Filtration Curves for Graph Representation

Leslie O'Bray*
ETH Zürich
SIB Swiss Institute of Bioinformatics
Switzerland
leslie.obray@bsse.ethz.ch

Bastian Rieck*
ETH Zürich
SIB Swiss Institute of Bioinformatics
Switzerland
bastian.rieck@bsse.ethz.ch

Karsten Borgwardt
ETH Zürich
SIB Swiss Institute of Bioinformatics
Switzerland
karsten.borgwardt@bsse.ethz.ch

ABSTRACT

The two predominant approaches to graph comparison in recent years are based on (i) enumerating matching subgraphs or (ii) comparing neighborhoods of nodes. In this work, we complement these two perspectives with a third way of representing graphs: using *filtration curves* from topological data analysis that capture both edge weight information and global graph structure. Filtration curves are highly efficient to compute and lead to expressive representations of graphs, which we demonstrate on graph classification benchmark datasets. Our work opens the door to a new form of graph representation in data mining.

CCS CONCEPTS

• **Mathematics of computing** → **Graph algorithms**; • **Computing methodologies** → **Machine learning approaches**.

KEYWORDS

Graph classification; graph representation

ACM Reference Format:
Leslie O'Bray, Bastian Rieck, and Karsten Borgwardt. 2021. Filtration Curves for Graph Representation. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '21), August 14–18, 2021, Virtual Event, Singapore.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3447548.3467442

1 INTRODUCTION

The search for ways to efficiently compare graphs is one of the classic tasks in data mining. This line of research is based on several decades of work in chemoinformatics, which brought about approaches based on graph isomporphism testing [30, 31], graph edit distances [7, 29], topological descriptors [5, 14], and later, frequent subgraph mining [26]. Over the last two decades, however, two alternative approaches have dominated this field: graph kernels [4, 22] and graph neural networks [37]. While many different flavors of both approaches exist, they are primarily based on (i) either enumerating matching subgraphs in two graphs to determine similarity or (ii) comparing (direct and higher-order) neighborhoods of all pairs of nodes in two graphs. A limitation of the approaches based on (i) is that enumeration approaches have difficulty including edge weight information, which is important in many application domains. A limitation of the approaches based on (ii) is that neighborhoods capture little information about the global structure of a graph, which also matters in many applications. Despite ongoing research to overcome these issues, the literature lacks an efficient-to-compute and powerful graph representation that can take edge weights and global graph structure into account. In this paper, we fill this gap by proposing a curve-based representation of a graph, which we term *filtration curves*.

Filtration curves are inspired by filtrations, a well-known concept from topological data analysis, where they typically occur in the context of persistent homology [2, 11]. While persistent homology is a powerful framework that has proven to be expressive and useful for graph classification, it imposes a larger inductive bias on classifiers, thus impeding its use in arbitrary neural network architectures. We assume a more generic view in this paper and disentangle the concept of graph filtrations from persistent homology, obtaining a more generic formulation in terms of *graph descriptor functions* for graphs. This perspective results in a surprisingly effective graph representation algorithm, which achieves the best cross-dataset performance compared to the more complicated state-of-the-art (SOTA) graph classification methods. Our approach is easy to implement[1] and can be completely parameter-free, giving rise to a new class of graph representation schemes.

2 RELATED WORK

The field of graph classification has seen increasing importance over the last two decades, resulting in a plethora of available methods. Ranging from *graph kernels* [4, 22], a mathematically principled way of addressing graph classification via (implicit) embeddings in Reproducing Kernel Hilbert Spaces (RKHS), to *graph neural networks* (GNNs) [18, 37], a family of neural networks based on message passing on graphs, numerous methods have been proposed, and it is hard to do proper justice to the richness of this field. Despite the large number of methods available, there are few approaches that are capable of dealing *explicitly* with the multi-scale structure of weighted graphs. While there have been many GNN methods in recent years, we highlight the Graph Isomorphism Network (GIN) [38] specifically, since it provides a theoretical framework for understanding the expressive power of GNNs and links that to the expressive power of the famous Weisfeiler–Lehman relabeling scheme and

KDD '21, August 14–18, 2021, Virtual Event, Singapore

ACM ISBN 978-1-4503-8332-5/21/08…$15.00
https://doi.org/10.1145/3447548.3467442

[1] Our code and data are available at https://github.com/BorgwardtLab/Filtration_Curves.

