptor spaces for chemical compound retrieval and classification. *Knowledge and Information Systems*, 14(3): 347–375, 2008.

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.

Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., and Yu, P. S. A comprehensive survey on graph neural networks. *CoRR*, abs/1901.00596, 2019.

Xu, K., Li, C., Tian, Y., Sonobe, T., Kawarabayashi, K., and Jegelka, S. Representation learning on graphs with jumping knowledge networks. In *International Conference on Machine Learning*, pp. 5453–5462, 2018.

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

Yadati, N., Nimishakavi, M., Yadav, P., Nitin, V., Louis, A., and Talukdar, P. P. HyperGCN: A new method for training graph convolutional networks on hypergraphs. In *Advances in Neural Information Processing Systems*, pp. 1509–1520, 2019.

Yan, X., Cheng, H., Han, J., and Yu, P. S. Mining significant graph patterns by leap search. In *ACM SIGMOD International Conference on Management of Data*, pp. 433–444, 2008.

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 *ACM SIGKDD International Conference on Knowledge Discovery and Data*, pp. 1365–1374. ACM, 2015b.

Ying, R., You, J., Morris, C., Ren, X., Hamilton, W. L., and Leskovec, J. Hierarchical graph representation learning with differentiable pooling. In *Advances in Neural Information Processing Systems*, pp. 4800–4810, 2018.

You, J., Ying, R., and Leskovec, J. Position-aware graph neural networks. In *International Conference on Machine Learning*, pp. 7134–7143, 2019.

Zhang, M., Cui, Z., Neumann, M., and Yixin, C. An end-to-end deep learning architecture for graph classification. In *AAAI Conference on Artificial Intelligence*, pp. 4428–4435, 2018.

Zhang, R., Zou, Y., and Ma, J. Hyper-SAGNN: A self-attention based graph neural network for hypergraphs. In *International Conference on Learning Representations*, 2020.

Zhou, D., Huang, J., and Schölkopf, B. Learning with hypergraphs: Clustering, classification, and embedding. In *Advances in Neural Information Processing Systems*, pp. 1601–1608, 2006.

Zhou, J., Cui, G., Zhang, Z., Yang, C., Liu, Z., Wang, L., Li, C., and Sun, M. Graph neural networks: A review of methods and applications. *CoRR*, abs/1812.08434, 2018.

Appendix

A. Expanded related work

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

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., 2019) and kernels based on shortest paths (Borgwardt & Kriegel, 2005). More recently, graph kernels' developments 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. (Morris et al., 2017) devised a local, set-based variant of the k-WL. However, the approach is (provably) weaker than the tuple-based algorithm, and they do not prove convergence to the original algorithm. Yanardag & Vishwanathan 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 works focus 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 (Togninalli et al., 2019; Rieck et al., 2019). For a theoretical investigation of graph kernels, see (Kriege et al., 2018), for a thorough survey of graph kernels, see (Kriege et al., 2020).

GNNs. Recently, graph neural networks (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; Fey et al., 2018; Hamilton et al., 2017; Velickovic et al., 2018), and the spectral approaches proposed in, e.g., (Bruna et al., 2014; Defferrard et al., 2016; Kipf & Welling, 2017; Monti et al., 2017)—all of which descend from early work in (Kireev, 1995; Merkwirth & Lengauer, 2005; Sperduti & Starita, 1997; Scarselli et al., 2009). Recent extensions and improvements to the GNN framework include approaches to incorporate different local structures (around subgraphs), e.g., (Abu-El-Haija et al., 2019; Flam-Shepherd et al., 2020; Jin et al., 2019; Niepert et al., 2016; Xu et al., 2018), novel techniques for pooling node representations in order perform graph classification, e.g., (Cangea et al., 2018; Gao & Ji, 2019; Ying et al., 2018; Zhang et al., 2018), incorporating distance information (You et al., 2019), and non-euclidian geometry approaches (Chami et al., 2019). Moreover, recently empirical studies on neighborhood aggregation functions for continuous vertex features (Corso et al., 2020), edge-based GNNs leveraging physical knowledge (Anderson et al., 2019; Klicpera et al., 2020), and sparsification methods (Rong et al., 2020) emerged. Loukas (Loukas, 2020) and Sato et al. studied the limits of GNNs when applied to combinatorial problems. A survey of recent advancements in GNN techniques can be found, e.g., in (Chami et al., 2020; Wu et al., 2019; Zhou et al., 2018). Garg et al. (Garg et al., 2020) and Verma & Zhang (Verma & Zhang, 2019) studied the generalization abilities of GNNs, and (Du et al., 2019) related wide GNNs to a variant of the neural tangent kernel (Arora et al., 2019; Jacot et al., 2018). Murphy et al. (Murphy et al., 2019b;a) and Sato et al. (Sato et al., 2020) extended the expressivity of GNNs by considering all possible permutations of a graph's adjacency matrix, or adding random node features, respectivley.

Recently, connections to Weisfeiler-Leman type algorithms have been shown (Barceló et al., 2020; Chen et al., 2019b; 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. Morris et al. (Morris et al., 2019) also 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, which is (provably) weaker than the variant considered here. Later, this was refined in (Maron et al., 2019a) by introducing *k-order invariant graph networks* (k-IGN), based on Maron et al. (Maron et al., 2019b), and references therein, which are equivalent to the folklore variant of the k-WL (Grohe, 2017) in terms of distinguishing non-isomorphic graphs. However, k-IGN may not scale since they rely on dense linear algebra routines. Chen et al. (Chen et al., 2019b) connect the theory of universal approximation of permutation-invariant functions and the graph isomorphism viewpoint and introduce a variation of the 2-WL, which is more powerful than the former. Our comprehensive treatment of higher-order, sparse, neural networks for arbitrary k subsumes all of the algorithms and neural architectures mentioned above.

Finally, there exists a new line of work focusing on extending GNNs to hypergraphs, see, e.g., (Bai et al., 2019; Yadati et al., 2019; Zhang et al., 2020), and a line of work in the data mining community incorporating global or higher-order information into graph or node embeddings, see, e.g., (Cao et al., 2015; Lee et al., 2019; Meng et al., 2018).

Theory. The Weisfeiler-Leman algorithm constitutes one of the earliest approaches to isomorphism testing (Weisfeiler, 1976; Weisfeiler & Leman., 1968), having been heavily investigated by the theory community over the last few decades (Grohe et al., 2014). 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 (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 (Immerman & Lander, 1990), Cai et al. showed that for each k there exists a pair of non-isomorphic graphs of size $\mathcal{O}(k)$ each that cannot be distinguished by the k-WL. Grohe et al. (Grohe et al., 2014) gives a thorough overview 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 for $k = 1$ (Berkholz et al., 2013; Kiefer & McKay, 2020), and the number of iterations for the folklore $k = 2$ (Kiefer & Schweitzer, 2016; Lichter et al., 2019) have been shown. For $k = 1$ and 2, Arvind et al. (Arvind et al., 2019) studied the abilities of the (folklore) k-WL to detect and count fixed subgraphs, extending the work of Fürer (Fürer, 2017). The former was refined in (Chen et al., 2020). The algorithm (for logarithmic k) plays a prominent role in the recent result of Babai (Babai, 2016) improving the best-known running time for the graph isomorphism problem. Recently, Grohe et al. (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 a recent survey of Grohe (Grohe, 2020).

B. Expanded preliminaries

We briefly describe the Weisfeiler-Leman algorithm and, along the way, introduce our notation. We also state a variant of the algorithm, introduced in (Malkin, 2014). As usual, let $[n] = \{1, \dots, n\} \subset \mathbb{N}$ for $n \geq 1$, and let $\{\!\{\dots\}\!\}$ denote a multiset.

Graphs. A *graph* G is a pair (V, E) with a *finite* set of *vertices* V and a set of *edges* $E \subseteq \{\{u, v\} \subseteq V \mid u \neq v\}$. We denote the set of vertices and the set of edges of G by $V(G)$ and $E(G)$, respectively. 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 $l\colon V(G) \cup E(G) \to \Sigma$, where Σ is some finite alphabet. Then $l(v)$ is a *label* of v for v in $V(G) \cup E(G)$. The *neighborhood* of v in $V(G)$ is denoted by $\delta(v) = N(v) = \{u \in V(G) \mid (v, u) \in E(G)\}$. Moreover, its complement $\bar{\delta}(v) = \{u \in V(G) \mid (v, u) \notin E(G)\}$. Let $S \subseteq V(G)$ then $G[S] = (S, E_S)$ is the *subgraph induced* by S with $E_S = \{(u, v) \in E(G) \mid u, v \in S\}$. A *tree* is a connected graph without cycles. A *rooted tree* is a tree with a designated vertex called *root* in which the edges are directed in such a way that they point away from the root. Let p be a vertex in a directed tree then we call its out-neighbors *children* with parent p.

We say that two graphs G and H are *isomorphic* if there exists an edge preserving bijection $\varphi\colon V(G) \to V(H)$, i.e., (u, v) is in $E(G)$ if and only if $(\varphi(u), \varphi(v))$ is in $E(H)$. If G and H are isomorphic, we write $G \simeq H$ and call φ 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 $l(v) = l(\varphi(v))$ for v in $V(G)$ and $l((u, v)) = l((\varphi(u), \varphi(v)))$ 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 components of $\mathbf{v}$, where the vertices are labeled with integers from $\{1, \dots, k\}$ corresponding to indices of $\mathbf{v}$.

Kernels. A *kernel* on a non-empty set $\mathcal{X}$ is a positive semidefinite function $k\colon \mathcal{X} \times \mathcal{X} \to \mathbb{R}$. Equivalently, a function k 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 $\mathcal{X}$. Let $\mathcal{G}$ be the set of all graphs, then a (positive semidefinite) function $\mathcal{G} \times \mathcal{G} \to \mathbb{R}$ is called a *graph kernel*.

