Redundancy-Free Message Passing for Graph Neural Networks

Anonymous Author(s)
Affiliation
Address
email

Abstract

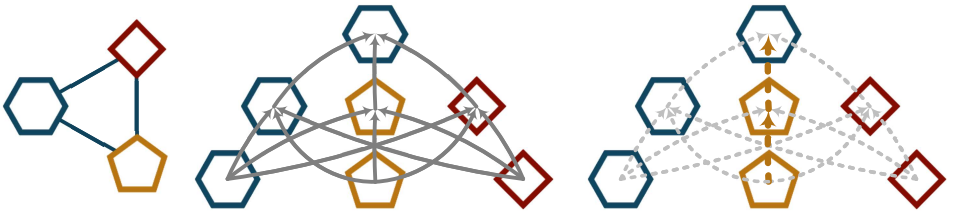
Graph Neural Networks (GNNs) resemble the Weisfeiler-Lehman (1-WL) test, which iteratively update the representation of each node by aggregating information from WL-tree. However, despite the computational superiority of the iterative aggregation scheme, it introduces redundant message flows to encode nodes. We found that the redundancy in message passing prevented conventional GNNs from propagating the information of long-length paths and learning graph similarities. In order to address this issue, we proposed Redundancy-Free Graph Neural Network (RFGNN), in which the information of each path (of limited length) in the original graph is propagated along a single message flow. Our rigorous theoretical analysis demonstrates the following advantages of RFGNN: (1) RFGNN is strictly more powerful than 1-WL; (2) RFGNN efficiently propagate structural information in original graphs, avoiding the over-squashing issue; and (3) RFGNN could capture subgraphs at multiple levels of granularity, and are more likely to encode graphs with closer graph edit distances into more similar representations. The experimental evaluation of graph-level prediction benchmarks confirmed our theoretical assertions, and the performance of the RFGNN can achieve the best results in most datasets.

1 Introduction

Graph Neural Networks (GNNs) generally imitate the Weisfeiler-Lehman (1-WL) test [1] in the sense that they iteratively update the representation of each node by aggregating information from its neighbors, which are referred to WL-test based GNNs in this study. The iterative neighborhood aggregation of a node encodes a subtree (called WL-subtree [2]) rooted at the node, where each subtree is a height-balanced tree constructed by expanding the neighbors recursively. Although the studies of GNNs have taken great strides and achieved remarkable success in recent years, the vast majority do not break the paradigm of repetitive neighborhood aggregation, which grants computational superiority but the tree like encoding generates some negative consequences due to message flow redundancy. Figure 1 illustrates an example of message flow redundancy in a computational graph. Redundancy of message flows is prevalent especially when the number of messaging passing iterations (a.k.a. neural network layers) increases.

The redundancy phenomenon is accompanied by the problem of *over-squashing*, which refers to the phenomenon in which the information originating from distant nodes can be distorted, thus limiting the efficiency of message passing for tasks that rely on long-distance interactions [3]. This issue is attributable to the bottleneck of compressing a rapidly growing number of neighboring nodes into a vector of finite length [4] and the failure of certain input features to influence the output representation [3]. However, in previous studies, the understanding of over-squashing and the explanation for why it occurs are still deficient. In this study, we carefully analyze the influence

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.



(a) Input Graph (b) 2-Layer GNN Computational Graph (c) A message flow that transmits redundant information (i.e., the orange node)

Figure 1: Redundancy exists in message flows, which are propagation chains in a GNN computational graph that deliver the information from input nodes to output nodes.

of an input feature on an output representation along a *trail* (extracted from the original graph) and we observe that longer trails typically have smaller impact than shorter trails in the message passing paradigm. One key reason behind is that short trails usually results in a great number of redundant message flows in the computational graph.

According to previous research [5, 6, 7, 8], a fundamental problem of graph learning is to quantify the similarities between graphs. In the real world, many graphs such as molecules have structures at coarse levels (such as chains, rings and trees) and fine levels (like chemical bonds and functional groups). Some scholars [9] believe that an ideal method of graph similarity measurement should be able to capture subgraphs at different levels of granularities. However, WL-subtree [2] (so as WL-test based GNNs) can only capture the graph similarity at coarse levels because WL-subtrees are height-balanced trees. Besides, WL-test based GNNs tend to encode two graphs that yield similar WL subtrees as similar representations. However, even if only one edge in the two original graphs is different, this can lead to two very different WL subtrees due to the redundant message flows. Thus, WL-test based GNNs may have insufficient expressive power to measure the graph similarity.

We propose Redundancy-Free Graph Neural Network (RFGNN) to address the aforementioned issues. We first eliminate the message flow redundancy by a concept called *path search tree*, which is constructed by Breadth-First Search (BFS). Since there are no repeated edges in each path in the path search tree, by using the tree as the message passing graph, the resulting model would not suffer from redundant message flow. However, this simple path search tree model cannot capture some graph structures, e.g., ring. To enhance the expressive power of the model, we define a new concept called extended path (epath), which allows the starting and ending nodes are identical so that epath is able to represent a ring structure. Based on the extended path, we propose a concept called Truncated ePaths Tree (TPT), a tree composed of all possible epaths with the same starting node. Given a TPT (of a graph node), our RFGNN is a neural network to perform bottom-up message passing, which encodes any subtree of the TPT by deactivating messages not belonging to the subtree. Finally, we collect the representations of all TPTs in order to update the representation of the entire graph, thereby encoding any subtree of the original graph into the representation of the graph.

