Filtration-Enhanced Graph Transformation

Zijian Chen[1] , **Rong-Hua Li**[1] , **Hongchao Qin**[1] , **Huanzhong Duan**[2] , **Yanxiong Lu**[2] , **Qiangqiang Dai**[1] and **Guoren Wang**[1]

[1]Beijing Institute of Technology
[2]Tencent
blockchanzj@gmail.com, lironghuabit@126.com, qhc.neu@gmail.com,
{boosterduan,alanlu}@tencent.com, qiangd66@gmail.com, wanggrbit@126.com

Abstract

Graph kernels and graph neural networks (GNNs) are widely used for the classification of graph data. However, many existing graph kernels and GNNs have limited expressive power, because they cannot distinguish graphs if the classic 1-dimensional Weisfeiler-Leman (1-WL) algorithm does not distinguish them. To break the 1-WL expressiveness barrier, we propose a novel method called filtration-enhanced graph transformation, which is based on a concept from the area of topological data analysis. In a nutshell, our approach first transforms each original graph into a filtration-enhanced graph based on a certain pre-defined filtration operation, and then uses the transformed graphs as the inputs for graph kernels or GNNs. The striking feature of our approach is that it is a plug-in method and can be applied in any graph kernel and GNN to enhance their expressive power. We theoretically and experimentally demonstrate that our solutions exhibit significantly better performance than the state-of-the art solutions for graph classification tasks.

1 Introduction

Graphs are well-studied structures which are utilized to model entities and their relationships. Graph classification is an important task in many application domains such as chemistry [Shervashidze *et al.*, 2011], biology [Kriege and Mutzel, 2012], social networks [O'Bray *et al.*, 2021] and natural language processing [Nikolentzos *et al.*, 2017a]. For example, in Chemistry, we are often interested in predicting the toxicity of chemical compounds using graph classification methods.

Graph kernels and graph neural networks (GNNs) are two main approaches for graph classification. Graph kernel methods have achieved state-of-the-art (SOTA) results in many datasets [Kriege *et al.*, 2020]. Most previous graph kernels are based on the classic R-convolution framework which decompose graphs into substructures and add up the pairwise similarities between these substructures with different criterions, such as subtrees [Shervashidze *et al.*, 2011], cycles [Horváth *et al.*, 2004], shortest paths [Borgwardt and Kriegel, 2005] and small subgraphs [Shervashidze *et al.*, 2009]. Current studies on graph kernels focus mainly on Weisfeiler-Leman (WL) based methods [Morris *et al.*, 2017; Kriege *et al.*, 2016; Morris *et al.*, 2020], continuous attributes [Kriege and Mutzel, 2012], multi-scale methods [Kondor and Pan, 2016] and others [Zhang *et al.*, 2018; Du *et al.*, 2019].

Another line of SOTA methods for graph classification are GNN based methods. Many existing GNNs first employ a message-passing aggregation scheme to learn low-dimensional vectors for nodes in a graph [Hamilton *et al.*, 2017; Xu *et al.*, 2019]. Then, a graph-level representation is obtained by performing a pooling operation [Lee *et al.*, 2019] which will be used to classify graphs. The main limitation of the 1-dimensional Weisfeiler-Leman (1-WL) graph kernels and the message-passing based GNNs is that they cannot distinguish graphs if the 1-WL test does not distinguish them [Xu *et al.*, 2019]. To break the 1-WL expressiveness barrier, recent works pay more attention on graph kernels or GNN beyond 1-WL test, such as high-dimensional WL kernels [Morris *et al.*, 2017], high-order GNN [Morris *et al.*, 2020], counting pre-defined substructures as additional features [Bouritsas *et al.*, 2020], calculating structural coefficients for message passing [Wijesinghe and Wang, 2022] and augmenting node identifiers [You *et al.*, 2021] and random features [Sato *et al.*, 2021; Abboud *et al.*, 2021] into GNNs.

In this work, we propose a novel approach to circumvent the expressive power limit in 1-WL graph kernels and message-passing based GNNs. Our work is inspired by a recent work which uses filtration curves to represent a graph [O'Bray *et al.*, 2021], where filtration is a well-known concept in topological data analysis. As shown in [O'Bray *et al.*, 2021], the filtration curve can capture the topological features of the graph structure. However, it may still miss some local graph structure features, since it drops the original graph structure and only uses the filtration-curve representation for graph classification. Unlike [O'Bray *et al.*, 2021], we propose a filtration-enhanced graph (FEG) transformation method which transforms each original graph into a *more expressive* graph based on a pre-defined filtration. The filtration-enhanced graphs are then fed into graph kernels or GNNs. We prove that existing graph kernels or GNNs using our filtration-enhanced graphs have strictly better expressive power than the same graph kernels or GNNs using the original graphs. Moreover, the proposed FEG transformation can be computed in linear time using linear space with respect to the original graph size, thus it can handle large graphs. Another nice feature of our approach is that it is a *plug-in* method

and can be used for any graph kernel and GNN to enhance their expressive power. To summarize, the main contributions of this paper are as follows.

- We propose a simple yet powerful framework to transform original graphs to filtration-enhanced graphs.
- We theoretically prove that filtration-enhanced graphs are more expressive than original graphs in WL test using relatively low time and space complexity.
- We conduct extensive experiments using several benchmark graph datasets. The results show that (1) our filtration-enhanced graphs are more expressive than the original graphs in graph classification tasks, (2) and the FEG based graph kernels or GNNs consistently outperform the state-of-the-art solutions.

2 Preliminaries

Let $G = (V, E)$ be an undirected graph, where V and E denotes the set of vertices and edges respectively. We refer to G as an attributed graph if its vertices or its edges have native attributes, otherwise G is called a non-attributed graph. Let $N(u)$ be the set of neighbors of vertex u in G, and $d_u = |N(u)|$ be the degree of u. An edge weight function of a graph, defined by $w : E \to \mathbb{R}$, assigns a real-value weight to each edge $e \in E$, which is often used to represent the similarity between the two end vertices of the edge. Below, we give some weighted functions used in our work.