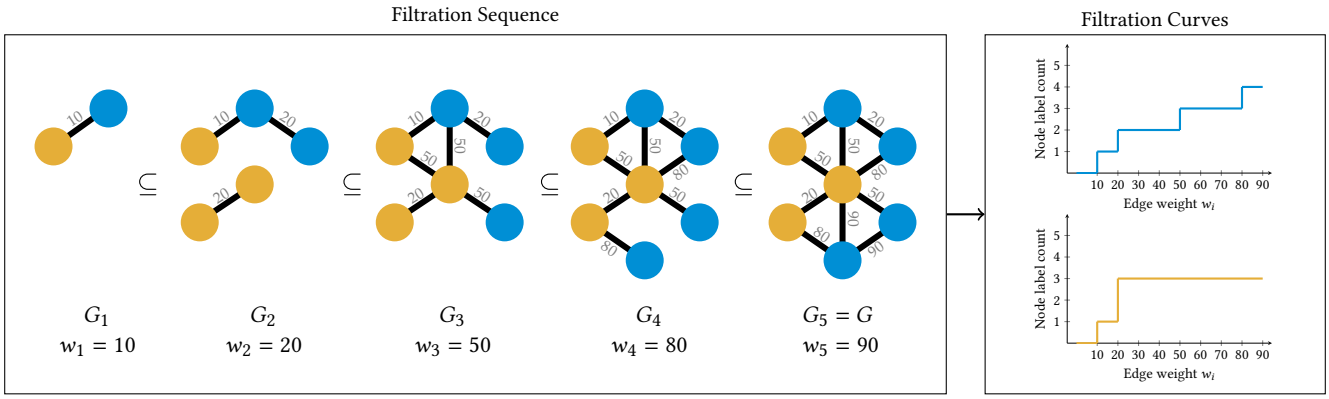


Figure 1: Our approach uses edge weights to transform a graph into a filtration ordering, i.e., a nested sequence of subgraphs. We evaluate a graph descriptor function f alongside this filtration to obtain a set of filtration curves for graph representation tasks. Here, we show the process of generating a filtration curve using the node label histogram descriptor function (FC-V) for a graph with native edge weights and node labels. In this example, the count of each node label is tracked in each graph in the filtration sequence, resulting in a filtration curve for each unique node label. The x-axis of the curve represents the edge weight w_i, and the y-axis represents the count of the respective node label present in the subgraph G_i, which is induced by all edges whose weight are less than or equal to w_i.

corresponding classification algorithm [30, 31]. The GIN is a useful GNN comparison partner since their work also describes an optimal configuration of GNNs for graph classification, and thus is a state-of-the-art GNN method. However, our method achieves better results than the GIN while being either parameter-free or having at most one parameter, and is a non-neighborhood based approach.

Recently, methods from computational topology started to exploit the multi-scale nature of graphs, with several approaches handling graphs through static topological descriptors [27, 40], or via topology-based neural networks [8, 16, 17], all making use of different kinds of filtrations, which we shall subsequently discuss. An approach by Hofer et al. [17] is particularly relevant as it *learns* a filtration end-to-end for graph classification. However, it requires an additional GNN-based initialization and is currently restricted to vertex-based and scalar-valued filtrations, displaying competitive performance for the analysis of unlabeled graphs.

Our approach targets edge-based filtrations of graphs, and thus can be used in both labeled and unlabeled graphs. Furthermore, we are not restricted to the framework of persistent homology, nor do we require the use of graph neural networks. Our method provides a new, more general formulation to the field. It thus naturally complements filtration learning, and future work might focus on a *hybrid*. Hence, while our approach borrows the concept of a filtration from computational topology, it replaces the use of topological descriptors with more generalized graph descriptor functions such as histogram functions, which we focus on in this paper.