In comparison to prior research in the field of GNN, the leading technologies and theories of this paper were summarized as follows, 1) To the best of our knowledge, it is the first theoretical examination of the redundancy in GNNs and its practical implications. 2) We show that the expressive powers of RFGNN, that is trained with injective aggregators and sufficient layers, can surpass the conventional GNNs. 3) De-redundancy balances the influence of epaths starting at the same node during message passing, propagates structural information in original graphs efficiently, and reduces over-squashing. 4) RFGNN is capable of encoding any subtree of the original graph, allowing them to compare the similarities between graphs at various levels of granularity.

The rest of this paper is structured as follows. Section 2 outlines the basic theories of GNN. In section 3, we demonstrate that redundancy is prevalent in GNNs and analyze why it contributes to the issue of over-squashing in GNNs. The framework of RFGNN is presented in section 4, which not only eliminates the redundancy in computational graphs, but also provides several additional advantages. Some related works are given in Section 5. Section 6 empirically validate RFGNN on a variety of

graph classification benchmarks, and demonstrate that RFGNN achieves state-of-the-art performance on the vast majority of datasets. The conclusion is given in Section 7.

2 Preliminaries

We begin by reviewing the most used message passing paradigm and its derivative GNNs, as well as introducing our notations. Let $G(V, E) \in \mathcal{G}$ be a graph, where $V = \mathcal{V}(G) = \{1, 2, \ldots\}$, $E = \mathcal{E}(G) \subseteq V \times V$, and $\mathcal{G}$ is the graph set consisted of all possible graphs. The nodes and edges in G can have feature vectors associated with them, denoted by $\mathbf{x}_u$ for $u \in V$ and $\mathbf{e}_{u,v}$ for $(u, v) \in E$. We consider the graph classification problem, where, given a set of graphs $\{G_1, \ldots, G_N\} \subseteq \mathcal{G}$ and their classes $\{y_1, \ldots, y_N\} \subseteq \mathcal{Y}$, we aim to learn a representation vector $\mathbf{h}_G$ that helps to predict the label of an entire graph by a linear classifier, $y_G = \rho(\mathbf{h}_G)$.

Message passing and graph neural networks. GNNs iteratively update the representation of the nodes by aggregating neighbor information. The l-th layer of GNN can be described as

$$\mathbf{h}_u^{(l)} = \psi^{(l)}\left(\mathbf{h}_u^{(l-1)}, \cup_{v \in \mathcal{N}(u)} \phi^{(l)}\left(\mathbf{e}_{(u,v)}^{(l-1)}, \mathbf{h}_v^{(l-1)}\right)\right), \tag{1}$$

where $\psi^{(l)}$ and $\phi^{(l)}$ denote differentiable functions such as MLPs (Multi Layer Perceptrons), $\cup$ denotes a differentiable, permutation invariant aggregator, (e.g., *sum* and *mean*), $\mathbf{h}_u^{(l)}$ is the node u representation, and $\mathbf{e}_{(u,v)}^{(l-1)}$ is the edge (u, v) representation transformed from $\mathbf{e}_{(u,v)}$ by a differentiable function such as a linear function. We initialize $\mathbf{h}_u^{(0)} = \mathbf{x}_u$. The readout χ function aggregates node representations from the final layer to obtain the entire graph representation $\mathbf{h}_G$:

$$\mathbf{h}_G = \chi\left(\left\{\mathbf{h}_u^{(k)} \mid u \in G\right\}\right). \tag{2}$$

The readout χ can be a simple permutation invariant function such as *sum*.

Weisfeiler-Lehman test and WL-subtree. The graph isomorphism problem is to identify whether two graphs are topologically equivalent, i.e., isomorphic. The Weisfeiler-Lehman (1-WL) test [1] iteratively updates the label of a node by hashing the set of its own and neighborhood labels. Graphs are deemed as non-isomorphic by 1-WL if their node labels differ at any iterations. Based on the 1-WL, Shervashidze et al. [2] propose the WL-subtree kernel, which measures the similarity of graphs based on the number of distinct node labels. A node label represents a subtree (WL-subtree), rooted at the node, deconstructed from its neighborhood. That means the WL-subtree kernel compares graph similarity by the number of distinct WL-subtrees.

Computational graph. Computational graphs represent neural networks as graphs on which gradients can be conveniently computed using chaining laws [10]. In message passing GNNs, the computational graph of a node representation is the message passing graph that aggregates neighborhood information into the node. The computational graph is jointly determined by the adopted message passing paradigm and the structure of the input graph.

3 Message Flow Redundancy and Practical Implications

In this section, we first explain the redundancy in the computational graph and then discuss the over-squashing issue. Without loss of generality, we discuss the computational graph of a k-layer WL-test based GNN for the representation of the output node u. All proofs can be found in Appendix.

3.1 Message Flow Redundancy in Computational Graph

Definition 1 (Message flow)**.** *A message flow p in a computational graph represents the propagation chain of an input feature from an input node to an output node, where $p(i)$ denotes the i-th node of the reversed flow p (zero based index).*

A WL-based test GNN iteratively updates the node representation by aggregating the neighborhood information, where the message passing process (of the neighborhood information) is analogous to traverse the graph by walks, defined as follows.