Edge weight functions. All weight functions are only used for filtration in this paper. For attributed graphs, we consider the following edge weight functions. For the attributed graphs that have native edge weights, the output of the edge weight function is equal to the native edge weight for each edge. For the attributed graphs whose vertices have native attributes, the output of the edge weight function is the L_2-distance between the attribute vectors of two end vertices of the edge.

For the non-attributed graphs, we consider three different edge weight functions. (1) *Common neighbors*: the edge weight is the number of common neighbors of two end vertices of an edge. Note that computing the number of common neighbors for each edge can be done in $O(\alpha m)$ time by using a triangle enumeration algorithm, where α denotes the arboricity of the graph which is often a small constant in real-world sparse graphs [Chiba and Nishizeki, 1985]. (2) *Core numbers*: the edge weight is the sum of the k-core numbers of two end vertices of an edge. Here a k-core of a graph G is a maximal subgraph of G in which all vertices have degree at least k [Seidman, 1983]. The core number of a vertex is the largest value k of a k-core containing that vertex. The core numbers for all vertices can be computed by a classic linear-time core decomposition algorithm [Matula and Beck, 1983]. (3) *Ricci curvature*: the edge weight based on Ricci curvature is defined as follows. For each vertex $u \in G$, we can define a probability measure m_u^α as (i) $m_u^\alpha = \alpha$ if $u = v$, (ii) $m_u^\alpha = (1-\alpha)/d_u$ if $v \in N(u)$, and (iii) $m_u^\alpha = 0$ otherwise. The Ricci curvature of an edge between u and v, denoted by $\kappa(u, v)$, is based on the Wasserstein distance $W(\cdot, \cdot)$ between their respective probability measures, i.e., $\kappa(u, v) = 1 - \frac{W(m_u^\alpha, m_v^\alpha)}{d(u,v)}$ [Lin *et al.*, 2011]. Note that Ricci curvature is a graph isomorphism invariant by accessing the similarity of neighbors between two nodes [Lin *et al.*, 2011], which is widely used in many graph learning tasks [Zhao and Wang, 2019; O'Bray *et al.*, 2021]. Following previous studies [Zhao and Wang, 2019; O'Bray *et al.*, 2021], we set $\alpha = 0.5$ in this paper.

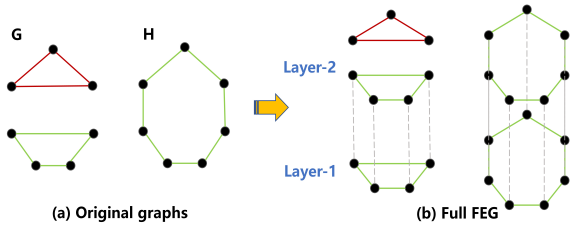


Figure 1: G and H are indistinguishable by 1-WL test, but can be distinguished after filtration-enhanced graph (FEG) transformation with a common neighbor based edge weight function.

Filtration. Filtration is a key concept in computational topology [Edelsbrunner *et al.*, 2002]. For a graph $G = (V, E)$ and an edge weigh function w, a filtration is a sequence of monotonically-growing subgraphs $\emptyset \subseteq G_1 \subseteq G_2 \cdots \subseteq G_l \subseteq G$ in which each subgraph G_i can be deemed as a *filtered graph* of G based on the edge weights. Without loss of generality, we assume that the weights take values from an interval $[w_{\min}, w_{\max}]$. Then, we can pick l different weight values $w_{\min} \leq w_1 < w_2 < \cdots < w_l \leq w_{\max}$ to derive a filtration. Specifically, the subgraph G_i in the filtration is the subgraph induced by all edges with weights no larger than w_i. Formally, we have $G_i = (V_i, E_i)$ where $E_i = \{(u, v) \in E | w(u, v) \leq w_i\}$ and $V_i = \{v \in V | (u, v) \in E_i\}$. Note that once the weights for all edges are determined, the filtration is easy to derive by a sorting algorithm [O'Bray *et al.*, 2021].

1-dimensional Weisfeiler-Leman test. The 1-dimensional Weisfeiler-Leman test (1-WL test) is a classic and efficient algorithm to check whether two given graph is isomorphism [Weisfeiler and Leman, 1968]. The 1-WL test is a color refinement algorithm which iteratively revises the vertex colors until a fixed point is reached. Specifically, the 1-WL test iteratively (1) aggregates the colors of nodes and their neighbors, and (2) hashes the color multi-set into unique new colors. The algorithm terminates when there is no vertex's color needed to update. The algorithm decides that two graphs are non-isomorphic if at a certain iteration the colors of vertices between two graphs differ. It is well-known that the 1-WL test cannot distinguish regular graphs (every vertex in the graph has the same degree). For example, we can easily verify that the graphs G and H shown in Fig. 1(a) cannot be distinguished by the 1-WL test.

3 The Proposed Method

As discussed in Section 1, many existing 1-WL graph kernels and GNNs suffer from the 1-WL expressiveness limit. To enhance the expressive power of those methods, we propose a simple but very effective approach, called filtration-enhanced graph (FEG) transformation, which first transforms each input graph into a *more expressive graph* based on the concept of filtration and then feeds those *more expressive graphs* into graph kernels or GNNs for downstream graph classification applications. Below, we describes the details of our solution,

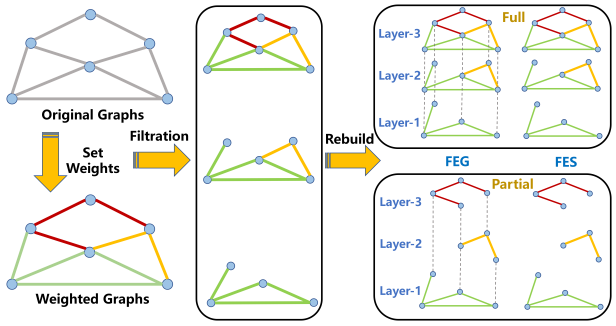


Figure 2: Illustration of the filtration-enhanced graph transformation

followed by the analysis of the expressiveness and computational complexity of our method.

