A Survey on The Expressive Power of Graph Neural Networks

Ryoma Sato
r.sato@ml.ist.i.kyoto-u.ac.jp
Kyoto University / RIKEN AIP

Abstract

Graph neural networks (GNNs) are effective machine learning models for various graph learning problems. Despite their empirical successes, the theoretical limitations of GNNs have been revealed recently. Consequently, many GNN models have been proposed to overcome these limitations. In this survey, we provide a comprehensive overview of the expressive power of GNNs and provably powerful variants of GNNs.

1 Introduction

Graph neural networks (GNNs) (Gori et al., 2005; Scarselli et al., 2009) are effective machine learning models for various graph-related problems, including chemo-informatics (Gilmer et al., 2017; Zhang et al., 2018), recommender systems (Ying et al., 2018; Wang et al., 2019a,b; Fan et al., 2019; Gong et al., 2019), question-answering systems (Schlichtkrull et al., 2018; Park et al., 2019), and combinatorial problems (Khalil et al., 2017; Li et al., 2018; Gasse et al., 2019). Comprehensive surveys on GNNs were provided by Hamilton et al. (2017a), Zhou et al. (2018), and Wu et al. (2019).

Despite GNNs' empirical successes in various fields, Xu et al. (2019) and Morris et al. (2019) demonstrated that GNNs cannot distinguish some pairs of graphs. This indicates that GNNs cannot correctly classify these graphs with any parameters unless the labels of these graphs are the same. This result contrasts with the universal approximation power of multi layer perceptrons (Cybenko, 1989; Hornik et al., 1989; Hornik, 1991). Furthermore, Sato et al. (2019a) showed that GNNs are at most as powerful as distributed local algorithms (Angluin, 1980; Suomela, 2013). Thus there are many combinatorial problems that GNNs cannot solve other than the graph isomorphism problem. Consequently, various provably powerful GNN models have been proposed to overcome the limitations of GNNs.

This survey provides an extensive overview of the expressive power of GNNs and various GNN models to overcome these limitations. Unlike other surveys on GNNs, which introduce architectures and applications, this survey focuses

Table 1: Notations.

Notations	Descriptions
$\{\dots\}$	A set.
$\{\!\{\dots\}\!\}$	A multiset.
$[n]$	The set $\{1, 2, \dots, n\}$.
$a, \boldsymbol{a}, \boldsymbol{A}$	A scalar, vector, and matrix.
$\boldsymbol{A}^\top$	The transpose of $\boldsymbol{A}$.
$G = (V, E)$	A graph.
$G = (V, E, \boldsymbol{X})$	A graph with attributes.
V	The set of nodes in a graph.
E	The set of edges in a graph.
n	The number of nodes.
m	The number of edges.
$\mathcal{N}(v)$	The set of the neighboring nodes of node v.
$\deg(v)$	The degree of node v.
$\boldsymbol{X} = [\boldsymbol{x}_1, \dots, \boldsymbol{x}_n]^\top \in \mathbb{R}^{n \times d}$	The feature matrix.
$\boldsymbol{h}_v^{(l)} \in \mathbb{R}^{d_l}$	The embedding of node v in the l-th layer (Eq. 2).
$f_{\text{aggregate}}^{(l)}$	The aggregation function in the l-th layer (Eq. 1).
$f_{\text{update}}^{(l)}$	The update function in the l-th layer (Eq. 2).
f_{readout}	The readout function (Eq. 3).
Δ	The maximum degree of input graphs.
$b(k)$	The k-th bell number.
$H(k)$	The k-th harmonic number $H(k) = \frac{1}{1} + \frac{1}{2} + \dots + \frac{1}{k}$.

on the theoretical properties of GNNs. This survey is organized as follows. In the rest of this chapter, we introduce notations and review the standard GNN models briefly. In section 2, we see that GNNs cannot distinguish some graphs using elementary arguments and concrete examples. In section 3, we introduce the connection between GNNs and the WL algorithm. In section 4, we introduce the combinatorial problems that GNNs can/cannot solve in the light of the connection with distributed local algorithms. In section 5, we summarize the relationships among GNNs, the WL algorithm, and distributed local algorithms as the XS correspondence.

1.1 Notations

In this section, we introduce the notations we use in this survey. $\{\dots\}$ denotes a set, and $\{\!\{\dots\}\!\}$ denotes a multiset. A multiset is a set with possibly repeating elements. For example, $\{3, 3, 4\} = \{3, 4\}$, but $\{\!\{3, 3, 4\}\!\} \neq \{\!\{3, 4\}\!\}$. We sometimes regard a set as a multiset and vise versa. For every positive integer $n \in \mathbb{Z}_+$, $[n]$ denotes the set $\{1, 2, \dots n\}$. A small letter, such as a, b, and c, denotes a scalar, a bold lower letter, such as $\boldsymbol{a}$, $\boldsymbol{b}$, and $\boldsymbol{c}$, denotes a vector, and a bold

upper letter, such as $\boldsymbol{A}$, $\boldsymbol{B}$, and $\boldsymbol{C}$, denotes a matrix or a tensor. $\boldsymbol{A}^\top$ denotes the transpose of $\boldsymbol{A}$. For vectors $\boldsymbol{a} \in \mathbb{R}^a$ and $\boldsymbol{b} \in \mathbb{R}^b$, $[\boldsymbol{a}, \boldsymbol{b}] \in \mathbb{R}^{a+b}$ denotes the concatenation of $\boldsymbol{a}$ and $\boldsymbol{b}$. A graph is represented as a pair of the set V of nodes and the set E of edges. n denotes the number of the nodes and m denotes that number of the edges when the graph is clear from the context. For each node v, $\mathcal{N}(v)$ denotes the set of the neighboring nodes of node v, and $\deg(v)$ denotes the number of neighboring nodes of v. If a graph involves node feature vectors $\boldsymbol{X} = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n]^\top \in \mathbb{R}^{n \times d}$, the graph is represented as a tuple $G = (V, E, \boldsymbol{X})$. Table 1 summarizes notations.

1.2 Problem Setting

This survey focuses on the following node classification problem and graph classification problem.

Node Classification Problem

Input: A Graph $G = (V, E, \boldsymbol{X})$ and a node $v \in V$.

Output: The label y_v of v.

Graph Classification Problem

Input: A Graph $G = (V, E, \boldsymbol{X})$.

Output: The label y_G of G.

In particular, we consider the class of functions $f\colon (G, v) \mapsto y_v$ and $f\colon G \mapsto y_G$ that GNNs can compute because GNNs cannot model all functions on graphs, as we will see later.

1.3 Graph Neural Networks

In this section, we introduce standard GNN models briefly.

History of GNNs. Sperduti and Starita (1997) and Baskin et al. (1997) first proposed GNN-like models. They extracted features from graph data using neural networks instead of using hand-engineered graph fingerprints. Sperduti and Starita (1997) recursively applied a linear aggregation operation and non-linear activation function, and Baskin et al. (1997) used parameter sharing to model the invariant transformations on the node and edge features. These characteristics are common with modern GNNs. Gori et al. (2005) and Scarselli et al. (2009) proposed novel graph learning models that used recursive aggregation and called these models graph neural networks. It should be noted that in this survey, GNNs do not stand only for their models, but GNNs is the general term for the following variants of their models. Li et al. (2016) extended the idea of Gori et al. (2005) and Scarselli et al. (2009) to Gated Graph Neural Networks. Molecular Graph Network (Merkwirth and Lengauer, 2005) is a concurrent model of the graph neural networks with similar architecture, which uses a

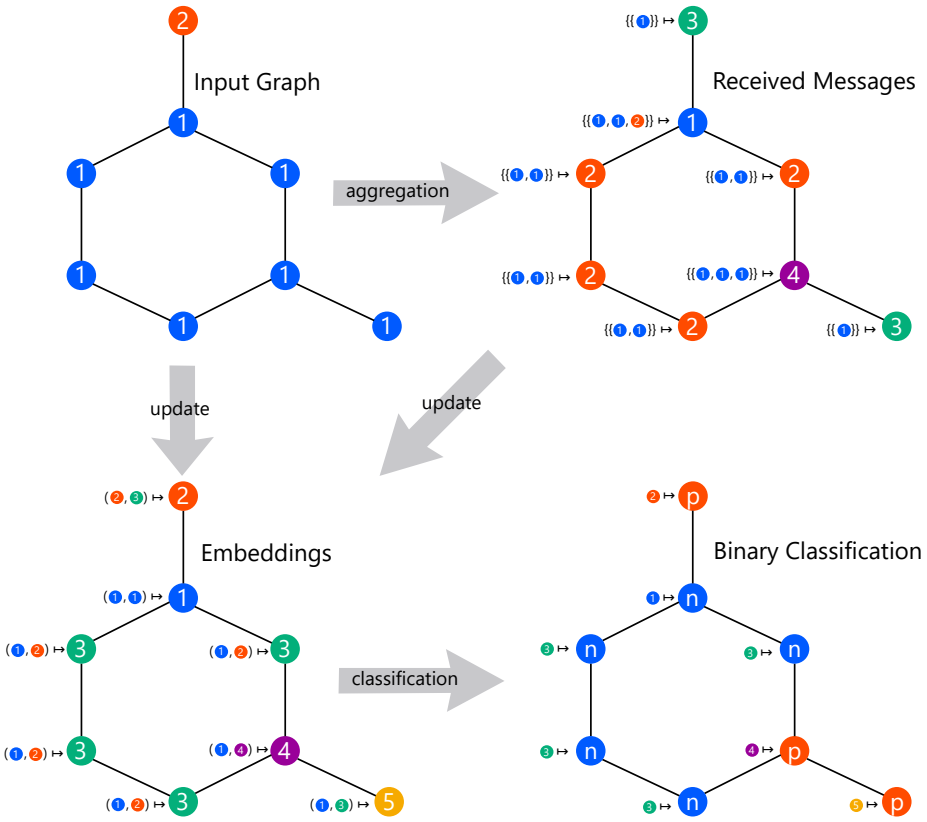


Figure 1: One-layered message passing graph neural networks.

constant number of layers. Duvenaud et al. (2015) constructed a GNN model inspired by circular fingerprints. Dai et al. (2016) proposed a GNN model inspired by the kernel message passing algorithm (Song et al., 2010, 2011). Gilmer et al. (2017) characterized GNNs using the message passing mechanism to provide a unified view of GNNs. In this survey, we do not consider spectral variants of GNN models, such as those by Bruna et al. (2014) and Defferrard et al. (2016), but spatial methods based on the message passing mechanism.

Message Passing Mechanism. In the light of the message passing mechanism, L-layered GNNs can be formulated as follows.

$$\begin{aligned}
\boldsymbol{h}_v^{(0)} &= \boldsymbol{x}_v && (\forall v \in V), \\
\boldsymbol{a}_v^{(k)} &= f_{\mathrm{aggregate}}^{(k)}(\{\!\{\boldsymbol{h}_u^{(k-1)} \mid u \in \mathcal{N}(v)\}\!\}) && (\forall k \in [L], v \in V), \quad (1) \\
\boldsymbol{h}_v^{(k)} &= f_{\mathrm{update}}^{(k)}(\boldsymbol{h}_v^{(k-1)}, \boldsymbol{a}_v^{(k)}) && (\forall k \in [L], v \in V), \quad (2)
\end{aligned}$$

where $f_{\mathrm{aggregate}}^{(k)}$ and $f_{\mathrm{update}}^{(k)}$ are (parameterized) functions. Here, $\boldsymbol{h}_u^{(k-1)}$ can be

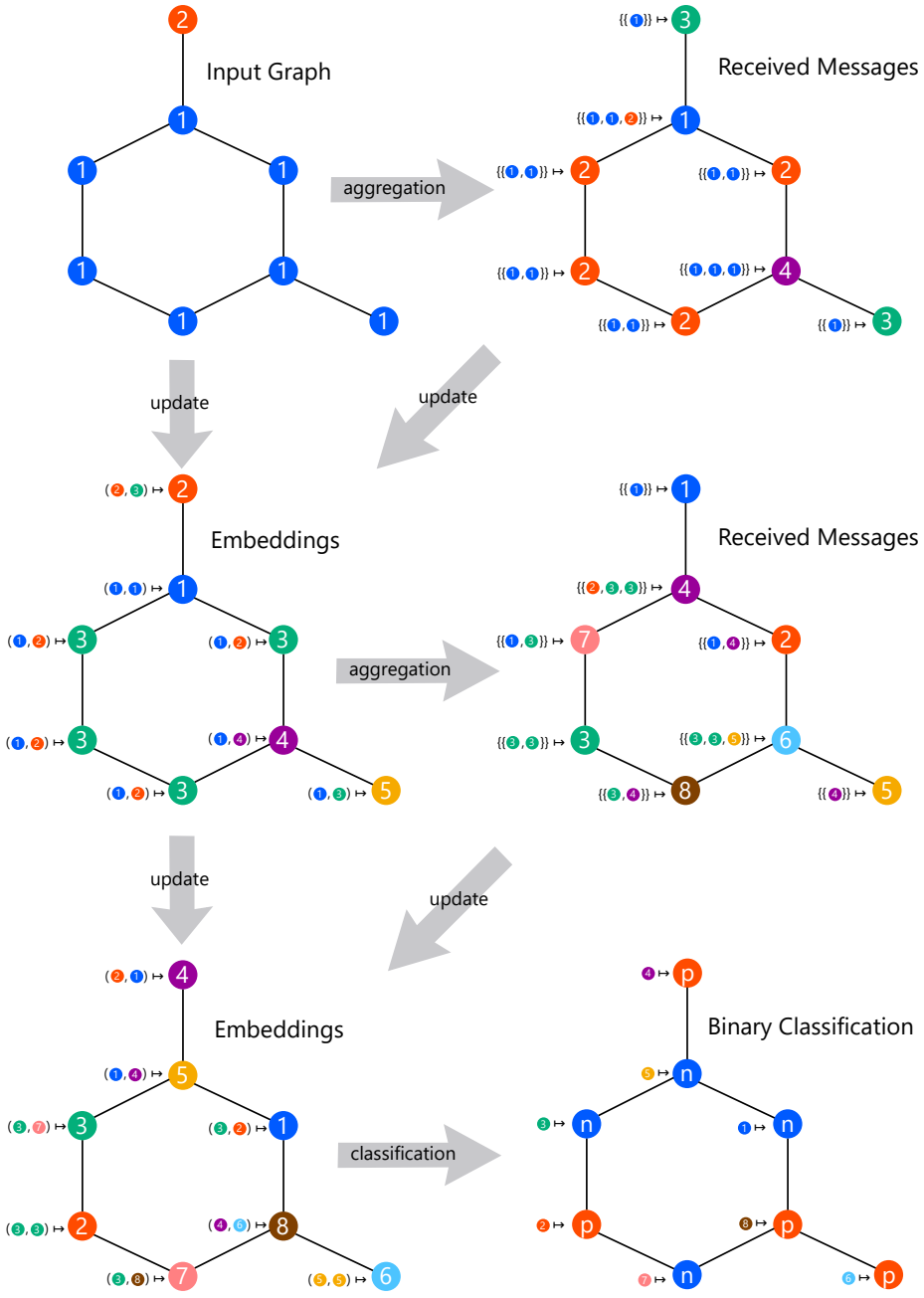


Figure 2: Two-layered message passing graph neural networks.

seen as a "message" of node u in the k-th message-passing phase. Each node aggregates messages from their neighboring nodes to compute the next message or embedding. GNNs classify node v based on the final embedding $\boldsymbol{h}_v^{(L)}$. When no node features $\boldsymbol{x}_v$ are available, we use the one-hot degree vector as the initial embedding, following Xu et al. (2019) and Knyazev et al. (2019). This scheme is illustrated in Figure 1 and 2, where colors stand for features and embeddings. The same color indicates the same vector. In this example, one-layered GNNs cannot distinguish nodes with embedding 3 in the lower-left graph in Figure 1. This indicates that if these nodes have different class labels, one-layered GNNs always fail to classify these nodes correctly because GNNs classify a node based only on the final embedding. In contrast, two-layered GNNs distinguish all nodes, as Figure 2 shows. In addition to the structural limitations, $f_{\text{aggregate}}^{(k)}$ and $f_{\text{update}}^{(k)}$ are not necessarily injective in general. For example, it is possible that $f_{\text{aggregate}}^{(k)}(\{\!\{1,1,2\}\!\}) = f_{\text{aggregate}}^{(k)}(\{\!\{1,1\}\!\})$ holds. This imposes more limitations on GNNs. This survey aims to determine the properties of graphs that GNNs can/cannot recognize.