C. Vertex refinement algorithms (expanded)

Let k be a fixed positive integer. As usual, let $V(G)^k$ denote the set of k-tuples of vertices of 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 isomorphism types of the tuples, i.e., two tuples $\mathbf{v}$ and $\mathbf{w}$ in $V(G)^k$ get a common color iff the mapping $v_i \to w_i$ induces an isomorphism between the labeled subgraphs $G[\mathbf{v}]$ and $G[\mathbf{w}]$. A *color class* corresponding to a color c is the set of all tuples colored c, i.e., the set $C^{-1}(c)$.

The *neighborhood* of a vertex tuple $\mathbf{v}$ in $V(G)^k$ is defined as follows. For j in $[k]$, let $\phi_j(\mathbf{v}, w)$ be the k-tuple obtained by

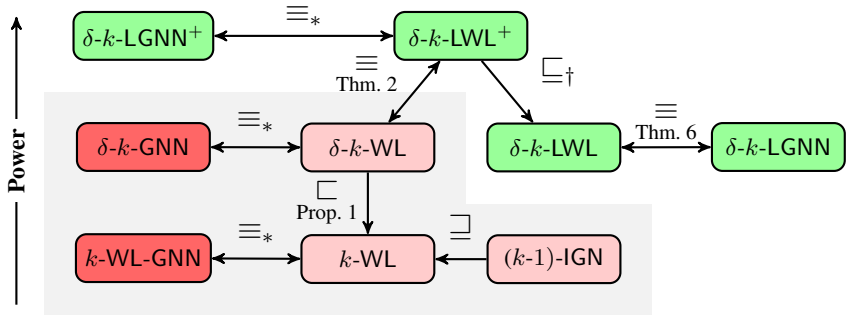


Figure 2: Overview of the power of proposed algorithms and neural architectures. The green and dark red nodes represent algorithms proposed in the present work. The grey region groups dense algorithms and neural architectures. $_*$—Follows directly from the proof of Theorem 2. $A \sqsubseteq B$ ($A \sqsubset B$, $A \equiv B$): algorithm A is more powerful (strictly more powerful, equally powerful) than B, $_\dagger$—Follows by definition, strictness open.

replacing the j^{th} component of $\mathbf{v}$ with the vertex 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 $\mathbf{v}$ a *j-neighbor* of $\mathbf{w}$. The neighborhood of $\mathbf{v}$ is thus defined as the set of all tuples $\mathbf{w}$ such that $\mathbf{w} = \phi_j(\mathbf{v}, w)$ for some j in $[k]$ and 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 $\mathbf{v}$ as a multiset $S_j = \{\!\{C(\phi_j(\mathbf{v}, w)) \mid w \in V(G)\}\!\}$. Then, for a tuple $\mathbf{v}$, define

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

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. Observe that the new color $\widehat{C}(\mathbf{v})$ of $\mathbf{v}$ is solely dictated by the color histogram of its neighborhood. 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.[4] 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 $C_\ell = \widehat{C_\ell}$. In the end, 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

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

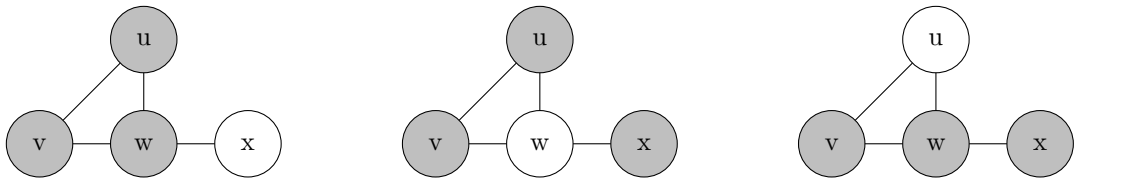
i.e., there exist an unequal number of c-colored tuples in $V(G)^k$ and $V(H)^k$. Hence, two graphs distinguished by the k-WL must be non-isomorphic.

In fact, there exist several variants of the above defined k-WL. These variants result from the application of different aggregation maps $M(\cdot)$. 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 a well-studied variant of the k-WL (see, e.g., (Cai et al., 1992)), commonly known as "folklore" k-WL in machine learning literature. It holds that the k-WL using Equation (3) is as powerful as the folklore $(k-1)$-WL (Grohe & Otto, 2015).

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



(a) Underlying graph G, with tuple (u, v, w)

(b) (u, v, x) is a local 3-neighbor of (u, v, w)

(c) (x, v, w) is a global 1-neighbor of (u, v, w)

Figure 3: Illustration of the local and global neighborhood of the 3-tuple (u, v, w).

C.1. δ-Weisfeiler-Leman algorithm

Let $\mathbf{w} = \phi_j(\mathbf{v}, w)$ be a j-neighbor of $\mathbf{w}$. Call $\mathbf{v}$ a *local* j-neighbor of $\mathbf{w}$ if w is adjacent to the replaced vertex v_j. Otherwise, call $\mathbf{v}$ a *global* j-neighbor of $\mathbf{w}$. Figure 3 illustrates this definition for a 3-tuple (u, v, w). For tuples $\mathbf{v}$ and $\mathbf{w}$ in $V(G)^k$, the function

$$\text{adj}((\mathbf{v}, \mathbf{w})) = \begin{cases} \text{L} & \text{if } \mathbf{w} \text{ is a local neighbor of } \mathbf{u} \\ \text{G} & \text{if } \mathbf{w} \text{ is a global neighbor of } \mathbf{u} \end{cases}$$

indicates whether $\mathbf{w}$ is a local or global neighbor of $\mathbf{u}$.

The *δ-Weisfeiler-Leman algorithm*, denoted by δ-k-WL, is a variant of the classic k-WL which *differentiates* between the local and the global neighbors during neighborhood aggregation (Malkin, 2014). Formally, the δ-k-WL algorithm refines a coloring C_i (obtained after i rounds) via the aggregation function

$$\begin{aligned} M_i^{\delta,\overline{\delta}}(\mathbf{v}) = \big(&\{\!\{(C_i(\phi_1(\mathbf{v}, w), \text{adj}(\mathbf{v}, \phi_1(\mathbf{v}, w))) \mid w \in V(G)\}\!\}, \ldots, \\ &\{\!\{(C_i(\phi_k(\mathbf{v}, w), \text{adj}(\mathbf{v}, \phi_k(\mathbf{v}, w))) \mid w \in V(G)\}\!\}\big), \end{aligned} \tag{4}$$

instead of the k-WL aggregation specified by Equation (3). We define the 1-WL to be the δ-1-WL, which is commonly known as color refinement or naive vertex classification.

Comparing k-WL variants. Given that there exist several variants of k-WL, corresponding to different aggregation maps $M(\cdot)$, it is natural to ask whether they are equivalent in power, vis-a-vis distinguishing non-isomorphic graphs. Let A_1 and A_2 denote two vertex refinement algorithms, we write $A_1 \sqsubseteq A_2$ if A_1 distinguishes between all non-isomorphic pairs A_2 does, and $A_1 \equiv A_2$ if both directions hold. The corresponding strict relation is denoted by $\sqsubset$.

The following result relates the power of k-WL and δ-k-WL. Since for a graph $G = (V, E)$, $M_i^{\delta,\overline{\delta}}(\mathbf{v}) = M_i^{\delta,\overline{\delta}}(\mathbf{w})$ implies $M_i(\mathbf{v}) = M_i(\mathbf{w})$ for all $\mathbf{v}$ and $\mathbf{w}$ in $V(G)^k$ and $i \geq 0$, it immediately follows that δ-k-$\mathsf{WL} \sqsubseteq k$-$\mathsf{WL}$. For $k = 1$, these two algorithms are equivalent by definition. For $k \geq 2$, this relation can be shown to be strict (the proof is deferred to the appendix).

Proposition 3. For all graphs and $k \geq 2$, the following holds:

$$\delta\text{-}k\text{-}\mathsf{WL} \sqsubset k\text{-}\mathsf{WL}.$$

D. Local δ-k-dimensional Weisfeiler-Leman algorithm (Expanded)

In this section, we define the new *local δ-k-dimensional Weisfeiler-Leman algorithm* (δ-k-LWL). This variant of δ-k-WL 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 (obtained after i rounds) via the aggregation function,

$$M_i^{\delta}(\mathbf{v}) = \big(\{\!\{C_i(\phi_1(\mathbf{v}, w)) \mid w \in N(v_1)\}\!\}, \ldots, \{\!\{C_i(\phi_k(\mathbf{v}, w)) \mid w \in N(v_k)\}\!\}\big), \tag{5}$$

instead of Equation (4). That is, the algorithm only considers the local j-neighbors of the vertex $\mathbf{v}$ in each iteration. Therefore, the indicator function adj used in Equation (4) is trivially equal to L here, and is hence omitted. The coloring function for the δ-k-LWL is defined by

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

We also define δ-k-LWL^+, a minor variation of δ-k-LWL. Later, we will show that δ-k-LWL^+ is equivalent in power to δ-k-WL (Theorem 4). Formally, the δ-k-LWL^+ algorithm refines a coloring C_i (obtained after i rounds) via the aggregation function,

$$M^{\delta,+}(\mathbf{v}) = \big(\{\!\{(C_i(\phi_1(\mathbf{v},w)), \#_i^1(\mathbf{v},\phi_1(\mathbf{v},w))) \mid w \in N(v_1)\}\!\}, \ldots, \{\!\{(C_i(\phi_k(\mathbf{v},w)), \#_i^k(\mathbf{v},\phi_k(\mathbf{v},w))) \mid w \in N(v_k)\}\!\}\big), \tag{6}$$

instead of δ-k-LWL aggregation defined in Equation (5). Here, the function

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

where $\mathbf{w} \sim_j \mathbf{v}$ denotes that $\mathbf{w}$ is 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). For a fixed $\mathbf{v}$, the function $\#_i^j(\mathbf{v},\cdot)$ is uniform over the set $S \cap N_j$, where S is a color class obtained after i iterations of the δ-k-LWL^+ and N_j denotes the set of j-neighbors of $\mathbf{v}$. Note that after the stable partition has been reached $\#_i^j(\mathbf{v})$ will not change anymore. Observe that each iteration of the δ-k-LWL^+ has the same asymptotic running time as an iteration of the δ-k-LWL.

