Towards Efficient and Expressive GNNs for Graph Classification via Subgraph-aware Weisfeiler-Lehman

Anonymous Author(s)
Anonymous Affiliation
Anonymous Email

Abstract

The expressive power of GNNs is upper-bounded by the Weisfeiler-Lehman (WL) test. To achieve GNNs with high expressiveness, researchers resort to subgraph-based GNNs (WL/GNN on subgraphs), deploying GNNs on subgraphs centered around each node to encode subgraphs instead of rooted subtrees like WL. However, deploying multiple GNNs on subgraphs suffers from much higher computational cost than deploying a single GNN on the whole graph, limiting its application to large-size graphs. In this paper, we propose a novel paradigm, namely Subgraph-aware WL (SaWL), to obtain graph representation that reaches subgraph-level expressiveness with a single GNN. We prove that SaWL has beyond-WL capability for graph isomorphism testing, while sharing similar runtime to WL. To generalize SaWL to graphs with continuous node features, we propose a neural version named Subgraph-aware GNN (SaGNN) to learn graph representation. Both SaWL and SaGNN are more expressive than 1-WL while having similar computational cost to 1-WL/GNN, without causing exponentially higher complexity like other more expressive GNNs. Experimental results on several benchmark datasets demonstrate that fast SaWL and SaGNN significantly outperform competitive baseline methods on the task of graph classification, while achieving high efficiency.

1 Introduction

Graph-structured data widely exist in the real world, and modeling graphs has become an important topic in the field of machine learning. Graph learning has widespread applications [1–3], and many valuable applications can be formulated as graph classification, e.g., molecular property prediction [4], drug toxicity prediction [5]. Graph classification aims to predict the label of the given graph by exploiting graph structure and feature information. Learning expressive representations of graphs is crucial for classifying graphs of different structural characteristics.

Recently, Graph Neural Networks (GNNs) have achieved great success in graph classification tasks [6–8]. GNNs that follow a message passing scheme first iteratively aggregate neighbor information to update node representations, then pool node representations into graph-level representations [9]. Essentially, GNNs are parameterized generalizations of the 1-dimensional Weisfeiler-Lehman algorithm (1-WL) [10], which encodes each node by its rooted subtree pattern [11], as shown in Figure 1 (a). Despite the success of traditional message passing GNNs, the expressive power of GNNs is theoretically upper-bounded by 1-WL, which is known to have limited power in distinguishing many non-isomorphic graphs [12–14].

To uplift the expressive power of GNNs, researchers adopt a paradigm of *WL/GNN on subgraphs* (Figure 1 (b)), which encodes rooted subgraphs instead of rooted subtrees as node representations [15–17]. Methods under the paradigm first extract rooted subgraphs (i.e., subgraph induced by the neighbor nodes within h hops of a center node), and then apply GNNs on each extracted subgraph respectively. However, as GNNs are applied to subgraphs extracted from each node of the graph, the computational cost of these methods is much higher than that of traditional message passing GNNs, especially when the subgraphs have similar sizes to the whole graph.

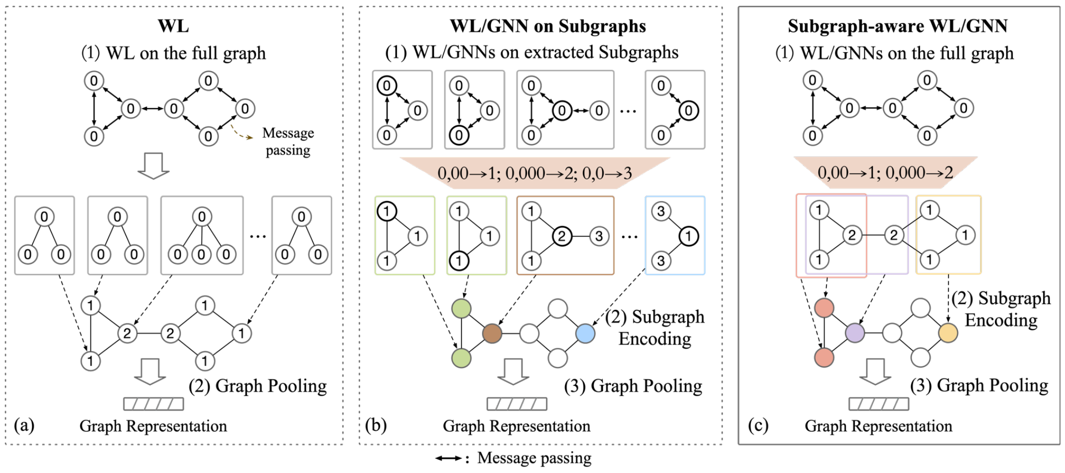


Figure 1: (a) WL encodes nodes by rooted subtrees, which has limited expressiveness. (b) WL/GNN on Subgraphs paradigm extracts rooted subgraphs and applies GNNs on each rooted subgraph, which is computationally expensive. (c) Our Subgraph-aware WL/GNN applies WL/GNN on the full graph and then encodes rooted subgraphs by aggregating nodes within the subgraph. The proposed paradigm possesses higher expressive power than 1-WL while keeping the computational cost low.

In this paper, we propose a novel paradigm of *Subgraph-aware WL/GNN* (**SaWL**), which reaches higher expressiveness than 1-WL with a single GNN (Figure 1 (c)). It first deploys WL/GNN on the full graph to obtain node representations, and then aggregates the nodes within each subgraph to achieve subgraph awareness. The proposed paradigm greatly reduces the computational cost of existing WL-on-subgraph methods, while achieving higher expressive power than 1-WL. Under the paradigm, we propose an algorithm as fast implementation of SaWL, which consists of a WL encoder and a subgraph operator (S operator). We first apply a standard WL on the full graph to iteratively update each node label based on its current label and the labels of its neighbors [18]. After each iteration of WL, we use the S operator to encode the rooted subgraph of each node by aggregating the current labels of nodes within the subgraph. The whole graph feature mapping at this iteration is obtained further by pooling the subgraph feature mapping. Finally, we concatenate graph feature mappings at different iterations into a final graph feature mapping for graph classification. We then generalize SaWL to a neural version, Subgraph-aware GNN (**SaGNN**).

Compared to the paradigm of *WL/GNN-on-subgraphs*, the proposed *Subgraph-aware WL/GNN* does not need to copy a full n-node graph into n subgraphs (each rooted at a node) and run WL/GNN on each subgraph separately (thus the same node can have multiple representations when appearing in different subgraphs). Instead, Subgraph-aware WL/GNN only runs WL/GNN on the full graph and encodes subgraph information based on the "global" WL/GNN node representations. It encodes the subgraph information while avoiding the need to apply WL/GNN on each extracted subgraph respectively, which improves the expressiveness and keeps low computational cost at the same time.

We evaluate the effectiveness of the proposed fast SaWL and SaGNN on graph classification tasks via several benchmark datasets, and then conduct the expressive power evaluation to verify the high distinguishing power of our methods. We further compare the running time of our methods with other high expressive methods. The experimental results show that our methods have both high effectiveness and high efficiency.

2 Preliminary

2.1 Weisfeiler-Lehman and Feature Mapping

Weisfeiler-Lehman (1-WL) [10] is one of the most widely used algorithms which can tackle graph isomorphism testing for a broad class of graphs [19, 20]. Specifically, 1-WL proceeds in iterations denoted by h, and each iteration includes multisets determination, injective mapping and relabeling [18].

Given two graphs G and H, **firstly**, WL aggregates the labels of neighbor nodes as a multiset M_v^h. For $h = 0$, $M_v^0 = l_v^0$, and for $h > 0$, $M_v^h = \{\!\{l_u^{h-1} | u \in \mathcal{N}(v)\}\!\}$, where l_v^h is the label of node v in the h-th iteration, $\mathcal{N}(v)$ denotes the neighbor nodes of v and $\{\!\{\}\!\}$ denotes a multiset. Note that multiset is a generalized set that allows repeated elements [13]. **Then**, an injective function is required to update the label of node, $l_v^h := \text{HASH}\left(\left(l_v^{h-1}, M_v^h\right)\right)$. The procedures repeat until the multisets of node labels of two graphs differ, the number of iterations reaches a predetermined value, or the node labels do not change in one iteration. ***The feature mapping*** of the whole graph can be obtained after each iteration. We can use the multiset of node labels in the h-th iteration to represent the whole graph [18]. Although 1-WL works well in testing isomorphism on many graphs, the distinguishing power of the 1-WL is limited [12, 21].

2.2 Graph Neural Networks

Traditional message passing Graph Neural Networks (GNNs) follow an aggregation and update scheme, which can be viewed as the neural implementation of the 1-WL [13, 22]. Nodes aggregate features of neighbor nodes, combine them with its features and update to new representations:

$$\boldsymbol{h}_v^k = \text{UPDATE}\left(\boldsymbol{h}_v^{k-1}, \text{AGGREGATE}\left(\boldsymbol{h}_u^{k-1} | u \in \mathcal{N}(v)\right)\right), \tag{1}$$

where the UPDATE and AGGREGATE functions are implemented with neural networks. Then, the whole graph representation can be computed by a pooling/readout operation like sum [23–25]:

$$\boldsymbol{h}^k(G) = \text{READOUT}\left(\boldsymbol{h}_v^k | v \in \mathcal{V}(G)\right). \tag{2}$$

GNNs have been popular architectures for representation learning on graphs. However, it has been proved that the expressive power of message passing GNNs is upper bounded by the 1-WL algorithm [13, 14], which limits the performance on graph classification tasks.