Definition 2 (Walk)**.** *A walk is a sequence of nodes and edges in a graph that can be generated by a network traversal that permits backtracking and access to node self-loops.*

Corollary 1. *A message flow of node u in the computational graph is a walk of the same length that ends at u in the original graph.*

Definition 3 (Trail)**.** *A trail is a sequence of nodes and edges of a graph in which no edge is repeated.*

For instance, $u_1 \to u_2 \to u_3 \to u_1 \to u_4$ is a valid trail that has no repeated edge existed. Given a trail tr, we denote $tr(i)$ as the i-th node of the reversed trail tr (zero based index).

Lemma 1. *The message flows of a trail are the reversed walks of the trail, i.e., from the ending node to the beginning node.*

Remark 1. *A trail can not be compressed since there is no repeated edge in the trail. Message flows derived from the same trail should always carry identical trail-related information. Therefore, it is important to analyze the role of the trail playing in the message passing paradigm.*

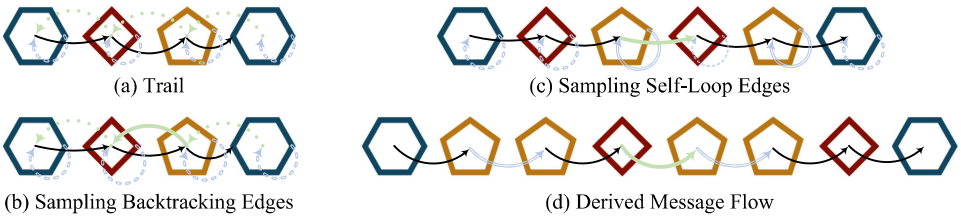


Figure 2: The process of deriving a message flow from a trail: Firstly, backtracking edges are sampled and placed into the trail. Then, self-loop edges are sampled and inserted into the trail. Finally, reversing the trail.

The number of message flows. Without loss of generality, suppose a trail of length r and a message flow of length k derived from the trail. According to Lemma 1, we can construct the message flow by adding backtracking edges and self-loop edges into the trail as illustrated in Figure 2. The backtracking edge of a self-loop edge is a self-loop edge. Thus, we begin by unorderly sampling s of the r backtracking edges with replacement. After the backtracking edges are placed, we unorderly sample $k - r - 2s$ of the $r + 1 + 2s$ self-loop edges with replacement.

Lemma 2. *Given a length-r trail, the number of length-k message flows derived from the trail is*

$$\sum_{s=0}^{\lfloor \frac{k-r}{2} \rfloor} r^s (r+1+2s)^{(k-r-2s)}$$

where $1 \leq r \leq k$.

Lemma 2 demonstrates that redundancy is prevalent since a trail of a length shorter than the number of GNN layers would result in many message flows.

3.2 Redundancy Causes Message Over-squashing

The message flow redundancy in computational graphs comes with a phenomenon, named *over-squashing*, which occurs when GNNs are unable to propagate information effectively across a graph [4]. Recently, Topping et al. [3] have proposed the *Jacobian* of a GNN-output to quantify the effect of over-squashing.

The influence of a change Given a graph with nodes u and v, as well as a k-layer WL-test based GNN to update node representations. the *Jacobian* measures how a change of the input feature $\mathbf{x}_v$ affects the node output $\mathbf{h}_u^{(k)}$.

$$\partial \mathbf{h}_u^{(k)} / \partial \mathbf{x}_v \tag{3}$$

Similar to [3], to avoid heavy notations, the node attributes and representations are assumed to be scalar; this assumption simplifies the analysis, whereas the vector case yields equivalent results.

Lemma 3. *Given a graph with nodes u and v, the computational graph of u by a k-layer WL-test based GNN. If all length-k paths in the computational graph of u are activated with the same probability, we can measure the* ***relative influence of the input feature*** $\mathbf{x}_v$ ***on the node output*** $\mathbf{h}_u^{(k)}$*, on average, as*

$$\mathbb{E}\left(\frac{\partial \mathbf{h}_u^{(k)}/\partial \mathbf{x}_v}{\sum_{v'\in V}\partial \mathbf{h}_u^{(k)}/\partial \mathbf{x}_{v'}}\right) = \frac{\left[\prod_{l=k}^{1}\mathcal{W}^{(l)}\right]_{u,v}}{\sum_{v'\in V}\left[\prod_{l=k}^{1}\mathcal{W}^{(l)}\right]_{u,v'}}, \tag{4}$$

where V is the node set of the graph and $\mathcal{W}^{(l)}$ is the weight matrix with $\mathcal{W}_{i,j}^{(l)}$ denoting the weight of the edge $(i,j)\in E$ in the l-th GNN layer.

The influence of a change along a message flow Given a graph, the computational graph of a node by a k-layer WL-test based GNN and its message flow p. The effect of a change in $\mathbf{x}_{p(0)}$ to the output node $\mathbf{h}_{p(k)}^{(k)}$ along a message flow p (a.k.a, **the influence of** $\mathbf{x}_{p(0)}$ **on** $\mathbf{h}_{p(k)}^{(k)}$ **along** p) can be computed as

$$\frac{\partial \mathbf{h}_{p(k)}^{(k)}}{\partial p} = \prod_{l=k}^{1}\frac{\partial \mathbf{h}_{p(l)}^{(l)}}{\partial \mathbf{h}_{p(l-1)}^{(l-1)}}. \tag{5}$$