The following theorem shows that the local variant δ-k-LWL^+ is at least as powerful as δ-k-WL when restricted to the class of connected graphs. In other words, given two *connected* graphs G and H, if these graphs are distinguished by δ-k-WL, then they must also be distinguished by δ-k-LWL^+. On the other hand, it is important to note that, in general, the δ-k-LWL^+ might need a larger number of iterations to distinguish two graphs, as compared to δ-k-WL. However, this leads to advantages in a machine learning setting, see Section 5.

Theorem 4. For the class of connected graphs, the following holds for all $k \geq 1$:

$$\delta\text{-}k\text{-}\mathsf{LWL}^+ \equiv \delta\text{-}k\text{-}\mathsf{WL}.$$

Along with Proposition 3, we obtain the following corollary relating the power of k-WL and δ-k-LWL^+.

Corollary 5. For the class of connected graphs, the following holds for all $k \geq 2$:

$$\delta\text{-}k\text{-}\mathsf{LWL}^+ \sqsubset k\text{-}\mathsf{WL}.$$

In fact, the proof of Proposition 3 shows that the infinite family of graphs G_k, H_k witnessing the strictness condition can even be distinguished by δ-k-LWL, for each corresponding $k \geq 2$. We note here that the restriction to connected graphs can easily be circumvented by adding a specially marked vertex, which is connected to every other vertex in the graph.

D.1. Kernels based on vertex refinement algorithms

The idea for a kernel based on the δ-k-LWL (and the other vertex refinements algorithms) is to compute it for $h \geq 0$ iterations resulting in a coloring function $C^{k,\delta}\colon V(G) \to \Sigma_i$ for each iteration i. Now, after each iteration, we compute a *feature vector* $\phi_i(G)$ in $\mathbb{R}^{|\Sigma_i|}$ for each graph G. Each component $\phi_i(G)_c$ counts the number of occurrences of k-tuples labeled by c in Σ_i. The overall feature vector $\phi_{\text{LWL}}(G)$ is defined as the concatenation of the feature vectors of all h iterations, i.e., $\phi_{\text{LWL}}(G) = \big[\phi_0(G), \ldots, \phi_h(G)\big]$. The corresponding kernel for h iterations then is computed as $k_{\text{LWL}}(G,H) = \langle \phi_{\text{LWL}}(G), \phi_{\text{LWL}}(H)\rangle$, where $\langle\cdot,\cdot\rangle$ denotes the standard inner product.

D.2. Local converges to global: Proof of Theorem 1

The main technique behind the proof is to encode the colors assigned by k-WL (or its variants) as rooted directed trees, called *unrolling trees*. The exact construction of the unrolling tree depends on the aggregation map $M(\cdot)$ used by the k-WL variant under consideration. We illustrate this construction for the k-WL. For other variants such as the δ-k-WL, δ-k-LWL, and δ-k-LWL^+, we will specify analogous constructions.

Unrollings ("Rolling in the deep"). Given a graph G, tuple $\mathbf{v}$ in $V(G)^k$, and an integer $\ell \geq 0$, the *unrolling* $\mathsf{UNR}\,[G,\mathbf{s},\ell]$ is a rooted, directed tree with vertex and edge labels, defined recursively as follows.

- For $\ell = 0$, $\mathsf{UNR}\,[G,\mathbf{v},0]$ is defined to be a single vertex, labeled with the isomorphism type $\tau(\mathbf{s})$. This lone vertex is also the root vertex.

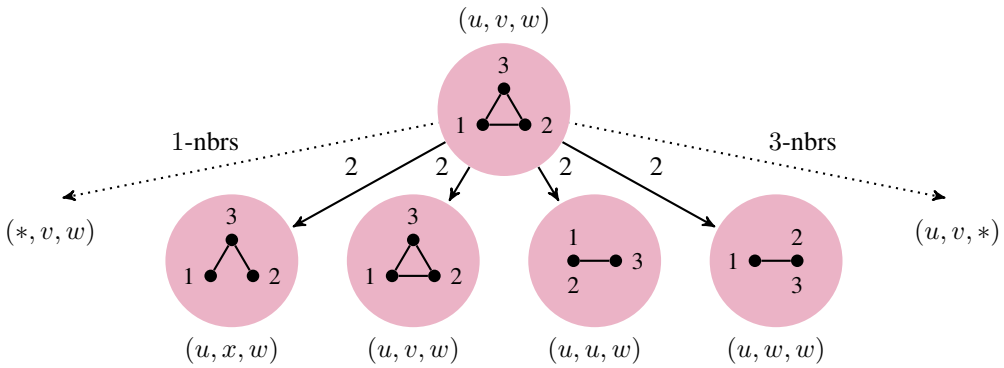


Figure 4: Unrolling at the tuple (u, v, w) of depth one.

- For $\ell > 0$, $\mathsf{UNR}\,[G, \mathbf{v}, \ell]$ is defined as follows. First, introduce a root vertex r, labeled with the isomorphism type $\tau(\mathbf{v})$. Next, for each $j \in [k]$ and for each j-neighbor $\mathbf{w}$ of $\mathbf{v}$, append the rooted subtree $\mathsf{UNR}\,[G, \mathbf{w}, \ell - 1]$ below the root r. Moreover, the directed edge e from r to the root of $\mathsf{UNR}\,[G, \mathbf{w}, \ell - 1]$ is labeled j iff $\mathbf{w}$ is a j-neighbor of $\mathbf{v}$.

We refer to $\mathsf{UNR}\,[G, \mathbf{v}, \ell]$ as the unrolling of the graph G *at* $\mathbf{v}$ *of depth* ℓ. Figure 4 partially illustrates the recursive construction of unrolling trees: it describes the unrolling tree for the graph in Figure 3 at the tuple (u, v, w), of depth 1. Each node w in the unrolling tree is associated with some k-tuple $\mathbf{w}$, indicated alongside the node in the figure. We call $\mathbf{w}$ the tuple corresponding to the node w.

Analogously, we can define unrolling trees δ-UNR, L-UNR, and L^+-UNR for the k-WL-variants δ-k-WL, δ-k-LWL, and δ-k-LWL^+ respectively. The minor differences lie in the recursive step above, since the unrolling construction needs to faithfully represent the aggregation process.

- For δ-UNR, we additionally label the directed edge e with (j, L) or (j, G) instead of just j, depending on whether the neighborhood is local or global.
- For L-UNR, we consider only the subtrees L-$\mathsf{UNR}\,[G, \mathbf{w}, \ell - 1]$ for local j-neighbors $\mathbf{w}$.
- For L^+-UNR, we again consider only the subtrees L^+-$\mathsf{UNR}\,[G, \mathbf{w}, \ell - 1]$ for local j-neighbors $\mathbf{w}$. However, the directed edge e to this subtree is also labeled with the $\#$ counter value $\#^j_{\ell-1}(\mathbf{v}, \mathbf{w})$.

Encoding Colors as Trees. The following Lemma shows that the computation of k-WL can be faithfully encoded by the unrolling trees. Formally, let $\mathbf{s}$ and $\mathbf{t}$ be two k-vertex-tuples in $V(G)^k$.

Lemma 6. The colors of $\mathbf{s}$ and $\mathbf{t}$ after ℓ rounds of k-WL are identical if and only if the unrolling tree $\mathsf{UNR}\,[G, \mathbf{s}, \ell]$ is isomorphic to the unrolling tree $\mathsf{UNR}\,[G, \mathbf{t}, \ell]$.

Proof. By induction on ℓ. For the base case $\ell = 0$, observe that the initial colors of $\mathbf{s}$ and $\mathbf{t}$ are equal to the respective isomorphism types $\tau(\mathbf{s})$ and $\tau(\mathbf{t})$. On the other hand, the vertex labels for the single-vertex graphs $\mathsf{UNR}\,[G, \mathbf{s}, 0]$ and $\mathsf{UNR}\,[G, \mathbf{t}, 0]$ are also the respective isomorphism types $\tau(\mathbf{s})$ and $\tau(\mathbf{t})$. Hence, the statement holds for $\ell = 0$.

For the inductive case, we proceed with the forward direction. Suppose that k-WL assigns the same color to $\mathbf{s}$ and $\mathbf{t}$ after ℓ rounds. For each j in $[k]$, the j-neighbors of $\mathbf{s}$ form a partition $\mathbf{C}_1, \ldots, \mathbf{C}_p$ corresponding to their colors after $\ell - 1$ rounds of k-WL. Similarly, the j-neighbors of $\mathbf{t}$ form a partition $\mathbf{D}_1, \ldots, \mathbf{D}_p$ corresponding to their colors after $\ell - 1$ rounds of k-WL, where for i in $[p]$, $\mathbf{C}_i$ and $\mathbf{D}_i$ have the same size and correspond to the same color. By inductive hypothesis, the corresponding depth $\ell - 1$ unrollings $\mathsf{UNR}\,[G, \mathbf{c}, \ell - 1]$ and $\mathsf{UNR}\,[G, \mathbf{d}, \ell - 1]$ are isomorphic, for every $\mathbf{c}$ in $\mathbf{C}_i$ and $\mathbf{d}$ in $\mathbf{D}_i$. Since we have a bijective correspondence between the depth $\ell - 1$ unrollings of the j-neighbors of $\mathbf{s}$ and $\mathbf{t}$, respectively, there exists an isomorphism between $\mathsf{UNR}\,[G, \mathbf{s}, \ell]$ and $\mathsf{UNR}\,[G, \mathbf{t}, \ell]$. Moreover, this isomorphism preserves vertex labels (corresponding to isomorphism types) and edges labels (corresponding to j-neighbors).