3 Subgraph-aware Weisfeiler-Lehman

We propose a new paradigm of **Subgraph-aware Weisfeiler-Lehman (SaWL)**, which exceeds the expressive power of 1-WL while keeping low computational complexity. The paradigm first iteratively applies WL/GNN to the original input graph. With the obtained node representations at each iteration, the paradigm encodes each rooted subgraph by hashing the node representations within its range. Then, the subgraph representations are pooled to obtain the whole graph representation.

3.1 SaWL for Graph Classification

SaWL consists of a WL encoder, a subgraph encoding operator (the S operator) and a graph feature mapping module. For graph G, the **WL encoder** executes normal WL steps described in section 2.1, which outputs the updated node labels $\{l_v^h | v \in \mathcal{V}(G)\}$, where l_v^h is the label of node v in the h-th iteration. The core of the proposed SaWL lies in the additional S operator, which encodes subgraph information with the results of each WL iteration. We describe the S operator in the following.

S operator. We employ an injective hash function that acts on labels of nodes within the subgraph to encode the subgraph information into a subgraph feature mapping:

$$\boldsymbol{\phi}^{(h)}\left(G_v^h\right) = \text{HASH}\left(\{\!\{l_v^h | v \in \mathcal{V}(G_v^h)\}\!\}\right), \tag{3}$$

where G_v^h is the h-hop rooted subgraph around node v. The hash function can be designed freely. Essentially, the S operator encodes the multiset of node labels within G_v^h (obtained by running h iterations of WL on the full graph) into a subgraph representation.

Graph Feature Mapping Module. With the subgraph feature mapping, an injective readout function is adopted to obtain the whole graph feature mapping in the h-th iteration, i.e.,

$$\boldsymbol{\psi}^{(h)}(G) = \text{READOUT}\left(\boldsymbol{\phi}^{(h)}(G_v^h) | v \in \mathcal{V}(G)\right). \tag{4}$$

The readout function can be chosen freely. To retain the structural information at all iterations, the final graph feature mapping is obtained by concatenation, i.e., $\boldsymbol{\psi}(G) = \text{CONCAT}\left(\boldsymbol{\psi}^{(0)}(G), \boldsymbol{\psi}^{(1)}(G), ..., \boldsymbol{\psi}^{(H)}(G)\right)$, where H is the maximum iteration number.

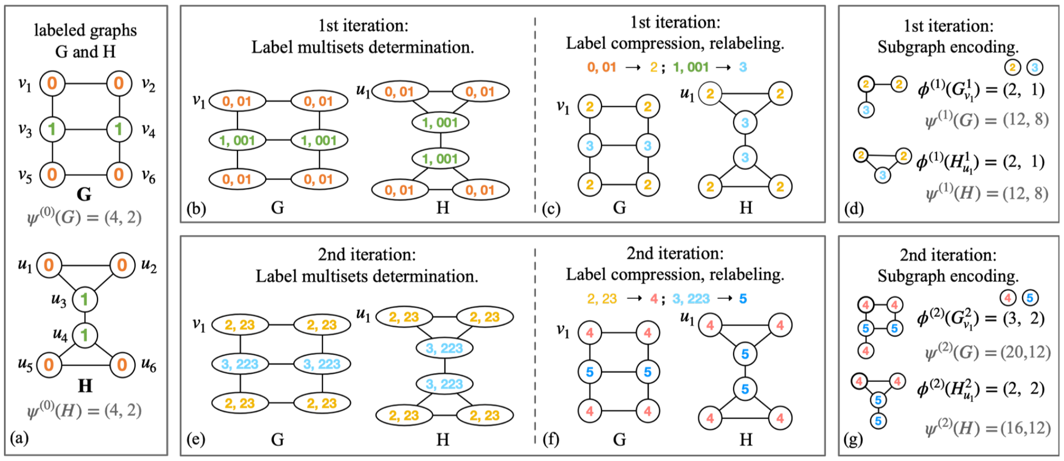

labeled graphs G and H
G
$\psi^{(0)}(G) = (4, 2)$
H
$\psi^{(0)}(H) = (4, 2)$
1st iteration: Label multisets determination.
1st iteration: Label compression, relabeling.
0, 01 → 2 ; 1, 001 → 3
1st iteration: Subgraph encoding.
$\phi^{(1)}(G^1_{v_1}) = (2, 1)$
$\psi^{(1)}(G) = (12, 8)$
$\phi^{(1)}(H^1_{u_1}) = (2, 1)$
$\psi^{(1)}(H) = (12, 8)$
2nd iteration: Label multisets determination.
2nd iteration: Label compression, relabeling.
2, 23 → 4 ; 3, 223 → 5
2nd iteration: Subgraph encoding.
$\phi^{(2)}(G^2_{v_1}) = (3, 2)$
$\psi^{(2)}(G) = (20,12)$
$\phi^{(2)}(H^2_{u_1}) = (2, 2)$
$\psi^{(2)}(H) = (16,12)$

Figure 2: Illustration of the fast SaWL. Colored numbers denote node labels. In (b), (c), (e) and (f), neighbor nodes are aggregated as multiset and compressed to updated labels (the same as 1-WL). In (d) and (g), the S operator encodes each rooted subgraph into a feature mapping. After the 2nd iteration, the feature mapping of $G^2_{v_1}$ is no longer equal to that of $H^2_{u_1}$, so that graph G and H can be discriminated by SaWL (but not by 1-WL).

Discussion. Compared to plain WL, which directly uses node labels at h-th iteration to obtain the graph representation, SaWL additionally uses the multiset of labels of node v's neighbors within h-hop to enhance WL with subgraph information. To understand SaWL's benefits over plain WL, from one point of view, SaWL encodes the node-subgraph-graph hierarchy instead of the node-graph hierarchy of WL, which better captures the hierarchical structural characteristics of the graph. From another point of view, plain WL encodes a node by its rooted subtree pattern, which can have repeated nodes. The repetitions of the same node are regarded as distinct nodes, and the actual number of nodes in the subtree pattern might be corrupted. The hash function in the S operator further characterizes the information of the actual number of nodes in the subgraph (which also equals the actual number of nodes in the subtree pattern, because the subgraph G^h_v does not have repeated nodes).

3.2 A Fast Implementation of SaWL

To illustrate the idea of SaWL, we provide a particular implementation here named **fast SaWL**. For the S operator, we design HASH function as a counting mapping that counts the occurrence of different node labels in the subgraph. Then, we adopt sum pooling as the READOUT function to obtain the whole graph feature mapping.

Definition 1 (Counting mapping). *Let $\mathcal{L}_h \subseteq \mathcal{L}$ denote the set of node labels that occur at least once in the h-th iteration. $\mathcal{L}_h = (\ell^h_1, \ell^h_2, ..., \ell^h_{|\mathcal{L}_h|})$ and we assume that $\mathcal{L}_h$ is ordered. Assume $G^h_v \in \mathcal{G}$, where $\mathcal{G}$ is the complete graph space. For each iteration h, we define a counting mapping $c_h : \mathcal{G} \times \mathcal{L}_h \to \mathbb{N}$, where $c_h(G^h_v, \ell^h_i)$ is the number of the occurrences of the i-th node label ℓ^h_i in subgraph G^h_v at the h-th iteration.*

With counting mapping, the feature mapping of the subgraph G^h_v can be obtained by $\phi^{(h)}(G^h_v) = \Big(c_h(G^h_v, \ell^h_1), ..., c_h(G^h_v, \ell^h_{|\mathcal{L}_h|})\Big)$, where the value of the i-th position of the vector represents the occurrence number of label ℓ^h_i in the h-th iteration. Essentially, the S operator encodes subgraph by mapping the multiset of node labels within the subgraph to a vector, recording the occurrence number of each label. Then, the whole graph feature mapping is obtained by applying sum pooling to the subgraph feature mappings. Although the sum pooling is not an injective readout function, as we will show, it allows fast computation (acceleration) via an implementation trick.

Illustration. We illustrate the fast SaWL in Figure 2. Given two graphs G and H where colored numbers indicate node labels. The WL encoder of fast SaWL updates node labels in (b), (c), (e) and (f).

S operator encodes rooted subgraphs, and we take two rooted subgraphs as examples in Figure 2(g). The feature mapping of the subgraph $G^2_{v_1}$ in the 2nd iteration is $\phi^{(2)}(G^2_{v_1}) = (3, 2)$, which means the label 4 occurs three times and label 5 occurs twice in the subgraph. Then the subgraphs are pooled to obtain the graph feature mapping in the 2nd iteration, e.g., for graph G, $\psi^{(2)}(G) = \phi^{(2)}\left(G^2_{v_1}\right) + \phi^{(2)}\left(G^2_{v_2}\right) + ... + \phi^{(2)}\left(G^2_{v_6}\right) = (20, 12)$. And for graph H, $\psi^{(2)}(H) = (16, 12)$. Finally, the whole graph feature mappings are $\psi(G) = (4, 2, 12, 8, 20, 12)$, and $\psi(H) = (4, 2, 12, 8, 16, 12)$. The graph G and H cannot be discriminated by 1-WL, but they can be discriminated by our fast SaWL.