While graph kernels, GNNs, and computational topology methods have seen good performance, they present some drawbacks. Graph kernels can suffer from a computational complexity that is quadratic in the size of the dataset, since kernels typically require a pairwise comparison of each graph in the dataset (unless graph kernels with explicit feature vectors are being employed). GNNs can leverage faster computing resources found in a GPU, but still require a large quantity of parameters to be tuned, and can have empirically long runtimes when trained using only CPUs. Moreover, they are known to suffer from oversmoothing [9]. Computational topology methods often exhibit substantial complexity, both in terms of computational aspects, but also in terms of assumptions about the data. They are frequently presented without a clear picture of *which* aspect of the complexity—e.g., the choice of topological features or the selected representations—drives performance.

By contrast, our approach offers a fast, lightweight alternative to existing methods, and we find it to perform comparably or even better than the comparison partners, which comprise graph kernels, GNNs, and topological methods.

3 METHODS

We refer to *labeled* graphs as those containing discrete node labels in addition to continuous edge or node attributes, and *unlabeled* graphs as those which do not contain any discrete or continuous node or edge attributes. Given a graph $G = (V, E)$ with an edge weight function $\mathrm{w}\colon E \to \mathbb{R}$, a *filtration* [11] is a series of monotonically-growing subgraphs $\emptyset \subseteq G_1 \subseteq G_2 \cdots \subseteq G_m \subseteq G$ that defines a decomposition, which can be understood as thresholding a graph. Without loss of generality, we assume that $\mathrm{w}(\cdot)$ only takes on a finite number of non-decreasing values $w_1 \leq \cdots \leq w_m$. Then, G_i (the i^{th} graph in the filtration) is the subgraph induced by all edges whose weight is less than or equal to w_i, i.e., $G_i := (V', E')$, with $E' := \{(u, v) \in E \mid \mathrm{w}(u, v) \leq w_i\}$ and $V' := \{v \in V \mid (u, v) \in E' \text{ or } (v, u) \in E'\}$. Intuitively, a filtration observes the graph as each edge is added one at a time, thus turning any graph into a *growth process*. Using a fixed filtration, assuming that two graphs are generated by a similar process, similar substructures will tend to appear at similar "times." If weights are available, creating a filtration is computationally as efficient as *sorting* all edges. We will

also write $\mathcal{F}_G$ as a shorthand for the sequence of graphs that occur in a filtration of a graph G.

3.1 Generating Filtration Curves

The generation of a *filtration curve* requires two core components: choosing a weight function to define the filtration and a graph descriptor function that will be used to build the curve alongside that filtration. Equipped with these two things, the process decomposes into a few steps. First, we convert the graph to a filtration of graphs using the designated edge weight function. Second, we evaluate the graph descriptor function on each subgraph in the filtration. Third, we generate a curve for each graph using the values of the edge weights in the filtration and the values from the graph descriptor function. An example is presented in Figure 1.

In this section we will detail effective filtrations and descriptor functions for two typical scenarios in graph classification: one for labeled graphs and one for unlabeled graphs. We then show how one can construct a *filtration curve* from such a process that can be used for classification.

Creating such a curve is of interest for two key reasons. First, it allows easy and straightforward standardization of graphs, enabling a graph to be vectorized, which enables their use with classic machine learning algorithms. Second, the representation of a graph as a curve has beneficial theoretical properties. Both of these will be considered in more detail in Section 3.2.

3.1.1 Defining a Filtration. We defined a filtration at the start of the section, but did not explicitly mention how one chooses the edge weight function w that defines the filtration. In this section we will introduce different various ways to define a filtration on a graph, which provides the critical first step in generating a filtration curve. We will detail the following weight functions that can be assigned to edges in order to build a filtration: native edge weights, max degree, Ricci curvature, and the heat kernel signature. We apply the first to labeled graphs, since it is an inherent feature in the dataset, and then discuss the latter three as ways to build a filtration when there are no attributes present. Each weight function has its own properties and expressive power, which we shall briefly discuss.