For the backward direction, suppose that $\mathsf{UNR}\,[G, \mathbf{s}, \ell]$ is isomorphic to $\mathsf{UNR}\,[G, \mathbf{t}, \ell]$. Then, we have a bijective correspondence between the depth $\ell - 1$ unrollings of the j-neighbors of $\mathbf{s}$ and of $\mathbf{t}$, respectively. For each j in $[k]$, the j-neighbors of $\mathbf{s}$ form a partition $\mathbf{C}_1, \ldots, \mathbf{C}_p$ corresponding to their unrolling trees after $\ell - 1$ rounds of k-WL. Similarly, the j-neighbors of $\mathbf{t}$ form a

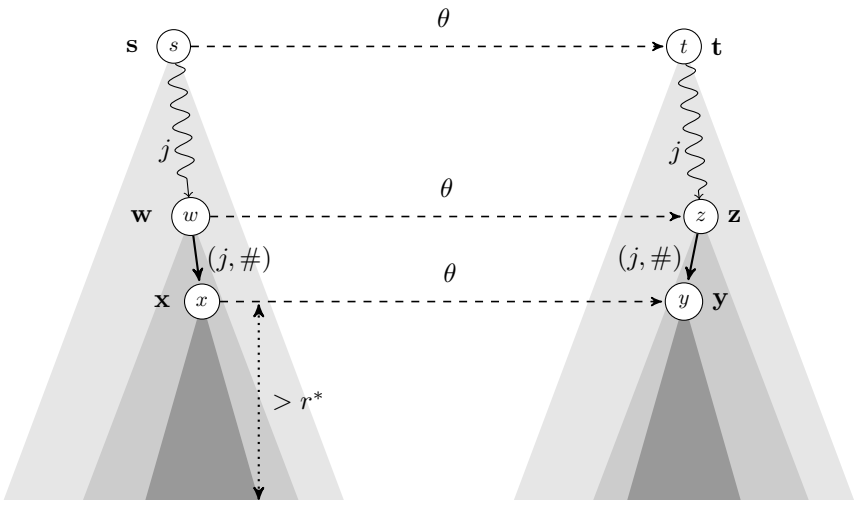


Figure 5: Unrollings $L_1 = \mathsf{L}^+\text{-}\mathsf{UNR}\,[G, \mathbf{s}, q]$ and $L_2 = \mathsf{L}^+\text{-}\mathsf{UNR}\,[G, \mathbf{t}, q]$ of sufficiently large depth.

partition $\mathbf{D}_1, \ldots, \mathbf{D}_p$ corresponding to their unrolling trees after $\ell - 1$ rounds of k-WL, where for i in $[p]$, C_i, and D_i have the same size and correspond to the same isomorphism type of the unrolling tree. By induction hypothesis, the j-neighborhoods of $\mathbf{s}$ and $\mathbf{t}$ have an identical color profile after $\ell - 1$ rounds. Finally, since the depth $\ell - 1$ trees $\mathsf{UNR}\,[G, \mathbf{s}, \ell - 1]$ and $\mathsf{UNR}\,[G, \mathbf{t}, \ell - 1]$ are trivially isomorphic, the tuples $\mathbf{s}$ and $\mathbf{t}$ have the same color after $\ell - 1$ rounds. Therefore, k-WL must assign the same color to $\mathbf{s}$ and $\mathbf{t}$ after ℓ rounds. □

Using identical arguments, we can state the analogue of Lemma 6 for the algorithms δ-k-WL, δ-k-LWL, δ-k-LWL^+, and their corresponding unrolling constructions δ-UNR, L-UNR and L^+-UNR. The proof is identical and is hence omitted.

Lemma 7. The following statements hold.

1. The colors of $\mathbf{s}$ and $\mathbf{t}$ after ℓ rounds of δ-k-WL are identical if and only if the unrolling tree δ-$\mathsf{UNR}\,[G, \mathbf{s}, \ell]$ is isomorphic to the unrolling tree δ-$\mathsf{UNR}\,[G, \mathbf{t}, \ell]$.
2. The colors of $\mathbf{s}$ and $\mathbf{t}$ after ℓ rounds of δ-k-LWL are identical if and only if the unrolling tree L-$\mathsf{UNR}\,[G, \mathbf{s}, \ell]$ is isomorphic to the unrolling tree L-$\mathsf{UNR}\,[G, \mathbf{t}, \ell]$.
3. The colors of $\mathbf{s}$ and $\mathbf{t}$ after ℓ rounds of δ-k-LWL^+ are identical if and only if the unrolling tree L^+-$\mathsf{UNR}\,[G, \mathbf{s}, \ell]$ is isomorphic to the unrolling tree L^+-$\mathsf{UNR}\,[G, \mathbf{t}, \ell]$.

Equivalence. The following Lemma establishes that the local algorithm δ-k-LWL^+ is at least as powerful as the global δ-k-WL, for connected graphs, i.e., δ-k-$\mathsf{LWL}^+ \sqsubseteq \delta$-$k$-$\mathsf{WL}$.

Lemma 8. Let G be a connected graph, and let $\mathbf{s}, \mathbf{t} \in V(G)^k$. If the stable colorings of $\mathbf{s}$ and $\mathbf{t}$ under δ-k-LWL^+ are identical, then the stable colorings of $\mathbf{s}$ and $\mathbf{t}$ under δ-k-WL are also identical.

Proof. Let r^* denote the number of rounds needed to attain the stable coloring under δ-k-LWL^+. Consider unrollings $L_1 = \mathsf{L}^+\text{-}\mathsf{UNR}\,[G, \mathbf{s}, q]$ and $L_2 = \mathsf{L}^+\text{-}\mathsf{UNR}\,[G, \mathbf{t}, q]$ of sufficiently large depth $q = r^* + |V(G)| + 1$. Since $\mathbf{s}$ and $\mathbf{t}$ have the same stable coloring under δ-k-LWL^+, the trees L_1 and L_2 are isomorphic (by Lemma 7). Let θ be an isomorphism from L_1 to L_2.

We prove the following equivalent statement. If L_1 and L_2 are isomorphic, then for all $i \geq 0$, δ-$\mathsf{UNR}\,[G, \mathbf{s}, i] = \delta$-$\mathsf{UNR}\,[G, \mathbf{t}, i]$. The proof is by induction on i. The base case $i = 0$ follows trivially by comparing the isomorphism types of $\mathbf{s}$ and $\mathbf{t}$.

For the inductive case, let $j \in [k]$. Let $\mathbf{X}_j$ be the set of j-neighbors of $\mathbf{s}$. Similarily, let $\mathbf{Y}_j$ be the set of j-neighbors of $\mathbf{t}$. Our goal is to construct, for every $j \in [k]$, a corresponding bijection σ_j between $\mathbf{X}_j$ and $\mathbf{Y}_j$ satisfying the following conditions.

1. For all $\mathbf{x}$ in $\mathbf{X}_j$, $\mathbf{x}$ is a local j-neighbor of $\mathbf{s}$ if and only if $\sigma_j(\mathbf{x})$ is a local j-neighbor of $\mathbf{t}$.

2. For all $\mathbf{x}$ in $\mathbf{X}_j$, δ-$\mathsf{UNR}\,[G, \mathbf{x}, i-1] = \delta$-$\mathsf{UNR}\,[G, \sigma_j(\mathbf{x}), i-1]$, i.e., $\mathbf{x}$ and $\sigma_j(\mathbf{x})$ are identically colored after $i-1$ rounds of δ-k-WL.

From the definition of δ-UNR trees, the existence of such $\sigma_1, \ldots, \sigma_k$ immediately implies the desired claim δ-$\mathsf{UNR}\,[G, \mathbf{s}, i] = \delta$-$\mathsf{UNR}\,[G, \mathbf{t}, i]$. First, we show the following claim.

Claim 9. Let $\mathbf{C}$ be a color class in the stable coloring of G under δ-k-LWL^+. Let $j \in [k]$. Then, $|\mathbf{C} \cap \mathbf{X}_j| = |\mathbf{C} \cap \mathbf{Y}_j|$.

Proof. Either $|\mathbf{C} \cap \mathbf{X}_j| = |\mathbf{C} \cap \mathbf{Y}_j| = 0$, in which case we are done. Otherwise, assume without loss of generality that $|\mathbf{C} \cap \mathbf{X}_j| \neq 0$. Let $\mathbf{x}$ in $\mathbf{C} \cap \mathbf{X}_j$. Since G is connected, we can start from the root s of L_1, go down along j-labeled edges, and reach a vertex x such that x corresponds to the tuple $\mathbf{x}$. Let w be the parent of x, and let $\mathbf{w}$ be the tuple corresponding to w. Note that $\mathbf{x}$ is a local j-neighbor of $\mathbf{w}$. Moreover, the depth of $\mathbf{w}$ is at most $n-1$. Hence, the height of the subtree of L_1 rooted at w is at least $q - (n-1) > r^*$.

Consider the tuple $\mathbf{z}$ corresponding to the vertex $z = \theta(w)$ in L_2. Observe that the path from the root t of L_2 to the vertex $z = \theta(w)$ consists of j-labeled edges. Therefore, $\mathbf{z}$ is j-neighbor of $\mathbf{t}$, and hence $\mathbf{z}$ in $\mathbf{Y}_j$. The stable colorings of $\mathbf{w}$ and $\mathbf{z}$ under δ-k-LWL^+ are identical, because the subtrees rooted at w and z are of depth more than r^*. Let $\mathbf{C}$ denote the common color class of $\mathbf{w}$ and $\mathbf{z}$, in the stable coloring of G under δ-k-LWL^+.