Lemma 4. *Given a graph, the computational graph of a node by a k-layer WL-test based GNN and its message flow p. We compute* ***the relative influence of*** $\mathbf{x}_{p(0)}$ ***on*** $\mathbf{h}_{p(k)}^{(k)}$ ***along*** *p, on average, as*

$$\mathbb{E}\left(\frac{\partial \mathbf{h}_{p(k)}^{(k)}/\partial p}{\sum_{p'\in P}\partial \mathbf{h}_{p(k)}^{(k)}/\partial p'}\right) = \frac{\prod_{l=k}^{1}\mathcal{W}_{p(l),p(l-1)}^{(l)}}{\sum_{v'\in V}\left[\prod_{l=k}^{1}\mathcal{W}^{(l)}\right]_{p(k),v'}},$$

where P is the set of message flows that end at $tr(k)$, V is the node set of the graph, $\mathcal{W}^{(l)}$ is the weight matrix with $\mathcal{W}_{i,j}^{(l)}$ denoting the weight of the edge $(i,j)\in E$ in the l-th GNN layer.

The influence of a change along a trail Given a graph with a trail tr and the computational graph of tr's head node by a k-layer WL-test based GNN. Let P^* be the set of message flows in the computational graph of the node $tr(k)$ derived from tr. How a change in $\mathbf{x}_{tr(0)}$ will be propagated along tr and affect $\mathbf{h}_{tr(r)}^{(k)}$ (a.k.a, **the influence of** $\mathbf{x}_{tr(0)}$ **on** $\mathbf{h}_{tr(r)}^{(k)}$ **along** tr) , denoted as $\partial \mathbf{h}_{tr(r)}^{(k)}/\partial tr$, is computed as the summation of the influence of $\mathbf{x}_{tr(0)}$ on $\mathbf{h}_{tr(r)}^{(k)}$ along all message flows belonging to P^*, i.e.,

$$\frac{\partial \mathbf{h}_{tr(r)}^{(k)}}{\partial tr} = \sum_{p'\in P^*}\frac{\partial \mathbf{h}_{tr(r)}^{(k)}}{\partial p'} \tag{6}$$

Corollary 2. *Given a graph with a trail tr and the computational graph of tr's head node by a k-layer WL-test based GNN. We compute* ***the relative influence along*** *tr, on average, as*

$$\mathbb{E}\left(\frac{\partial \mathbf{h}_{tr(r)}^{(k)}/\partial tr}{\sum_{p'\in P}\partial \mathbf{h}_{p(k)}^{(k)}/\partial p'}\right) = \frac{\sum_{p'\in P^*}\prod_{r=k}^{1}\mathcal{W}_{p'(r),p'(r-1)}^{(r)}}{\sum_{v'\in V}\left[\prod_{r=k}^{1}\mathcal{W}^{(r)}\right]_{p(k),v'}},$$

where P is the set of message flows that end at $tr(k)$, P^* is the set of message flows derived from tr, and $\mathcal{W}^{(l)}$ is the weight matrix with $\mathcal{W}_{i,j}^{(l)}$ denoting the weight of the edge $(i,j)\in E$ in the l-th GNN layer.

Corollary 2 indicates the relative influence along a trail is proportional (in terms of probability) to the number of message flows derived from the trail. According to Lemma 2, a trail of longer length usually yields fewer messaging paths than a child trail of shorter length. Consequently, the influence

along a long path is always less than that of its sub-paths. Therefore, **the redundancy in the message passing paradigm is one key factor for the over-squashing phenomenon**. Topping et al. [3] stated the phenomenon of over-squashing occurs when the absolute value of the *Jacobian* $\partial \mathbf{h}_u^{(k)} / \partial \mathbf{x}_v$ is small. Nevertheless, we find the phenomenon of over-squashing would occur even if the absolute value of the *Jacobian* $\partial \mathbf{h}_u^{(k)} / \partial \mathbf{x}_v$ is large. Because, if the influence of $\mathbf{x}_v$ on $\mathbf{h}_u^{(k)}$ along a trail tr is relatively small, then the information of tr is hard to propagate into $\mathbf{h}_u^{(k)}$. In this situation, the relative influence of $\mathbf{x}_v$ on $\mathbf{h}_u^{(k)}$ along tr will decrease as the absolute value of the *Jacobian* $\partial \mathbf{h}_u^{(k)} / \partial \mathbf{x}_v$ increases.

4 Redundancy-Free Graph Neural Network

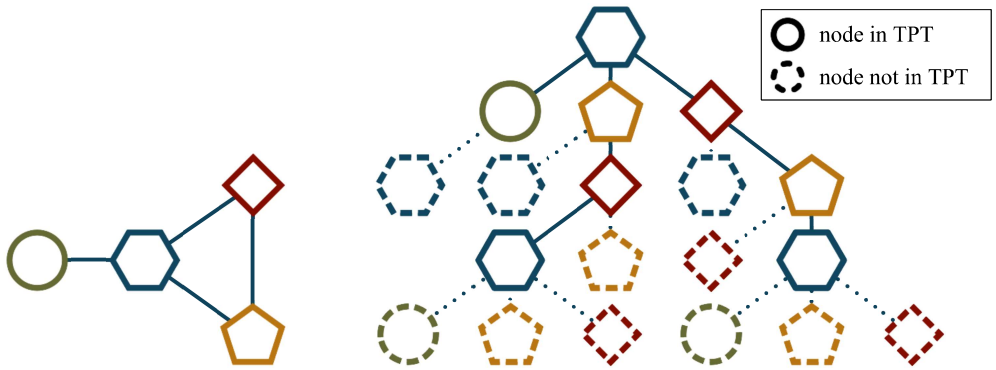
In the last section, we have discussed the message flow redundancy and its practical implications. To overcome this issue, we propose Redundancy-Free GNN (RFGNN), a kind of message passing GNN which is not only message flow redundancy-free but also improves the overall expressive power, message passing capability, and thus generates better performance. All proofs, and the space and time complexity analysis can be found in Appendix.