Native edge weights. For edge-weighted graphs, the edge weights themselves provide a natural way to build a filtration. One can easily extend this to graphs with continuous node attributes, but no edge weight, by assigning an edge weight equal to the Euclidean distance between the attributes of the two nodes.

Max degree. One of the simplest filtrations assigns an edge weight w_{ij} between nodes i and j using the degree function of the two nodes incident to it: $w_{ij} = \max(\text{degree}(i), \text{degree}(j))$. The degree filtration is typically employed in computational topology to classify unlabeled graphs; it was seen to be particularly effective for the analysis of social network datasets, and can be seen as a suitable "first try" because degree calculations presuppose no additional information. At the same time, this filtration is limited in expressive power because it can only incorporate information about degrees, but no further information about vertex neighborhoods.

Ricci curvature. Following previous work [39, 40], one can endow the edges of the graph with a weight by means of the *Ricci curvature* [23]. Curvature is a characteristic invariant property arising from manifold geometry, which focuses on intrinsic properties of a manifold and assesses to what extent points of the manifold can be described by model geometries, such as surfaces. Graph Ricci curvature is a graph isomorphism invariant that assesses how much deviation there is from the "flat" case, i.e., a grid graph, by assessing the similarity of the neighborhoods between two nodes. To see the correspondence to the manifold case, we note that a tree, i.e., an acyclic graph, where two nodes have no overlapping neighbors, has *negative* curvature, whereas a complete graph, where all nodes are connected to all others, has *positive* curvature. There are multiple ways of defining curvature; we follow the definition by Ollivier–Ricci. Formally, we turn the graph into a metric space with distance d. There are several ways of defining such a distance, but when working with unlabeled graphs, the shortest path distance is a common choice. For a fixed base node $x \in V$, we then calculate a probability measure m_x^α via

$$m_x^\alpha(v) = \begin{cases} \alpha & v = x \\ \frac{1-\alpha}{\text{degree}(x)} & v \in \mathcal{N}(x) \\ 0 & otherwise, \end{cases} \tag{1}$$

where $\mathcal{N}(x)$ denotes the set of neighbors of x. Intuitively, the α parameter controls how much mass a given node keeps for itself as opposed to distributing it to its neighbors. We then obtain the Olivier–Ricci curvature of an edge between two nodes x and y via the Wasserstein distance $W(\cdot,\cdot)$ between their respective probability measures:

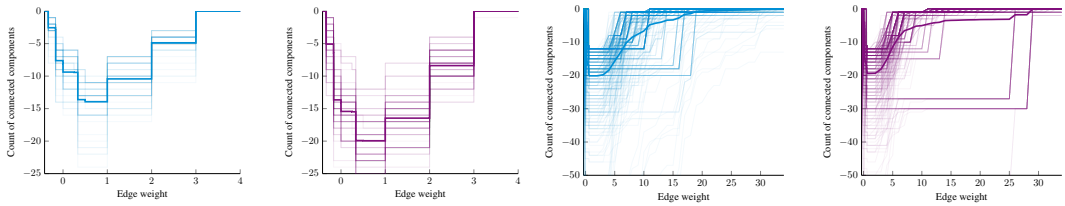
$$\kappa_\alpha(x,y) = 1 - \frac{W\left(m_x^\alpha, m_y^\alpha\right)}{d(x,y)} \tag{2}$$

The Wasserstein distance is a well-known concept from Optimal Transport theory, often also known as the Earth Mover's Distance. It affords efficient implementations, which can be approximated in linear time [1, 24], making Ricci curvature calculations highly scalable. In this paper, we set $\alpha = 0.5$, as this distributes probability mass equally over a vertex and its neighbors.