Since $\mathbf{x}$ is a local neighbor of $\mathbf{w}$, the agreement of the $\#$ function values ensures that the number of j-neighbors (local or global) of $\mathbf{w}$ in $\mathbf{C}$ is equal to the number of j-neighbors (local or global) of $\mathbf{z}$ in $\mathbf{C}$. Finally, the set of j-neighbors of $\mathbf{w}$ is equal to the set of j-neighbors of $\mathbf{s}$, which is $\mathbf{X}_j$. Similarily, the set of j-neighbors of $\mathbf{z}$ is equal to the set of j-neighbors of $\mathbf{t}$, which is $\mathbf{Y}_j$. Hence, $|\mathbf{C} \cap \mathbf{X}_j| = |\mathbf{C} \cap \mathbf{Y}_j|$. □

Moreover, for each $j \in [k]$, the number of local j-neighbors of $\mathbf{s}$ in $\mathbf{C} \cap \mathbf{X}_j$ is equal to the number of local j-neighbors of $\mathbf{t}$ in $\mathbf{C} \cap \mathbf{Y}_j$. Otherwise, we could perform one more round of δ-k-LWL^+ and derive different colors for $\mathbf{s}$ and $\mathbf{t}$, a contradiction.

Hence, we can devise the required bijection $\sigma_j = \sigma_j^L \dot\cup\, \sigma_j^G$ as follows. We pick an arbitrary bijection σ_j^L between the set of local j-neighbors of $\mathbf{s}$ inside $\mathbf{C}$ and the set of local j-neighbors of $\mathbf{t}$ inside $\mathbf{C}$. We also pick an arbitrary bijection σ_j^G between the set of global j-neighbors of $\mathbf{s}$ inside $\mathbf{C}$ and the set of global j-neighbors of $\mathbf{t}$ inside $\mathbf{C}$. Clearly, σ_j satisfies the first stipulated condition. By induction hypothesis, the second condition is also satisifed. Hence, we can obtain a desired bijection σ_j satisfying the two stipulated conditions. Since we obtain the desired bijections $\sigma_1, \ldots, \sigma_k$, this finishes the proof of the lemma. □

Finally, since for a graph $G = (V, E)$, $M_i^{\delta,\overline{\delta}}(\mathbf{v}) = M_i^{\delta,\overline{\delta}}(\mathbf{w})$ implies $M_i^{\delta,+}(\mathbf{v}) = M_i^{\delta,+}(\mathbf{w})$ for all $\mathbf{v}$ and $\mathbf{w}$ in $V(G)^k$ and $i \geq 0$, it holds that δ-k-$\mathsf{WL} \sqsubseteq \delta$-$k$-$\mathsf{LWL}^+$. Together with Lemma 8 above, this finishes the proof of Theorem 4.

E. Higher-order neural architectures (Expanded)

Although the discrete kernels defined in the previous section are quite powerful, they are limited due to their fixed feature construction scheme, hence suffering from (a) poor adaptation to the learning task at hand and (b) suffer from the inability to handle continuous node and edge labels in a meaningful way. Moreover, they often result in high-dimensional embeddings forcing one to resort to non-scalable, kernelized optimization procedures. To address this, we derive a neural architecture, called *local* δ-k-GNNs (δ-k-LGNN). We show that it has the same power as the δ-k-LWL, can naturally handle continuous information, and learn fixed dimensional graph embeddings straightforwardly. Moreover, it uses the same number of parameters as a corresponding (1-dimensional) GNN.

Let (G, l) be a labeled graph, following (Morris et al., 2019), we assume an initial node coloring $f^{(0)} \colon V(G)^k \to \mathbb{R}^{1\times d}$ such that each tuple $\mathbf{v}$ is annotated with a feature $f^{(0)}(v)$ in $\mathbb{R}^{1\times d}$ where $f^{(0)}(\mathbf{v}) = f^{(0)}(\mathbf{u})$ if and only if the labeled isomorphism types $G[\mathbf{v}]$ and $G[\mathbf{w}]$ match. See the paragraph at the end of this section on how to deal with continous information. In each layer $t > 0$, we compute a new feature

$$f^{(t)}(\mathbf{v}) = f_{\text{merge}}^{W_1}\Big(f^{(t-1)}(\mathbf{v}), f_{\text{aggr}}^{W_2}\big(\{\!\{f^{(t-1)}(\phi_1(\mathbf{v}, w)) \mid w \in \delta(v_1)\}\!\}, \ldots, \{\!\{f^{(t-1)}(\phi_k(\mathbf{v}, w)) \mid w \in \delta(v_k)\}\!\}\big)\Big), \tag{7}$$

in $\mathbb{R}^{1\times e}$ for a tuple $\mathbf{v}$, where $W_1^{(t)}$ and $W_2^{(t)}$ are parameter matrices from $\mathbb{R}^{d\times e}$ and σ denotes a component-wise non-linear function, e.g., a sigmoid or a ReLU.[5] Here, $f_{\text{aggr}}^{W_1}$ aggregates over the set of local neighbors and $f_{\text{merge}}^{W_2}$ merges the tuple's representations from step $(t-1)$ with the computed neighborhood features. Both $f_{\text{aggr}}^{W_1}$ and $f_{\text{merge}}^{W_2}$ may be arbitrary differentiable, (permutation-invariant, in the case of the former) functions. Initially, we set $f^{(0)}(\mathbf{v})$ to a one-hot encoding of the (labeled) isomorphism type of $G[\mathbf{v}]$. Note that we can naturally handle discrete node and edge labels as well as directed graphs. In order to adapt the parameters W_1 and W_2 of Equation (7), to a given data distribution, we optimize them in an end-to-end fashion together with the parameters of the neural network used for the learning task.

Following (Morris et al., 2019), let $\mathbf{W}^{(t)} = \big(W_1^{(t')}, W_2^{(t')}\big)_{t'\leq t}$ denote parameters given by Equation (7) up to iteration t. The following result shows that there is a sequence of parameter matrices such that the δ-k-GNN has the same power as the δ-k-LWL in terms of distinguishing non-isomorphic graphs. Moreover, it inherits the strengths of the later, i.e., it can be implemented using sparse matrix multiplication.

Theorem 10. Let (G, l) be a labeled graph. Then for all $t \geq 0$ there exists a sequence of weights $\mathbf{W}^{(t)}$ such that

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

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

$$\delta\text{-}k\text{-LGNN} \equiv \delta\text{-}k\text{-LWL}.$$

Proof sketch. First, observe that the δ-k-LGNN can be simulated on an appropriate node- and edge-labeled graph on n^k vertices. Secondly, following the proof of Theorem 2 in (Morris et al., 2019), there exists a parameter matrix $W_2^{(t)}$ such that we can injectively map each multiset in Equation (7), 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 the proof of Theorem 2 in (Morris et al., 2019), 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. □

Note that it is not possible to come up with an architecture, i.e., instantiations of $f_{\text{aggr}}^{W_1}$ and $f_{\text{merge}}^{W_2}$, such that it becomes more powerful than the k-δ-LWL, see (Morris et al., 2019). However, all results from the previous section can be lifted to the neural setting.

That is, in the same spirit as Equation (7), one can derive neural architectures based on the δ-k-WL and k-WL, called δ-k-GNN and k-WL-GNN, respectively, and prove results analogous to Theorem 10. For example, we can express δ-k-GNN in a similar fashion to Equation (7) by learning separate embeddings for local and global neighbors. We can then learn a joint representation, e.g., using a multi-layer perceptron, to weight the importance of local and global information in a data-driven way. We further explore this in the experimental section. In principle, it is also possible to devise an architecture (δ-k-GNN$^+$) with the same power as the δ-k-LWL$^+$. However, due to the discrete nature of the # labeling function, it has to be precomputed in advance and encoded as additional node features.

Generalization abilites of the neural architecture. Garg et al. (Garg et al., 2020), studied the generalization abilities of a standard GNN architecture for binary classification using a margin loss. Under certain mild conditions, they bounded the empirical Rademacher complexity as $\tilde{\mathcal{O}}(rdL/\sqrt{m}\gamma)$, where d is the maximum degree of the employed graphs, r is the number of components of the node features, L is the number of layers, and γ is a parameter of the loss function. It is straightforward to transfer the above bound to the higher-order (local) layer from above. Hence, this shows that local, sparsity-aware, higher-order variants, e.g., δ-k-LGNN, exhibit a smaller generalization error compared to dense, global variants like the k-WL-GNN.

Incorporating continous information. Since many real-world graphs, e.g., molecules, have continuous features (real-valued vectors) attached to vertices and edges, using a one-hot encoding of the (labeled) isomorphism type is not a sensible choice. Let $a\colon V(G) \to \mathbb{R}^{1\times d}$ be a function such that each vertex v is annotated with a feature $a(v)$ in $\mathbb{R}^{1\times d}$, and let $\mathbf{v} = (v_1, \ldots, v_k)$ be a k-tuple of vertices. Then we can compute an inital feature

$$f^{(0)}(\mathbf{v}) = f_{\text{enc}}^{W_3}\big((a(v_1), \ldots, a(v_k))\big), \tag{8}$$

[5] For clarity of presentation we omit biases.

for the tuple $\mathbf{v}$. Here, $f_{\text{enc}} \colon \left(\mathbb{R}^{1\times d}\right)^k \to \mathbb{R}^{1\times e}$ is a arbitrary differentiable, parameterized function, e.g., a multi-layer perceptron or a standard GNN aggregation function, that computes of joint representation of the k node features $a(v_1), \ldots, a(v_k)$. Moreover, it also straightforward to incorporate the labeled isomorphism type and continuous edge label information. We further explore this in the experimental section.

F. Practicality, barriers ahead, and possible road maps

As Theorem 4 shows, the δ-k-LWL$^+$ and its corresponding neural architecture, the δ-k-LGNN$^+$, have the same power in distinguishing non-isomorphic graphs as δ-k-WL. Although for dense graphs, the local algorithms will have the same running time, for sparse graphs, the running time for each iteration can be upper-bounded by $|n^k| \cdot kd$, where d denotes the maximum or average degree of the graph. Hence, the local algorithm takes the sparsity of the underlying graph into account, resulting in improved computation times compared to the non-local δ-k-WL and the k-WL (for the same number of iterations). These observations also translate into practice, see Appendix G. The same arguments can be used in favor of the δ-k-LWL and δ-k-LGNN, which lead to even sparser algorithms.