In the graph classification problem, GNNs compute the graph embedding $\boldsymbol{h}_G$ using the readout function.

$$\boldsymbol{h}_G = f_{\text{readout}}(\{\!\{\boldsymbol{h}_v^{(L)} \mid v \in V\}\!\}), \tag{3}$$

where f_{readout} is a (parameterized) function. GNNs classify graph G based on the graph embedding $\boldsymbol{h}_G$. Typical GNN models can be formulated in the message passing framework as follows.

GraphSAGE-mean (Hamilton et al., 2017b).

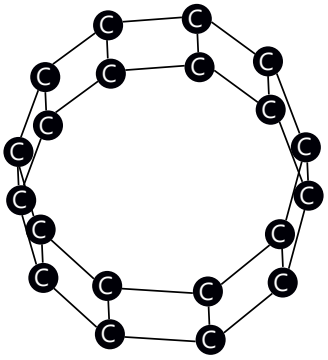
$$f_{\text{aggregate}}^{(k)}(\{\!\{\boldsymbol{h}_u^{(k-1)} \mid u \in \mathcal{N}(v)\}\!\}) = \frac{1}{\deg(v)} \sum_{u \in \mathcal{N}(v)} \boldsymbol{h}_u^{(k-1)},$$
$$f_{\text{update}}^{(k)}(\boldsymbol{h}_v^{(k-1)}, \boldsymbol{a}_v^{(k)}) = \sigma(\boldsymbol{W}^{(l)}[\boldsymbol{h}_v^{(k-1)}, \boldsymbol{a}_v^{(k)}]).$$

where $\boldsymbol{W}^{(l)}$ is a parameter matrix and σ is an activation function such as sigmoid and ReLU.

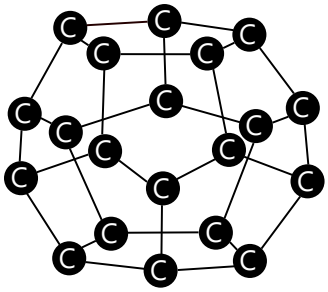
Graph Convolutional Networks (GCNs) (Kipf and Welling, 2017).

$$f_{\text{aggregate}}^{(k)}(\{\!\{\boldsymbol{h}_u^{(k-1)} \mid u \in \mathcal{N}(v)\}\!\}) = \sum_{u \in \mathcal{N}(v)} \frac{\boldsymbol{h}_u^{(k-1)}}{\sqrt{\deg(v)\deg(u)}},$$
$$f_{\text{update}}^{(k)}(\boldsymbol{h}_v^{(k-1)}, \boldsymbol{a}_v^{(k)}) = \sigma(\boldsymbol{W}^{(l)}\boldsymbol{a}_v^{(k)}).$$

Graph Attention Networks (GATs) (Veličković et al., 2018).



(a) Decaprismane.



(b) Dodecahedrane.

Figure 3: GNNs cannot distinguish these two molecules because both are 3-regular graphs with 20 nodes.

$$\alpha_{vu}^{(l)} = \frac{\exp(\textsc{LeakyReLU}(\boldsymbol{a}^{(l)\top}[\boldsymbol{W}^{(l)}\boldsymbol{h}_v^{(l-1)}, \boldsymbol{W}^{(l)}\boldsymbol{h}_u^{(l-1)}]))}{\sum_{u' \in \mathcal{N}(v)} \exp(\textsc{LeakyReLU}(\boldsymbol{a}^{(l)\top}[\boldsymbol{W}^{(l)}\boldsymbol{h}_v^{(l-1)}, \boldsymbol{W}^{(l)}\boldsymbol{h}_{u'}^{(l-1)}]))},$$

$$f_{\text{aggregate}}^{(k)}(\{\!\{\boldsymbol{h}_u^{(k-1)} \mid u \in \mathcal{N}(v)\}\!\}) = \sum_{u \in \mathcal{N}(v)} \alpha_{vu}^{(l)} \boldsymbol{h}_u^{(k-1)},$$

$$f_{\text{update}}^{(k)}(\boldsymbol{h}_v^{(k-1)}, \boldsymbol{a}_v^{(k)}) = \sigma(\boldsymbol{W}^{(l)}\boldsymbol{a}_v^{(k)}).$$

Technically, in these models, $f_{\text{aggregate}}^{(k)}$ are not functions of $\{\!\{\boldsymbol{h}_u^{(k-1)}\}\!\}$ but use side information such as the degrees of the neighboring nodes and attention weights. However, such information can be considered to be included in the message $\boldsymbol{h}_u^{(k-1)}$. Thus this abuse of notation does not affect the class of functions that these models can compute. Many other examples of message passing GNNs are provided by Gilmer et al. (2017).

2 Graphs That GNNs Cannot Distinguish

In this section, we discuss graphs that vanilla GNNs cannot distinguish via elementary arguments and concrete examples. A k-regular graph is a graph where each node has exactly k neighboring nodes. Decaprismane $C_{20}H_{20}$ (Schultz, 1965) and dodecahedrane $C_{20}H_{20}$ (Paquette et al., 1983) are examples of 3-regular graphs, as illustrated in Figure 3. It is easy to see that message passing GNNs cannot distinguish k-regular graphs with the same size and identical node features. This phenomenon is illustrated in Figure 4. These two graphs are not isomorphic because the right graph contains triangles, but the left graph does

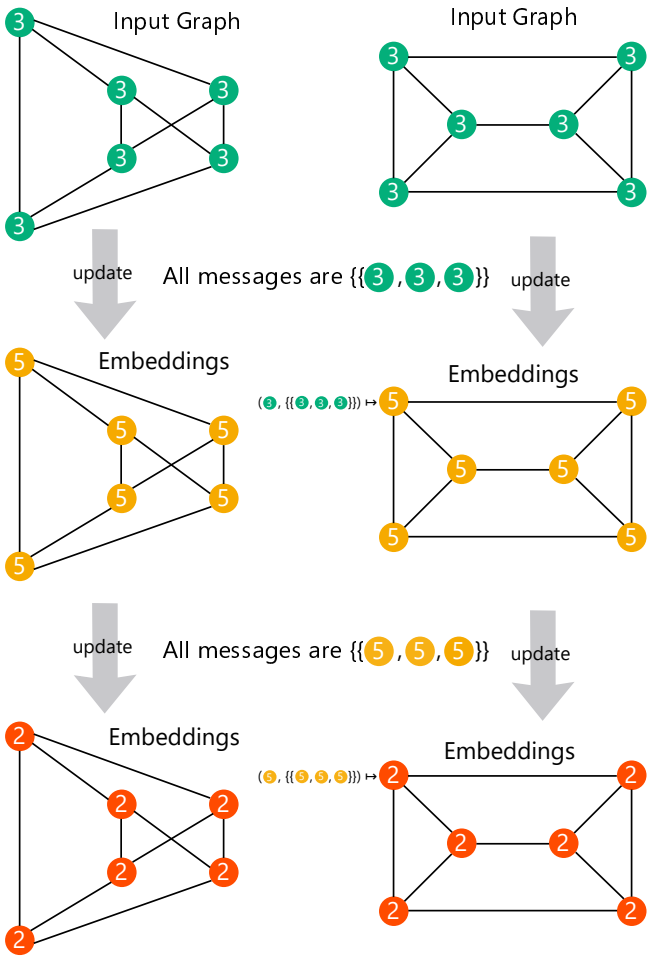


Figure 4: Message passing GNNs cannot distinguish any pair of regular graphs with the same degree and size even if they are not isomorphic.

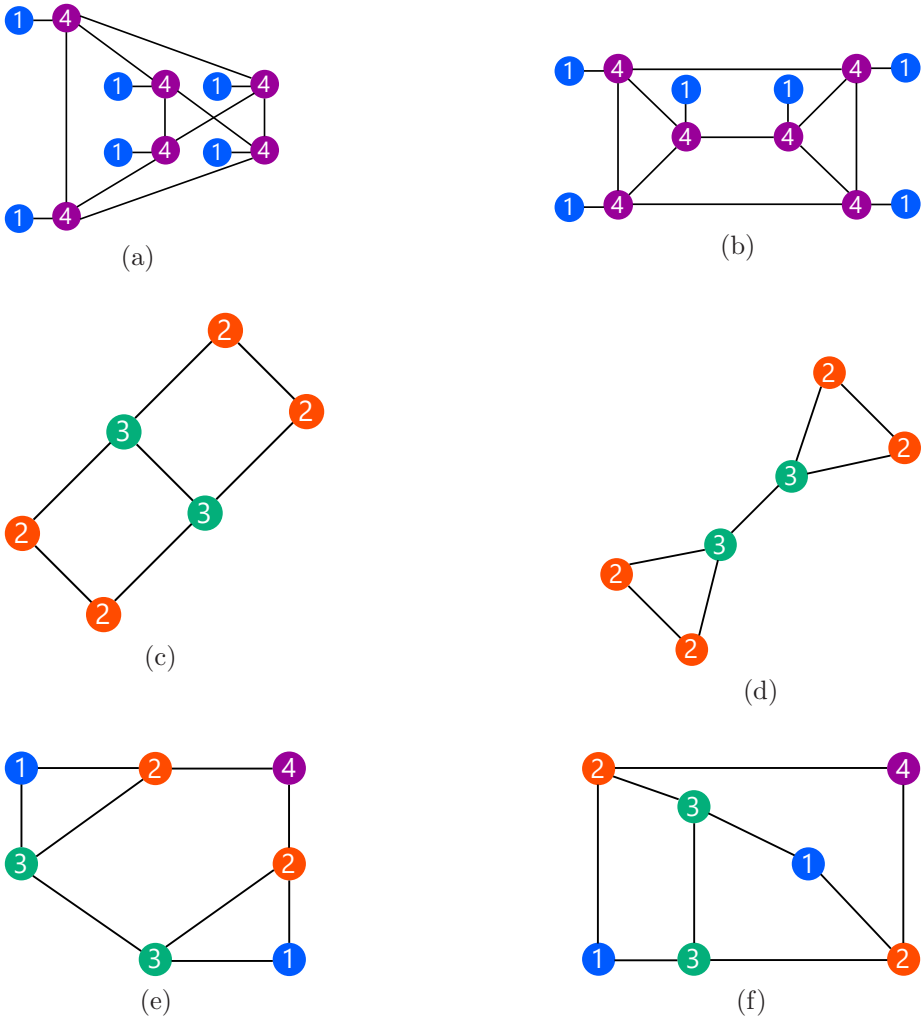
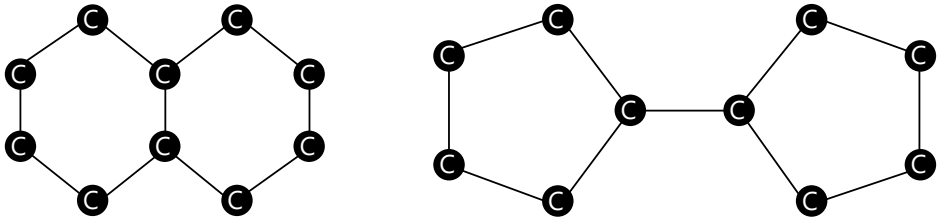


Figure 5: Although these graphs are not isomorphic or regular, GNNs cannot distinguish (a) from (b), (c) from (d), and (e) from (f)



(a) Decalin. (b) Bicyclopentyl.

Figure 6: GNNs cannot distinguish these two molecules event though these graphs are not isomorphic or regular.

not. However, all the messages are identical in all the nodes in both graphs. Thus all the final embeddings computed by the GNN are identical. This is not desirable because if two regular graphs (e.g., decaprismane $C_{20}H_{20}$ and dodecahedrane $C_{20}H_{20}$) have different class labels, message passing GNNs always fail to classify them regardless of the model parameters.

In addition to regular graphs, there are many non-regular non-isomorphic graphs that GNNs cannot distinguish. Figure 5 lists examples of pairs of non-regular non-isomorphic graphs that GNNs cannot distinguish. Figure 5 (a) and (b) are generated by attaching a "leaf" node to each node in the regular graphs in Figure 4, just as attaching a hydrogen atom. Figure 5 (e) and (f) contain node features other than degree features, but GNNs cannot distinguish them. Decalin $C_{10}H_{18}$ and bicyclopentyl $C_{10}H_{18}$ (Figure 6) are real-world graphs that GNNs cannot distinguish, by the same reason as Figure 5 (c) and (d). Again, this indicates that if these two molecules have different class labels, GNNs always fail to classify them regardless of the model parameters. There seems infinitely many graphs that GNNs cannot distinguish. Can we characterize these graphs? In the next section, we introduce the results by Xu et al. (2019) and Morris et al. (2019), who characterized them by the Weisfeiler-Lehman algorithm.

3 Connection with The WL Algorithm

In this section, we introduce the connection between GNNs and the Weisfeiler-Lehman algorithm.

3.1 Weisfeiler-Lehman Algorithm

The graph isomorphism problem is a decision problem that decides whether a pair of graphs are isomorphic or not.

Graph Isomorphism Problem

Input: A pair of graphs $G = (V, E, \boldsymbol{X})$ and $H = (U, F, \boldsymbol{Y})$.

Output: Decide whether there exists a bijection $f\colon V \to U$ such that $\boldsymbol{X}_v = \boldsymbol{Y}_{f(v)}$ $\forall v \in V$ and $\{u, v\} \in E$ iff $\{f(u), f(v)\} \in F$.

If two graphs are isomorphic, these two graphs are considered to be equivalent, and GNNs should output the same embeddings. The Weisfeiler-Lehman (WL) algorithm (Weisfeiler and Lehman, 1968) is an algorithm for the graph isomorphism problem. There are several variants of the WL algorithm. The 1-dimensional WL algorithm is the most standard variant, which is sometimes refereed to as the WL algorithm. This algorithm assigns a color to each node and refines colors until convergence.

1-dimensional WL (1-WL) algorithm (a.k.a. color refinement)
Input: A pair of graphs $G = (V, E, \boldsymbol{X})$ and $H = (U, F, \boldsymbol{Y})$.