4.1 The RFGNN Framework

We first discuss the redundancy-free message passing paradigm based on a concept called extended path, $epath$, formally defined as

Definition 4 (Extended Path, *epath*)**.** *An* epath *is a path that has no repeated node along the path except that the starting node is allowed to be the ending node when the length of the path is larger than 2.*

For instance, $u_1 \to u_2 \to u_3 \to u_1$ is an epath but not a path since u_1 is repeated at the ending node. Note that $u_1 \to u_2 \to u_1$ is not an epath since the length is not larger than 2. Based on the extended path, we define a concept called *Truncated ePath Tree (TPT)* as follows.



(a) Input Graph (b) Process of ePaths Search

Figure 3: The height-4 Truncated ePath Tree of G from u (the blue one).

Definition 5 (Truncated ePath Tree (TPT))**.** *Given a graph $G = (V, E)$ and a node $u \in V$, the height-k TPT $\mathtt{TPT}^k_{(G,u)}$ is the tree rooted at u that is composed of all possible epaths with length up to k.*

As shown in Figure 3, the original graph can form a height-4 truncated epath tree (TPT) as a tree structure of 8 nodes, where every tree path (from the root to the leaf) is an extended path of length smaller than or equal to 4. For ease of discussion, given a node v on level k, we define $\mathcal{C}^k(v)$ as the child node set of the node v.

Definition 6 (Truncated ePaths Forest (TPF)). *Given a graph $G = (V, E)$, the height-k TPF of G is a multiset for all nodes $u \in V$ as*

$$\mathtt{TPF}_G^k := \left\{ \left\{ \mathtt{TPT}_{(G,u)}^k \mid \forall u \in V \right\} \right\}.$$

Redundancy-Free Graph Neural Network (RFGNN). Given $\mathtt{TPT} \in \mathtt{TPF}$ and $u \in \mathcal{V}(\mathtt{TPT})$ at level $(l-1)$. We denote $\mathbf{x}_u^{(l-1)}$ as a hidden feature transformed from the node features of u, $\mathbf{h}_v^{(l)}$ as a hidden feature of the subtree rooted at node $v \in \mathcal{C}^{(l-1)}(u)$, and $\mathbf{e}_{u,v}^{(l)}$ as a hidden feature transformed from the edge features of (u, v). We can describe the RFGNN representation update process of a node u at level $(l-1)$ as follows.

$$\mathbf{h}_u^{(l-1)} = \psi^{(l)} \left(\mathbf{x}_u^{(l-1)}, \cup_{v \in \mathcal{C}^{(l-1)}(u)} \phi^{(l)} \left(\mathbf{e}_{u,v}^{(l)}, \mathbf{h}_v^{(l)} \right) \right) \tag{7}$$

where $\cup$ denotes a differentiable and permutation invariant function, e.g., *sum* and *mean*, and ϕ and ψ denote differentiable functions such as MLPs (Multi Layer Perceptrons).

To obtain the entire graph representation $\mathbf{h}_G$, we adopt a readout function χ as in Equation 8 that aggregates the root node representations of the TPTs.

$$\mathbf{h}_G = \chi \left(\left\{ \mathbf{h}_{root(\mathtt{TPT})}^{(0)} \mid \mathtt{TPT} \in \mathtt{TPF}_G \right\} \right). \tag{8}$$

where the readout function χ can be a simple permutation invariant function such as *sum*.

Symmetries. Given a graph G with adjacency matrix A and feature matrix $\mathbf{x}$, a function f is (node) permutation equivalent if $f(A, \mathbf{x}) = f\left(MAM^T, M\mathbf{x}\right)$, for any permutation matrix M. GNN layers respect this equation, which ensures they compute the same functions up to a permutation (i.e. relabeling) of the nodes. Similarly, RFGNN is equivalent with respect to permutations of nodes, since the resulted TPTs and TPF is independent on any node permutation.

Corollary 3. *Redundancy-Free GNNs are permutation equivalent.*

4.2 The Expressive Power of RFGNN

1-WL is recognized as an indicator for the expressive power of GNNs [11, 12, 13]. In line with these work, we measure the expressive power of our model using a hashing procedure of the TPF. We first define the hashing procedure of the TPF (TPFH), and then show the expressive power of TPFH is better than that of 1-WL.

TPF Hashing (TPFH). TPFH is a hashing scheme for a k-height TPF, which is a generalization of 1-WL [1]. In short, the hashing scheme is to assign labels (colours) to all nodes. For ease of discussion, we first denote $u^{(l)} \in \mathtt{TPT}$ as a node u at level l of the truncated epath tree $\mathtt{TPT}$. Let $c_u^{(l)}$ be the colour assigned to node $u^{(l)}$ and let $t_u^{(l)}$ be the colour assigned to the subtree rooted at node $u^{(l)}$. We initialize the colour of the leaf nodes (at level k) by their colour in the original graph. The hashing procedure is conducted recursively from the level $(k-1)$ to the root. Specifically, we collect the tree colour of the child nodes of u, denoted as $\{\{t_v^{(l+1)} \mid \forall v \in \mathcal{C}^{(l)}(u)\}\}$ (note it would a null multiset if node u has no child nodes). Then, we compute the tree colour of u by an injective mapping, $t_u^{(l)} = \mathrm{HASH}(c_u^{(l)}, \{\{t_v^{(l+1)} \mid \forall v \in \mathcal{C}^{(l)}(u)\}\})$, where HASH is a perfect hash function. Finally, we collect colours of all TPTs of the TPF as a multiset and injectively map it into a label by a perfect hash function. Two graphs are considered non-isomorphic if labels of their TPFs are equal. Otherwise, the test is inconclusive.