3.1 Filtration-enhanced Graph Transformation

The filtration-enhanced graph (FEG) transformation aims to transform a graph into an FEG based on a pre-defined filtration. Specifically, an FEG contains l layers; each layer is a subgraph in the filtration of the original graph and there are some *cross* edges that connect the same vertex across two different layers.

There are two steps to create an FEG with l layers. First, we assign a weight for each edge in the graph based on an edge weight function, and then compute an l-layer filtration using l different weight values $w_0 = w_{\min} \leq w_1 < w_2 < \cdots < w_{l-1} \leq w_{\max} = w_l$. Here we just partition the interval $[w_{\min}, w_{\max}]$ into l sub-intervals with the same length ($|w_i - w_{i-1}|$ equals the same value for all i). Clearly, after this step, we can obtain l subgraphs by filtering the edges with weights w_i for $i = 1, 2, \cdots l$, i.e., $G_1 \subseteq G_2 \subseteq \cdots \subseteq G_l = G$. Second, we add a *cross* edge between two vertices cross two *closest layers* if these two vertices are the same vertex in the original graph, i.e., adding an edge $(v(G_i), v(G_j))$ if v appears in both G_i and G_j and $j - i$ is minimum for all $i < j \leq l$. For convenience, we refer to such an FEG as a full FEG. Clearly, the number of edges of the full FEG is at most $O(l)$ times of that of the original graph. To reduce the graph size, we can only consider the new edges for each layer and drop the edges that are already contained in the previous layers. In other words, we only maintain the *incremental subgraphs* in each layer and the resulting FEG is composed of $G_1, G_2 \setminus G_1, \cdots, G_l \setminus \{\cup_{i=1}^{l-1} G_i\}$. Similar to the full FEG, we also add the *cross* edges between two same vertices cross two closest layers. To distinguish the full FEG, we refer to such an FEG as a partial FEG. Fig. 2 illustrates the detailed procedure of the FEG transformation. The dashed lines in Fig. 2 represent the cross edges of vertices in different layers.

Since our FEGs are generated by a filtration, they can capture the topological features of the original graph with an edge weight function. The intuition behind our method is that different edge weights can be seen as a changing sequence according to the filtration. With these weights, the original graph is divided into different layers which reflect how the original graph is generated. The cross edges in our FEGs further represent the properties of vertices that are updated. As a result, FEGs express the change of the original graph with the structure of itself. The difference between full FEG and partial FEG is whether we take the *out-dated* edges into account. For example, in Fig. 2, the green edges are out-dated in the second layer of the partial FEG, but we still add them in the second layer of the full FEG. Intuitively, since the original graph is augmented by our FEG transformation with some additional topological features, its expressive power should be better than the original graph. Indeed, as shown in Fig. 1, the two original graphs G and H are indistinguishable by the 1-WL test, but they can be distinguished after FEG transformation. In Section 3.2, we will formally analyze the expressive power of our approach.

Filtration-enhanced snapshots. We also introduce a variant of FEG, called filtration-enhanced snapshots (FES), by dropping the cross edges in an FEG. Specifically, an FES is a set of subgraphs $G_1 \subseteq G_2 \subseteq \cdots \subseteq G_l = G$ generated by a filtration. Similar to FEGs, we have full FES and partial FES, and the difference between them is whether we consider the *out-dated* edges or not. Fig. 2 illustrates the concepts of full FES and partial FES. Note that both full FEG and partial FEG are a single graph which can be directly fed into any graph kernel or GNN for downstream graph classification tasks. However, both full FES and partial FES are a set of graphs, thus they cannot be directly used for graph kernels. To make them also feasible for graph kernels, we can compare the graphs in each layer and then take the sum over all layers.

Specifically, let f be a graph kernel function, $FES = \{G_i | 1 \leq i \leq l\}$ and $FES' = \{G'_i | 1 \leq i \leq l\}$ are two FESes, then the graph kernel of FES is defined by $f_\delta(FES, FES') = \sum_{i=1}^{l} f_\delta(G_i, G'_i)$. Here f_δ is defined as

$$f_\delta(G_i, G'_i) = \begin{cases} \delta & G_i = G'_i = \emptyset \\ f(G_i, G'_i) & G_i \neq \emptyset \ \& \ G'_i \neq \emptyset \\ 0 & \text{otherwise,} \end{cases} \quad (1)$$

where f is basic graph kernel function and δ is a constant which is equal to the minimum $f(G_i, G'_i)$ for any $G_i \neq \emptyset$ and $G'_i \neq \emptyset$. Note that the reason we use δ is that it can measure the similarity of two empty subgraphs in the filtration. The following theorem shows that the kernel matrix obtained by f_δ is still semi-positive definite if the kernel matrix derived by the original kernel function f is semi-positive definite. Due to the space limit, all proofs in this paper can be found in the supplementary document.

Theorem 1. *If f is a kernel function on graphs, then f_δ is also valid kernel function.*

3.2 Expressiveness

Here we analyze the expressiveness of FEG and FES in WL test. Let $G = (V, E)$ be the original graph, $FEG = (\cup_{i=1}^{l} V_i, E_1 \cup \cdots \cup E_l \cup E_{1,2} \cup \cdots \cup E_{l-1,l})$ be the full FEG of G with l layers of the original graph, $FES = \cup_{i=1}^{l} (V_i, E_i)$ be the full FES of G with l layers of the original graph, where V_i and E_i are vertices and edges of the i-th layer subgraph respectively, $E_{i,i+1}$ is the cross edges of two adjacent layers. $V_i(FEG)$ is the vertex set of the i-th layer. We assign each vertex a layer-wise color, expressed as a 2-tuple $\langle layer, color \rangle$. Let $C^i(u)$ be the color of u at the i-th iteration of WL test and $C^i(S) = \{C^i(x) \mid \forall x \in S\}$. Then, by the classic WL algorithm [Weisfeiler and Leman, 1968],

we have $C^{i+1}(u) = \phi(C^i(u), C^i(N(u)))$, where ϕ is a hash function and $N(u)$ is the set of neighbors of u.

