The Power of the Weisfeiler-Leman Algorithm for Machine Learning with Graphs

Christopher Morris¹, Matthias Fey² and Nils M. Kriege³
¹CERC in Data Science for Real-Time Decision-Making, Polytechnique Montréal
²Department of Computer Science, TU Dortmund University
³University of Vienna, Faculty of Computer Science, Vienna, Austria
chris@christopher.morris.info, matthias.fey@udo.edu, nils.kriege@univie.ac.at

Abstract

In recent years, algorithms and neural architectures based on the Weisfeiler-Leman algorithm, a well-known heuristic for the graph isomorphism problem, emerged as a powerful tool for (supervised) machine learning with graphs and relational data. Here, we give a comprehensive overview of the algorithm’s use in a machine learning setting. We discuss the theoretical background, show how to use it for supervised graph- and node classification, discuss recent extensions, and its connection to neural architectures. Moreover, we give an overview of current applications and future directions to stimulate research.

1 Introduction

Graph-structured data is ubiquitous across application domains ranging from chemoinformatics [Barabási and Oltvai, 2004; Stokes et al., 2020] to computer vision [Simonovsky and Komodakis, 2017], and social network analysis [Easley and Kleinberg, 2010]. To develop successful machine learning models in these domains, we need techniques to exploit the rich information inherent in the graph structure and the feature information within nodes and edges. Due to the highly non-regular structure of real-world graphs, we first have to learn a vectorial representation of each graph or node to apply standard machine learning tools such as support vector machines or neural networks. Here, numerous approaches have been proposed in recent years—most notably, approaches based on graph kernels [Kriege et al., 2020] or neural architectures [Gilmer et al., 2017; Chami et al., 2020]. Especially, graph kernels based on the Weisfeiler-Leman algorithm [Weisfeiler and Leman, 1968], and corresponding neural architectures, known as Graph Neural Networks (GNNs), have recently advanced the state-of-the-art in supervised node- and graph representation learning.

The (1-dimensional) Weisfeiler-Leman (1-WL) or color refinement algorithm is a well-known heuristic for deciding whether two graphs are isomorphic: Given an initial coloring or labeling of the nodes of both graphs, e.g., their degree or application-specific information, in each iteration, two nodes with the same label get different labels if the number of identically labeled neighbors is not equal. If, after some iteration, the number of nodes annotated with a certain label is different in both graphs, the algorithm terminates, and we conclude that the two graphs are not isomorphic. This simple algorithm is already quite powerful in distinguishing non-isomorphic graphs [Babai et al., 1980], and has been therefore applied in many areas [Grohe et al., 2014; Kersting et al., 2014; Li et al., 2016; Yao and Holder, 2015; Zhang and Chen, 2017]. On the other hand, it is easy to see that the algorithm cannot distinguish all non-isomorphic graphs [Cai et al., 1992]. For example, it cannot distinguish graphs with different triangle counts, cf. Figure 1, which is an important feature in social network analysis. Therefore, it has been generalized to $k$-tuples leading to a more powerful graph isomorphism heuristic, which has been investigated in depth by the theoretical computer science community [Cai et al., 1992; Kiefer and Schweitzer, 2016; Babai, 2016; Grohe, 2017]. In Shervashidze et al. [2011], the 1-WL was first used to obtain a graph kernel, the so-called Weisfeiler-Lehman subtree kernel. The kernel’s idea is to compute the 1-WL for a fixed number of steps, resulting in a color histogram or feature vector for each graph. The kernel is then computed by taking the pairwise inner product between these vectors. Hence, the kernel measures the similarity between two graphs by counting the number of common colors in all refinement steps. We note here that similar approaches are also used in chemoinformatics [Rogers and Hahn, 2010].