Acceleration. In fast SaWL, the calculation of the S operator can be executed simultaneously with the WL encoder, which reduces the computational time. Since the subgraph feature mappings are summed as the whole graph feature mapping, the frequency of one node contributing to the whole graph feature mapping is equal to the number of occurrences of this node in all h-hop rooted subgraphs. We use graph H (adapted from Figure 2(f)) as an example. In Figure 3(a), each tuple (a, b) represents the feature mapping of the node's rooted subgraph. The whole graph feature mapping can be computed by summing all subgraphs' feature mappings: $\psi^{(2)}(H) = (2, 2) + ... + (4, 2) + ... + (2, 2) = (16, 12)$. However, we can actually compute the whole graph feature mapping from a global perspective. E.g., node u_1 contributes to the 2-hop rooted subgraphs of nodes u_1, u_2, u_3, u_4. And the number of u_1's contributions to the whole graph feature mapping is exactly the size of node u_1's 2-hop rooted subgraph, i.e., $|\mathcal{V}(H^{(2)}_{u_1})| = 4$. Similarly, we mark each node's contribution number beside it in Figure 3(b). The whole graph feature mapping can be alternatively computed by summing the contribution numbers for each label dimension, i.e., $\psi^{(2)}(H) = (4+4+4+4, 6+6) = (16, 12)$. The sizes of rooted subgraphs can be computed together in the multiset determination of WL run on the original graph by propagating node label and ID simultaneously. We present the steps of the accelerating version of the fast SaWL for graph classification in Algorithm 1 of the Appendix. We additionally detail how to use the version for graph isomorphism testing in Appendix A.7.

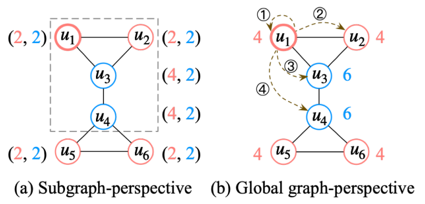


Figure 3: u_1 contributes to the feature mappings of rooted subgraphs of u_1, u_2, u_3, u_4. The contribution number equals the size of rooted subgraph $H^{(2)}_{u_1}$.

3.3 The Expressive Power of SaWL

We first analyze the expressive power of SaWL by comparing it with 1-WL. Once the graphs can be discriminated by 1-WL, they can be discriminated by SaWL as well.

Proposition 1. *Given two graphs G and H, if they can be distinguished by 1-WL, i.e., $\phi^{(h)}(G) \neq \phi^{(h)}(H)$, then they must be distinguished by the SaWL, i.e., $\psi^{(h)}(G) \neq \psi^{(h)}(H)$.*

See Appendix A.2 for proof. If the graph pair can be discriminated by 1-WL, the counting mappings of the whole graphs are different. There must exist subgraphs with different counting mappings in the graph pair. Therefore, the final feature mappings of the two graphs obtained by SaWL are different.

Proposition 2. *We define the number of h-shortest neighbors of each node as s^h_v, which is the number of nodes with the exact shortest distance h from the center node v. For graphs G and H, if $\{\!\{s^h_v | v \in \mathcal{V}(G)\}\!\} \neq \{\!\{s^h_u | u \in \mathcal{V}(H)\}\!\}$, then the two graphs can be distinguished by the h-layer SaWL.*

From a global graph perspective, if the multisets of numbers of the h-shortest neighbor of nodes in graph G and H are different, there exist at least two subgraphs in the graphs with different encodings. Then from a subgraph perspective, the multiset of subgraph encodings of the two graphs are different and they can be discriminated by SaWL. We provide a detailed explanation in the Appendix A.3.

Theorem 1. *The expressive power of SaWL is higher than that of 1-WL in distinguishing graphs.*

As proved in Proposition 1, once the graphs can be discriminated by 1-WL, they must be discriminated by SaWL. There are also many graphs that can be discriminated by SaWL, but not by 1-WL, e.g., graphs G and H in Figures 2, we provide more examples in Appendix A.4. To sum up, the expressive power of SaWL is strictly higher than that of 1-WL. According to recent research on subgraph

GNNs [26], SaWL's k-hop subgraph selection and encoding scheme can be implemented by 3-order Invariant Graph Networks (3-IGNs), whose expressive power is bounded by 3-WL [27]. Thus, SaWL's expressive power is also bounded by 3-WL.

3.4 Complexity

We analyze the computational complexity of the fast SaWL and the corresponding accelerating version respectively. Given the graph G with node number N, average node degree D and edge number M, where $M = ND$. We assume the average node number of the subgraphs is n. For **the fast SaWL**, the multiset determination, the label compression and relabeling in the WL encoder take a total runtime of $O(ND)$ [18]. In the S operator, the feature mapping computing of one subgraph with n nodes takes $O(n)$, and that of the N subgraphs takes $O(Nn)$. To sum up, the time complexity is $O(ND) + O(Nn)$. For **the accelerating version**, the S operator can be executed simultaneously with the multiset determination of the WL encoder. Specifically, determining the label multisets and identity sets for all nodes takes $O(ND)$ operations which can be accomplished simultaneously. The runtime of the identity set can be achieved by using a hash table. Therefore, the total time complexity of the accelerating version is $O(ND)$, which equals that of 1-WL algorithm [18].

4 Subgraph-aware Graph Neural Network

In order to generalize SaWL to scenarios with continuous features, we propose a neural version of SaWL, namely Subgraph-aware GNN (SaGNN). Each component in the SaWL is replaced with a neural network in SaGNN.

Model. The neural version SaGNN includes two components: the GNN encoder and the S operator. Any standard neural version of the 1-WL algorithm can be utilized as the GNN encoder. Given input graphs, ***GNN encoder*** updates nodes with its previous state and representations of neighbor nodes (Eq. 1). Specifically, we adopt GIN with $\epsilon = 0$ to obtain the node representations in the k-th layer, i.e., $\boldsymbol{h}_v^{(k)} = \mathrm{MLP}^{(k)}\left(\boldsymbol{h}_v^{(k-1)} + \sum_{u\in\mathcal{N}(v)} \boldsymbol{h}_u^{(k-1)}\right)$, where $\mathcal{N}(v)$ denotes the neighbor nodes of node v, and $\boldsymbol{h}_v^{(k)} \in \mathbb{R}^{N\times D_1}$, D_1 is the feature dimension. In each layer, node representations are updated by the GNN encoder applied to the full graph.

With the updated node representations, S ***operator*** in SaGNN are designed to further encode k-hop subgraphs around each node, which provides extra expressive power beyond plain GNN. An injective function is utilized for encoding subgraph information by aggregating nodes within the subgraph (Eq. 3). In this paper, we adopt MLP with SUM as the hash function, as given the input from the countable space, the combination achieves injective [13]. The representation of the subgraph around node v is obtained by $\boldsymbol{h}_{s,v}^{(k)} = \mathrm{MLP}\left(\sum_{q\in\mathcal{V}(G_v^k)} \boldsymbol{h}_q^{(k)}\right)$.

Then, graph representations in the k-th layer are calculated with a readout (pooling) function (Eq. 4). In SaGNN, we adopt sum pooling as the readout function, i.e., $\boldsymbol{H}^{(k)}(G) = \mathrm{SUM}\left(\boldsymbol{h}_{s,v}^{(k)} | v \in V(G)\right)$. Then the representations of graph G in all layers are concatenated as the final graph representation,i.e., $\boldsymbol{H}(G) = \mathrm{CONCAT}\left(\boldsymbol{H}^{(1)}(G), \boldsymbol{H}^{(2)}(G), ..., \boldsymbol{H}^{(k)}(G)\right)$, and $\boldsymbol{H}(G) \in \mathbb{R}^{D_1 * k}$.

Discussion. Since the SaGNN is the neural version of SaWL, and the SaWL have been shown to be more expressive than 1-WL, the expressive power of SaGNN is higher than that of 1-WL. The computational complexity of SaGNN is also the same as the fast SaWL (section 3.4), which is $O(ND + Nn)$. Besides, both the proposed SaGNN and the existing methods of WL-on-subgraph paradigm [15–17] intend to uplift GNNs by encoding subgraphs. However, methods of WL-on-subgraph paradigm bring high computational cost by extracting rooted subgraphs and applying multiple GNNs. Instead, SaGNN encodes rooted subgraphs with the nodes updated in full graphs, which keeps the computational cost low. We present a detailed comparison in Appendix A.5.

5 Experiments

In this section, we first evaluate the effectiveness of the proposed fast SaWL and SaGNN on graph classification tasks. Then we conduct experiments to verify the expressiveness of the methods.

Table 1: 10-Fold Cross Validation average test accuracy (%) on TU datasets.

Methods	MUTAG	PTC_MR	Mutagenicity	NCI1	NCI109
SP kernel	87.28 ± 0.55	58.24 ± 2.44	71.63 ± 2.19	73.47 ± 0.21	73.07 ± 0.11
WL kernel	82.05 ± 0.36	57.97 ± 0.49	-	82.19 ± 0.18	82.46 ± 0.24
DGK	87.44 ± 2.72	60.08 ± 2.55	-	73.55 ± 0.51	73.26 ± 0.26
GCN	78.69 ± 6.56	66.73 ± 4.65	80.84 ± 1.35	78.39 ± 1.79	77.57 ± 1.79
GIN	81.51 ± 8.47	54.09 ± 6.20	77.70 ± 2.50	80.0 ± 1.40	70.20 ± 3.21
Diffpool	80.00 ± 6.98	57.14 ± 7.11	80.55 ± 1.98	78.88 ± 3.05	76.76 ± 2.38
SortPool	85.83 ± 1.66	58.59 ± 2.47	80.41 ± 1.02	74.44 ± 0.47	-
1-2-3-GNN	86.10 ± 0.0	60.9 ± 0.0	-	76.2 ± 0.0	-
3-hop GNN	87.56 ± 0.72	-	-	80.61 ± 0.34	-
Nested GIN	87.90 ± 8.20	54.1 ± 7.70	82.40 ± 2.00	78.60 ± 2.30	77.20 ± 2.90
GraphSNN	$\mathbf{91.57 \pm 2.80}$	66.70 ± 3.70	-	81.60 ± 2.80	-
SaWL Kernel	87.31 ± 7.04	63.40 ± 7.30	81.05 ± 1.96	83.80 ± 1.80	82.48 ± 2.54
fast SaWL	$\mathbf{90.00 \pm 3.89}$	$\mathbf{70.33 \pm 5.32}$	$\mathbf{84.32 \pm 1.48}$	$\mathbf{84.45 \pm 0.66}$	$\mathbf{85.37 \pm 0.81}$
SaGNN	88.81 ± 5.21	$\mathbf{71.78 \pm 4.43}$	$\mathbf{84.13 \pm 1.31}$	$\mathbf{83.78 \pm 1.03}$	$\mathbf{83.35 \pm 0.56}$

Besides, We compare the computation time of our methods with 1-WL and methods of WL-on-subgraph paradigm to verify the efficiency of our methods.