Lemma 1. *Let G and G' be two graphs or two filtration-enhanced graphs. The following statements are true: (a) $\forall u \in G, u' \in G'$, if $C^i(u) \neq C^i(u')$, then $C^{i+1}(u) \neq C^{i+1}(u')$. (b) $\forall u, v \in G$, if $C^i(u) \neq C^i(v)$, then $C^{i+1}(u) \neq C^{i+1}(v)$.*

Lemma 2 (Different layers of FEG are independent in WL test)**.** *Let FEG and FEG' be any two full filtration-enhanced graphs with $l \geq 2$ layers. If $\{C^i(u) \mid \forall u \in V_l(FEG)\} \neq \{C^i(u') \mid \forall u' \in V_l(FEG')\}$, then $\{C^i(u) \mid \forall u \in V(FEG)\} \neq \{C^i(u') \mid \forall u' \in V(FEG')\}$.*

Based on Lemma 2, it is sufficient to analyze the colors of $V_l(FEG)$ in the WL test.

Lemma 3. *For $\forall u_l \in V_l(FEG), u'_l \in V_l(FEG')$, the according vertices in original graph are $u \in G, u' \in G'$ respectively. If $C^i(u) \neq C^i(u')$, then $C^i(u_l) \neq C^i(u'_l)$*

Theorem 2. *If G and G' differ at the i-th iteration of WL test for the first time, FEG and FEG' differ at the i-th iteration of WL test at least.*

Theorem 3. *If G and G' differ with the WL test, FES and FES' also differ with the WL test.*

By Theorem 2 and Theorem 3, the expressive power of the full FEG or FES is at least as strong as the original graphs, no matter what filtration is defined on it. Since there exist graphs that cannot be distinguished by WL test but their corresponding full FEGs or FESes can be distinguished with an appropriate filtration method (as shown in Fig. 1), our FEG or FES based solutions can achieve strictly more expressive power compared to the approaches based on the original graphs.

3.3 Complexity Analysis

Recall that the filtration-enhanced graph divides edges of the original graph into l layers, and adds new edges called cross edges between layers if needed. In the worst case, there are n vertices and m edges in each layer of the full FEG and the number of cross edges is at most n between two adjacent layers. For partial FEG, the number of vertices size is at most $2m$ and the number of cross edges is bounded by $2m$ in the worst case (one edge per layer). There is no cross edge between different layers in FES, so the vertex size and edge size are at most $2m$ and m respectively. Clearly, the vertex size and edge size are $l \times n$ and $l \times m$ in the worst case for the full FES. In conclusion, the space complexity of full-FEG, partial-FEG, full-FES and partial-FES are $O(l \times m)$, $O(m)$, $O(l \times m)$ and $O(m)$ respectively. And the time complexity is same as the space complexity when ignoring the time taken for computing the filtration. Since we do not need too much layers in practice (e.g., no larger than 5 layers in most cases), the space and time complexity is linear w.r.t. the graph size.

3.4 Base Graph Kernels and GNNs

We apply the proposed FEG and FES to following three base graph kernels and three GNNs. Note that our methods are also able to use to other graph kernels and GNNs.

Base graph kernels. (1) Weisfeiler-Leman subtree kernel (WL) for a number of iterations counts pairs of matching subtree patterns in two graphs, while at each iteration updates colors of two graphs [Shervashidze *et al.*, 2011]. (2) Shortest path kernel (SP) counts pairs of shortest paths in two graphs that have the same source and sink labels and identical length [Borgwardt and Kriegel, 2005]. (3) Graphlet kernel (GL) counts identical pairs of graphlets with no larger than 5 vertices [Shervashidze *et al.*, 2009].

Base GNNs. GraphSAGE (SAGE) [Hamilton *et al.*, 2017], Graph Isomorphism Network(GIN) [Xu *et al.*, 2019] and GraphSNN (SNN) [Wijesinghe and Wang, 2022] are three graph neural networks proposed recently, which differ in message passing mechanism. (1) SAGE first aggregates message of neighbors evenly, and then concatenates message of neighbors and itself to derive the new message at next layer. (2) GIN adds learned parameter ϵ to the message of itself, and uses MLP to derive message at next layer, which was proved as powerful as 1-WL test in distinguishing graphs. (3) SNN calculates structural coefficient of edges to adjust message of itself and neighbors, which was also proved more powerful than 1-WL test in distinguishing graphs.

4 Experiments

In this section, we aims to evaluate the potential benefits of our filtration-enhanced graph transformation methods for graph classification tasks using graph kernels or GNNs. In particular, we address the following questions:
Q1: Does FEG or FES shows better results than the original graphs with the same base graph kernel or GNN?
Q2: Does FEG or FES with graph kernels and GNNs outperform the state-of-the-art (SOTA) methods in terms of graph classification accuracy?
Q3: How the number of layers in FEG or FES affects the performance on different methods?
Q4: How filtration used in FEG or FES affects the performance on different methods?

Our source code are available at https://github.com/BlockChanZJ/Filtration-Enhanced-Graph-Transformation.

Datasets. We use 7 benchmark attributed graph datasets including 3 datasets with native edge weights (BZR_MD, COX2_MD, ER_MD) and 4 datasets with continuous vertex attributes (BZR, DHFR, ENZYMES, PROTEINS). All these 7 benchmark datasets are widely used in graph classification studies [Kriege and Mutzel, 2012; O'Bray *et al.*, 2021]. Note that for all attributed graphs, we use the native attributes to derive a filtration. In addition, to answer **Q3**, we also use 7 benchmark non-attributed graph datasets derived from chemoinformatic (MUTAG, NCI1, D&D) [Kriege and Mutzel, 2012; Shervashidze *et al.*, 2011] and social networks (IMDB-BINARY, IMDB-MULTI, REDDIT-BINARY, REDDIT-MULTI-5K, abbreviated as IMDB-B, IMDB-M, RDT-B and RDT-5K respectively) [Yanardag and Vishwanathan, 2015], which are used to evaluate our algorithms with different edge weight functions defined in Section 2. All the datasets are available at ls11-www.cs.tu-dortmund.de/staff/morris/graphkerneldataset.