1. $c_v^{(0)} \leftarrow \textsc{Hash}(\boldsymbol{X}_v)$ $(\forall v \in V)$
2. $d_u^{(0)} \leftarrow \textsc{Hash}(\boldsymbol{Y}_u)$ $(\forall u \in U)$
3. for $l = 1, 2, \ldots$ (until convergence)
   (a) if $\{\!\{c_v^{(l-1)} \mid v \in V\}\!\} \neq \{\!\{d_u^{(l-1)} \mid u \in U\}\!\}$ then return "non-isomorphic"
   (b) $c_v^{(l)} \leftarrow \textsc{Hash}(c_v^{(l-1)}, \{\!\{c_w^{(l-1)} \mid w \in \mathcal{N}_G(v)\}\!\})$ $(\forall v \in V)$
   (c) $d_u^{(l)} \leftarrow \textsc{Hash}(d_u^{(l-1)}, \{\!\{d_w^{(l-1)} \mid w \in \mathcal{N}_H(u)\}\!\})$ $(\forall u \in U)$
4. return "possibly isomorphic"

where Hash is an injective hash function. If 1-WL outputs "non-isomorphic", then G and H are not isomorphic, but even if 1-WL outputs "possibly isomorphic", it is possible that G and H are not isomorphic. For example, 1-WL outputs that the graphs in Figure 4 are "possibly isomorphic", although they are not isomorphic. It is guaranteed that the 1-WL algorithm stops within $O(|V| + |U|)$ iterations (Cai et al., 1992; Grohe, 2017).

The k-dimensional WL algorithm is a generalization of the 1-dimensional WL algorithm. This algorithm assigns a color to each k-tuple of nodes.

k-dimensional WL (k-WL) algorithm
Input: A pair of graphs $G = (V, E, \boldsymbol{X})$ and $H = (U, F, \boldsymbol{Y})$.

1. $c_{\boldsymbol{v}}^{(0)} \leftarrow \textsc{Hash}(G[\boldsymbol{v}])$ $(\forall \boldsymbol{v} \in V^k)$
2. $d_{\boldsymbol{u}}^{(0)} \leftarrow \textsc{Hash}(H[\boldsymbol{u}])$ $(\forall \boldsymbol{u} \in U^k)$
3. for $l = 1, 2, \ldots$ (until convergence)
   (a) if $\{\!\{c_{\boldsymbol{v}}^{(l-1)} \mid \boldsymbol{v} \in V^k\}\!\} \neq \{\!\{d_{\boldsymbol{u}}^{(l-1)} \mid \boldsymbol{u} \in U^k\}\!\}$ return "non-isomorphic"
   (b) $c_{\boldsymbol{v},i}^{(l)} \leftarrow \{\!\{c_{\boldsymbol{w}}^{(l-1)} \mid \boldsymbol{w} \in \mathcal{N}_{G,i}^{k\text{-WL}}(\boldsymbol{v})\}\!\}$ $(\forall \boldsymbol{v} \in V^k, i \in [k])$
   (c) $c_{\boldsymbol{v}}^{(l)} \leftarrow \textsc{Hash}(c_{\boldsymbol{v}}^{(l-1)}, c_{\boldsymbol{v},1}^{(l)}, c_{\boldsymbol{v},2}^{(l)}, \ldots, c_{\boldsymbol{v},k}^{(l)})$ $(\forall \boldsymbol{v} \in V)$
   (d) $d_{\boldsymbol{u},i}^{(l)} \leftarrow \{\!\{d_{\boldsymbol{w}}^{(l-1)} \mid \boldsymbol{w} \in \mathcal{N}_{H,i}^{k\text{-WL}}(\boldsymbol{u})\}\!\}$ $(\forall \boldsymbol{u} \in U^k, i \in [k])$
   (e) $d_{\boldsymbol{u}}^{(l)} \leftarrow \textsc{Hash}(d_{\boldsymbol{u}}^{(l-1)}, d_{\boldsymbol{u},1}^{(l)}, d_{\boldsymbol{u},2}^{(l)}, \ldots, d_{\boldsymbol{u},k}^{(l)})$ $(\forall \boldsymbol{u} \in U)$

4. return "possibly isomorphic"

where $\mathcal{N}^{k\text{-WL}}_{G,i}((v_1, v_2, \ldots, v_k)) = \{(v_1, \ldots, v_{i-1}, w, v_{i+1}, \ldots, v_k) \mid w \in V\}$ is the i-th neighborhood, which replaces the i-th element of a k-tuple with every node of G. HASH is an injective hash function, which assigns the same color to the same isomorphic type. In other words, $\textsc{Hash}(G[\boldsymbol{v}^1]) = \textsc{Hash}(G[\boldsymbol{v}^2])$ if and only if (1) $\boldsymbol{X}_{\boldsymbol{v}^1_i} = \boldsymbol{X}_{\boldsymbol{v}^2_i}$ $\forall i \in [k]$ and (2) $\{\boldsymbol{v}^1_i, \boldsymbol{v}^1_j\} \in E$ if and only if $\{\boldsymbol{v}^2_i, \boldsymbol{v}^2_j\} \in E$ $\forall i, j \in [k]$. The same thing holds for $\textsc{Hash}(H[\boldsymbol{u}^1])$ and $\textsc{Hash}(H[\boldsymbol{u}^2])$, and for $\textsc{Hash}(G[\boldsymbol{v}])$ and $\textsc{Hash}(H[\boldsymbol{u}])$.

The k-dimensional folklore WL algorithm is another generalization of the 1-dimensional WL algorithm.

k-dimensional folklore WL (k-FWL) algorithm

1. $c^{(0)}_{\boldsymbol{v}} \leftarrow \textsc{Hash}(G[\boldsymbol{v}])$ $(\forall \boldsymbol{v} \in V^k)$
2. $d^{(0)}_{\boldsymbol{u}} \leftarrow \textsc{Hash}(H[\boldsymbol{u}])$ $(\forall \boldsymbol{u} \in U^k)$
3. for $l = 1, 2, \ldots$ (until convergence)
   (a) if $\{\!\{c^{(l-1)}_{\boldsymbol{v}} \mid \boldsymbol{v} \in V^k\}\!\} \neq \{\!\{d^{(l-1)}_{\boldsymbol{u}} \mid \boldsymbol{u} \in U^k\}\!\}$ return "non-isomorphic"
   (b) $c^{(l)}_{\boldsymbol{v},w} \leftarrow (c^{(l-1)}_{\boldsymbol{v}[0]\leftarrow w}, c^{(l-1)}_{\boldsymbol{v}[1]\leftarrow w}, \ldots, c^{(l-1)}_{\boldsymbol{v}[k]\leftarrow w})$ $(\forall \boldsymbol{v} \in V^k, w \in V)$
   (c) $c^{(l)}_{\boldsymbol{v}} \leftarrow \textsc{Hash}(c^{(l-1)}_{\boldsymbol{v}}, \{\!\{c^{(l)}_{\boldsymbol{v},w} \mid \boldsymbol{w} \in V\}\!\})$ $(\forall \boldsymbol{v} \in V^k)$
   (d) $d^{(l)}_{\boldsymbol{u},w} \leftarrow (d^{(l-1)}_{\boldsymbol{u}[0]\leftarrow w}, d^{(l-1)}_{\boldsymbol{u}[1]\leftarrow w}, \ldots, d^{(l-1)}_{\boldsymbol{u}[k]\leftarrow w})$ $(\forall \boldsymbol{u} \in U^k, w \in U)$
   (e) $d^{(l)}_{\boldsymbol{u}} \leftarrow \textsc{Hash}(d^{(l-1)}_{\boldsymbol{u}}, \{\!\{d^{(l)}_{\boldsymbol{u},w} \mid w \in U\}\!\})$ $(\forall \boldsymbol{u} \in U^k)$
4. return "possibly isomorphic"

where $c_{(v_1,v_2,\ldots,v_k)[i]\leftarrow w} = c_{(v_1,\ldots,v_{i-1},w,v_{i+1},\ldots,v_k)}$. k-WL and k-FWL are also sound but not complete. In other words, if k-WL or k-FWL output "non-isomorphic", then G and H are not isomorphic, but even if k-WL or k-FWL output "possibly isomorphic", it is possible that G and H are not isomorphic. It should be noted that the folklore WL algorithm is sometimes refereed to as the WL algorithm in the theoretical computer science literature.

Several relations are known about the capability of the variants of the WL algorithm.

- 1-WL is as powerful as 2-WL. In other words, for any pair of graphs, the outputs of both algorithms are the same. (see, e.g., (Cai et al., 1992; Grohe and Otto, 2015b; Grohe, 2017).)
- For all $k \geq 2$, k-FWL is as powerful as $(k+1)$-WL. (see, e.g., (Grohe and Otto, 2015b; Grohe, 2017).)
- For all $k \geq 2$, $(k+1)$-WL is strictly more powerful than k-WL. In other words, there exists a pair of non-isomorphic graph (G, H) such that k-WL outputs "possibly isomorphic" but $(k+1)$-WL outputs "non-isomorphic". (see, e.g., (Grohe and Otto, 2015b, Observation 5.13 and Theorem 5.17).)

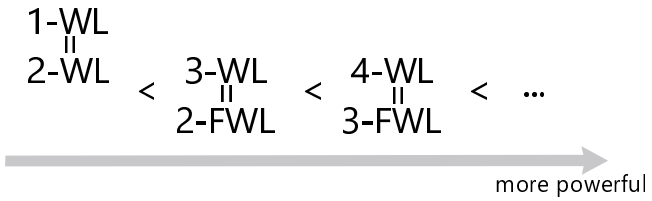


Figure 7: The hierarchy of the variants of the WL algorithm.

For example, 1-WL cannot distinguish graphs illustrated by Figure 5 (c) and (d), but 3-WL can distinguish them by detecting triangles. Figure 7 summarizes the relations.

Moreover, the graphs that the WL algorithms can/cannot distinguish have been studied extensively. Notably,

- If two graphs are taken uniformly randomly, the probability that the 1-WL algorithm fails goes to 0 as the size of graphs goes to infinity. (see, e.g., (Babai et al., 1980).)
- For all $k \geq 2$ there exists a pair of non-isomorphic graph (G, H) of size $O(k)$ such that k-WL outputs "possibly isomorphic". (see, e.g., (Cai et al., 1992, Corollary 6.5).)
- For any pair of non-isomorphic trees S and T, the 1-WL algorithm outputs S and T are "non-isomorphic". (see, e.g., (Immerman and Lander, 1990, Corollary 1.8.2).)
- For any positive integers $k, n \in \mathbb{Z}^+$ and a pair of k-regular graphs G and H with n vertices, the 1-WL algorithm outputs G and H are "possibly isomorphic". (see, e.g., (Immerman and Lander, 1990; Cai et al., 1992).)
- Graphs that the 1-WL algorithm can distinguish from any non-isomorphic graphs are recognizable in quasi linear time. (see, e.g, (Arvind et al., 2015).)

It should be noted that Babai's graph canonization algorithm (Babai et al., 1980) can be seen as a weak version of the two-step 1-WL algorithm. $f(v)$ in the Babai's algorithm corresponds to $c_v^{(1)}$ in 1-WL, but $f(v)$ discards the information from low degree colors.

3.2 Connection between GNNs And The WL Algorithm

In this section, we introduce the connection between GNNs and the WL algorithm, found by Xu et al. (2019) and Morris et al. (2019). First, the following theorem is easy to see.

Theorem 1 (Xu et al. (2019); Morris et al. (2019)). *For any message passing GNN and for any graphs G and H, if the 1-WL algorithm outputs that G and H are "possibly isomorphic", the embeddings $\boldsymbol{h}_G$ and $\boldsymbol{h}_H$ computed by the GNN are the same.*

In other words, message passing GNNs are less powerful than the 1-WL algorithm. This is because the aggregation and update functions can be seen as the hash function of the 1-WL algorithm, and the aggregation and update functions are not necessarily injective. In the light of the correspondence between the GNNs and the 1-WL algorithm, Xu et al. (2019) proposed a GNN model that is as powerful as the 1-WL algorithm, by making the aggregation and update functions injective.

Graph Isomorphic Networks (GINs) (Xu et al., 2019).

$$f^{(k)}_{\text{aggregate}}(\{\!\{\boldsymbol{h}^{(k-1)}_u \mid u \in \mathcal{N}(v)\}\!\}) = \sum_{u \in \mathcal{N}(v)} \boldsymbol{h}^{(k-1)}_u,$$

$$f^{(k)}_{\text{update}}(\boldsymbol{h}^{(k-1)}_v, \boldsymbol{a}^{(k)}_v) = \text{MLP}((1+\varepsilon^{(k)})\boldsymbol{h}^{(k-1)}_v + \boldsymbol{a}^{(k)}_v)),$$

where $\varepsilon^{(k)}$ is a scalar parameter, and MLP is a multi layer perceptron. Owing to the theory of deep multisets (Xu et al., 2019; Zaheer et al., 2017), which states that the aggregation of GINs is injective under some assumptions, GINs are as powerful as 1-WL.

Theorem 2 (Xu et al. (2019)). *For all $L \in \mathbb{Z}_+$, there exist parameters of L-layered GINs such that if the degrees of the nodes are bounded by a constant and the size of the support of node features is finite, for any graphs G and H, if the 1-WL algorithm outputs that G and H are "non-isomorphic" within L rounds, the embeddings $\boldsymbol{h}_G$ and $\boldsymbol{h}_H$ computed by the GIN are different.*

This result is strong because this says that there exists a fixed set of parameters of GINs that can distinguish all graphs in a graph class. This result contrasts with other results, which we will see later, that restrict the size of graphs or fix a pair of graphs beforehand. It should be noted that Theorem 5 in Xu et al. (2019) assumes that the support of node features is countable, but MLPs cannot necessarily approximate the function f keeping injective if the support size is infinite. Thus Theorem 2 assumes the support size is finite. The following corollary is straightforward from Theorem 2.

Corollary 3. *For any graphs G and H, if the 1-WL algorithm outputs that G and H are "non-isomorphic", there exist parameters of GINs such that the embeddings $\boldsymbol{h}_G$ and $\boldsymbol{h}_H$ computed by the GIN are different.*

Unlike Theorem 2, Corollary 3 does not assume that the degrees of the nodes or the size of the support of the node features are bounded because once the input graphs are fixed, the degrees of the nodes, the size of the support, and the round that 1-WL stops are constants. Since the 1-WL algorithm defines

the upper bound of the expressive power of message passing GNNs, GINs are sometimes referred to as the most powerful message passing GNNs. How can we build more powerful GNNs than the 1-WL algorithm? Morris et al. (2019) proposed a more powerful model based on a variant of the k-WL algorithm. They used the set k-WL algorithm instead of the k-WL algorithm to reduce memory consumption. We first introduce the set k-WL algorithm. Let $[V]_k = \{S \subseteq V \mid |S| = k\}$ be the set of k-subsets of V, and $\mathcal{N}^{\mathrm{set}}_{V,k}(S) = \{W \in [V]_k \mid |W \cup V| = k-1\}$. The set k-WL algorithm assigns a color to each k-set of the nodes.