5.1 Datasets

In the tasks of graph classification, we evaluate fast SaWL and SaGNN with seven datasets, including TU datasets [28], and Open Graph Benchmark (OGB) dataset [29]. Graphs in these datasets represent chemical molecules, nodes represent atoms, and edges represent chemical bonds. TU datasets include MUTAG [30], PTC_MR [31], Mutagenicity [32], NCI1 [33] and NCI109 [33]. The task is binary classification, and the metric is classification accuracy. Task on OGB dataset ogbg-molhiv is molecular prediction. It is a binary classification, and the metric is ROC-AUC. We evaluate the **expressiveness** of our methods on the EXP [34], CSL [35] and SR25 datasets [36], which are three synthetic datasets containing 1-WL undistinguishable regular graphs. We provide more description and statistics of the datasets in Appendix A.6.

5.2 Baselines

In the experiment of the graph classification task on TU, we adopt three graph kernel methods, some GNNs methods based on the 1-WL, and some methods with higher expressive power than 1-WL as baselines. Graph kernel methods include shortest path kernel [37], WL subtree kernel [18] and deep graph kernel [38]. GNNs methods based on the 1-WL include GCN [22], GIN [13], Diffpool [25], and Sortpool [39]. For GCN, graph representations are obtained by the learned nodes representations and sum pooling. Higher expressive methods include 1-2-3 GNN [14], 3-hop GNN [17] Nested GNN [15] and GraphSNN [40]. On OGB dataset, we compare with the traditional message passing GNNs, and the higher expressive methods Deep LRP-1-3 [41], Nested GNN [15] and GIN-AK$^+$ [16]. Results of baselines are obtained either from raw paper or source code with published experimental settings ("-" indicates that results are not available). For GCN and GIN, we search the model layer in $\{2, 3, 4, 5\}$, and hidden dimensions in $\{32, 64, 128\}$. For Nested GNN, we choose the best-performing Nested GIN as the baseline according to the results in the original paper. And the results on the datasets Mutagenicity, NCI and NCI109, we search the subgraph height in $\{2, 3, 4, 5\}$ with 4 model layers.

5.3 Experimental Setup

In graph classification tasks, we adopt multilayer perceptrons (MLPs) with softmax as the classifier to predict the class label of the graph. On the TU datasets, we perform 10-fold cross-validation where 9 folds for training, 1 fold for testing. 10% split of the training set is used for model selection [42]. We report the average and standard deviation (in percentage) of test accuracy across the 10 folds. We train the models with batch size 32. On the OGB dataset, the experiments are conducted 10 times, and the average scores of ROC-AUC are reported. We train the models with batch size 256. For all datasets, we implement experiments with PyTorch and employ Adam optimizer with the learning rate of 0.001 to optimize the model. We search the iteration times of our methods in $\{2, 3, 4\}$. In the training process, we adopt the early stopping strategy with patience 30, and we report the test results

Table 2: Performance Evaluation on OGB dataset.

Methods	ogbg-molhiv (AUC)	
	Validation	Test
GCN [22]	82.04 ± 1.41	76.06 ± 0.97
GIN [13]	82.32 ± 0.90	75.58 ± 1.40
Deep LRP-1-3 [41]	81.31 ± 0.88	76.87 ± 1.80
Nested GNN [15]	83.17 ± 1.99	78.34 ± 1.86
GIN-AK$^+$ [16]	-	$\mathbf{79.61 \pm 1.19}$
fast SaWL	79.13 ± 0.69	78.29 ± 0.48
SaGNN	81.06 ± 1.14	$\mathbf{78.86 \pm 0.73}$

Table 3: Evaluation of Expressiveness.

Model	EXP (ACC)	CSL (ACC)	SR25 (ACC)
GCN [22]	50.0 ± 0.00	10.0 ± 0.00	6.67
GIN [13]	50.0 ± 0.00	10.0 ± 0.00	6.67
GCN-RNI [34]	98.0 ± 1.85	16.0 ± 0.00	6.67
PPGN [43]	$\mathbf{100.0 \pm 0.00}$	-	6.67
3-GCN [14]	99.7 ± 0.004	95.70 ± 14.85	6.67
Nested GNN [15]	99.9 ± 0.26	-	6.67
GIN-AK$^+$ [16]	$\mathbf{100.0 \pm 0.00}$	-	6.67
fast SaWL	99.50 ± 0.70	80.67 ± 8.04	6.67
SaGNN	99.67 ± 0.70	84.67 ± 10.45	6.67

at the epoch of best validation. The experimental setups of the expressive power evaluation are kept the same with [34–36]. We use the Nvidia V100 GPUs to run the experiments.

5.4 Effectiveness Evaluation

Performance on Graph Classification Task. Results of the graph classification on TU and OGB datasets are shown in Tables 1, 2. We take our SaWL with linear SVM as a graph kernel method and report the results on TU datasets. Compared with graph kernel methods and traditional GNNs based on 1-WL, our SaWL kernel gain strong improvements. Especially, SaWL kernel achieves better performance than WL subtree kernel, which proves the effectiveness of the S operator experimentally. It verifies that the augmented subgraph information on the basis of the subtree pattern enhances the expressive power on the graph classification task. For methods with higher expressive power than traditional message passing GNNs, i.e., 1-2-3-GNN, 3-hop GNN, Nested GNN and GraphSNN, our fast SaWL and SaGNN still outperform the methods on most TU datasets. Especially, our fast SaWL gains such progress with low computational cost. The proposed methods achieve comparable results to other highly expressive methods on the larger-scale OGB dataset. The results show that our methods achieve higher or comparable performance to methods with high computational cost. We adopt GIN as the GNN encoder in SaGNN. The improvements compared to GIN verify the effectiveness of the S operator, which provides additional subgraph information in graph classification. The neural version SaGNN achieves slightly lower performance than fast SaWL on some small-scale datasets, which may be because the neural model is not sufficiently trained with insufficient training data. On the larger-scale OGB dataset, the neural version SaGNN achieves better results than fast SaWL with sufficient training. In summary, fast SaWL and SaGNN achieve improvement compared with competitive baselines on the graph classification task.

Expressive Power Evaluation. We first evaluate the expressiveness on the EXP, CSL and SR25 datasets, and then show cases of graph isomorphism testing in Appendix A.7. Results of empirical evaluation are shown in Table 3, and some results of baselines are from [34, 44]. Each pair of graphs in the three datasets is non-isomorphic and 1-WL indistinguishable, and the results of GCN and GIN verify this. We adopt five methods with highly expressive power as baselines [14–16, 34, 43]. On EXP, our fast SaWL and SaGNN consistently achieve very high accuracy, which can distinguish nearly all graph pairs. The results are comparable with the k-GNNs [14, 43] and Nested GNN [15], which are more computationally complex. On CSL, our methods significantly outperform 1-WL based GNNs and are lower than 3-GCN. The results verify the high expressive power of fast SaWL and SaGNN, which have been stated theoretically. Strongly regular graphs in SR25 are 3-WL equivalent [45] and cannot be distinguished by the methods in Table 3.

5.5 Efficiency Evaluation

We compare the running time of the proposed methods with baselines to verify the high efficiency in practice. Our **fast SaWL** has higher discriminating power than 1-WL, while the accelerating version of the fast SaWL has the same time complexity as 1-WL, which have been demonstrated in section 3.4.

Table 4: Runtime Comparison of fast SaWL with 1-WL (second).

Methods	Mutagenicity	NCI1	NCI109	ogbg-molhiv
1-WL	4.90 ± 0.23	4.69 ± 0.16	4.73 ± 0.20	112.25 ± 0.68
fast SaWL	4.99 ± 0.22	4.81 ± 0.20	4.96 ± 0.20	115.11 ± 0.71

We record the running time of fast SaWL and 1-WL in obtaining feature mappings of all graphs in four datasets respectively. The average running time with ten runs are shown in Tabel 4. The running time of fast SaWL is similar to that of 1-WL. The time difference is less than 0.5 seconds on the TU dataset and less than 3 seconds on the ogbg-molhiv, which contains 41127 graphs. We further conduct the t-test as a significance test. The p-value is 0.8413, and $0.8413 > 0.05$, which demonstrates that there is no significant difference in the running time of fast SaWL and 1-WL on graph feature mapping calculation. For **SaGNN**, we compare the running time with an example method of the WL-on-subgraph paradigm, i.e., Nested GNN (NGNN) [15] in Figure 4. On TU datasets, the running time of the Nested GNN is more than three times that of SaGNN. On the ogbg-molhiv dataset (abbreviated as ogb in Figure 4), we compare the epoch time and the whole training time. The running time of the Nested GNN is more than ten times that of SaGNN on both each epoch and the whole training process, e.g., the average training time of Nested GNN on an epoch is 134.91 ± 21.30 seconds, and that of SaGNN is 9.71 ± 0.49 seconds. The time comparison demonstrates that our SaGNN is significantly more efficient than methods of the WL-on-subgraph paradigm.