Settings for graph kernels. For graph kernels, we use C-SVM as a classifier and perform 10-fold cross-validation. The evaluation process was repeated 10 times for each dataset and each method. The parameter C of the SVM is tuned

		Native edge weights			Native vertex attributes			
		BZR_MD	COX2_MD	ER_MD	BZR	DHFR	ENZYMES	PROTEINS
SOTA	CSM	71.0 ± 0.5	OOT	OOT	78.8 ± 0.1	79.3 ± 1.3	49.8 ± 0.9	OOT
	WLOA	64.9 ± 0.9	58.5 ± 2.2	71.1 ± 1.1	80.6 ± 0.9	82.7 ± 0.5	60.1 ± 1.3	75.5 ± 0.3
	PM	67.7 ± 1.0	64.8 ± 0.4	73.3 ± 0.9	82.2 ± 1.0	79.0 ± 0.5	45.6 ± 1.3	74.8 ± 0.2
	MLG	64.3 ± 1.1	47.0 ± 3.0	65.2 ± 0.9	85.4 ± 1.2	80.9 ± 0.4	47.8 ± 0.9	74.9 ± 0.4
	FC-V	**76.5 ± 1.1**	73.2 ± 1.6	82.5 ± 1.1	85.8 ± 0.7	81.1 ± 0.4	55.5 ± 1.0	74.6 ± 0.6
WL	Original graph	60.0 ± 2.3	56.8 ± 2.4	67.5 ± 0.7	78.9 ± 0.3	81.1 ± 0.4	52.8 ± 0.9	76.0 ± 0.4
	Full FEG	70.1 ± 1.4	69.3 ± 1.4	79.5 ± 0.6	80.6 ± 0.7	81.8 ± 0.6	57.5 ± 1.5	76.7 ± 0.6
	Partial FEG	75.2 ± 2.7	71.9 ± 1.9	**82.5 ± 0.8**	**81.9 ± 0.3**	81.3 ± 0.8	**58.7 ± 1.1**	77.5 ± 0.2
	Full FES	69.1 ± 2.2	69.3 ± 1.4	78.2 ± 0.6	80.3 ± 0.5	**82.5 ± 1.0**	56.9 ± 1.1	76.8 ± 0.5
	Partial FES	**76.2 ± 2.7**	**72.8 ± 2.1**	81.6 ± 1.2	81.8 ± 0.6	81.7 ± 0.8	53.8 ± 0.7	**77.9 ± 0.5**
SP	Original graph	70.4 ± 1.1	61.4 ± 2.0	59.2 ± 2.0	81.3 ± 0.9	77.3 ± 0.8	40.9 ± 1.7	75.9 ± 0.4
	Full FEG	70.4 ± 1.5	72.7 ± 1.8	75.0 ± 0.7	81.2 ± 1.0	**83.5 ± 0.6**	58.0 ± 1.3	77.3 ± 0.6
	Partial FEG	71.4 ± 2.4	74.4 ± 1.4	**82.1 ± 0.6**	**87.4 ± 1.2**	80.8 ± 1.0	**62.6 ± 0.6**	**77.4 ± 0.2**
	Full FES	71.2 ± 1.6	73.9 ± 1.6	75.7 ± 1.0	82.0 ± 0.3	80.1 ± 0.5	52.3 ± 1.0	77.4 ± 0.6
	Partial FES	**74.4 ± 1.8**	**75.8 ± 2.4**	81.0 ± 0.7	79.8 ± 0.6	77.9 ± 0.6	50.9 ± 1.2	76.1 ± 0.5
GL	Original graph	49.6 ± 1.4	51.1 ± 1.9	59.4 ± 0.0	78.9 ± 0.1	61.0 ± 0.0	25.0 ± 1.1	71.6 ± 0.2
	Full FEG	59.9 ± 1.4	48.7 ± 1.8	66.6 ± 0.7	78.9 ± 0.2	61.0 ± 0.0	22.3 ± 0.8	71.8 ± 0.2
	Partial FEG	63.0 ± 1.9	55.1 ± 1.6	61.3 ± 1.5	79.2 ± 0.2	61.0 ± 0.0	25.7 ± 1.0	74.0 ± 0.4
	Full FES	63.9 ± 0.3	56.1 ± 2.9	68.6 ± 0.9	79.2 ± 0.2	61.0 ± 0.0	25.6 ± 1.2	73.9 ± 0.5
	Partial FES	**68.2 ± 2.2**	**63.8 ± 1.1**	**76.1 ± 0.8**	**80.7 ± 0.2**	**76.4 ± 0.9**	**37.4 ± 1.0**	**75.9 ± 0.3**

Table 1: Comparison between SOTA methods and the best results of three basic graph kernels with filtration by native attributes. We report the average of 10 runs of 10-fold CV, with the best results in red. OOT means that the method did not finish in 24 hours.