Set k-dimensional WL (set k-WL) algorithm

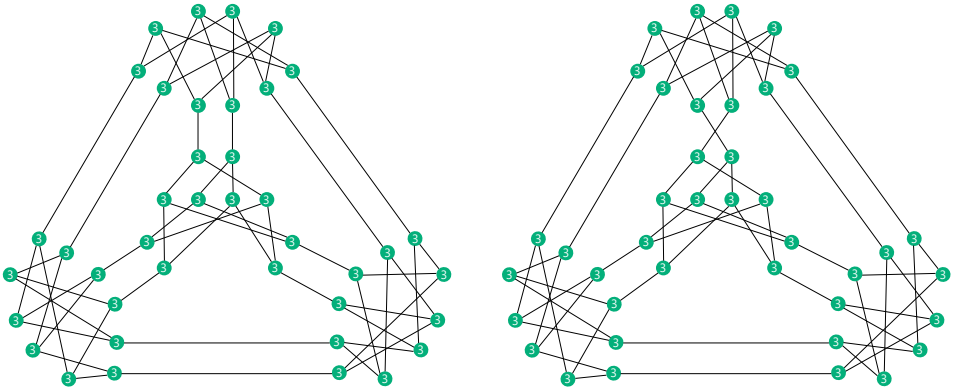
1. $c_S^{(0)} \leftarrow \textsc{Hash}(G[S])$ $(\forall S \in [V]_k)$
2. $d_T^{(0)} \leftarrow \textsc{Hash}(H[T])$ $(\forall T \in [U]_k)$
3. for $l = 1, 2, \ldots$ (until convergence)
   (a) if $\{\!\{c_S^{(l-1)} \mid S \in [V]_k\}\!\} \neq \{\!\{d_T^{(l-1)} \mid T \in [U]_k\}\!\}$ return "non-isomorphic"
   (b) $c_S^{(l)} \leftarrow \textsc{Hash}(c_S^{(l-1)}, \{\!\{c_W^{(l-1)} \mid W \in \mathcal{N}^{\mathrm{set}}_{V,k}(S)\}\!\})$ $(\forall S \in [V]_k)$
   (c) $d_T^{(l)} \leftarrow \textsc{Hash}(d_T^{(l-1)}, \{\!\{d_W^{(l-1)} \mid W \in \mathcal{N}^{\mathrm{set}}_{U,k}(T)\}\!\})$ $(\forall T \in [U]_k)$
4. return "possibly isomorphic"

where $\textsc{Hash}(G[S])$ is an injective hash function that assigns a color based on the subgraph induced by S. The set 3-WL algorithm can detect the number of triangles, whereas 1-WL cannot. Therefore, the set 3-WL algorithm can distinguish Figure 5 (a) from (b), (c) from (d), and (e) from (f), whereas the 1-WL algorithm cannot. This characteristic of the set 3-WL is desirable because the number of triangles (i.e., clustering coefficient) plays an important role in various networks (Milo et al., 2002; Newman, 2003). However, the set 3-WL algorithm cannot distinguish graphs in Figure 8 (a) and (b) because the 3-WL algorithm cannot distinguish them (Cai et al., 1992; Grohe, 2017). It should be noted that the set k-WL algorithm is strictly weaker than the k-WL algorithm. For example, 3-WL can distinguish Figure 9 (a) and (b) because the 3-WL algorithm can detect the number of 4-cycles (Fürer, 2017, Theorem 2), but the set 3-WL algorithm cannot.

Morris et al. (2019) proposed k-dimensional GNNs (k-GNNs), based on the set k-WL algorithm. k-GNNs assigns an embedding to each k-set of the nodes as follows.

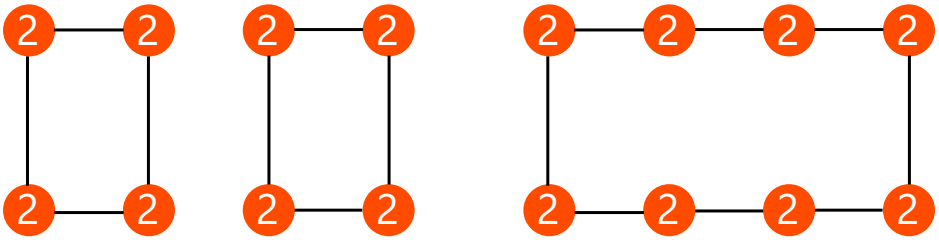
k-dimensional GNNs (k-GNNs).

$$\begin{aligned}
\boldsymbol{h}_S^{(0)} &= f^{(0)}(G[S]), \\
\boldsymbol{h}_S^{(k)} &= \mathbf{W}_1^{(k)} \boldsymbol{h}_S^{(k-1)} + \sum_{W \in \mathcal{N}^{\mathrm{set}}_{V,k}(S)} \mathbf{W}_2^{(k)} \boldsymbol{h}_W^{(k-1)},
\end{aligned}$$



(a) (b)

Figure 8: Although these graphs are not isomorphic, neither the 3-dimensional WL algorithm nor 3-GNNs can distinguish (a) from (b)



(a) (b)

Figure 9: The 3-dimensional WL algorithm can distinguish these graphs, but the set 3-dimensional WL algorithm cannot.

where $f^{(0)}(G[S])$ assigns a feature vector based on the subgraph induced by S. k-GNNs can be seen as message passing GNNs that operate on the extended graph $G^{\otimes k}$, where the set of the nodes is $[V]_k$, and there exists an edge between $S \in [V]_k$ and $T \in [V]_k$ if and only if $T \in \mathcal{N}^{\text{set}}_{V,k}(S)$. k-GNNs are as powerful as the set k-WL algorithm.

Theorem 4 (Morris et al. (2019), Proposition 4)**.** *For any graphs G and H, $k \geq 2$, if the set k-WL algorithm outputs that G and H are "non-isomorphic", there exist parameters of k-GNNs such that the embeddings $\boldsymbol{h}_G$ and $\boldsymbol{h}_H$ computed by the k-GNNs are different.*

A drawback of k-GNNs is that k-GNNs consume too much memory by maintaining $O(n^k)$ embeddings. Maron et al. (2019a) alleviated this problem by proposing a GNN model that maintains $O(n^2)$ embeddings but has the same power as 3-WL, based on the 2-FWL algorithm. We first introduce higher order invariant and equivariant GNNs, which are the building blocks of the model of

Maron et al. (2019a).

3.3 Higher Order Graph Neural Networks

In this section, we introduce the higher order invariant and equivariant GNNs (Maron et al., 2019b). First, we define the concept of invariance and equivariance formally. Let S_n be the symmetric group over $[n]$. For a tensor $\boldsymbol{X} \in \mathbb{R}^{n^k}$ and a permutation $p \in S_n$, we define $(p \cdot \boldsymbol{X}) \in \mathbb{R}^{n^k}$ be $(p \cdot \boldsymbol{X})_{p(i_1),p(i_2),\ldots,p(i_n)} = \boldsymbol{X}_{i_1,i_2,\ldots,i_n}$. For a tensor $\boldsymbol{X} \in \mathbb{R}^{n^k \times d}$ and a permutation $p \in S_n$, we define $(p \cdot \boldsymbol{X}) \in \mathbb{R}^{n^k \times d}$ be $(p \cdot \boldsymbol{X})_{p(i_1),p(i_2),\ldots,p(i_n),j} = \boldsymbol{X}_{i_1,i_2,\ldots,i_n,j}$.

Definition 5 (Invariance)**.** A function $f\colon \mathbb{R}^{n^k} \to \mathbb{R}$ is invariant if for any $\boldsymbol{X} \in \mathbb{R}^{n^k}$ and $p \in S_n$, $f(p \cdot \boldsymbol{X}) = f(\boldsymbol{X})$ holds. A function $f\colon \mathbb{R}^{n^k \times d} \to \mathbb{R}$ is invariant if for any $\boldsymbol{X} \in \mathbb{R}^{n^k \times d}$ and $p \in S_n$, $f(p \cdot \boldsymbol{X}) = f(\boldsymbol{X})$ holds.

Definition 6 (Equivariance)**.** A function $f\colon \mathbb{R}^{n^k} \to \mathbb{R}^{n^l}$ is invariant if for any $\boldsymbol{X} \in \mathbb{R}^{n^k}$ and $p \in S_n$, $f(p \cdot \boldsymbol{X}) = p \cdot f(\boldsymbol{X})$ holds. A function $f\colon \mathbb{R}^{n^k \times d} \to \mathbb{R}^{n^l \times d'}$ is equivariant if for any $\boldsymbol{X} \in \mathbb{R}^{n^k \times d}$ and $p \in S_n$, $f(p \cdot \boldsymbol{X}) = p \cdot f(\boldsymbol{X})$ holds.

Thus invariance is a special case of equivariance with $l = 0$. It is natural to ask that graph learning models should be invariance or equivariance because a graph is not changed by any node permutation. Maron et al. (2019b) generalized the ideas of Zaheer et al. (2017), Kondor et al. (2018), and Hartford et al. (2018), and enumerated *all* invariant and equivariant linear transformations. Surprisingly, they found that the sizes of bases of the invariant and equivariant linear transformations are independent of the dimension n. Specifically, the size of a basis of the linear invariant transformations $\mathbb{R}^{n^k} \to \mathbb{R}$ is $b(k)$, and the size of a basis of the linear invariant transformations $\mathbb{R}^{n^k} \to \mathbb{R}^{n^l}$ is $b(k+l)$, where $b(k)$ is the k-th bell number (i.e., the number of the partitions of $[k]$). The bases can be constructed as follows. Let $\mathcal{B}_k$ be the set of all the partitions of $[k]$. For example, $\{\{1,3\},\{2,5\},\{4\}\} \in \mathcal{B}_5$. For each partition $B \in \mathcal{B}_k$ and index $\boldsymbol{a} \in [n]^k$, let $\boldsymbol{B}^B_{\boldsymbol{a}} = 1$ if $\boldsymbol{a}_x = \boldsymbol{a}_y \Leftrightarrow \exists S \in B$ s.t. $\{x, y\} \subseteq S$ holds, and $\boldsymbol{B}^B_{\boldsymbol{a}} = 0$ otherwise. For example, $\boldsymbol{B}^{\{\{1,3\},\{2,5\},\{4\}}_{(7,3,7,6,3)} = 1$.

Theorem 7 (Maron et al. (2019b), Proposition 2)**.** $\{\boldsymbol{B}^B \in \mathbb{R}^{n^k} \mid B \in \mathcal{B}_k\}$ *forms an orthogonal basis of the linear invariant transformations* $\mathbb{R}^{n^k} \to \mathbb{R}$.

Theorem 8 (Maron et al. (2019b))**.** $\{\boldsymbol{B}^B \in \mathbb{R}^{n^k \times n^l} \mid B \in \mathcal{B}_{k+l}\}$ *forms an orthogonal basis of the linear equivariant transformations* $\mathbb{R}^{n^k} \to \mathbb{R}^{n^l}$.

Therefore, any linear invariant and equivariant transformation can be modeled by a linear combination of $b(k)$ and $b(k+l)$ basis elements, respectively. For example, in the case of $k = l = 2$, there are $b(2) = 2$ elements (i.e., the summation of diagonal elements and the summation of all the non-diagonal elements) in a basis of the linear invariant transformations and $b(2+2) = b(4) = 15$

elements in a basis of the linear equivariant transformations. If tensors have a features axis (i.e., $\boldsymbol{X} \in \mathbb{R}^{n^k \times d}$), the basis of linear invariant transformations $\mathbb{R}^{n^k \times d} \to \mathbb{R}^{d'}$ consists of $dd'b(k)$ elements, and the basis of linear equivariant transformations $\mathbb{R}^{n^k \times d} \to \mathbb{R}^{n^l \times d'}$ consists of $dd'b(k+l)$ elements. Therefore, a linear invariant layer $\mathbb{R}^{n^k \times d} \to \mathbb{R}^{d'}$ has $dd'b(k) + d'$ parameters, and a linear equivariant layer $\mathbb{R}^{n^k \times d} \to \mathbb{R}^{n^l \times d'}$ has $dd'b(k+l) + d'b(l)$ parameters including biases. Maron et al. (2019b) proposed to utilize these transformations as building blocks of (not message passing) GNNs. In particular, they consider GNNs of the form $f = m \circ h \circ \sigma \circ g_L \circ \sigma \circ g_{L-1} \circ \sigma \circ \cdots \circ \sigma \circ g_1$, where $m \colon \mathbb{R}^{d_L} \to \mathbb{R}^{d_L+1}$ is modeled by a multi layer perceptron, $h \colon \mathbb{R}^{n^{k_L} \times d_L} \to \mathbb{R}^{d_L}$ is a linear invariant layer, σ is an elementwise activation function (e.g., a sigmoid function or ReLU function), and $g_l \colon \mathbb{R}^{n^{k_{l-1}} \times d_{l-1}} \to \mathbb{R}^{n^{k_l} \times d_l}$ is a linear equivariant layer. How can we feed a graph into this model? When the input is a graph $G = (V, E, \boldsymbol{X})$ with n nodes and d dimensional attributes, the order of the input tensor $\boldsymbol{C}$ is $k_0 + 1 = 3$, and the input tensor $\boldsymbol{C} \in \mathbb{R}^{n \times n \times (d+1)}$ consists of three parts. The last channel is the adjacency matrix $\boldsymbol{A}$ of G (i.e., $\boldsymbol{C}_{i,j,d} = \boldsymbol{A}_{i,j}$), the diagonal elements of the first d channels are feature vectors (i.e., $\boldsymbol{C}_{i,i,k} = \boldsymbol{x}_{i,k}$), and the non-diagonal elements of the first d channels are zero (i.e., $\boldsymbol{C}_{i,j,k} = 0$ $(i \neq j)$). We call this model higher order GNNs. If the orders of the tensors are at most k, we call this model k-th order invariant graph networks (k-IGNs). Obviously, higher order GNNs are always invariant because each layer is invariant or equivariant.

Maron et al. (2019c) showed that for any subgroup γ of S_n, higher order GNNs can approximate any γ-invariant function, and that order $n(n-1)/2$ is sufficient for universality. Keriven and Peyré (2019) showed that a variant of higher order GNNs has the universality for first order S_n-equivariant functions $\mathbb{R}^{n^k} \to \mathbb{R}^n$. Although universality is a strong merit of higher order GNNs, they have two major drawbacks. First, the number n of nodes is fixed (or bounded) in their analyses, and these theoretical results cannot be applied to graphs of variable sizes. Second, they use too much parameters for practical use because the bell number grows super-exponentially. For example, even with $n = 6$ nodes, there are at least $b(n(n-1)/2) = b(15) = 1382958545 \approx 10^9$ parameters in the invariance case, and Keriven and Peyré (2019) did not bound the order of tensors in the equivariance case. Maron et al. (2019a) alleviated the latter problem by pointing out the connection between the higher-order GNNs and the higher-order WL algorithm, and proposing a new higher-order GNN model based on the higher-order FWL algorithm. First, Maron et al. (2019a) showed that the higher-order GNNs with k-th order tensors are as powerful as k-WL.

Theorem 9 (Maron et al. (2019a), Theorem 1)**.** *For any graphs G, H, if the k-WL algorithm outputs that G and H are "non-isomorphic", there exist parameters of k-IGNs such that the embeddings of G and H computed by the k-IGN are different.*

This indicates that n-th order tensors are sufficient and necessary to distinguish any pair of non-isomorphic graphs because **necessity:** there is a pair of

non-isomorphic graphs of size $O(k)$ that the k-WL algorithm cannot distinguish (Cai et al., 1992), and **sufficiency:** the n-WL can distinguish all graphs of size n. This is better bound than the order $n(n-1)/2$. It should be noted that Theorem 9 does not state that there exists a fixed set of parameters of higher order GNNs that can distinguish all graphs that the k-WL algorithm can distinguish, but it states that there exists a set of parameters for every pair of graphs that the k-WL algorithm can distinguish. In particular, a k-th order GNN with a fixed set of parameters is not necessarily as powerful as the k-WL algorithm when the number of nodes is not bounded. Although Maron et al. (2019a) did not show the tight upper bound of the power of k-IGNs, later, Chen et al. (2020) showed the tight upper bound for 2-IGNs and thus the equivalence of 2-IGNs and 2-WL.