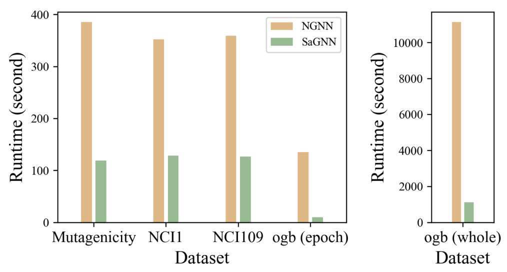


Figure 4: Training Time Comparison of SaGNN with Method of the WL-on-subgraph paradigm.

6 Related Works

The expressiveness of graph neural networks is a key research topic in graph machine learning. Many approaches with higher expressive power than 1-WL have been proposed, including high-dimension WL based [14, 43], feature augmentation based [34, 46], subgraph encoding based [15, 16, 47] and equivariant models [26, 27, 48]. We provide a breif review here. **(1)** It's natural to build GNNs based on a high-dimensional WL algorithm for high expressive power, e.g., PPNG [43] based on the high-order graph networks, k-GNNs [14] based on the set k-WL algorithm. However, the high dimension WL algorithms require enumeration of the node tuples, which limits the scalability and generalization with high computational cost. **(2)** Some researchers propose to improve the expressive power of GNN by adding additional features. They augment GNNs by concatenating pre-extracted sub-structural information or random features as additional node features [34, 41, 49]. E.g., Graph Structure Networks (GSN) [49] encodes structural information in the additional preprocessing stage by counting the appearance of certain substructures as the structural feature vector. Then the structural features are utilized in message passing. GCN-RNI [34] enhances GNNs with random node initialization. rGINs [46] concatenates random features with node features and then applies GINs on the combined features. However, such additional feature augmentation-based methods limit the generalization ability of the methods. **(3)** Many existing subgraph-based methods first extract subgraphs centered on each node of graphs, then apply GNNs on the extracted subgraphs [15, 16]. E.g., Nested GNN [15] implements base GNN on the extracted subgraphs then obtains the whole graph representations by a global pooling. These methods can be summarized as WL-on-subgraph paradigm (Figure 1 (b)), and the computational cost are much higher than 1-WL, which limits their application to the large scale graphs. We provide more related works in Appendix A.10.

7 Conclusion

The traditional message passing graph neural networks (GNNs) are at most as powerful as 1-WL algorithm. Since the representative power of the subgraph is higher than that of the subtree, methods of the WL-on-subgraph paradigm are proposed to improve GNNs, which brings expensive computational cost. As a contrast, we propose the subgraph-aware WL (SaWL) paradigm in this paper, which uplifts GNNs and keeps computation complexity low. Under the paradigm, we first implement an algorithm named fast SaWL, where the additional S operator encodes subgraph information on the basis of the WL on the full graph. We then present the neural version of the SaWL named SaGNN, which replace the components in SaWL with neural networks. SaWL and SaGNN are proved to be more expressive than 1-WL, and have achieved significant improvements in the experiments.

References

[1] Wenqi Fan, Yao Ma, Qing Li, Yuan He, Eric Zhao, Jiliang Tang, and Dawei Yin. Graph neural networks for social recommendation. In *The world wide web conference*, pages 417–426, 2019. 1

[2] T Gaudelet, B Day, AR Jamasb, J Soman, C Regep, G Liu, et al. Utilising graph machine learning within drug discovery and development (2020). *arXiv preprint arXiv:2012.05716*, 2020. 17

[3] Alexey Strokach, David Becerra, Carles Corbi-Verge, Albert Perez-Riba, and Philip M Kim. Fast and flexible protein design using deep graph neural networks. *Cell systems*, 11(4):402–411, 2020. 1

[4] Zhongkai Hao, Chengqiang Lu, Zhenya Huang, Hao Wang, Zheyuan Hu, Qi Liu, Enhong Chen, and Cheekong Lee. Asgn: An active semi-supervised graph neural network for molecular property prediction. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 731–752, 2020. 1

[5] Lesong Wei, Xiucai Ye, Yuyang Xue, Tetsuya Sakurai, and Leyi Wei. Atse: a peptide toxicity predictor by exploiting structural and evolutionary information based on graph neural network and attention mechanism. *Briefings in Bioinformatics*, 22(5):bbab041, 2021. 1

[6] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 32(1):4–24, 2020. 1

[7] Sergi Abadal, Akshay Jain, Robert Guirado, Jorge López-Alonso, and Eduard Alarcón. Computing graph neural networks: A survey from algorithms to accelerators. *ACM Computing Surveys (CSUR)*, 54(9):1–38, 2021.

[8] Yu Zhou, Haixia Zheng, Xin Huang, Shufeng Hao, Dengao Li, and Jumin Zhao. Graph neural networks: Taxonomy, advances, and trends. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 13(1):1–54, 2022. 1

[9] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International Conference on Machine Learning (ICML)*, pages 1263–1272. PMLR, 2017. 1

[10] B. Y. Weisfeiler and A. A. Leman. A reduction of a graph to a canonical form and an algebra arising during this reduction (in russian). 1968. 1, 2

[11] Brendan L Douglas. The weisfeiler-lehman method and graph isomorphism testing. *arXiv preprint arXiv:1101.5211*, 2011. 1, 16

[12] Ryoma Sato. A survey on the expressive power of graph neural networks. *arXiv preprint arXiv:2003.04078*, 2020. 1, 3, 14

[13] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *Proceedings of the Information Conference of Learning Representation (ICLR)*, 2018. 3, 6, 7, 8, 18

[14] Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *AAAI conference on artificial intelligence*, 2019. 1, 3, 7, 8, 9, 18

[15] Muhan Zhang and Pan Li. Nested graph neural networks. *Advances in Neural Information Processing Systems*, 34, 2021. 1, 6, 7, 8, 9, 15, 17

[16] Lingxiao Zhao, Wei Jin, Leman Akoglu, and Neil Shah. From stars to subgraphs: Uplifting any gnn with local structure awareness. In *Proceedings of the Information Conference of Learning Representation (ICLR)*, 2021. 7, 8, 9, 15, 17

[17] Giannis Nikolentzos, George Dasoulas, and Michalis Vazirgiannis. k-hop graph neural networks. *Neural Networks*, 130:195–205, 2020. 1, 6, 7, 15, 18

[18] Nino Shervashidze, Pascal Schweitzer, Erik Jan Van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(9), 2011. 2, 3, 6, 7, 16, 18

[19] Nils M Kriege, Fredrik D Johansson, and Christopher Morris. A survey on graph kernels. *Applied Network Science*, 5(1):1–42, 2020. 2

[20] László Babai and Ludik Kucera. Canonical labelling of graphs in linear average time. In *20th Annual Symposium on Foundations of Computer Science (sfcs 1979)*, pages 39–46. IEEE, 1979. 2

[21] Martin Grohe. *Descriptive complexity, canonisation, and definable graph structure theory*, volume 47. Cambridge University Press, 2017. 3

[22] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *Proceedings of the Information Conference of Learning Representation (ICLR)*, 2017. 3, 7, 8, 18

[23] Junhyun Lee, Inyeop Lee, and Jaewoo Kang. Self-attention graph pooling. In *International Conference on Machine Learning (ICML)*, pages 3734–3743. PMLR, 2019. 3, 18

[24] Yao Ma, Suhang Wang, Charu C Aggarwal, and Jiliang Tang. Graph convolutional networks with eigenpooling. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 723–731, 2019.

[25] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. In *Advances in Neural Information Processing Systems (NIPS)*, pages 4800–4810, 2018. 3, 7, 18

[26] Fabrizio Frasca, Beatrice Bevilacqua, Michael M Bronstein, and Haggai Maron. Understanding and extending subgraph gnns by rethinking their symmetries. *arXiv preprint arXiv:2206.11140*, 2022. 6, 9

[27] Waïss Azizian and Marc Lelarge. Expressive power of invariant and equivariant graph neural networks. In *ICLR 2021*, 2021. 6, 9

[28] Christopher Morris, Nils M Kriege, Franka Bause, Kristian Kersting, Petra Mutzel, and Marion Neumann. Tudataset: A collection of benchmark datasets for learning with graphs. *arXiv preprint arXiv:2007.08663*, 2020. 7

[29] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *arXiv preprint arXiv:2005.00687*, 2020. 7

[30] Asim Kumar Debnath, Rosa L Lopez de Compadre, Gargi Debnath, Alan J Shusterman, and Corwin Hansch. Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity. *Journal of medicinal chemistry*, 34(2):786–797, 1991. 7

[31] Hannu Toivonen, Ashwin Srinivasan, Ross D King, Stefan Kramer, and Christoph Helma. Statistical evaluation of the predictive toxicology challenge 2000–2001. *Bioinformatics*, 19(10): 1183–1193, 2003. 7

[32] Jeroen Kazius, Ross McGuire, and Roberta Bursi. Derivation and validation of toxicophores for mutagenicity prediction. *Journal of medicinal chemistry*, 48(1):312–320, 2005. 7, 17