		Native edge weights			Native vertex attributes			
		BZR_MD	COX2_MD	ER_MD	BZR	DHFR	ENZYMES	PROTEINS
GIN	Original graph	69.3 ± 11.0	61.0 ± 9.7	72.7 ± 7.8	84.8 ± 3.8	74.0 ± 2.9	54.2 ± 7.7	72.1 ± 5.3
	Full FEG	72.3 ± 7.3	70.3 ± 7.7	76.6 ± 5.8	85.5 ± 3.8	**77.3 ± 4.5**	**60.2 ± 8.1**	**74.8 ± 4.4**
	Partial FEG	**76.0 ± 6.8**	**76.0 ± 8.3**	**79.3 ± 4.1**	**86.5 ± 4.8**	77.2 ± 4.4	59.7 ± 4.9	73.9 ± 4.1
SAGE	Original graph	70.0 ± 6.5	55.3 ± 13.4	67.7 ± 8.6	79.2 ± 5.8	67.9 ± 7.1	41.5 ± 7.9	75.1 ± 4.1
	Full FEG	71.0 ± 6.3	**65.7 ± 11.7**	70.9 ± 8.7	80.0 ± 5.9	64.1 ± 5.0	**42.8 ± 10.2**	**75.5 ± 3.7**
	Partial FEG	**72.3 ± 7.2**	63.7 ± 11.2	**74.6 ± 7.0**	**81.5 ± 3.9**	**67.9 ± 4.7**	33.0 ± 5.4	75.3 ± 3.9
SNN	Original graph	67.7 ± 9.8	62.7 ± 7.9	72.0 ± 5.8	86.5 ± 5.2	75.7 ± 4.1	59.2 ± 6.4	74.8 ± 5.4
	Full FEG	71.3 ± 7.0	76.3 ± 3.5	76.1 ± 6.8	85.5 ± 5.0	78.3 ± 3.0	**61.8 ± 6.6**	74.9 ± 4.4
	Partial FEG	**74.3 ± 6.3**	**77.0 ± 6.7**	**78.9 ± 5.6**	**86.5 ± 4.4**	**79.5 ± 5.5**	60.7 ± 7.2	**75.5 ± 5.1**

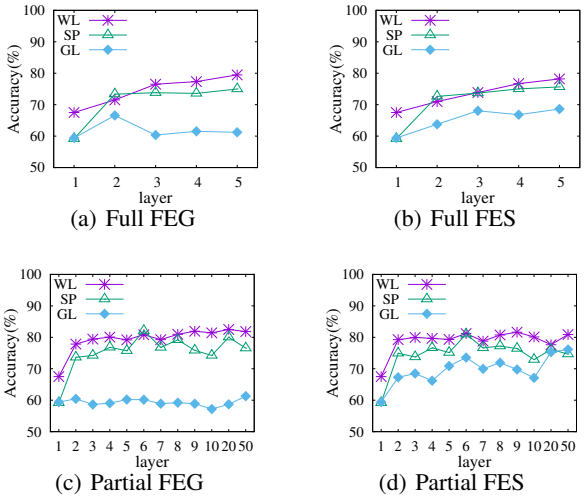
Table 2: Comparison of different GNNs on attributed graphs. GIN and SNN with original graphs denote the existing SOTA methods.

from $\{10^{-3}, \cdots, 10^{3}\}$. The layers of FEG/FES are chosen from $\{2, \cdots, 5\}$ for full FEG/FES, and $\{2, \cdots, 10, 20, 50\}$ for partial FEG/FES. All graph kernels are implemented in Python using the GraKeL library [Siglidis *et al.*, 2020]. The parameters of graph kernels were chosen as follows. For Weisfeiler-Leman subtree kernel, we chose the number of iterations from $\{1, \cdots, 7\}$. For graphlet kernel, we sampled 500 graphlets of size up to 5. For shortest path kernel, all edge weight equals to 1. We compare our methods with most existing SOTA graph kernels as compared in [O'Bray *et al.*, 2021] which include subgraph matching kernel (CSM, $k \in \{3, 4, 5\}$) [Kriege and Mutzel, 2012], Weisfeiler-Leman optimal assignment kernel (WLOA, $h \in \{1, \cdots, 7\}$) [Kriege *et al.*, 2016], multiscale Laplacian graph kernel (MLG, $\gamma, \eta \in \{0.1, 0.01\}$) [Kondor and Pan, 2016], pyramid matching kernel (PM,$L \in \{2, 4, 6\}, d \in \{4, 6, 8, 10\}$) [Nikolentzos *et al.*, 2017b] and filtration curves (FC) [O'Bray *et al.*, 2021].

Settings for GNNs. We mainly evaluate FEGs with three basic graph neural networks described in Section 3.4 including GraphSAGE [Hamilton *et al.*, 2017], GIN [Xu *et al.*, 2019], GraphSNN [Wijesinghe and Wang, 2022], and the results of FESes are similar. Note that GIN and GraphSNN are the SOTA GNN methods for graph classification as reported in [Xu *et al.*, 2019; Wijesinghe and Wang, 2022]. The layers of FEG are chosen from $\{2, \cdots, 5\}$ for full FEG, and $\{2, \cdots, 5, 10\}$ for partial FEG. For a fair comparison, we follow the standard data splits in [Errica *et al.*, 2020]. We report the best result of all layers and filtration. The parameters of GNNs are chosen as follows. All three GNNs are trained for 500 epochs with 50 epoch patience to early stop and hidden units of 64.The convolution layer numbers are selected from $\{2, 3, 4\}$. For GraphSAGE and GIN, we set the learning rate parameter as 0.001, batch size as 128, and dropout is chosen from $\{0, 0.5\}$. For GraphSNN, we use dropout of 0.5, batch size of 64 and learning rate chosen from $\{0.01, 0.001\}$. For GNNs, the graph representation is derived from sum pooling and mean pooling.