Theorem 10 (Chen et al. (2020), Theorem 6)**.** *For any graphs G, H, if the 2-WL algorithm outputs that G and H are "possibly isomorphic", there do not exist parameters of 2-IGNs such that the embeddings of G and H computed by the 2-IGN are different.*

Then, Maron et al. (2019a) proposed a GNN model that is as powerful as 2-FWL. The model has the form $f = m \circ h \circ g_L \circ g_{L-1} \circ \cdots \circ g_1$, where $m \colon \mathbb{R}^{d_L} \to \mathbb{R}^{d_L+1}$ is modeled by a multi layer perceptron, $h \colon \mathbb{R}^{n^2 \times d_L} \to \mathbb{R}^{d_L}$ is a linear invariant layer, and $g_l \colon \mathbb{R}^{n^2 \times d_{l-1}} \to \mathbb{R}^{n^2 \times d_l}$ consists of three multi layer perceptrons $m_1, m_2 \colon \mathbb{R}^{d_{l-1}} \to \mathbb{R}^{d'_l}$ and $m_3 \colon \mathbb{R}^{d_{l-1}} \to \mathbb{R}^{d''_l}$. These multi layer perceptrons are applied to each feature of the input tensor $\boldsymbol{X}$ independently (i.e., $m_l(\boldsymbol{X})_{i_1,i_2,:} = m_l(\boldsymbol{X}_{i_1,i_2,:})$). Therefore, $m_1(\boldsymbol{X}), m_2(\boldsymbol{X}) \in \mathbb{R}^{n^2 \times d'_l}$. Then, matrix multiplication is performed $\boldsymbol{Y}_{:,:,i_3} = m_1(\boldsymbol{X})_{:,:,i_3} m_2(\boldsymbol{X})_{:,:,i_3}$. The output of the layer g_l is the concatenation $[m_3(\boldsymbol{X}), \boldsymbol{Y}] \in \mathbb{R}^{n^2 \times (d'_l + d''_l)}$ of the third multi layer perceptron and $\boldsymbol{Y}$. We call this model second order folklore GNNs. Second order folklore GNNs are as powerful as the 2-FWL algorithm. Intuitively, the matrix multiplication $\boldsymbol{C}_{ij} = \sum_{w \in V} \boldsymbol{A}_{iw} \boldsymbol{B}_{wj}$ corresponds to the aggregation $\{\!\{(c^{(l-1)}_{(i,w)}, c^{(l-1)}_{(w,j)}) \mid w \in V\}\!\}$ of 2-FWL. Formally, the following theorem holds.

Theorem 11 (Maron et al. (2019a), Theorem 2)**.** *For any graphs G, H, if the 2-FWL algorithm outputs that G and H are "non-isomorphic", there exist parameters of second order folklore GNNs such that the embeddings of G and H computed by the second order folklore GNN are different.*

Again, Theorem 11 does not state that there exists a fixed set of parameters of second order folklore GNNs that can distinguish all graphs that the 2-FWL algorithm can distinguish. Since 2-FWL is as powerful as 3-WL, the second order folklore GNNs are also as powerful as 3-WL. The strong point of the second order folklore GNNs is that they maintain only $O(n^2)$ embeddings. This means that second order folklore GNNs are more memory efficient than third order GNNs. In parallel to second order folklore GNNs, Chen et al. (2019) proposed Ring-GNNs, which also uses matrix multiplication and similar architecture to second order folklore GNNs. Maron et al. (2019a) further generalized second order folklore GNNs to higher order by using higher order tensor multiplication.

3.4 Relational Pooling

In this section, we introduce another approach to build powerful GNNs proposed by Murphy et al. (2019b). The idea is very simple. The relational pooling layer takes an average of all permutations, as Jannosy pooling (Murphy et al., 2019a). Namely, let f be a message passing GNN, $\boldsymbol{A}$ be the adjacency matrix of $G = (V, E, \boldsymbol{X})$, $\boldsymbol{I}_n \in \mathbb{R}^{n \times n}$ be the identity matrix, and S_n be the symmetric group over V. Then, relational pooling GNNs (RP-GNNs) are defined as follows.

$$\bar{\bar{f}}(\boldsymbol{A}, \boldsymbol{X}) = \frac{1}{n!} \sum_{p \in S_n} f(\boldsymbol{A}, [\boldsymbol{X}, p \cdot \boldsymbol{I}_n]).$$

In other words, RP-GNNs concatenate a one-hot encoding of the node index to the node feature, and take an average of all permutations. The strong points of RP-GNNs is that they are obviously permutation invariant by construction and are more powerful than GINs and the 1-WL algorithm.

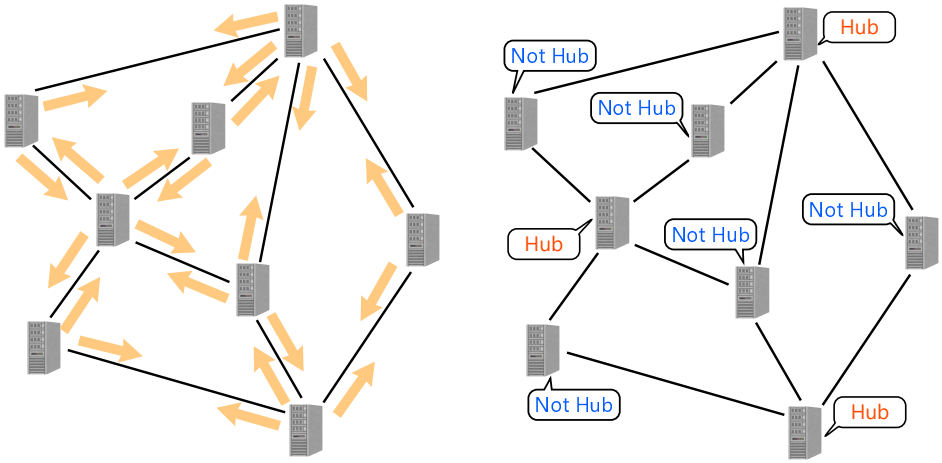
Theorem 12 (Murphy et al. (2019b), Theorem 2.2)**.** *The RP-GNNs are more powerful than the original message passing GNNs f. In particular, if f is modeled by GINs (Xu et al., 2019), and the graph has discrete attributes, the RP-GNNs are more powerful than the 1-WL algorithm.*

However, RP-GNNs have two major drawbacks. First, RP-GNNs cannot handle graphs with different sizes since the dimension of feature vectors depend on the number n of nodes. This drawback is alleviated by adding dummy nodes when the upper bound of the number of nodes is known beforehand. The second drawback is its computation cost since it takes average of $n!$ terms, which is not tractable in practice (e.g., $15! = 1307674368000 \approx 10^{12}$). To overcome the second issue, Murphy et al. (2019b) proposed three approximation methods. The first method uses canonical orientations, which first computes one or several canonical labeling by breath-first search, depth-first search, or using centrality scores as Niepert et al. (2016). The second method samples some permutations, instead of using all permutations. The third method uses only a part of graphs, instead of all nodes.

Chen et al. (2020) proposed another model based on the relational pooling. They considered the substructure counting problem to assess the expressive power of GNNs and showed that message passing GNNs cannot count any connected substructures with 3 or more nodes. They also showed that even k-th order GNNs cannot count some substrucutures (namely long paths). To overcome this issue, they proposed Local Relational Pooling, which applies the relational pooling to each egonet separately.

4 Connection with Combinatorial Problems

So far, we have considered the graph isomorphic problem. In this section, we consider other combinatorial graph problems such as the minimum vertex cover problem, minimum dominating set problem, and maximum matching problem,



(a) Communicating with neighboring computers.

(b) Each node answers after a constant number of communications.

Figure 10: Illustration of distributed local algorithms.

and assess the expressive power of GNNs via the lens of the efficiency of algorithms that GNNs can compute. In particular, we introduce the connection of GNNs with distributed local algorithms. It should be noted that GNNs are not necessarily invariant or equivariant in this section for modeling combinatorial algorithms.

4.1 Distributed Local Algorithms

In this section, we introduce distributed local algorithms briefly. A distributed local algorithm is a distributed algorithm that runs in constant time. In particular, a distributed local algorithm solves a problem on the computer network itself, each computer runs the same program, and each computer decides the output after a constant number of synchronous communication rounds with neighboring computers. For example, suppose mobile devices construct a communication network with near devices. The communication network must contain hub nodes to control communications, and hubs must form a vertex cover of the network (i.e., each edge is covered by at least hub nodes). The number of hub nodes should be as small as possible, but each mobile devices have to declare to be a hub within five communication rounds. How can we design the algorithm for these devices? Figure 10 illustrates this problem.

Distributed local algorithms were first studied by Angluin (1980), Linial (1992), and Naor and Stockmeyer (1995). Angluin (1980) introduced a port numbering model and showed that deterministic distributed algorithms cannot find a center of a graph without unique node identifiers. Linial (1992) showed that no distributed *local* algorithms can solve 3-coloring of cycles, and they require $\Omega(\log^* n)$ communication rounds for distributed algorithms to solve the

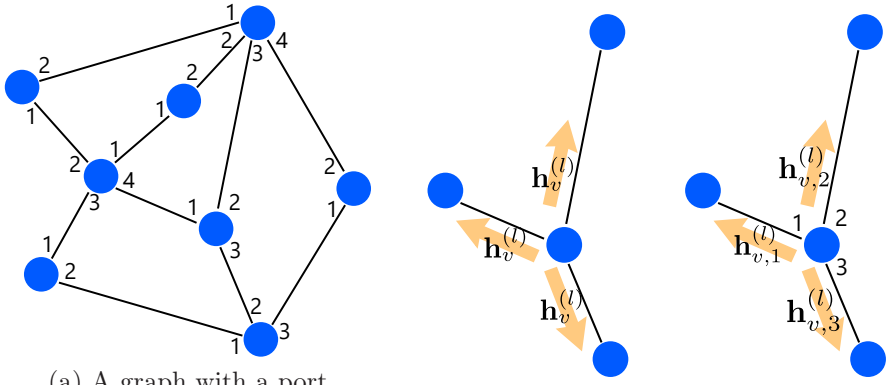
problem. Naor and Stockmeyer (1995) showed positive results for distributed local algorithms for the first time. For example, distributed local algorithms can find weak 2-coloring and solve a variant of the dining philosophers problem. Later, several non-trivial distributed local algorithms were found, including a 2-approximation algorithm for the minimum vertex cover problem (Åstrand et al., 2009). It should be noted that although classical 2-approximation algorithm of the minimum vertex cover problem is simple, it is not trivial how to compute a 2-approximation solution to the minimum vertex cover problem in a distributed way. An extensive survey on distributed local algorithms is provided by Suomela (2013).

There are many computational models of distributed algorithms. Among other computational models of distributed local algorithms, the standard computational model uses a port numbering. In this model, each node v has $\deg(v)$ ports, and each edge incident to v is connected to one of the ports. Only one edge can be connected to one port. In each communication round, each node sends a message to each port simultaneously. In general, different messages are sent to different ports. Then, each node receives messages simultaneously. Each node knows the port number that the neighboring node submits the message to, and the port number that the message comes from. Each node computes the next messages and next state based on these messages. After a constant number of rounds, each node outputs an answer (e.g., declares to be a hub) based on the states and received messages. This model is called the local model (Suomela, 2013), or the vector-vector consistent model ($\mathrm{VV_C}(1)$ model) for distinguishing with other computational models (Hella et al., 2012). Here, "(1)" means that this model stops after a constant number of communication rounds. Hella et al. (2012) considered weaker models than the $\mathrm{VV_C}(1)$ model. The multiset-broadcasting (MB(1)) model does not have a port numbering but sends the same message to all the neighboring nodes (i.e., broadcasting). The set-broadcasting (SB(1)) model does not have port numbering as the MB(1) model, and the SB(1) model receives messages as a set so that this model cannot count the number of repeating messages. Hella et al. (2012) showed that the $\mathrm{VV_C}(1)$ model can solve strictly wider problems than the MB(1) model, and the MB(1) model can solve strictly wider problems than the SB(1) model.

4.2 Connection with Local Algorithms

In this section, we introduce the connection between GNNs and distributed local algorithms, found by Sato et al. (2019a). First, Sato et al. (2019a) classified GNN models based on the computational models of distributed local algorithms.

MB-GNNs. MB-GNNs are standard message passing GNNs. They correspond



(a) A graph with a port numbering. (b) MBGNNs. (c) $\mathrm{VV_C}$-GNNs.

Figure 11: (a) An illustration of a port numbering. (b) MBGNNs send the same message to all the neighboring nodes. (c)$\mathrm{VV_C}$-GNNs send different messages to the neighboring nodes

to the MB(1) model.

$$\begin{aligned}
\boldsymbol{h}_v^{(0)} &= \boldsymbol{x}_v && (\forall v \in V),\\
\boldsymbol{a}_v^{(k)} &= f_{\mathrm{aggregate}}^{(k)}(\{\!\{\boldsymbol{h}_u^{(k-1)} \mid u \in \mathcal{N}(v)\}\!\}) && (\forall k \in [L], v \in V),\\
\boldsymbol{h}_v^{(k)} &= f_{\mathrm{update}}^{(k)}(\boldsymbol{h}_v^{(k-1)}, \boldsymbol{a}_v^{(k)}) && (\forall k \in [L], v \in V),
\end{aligned}$$

GraphSAGE-mean (Hamilton et al., 2017b), GCNs (Kipf and Welling, 2017), and GATs (Veličković et al., 2018) are examples of MB-GNNs. Although the messages of GATs are weighted by attentions, each node broadcasts the current embedding to all the neighboring nodes, and attention weights and weighted sum can be computed in each node. Thus, GATs are MB-GNNs.

SB-GNNs. SB-GNNs are a restricted class of GNNs that aggregate embeddings as a set. They corerspond to the SB(1) model.

$$\begin{aligned}
\boldsymbol{h}_v^{(0)} &= \boldsymbol{x}_v && (\forall v \in V),\\
\boldsymbol{a}_v^{(k)} &= f_{\mathrm{aggregate}}^{(k)}(\{\boldsymbol{h}_u^{(k-1)} \mid u \in \mathcal{N}(v)\}) && (\forall k \in [L], v \in V),\\
\boldsymbol{h}_v^{(k)} &= f_{\mathrm{update}}^{(k)}(\boldsymbol{h}_v^{(k-1)}, \boldsymbol{a}_v^{(k)}) && (\forall k \in [L], v \in V),
\end{aligned}$$

GraphSAGE-pool (Hamilton et al., 2017b) is an example of SB-GNNs. In the light of the taxonomy proposed by Hella et al. (2012), Sato et al. (2019a) proposed a class of GNNs that correspond to the $\mathrm{VV_C}(1)$ model.