[33] Nikil Wale, Ian A Watson, and George Karypis. Comparison of descriptor spaces for chemical compound retrieval and classification. *Knowledge and Information Systems*, 14(3):347–375, 2008. 7

[34] Ralph Abboud, İsmail İlkan Ceylan, Martin Grohe, and Thomas Lukasiewicz. The surprising power of graph neural networks with random node initialization. In *Proceedings of the Thirtieth International Joint Conference on Artifical Intelligence (IJCAI)*, 2021. 7, 8, 9, 16

[35] Ryan Murphy, Balasubramaniam Srinivasan, Vinayak Rao, and Bruno Ribeiro. Relational pooling for graph representations. In *International Conference on Machine Learning*, pages 4663–4673. PMLR, 2019. 7, 16, 17

[36] Muhammet Balcilar, Pierre Héroux, Benoit Gauzere, Pascal Vasseur, Sébastien Adam, and Paul Honeine. Breaking the limits of message passing graph neural networks. In *International Conference on Machine Learning*, pages 599–608. PMLR, 2021. 7, 8, 16

[37] Karsten M Borgwardt and Hans-Peter Kriegel. Shortest-path kernels on graphs. In *Fifth IEEE International Conference on Data Mining (ICDM)*, pages 8–pp. IEEE, 2005. 7, 18

[38] Pinar Yanardag and SVN Vishwanathan. Deep graph kernels. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1365–1374, 2015. 7

[39] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. An end-to-end deep learning architecture for graph classification. In *Thirty-second AAAI conference on artificial intelligence*, 2018. 7, 18

[40] Asiri Wijesinghe and Qing Wang. A new perspective on" how graph neural networks go beyond weisfeiler-lehman?". In *International Conference on Learning Representations*, 2021. 7

[41] Zhengdao Chen, Lei Chen, Soledad Villar, and Joan Bruna. Can graph neural networks count substructures? *Advances in neural information processing systems*, 33:10383–10395, 2020. 7, 8, 9, 18

[42] Federico Errica, Marco Podda, Davide Bacciu, and Alessio Micheli. A fair comparison of graph neural networks for graph classification. *Proceedings of the Information Conference of Learning Representation (ICLR)*, 2019. 7

[43] Haggai Maron, Heli Ben-Hamu, Hadar Serviansky, and Yaron Lipman. Provably powerful graph networks. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, pages 2156–2167, 2019. 8, 9

[44] Vijay Prakash Dwivedi, Chaitanya K Joshi, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Benchmarking graph neural networks. *arXiv preprint arXiv:2003.00982*, 2020. 8

[45] Vikraman Arvind, Frank Fuhlbrück, Johannes Köbler, and Oleg Verbitsky. On weisfeiler-leman invariance: Subgraph counts and related graph properties. *Journal of Computer and System Sciences*, 113:42–59, 2020. 8

[46] Ryoma Sato, Makoto Yamada, and Hisashi Kashima. Random features strengthen graph neural networks. In *Proceedings of the 2021 SIAM International Conference on Data Mining (SDM)*, pages 333–341. SIAM, 2021. 9

[47] Leonardo Cotta, Christopher Morris, and Bruno Ribeiro. Reconstruction for powerful graph representations. *Advances in Neural Information Processing Systems*, 34:1713–1726, 2021. 9

[48] Beatrice Bevilacqua, Fabrizio Frasca, Derek Lim, Balasubramaniam Srinivasan, Chen Cai, Gopinath Balamurugan, Michael M Bronstein, and Haggai Maron. Equivariant subgraph aggregation networks. In *International Conference on Learning Representations*, 2021. 9, 18

[49] Giorgos Bouritsas, Fabrizio Frasca, Stefanos P Zafeiriou, and Michael Bronstein. Improving graph neural network expressivity via subgraph isomorphism counting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022. 9, 18

[50] Oliver Wieder, Stefan Kohlbacher, Mélaine Kuenemann, Arthur Garon, Pierre Ducrot, Thomas Seidel, and Thierry Langer. A compact review of molecular property prediction with graph neural networks. *Drug Discovery Today: Technologies*, 37:1–12, 2020. 17

[51] S Vichy N Vishwanathan, Nicol N Schraudolph, Risi Kondor, and Karsten M Borgwardt. Graph kernels. *Journal of Machine Learning Research*, 11:1201–1242, 2010. 18

[52] Nino Shervashidze, SVN Vishwanathan, Tobias Petri, Kurt Mehlhorn, and Karsten Borgwardt. Efficient graphlet kernels for large graph comparison. In *Artificial Intelligence and Statistics*, pages 488–495, 2009. 18

[53] Lanning Wei, Huan Zhao, Quanming Yao, and Zhiqiang He. Pooling architecture search for graph classification. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pages 2091–2100, 2021. 18

[54] Haoteng Tang, Guixiang Ma, Lifang He, Heng Huang, and Liang Zhan. Commpool: An interpretable graph pooling framework for hierarchical graph representation learning. *Neural Networks*, 143:669–677, 2021. 18

[55] Jiarui Feng, Yixin Chen, Fuhai Li, Anindya Sarkar, and Muhan Zhang. How powerful are k-hop message passing graph neural networks. *Advances in Neural Information Processing Systems*, 2022. 18

A Appendix

A.1 Acceleration of the fast SaWL

The S operator in the fast SaWL (Section 3.2) can be calculated simultaneously with the WL encoder, which leads to the accelerating version. The idea of the acceleration is illustrated in Figure 3, each node contributes to the feature mappings of m rooted subgraphs, where m equals the size of rooted subgraph centered in the node. The sizes of rooted subgraphs can be computed simultaneously with the multiset determination of the WL encoder. We then present the steps of the accelerating version.

The accelerating version proceeds in iterations. Each iteration consists of **five steps** (Algorithm 1), which are multisets determination, multisets sorting, label compression, relabeling and feature mapping obtaining. Specifically, given two graphs G and G', for node v, the label is denoted as l_v^h and the identity is denoted as id_v. In **step 1**, we aggregate the labels and identity sets of neighbor nodes respectively. Node labels of neighbor nodes are aggregated as a multiset M_v^h. For $h = 0$, $M_v^0 = l_v^0$, and for $h > 0$, $M_v^h = \{\!\{l_u^{h-1} | u \in \mathcal{N}(v)\}\!\}$, where $\mathcal{N}(v)$ denotes the neighbor nodes of v and $\{\!\{\}\!\}$ denotes a multiset. Identity sets of neighbor nodes are aggregated and combined with the identity of the center node which forms a new set t_v^h. For $h = 0$, $t_v^0 = \{id_v\}$, and for $h > 0$, $t_v^h = \{id_v, id_w | w \in t_u^{h-1}, u \in \mathcal{N}(v)\}$. In **step 2**, each label multiset M_v^h is sorted and converted to a string $\mathcal{S}_v^h$ with the prefix l_v^{h-1}, which prepares for the label compression. In **step 3**, each string is compressed to a new label with a hash function $\boldsymbol{g} : \sum * \rightarrow \sum$ and $\boldsymbol{g}$ should be an injective function. The mapping alphabet is shared across graphs, which guarantees a common feature space. In **step 4**, we relabel each node in graph G and G' as $l_v^h := \boldsymbol{g}(\mathcal{S}_v^h)$.

We assume the minimum label in h-th iteration is l_m. Then, from a global-graph perspective, the value of the i-th position (i starts from 0) of the final graph feature mapping in layer h is:

$$\psi_i^{(h)}(G) = \sum_{l_v^h = l_m + i, v \in V} \left| t_v^h \right|, \tag{5}$$

which means the summation of the occurrences of label $l_m + i$ in all h-hop subgraphs. The final graph feature mappings obtained by the fast SaWL and the accelerating version are equivalent. In the accelerating version, the feature mappings of subgraphs do not require to be calculated separately, which reduces the computational cost and speeds up the computation.

Algorithm 1 Accelerating version of fast SaWL for Graph Classification

```
Input: Node Labels (features) $\mathbf{X}$; Adjacency Matrix $\mathbf{A}$
  for $h = 1$ to $H$ do
    1. Label multisets and identity sets determination
    • Aggeregate labels of neighbor nodes centered in each node $v$ in graph $G$ as multiset $M_v^h$.
      For $h = 0$, $M_v^0 = l_v^0$, and for $h > 0$, $M_v^h = \{\!\{l_u^{h-1} | u \in \mathcal{N}(v)\}\!\}$.
    • Aggregate identity sets of neighbor nodes centered in each node $v$ in graph $G$. Identity of
      node $v$ and elements in identity sets of neighbor nodes compose the new identity set. For
      $h = 0$, $t_v^0 = \{id(v)\}$, for $h > 0$, $t_v^h = \{id_v, id_w | w \in t_u^{h-1}, u \in \mathcal{N}(v)\}$.
    2. Sorting labels in each label multiset
    • Sort label elements in the label multiset in ascending order and concatenate them into a
      string $\mathcal{S}_v^h$.
    • Add $l_v^{h-1}$ as a prefix to $\mathcal{S}_v^h$.
    3. Label compression
    • Map each string $\mathcal{S}_v^h$ to a compressed label using a hash function $g : \sum * \rightarrow \sum$ such that
      $g(\mathcal{S}_v^h) := g(\mathcal{S}_w^h)$ if and only if $\mathcal{S}_v^h = \mathcal{S}_w^h$.
    4. Relabeling
    • Set $l_v^h := g(\mathcal{S}_v^h)$ for all nodes in the graph.
    5. $i$-th position of graph feature vector of $h$ layer
    • $\psi_i^{(h)}(G) = \sum_{l_v^h = l_m + i, v \in V} \left| t_v^h \right|$.
  end for
Output: Graph Feature Vector $\psi(G) = \left[\psi^{(0)}(G), ..., \psi^{(H)}(G)\right]$
```