Graph kernels were dominant in graph classification for several years, which led to new state-of-the-art results on many graph classification tasks. However, they are limited because they cannot effectively adapt their feature representations to a given data distribution since they generally rely on a fixed set of pre-computed features. GNNs have emerged as a machine learning framework addressing the above limitations. Primarily, they can be viewed as a neural version of the 1-WL algorithm, where continuous feature vectors replace colors, and neural networks are used to aggregate over local node neighborhoods [Hamilton et al., 2017; Gilmer et al., 2017]. Recently, there has been progress in connecting the two above algorithms. That is, it has been shown that any possible GNN architecture cannot be more powerful than the 1-WL in terms of distinguishing non-isomorphic graphs [Morris et al., 2019; Xu et al., 2019a]. Moreover, this line of work has been extended by deriving more powerful neural architectures, e.g., based on the $k$-dimensional Weisfeiler-Leman algorithm, e.g., [Morris et al., 2019; Morris et al., 2020a] and [Maron et al., 2019], some of which we survey below.

Present work. Here, we give a comprehensive overview of the applications of the 1-WL and GNNs in machine learning for graph- and relational data. We discuss the theoretical background,
show how they can be used for supervised graph- and node classification, and discuss recent extensions and its connections. Moreover, we give an overview of current applications and future directions to stimulate research.

2 Background

A graph $G$ is a pair $(V, E)$ with a finite set of nodes $V(G)$ and a set of edges $E(G) \subseteq \{(u, v) \subseteq V \mid u \neq v\}$. For ease of notation, we denote an edge $(u, v)$ as $(u, v)$. A labeled graph $G$ is a triple $(V, E, l)$ with a label function $l: V(G) \rightarrow \Sigma$, where $\Sigma$ is some finite alphabet. Then $l(v)$ is a label of $v$ for $v \in V(G)$. The neighborhood of $v$ in $V(G)$ is denoted by $N(v) = \{u \in V(G) \mid (u, v) \in E(G)\}$. We say that two graphs $G$ and $H$ are isomorphic if there exists an edge preserving bijection $\phi: V(G) \rightarrow V(H)$, i.e., $(u, v) \in E(G)$ if and only if $(\phi(u), \phi(v))$ is in $E(H)$ for all $u, v \in V(G)$. In the case of labeled graphs, we additionally require that $l(v) = l(\phi(v))$ for $v \in V(G)$. We define $[n] = \{1, \ldots, n\} \subseteq \mathbb{N}$ for $n \geq 1$, and denote a multiset by $\{\ldots\}$. A kernel on a non-empty set $X$ is a positive semidefinite function $k: X \times X \rightarrow \mathbb{R}$. Equivalently, a function $k$ is a kernel if there is a feature map $\phi: X \rightarrow \mathcal{H}$ 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 $X$. Let $G$ be the set of all graphs, then a positive semidefinite function $G \times G \rightarrow \mathbb{R}$ is a graph kernel.

2.1 The Weisfeiler-Leman Algorithm

We now formally describe the 1-WL algorithm for labeled graphs. Let $G = (V, E, l)$ be a labeled graph, in each iteration, $t \geq 0$, the 1-WL computes a node coloring $c^{(t)}_1: V(G) \rightarrow \Sigma$, which depends on the coloring of the neighbors. In iteration $t > 0$, we set $$c^{(t)}_1(v) = \text{RELABEL}\left(\{c^{(t-1)}_1(v), \{c^{(t-1)}_1(u) \mid u \in N(v)\}\}\right),$$ where RELABEL injectively maps the above pair to a unique value in $\Sigma$, which has not been used in previous iterations. That is, in each iteration the algorithm computes a new color for a node based on the colors of its direct neighbors, cf. Figure 2 for an illustration. In iteration 0, we set $c^{(0)}_1 = l$. Hence, after $k$ iterations the color of a node reflects the structure of its $k$-hop neighborhood. To test if two graphs $G$ and $H$ are non-isomorphic, we run the above algorithm in “parallel” on both graphs. If the two graphs have a different number of nodes with a specific color, the 1-WL concludes that the graphs are not isomorphic. If the number of colors between two iterations does not change, i.e., the cardinalities of the images of $c^{(t-1)}_1$ and $c^{(t)}_1$ are equal, the algorithm terminates. Termination is guaranteed after at most $\max\{\lvert V(G)\rvert, \lvert V(H)\rvert\}$ iterations [Grohe, 2017].