Lemma 5. *For two graphs G_1 and G_2, and two nodes $u \in \mathcal{V}(G_1)$ and $v \in \mathcal{V}(G_2)$, if the k-height WL-subtree of u is non-isomorphic to that of v, the k-height TPTs of them are non-isomorphic too.*

Lemma 6. *There exist two graphs G_1 and G_2 and two nodes, $u \in \mathcal{V}(G_1)$ and $v \in \mathcal{V}(G_2)$, where the k-height WL-subtree of u is isomorphic to that of v, but the k-height TPTs of them are non-isomorphic.*

According to Lemma 5 and Lemma 6, the expressive power of TPFH is better than that of 1-WL. Similar to the proof in WL-test based GNNs [11], RFGNN can be shown to provide the same expressive power of TPFH as long as they are equipped with a sufficient number of layers and the parametric local aggregators they use can learn to be injective.

4.3 Understanding the Learning of RFGNN

We attempt to investigate what a RFGNN has learned by measuring how the output of a k-layer RFGNN, as in Equation 7 , is influenced by input values and hidden features.

Lemma 7. *Assume a k-layer RFGNN as in Equation 7, a graph $G = (V, E)$, and an epath ep within G of length $l \leq k$. We can measure the* ***influence of the node feature*** $\mathbf{x}^{(i)}_{ep[i]}$ ***on the tree representation*** $\mathbf{h}^{(0)}_{ep[0]}$*, where $0 \leq i \leq l$, as*

$$\frac{\partial \mathbf{h}^{(0)}_{ep[0]}}{\partial \mathbf{x}_{ep[i]}} = \left(\prod_{j=0}^{i-1} \left(\partial_2 \psi^{(j)} \partial_3 \phi^{(j)} \right) \right) \left(\partial_{\mathbf{x}_{ep[i]}} \psi^{(i)} + \partial_2 \psi^{(i)} \partial_{\mathbf{x}_{ep[i]}} \phi^{(i)} \right)$$

Similarly, we can measure the ***influence of the subtree representation*** $\mathbf{h}^{(i)}_{ep[i]}$ ***on the tree representation*** $\mathbf{h}^{(0)}_{ep[0]}$*, where $0 \leq i \leq l$, as*

$$\frac{\partial \mathbf{h}^{(0)}_{ep[0]}}{\partial \mathbf{h}^{(i)}_{ep[i]}} = \left(\prod_{j=0}^{i-1} \left(\partial_2 \psi^{(j)} \partial_2 \phi^{(j)} \right) \right) \partial_2 \psi^{(i)} \partial_{\mathbf{h}^{(i)}_{ep[i]}} \phi^{(i)}$$

Avoid over-quashing. Lemma 7 points out that if all edges in the computational graph of the RFGNN is activated with the same probability, on average, epaths starting with the same node and having the same length would have the same influence on their shared starting node. This is similar to the finding of [14, 3]. Thus, the problem of over-squashing induced by message flow redundancy in the computational graph is avoided.

Capture arbitrary subtrees According to Lemma 7, given a k-layer RFGNN as in Equation 7, a graph $G = (V, E)$, and an epath ep within G of length $l \leq k$, the influence of the input feature $\mathbf{x}^{(i)}_{ep[i]}$ on the node output $\mathbf{h}^{(0)}_{ep[0]}$ depends on the features of nodes $ep[0], ep[1], \cdots, ep[i]$, and the representation of the tree rooted at $ep[i]$. Thus, RFGNN is able to to detect the presence of the epath ep within the TPT rooted at $ep[0]$, and the presence of a subtree rooted at $ep[i]$ within this TPT. Accordingly, RFGNN is able to to detect the presence of an arbitrary subtree of the TPT rooted at $ep[0]$, by detecting the paths and subtrees that comprise the TPT. As a result, RFGNN is able to estimate the similarity between graphs at various levels of granularity.

Lemma 8. *Given a graph $G = (V, E)$, a node $u \in V$, and an edge $e \in E$ within the k-hop neighborhood of u. Let n be the number of rings in the k-hop neighborhood of u. We have that the number of repeations of e within the k-height TPT of u is no more than $1 + n$.*

Graph similarity comparison and graph overlap detection. Intuitively, GNNs should be capable of encoding graphs with near graph edit distance (GED) [15] into similar representations. As stated in the introduction, WL-test based GNNs would generate vastly different message passing graphs for similar graphs. According to Lemma 8, RFGNN always generate similar message passing graphs for similar graphs, which should learn better representations of similar graphs than WL-test based GNNs. Besides, RFGNN should handle the graph overlaping structure better than WL-test based GNNs since overlapping subgraphs always generate a large number of the same epaths in the computational graphs (and thus overlapping subgraphs can be captured). Note that RFGNN should generate fewer message flows than other WL-test based GNNs due to the redundancy-free property.