A.2 Proof of Proposition 1

Proof. For graphs G and H, if they can be discriminated by 1-WL, there must exits a constant h that $\phi^{(h)}(G) \neq \phi^{(h)}(H)$. Since $\phi^{(h)}(G) = (c_h(G, \ell_1^h), ..., c_h(G, \ell_{|\mathcal{L}_h|}^h))$, there must exist a ℓ_i^h, such that $c_h(G, \ell_i^h) \neq c_h(H, \ell_i^h)$. Then there must be different subgraphs in the two graphs such that $c_h(G_v^h, \ell_i^h) \neq c_h(H_u^h, \ell_i^h)$, where G_v^h is a h-hop subgraph around node v of G. As a result, the sets of subgraph feature mappings of graph G and H are not equal, i.e., $\{\phi(G_v^h)|v \in \mathcal{V}(G)\} \neq \{\phi(H_u^h)|u \in \mathcal{V}(H)\}$. With the condition that $\mathrm{READOUT}$ is a injective function, we have $\mathrm{READOUT}(\{\phi(G_v^h)|v \in V(G)\}) \neq \mathrm{READOUT}(\{\phi(H_u^h)|u \in \mathcal{V}(H)\})$, i.e., $\psi_h(G) \neq \psi_h(H)$. In other words, the graph G and H can also be discriminated by the SaWL.

A.3 Explaination of Proposition 2

We further explain the Proposition 2 in section 3.3. For graphs G and H, if $\{\!\{s_v^h|v \in \mathcal{V}(G)\}\!\} \neq \{\!\{s_u^h|u \in \mathcal{V}(H)\}\!\}$, then the two graphs can be distinguished by the h-layer SaWL. s_v^h is the number of nodes with the exact shortest distance h from node v. When $h = 1$, if the numbers of 1-hop neighbor nodes are different in G and H, 1-WL can discriminate the two graphs, i.e., $\phi^{(h)}(G) \neq \phi^{(h)}(H)$. According to Proposition 1, SaWL can discriminate the graphs as well. Assume the number of 1-hop neighbor nodes are the same; when $h = 2$, the number of nodes with the shortest distance 2 are different in G and H. Then the sizes of 2-hop rooted subgraphs in G and H are different, which leads to the difference in the multisets of rooted subgraphs in the two graphs. With the injective readout function, the final graph feature mappings of the graph G and H are different. Similarly, assume the numbers of $(h-1)$-hop neighbor nodes in two graphs are the same. Then if the numbers of h-shortest distance nodes in two graphs are different, it results in the different multisets of rooted subgraphs and the different graph feature mappings. Therefore, the graphs G and H can be discriminated by SaWL.

For a further intuitive understanding, we take the implemented algorithm of SaWL, i.e., fast SaWL, as an example. From the perspective of the accelerating version, the size of the rooted subgraph equals the contribution of the center nodes to the whole graph feature mapping (shown in Figure 3(b)). Therefore, different sizes of rooted subgraphs directly lead to different feature mappings of the graph G and H. The graphs can be discriminated by fast SaWL.

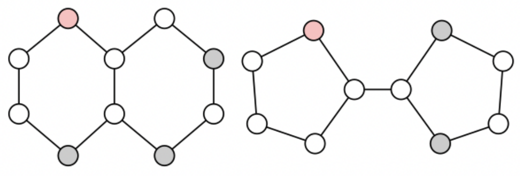
A.4 Graph Examples

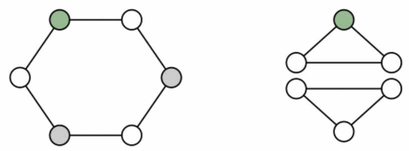
In this subsection, we provide two classes of graphs that cannot be discriminated by WL [12], but can be discriminated by the proposed SaWL. Note that the labels of all nodes in Figure 5 are the same. SaWL can discriminate the graphs only by utilizing the graph structure, and the additional label information of nodes can leave the discrimination easier.

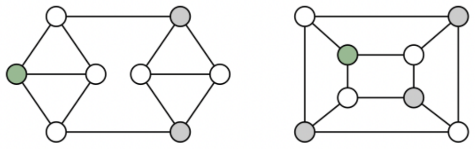
The first class is k-regular graphs of the same size (Figure 5(b)-(f)). The 6-nodes 2-regular graph in Figure 5(b), 8-nodes 3-regular graphs in Figure 5(c), 12-nodes 4-regular graphs in Figure 5(d) and two pairs of circulant graphs in Figure 5(e), 5(f) can be discriminated by 2-layer SaWL. The green nodes are center nodes, and the grey nodes are 2-hop neighbors of the green nodes. We take Figure 5(c) as example. There are two 2-hop shortest neighbors of the green node in the left graph, which are marked as grey. While for the green node in the right graph, the number of the 2-hop shortest neighbor is three (grey nodes in the right graph). According to proposition 2 in section 3.3, the left graph and the right graph can be discriminated by SaWL with two layers.

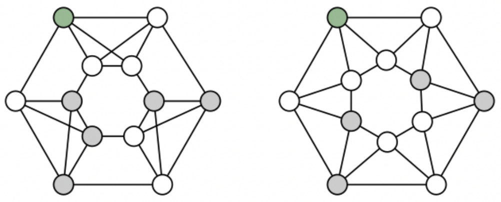
For a more intuitive understanding, we present the feature mappings of graphs in Figure 5(c) with 1-WL and our fast SaWL. We assume the initial label of each node is 0. For 1-WL, the multiset determination in the 1st and the 2nd iteration includes $0,000 \rightarrow 1; 1,111 \rightarrow 2$. The feature mappings of the graph in the left and right after the 2nd iteration are equal, i.e., $\phi(G_{left}) = \phi(G_{right}) = (8, 8, 8)$. For our fast SaWL, the feature mapping of the graph in the left is $\psi(G_{left}) = (8, 32, 52)$, while that of the right graph is $\psi(G_{right}) = (8, 32, 56)$. The difference comes from the green node and its equivalent nodes. In the left graph, label 2 occurs 52 times in all rooted subgraphs, and it occurs 56 times in the rooted subgraphs of the right graph.

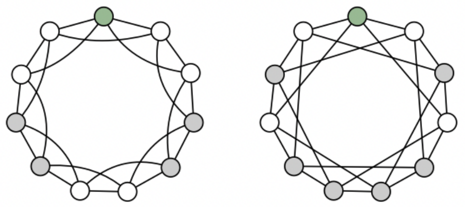
The second class includes some non-regular non-isomorphic graphs, e.g., Figure 5(a). The two graphs are non-regular graphs, but WL cannot distinguish them. SaWL can discriminate the two graphs with three layers. We take pink nodes as center nodes. For the left graph, there are three 3-hop shortest


(a) non-regular graphs.


(b) 6-nodes 2-regular graphs.


(c) 8-nodes 3-regular graphs.


(d) 12-nodes 4-regular graphs.


(e) 11-circulant graph with 2, 3 skip links.

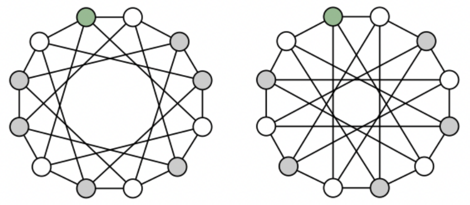

(f) 12-circulant graph with 4, 5 skip links.

Figure 5: Graph pairs can discriminated by SaWL, but not WL.

neighbors of the pink node. While for the right graph, there exist two 3-hop shortest neighbors of the pink node, which are marked as grey. Therefore, the two graphs can be distinguished by SaWL.

A.5 Comparison with WL-on-subgraphs methods

We discuss relations of the proposed methods of subgraph-aware WL (Figure 1(c)) paradigm with other methods of WL-on-subgraph paradigm (Figure 1(b)). Methods of WL-on-subgraph paradigm usually extract subgraphs around each node of the graph, then apply GNNs on each extracted subgraph respectively, such as Nested GNN [15], GNN-AK [16] and k-hop GNN [17]. However, the computation complexity of this paradigm is much higher than our proposed subgraph-aware WL paradigm. Given a graph G with N nodes, the average degree of nodes is denoted as D, and the average nodes number of subgraphs is denoted as n. Extracting k-hop subgraphs from each node takes $O(k \cdot N \cdot D)$. Applying GNNs on all extracted subgraphs takes $O(N \cdot n \cdot D)$. Totally, the computation cost is $O(k \cdot N \cdot D + N \cdot n \cdot D)$. Compared to high dimensional GNNs based on k-WL, methods of WL-on-Subgraph paradigm reduce the computational cost. However, the complexity is still much higher than that of 1-WL and our proposed methods.

Essentially, Both the proposed methods of subgraph-aware WL paradigm and the existing methods of WL-on-subgraph paradigm intend to uplift GNNs by encoding subgraphs. However, the WL-on-subgraph methods apply GNNs on all extracted subgraphs respectively, which brings high computational cost. As a contrast, our subgraph-aware WL methods encode subgraphs while keeping the computational cost low (shown in Section 3.4).

A.6 Datasets Description

We provide statistics of the datasets utilized in graph classification tasks in table 5. We adopt molecular datasets for evaluation, including TU datasets and OGB dataset. Nodes in these datasets denote atoms, and the edges denote chemical bonds. For each dataset, we present the total number of graphs, the number of positive ground truth labels, the average numbers of nodes and edges, and the types of node labels.