A Hierarchy of More Powerful Algorithms. The $k$-WL was first proposed by László Babai, cf. [Cai et al., 1992], based on algorithms proposed by Weisfeiler and Leman, cf. [Weisfeiler and Leman., 1968; Weisfeiler, 1976]. To make the algorithm more powerful, it colors tuples from $V(G)^k$ for $k > 1$ instead of nodes. By defining a neighborhood between these tuples, we can define an update rule similar to the 1-WL. Formally, the algorithm computes a coloring $c^{(t)}_k: V(G)^k \rightarrow \Sigma$, and we define the $j$-th neighborhood $$N_j(s) = \{(s_1, \ldots, s_{j-1}, r, s_{j+1}, \ldots, s_k) \mid r \in V(G)\}$$ of a $k$-tuple $s = (s_1, \ldots, s_k)$ in $V(G)^k$. That is, the $j$-th neighborhood $N_j(s)$ of $s$ is obtained by replacing the $j$-th component of $s$ by every node from $V(G)$. In iteration 0, the algorithm labels each $k$-tuple with its atomic type, i.e., two $k$-tuples $s$ and $s'$ in $V(G)^k$ get the same color if the map $s_j \mapsto s'_j$ induces a (labeled) isomorphism between the subgraphs induced by the nodes from $s$ and $s'$, respectively. For iteration $t > 0$, we define $$c^{(t)}_k(s) = \text{RELABEL}\left(\{c^{(t-1)}_k(s'), s' \in N_j(s)\}\right), \quad (1)$$ and set $$c^{(t)}_k(s) = \text{RELABEL}\left(\{c^{(t-1)}_k(s'), (c^{(t-1)}_1(s), \ldots, c^{(t-1)}_k(s))\}\right).$$

Hence, two tuples $s$ and $s'$ with $c^{(t-1)}_k(s) = c^{(t-1)}_k(s')$ get different colors in iteration if there exists $j \in [k]$ such that the number of $j$-neighbors of $s$ and $s'$, respectively, colored with a certain color is different. The algorithm then proceeds analogously to the 1-WL.

By increasing $k$, the algorithm gets more powerful in distinguishing non-isomorphic graphs, i.e., for each $k \geq 2$, there are non-isomorphic graphs distinguished by the $(k + 1)$-WL but not by the $k$-WL [Cai et al., 1992]. We note here that the above variant is not equal to the folklore variant of $k$-WL described in [Cai et al., 1992], which differs slightly in its update rule. It holds that the $k$-WL using Equation (1) is as powerful as the folklore $(k-1)$-WL [Grohe, 2021].

Properties of the Algorithm. The Weisfeiler-Leman algorithm constitutes one of the earliest approaches to isomorphism testing [Weisfeiler, 1976; Weisfeiler and Leman., 1968; Immerman and Lander, 1990] and has been heavily investigated by the theory community over the last few decades [Grohe and Otto, 2015]. 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 and Lander, 1990], Sherali-Adams relaxations of the natural integer linear program for the graph isomorphism problem [Atserias and Maneva, 2013; Grohe and Otto, 2015; Malkin, 2014], homomorphism counts [Dell et al., 2018], and quantum isomorphism games [Atserias et al., 2019]. In their seminal paper, Cai et al. [1992] showed that for each $k$ there exists a pair of non-isomorphic graphs of size $O(k)$ each that the $k$-WL cannot distinguish. Grohe and Otto [2015] 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 [Berkholz et al., 2013], and the number of iterations for $k = 1$ [Kiefer and McKay, 2020], and the number of iterations for $k = 2$ [Kiefer and Schweitzer, 2016; Lichter et al., 2019] have been shown. For $k = 1$ and 2, Arvind et al. [2019] studied the abilities of the (folklore) $k$-WL to detect and count fixed subgraphs, extending the work of Fürer [2017]. The former was refined in Chen et al. [2020]. The algorithm (for logarithmic $k$) plays a prominent role in the recent result of Babai [2016] improving the best-known running time for solving the graph isomorphism problem. Recently, Grohe et al. [2020] introduced the framework of Deep Weisfeiler Leman algorithms, which allow the design of a more powerful graph isomorphism test than Weisfeiler-Leman type algorithms.