Obstacles. The biggest obstacle in applying the algorithms to truly large graphs is the fact that the algorithm considers all possible k-tuples leading to a lower bound on the running of $\Omega(n^k)$. Lifting the results to the folklore k-WL, e.g., (Maron et al., 2019a), only "shaves off one dimension". Moreover, applying higher-order algorithms for large k might lead to severe overfitting issues, see also Appendix G.

Possible solutions. Recent sampling-based approaches for graph kernels or GNNs, see, e.g., (Chen et al., 2018a;b; Hamilton et al., 2017; Huang et al., 2018; Morris et al., 2017) address the dependence on n^k, while appropriate pooling methods along the lines of Equation (8) address the overfitting issue. Finally, new directions from the theory community, e.g., (Grohe et al., 2020) paint further directions, which might result in more scalable algorithms.

G. Expanded experimental evaluation

Our intention here is to investigate the benefits of the local, sparse algorithms, both kernel and neural architectures, compared to the global, dense algorithms, and standard kernel and GNN baselines. More precisely, we address the following questions:

Q1 Do the local algorithms, both kernel and neural architectures, lead to improved classification and regression scores on real-world benchmark datasets compared to global, dense algorithms and standard baselines?
Q2 Does the δ-k-LWL$^+$ lead to improved classification accuracies compared to the δ-k-LWL? Does it lead to higher computation times?
Q3 How much do the local algorithms speed up the computation time compared to the non-local algorithms or dense neural architectures?

G.1. Datasets, graph kernels, and neural architectures

In the following, we give an overview of employed datasets, (baselines) kernels, and (baseline) neural architectures.

Datasets To evaluate kernels, we use the following, well-known, small-scale ENZYMES (Schomburg et al., 2004; Borgwardt & Kriegel, 2005), IMDB-BINARY, IMDB-MULTI (Yanardag & Vishwanathan, 2015b), NCI1, NCI109 (Wale et al., 2008), PTC_FM[6], PROTEINS (Dobson & Doig, 2003; Borgwardt & Kriegel, 2005), and REDDIT-BINARY (Yanardag & Vishwanathan, 2015b) datasets. To show that our kernels also scale to larger datasets, we additionally used the mid-scale YEAST, YEASTH, UACC257, UACC257H, OVCAR-8, OVCAR-8H (Yan et al., 2008)[7] datasets. For the neural architectures, see Appendix E, we used the large-scale molecular regression datasets ZINC (Dwivedi et al., 2020; Jin et al., 2018) and ALCHEMY (Chen et al., 2019a). 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 further compare to the (hierarchical) k-GNN (Morris et al., 2019) and k-IGN (Maron et al., 2019a), and show the benefits of our architecture in presence of continuous features, we used the QM9 (Ramakrishnan et al., 2014; Wu et al., 2018) regression

[6] https://www.predictive-toxicology.org/ptc/
[7] https://sites.cs.ucsb.edu/~xyan/dataset.htm

Table 3: Dataset statistics and properties, †—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 vertices	∅ Number of edges	Vertex 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	✗	✗
NCI1	4 110	2	29.9	32.3	✓	✗
NCI109	4 127	2	29.7	32.1	✓	✗
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	✗	✗
YEAST	79 601	2	21.5	22.8	✓	✓
YEASTH	79 601	2	39.4	40.7	✓	✓
UACC257	39 988	2	26.1	28.1	✓	✓
UACC257H	39 988	2	46.7	48.7	✓	✓
OVCAR-8	40 516	2	26.1	28.1	✓	✓
OVCAR-8H	40 516	2	46.7	48.7	✓	✓
ZINC	249 456	12	23.1	24.9	✓	✓
ALCHEMY	202 579	12	10.1	10.4	✓	✓
QM9	129 433	12	18.0	18.6	✓(13+3D)†	✓(4)

dataset.[8] To study data efficiency, we also used smaller subsets of the ZINC and ALCHEMY dataset. That is, for the ZINC 10K (ZINK 50K) dataset, following (Dwivedi et al., 2020), we sampled 10 000 (50 000) graphs from the training, and 1 000 (5 000) from the training and validation split, respectively. For ZINC 10K, we used the same splits as provided by (Dwivedi et al., 2020). For the ALCHEMY 10K (ALCHEMY 50K) dataset, as there is no fixed split available for the full dataset[9], we sampled the (disjoint) training, validation, and test splits uniformly and at random from the full dataset. See Table 3 for dataset statistics and properties.[10]

Kernels We implemented the δ-k-LWL, δ-k-LWL$^+$, δ-k-WL, and k-WL kernel for k in $\{2, 3\}$. We compare 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 (Shervashidze et al., 2009) (GR), and the shortest-path kernel (Borgwardt & Kriegel, 2005) (SP). All kernels were (re-)implemented in C++11. For the graphlet kernel we counted (labeled) connected subgraphs of size three.

Neural architectures We used the GIN and GIN-ε architecture (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 neural architectures of Appendix E, δ-k-LGNN, δ-k-GNN, k-WL-GNN, 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 $f^{W_1}_{\text{merge}}$ and $f^{W_2}_{\text{aggr}}$ of Equation (7). Finally, we used the PYTORCH (Paszke et al., 2019) implementations of the 3-IGN (Maron et al., 2019a), and 1-2-GNN, 1-3-GNN, 1-2-3-GNN (Morris et al., 2019) made available by the respective authors.

For the QM9 dataset, we additionally used the MPNN architecture as a baseline, closely following the setup of (Gilmer et al., 2017). For the GINE-ε and the MPNN architecture, following Gilmer et al. (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 (Klicpera et al., 2020) or Cormorant (Anderson et al., 2019), but to solely show the benefits of the (local) higher-order architectures compared to the corresponding (1-dimensional) GNN. For the δ-2-GNN, to implement Equation (8), for each 2-tuple we concatenated the (two) node and edge features, computed pairwise ℓ_2 distances based on the 3D-coordinates, and a one-hot

[8] We opted for comparing on the QM9 dataset to ensure a fair comparison concerning hyperparameter selection.

[9] Note that the full dataset is different from the contest dataset, e.g., it does not provide normalized targets, see https://alchemy.tencent.com/.

[10] All datasets can be obtained from http://www.graphlearning.io.

Table 4: 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	NCI1	NCI109	PTC_FM	PROTEINS	REDDIT-BINARY
Baseline	GR	29.8 ±1.0	59.5 ±0.4	40.6 ±0.5	66.3 ±0.2	66.7 ±0.2	62.3 ±0.9	71.6 ±0.2	60.0 ±0.2
	SP	42.3 ±1.3	59.2 ±0.3	39.6 ±0.4	74.5 ±0.3	73.4 ±0.1	63.2 ±0.6	76.4 ±0.4	84.7±0.2
	1-WL	53.4 ±1.4	72.4 ±0.5	50.6 ±0.6	85.1 ±0.2	85.2 ±0.2	62.9 ±1.6	73.7 ±0.5	75.3 ±0.3
	WLOA	59.7 ±1.2	73.1 ±0.7	50.3 ±0.6	85.6 ±0.2	86.0 ±0.3	63.7 ±0.7	73.7 0.5±	88.7 ±0.2
Neural	Gin-0	39.6 ±1.3	72.8 ±0.9	50.0 ±0.1	78.5 ±0.5	77.1 ±0.6	58.0 ±1.4	71.7 ±0.9	90.7 ±0.9
	Gin-ε	38.7 ±2.1	73.0 ±1.0	49.8 ±0.6	78.8 ±0.3	77.2 ±0.3	58.7 ±1.7	70.4 ±1.2	89.4 ±1.2
Global	2-WL	38.9 ±0.8	69.2 ±0.6	48.0 ±0.5	67.5 ±0.3	68.3 ±0.2	64.3 ±0.6	75.3 ±0.3	OOM
	3-WL	45.9 ±0.8	69.2 ±0.4	47.9 ±0.7	OOT	OOT	64.4 ±0.6	OOM	OOM
	δ-2-WL	39.1 ±1.1	69.4 ±0.7	48.0 ±0.4	67.4 ±0.3	68.3 ±0.3	**64.5** ±0.4	75.2 ±0.5	OOM
	δ-3-WL	45.9 ±0.9	69.1 ±0.6	47.9 ±0.8	OOT	OOT	64.4 ±0.6	OOM	OOM
Local	δ-2-LWL	57.7 ±1.0	73.3 ±0.7	50.9 ±0.6	85.4 ±0.2	84.8 ±0.2	62.7 ±1.3	74.5 ±0.6	90.0 ±0.2
	δ-2-LWL$^+$	57.0 ±0.8	78.9 ±0.6	**64.0** ±0.4	**91.8** ±0.2	**90.8** ±0.2	62.7 ±1.4	**82.6** ±0.4	**91.5** ±0.2
	δ-3-LWL	**60.4** ±0.8	73.5 ±0.5	49.6 ±0.7	84.0 ±0.3	83.0 ±0.3	62.6 ±1.2	OOM	OOM
	δ-3-LWL$^+$	58.9 ±1.1	**80.6** ±0.5	60.3 ±0.4	83.9 ±0.3	82.9 ±0.3	62.4 ±1.2	OOM	OOM

Table 5: Classification accuracies in percent and standard deviations on medium-scale datasets.

	Method	Dataset					
		YEAST	YEASTH	UACC257	UACC257H	OVCAR-8	OVCAR-8H
	1-WL	88.9 < 0.1	88.9 < 0.1	96.8 < 0.1	96.9 < 0.1	96.3 < 0.1	96.3 < 0.1
Local Neural	GINE	88.3 < 0.1	88.3 < 0.1	95.9 < 0.1	95.9 < 0.1	94.9 < 0.1	94.9 < 0.1
	GINE-ε	88.3 < 0.1	88.3 < 0.1	95.9 < 0.1	95.9 < 0.1	95.0 < 0.1	94.9 < 0.1
	δ-2-LWL	88.6 < 0.1	88.5 < 0.1	96.8 < 0.1	96.5 < 0.1	96.1 < 0.1	95.9 < 0.1
	δ-2-LWL$^+$	**98.9** < 0.1	**99.1** < 0.1	**99.2** < 0.1	**98.9** < 0.1	**99.3** < 0.1	**99.0** < 0.1

encoding of the (labeled) isomorphism type. Finally, we used a 2-layer MLP to learn a joint, initial vectorial representation.