To empirical evaluate the expressive power, we adopted EXP [34], CSL [35] and SR25 datasets [36]. **EXP** contains 600 pair of non-isomorphic graphs, which cannot be distinguished by 1-WL. The task is to classify the graphs to 2 classes. **CSL dataset** [35] contains 150 4-regular graphs which cannot be distinguised by 1-WL. Each graph contains 41 nodes with same degree 4 and 164 edges. The task is to classify the regular graphs to 10 isomorphism classes. **SR25 dataset** [36] contains 15 strongly regular graphs. Each graph contains 25 nodes and 300 edges. The task is to classify the regular graphs to 15 different isomorphism classes. There's no node feature and edge feature in these three datasets. The model needs to utilize purely structural information to distinguish graphs.

Table 5: Statistics of datasets.

Dataset	#Graphs	#Positive	#Avg. Nodes	#Avg. Edges	#Nodes Types
MUTAG	188	125	17.9	19.8	7
PTC_MR	344	152	25.6	29.4	18
Mutagenicity	4337	2401	30.3	30.8	13
NCI1	4110	2057	29.9	32.3	37
NCI109	4127	2079	29.6	32.1	38
ogbg-molhiv	41127	1443	25.5	27.5	119

A.7 The Accelerating Version for graph isomorphism testing

The accelerating version of fast SaWL provided in Appendix A.1 can be utilized for the graph isomorphism testing, which has the same time complexity as 1-WL, but higher discriminating power than 1-WL. We first present the definition of the graph isomorphism testing, and then we explain the steps and the termination condition of the accelerating version in the graph isomorphism testing.

Graph Isomorphism Testing. Given a graph G, $\mathcal{V}(G)$ and $\mathcal{E}(G)$ are the sets of nodes and edges respectively. Two graphs G and H are isomorphic if there exists a bijection ξ between $\mathcal{V}(G)$ and $\mathcal{V}(H)$. $\xi : \mathcal{V}(G) \to \mathcal{V}(H)$ and it preserves the edge relation, i.e., $(u, v) \in \mathcal{E}(G)$ if and only if $(\xi(u), \xi(v)) \in \mathcal{E}(H)$ for all $u, v \in \mathcal{V}(G)$. Although the exact complexity of the graph isomorphism problem is still uncertain, there are some efficient graph isomorphism algorithms [11].

The Accelerating Version of Fast SaWL for Graph Isomorphism Testing. When used for the graph isomorphism testing, each iteration of the accelerating version consists of four steps, i.e., steps 1-4 of Algorithm 1. Given two graphs G and H, the accelerating version terminates after iteration h if:

$$\{(l_v^h, |t_v^h|) | v \in \mathcal{V}(G)\} \neq \{(l_u^h, |t_u^h|) | u \in \mathcal{V}(H)\}. \tag{6}$$

l_v^h denotes the label of node v in the h-th iteration, and it represents a h-height subtree pattern. t_v^h denotes the set of the node identities (IDs). It contains node identities in the subtree pattern without repetition due to the uniqueness of the node identity. The termination condition implies that fast SaWL can determine that two graphs are non-isomorphic once the updated labels or the number of nodes in the subtree patterns are different.

The terminating condition of the 1-WL can be denoted as $\{l_v^h | v \in \mathcal{V}(G)\} \neq \{l_u^h | u \in \mathcal{V}(H)\}$ [18]. The terminating condition of the accelerating version of fast SaWL (Eq. 6) is stricter than that of 1-WL by adding a new structural constraint. Therefore, once the graphs are determined unequal by the 1-WL algorithm, they must also be determined unequal by the proposed implementation. Besides, there exist many graphs that WL cannot discriminate, which can be determined as non-isomorphic (e.g., graph pairs in Figure 5). To conclude, the discriminating power of the SaWL is higher than that of 1-WL in the graph isomorphism testing.

Cases. We take the graph pair in Figure 5(c) as an example, the iteration process has been described in Appendix A.4. We denote the left graph as G and the right graph as H. After the 2nd iteration, for our fast SaWL, the set of graph G is $\{(2,6),(2,7)|v \in \mathcal{V}(G)\}$. The set of graph H is $\{(2,7)|u \in \mathcal{V}(H)\}$, and $\{(2,6),(2,7)|v \in \mathcal{V}(G)\} \neq \{(2,7)|u \in \mathcal{V}(H)\}$. The terminating condition is satisfied, and the two graphs are determined as non-isomorphic. While for 1-WL, $\{2|v \in \mathcal{V}(G)\} = \{2|u \in \mathcal{V}(H)\}$, the two graphs cannot be discriminated. All graph pairs in Figure 5 can be discriminated by fast SaWL in this way.

A.8 Ablation Study

In this section, we conduct ablation studies on number of iteration. We adopt one dataset for expressive power evaluation and one dataset for graph classification, i.e., CSL [35] and Mutagenicity [32]. We test the performance of our fast SaWL and SaGNN with different numbers of iteration from 1 to 5.

We report average accuracy of ten times running in Table 6. I=2 denotes two times iterations. From the results, it can be observed that as the number of iterations increases, the performance first improves and then drops a little. It is basically similar on both datasets. The expressive power of the models increases first and then tends to remain unchanged. The methods achieve the best results when the number of iterations is 3 or 4. When the number of iterations is 5, the performance is slightly worse, which may be caused by the increase of the dimension of the feature mapping and the increase of the model parameters. Relatively, the neural version SaGNN requires more iterations than the fast SaWL to get the best results. When the training data is sufficient, SaGNN can achieve better performance, which can be observed in Table 2 as well.

Table 6: Ablation Study on Number of Iteration (ACC).

Datasets	Iteration	I=1	I=2	I=3	I=4	I=5
CSL	fast SaWL	14.67	45.33	82.67	80.67	81.33
	SaGNN	12.67	23.33	56.67	84.67	80.00
Mutagenicity	fast SaWL	79.81	81.77	83.41	84.16	82.16
	SaGNN	79.07	81.94	83.12	84.13	83.08

A.9 Training Time Comparison

In this section, we provide comparisons of training time with two methods of WL-on-subgraph paradigm, i.e., Nested GNN [15] and GNN-AK [16]. The experimental setups are the same with Section 5.5. For a fair comparison, we set the model layer and hidden dimension the same. And for GNN-AK, we avoid using subgraph sampling modules. It is observed that the training time of the two methods of WL-on-subgraph is much higher than our SaGNN. The whole training time of GNN-AK is about 5 times that of us, and 8 times that of us for Nested GNN. Our method has a better generalization to large-scale graphs compared to the methods of WL-on-subgraph.

Table 7: Training Time Comparison(second).

Methods	Mutagenicity	NCI1	NCI109	ogbg-molhiv
Nested GNN	385	352	359	10331
GNN-AK	-	-	-	6100
SaGNN	119	128	126	1206

A.10 More Related Works

We present more related works, including graph kernel methods and traditional message passing GNNs based on the 1-WL algorithm here.

Graph kernels. Graph classification is an important task with many valuable downstream applications, such as chemical molecular property prediction [50] and pharmaceutical drug research [2]. Graph classification aims to predict the labels of given graphs by utilizing graph structure and feature information. Historically, graph kernels have been the dominant approaches for graph classification. Graph kernels first decompose the graph into different substructures, e.g., path, graphlet, and subtree, then the kernel matrix of the graphs is calculated by comparing the predefined substructures. Typical graph kernel methods include shortest path kernel [37], random walk graph kernel [51], graphlet kernel [52], and WL subtree kernel [18]. Kernel matrix is sent to kernel machine to obtain the predicted labels of graphs. However, graph kernel methods are limited for heuristic feature extraction.

GNNs based on 1-WL algorithm. Recently, Graph Neural Networks (GNNs) have been popular methods for graph classification, which made a great success [39, 53]. These methods can be viewed as the neural implementation of the 1-WL [13, 22], which first updates node representations by aggregating neighbor nodes, and then pools the nodes to obtain the graph representation. Many pooling strategies have been proposed for graph classification [23, 25, 54]. However, it has been proved that the expressive power of traditional GNNs based on 1-WL is at most as large as 1-WL [13, 14], which limits the performance of GNN-pooling methods on the graph classification task.

Substructure encoding based methods. Some methods utilize subgraph/substructure information as additional node features [41, 49]. For example, Graph Structure Networks (GSN) proposed in [49] encodes structural information in the additional preprocessing stage by counting the appearance of certain substructures as the structural feature vector. Then the structural features are utilized in message passing. The structure encoding in these method is more like a heuristic feature engineering. The selection of the certain substructures requires domain knowledge. This kind of method lacks flexibility and cannot guarantee generalization. It also requires high computational cost as choosing good substructures remains an open problem due to its combinatorial nature.

More highly expressive GNNs. ESAN [48] encodes a graph by a bag of subgraphs to achieve higher expressive power, which shares some similarities with us. However, ESAN needs some predefined policy to obtain subgraphs. The obtained subgraphs are then encoded by an equivariant architecture. It relies on the subgraph selection policy to achieve high expressivity, which loses some generalization. K-hop GNNs [17, 55] propose to aggregate the node with the information from its k-hop neighborhood, rather than only from its direct neighbors, which can identity fundamental graph properties such as connectivity and triangle freeness. K-hop GNNs leverage multi-hop information to improve the expressive power, while it has some differences from methods of WL-on-subgraph, which are discussed in [55].