3 Applications to Machine Learning

We now formally define the Weisfeiler-Lehman subtree kernel [Shervashidze et al., 2011]. The idea is to compute the 1-WL for $h \geq 0$ iterations resulting in a coloring $c^{(i)}: V(G) \rightarrow \Sigma_i$ for each iteration $i$. 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 nodes labeled by $c$ in $\Sigma_i$. The overall feature vector $\phi_{WL}(G)$ is defined as the concatenation of the feature vectors over all $h$ iterations, i.e., $\phi_{WL}(G) = [\phi_0(G), \ldots, \phi_h(G)]$. The corresponding kernel for $h$ iterations then is computed as $k_{WL}(G, H) = \langle \phi_{WL}(G), \phi_{WL}(H) \rangle$, where $\langle \cdot, \cdot \rangle$ denotes the standard inner product. The running time for a single feature vector computation is in $O(hm)$ and $O(Nh m + N^2 h n)$ for the computation of the Gram matrix for a set of $N$ graphs [Shervashidze et al., 2011], where $n$ and $m$ denote the maximum number of nodes and edges, respectively, over all $N$ graphs. The algorithm scales well to large graphs and datasets, and can be used together with linear SVMs to avoid the quadratic overhead of computing the Gram matrix. For bounded-degree graphs, the kernel can be even approximated in constant time [Morris et al., 2017], i.e., the running time is independent of the number of nodes of the graph.

Variants. The Weisfeiler-Lehman optimal assignment kernel is defined as the weight of an optimal assignment between the vertices of two graphs, where the similarity between pairs of vertices is determined by their 1-WL colors [Kriege et al., 2016]. Similarly, the Wasserstein Weisfeiler-Lehman graph kernel [Togninalli et al., 2019] is obtained from the Wasserstein distance wrt. a ground metric obtained from 1-WL. Nikolentzos et al. [2017] introduced the concept of optimal assignments in the neighborhood aggregation step. Rieck et al. [2019] combined the 1-WL with persistent homology to extract topological features such as cycles. Finally, Nguyen and Maehara [2020] leveraged the link between WL features and graph homomorphisms, established by Dell et al. [2018], to define graph kernels.

Continuous Information. Most real-world graphs have attributes, mostly real-valued vectors, associated with their nodes and edges. For example, atoms of chemical molecules have physical and chemical properties, individuals in social networks have demographic information, and words in documents carry semantic meaning. The 1-WL and the corresponding kernel are discrete. That is, they can only deal with a discrete (countable) label alphabet. Hence, two nodes are regarded as similar if and only if they exactly match, structure-wise and attribute-wise. However, in most applications, it is desirable to compare real-valued attributes with more nuanced similarity measures such as the Gaussian RBF kernel. Hence, extensions of the 1-WL kernel have been proposed that can deal with continuous information [Orsini et al., 2015; Togninalli et al., 2019]. For example, a basic instantiation of the GraphInvariant kernel [Orsini et al., 2015], can be expressed as

$$k_{WV}(G, H) = \sum_{v \in V(G)} \sum_{v' \in V(H)} k_V(v, v') \cdot k_W(c_{h}^{(h)}(v), c_{h}^{(h)}(v')).$$

Here, $k_V$ is a user-specified kernel comparing (continuous) node attributes, and $k_W$ is a kernel determining a weight for a node pair based on their color given by the 1-WL after $h$ iterations. However, due to the quadratic overhead of computing the above kernel for each pair of graphs, the algorithm does not scale to large datasets. Therefore, Morris et al. [2016] introduced a scalable framework to compare attributed graphs. The idea is to iteratively turn the continuous attributes of a graph into discrete labels using randomized hash functions. This allows applying fast explicit graph feature maps, which are limited to graphs with discrete annotations such as the one associated with the Weisfeiler-Lehman subtree kernel. For special hash functions, the authors obtain approximation results for several state-of-the-art kernels handling continuous information. Moreover, they derived a variant of the Weisfeiler-Lehman subtree kernel, which can handle continuous attributes.