4.1 Experimental Results

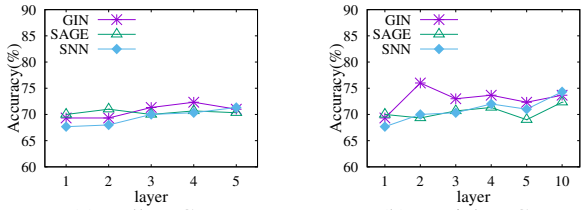
First, to address **Q1**, we provide a comparison of results on three base graph kernels (Table 1) and three base GNNs (Table 2). Table 1 shows that FEG or FES based graph kernels consistently outperform the original-graph based graph kernels on attributed graphs. The results on non-attributed graphs are consistent, which are given in the supplementary document due to the space limit. For graphs with native edge weights, all base graph kernels have more than 10% accuracy gains with our FEG or FES technique. For example, the classification accuracy of the partial FES based WL kernel is 76.2% while the WL kernel with the original graphs can only achieve a 60% accuracy on BZR_MD. Similarly, for graphs with native vertex attributes, our solutions are also consistently better than the base kernels with the original graphs



(a) Full FEG (b) Full FES

(c) Partial FEG (d) Partial FES

Figure 3: Results of graph kernels with various layers on ER_MD



(a) Full FEG (b) Partial FEG

Figure 4: Results of GNNs with various layers on BZR_MD

(more than 10% accuracy gains on ENZYMES). As shown in Table 2, the results for GNNs are consistent with the results for graph kernels. These results indicate that our FEG transformation technique can indeed enhance the expressive power of graph kernels or GNNs in distinguishing graphs.

Second, to address **Q2**, we compare our best method with the SOTA methods in terms of classification accuracy. The results for graph kernels and GNNs are also shown in Table 1 and Table 2, respectively. As can be seen, our best method significantly outperforms all SOTA methods on all datasets except for BZR_MD. For example, on BZR dataset, our best algorithm can achieve a 87.4% accuracy, while the accuracy of the best existing algorithm is 85.8%. Moreover, even on BZR_MD, our best method is still competitive with the SOTA method (the accuracy of our best algorithm and the SOTA algorithm is 76.2 and 76.5 respectively). Likewise, our best method consistently beats the SOTA GNNs on all datasets, including GIN and GraphSNN, as shown in Table 2. The reason could be that the graph augmented by our FEG transformation not only preserves the original graph structural features, but it can also capture the topological feature which may be useful for graph classification. These results further confirm the superiority of our solution.

Third, to address **Q3**, we provide the results of different layers for all base kernels on ER_MD (Fig. 3) and for all base GNNs on BZR_MD (Fig. 4). The results on the other datasets are consistent. For graph kernels, the accuracy increases when the number of layers l increases for both full FEG and full FES as shown in Figs. 3(a-b). When $l \geq 2$, the increasing trend tends to be smooth. However, as shown in Figs. 3(c-d), the accuracy is relatively robust w.r.t. l for partial FEG and partial FES when $l \geq 2$. For GNNs, the accuracy seems to be not very sensitive w.r.t. l for both full and partial FEG. These results suggest that the hyper-parameter l in our methods is easy to tune for practical applications.

	Datasets	CN	CORE	RC
WL kernel	MUTAG	-	**90.6 ± 0.8**	88.7 ± 0.9
	IMDB-B	**77.3 ± 0.3**	74.7 ± 0.6	75.2 ± 0.4
	IMDB-M	52.1 ± 0.5	**52.2 ± 0.5**	51.2 ± 0.6
	NCI1	**85.9 ± 0.2**	85.1 ± 0.2	85.7 ± 0.2
	D&D	80.2 ± 0.5	**80.2 ± 0.2**	80.1 ± 0.3
	RDT-B	82.2 ± 0.5	82.8 ± 0.2	**85.5 ± 0.3**
	RDT-5K	53.2 ± 0.2	**54.3 ± 0.2**	53.4 ± 0.1
GIN	PROTEINS	**74.8 ± 3.8**	74.5 ± 3.9	73.9 ± 4.1
	IMDB-B	73.8 ± 3.0	73.5 ± 3.1	**73.9 ± 2.9**
	IMDB-M	**51.0 ± 3.4**	50.9 ± 4.0	49.1 ± 4.5
	NCI1	79.3 ± 1.6	78.8 ± 2.3	**79.5 ± 1.8**
	D&D	**78.0 ± 3.9**	76.8 ± 3.7	77.1 ± 5.2

Table 3: Results of different filtration for WL kernel and GIN. CN, CORE and RC represent common neighbor, core number and ricci-curvature respectively. "-" means no common neighbor in MUTAG.

Finally, to answer **Q4**, we show the results of three different filtration methods used in FEG transformation for WL kernel and GIN (Table 3). Similar results can also be observed for the other base graph kernels and GNNs. As reported in Table 3, different filtration methods may affect the accuracy for both graph kernels and GNNs. For the WL kernel, the core number based filtration generally performs better than the others. However, for GIN, the core number based filtration is less accurate than the other filtration methods. These results indicate that the filtration method is important for our approach to achieve a good classification accuracy. The three heuristic filtration methods introduced in Section 2 perform well in practice. An interesting future direction is to seek a better filtration for FEG transformation by machine learning methods (like [Hofer *et al.*, 2020]).

5 Conclusion

In this paper, we proposed a filtration-enhanced graph (FEG) transformation technique which enables a general approach to enhance the expressive power of graph kernels and GNNs. We prove that our FEG based approaches have strictly better expressive power over the original graph based solutions for graph classification tasks. A nice feature of our technique is that it is a plug-in method and it can be used to any graph kernel and GNN to enhance their expressiveness. Moreover, the FEG transformation can be computed in linear time using linear space. Extensive experiments over several benchmark datasets indicate that our solutions significantly outperform all the SOTA methods for graph classification tasks.

Acknowledgments

This work was partially supported by (i) National Key Research and Development Program of China 2020AAA0108503, (ii) NSFC Grants 62072034 and U1809206, (iii) "Research on Key Technologies of trusted data sharing under the integration of blockchain and IPFS" Project in 2021 of SIT under Grant 2111010, Guangdong Philosophy and Social Sciences Planning Project (GD21CYJ21). Rong-Hua Li is the corresponding author of this paper.

References

[Abboud *et al.*, 2021] Ralph Abboud, İsmail İlkan Ceylan, Martin Grohe, and Thomas Lukasiewicz. The surprising power of graph neural networks with random node initialization. In *IJCAI*, 2021.

[Borgwardt and Kriegel, 2005] Karsten M. Borgwardt and Hans-Peter Kriegel. Shortest-path kernels on graphs. In *ICDM*, 2005.