Memory consumption Even RFGNN can provide better expressive power than WL-test based GNNs. However, its memory consumption may also be higher since it requires constructing a Truncated ePath Forest (TPF). In the future work, we will carefully study potential techniques, e.g., stochastic sampling, to overcome this issue.

5 Related Work

High order GNNs. Recently, expressive GNNs have been developed, which based on enhanced WL algorithms. such as PPGN (based on k-WL) [12], CWNs (based on Cellular WL) [16], and Nested GNNs (based on Nested 1-WL) [17]. Nonetheless, none of them breaks the iterative aggregation-update scheme.

Over-squashing issue. The *Jacobian* of a GNN-output is limited by a term given as the power of the graph normalised adjacency matrix [3]. And, [4, 3] both argued that small absolute values of the *Jacobian* are indicative of poor information propagation or over-squashing. Furthermore, [3] investigated graphs from a Ricci curvature perspective and demonstrated how negative curved edges contribute to the problem of over-squashing. In this work, we prove that the redundancy in the computational graph is one key factor for the over-squashing issue, indicating that the over-squashing phenomenon will occur regardless of the absolute value of the *Jacobian*.

Graph kernels. Graph kernels are frequently employed to measure the similarities between graphs by decomposing graphs into substructures and comparing these substructures [18]. The measurement of graph similarity could be utilized for subsequent tasks such as classifications [18]. In comparison to earlier graph kernels, Tree++ [5] has the highest classification accuracy on numerous benchmark datasets. The main reason for its success is that this method allows comparing graphs at multiple levels of granularities.

6 Experiments

In this section, we study the effectiveness of the RFGNN for graph classification tasks. We implement RFGNN based on the Deep Graph Library (DGL) [19]. The experiments were run on a Linux server with 64GB memory, one GeForce GTX 1070 (8G) GPU and an Intel 16-core i9-9900K CPU.

To demonstrate the theoretical results in this paper we ran a suite supervised graph classification tasks comparing our RFGNN to graph kernels (such as WL-subtree [2] and Tree++ [5]) and GNNs (such as GIN [11], PPGN [13], and CIN [16]). We evaluate the methods on 8 benchmarks: 5 bioinformatics datasets (NCI1 [20], PTC [21], MUTAG [22], PROTEINS [23], ENZYMES [23]) and 3 small molecules datasets (BZR, COX2, DHFR) from TU Datasets [24]. We follow the experiment configurations of [11], such as the number of hidden units (32), the batch size (128), 10-fold cross validation, and learning rate (initialized as 0.02 decayed by 0.5 every 50 epochs). In addition, our RFGNN employs a TPF with a height of 3, all ϕ functions in Equation 7 have one layer, ψ functions in Equation 7 have two layers, and 0.5 dropout ratio is only applied on the representations of TPTs. For baseline methods, we report the accuracies reported in the original papers. Dataset statistics and performance comparison over them are summarized in Table 1, and more details can be found in Appendix. We see that RFGNN outperforms GIN and even high order GNNs on most datasets, supporting our argument that RFGNN is a theoretically sound method for improving GNN performance. Also, we see RFGNN outperforms Tree++, which allows comparing graphs at multiple levels of granularities.

Table 1: Statistics of the datasets and performance comparison.

Datasets	MUTAG	PROTEINS	PTC	NCI1	ENZYMES	BZR	COX2	DHFR
#graphs	188	1113	344	4110	600	405	467	756
#classes	2	2	2	2	6	2	2	2
Avg. #nodes	17.93	39.06	25.56	29.87	32.63	35.75	41.22	42.43
Avg. #edges	19.79	72.82	25.96	32.30	62.14	38.36	43.44	44.54
TPF Avg. # nodes	232.84	2165.3	336.56	376.48	1908.11	503.12	564.87	563.00
TPF Avg. # edges	216.77	2160.0	311.41	349.05	1904.98	469.98	525.87	522.69
WL subtree	90.4 ± 5.7	75.0 ± 3.1	59.9 ± 4.3	$\mathbf{86.0 \pm 1.8}$	62.6 ± 1.2	87.3 ± 0.8	81.2 ± 1.1	82.4 ± 0.9
Tree++	–	75.5 ± 0.5	–	85.8 ± 0.1	–	87.9 ± 1.0	84.3 ± 0.9	83.7 ± 0.6
GIN	90.0 ± 8.8	76.2 ± 2.8	64.6 ± 7.0	82.7 ± 1.7	72.5 ± 6.1,	89.9 ± 3.8	$\mathbf{87.6 \pm 4.0}$	82.9 ± 3.9
1-2-3-GNN	92.2 ± 4.4	77.3 ± 4.2	–	61.0 ± 5.3	–	82.0 ± 3.3	78.7 ± 1.3	63.4 ± 4.9
PPGN	90.6 ± 8.7	77.2 ± 4.7	66.2 ± 6.5	83.2 ± 1.1	63.3 ± 11.8	89.4 ± 4.0	86.7 ± 4.1	79.6 ± 4.5
CIN	92.7 ± 6.1	77.0 ± 4.3	68.2 ± 5.6	83.6 ± 1.4	–	78.8 ± 1.0	78.4 ± 1.1	68.1 ± 3.1
RFGNN	$\mathbf{93.3 \pm 6.0}$	$\mathbf{77.5 \pm 3.4}$	$\mathbf{71.5 \pm 3.2}$	83.6 ± 1.6	$\mathbf{75.3 \pm 5.0}$	$\mathbf{90.9 \pm 3.2}$	87.4 ± 4.6	$\mathbf{84.5 \pm 4.6}$

7 Conclusions

In this work, we carefully investigate the message flow redundancy issue in recent GNNs and its implications, especially, we have shown the message flow redundancy in one key factor for the over-squashing phenomenon. To address the message flow redundancy issue, we have proposed Redundancy-Free Graph Neural Networks (RFGNN), which conducts redundancy-free message passing. Theoretically, we demonstrate the benefits of our method, and experimentally, we show that our RFGNN outperforms existing WL-test based GNNs on several datasets.