$\mathrm{VV_C}$-GNNs. $\mathrm{VV_C}$-GNNs utilize a port numbering. $\mathrm{VV_C}$-GNNs first compute an arbitrary port numbering p, then compute embeddings of nodes by the

following formulae.

$$\boldsymbol{h}_v^{(0)} = \boldsymbol{x}_v \qquad (\forall v \in V),$$
$$\boldsymbol{a}_v^{(k)} = f_{\text{aggregate}}^{(k)}(\{(p(v,u), p(u,v), \boldsymbol{h}_u^{(k-1)}) \mid u \in \mathcal{N}(v)\}) \qquad (\forall k \in [L], v \in V),$$
$$\boldsymbol{h}_v^{(k)} = f_{\text{update}}^{(k)}(\boldsymbol{h}_v^{(k-1)}, \boldsymbol{a}_v^{(k)}) \qquad (\forall k \in [L], v \in V),$$

where $p(v,u)$ is the port number of v that the edge $\{v,u\}$ connects to. $\mathrm{VV_C}$-GNNs can send different messages to different neighboring nodes, while MB-GNNs always send the same message to all the neighboring nodes. Figure 11 illustrates MB-GNNs and $\mathrm{VV_C}$-GNNs. Sato et al. (2019a) showed that these classes of GNNs are as powerful as the corresponding classes of the computational models of local algorithms.

Theorem 13 (Sato et al. (2019a)). *Let $\mathcal{L}$ be MB, SB, or VV_C. For any algorithm $\mathcal{A}$ of the $\mathcal{L}(1)$ model, there exists a $\mathcal{L}$-GNN that the output is same as $\mathcal{A}$. For any $\mathcal{L}$-GNN $\mathcal{N}$, there exists an algorithm $\mathcal{A}$ of the $\mathcal{L}(1)$ model that the output is the same as the embedding computed by $\mathcal{N}$.*

Theorem 13 is easy to see because $f_{\text{aggregate}}^{(k)}$ is arbitrary, for example, $f_{\text{aggregate}}^{(k)}$ can be a function computed by a distributed local algorithm. Theorem 13 can be used to derive the hierarchy of GNNs in terms of expressive power because the expressive power of the computational models of distributed algorithms are known.

Theorem 14 (Hella et al. (2012)). *The class of the functions that the $VV_C(1)$ model can compute is strictly wider than the class of the functions that the MB(1) model can compute, and the class of the functions that the MB(1) model can compute is strictly wider than the class of the functions that the SB(1) model can compute.*

Corollary 15. *The class of the functions that the VV_C-GNNs model can compute is strictly wider than the class of the functions that the MB-GNNs model can compute, and the class of the functions that the MB-GNNs model can compute is strictly wider than the class of the functions that the SB-GNNs model can compute.*

It is already known that the MB-GNNs model can compute is strictly wider than the functional class that the SB-GNNs model can compute (Xu et al., 2019). This corollary provides another proof of this fact. Furthermore, this result indicates that GNNs can be more powerful by introducing a port numbering. Sato et al. (2019a) proposed a neural model called consistent port numbering GNNs (CPNGNNs).

CPNGNNs. CPNGNNs concatenate neighboring embeddings in the order of the port numbering.

$$\boldsymbol{h}_v^{(0)} = \boldsymbol{x}_v \qquad (\forall v \in V),$$
$$\boldsymbol{a}_v^{(k)} = \boldsymbol{W}^{(k)}[\boldsymbol{h}_v^{(k-1)\top}, \boldsymbol{h}_{u_{v,1}}^{(k-1)\top}, p(u_{v,1}v), \ldots, \boldsymbol{h}_{u_{v,\Delta}}^{(k-1)\top}, p(u_{v,\Delta}, v)]^\top \qquad (\forall k \in [L], v \in V),$$
$$\boldsymbol{h}_v^{(k)} = \mathrm{ReLU}(\boldsymbol{a}_v^{(k)}) \qquad (\forall k \in [L], v \in V),$$

where Δ is the maximum degree of input graphs, $u_{v,i}$ is the neighboring node of v that connects to the i-th port of v, and $\boldsymbol{W}^{(l)}$ are learnable parameters. CPNGNNs appropriately zero-padding if the degree of nodes are less than Δ. CPNGNNs are the most powerful among $\mathrm{VV_C}$-GNNs.

Theorem 16 (Sato et al. (2019a), Theorem 3)**.** *If the degrees of the nodes are bounded by a constant and the size of the support of node features is finite, for any VV_C-GNNs $\mathcal{N}$, there exist parameters of CPNGNNs such that for any (bounded degree) graph $G = (V, E, \boldsymbol{X})$, the embedding computed by the CPNGNN is arbitrary close to the embedding computed by $\mathcal{N}$.*

This theorem is strong because this says that there exist a fixed set of parameters that approximate any $\mathrm{VV_C}$-GNNs. This means that CPNGNNs can solve the same set of problems as the $\mathrm{VV_C}(1)$ model. Since the problems that the $\mathrm{VV_C}(1)$ model can/cannot solve are well studied in the distributed algorithm field, we can derive many properties about CPNGNNs. In particular, Sato et al. (2019a) showed the approximation ratios of algorithms that CPNGNNs can compute. They consider the following three problems.

Minimum Vertex Cover Problem

Input: A Graph $G = (V, E)$.

Output: A set of nodes $U \subseteq V$ of the minimum size that satisfies the following property. For any edge $\{u, v\} \in E$, u or v is in U.

Minimum Dominating Set Problem

Input: A Graph $G = (V, E)$.

Output: A set of nodes $U \subseteq V$ of the minimum size that satisfies the following property. For any node $v \in V$, v or at least one of the neighboring nodes of v is in V.

Maximum Matching Problem

Input: A Graph $G = (V, E)$.

Output: A set of edges $F \subseteq E$ of the maximum size that satisfies the following property. For any pair of edges $e, f \in F$, e and f does not share a node.

These three problems are well know combinatorial optimization problems (Cormen et al., 2009; Korte et al., 2012), and well studied in the distributed algorithm field. In the following discussions, the node feature vector of GNNs is a one-hot encoding of the degree of the node, and the initial state of the distributed algorithm only knows the degree of the node.

Lemma 17 (Lenzen and Wattenhofer (2008); Czygrinow et al. (2008); Åstrand et al. (2010))**.** *Let Δ be the maximum degree of input graphs. There exists an algorithm on the VV_C model(1) that computes a solution to the minimum dominating set problem with an approximation factor of $\Delta + 1$, but there does not exist*

an algorithm on the VV_C model(1) that computes a solution to the minimum dominating set problem with an approximation factor of less than $\Delta + 1$.

Theorem 18 (Sato et al. (2019a), Theorem 4)**.** *There exists a set of parameters of CPNGNNs that computes a solution to the minimum dominating set problem with an approximation factor of Δ+1, but there does not exist a set of parameters of CPNGNNs that computes a solution to the minimum dominating set problem with an approximation factor of less than $\Delta + 1$.*

Lemma 19 (Åstrand et al. (2009); Lenzen and Wattenhofer (2008); Czygrinow et al. (2008))**.** *There exists an algorithm on the VV_C model(1) that computes a solution to the minimum vertex cover problem with an approximation factor of 2, but there does not exist an algorithm on the VV_C model(1) that computes a solution to the minimum vertex cover problem with an approximation factor of less than 2.*

Theorem 20 (Sato et al. (2019a), Theorem 7)**.** *There exists a set of parameters of CPNGNNs that computes a solution to the minimum vertex cover problem with an approximation factor of 2, but there does not exist a set of parameters of CPNGNNs that computes a solution to the minimum vertex cover problem with an approximation factor of less than 2.*

Since the vertex cover problem cannot be approximated within an approximation factor of 2 under the unique games conjecture (Khot and Regev, 2008), CPNGNNs can compute an optimal algorithm in terms of an approximation ratio under the unique games conjecture.

Lemma 21 (Czygrinow et al. (2008); Åstrand et al. (2010))**.** *There does not exist an algorithm on the VV_C model(1) that computes a solution to the maximum matching problem with any constant approximation factor.*

Theorem 22 (Sato et al. (2019a), Theorem 8)**.** *There does not exist a set of parameters of CPNGNNs that computes a solution to the maximum matching problem with any constant approximation factor.*

Further, Sato et al. (2019a) noticed that adding features other than the degree feature improves the approximation ratio. In particular, they considered a weak 2-coloring. A weak 2-coloring is an assignment of 2 colors to the nodes of a graph such that for each node, there exists at least one neighboring node with the other color. Figure 12 illustrates a weak 2-coloring. A weak 2-coloring can be computed in linear time by the breadth-first search. Sato et al. (2019a) showed that adding am arbitrary weak 2-coloring to the node features enables GNN to know more about the input graph and to achieve a better ratio.

Lemma 23 (Åstrand et al. (2010))**.** *If the initial state of the distributed algorithm knows the degree of the node and the color of a weak coloring, weak 2-coloring, there exists an algorithm on the VV_C model(1) that computes a solution to the minimum dominating set problem with an approximation factor of $\frac{\Delta+1}{2}$, but there does not exist an algorithm on the VV_C model(1) that computes*

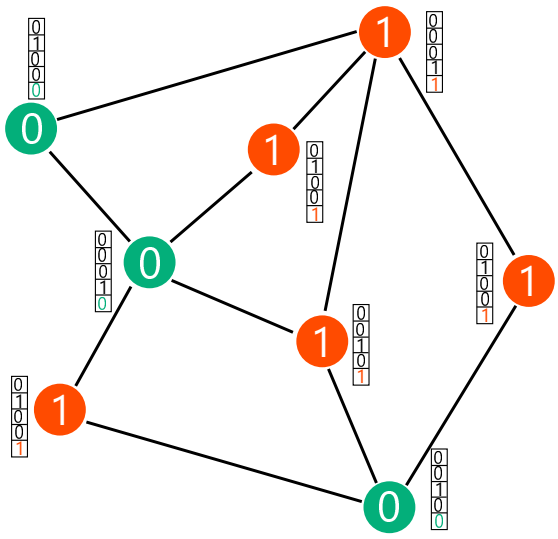


Figure 12: A weak 2-coloring. Each green (zero) node is adjacent to at least one red node, and each red (one) node is adjacent to at least one green node.

a solution to the minimum dominating set problem with an approximation factor of less than $\frac{\Delta+1}{2}$.

Theorem 24 (Sato et al. (2019a), Theorem 5)**.** *If the node feature is the degree of the node and the color of a weak coloring, weak 2-coloring, there exists an algorithm on the* VV_C *model(1) that computes a solution to the minimum dominating set problem with an approximation factor of* $\frac{\Delta+1}{2}$*, but there does not exist an algorithm on the* VV_C *model(1) that computes a solution to the minimum dominating set problem with an approximation factor of less than* $\frac{\Delta+1}{2}$.

Lemma 25 (Åstrand et al. (2010))**.** *If the initial state of the distributed algorithm knows the degree of the node and the color of a weak coloring, weak 2-coloring, there exists an algorithm on the* VV_C *model(1) that computes a solution to the maximum matching problem with an approximation factor of* $\frac{\Delta+1}{2}$*, but there does not exist an algorithm on the* VV_C *model(1) that computes a solution to the maximum matching problem with an approximation factor of less than* $\frac{\Delta+1}{2}$.

Theorem 26 (Sato et al. (2019a), Theorem 5)**.** *If the node feature is the degree of the node and the color of a weak coloring, weak 2-coloring, there exists an algorithm on the* VV_C *model(1) that computes a solution to the maximum matching problem with an approximation factor of* $\frac{\Delta+1}{2}$*, but there does not exist an algorithm on the* VV_C *model(1) that computes a solution to the maximum matching problem with an approximation factor of less than* $\frac{\Delta+1}{2}$.

Later, Garg et al. (2020) studied the expressive power of CPNGNNs more precisely. Garg et al. (2020) showed that CPNGNNs are more powerful than

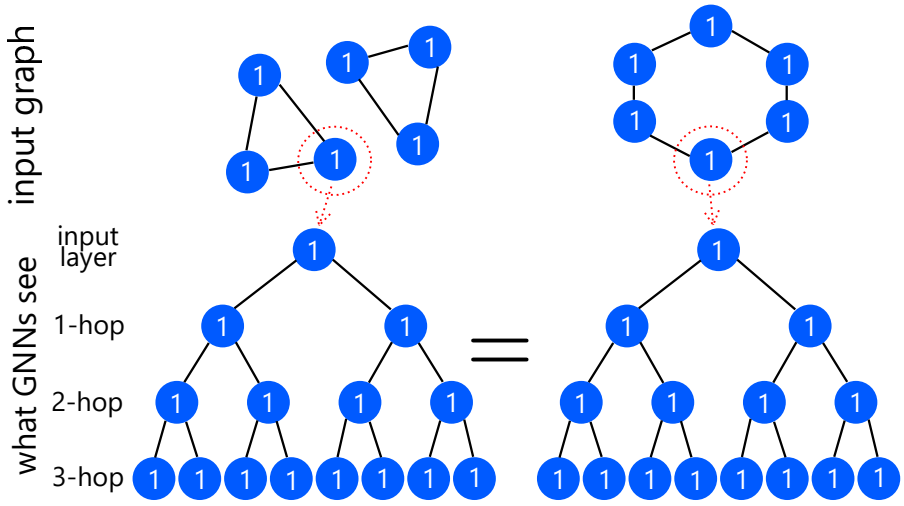
message passing GNNs if the port number is appropriate, but CPNGNNs fail to distinguish two triangles from one hexagon depending on port numberings. Loukas (2020) also pointed out the connection between the GNNs and local algorithms and characterized what GNNs cannot learn. In particular, he showed that message passing GNNs cannot solve many tasks even with powerful mechanisms unless the product of their depth and width depends polynomially on the number of nodes, and the same lower bounds also hold for strictly less powerful networks.

An exact polynomial time algorithm for the maximum matching (Edmonds, 1965) and an $O(\log \Delta)$ approximation algorithm for the minimum dominating set problem (Johnson, 1974; Lovász, 1975) are known. This indicates that the approximation ratios of the algorithms that CPNGNNs can compute are far from optimal. How can we improve these ratios? Sato et al. (2020) showed these ratios can be improved easily, just by adding random features to each node.