The source code of all methods and evaluation procedures will be made available at `www.github.com/chrsmrrs/sparsewl`.

G.2. Experimental protocol and model configuration

In the following, we describe the experimental protocol and hyperparameter setup.

Kernels For the smaller datasets (first third of Table 3), for each kernel, we computed the (cosine) normalized gram matrix. We computed the classification accuracies using the C-SVM implementation of LIBSVM (Chang & Lin, 2011), using 10-fold cross-validation. The C-parameter was selected from $\{10^{-3}, 10^{-2}, \ldots, 10^{2}, 10^{3}\}$ by (inner) 10-fold cross-validation on the training folds. We repeated each 10-fold cross-validation ten times with different random folds, and report average accuracies and standard deviations. For the larger datasets (second third of Table 3), we computed sparse feature vectors for each graph and used the linear C-SVM implementation of LIBLINEAR (Fan et al., 2008), using 10-fold cross-validation. The C-parameter was again 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). For measuring the classification accuracy, the number of iterations of the 1-WL, WLOA, δ-k-LWL, the δ-k-LWL$^+$, and the k-WL were selected from $\{0, \ldots, 5\}$ using 10-fold cross validation on the training folds only, or using the validation set for the medium-scale datasets.[11] Moreover, for the δ-k-LWL$^+$, we only added the additional label function $\#$ on the last iteration to prevent overfitting. We report

[11] As already shown in (Shervashidze et al., 2011), choosing the number of iterations too large will lead to overfitting.

Table 6: Mean MAE (mean std. MAE, logMAE) on large-scale (multi-target) molecular regression tasks.

	Method	Dataset					
		ZINC (10k)	ZINC (50k)	ZINC (FULL)	ALCHEMY (10K)	ALCHEMY (50K)	ALCHEMY (FULL)
Baseline	GINE-ε	**0.278** ±0.022	0.145 ±0.006	0.084 ±0.004	0.185 ±0.007 -1.864 ±0.062	0.127 ±0.004 -2.415 ±0.053	0.103 ±0.001 -2.956 ±0.029
	2-WL-GNN	0.399 ±0.006	0.357 ±0.017	0.133 ±0.013	0.149 ±0.004 -2.609 ±0.029	0.105 ±0.001 -3.139 ±0.020	0.093 ±0.001 -3.394 ±0.035
	δ-2-GNN	0.374 ±0.022	0.150 ±0.064	**0.042** ±0.003	**0.118** ±0.001 -2.679 ±0.044	**0.085** ±0.001 -3.239 ±0.023	**0.080** ±0.001 -3.516 ±0.021
	δ-2-LGNN	0.306 ±0.044	**0.100** ±0.005	0.045 ±0.006	0.122 ±0.003 -2.573 ±0.078	0.090 ±0.001 -3.176 ±0.020	0.083 ±0.001 -3.476 ±0.025

Table 7: Overall computation times for the whole datasets in seconds (Number of iterations for 1-WL, 2-WL, 3-WL, δ-2-WL, WLOA, δ-3-WL, δ-2-LWL, and δ-3-LWL: 5), OOT— Computation did not finish within one day (24h), OOM— Out of memory.

	Graph Kernel	Dataset							
		ENZYMES	IMDB-BINARY	IMDB-MULTI	NCI1	NCI109	PTC_FM	PROTEINS	REDDIT-BINARY
Baseline	GR	<1	<1	<1	1	1	<1	<1	2
	SP	<1	<1	<1	2	2	<1	<1	1 035
	1-WL	<1	<1	<1	2	2	<1	<1	2
	WLOA	<1	<1	<1	14	14	<1	1	15
Global	2-WL	302	89	44	1 422	1 445	11	14 755	OOM
	3-WL	74 712	18 180	5 346	OOT	OOT	5 346	OOM	OOM
	δ-2-WL	294	89	44	1 469	1 459	11	14 620	OOM
	δ-3-WL	64 486	17 464	5 321	OOT	OOT	1119	OOM	OOM
Local	δ-2-LWL	29	25	20	101	102	1	240	59 378
	δ-2-LWL$^+$	35	31	24	132	132	1	285	84 044
	δ-3-LWL	4 453	3 496	2 127	18 035	17 848	98	OOM	OOM
	δ-3-LWL$^+$	4 973	3 748	2 275	20 644	20 410	105	OOM	OOM

computation times for the 1-WL, WLOA, the δ-k-LWL, the δ-k-LWL$^+$, and the k-WL with five refinement steps. All kernel experiments were conducted on a workstation with an Intel Xeon E5-2690v4 with 2.60GHz and 384GB of RAM running Ubuntu 16.04.6 LTS using a single core. Moreover, we used the GNU C++ Compiler 5.5.0 with the flag `-O2`.

Neural architectures For comparing to kernel approaches, see Tables 4 and 5, we used 10-fold cross-validation. For the small-scale datasets, 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 selected using a validation set sampled uniformly at random from the training fold (using 10% of the training fold). For the medium-scale datasets, due to computation time constraints, we set the number of (hidden) node features to 64 and the number of layers to 3. We used mean pooling to pool the learned node embeddings to a graph embedding and used 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 accuracy 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, ZINC and ALCHEMY, see Table 6, we closely followed the hyperparameters found in (Dwivedi et al., 2020) and (Chen et al., 2019a), respectively, for the GINE-ε layers. That is, for ZINC, we used four GINE-ε layers with a hidden dimension of 256 followed by batch norm and a 4-layer MLP for the joint regression of the twelve targets, after applying mean pooling. For ALCHEMY and QM9, 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 exactly the same hyperparameters for the (local) δ-2-LGNN, and the dense variants δ-2-GNN and 2-WL-GNN.

For ZINC, we used the given train, validation split, test split, and report the MAE over the test set. For the ALCHEMY and QM9 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 (Klicpera et al., 2020), we report mean standardized MAE and mean standardized logMAE. We repeated each experiment five times (with different random splits in case of ALCHEMY and QM9) and report average scores and standard deviations.

Table 8: Overall computation times for the whole datasets in seconds on medium-scale datasets (Number of iterations for 1-WL, δ-2-LWL, and δ-3-LWL: 2).

	Graph Kernel	Dataset					
		YEAST	YEASTH	UACC257	UACC257H	OVCAR-8	OVCAR-8H
	1-WL	11	19	6	10	6	10
Local	δ-2-LWL	1 499	5 934	1 024	3 875	1 033	4 029
Local	δ-2-LWL$^+$	2 627	7 563	1 299	4 676	1 344	4 895

Table 9: Additional results for the neural architectures.

(a) Mean std. MAE and mean std. logMAE compared to (Maron et al., 2019a; Morris et al., 2019; Gilmer et al., 2017)

	Method	Dataset
		QM9
Baseline	GINE-ε	0.081 $\pm$0.003 -3.400 $\pm$0.094
Baseline	MPNN (Gilmer et al., 2017)	0.034 $\pm$0.001 **-4.156** $\pm$0.030
Baseline	1-2-GNN (Morris et al., 2019)	0.068 $\pm$0.001 -3.413 $\pm$0.025
Baseline	1-3-GNN (Morris et al., 2019)	0.088 $\pm$0.007 -2.704 $\pm$0.170
Baseline	1-2-3-GNN (Morris et al., 2019)	0.062 $\pm$0.001 -3.534 $\pm$0.030
Baseline	3-IGN (Maron et al., 2019a)	0.046 $\pm$0.001 -3.567 $\pm$0.022
	δ-2-LGNN	**0.029** $\pm$0.001 -4.054 $\pm$0.036

(b) Average speed up ratios over all epochs (training and testing)

	Method	Dataset	
		ZINC (10K)	ALCHEMY (10K)
Baseline	GINE-ε	0.2	0.4
Baseline	2-WL-GNN	2.2	1.1
Baseline	δ-2-GNN	2.5	1.7
	δ-2-LGNN	1.0	1.0

To compare training and testing times between the δ-2-LGNN, the dense variants δ-2-GNN and 2-WL-GNN, and the (1-dimensional) GINE-ε layer, we trained all four models on ZINC (10K) and ALCHEMY (10K) to convergence, divided by the number of epochs, and calculated the ratio with regard to the average epoch computation time of the δ-2-LGNN (average computation time of dense/baseline layer divided by average computation time of the δ-2-LGNN). All neural experiments were conducted on a workstation with four Nvidia Tesla V100 GPU cards with 32GB of GPU memory running Oracle Linux Server 7.7.

We trained all neural architecutes (excluding 3-IGN (Maron et al., 2019a)) withADAM (Kingma & Ba, 2015) with a learning rate decay of 0.5 and a patience parameter of 5, a starting learning rate of 0.01 and a minimum of 10^{-6}. For the 3-IGN, we used the parameters provided by Maron et al. as this lead to better results (a learning rate of 0.8 with a patience parameter of 20).