References

[1] Boris J Weisfeiler and Andrei A Lehman. A reduction of a graph to a canonical form and an algebra arising during this reduction. nauchno-technicheskaya informatsiya, 2 (9): 12–16, 1968.

[2] Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M. Borgwardt. Weisfeiler-Lehman Graph Kernels. *Journal of Machine Learning Research*, 12(77):2539–2561, 2011.

[3] Jake Topping, Francesco Di Giovanni, Benjamin Paul Chamberlain, Xiaowen Dong, and Michael M. Bronstein. Understanding over-squashing and bottlenecks on graphs via curvature. In *International Conference on Learning Representations*, 2022.

[4] Uri Alon and Eran Yahav. On the bottleneck of graph neural networks and its practical implications. In *9th International Conference on Learning Representations, ICLR 2021*, 2021.

[5] Wei Ye, Zhen Wang, Rachel Redberg, and Ambuj K. Singh. Tree++: Truncated tree based graph kernels. *IEEE Transactions on Knowledge and Data Engineering*, 33:1778–1789, 2021.

[6] Yu Su, Fangqiu Han, Richard E Harang, and Xifeng Yan. A fast kernel for attributed graphs. In *Proceedings of the 2016 SIAM international conference on data mining*, pages 486–494. SIAM, 2016.

[7] Giannis Nikolentzos, Polykarpos Meladianos, and Michalis Vazirgiannis. Matching node embeddings for graph similarity. In *Thirty-first AAAI conference on artificial intelligence*, 2017.

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

[9] Anton Tsitsulin, Davide Mottin, Panagiotis Karras, Alexander M. Bronstein, and Emmanuel Müller. Netlsd: Hearing the shape of a graph. *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018.

[10] Michael Collins. Computational graphs, and backpropagation. *Lecture Notes, Columbia University.*, pages 11–23, 2018.

[11] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019.

[12] Haggai Maron, Heli Ben-Hamu, Nadav Shamir, and Yaron Lipman. Invariant and equivariant graph networks. *ArXiv*, abs/1812.09902, 2019.

[13] 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 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 2153–2164, 2019.

[14] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. In *ICML*, pages 5449–5458, 2018.

[15] Xinbo Gao, Bing Xiao, Dacheng Tao, and Xuelong Li. A survey of graph edit distance. *Pattern Analysis and applications*, 13(1):113–129, 2010.

[16] Cristian Bodnar, Fabrizio Frasca, Nina Otter, Yu Guang Wang, Pietro Lio', Guido Montúfar, and Michael M. Bronstein. Weisfeiler and lehman go cellular: Cw networks. In *NeurIPS*, 2021.

[17] Muhan Zhang and Pan Li. Nested graph neural networks. *Advances in Neural Information Processing Systems*, 34, 2021.

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

[19] Minjie Wang, Da Zheng, Zihao Ye, Quan Gan, Mufei Li, Xiang Song, Jinjing Zhou, Chao Ma, Lingfan Yu, Yujie Gai, Tianjun Xiao, Tong He, George Karypis, Jinyang Li, and Zheng Zhang. Deep graph library: A graph-centric, highly-performant package for graph neural networks. *arXiv: Learning*, 2019.

[20] John Boaz Lee, Ryan Rossi, and Xiangnan Kong. Graph classification using structural attention. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1666–1674, 2018.

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

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

[23] Karsten M Borgwardt, Cheng Soon Ong, Stefan Schönauer, SVN Vishwanathan, Alex J Smola, and Hans-Peter Kriegel. Protein function prediction via graph kernels. *Bioinformatics*, 21(suppl_1):i47–i56, 2005.

[24] Ahmed El Kerdawy, Stefan Güssregen, Hans Matter, Matthias Hennemann, and Timothy Clark. Quantum mechanics-based properties for 3d-qsar. *Journal of chemical information and modeling*, 53 6:1486–502, 2013.

[25] Raghunathan Ramakrishnan, Pavlo O. Dral, Matthias Rupp, and O. Anatole von Lilienfeld. Quantum chemistry structures and properties of 134 kilo molecules. *Scientific Data*, 1, 2014.

[26] Zhenqin Wu, Bharath Ramsundar, Evan N. Feinberg, Joseph Gomes, Caleb Geniesse, Aneesh S. Pappu, Karl Leswing, and Vijay S. Pande. Moleculenet: A benchmark for molecular machine learning. *arXiv: Learning*, 2017.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [No]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]
   (b) Did you include complete proofs of all theoretical results? [Yes]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] See 6. Our work uses DGL (a graph neural network library). We cite the creator.
   (b) Did you mention the license of the assets? [No] No license should be mentioned.
   (c) Did you include any new assets either in the supplemental material or as a URL? [No]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [No] All datasets used in this work are open access.
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [No] No such data.
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [No]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [No]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [No]