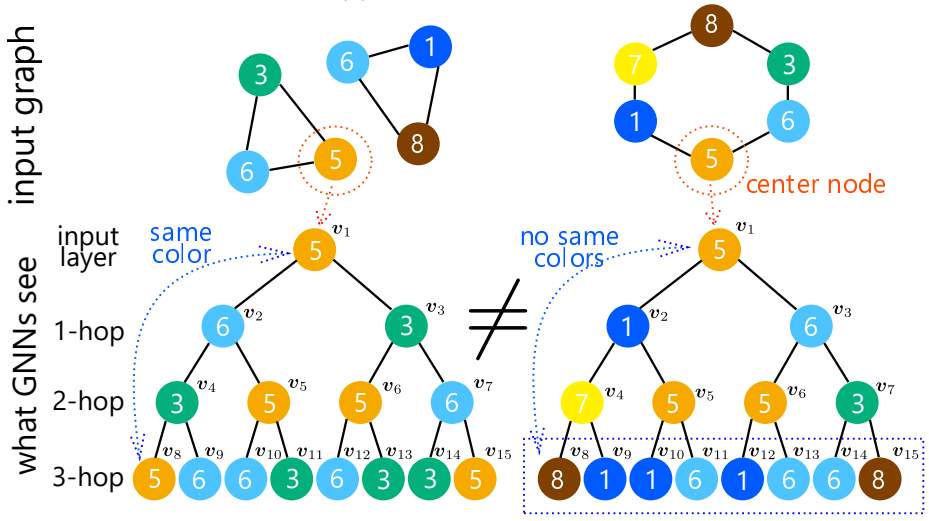
4.3 Random Features Strengthen GNNs

In this section, we introduce that adding random features to each node enables GNNs to distinguish a wider class of graphs and to model more efficient algorithms (Sato et al., 2020). They proposed GINs with random features (rGINs). rGINs assign a random feature every time the procedure called. Specifically, rGINs takes a graph $G = (V, E, \boldsymbol{X})$ as input, draw a random feature $\boldsymbol{r}_v \sim \mu$ from a discrete distribution μ for each node v in an i.i.d. manner, and compute embeddings of the nodes or the graph using the graphs $G = (V, E, \boldsymbol{X}')$ with random features, where $\boldsymbol{x}'_v = [\boldsymbol{x}_v, \boldsymbol{r}_v]$ and $\boldsymbol{X}' = [\boldsymbol{x}'_1, \boldsymbol{x}'_2, \ldots, \boldsymbol{x}'_n]$. Surprisingly, even though rGINs assign different random features in the test time from those in the training time, rGINs can generalize to unseen graphs in the test time.

Figure 13 provides an intuition. As Xu et al. (2019) showed, message passing GNNs cannot know the entire input graph, but what message passing GNNs can know is the breadth first search tree. As Figure 13 (a) shows, message passing GNNs cannot distinguish two triangles with one hexagon because the breadth first search trees of them are identical. In this example, we consider a simple model that concatenates all the embeddings in the breadth first search tree and 2-regular graphs for the sake of simplicity. Let the first dimension $\boldsymbol{v}_1$ of the node embedding $\boldsymbol{v}$ is the random feature of the center node. The second and third dimensions are the random features of the one-hop nodes (e.g., in the sorted order). The fourth to seventh dimensions are the random features of the two-hop nodes. The eighth to fifteenth dimensions are the random features of the three-hop nodes. Then, as Figure 13 (b) shows, irrespective of the random features, the center node is involved in a triangle if and only if there exists a leaf node of the same color as the center node unless the random features accidentally coincide. This condition can be formulated as $\boldsymbol{v}_1 = \boldsymbol{v}_8$ or $\boldsymbol{v}_1 = \boldsymbol{v}_9$ or $\ldots$ or $\boldsymbol{v}_1 = \boldsymbol{v}_{15}$. Therefore, we can check whether the center node is involved in a triangle by checking the embedding on the union of certain hyperplanes. This property is valid even if the random features are re-assigned; a center node is



(a) Identical Features.



(b) Random Features.

Figure 13: (a) Message passing GNNs cannot distinguish two triangles with one hexagon if node features are identical. (b) Message passing GNNs *can* distinguish two triangles with one hexagon with random node features.

involved in a triangle always falls on the union of these hyperplanes irrespective of the random features. A similar property is valid for substructures other than a triangle. Therefore, if the positive examples of a classification problem have characteristic substructures, the model can classify the nodes by checking the embedding on certain hyperplanes. This fact is formally stated in Theorem 28 It is noteworthy that the values of random features are not important; however, the relationship between the values is important because the values are random.

Furthermore, rGINs can handle arbitrary large graphs in the test time, especially larger graphs than training graphs, whereas relational pooling GNNs (Murphy et al., 2019b) cannot deal with larger graphs than the training graphs. Even if node index is provided as a scalar value, a neural network may behave badly when the model takes unseen node index as input. In contrast, rGINs can handle arbitrary large graphs because rGINs draw random features from the same distribution irrespective to the graph size. Sato et al. (2020) first showed that rGINs can distinguish any substructures. To state the theorem, we first define isomorphism with a center node. A pair of graphs with a center node is isomorphic if there exists an isomorphism that keeps the center node.

Definition 27 (Isomorphism with a center node)**.** Let $G = (V, E, \boldsymbol{X})$ and $G' = (V', E', \boldsymbol{X}')$ be graphs and $v \in V$ and $v' \in V'$ be nodes. (G, v) and (G', v') are isomorphic if there exists a bijection $f\colon V \to V'$ such that $(x, y) \in E \Leftrightarrow (f(x), f(y)) \in E'$, $\boldsymbol{x}_x = \boldsymbol{x}'_{f(x)}$ $(\forall x \in V)$, and $f(v) = f(v')$. $(G, v) \simeq (G', v')$ denotes (G, v) and (G', v') are isomorphic.

Theorem 28 (Sato et al. (2020), Theorem 1)**.** *$\forall L \in \mathbb{Z}^+, \Delta \in \mathcal{Z}^+$, for any finite feature space $\mathcal{C}$ ($|\mathcal{C}| < \infty$), for any set $\mathcal{G} = \{(G, v)\}$ of pairs of a graph $G = (V, E, \boldsymbol{X})$ and a center node $v \in V$ such that the maximum degree of G is at most Δ and $\boldsymbol{x}_u \in \mathcal{C}$ $(\forall u \in V)$, there exists $q \in \mathbb{R}^+$ such that for any discrete distribution μ with finite support X such that $\mu(x) \leq q$ $(\forall x \in X)$, there exists a set of parameters $\boldsymbol{\theta}$ such that for any pair of a graph $G = (V, E, \boldsymbol{X})$ and a center node $v \in V$ such that the maximum degree of G is at most Δ and $\boldsymbol{x}_u \in \mathcal{C}$ $(\forall u \in V)$*

- *if $\exists (G', v') \in \mathcal{G}$ such that $(G', v') \simeq (R(G, v, L), v)$ holds,* $\mathrm{rGIN}(G, \mu, \boldsymbol{\theta})_v > 0.5$ *holds with high probability.*
- *if $\forall (G', v') \in \mathcal{G}$, $(G', v') \not\simeq (R(G, v, L), v)$ holds,* $\mathrm{rGIN}(G, \mu, \boldsymbol{\theta})_v < 0.5$ *holds with high probability.*

For example, let $\mathcal{G}$ be the set of all pairs of a graph and a node v with at least one triangle incident to v. Then Theorem 28 shows that rGINs can classify the nodes by presence of the triangle structure, while message passing GNNs or CPNGNNs cannot determine the existence of a triangle in general (Maron et al., 2019a; Garg et al., 2020). Moreover, let $\mathcal{G}$ be a set of all graphs with certain chemical functional groups, then rGINs can classify atoms based on the functional groups that the atom belongs. Furthermore, rGINs maintain only n embeddings and run in a linear time with respect to the input size, whereas k-GNNs (Morris et al., 2019) maintain $O(n^k)$ embeddigns, k-th order GNNs

Table 2: The summary of approximation ratios of the minimum dominating set problem (MDS) and maximum matching problem (MM). * indicates that these ratios match the lower bounds. Δ denotes the maximum degree, $H(k)$ denotes the k-th harmonic number, $\varepsilon > 0$ is an arbitrary constant, and C is a fixed constant. The approximation ratios of rGINs match the best approximation ratios of polynomial algorithms except constant terms, and they also match the lower bounds except insignificant terms.

Problem	GINs / CPNGNNs	CPNGNNs + weak 2-coloring	rGINs	Polynomial Time	Lower Bound
MDS	$\Delta + 1^*$ Sato et al. (2019a)	$\frac{\Delta+1}{2}^*$ Sato et al. (2019a)	$H(\Delta+1)+\varepsilon$ Sato et al. (2020)	$H(\Delta+1)-\frac{1}{2}$ Duh and Fürer (1997)	$H(\Delta+1)-C\ln\ln\Delta$ Chlebík and Chlebíková (2008)
MM	∞^* Sato et al. (2019a)	$\frac{\Delta+1}{2}^*$ Sato et al. (2019a)	$1+\varepsilon^*$ Sato et al. (2020)	1^* Edmonds (1965)	1

(Maron et al., 2019b,c,a) maintain $O(n^k)$ embeddings, and (exact) relational pooling GNNs (Murphy et al., 2019b) run in $O(n!)$ time. This indicates that rGINs are efficient, keeping the expressive capability powerful. It should be noted that rGINs are similar to the sampling approximation of the relational pooling GNNs, where they assign random index to each node. The theorems of Sato et al. (2020) can be seen as the theoretical justification of the sampling approximation of the relational pooling GNNs. Then, Sato et al. (2020) showed the approximation ratios of rGINs.

Theorem 29 (Sato et al. (2020), Theorem 4)**.** *Let $H(k) = \frac{1}{1} + \frac{1}{2} + \cdots + \frac{1}{k}$ be the k-th harmonic number. For any $\varepsilon > 0$, there exists a set of parameters of rGINs that computes a solution to the minimum dominating set problem with an approximation factor of $H(\Delta+1)+\varepsilon$ with high probability.*

Theorem 30 (Sato et al. (2020), Theorem 6)**.** *For any $\varepsilon > 0$, there exists a set of parameters of rGINs that computes a solution to the maximum matching problem with an approximation factor of $1+\varepsilon$ with high probability.*

Table 2 summarizes the approximation ratios. This tables shows that rGINs can compute much better algorithms than GINs and CPNGNNs, and rGINs can compute almost optimal algorithms in terms of approximation ratios.

4.4 Time Complexity

In this section, we consider the problems that GNNs can/cannot solve via the lens of time complexity. In section 3, we considered the graph isomorphism problem, and we saw that message passing GNNs are as powerful as the 1-WL algorithm and cannot solve the graph isomorphism problem. Although the graph isomorphism problem can be solved in quasi-polynomial time (Babai, 2016), the graph isomorphism problem is not known in co-NP, and no polynomial time algorithm of the graph isomorphism problem is known (Babai, 2016). Therefore, from a structural complexity point of view, it is difficult to construct

universal GNNs that run in polynomial time. In Section 4, we saw the combinatorial algorithms that GNNs can compute. However, the minimum vertex cover problem and the minimum dominating set problem are both NP-hard problem, which indicate that these problems are not solvable in polynomial time under the P $\neq$ NP hypothesis. Therefore, although CPNGNNs and rGINs can approximate the minimum vertex cover problem within a factor of 2 and the minimum dominating set problem within a factor of $H(\Delta + 1) + \varepsilon$, it is impossible to construct efficient GNNs that solve these problems exactly under the P $\neq$ NP hypothesis.

Another useful tool to analyze the hardness of problems for GNNs is fine-grained complexity (Williams, 2005, 2018). Fine-grained complexity shows that some problem is not solvable in $O(n^{c-\varepsilon})$ time under some hypothesis, just like NP-hard problems are shown to be not solvable efficiently under the P $\neq$ NP hypothesis. For example, Roditty and Williams (2013) showed that the diameter of undirected unweighted sparse graphs cannot be determined in $O(n^{2-\varepsilon})$ time under the strongly exponential time hypothesis (SETH). Williams and Williams (2010) showed that the existence of a negative triangle of weighted graphs cannot be determined in $O(n^{3-\varepsilon})$ time under the all pairs shortest path (APSP) hypothesis. These results indicate that it is impossible to construct GNNs that determine these properties in linear time under these hypotheses.

Sato et al. (2019b) studied the time complexity of GNNs. They showed that many message passing GNNs, including GraphSAGE-mean (Hamilton et al., 2017b), GCNs (Kipf and Welling, 2017), and GATs (Veličković et al., 2018), can be approximated in constant time, whereas it is impossible to approximate GraphSAGE-pool in constant time by any algorithm. This reveals graph problems that these GNNs cannot solve via the lens of time complexity. For example, let a node of a graph with node features 0 or 1 be positive if there exists at least one neighboring node with feature 1, and negative otherwise. This problem is not solvable in sublinear time because a star graph with no "1" nodes and a star graph with only one "1" leaf node are counterexamples. Therefore, GraphSAGE-mean, GCNs, and GATs cannot solve this problem. In contrast, GraphSAGE-pool (Hamilton et al., 2017b) can solve this problem owing to the pooling operator. In general, the time complexity of a model and the class of the problems that the model can solve is in a trade-off relation.

5 XS Correspondence

As we saw in Section 3 and 4, GNNs are closely related to the WL algorithm and distributed local algorithms. In this section, we summarize the results of the expressive power of GNNs in the light of relations among GNNs, the WL algorithm, and distributed local algorithms. We call their relations the XS correspondence, named after Xu et al. (2019) and Sato et al. (2019a). The observations of Xu et al. (2019) and Sato et al. (2019a) provide concrete correspondences between elements of GNNs, the WL algorithm, and distributed local algorithms. Table 3 summarizes these correspondences. For example, the num-

Table 3: The XS correspondence provides concrete correspondences between elements of GNNs, the WL algorithm, and distributed local algorithms.

GNNs	WL algorithm	Local algorithms
graph	graph	computer network
node	node	computer
edge	edge	interconnection
feature/embedding	color	state of algorithm
parameters	hash function	algorithm
layer	refinement round	communication round
readout	readout	-
port	-	port

ber of communication round needed to solve combinatorial problems are studied in the distributed algorithm field. Åstrand et al. (2009) showed that $(\Delta+1)^2$ rounds are sufficient for a distributed 2-approximation algorithm, where Δ is the maximum degree. Babai et al. (1980) showed that sufficiently large random graphs can be determined by the 1-WL algorithm within 2 rounds with high probability. These results can be used to design the number of layers of GNNs owing to the XS correspondence. In particular, the result of Babai et al. (1980) can be a justification of two-layered GNNs.

In addition, it is known that the WL algorithm and distributed local algorithms have connections with many other fields. For example, the k-WL algorithm is known to have connections with the first-order logic with counting quantifiers (Immerman and Lander, 1990; Cai et al., 1992), pebbling games (Immerman and Lander, 1990; Grohe and Otto, 2015a), and linear programming (Tinhofer, 1991; Ramana et al., 1994) and the Sherali–Adams relaxation (Atserias and Maneva, 2013; Malkin, 2014; Grohe and Otto, 2015a). Distributed local algorithms have connections to modal logic (Hella et al., 2012) and constant time algorithms (Parnas and Ron, 2007). Specifically,

- For every $k \geq 2$, there exists a k-variable first-order logic sentence φ with counting quantifiers such that $G \models \varphi$ and $H \not\models \varphi$ if and only if the k-WL algorithm outputs that G and H are "non-isomorphic" (Immerman and Lander, 1990; Cai et al., 1992).
- Player II has a winning strategy for the C_k game on G and H if and only if the k-WL algorithm outputs that G and H are "possibly isomorphic" (Immerman and Lander, 1990; Cai et al., 1992).
- Let $\boldsymbol{A}$ and $\boldsymbol{B}$ be the adjacency matrices of G and H. There exists a doubly-stochastic real matrix $\boldsymbol{X}$ such that $\boldsymbol{AX} = \boldsymbol{XB}$ if and only if the 1-WL algorithm outputs G and H are "possibly isomorphic" (Tinhofer, 1991; Ramana et al., 1994).

- Let $\boldsymbol{A}$ and $\boldsymbol{B}$ be the adjacency matrices of G and H. For every $k \geq 2$, there exists a solution to the rank-k Sherali-Adams relaxation of $\boldsymbol{AX} = \boldsymbol{XB}$ such that $\boldsymbol{X}$ is doubly-stochastic if the $(k+2)$-WL algorithm outputs G and H are "possibly isomorphic" (Atserias and Maneva, 2013; Malkin, 2014; Grohe and Otto, 2015a).