Global Information. Due to the purely local nature of the 1-WL, it might miss essential patterns in the given data. Hence, recently, graph kernels based on the $k$-WL have been proposed. In Morris et al. [2017], a set-based version of the $k$-WL is employed to derive a kernel. More recently, a local variant of the $k$-WL is proposed [Morris et al., 2020b], which considers a subset of the original neighborhood in each iteration. The cardinality of this local neighborhood only depends on the graph's sparsity. The authors showed that the local algorithm has at least the same
power as the original $k$-WL, prevents overfitting, and leads to state-of-the-art results on standard benchmark datasets [Morris et al., 2020a].

## 4 Graph Neural Networks

Intuitively, GNNs compute a vectorial representation, i.e., a $d$-dimensional vector, for each node in a graph by aggregating information from neighboring nodes. Each layer of a GNN aggregates local neighborhood information, i.e., neighbors’ features, within each node and then passes this aggregated information on to the next layer. Following [Gilmer et al., 2017], in full generality, a new feature $f^{(t)}(v)$ for a node $v$ is computed as

$$f^{W_1}_{\text{merge}} \left( f^{(t-1)}(v), f^{W_2}_{\text{agg}}(\{f^{(t-1)}(w) \mid w \in N(v)\}) \right), \quad (2)$$

where $f^{W_1}_{\text{merge}}$ aggregates over the multiset of neighborhood features and $f^{W_2}_{\text{agg}}$ merges the node’s representations from step $(t-1)$ with the computed neighborhood features. Both $f^{W_2}_{\text{agg}}$ and $f^{W_1}_{\text{merge}}$ may be arbitrary differentiable functions with parameters $W_1$ and $W_2$. See Figure 2 for an illustration of the architecture. A vector representation $f_{\text{GNN}}$ over the whole graph $G$ can be computed by aggregating the vector representations computed for all nodes, i.e., $f_{\text{GNN}}(G) = \sum_{v \in V(G)} f^{(T)}(v)$, where $T > 0$ denotes the last layer. More refined approaches use different pooling operators based on sorting [Zhang et al., 2018] or soft assignments [Ying et al., 2018]. Efficient GPU-based implementations of many GNN architectures can be found in [Fey and Lenssen, 2019], [Grattarola and Alippi, 2020], and [Wang et al., 2019].

### Connections to the Weisfeiler-Leman Algorithm.

A recent line of work [Morris et al., 2019; Xu et al., 2019a; Maron et al., 2019] connects the expressivity of GNNs to that of the $1$-WL algorithm. The results show that GNN architectures do not have more power to distinguish between non-isomorphic (sub-)graphs than the $1$-WL. More formally, let $f^{W_1}_{\text{merge}}$ and $f^{W_2}_{\text{agg}}$ be any two functions chosen in Equation (2). For every encoding of the labels $l(v)$ as vectors $f^{(0)}(v)$, and for every choice of $W_1$ and $W_2$, we have that if the GNN parameterized by the above weights distinguishes a pair of graphs, the $1$-WL also will. On the positive side, it has been shown that there exists a GNN architecture and corresponding weights such that it has the same power as the $1$-WL. That is, we get the following insight.

**Insight 1.** [Morris et al., 2019; Xu et al., 2019a] Any possible graph neural network architecture can be at most as powerful as the $1$-WL in terms of distinguishing non-isomorphic graphs.

A GNN architecture has the same power as the $1$-WL if the functions $f^{W_1}_{\text{merge}}$ and $f^{W_2}_{\text{agg}}$ are injective.

For a detailed discussion, see [Grove, 2021]. Hence, in light of the above results, GNNs may be viewed as an extension of the $1$-WL, which has the same power but is more flexible in adapting to the learning task at hand and can handle continuous node features. Moreover, the above results have been generalized to the $k$-WL. That is, Maron et al. [2019] and Morris et al. [2020b] derived neural architectures with the same power as the former.