[Bouritsas *et al.*, 2020] Giorgos Bouritsas, Fabrizio Frasca, Stefanos Zafeiriou, and Michael M. Bronstein. Improving graph neural network expressivity via subgraph isomorphism counting. *CoRR*, abs/2006.09252, 2020.

[Chiba and Nishizeki, 1985] Norishige Chiba and Takao Nishizeki. Arboricity and subgraph listing algorithms. *SIAM Journal on computing*, 14(1):210–223, 1985.

[Du *et al.*, 2019] Simon S. Du, Kangcheng Hou, Ruslan Salakhutdinov, Barnabás Póczos, Ruosong Wang, and Keyulu Xu. Graph neural tangent kernel: Fusing graph neural networks with graph kernels. In *NeurIPS*, 2019.

[Edelsbrunner *et al.*, 2002] Herbert Edelsbrunner, David Letscher, and Afra Zomorodian. Topological persistence and simplification. *Discret. Comput. Geom.*, 28(4):511–533, 2002.

[Errica *et al.*, 2020] Federico Errica, Marco Podda, Davide Bacciu, and Alessio Micheli. A fair comparison of graph neural networks for graph classification. In *ICLR*, 2020.

[Hamilton *et al.*, 2017] William L. Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *NIPS*, 2017.

[Hofer *et al.*, 2020] Christoph D. Hofer, Florian Graf, Bastian Rieck, Marc Niethammer, and Roland Kwitt. Graph filtration learning. In *ICML*, 2020.

[Horváth *et al.*, 2004] Tamás Horváth, Thomas Gärtner, and Stefan Wrobel. Cyclic pattern kernels for predictive graph mining. In *KDD*, 2004.

[Kondor and Pan, 2016] Risi Kondor and Horace Pan. The multiscale laplacian graph kernel. In *NIPS*, 2016.

[Kriege and Mutzel, 2012] Nils M. Kriege and Petra Mutzel. Subgraph matching kernels for attributed graphs. In *ICML*, 2012.

[Kriege *et al.*, 2016] Nils M. Kriege, Pierre-Louis Giscard, and Richard C. Wilson. On valid optimal assignment kernels and applications to graph classification. In *NIPS*, 2016.

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

[Lee *et al.*, 2019] Junhyun Lee, Inyeop Lee, and Jaewoo Kang. Self-attention graph pooling. In *ICML*, 2019.

[Lin *et al.*, 2011] Yong Lin, Linyuan Lu, and Shing-Tung Yau. Ricci curvature of graphs. *Tohoku Mathematical Journal, Second Series*, 63(4):605–627, 2011.

[Matula and Beck, 1983] David W. Matula and Leland L. Beck. Smallest-last ordering and clustering and graph coloring algorithms. *J. ACM*, 30(3):417–427, 1983.

[Morris *et al.*, 2017] Christopher Morris, Kristian Kersting, and Petra Mutzel. Glocalized weisfeiler-lehman graph kernels: Global-local feature maps of graphs. In *ICDM*, 2017.

[Morris *et al.*, 2020] Christopher Morris, Gaurav Rattan, and Petra Mutzel. Weisfeiler and leman go sparse: Towards scalable higher-order graph embeddings. In *NeurIPS*, 2020.

[Nikolentzos *et al.*, 2017a] Giannis Nikolentzos, Polykarpos Meladianos, François Rousseau, Yannis Stavrakas, and Michalis Vazirgiannis. Shortest-path graph kernels for document similarity. In *EMNLP*, 2017.

[Nikolentzos *et al.*, 2017b] Giannis Nikolentzos, Polykarpos Meladianos, and Michalis Vazirgiannis. Matching node embeddings for graph similarity. In *AAAI*, 2017.

[O'Bray *et al.*, 2021] Leslie O'Bray, Bastian Rieck, and Karsten Borgwardt. Filtration curves for graph representation. In *KDD*, 2021.

[Sato *et al.*, 2021] Ryoma Sato, Makoto Yamada, and Hisashi Kashima. Random features strengthen graph neural networks. In *SDM*, 2021.

[Seidman, 1983] Stephen B Seidman. Network structure and minimum degree. *Social networks*, 5(3):269–287, 1983.

[Shervashidze *et al.*, 2009] Nino Shervashidze, S. V. N. Vishwanathan, Tobias Petri, Kurt Mehlhorn, and Karsten M. Borgwardt. Efficient graphlet kernels for large graph comparison. In *AISTATS*, 2009.

[Shervashidze *et al.*, 2011] Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M. Borgwardt. Weisfeiler-lehman graph kernels. *J. Mach. Learn. Res.*, 12:2539–2561, 2011.

[Siglidis *et al.*, 2020] Giannis Siglidis, Giannis Nikolentzos, Stratis Limnios, Christos Giatsidis, Konstantinos Skianis, and Michalis Vazirgiannis. Grakel: A graph kernel library in python. *Journal of Machine Learning Research*, 21(54):1–5, 2020.

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

[Wijesinghe and Wang, 2022] Asiri Wijesinghe and Qing Wang. A new perspective on "how graph neural networks go beyond weisfeiler-lehman?". In *International Conference on Learning Representations*, 2022.

[Xu *et al.*, 2019] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *ICLR*, 2019.

[Yanardag and Vishwanathan, 2015] Pinar Yanardag and S. V. N. Vishwanathan. Deep graph kernels. In *KDD*, 2015.

[You *et al.*, 2021] Jiaxuan You, Jonathan M. Gomes-Selman, Rex Ying, and Jure Leskovec. Identity-aware graph neural networks. In *AAAI*, 2021.

[Zhang *et al.*, 2018] Zhen Zhang, Mianzhi Wang, Yijian Xiang, Yan Huang, and Arye Nehorai. Retgk: Graph kernels based on return probabilities of random walks. In *NeurIPS*, 2018.

[Zhao and Wang, 2019] Qi Zhao and Yusu Wang. Learning metrics for persistence-based summaries and applications for graph classification. In *NeurIPS*, 2019.