- Let $\boldsymbol{A}$ and $\boldsymbol{B}$ be the adjacency matrices of G and H. For every $k \geq 2$, there exists a solution to the rank-k Sherali-Adams relaxation of $\boldsymbol{AX} = \boldsymbol{XB}$ such that $\boldsymbol{X}$ is doubly-stochastic only if the $(k+1)$-WL algorithm outputs G and H are "possibly isomorphic" (Atserias and Maneva, 2013; Malkin, 2014; Grohe and Otto, 2015a).

- The $\mathrm{VV_C}(1)$ model can recognize logic formulas of graded multimodal logic on the corresponding Kripke model, and graded multimodal logic can simulate any algorithm on the $\mathrm{VV_C}(1)$ model (Hella et al., 2012).

- A distributed local algorithm can be converted to a constant time algorithm (Parnas and Ron, 2007).

Thanks to the XS correspondence, many theoretical properties of GNNs can be derived using the results in these fields. For example, Barceló et al. (2020) utilized the relationship between the WL algorithm and the first-order logic to build more powerful GNNs.

6 Conclusion

In this survey, we introduced the expressive capability of graph neural networks. Namely, we introduced that message passing GNNs are at most as powerful as the one dimensional WL algorithm, and how to generalize GNNs to the k dimensional WL algorithm. We then introduced the connection between GNNs and distributed algorithms, and showed the limitations of GNNs in terms of approximation ratios of combinatorial algorithms that GNNs can compute. We then showed that adding random features to each node improves approximation ratios drastically. Finally, we summarized the relationships among GNNs, the WL algorithm, and distributed local algorithms as the XS correspondence.

References

Dana Angluin. Local and global properties in networks of processors (extended abstract). In *Proceedings of the 12th Annual ACM Symposium on Theory of Computing, STOC*, pages 82–93, 1980.

Vikraman Arvind, Johannes Köbler, Gaurav Rattan, and Oleg Verbitsky. On the power of color refinement. In *Fundamentals of Computation Theory - 20th International Symposium, FCT*, pages 339–350, 2015.

Matti Åstrand, Patrik Floréen, Valentin Polishchuk, Joel Rybicki, Jukka Suomela, and Jara Uitto. A local 2-approximation algorithm for the vertex cover problem. In *Proceedings of 23rd International Symposium on Distributed Computing, DISC*, pages 191–205, 2009.

Matti Åstrand, Valentin Polishchuk, Joel Rybicki, Jukka Suomela, and Jara Uitto. Local algorithms in (weakly) coloured graphs. *arXiv preprint*, 2010.

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

László Babai. Graph isomorphism in quasipolynomial time [extended abstract]. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC*, pages 684–697, 2016.

László Babai, Paul Erdös, and Stanley M. Selkow. Random graph isomorphism. *SIAM J. Comput.*, 9(3):628–635, 1980.

Pablo Barceló, Egor V. Kostylev, Mikaël Monet, Jorge Pérez, Juan L. Reutter, and Juan Pablo Silva. The logical expressiveness of graph neural networks. In *8th International Conference on Learning Representations, ICLR*, 2020.

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

Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. In *2nd International Conference on Learning Representations, ICLR*, 2014.

Jin-yi Cai, Martin Fürer, and Neil Immerman. An optimal lower bound on the number of variables for graph identifications. *Combinatorica*, 12(4):389–410, 1992.

Zhengdao Chen, Soledad Villar, Lei Chen, and Joan Bruna. On the equivalence between graph isomorphism testing and function approximation with gnns. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS*, 2019.

Zhengdao Chen, Lei Chen, Soledad Villar, and Joan Bruna. Can graph neural networks count substructures? *arXiv preprint*, 2020.

Miroslav Chlebík and Janka Chlebíková. Approximation hardness of dominating set problems in bounded degree graphs. *Inf. Comput.*, 206(11):1264–1275, 2008.

Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2009.

George Cybenko. Approximation by superpositions of a sigmoidal function. *MCSS*, 2(4):303–314, 1989.

Andrzej Czygrinow, Michal Hanckowiak, and Wojciech Wawrzyniak. Fast distributed approximations in planar graphs. In *Proceedings of 22nd International Symposium on Distributed Computing, DISC*, pages 78–92, 2008.

Hanjun Dai, Bo Dai, and Le Song. Discriminative embeddings of latent variable models for structured data. In *Proceedings of the 33nd International Conference on Machine Learning, ICML*, pages 2702–2711, 2016.

Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems 29, NIPS*, pages 3837–3845, 2016.

Rong-chii Duh and Martin Fürer. Approximation of k-set cover by semi-local optimization. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing, STOC*, pages 256–264, 1997.

David Duvenaud, Dougal Maclaurin, Jorge Aguilera-Iparraguirre, Rafael Gómez-Bombarelli, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P. Adams. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, NIPS*, pages 2224–2232, 2015.

Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of mathematics*, 17:449–467, 1965.

Wenqi Fan, Yao Ma, Qing Li, Yuan He, Yihong Eric Zhao, Jiliang Tang, and Dawei Yin. Graph neural networks for social recommendation. In *The World Wide Web Conference, WWW*, pages 417–426, 2019.

Martin Fürer. On the combinatorial power of the weisfeiler-lehman algorithm. In *Algorithms and Complexity - 10th International Conference, CIAC*, pages 260–271, 2017.

Vikas K Garg, Stefanie Jegelka, and Tommi Jaakkola. Generalization and representational limits of graph neural networks. *arXiv preprint*, 2020.

Maxime Gasse, Didier Chételat, Nicola Ferroni, Laurent Charlin, and Andrea Lodi. Exact combinatorial optimization with graph convolutional neural networks. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS*, pages 15554–15566, 2019.

Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning, ICML*, pages 1263–1272, 2017.

Yu Gong, Yu Zhu, Lu Duan, Qingwen Liu, Ziyu Guan, Fei Sun, Wenwu Ou, and Kenny Q. Zhu. Exact-k recommendation via maximal clique optimization. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD*, pages 617–626, 2019.

Marco Gori, Gabriele Monfardini, and Franco Scarselli. A new model for learning in graph domains. In *Proceedings of the International Joint Conference on Neural Networks, IJCNN*, volume 2, pages 729–734, 2005.

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

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

Martin Grohe and Martin Otto. Pebble games and linear equations. *J. Symb. Log.*, 80(3):797–844, 2015b.

William L. Hamilton, Rex Ying, and Jure Leskovec. Representation learning on graphs: Methods and applications. *IEEE Data Eng. Bull.*, 40(3):52–74, 2017a.

William L. Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems 30, NIPS*, pages 1025–1035, 2017b.

Jason S. Hartford, Devon R. Graham, Kevin Leyton-Brown, and Siamak Ravanbakhsh. Deep models of interactions across sets. In *Proceedings of the 35th International Conference on Machine Learning, ICML*, pages 1914–1923, 2018.

Lauri Hella, Matti Järvisalo, Antti Kuusisto, Juhana Laurinharju, Tuomo Lempiäinen, Kerkko Luosto, Jukka Suomela, and Jonni Virtema. Weak models of distributed computing, with connections to modal logic. In *Proceedings of the ACM Symposium on Principles of Distributed Computing, PODC*, pages 185–194, 2012.

Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251–257, 1991.

Kurt Hornik, Maxwell B. Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

Neil Immerman and Eric Lander. Describing graphs: A first-order approach to graph canonization. In *Complexity theory retrospective*, pages 59–81. Springer, 1990.

David S. Johnson. Approximation algorithms for combinatorial problems. *J. Comput. Syst. Sci.*, 9(3):256–278, 1974.

Nicolas Keriven and Gabriel Peyré. Universal invariant and equivariant graph neural networks. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS*, 2019.

Elias B. Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, NIPS*, pages 6348–6358, 2017.

Subhash Khot and Oded Regev. Vertex cover might be hard to approximate to within 2-epsilon. *J. Comput. Syst. Sci.*, 74(3):335–349, 2008.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *Proceedings of the Fifth International Conference on Learning Representations, ICLR*, 2017.

Boris Knyazev, Graham W. Taylor, and Mohamed R. Amer. Understanding attention and generalization in graph neural networks. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS*, pages 4204–4214, 2019.

Risi Kondor, Hy Truong Son, Horace Pan, Brandon Anderson, and Shubhendu Trivedi. Covariant compositional networks for learning graphs. *arXiv preprint*, 2018.

Bernhard Korte, Jens Vygen, B Korte, and J Vygen. *Combinatorial optimization*, volume 2. Springer, 2012.

Christoph Lenzen and Roger Wattenhofer. Leveraging linial's locality limit. In *Proceedings of 22nd International Symposium on Distributed Computing, DISC*, pages 394–407, 2008.

Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard S. Zemel. Gated graph sequence neural networks. In *4th International Conference on Learning Representations, ICLR*, 2016.

Zhuwen Li, Qifeng Chen, and Vladlen Koltun. Combinatorial optimization with graph convolutional networks and guided tree search. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS*, pages 537–546, 2018.

Nathan Linial. Locality in distributed graph algorithms. *SIAM J. Comput.*, 21 (1):193–201, 1992.

Andreas Loukas. What graph neural networks cannot learn: depth vs width. In *8th International Conference on Learning Representations, ICLR*, 2020.

László Lovász. On the ratio of optimal integral and fractional covers. *Discrete Mathematics*, 13(4):383–390, 1975.

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

Haggai Maron, Heli Ben-Hamu, Hadar Serviansky, and Yaron Lipman. Provably powerful graph networks. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS*, 2019a.

Haggai Maron, Heli Ben-Hamu, Nadav Shamir, and Yaron Lipman. Invariant and equivariant graph networks. In *7th International Conference on Learning Representations, ICLR*, 2019b.

Haggai Maron, Ethan Fetaya, Nimrod Segol, and Yaron Lipman. On the universality of invariant networks. In *Proceedings of the 36th International Conference on Machine Learning, ICML*, pages 4363–4371, 2019c.

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

Ron Milo, Shai Shen-Orr, Shalev Itzkovitz, Nadav Kashtan, Dmitri Chklovskii, and Uri Alon. Network motifs: simple building blocks of complex networks. *Science*, 298(5594):824–827, 2002.

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 *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI*, pages 4602–4609, 2019.

Ryan L. Murphy, Balasubramaniam Srinivasan, Vinayak A. Rao, and Bruno Ribeiro. Janossy pooling: Learning deep permutation-invariant functions for variable-size inputs. In *7th International Conference on Learning Representations, ICLR*, 2019a.

Ryan L. Murphy, Balasubramaniam Srinivasan, Vinayak A. Rao, and Bruno Ribeiro. Relational pooling for graph representations. In *Proceedings of the 36th International Conference on Machine Learning, ICML*, pages 4663–4673, 2019b.

Moni Naor and Larry J. Stockmeyer. What can be computed locally? *SIAM J. Comput.*, 24(6):1259–1277, 1995.

Mark E. J. Newman. The structure and function of complex networks. *SIAM Review*, 45(2):167–256, 2003.

Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning convolutional neural networks for graphs. In *Proceedings of the 33nd International Conference on Machine Learning, ICML*, pages 2014–2023, 2016.

Leo A Paquette, Robert J Ternansky, Douglas W Balogh, and Gary Kentgen. Total synthesis of dodecahedrane. *Journal of the American Chemical Society*, 105(16):5446–5450, 1983.

Namyong Park, Andrey Kan, Xin Luna Dong, Tong Zhao, and Christos Faloutsos. Estimating node importance in knowledge graphs using graph neural networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD*, pages 596–606, 2019.

Michal Parnas and Dana Ron. Approximating the minimum vertex cover in sublinear time and a connection to distributed algorithms. *Theor. Comput. Sci.*, 381(1-3):183–196, 2007.

Motakuri V. Ramana, Edward R. Scheinerman, and Daniel Ullman. Fractional isomorphism of graphs. *Discret. Math.*, 132(1-3):247–265, 1994.

Liam Roditty and Virginia Vassilevska Williams. Fast approximation algorithms for the diameter and radius of sparse graphs. In *45th Symposium on Theory of Computing Conference, STOC*, pages 515–524, 2013.

Ryoma Sato, Makoto Yamada, and Hisashi Kashima. Approximation ratios of graph neural networks for combinatorial problems. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS*, 2019a.

Ryoma Sato, Makoto Yamada, and Hisashi Kashima. Constant time graph neural networks. *arXiv preprint*, 2019b.

Ryoma Sato, Makoto Yamada, and Hisashi Kashima. Random features strengthen graph neural networks. *arXiv preprint*, 2020.

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Trans. Neural Networks*, 20(1):61–80, 2009.

Michael Sejr Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *The Semantic Web - 15th International Conference, ESWC*, pages 593–607, 2018.

Harry P Schultz. Topological organic chemistry. polyhedranes and prismanes. *The Journal of Organic Chemistry*, 30(5):1361–1364, 1965.

Le Song, Arthur Gretton, and Carlos Guestrin. Nonparametric tree graphical models. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, AISTATS*, pages 765–772, 2010.

Le Song, Arthur Gretton, Danny Bickson, Yucheng Low, and Carlos Guestrin. Kernel belief propagation. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, AISTATS*, pages 707–715, 2011.

Alessandro Sperduti and Antonina Starita. Supervised neural networks for the classification of structures. *IEEE Trans. Neural Networks*, 8(3):714–735, 1997.

Jukka Suomela. Survey of local algorithms. *ACM Comput. Surv.*, 45(2):24:1–24:40, 2013.

Gottfried Tinhofer. A note on compact graphs. *Discrete Applied Mathematics*, 30(2-3):253–264, 1991.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *Proceedings of the Sixth International Conference on Learning Representations, ICLR*, 2018.

Hongwei Wang, Miao Zhao, Xing Xie, Wenjie Li, and Minyi Guo. Knowledge graph convolutional networks for recommender systems. In *The World Wide Web Conference, WWW*, pages 3307–3313, 2019a.

Xiang Wang, Xiangnan He, Yixin Cao, Meng Liu, and Tat-Seng Chua. KGAT: knowledge graph attention network for recommendation. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD*, pages 950–958, 2019b.

Boris Weisfeiler and Andrei A Lehman. A reduction of a graph to a canonical form and an algebra arising during this reduction. *Nauchno-Technicheskaya Informatsia*, 2(9):12–16, 1968.

Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theor. Comput. Sci.*, 348(2-3):357–365, 2005.

Virginia Vassilevska Williams. On some fine-grained questions in algorithms and complexity. In *Proceedings of the ICM*, volume 3, pages 3431–3472. World Scientific, 2018.

Virginia Vassilevska Williams and Ryan Williams. Subcubic equivalences between path, matrix and triangle problems. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS*, pages 645–654, 2010.

Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *arXiv preprint*, 2019.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *7th International Conference on Learning Representations, ICLR*, 2019.

Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD*, pages 974–983, 2018.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabás Póczos, Ruslan Salakhutdinov, and Alexander J. Smola. Deep sets. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, NIPS*, pages 3391–3401, 2017.

Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. An end-to-end deep learning architecture for graph classification. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, AAAI*, pages 4438–4445, 2018.

Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, and Maosong Sun. Graph neural networks: A review of methods and applications. *arXiv preprint*, 2018.