**Insight 2.** [Maron et al., 2019] There exists a GNN architecture that has the same power as the $(k + 1)$-WL in terms of distinguishing non-isomorphic graphs.

Recently, designing GNNs that overcome the limitations of the $1$-WL received much attention.

### Provably Powerful Graph Neural Networks.

Morris et al. [2019] extended the expressivity by proposing a higher-order GNN layer, which passes messages between subgraphs instead of vertices by defining a suitable notion of neighborhood between subgraphs. Murphy et al., Vignac et al. [2019; 2020] utilize vertex identifiers as node features to maintain information about which vertex in the receptive field has contributed to the aggregated information, leading to provably more powerful architectures. A similar idea was utilized in Sato et al. [2020], which includes additional random vertex features instead of vertex identifiers to the message passing phase. The latter was refined by Dasoulas et al.; Abboud et al. [2020; 2020] which investigated the connection between random coloring and universality. Finally, Bouritsas et al. [2020] used higher-order topology as features, while Li et al.; You et al. [2019; 2019] encoded distance information. These generalized GNN architectures show promising results, e.g., on regression tasks of quantum-chemical properties of molecules [J. Klicpera, 2020; Morris et al., 2019; Morris et al., 2020b]. Further, [Azizian and Lelarge, 2020] connect the universality of invariant and equivariant neural networks of graphs to the $k$-WL hierarchy.

## 5 Applications

Graphs define a universal language for describing complex, relational data and hence arise in a wide range of domains across supervised, semi-supervised and unsupervised settings and include tasks such as node- and graph classification, link- and community detection, graph similarity, and graph generation. Many real-world structures can be naturally modeled as a graph, e.g., physical systems, molecular structures, social networks, and knowledge graphs. However, even when there is no explicit graph structure available, underlying relational information can be synthetically induced to strengthen a model’s performance, e.g., for vision or natural language tasks. In the following, we provide a (non-exhaustive) list of applications relying on WL-based graph similarity, either by kernels or GNNs approaches.

### Graph Kernels and 1-WL

Since the introduction of the Weisfeiler-Lehman subtree kernel, the 1-WL has been applied to many application areas, ranging from chem- and bioinformatics [Stöcker et al., 2019], to neuroscience [Vega-Pons and Avesani, 2013], link prediction [Zhang and Chen, 2017], dimension reduction of systems of linear equations and linear programs [Grohe et al., 2014], graph matching [Kriege et al., 2019], RDF data [de Vries, 2013], malware detection [Narayanan et al., 2016], and detection of similar computer programs [Li et al., 2016]. Experimental studies, e.g., [Kriege et al., 2020; Morris et al., 2020a], show that the kernel is still a useful and competitive baseline on a wide range of graph classification tasks.

### Graph Neural Networks

Since the recent advancements in deep learning, research on GNN architectures has been thriving, and they are widely used in tasks with an underlying graph structure, e.g., social network prediction [Hamilton et al., 2017], traffic prediction [Yu et al., 2018] and recommender systems [Ying et al., 2018a]. They also...
enable reasoning in knowledge graphs [Schlichtkrull et al., 2018], such as answering queries from a knowledge database [Ren et al., 2020] or for cross-lingual knowledge alignment [Xu et al., 2019b; Fey et al., 2020]. Furthermore, relational information also occurs in real-world physical systems where objects are represented as nodes and their relations as edges. Graph-based learning then enables reasoning about those objects, their relations, and physics effectively [Kipf et al., 2018].

Chemoinformatics. In chemoinformatics, molecules can be naturally represented as graphs and relational reasoning enables the prediction of chemical properties and biological interaction of proteins [Duvenaud et al., 2015; Gilmer et al., 2017; Zitnik et al., 2018; J. Klicpera, 2020]. Both, pharmaceutical companies and academia, have an increasing interest in GNNs for computer-aided drug discovery [Wieder et al., 2020].