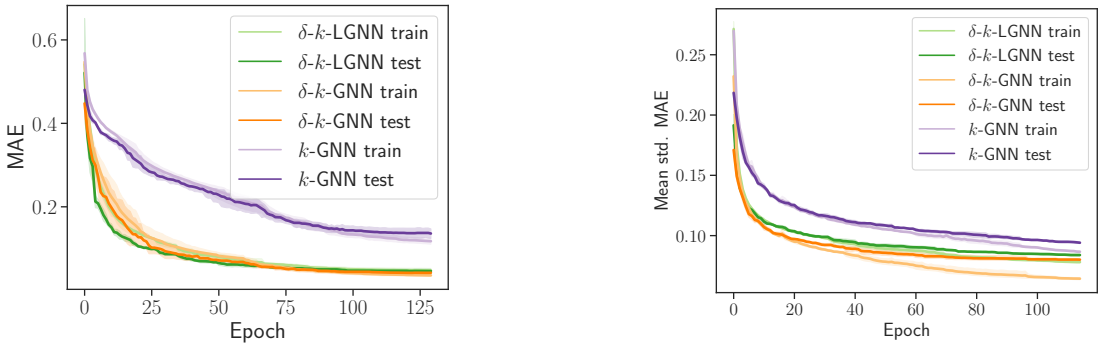
G.3. Results and discussion

In the following we answer questions **Q1** to **Q3**.

A1 **Kernels.** See Table 4. The local algorithm, for $k = 2$ and 3, severely improves the classification accuracy compared to the k-WL and the δ-k-WL. For example, on the ENZYMES dataset the δ-2-LWL achieves an improvement of almost 20%, and the δ-3-LWL achieves the best accuracies over all employed kernels, improving over the 3-WL and the δ-3-WL by almost 15%. This observation holds over all datasets (excluding PTC_FM and PROTEINS). However, it has to be noted that increasing k does not always result in increased accuracies. For example, on all datasets (excluding ENZYMES), the performance of the δ-2-LWL is better or on par with the δ-3-LWL. Hence, with increasing k the local algorithm is more prone to overfitting. Our algorithms also perform better than neural baselines.

Neural architectures. See Table 6. On the ZINC and ALCHEMY datasets, the δ-2-LGNN is on par or slightly worse than the δ-2-GNN. Hence, this is in contrast to the kernel variant. We assume that this is due to the δ-2-GNN being more flexible than its kernel variant, in weighing the importance of global and local neighbors. This is further highlighted by the worse performance of the 2-WL-GNN, which performs worse than GINE-ε on the ZINC dataset. Overall, the δ-k-LGNN performs better than the GINE-ε baseline on the (full) datasets. However, it seems less data-efficient than the GINE-ε as it performs worse on smaller ZINC (10K). However, this is not the case for the ALCHEMY (10K) dataset, which contains smaller graphs.

On the QM9 dataset, see Table 9, the δ-2-GNN performs better than higher-order methods from (Maron et al., 2019a) and (Morris et al., 2019) as well as the GINE-ε baseline while being on par with the MPNN architecture. We note here that



(a) ZINC (FULL)

(b) ALCHEMY (FULL)

Figure 6: Train and test error on ZINC (FULL) and ALCHEMY (FULL) over five runs.

the MPNN was specifically tuned to the QM9 dataset, which is not the case for the δ-2-LGNN (and the other higher-order architectures).

A2 See Tables 4 and 5. The δ-2-LWL$^+$ improves over the δ-2-LWL on all datasets excluding ENZYMES. For example, on IMDB-BINARY, IMDB-MULTI, NCI1, NCI109, and PROTEINS the algorithm achieves an improvement over of 4%, respectively, achieving a new state-of-the-art. The computation times are only increased slightly. Similar results can be observed on the larger datasets, see Table 5.

A3 **Kernels.** See Table 7. The local algorithm severely speeds up the computation time compared to the δ-k-WL and the k-WL for $k = 2$ and 3. For example, on the ENZYMES dataset the δ-2-LWL is over 10 times faster than δ-2-WL. The δ-3-LWL is even 14 times faster than the δ-3-WL. The improvement of the computation times can be observed across all datasets. For some datasets, the $\{2, 3\}$-WL and δ-$\{2, 3\}$-WL did not finish within the given time limit or went out of memory. For example, on four out of eight datasets, the δ-3-WL is out of time or out of memory. In contrast, for the corresponding local algorithm, this happens only two out of eight times. Hence, the local algorithm is more suitable for practical applications.
Neural architectures. See Table 9. The local algorithm severely speeds up the computation time of training and testing. Especially, on the ZINC dataset, which has larger graphs compared to the ALCHEMY dataset, the δ-2-GNN achieves a computation time that is more than two times lower compared to the δ-2-GNN and δ-2-WL-GNN.

H. Proof of Proposition 3

The weaker condition δ-k-WL $\sqsubseteq$ k-WL is straightforward because $M_i^{\delta,\overline{\delta}}(\mathbf{v}) = M_i^{\delta,\overline{\delta}}(\mathbf{w})$ implies $M_i(\mathbf{v}) = M_i(\mathbf{w})$ for all $\mathbf{v}$ and $\mathbf{w}$ in $V(G)^k$ and $i \geq 0$. Hence, it remains to show the strictness condition, i.e., δ-k-WL $\sqsubset$ k-WL. It suffices to show 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-WL distinguishes G_k and H_k. We proceed to the construction of this family, which is based on the classic result of (Cai et al., 1992).

Construction. Let K denote the complete graph on $k + 1$ vertices (there are no loops in K). The vertices of K are numbered from 0 to k. Let $E(v)$ denote the set of edges incident to v in K: clearly, $|E(v)| = k$ for all $v \in V(K)$. Define the graph G as follows:

1. For the vertex set $V(G)$, 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 $E(G)$, 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 $e \notin S$,

Define a companion graph H, in a similar manner to G, 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 $|V(G)| = |V(H)| = (k + 1) \cdot 2^{k-1} + \binom{k}{2} \times 2$. This finishes

the construction of graphs G and H. We set $G_k := G$ and $H_k := H$.

A set S of vertices is said to form a *distance-two-clique* if the distance between any two vertices in S is exactly two.

Lemma 11. The following holds for graphs G and H defined above.

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

Hence, G and H are non-isomorphic.

Proof. In the graph G, consider the vertex subset $S = \{(0,\emptyset),(1,\emptyset),\ldots,(k,\emptyset)\}$ of size $(k+1)$. That is, from each "cloud" of vertices of the form (v,S) for a fixed v, we pick the vertex corresponding to the trivial even subset, the empty set denoted by $\emptyset$. Observe that any two vertices in S are at distance two from each other. This holds because for any $i,j \in V(K)$, $(i,\emptyset)$ is adjacent to $\{i,j\}^0$ which is adjacent to $(j,\emptyset)$ (e.g. see Figure 1). Hence, the vertices in S form a distance-two-clique of size $k+1$.

On the other hand, for the graph H, suppose there exists a distance-two-clique, say $(0,S_0),\ldots,(k,S_k)$ in H, where each $S_i \subseteq E(i)$. If we compute the parity-sum of the parities of $|S_0|,\ldots,|S_k|$, we end up with 1 since there is exactly one odd subset in this collection, viz. S_0. On the other hand, we can also compute this parity-sum in an edge-by-edge manner: for each edge $(i,j) \in E(K)$, since (i,S_i) and (j,S_j) are at distance two, either both S_i and S_j contain the edge $\{i,j\}$ or neither of them contains $\{i,j\}$: hence, the parity-sum contribution of S_i and S_j to the term corresponding to e is zero. Since the contribution of each edge to the total parity-sum is 0, the total parity-sum must be zero. This is a contradiction, and hence, there does not exist a distance-two-clique in H. □

Next, we show that the local algorithm δ-k-LWL can distinguish G and H. Since δ-k-$\mathsf{WL} \sqsubseteq \delta$-$k$-$LWL$, the above lemma implies the strictness condition δ-k-$\mathsf{WL} \sqsubset k$-WL.

Lemma 12. δ-k-LWL distinguishes G and H.

Proof. The proof idea is to show that δ-k-LWL algorithm is powerful enough to detect distance-two-cliques of size $(k+1)$, which ensures the distinguishability of G and H. Indeed, consider the k-tuple $P = ((1,\emptyset),(2,\emptyset),\ldots,(k,\emptyset))$ in $V(G)^k$. We claim that there is no tuple Q in $V(H)^k$ such that the unrolling of P is isomorphic to the unrolling of Q. Indeed, for the sake of contradiction, assume that there does exist Q in $V(H)^k$ such that the unrolling of Q is isomorphic to the unrolling of P. Comparing isomorphism types, we know that the tuple Q must be of the form $((1,S_1),\ldots,(k,S_k))$.

Consider the depth-two unrolling of P: from the root vertex P, we can go down via two local-edges labeled 1, to hit the tuple $P_2 = ((2,\emptyset),(2,\emptyset),\ldots,(k,\emptyset))$. If we consider the depth-two unrolling of Q, the isomorphism type of P_2 implies that the vertices $(1,S_1)$ and $(2,S_2)$ must be at distance-two in the graph H. Repeating this argument, we obtain that $(1,S_1),\ldots,(k,S_k)$ form a distance-two-clique in H of size k. Our goal is to produce a distance-two-clique in H of size k, for the sake of contradiction.

For that, consider the depth-four unrolling of P: from the root vertex P, we can go down via two local-edges labeled 1 to hit the tuple $R = ((0,\emptyset),(2,\emptyset),\ldots(k,\emptyset)$. For each $2 \le j \le k$, we can further go down from R via two local edges labeled j to reach a tuple whose 1^{st} and j^{th} entry is $(0,\emptyset)$. Similarly, for the unrolling of Q, there exists a subset $S_0 \subseteq E(0)$ and a corresponding tuple $R' = ((0,S_0),(2,S_2),\ldots,(k,S_k))$, such that for each $2 \le j \le k$, we can further go down from R' via two local edges labeled j to reach a tuple whose 1^{st} and j^{th} entry is $(0,S_0)$. Comparing the isomorphism types of all these tuples, we deduce that $(0,S_0)$ must be at distance two from each of (i,S_i) for $i \in [k]$. This implies that the vertex set $\{(0,S_0),(1,S_1),\ldots,(k,S_k)\}$ is a distance-two-clique of size $k+1$ in H, which is impossible. Hence, there does not exist any k-tuple Q in $V(H)^k$ such that the unrolling of P and the unrolling of Q are isomorphic. Hence, δ-k-LWL distinguishes G and H. □