Heat kernel signature. Another option is the heat kernel signature, which is derived from the heat diffusion equation on graphs. Following the notation of Carrière et al. [8], the heat kernel signature can be calculated for a node v of a graph G via

$$\text{hks}(G,t,v) = \sum_{i=1}^{n} \exp(-t\lambda_i)\psi_i(v)^2, \tag{3}$$

where $t \in \mathbb{R}$ is the diffusion parameter, λ_i and $\psi_i(v)$ are the i^{th} eigenvalues and eigenfunctions of the graph Laplacian. The graph Laplacian, defined using the (weighted) adjacency matrix, is known to carry a substantial amount of shape information of graphs [15, 35]. The heat kernel signature therefore characterizes the structural role of vertices, making it a highly expressive summary to calculate. It requires a full eigendecomposition of a matrix, however, which could lead to scalability issues. Following the previous equation, we provide edge weights again as the maximum of the values that were computed on the nodes incident to it.



(a) MUTAG **(b) IMDB-BINARY**

Figure 2: Filtration curves form a vector space, meaning that addition and scalar multiplication are well defined. It is therefore possible to create a mean filtration curve of a dataset. Here we stack all the filtration curves of the graphs in the MUTAG and IMDB-BINARY datasets, where blue is one class, and purple is the other. The mean filtration curve of each class is identified by the thick, darker line. The difference between the two classes are visible in the mean filtration curve.

3.1.2 Choosing a Graph Descriptor Function. Once the edge weights have been defined on a graph, we can induce a filtration. The remaining step is to define a graph descriptor function that will be evaluated alongside the filtration, and from which we will generate the filtration curve itself.

Specifically, we require a function $f\colon G \to \mathbb{R}^d$ that evaluates certain attributes of a graph and embeds them into a d-dimensional real-valued space. We refer to this as a *graph descriptor function*. Given f and a filtration $\mathcal{F}_G$, we can evaluate f for each subgraph G_i of the filtration and represent G as a high-dimensional *path* via

$$\mathcal{P}_G := \bigoplus_{i=1}^{m} f(G_i) \in \mathbb{R}^{m \times d}, \tag{4}$$

where m indexes the number of thresholds (we will use this term interchangeably with edges or edge weights) in the filtration $\mathcal{F}_G$, and $\oplus$ refers to the concatenation operator. The resulting path can equivalently be seen as an $m \times d$ matrix whose *rows* correspond to filtration steps (which we will also refer to as thresholds) and whose *columns* correspond to features. The high-dimensional path $\mathcal{P}_G$ describes the evolution of a certain descriptor during the graph growth process; it can also be considered as a functional summary of G [3], from which we can obtain a curve.

These curves result in a sparse representation of a graph, as the filtration curve only changes at the values of the edge weights, i.e., the thresholds, and therefore only requires at most m points to fully describe the curve, which is already a sufficient representation for comparison. However, it is also easy to extend Eq. 4 to a standardized representation of all graphs in a dataset, which can be beneficial in certain scenarios. One can achieve this by creating a shared sorted index of all unique thresholds that exist in the dataset. While an individual graph will initially only have values of the graph descriptor function at the thresholds that exist in that graph itself, we can fill in the remaining values between the graph's thresholds with the previous value, due to the fact that the thresholds are ordered (i.e., *forward filling*). Thus, the value of the graph descriptor function of a single graph will not change until the next threshold from the graph in question is reached. While not strictly necessary to perform graph comparison, the ability to standardize the size of the graph representation unlocks a wealth of methods to the practitioner. We will subsequently discuss two possible choices of a graph descriptor function, one for labeled graphs and one for unlabeled graphs, which we use in this paper.

Node label histogram. A simple choice for f in labeled graphs is a node label histogram function, depicted in Figure 1. At each step of the filtration, one tracks the number of each node label that is present in the graph, resulting in a multi-scale descriptor of the graph, which is more descriptive than merely calculating a *single* histogram of the node labels at the end of a filtration. As seen in Figure 1, the high-dimensional path from Eq. 4 can be represented as d scalar-valued curves, resulting in one curve per unique node label over the set of all values in the filtration.