Computer Vision. Graphs have been also becoming popular in the computer vision domain for learning on scene graphs [Kaposo et al., 2017], image keypoints [Li et al., 2019; Fey et al., 2020], superpixels [Monti et al., 2017], 3D point clouds [Qi et al., 2017] and manifolds [Fey et al., 2018; Hanocka et al., 2019] where relational learning introduces an important inductive bias into the model. By incorporating both spatial and semantic information, these models tend to outperform their non-relational counterparts to a large extent.

Generative Models. Applications like discovering new chemical structures involve the need to synthesize real-world graphs using generative modeling. Here, graph-based learning is either used to encode the graph into a low-dimensional embedding space [Simonovsky and Komodakis, 2018] or to estimate the quality of generated samples in a GAN setup [De Cao and Kipf, 2018]. Generative models can be further distinguished by creating adjacency information at once [De Cao and Kipf, 2018] or sequentially in an autoregressive fashion [You et al., 2018; Liao et al., 2019].

Combinatorial Optimization. Recently, GNNs have been used aiding to solve NP-hard combinatorial optimization problem. For example, in Gasse et al.; Li et al. [2019; 2018] they are used to guide tree search algorithms for solving combinatorial optimization to optimality. They have also been used for similar tasks in a Q-learning setting [Khalil et al., 2017]. For a thorough overview, see [Cappart et al., 2021].

6 Discussion and Future Directions

As outline above, Weisfeiler-Leman type methods have been proven to be useful for learning with graphs. However, the learning performance of even the 1-WL is still poorly understood. To that, we propose several directions to stimulate future research.

Trading of Expressivity and Generalization. Although the 1-WL’s expressivity is well understood, it has not been sufficiently acknowledged that expressivity is not a significant concern for wide-spread benchmark datasets [Morris et al., 2020a]. To confirm this, we have computed the completeness ratio, i.e., the fraction of graphs that can be distinguished from all other non-isomorphic graphs in the dataset, see 3, revealing that the 1-WL is sufficiently expressive to distinguish all the non-isomorphic graphs. Thus, although devising provably powerful graph learning architectures is a meaningful theoretical endeavor, the key to improving real-world tasks is improving GNN’s generalization abilities. So far, only a few notable contributions in this direction have been made [Kriege et al., 2018; Garg et al., 2020]. Hence, we believe that more strides should be taken to understand the generalization performance of the 1-WL and GNNs.

Locality. The number of iterations of the 1-WL or GNN layers is typically selected by cross-validation and often small, e.g., ≤ 5. For larger values 1-WL’s features become too specific, leading to overfitting for graph kernels, while the GNNs’ node features become indistinguishable, a phenomenon referred to as over-smoothing [Liu et al., 2020]. Moreover, for GNNs, the bottleneck problem refers to the observation that large neighborhoods cannot be accurately represented [Alon and Yahav, 2020]. This prevents both methods from capturing global or long-range information. We believe that the development of new techniques to overcome these issues is a promising research direction.

Incorporating expert knowledge. Nowadays, kernels based on the 1-WL and GNNs are heavily used in application areas from the life sciences to engineering. However, their usage is often ad-hoc, not leveraging crucial expert knowledge. In chemoinformatics, for example, information on functional groups or pharmacophore properties is often available, but it is not straightforward to explicitly incorporate it into the 1-WL and GNN pipeline. Hence, we view the development of mechanisms to include such knowledge as a crucial step making WL-based learning with graphs more applicable to real-world domains.

7 Conclusion

We gave a comprehensive overview of applications of the Weisfeiler-Leman algorithm to machine learning for graph data. Moreover, we introduced GNNs and outlined their connections to the former and applications. We believe that our survey will spark further interest in Weisfeiler-Leman type algorithms, their connections to neural architectures, and their application to solving real-world problems.

Acknowledgements

NMK has been funded by the Vienna Science and Technology Fund (WWTF) through project VRG19-009. MF has been supported by the German Research Association (DFG) within the Collaborative Research Center SFB 876, project A6.
References