Count of connected components. When working with unlabeled graphs, we cannot use the node label histogram as before. Instead, we consider the points, i.e., *thresholds*, in the filtration at which a connected component is either created or destroyed in the graph, and use the count of such connected components as our descriptor. We define a connected component as being created when the filtration value is equal to its node degree, and the destruction point as being equal to the edge weight in which the component was merged with another connected component. As one moves along the filtration, the number of connected components increases at a node's creation point, as an isolated node constitutes a connected component, and decreases at its destruction point, since the node is now connected to another connected component. This particular representation then only requires tracking the thresholds at which there is a creation or destruction point, since at the other values the number of connected components does not change, leading to a further sparsified representation.

3.1.3 Connection to Persistent Homology. There is a relationship between the filtration curve and established concepts from topological data analysis, namely persistent homology and the Betti curve, also known as the Persistence Indicator Function [28, 36]. Indeed, one may see the Betti curve as a specific instance of our approach when analyzed with the proper descriptor. For example, the persistence diagram is represented by a set of tuples (c, d) marking the creation and destruction point of a given connected component. Node weights are commonly set to zero, meaning that at threshold zero, each node is created and forms its own unique connected component. In such a scenario, the list of tuples all take the form $(0, d)$ and thus connected components are only ever destroyed.

While the Betti curve from persistent homology is also of interest, we find our more general formulation compelling for a few reasons. First, it allows us to measure various topological information with

a given filtration, and therefore is more general than the persistence diagram and does not inherit any implicit biases. Second, the persistence diagram is not easily vectorized, i.e., converted into a feature vector for downstream analysis tasks. It therefore requires additional steps and decisions for converting the information from the diagram to a useful vector representation. Third, we find better performance in our generalized version: the general formulation yields better results than using the standard formulation of the Betti curve from persistent homology, since it effectively serves as a type of weighting factor for our curve. We will now discuss some of the beneficial properties of filtration curves in more detail.

3.2 Properties of Filtration Curves

While the embedding as defined via Equation 4 makes the path amenable to a sophisticated analysis in terms of high-dimensional path signatures [10], we want to focus on the properties of filtration curves if considered as scalar-valued functions. Hence, we consider a function $f\colon G \to \mathbb{R}^d$ to decompose into a set of scalar-valued functions $f_i\colon G \to \mathbb{R}$, with $i \in \{1, \ldots, d\}$. In this formulation, we lose the possibility to discover interactions in the image of f, but we gain a much more simplified perspective that we can exploit to obtain an efficient similarity measure. Each f_i s by construction a piecewise linear function (and can be decomposed into step functions), whose value changes only at finite values, namely the filtration weights. These functions form a vector space, i.e., their addition and scalar multiplication is well-defined. It is therefore possible to create a mean filtration curve, for instance, enabling visual comparison of the larger differences of the filtration curves by class. Figure 2 provides an illustration of such an analysis, and highlights how the filtration curves between different classes in a dataset can be visibly distinguished from one other. Since step functions also satisfy integrability, we obtain a norm as

$$\|f_i\|_p := \left(\int_{\mathbb{R}} |f_i(x)| \, \mathrm{d}x \right)^{\frac{1}{p}}, \tag{5}$$

i.e., the standard norm of an L^p space. Moreover, we obtain a similarity function that can be evaluated efficiently for two curves f and g, by calculating

$$\langle f, g \rangle = \int_{\mathbb{R}} f(x) g(x) \, \mathrm{d}x. \tag{6}$$

Since each function is supported on a compact set of at most m thresholds (and zero everywhere else), Equation 5 and Equation 6 boil down to calculating integrals of step functions. While Equation 6 can be implemented efficiently (in particular, both curves can be represented in a sparse fashion), our initial experiments yielded better results when *vectorizing* the curves. The vectorized representation enables us to use a random forest classifier, as opposed to evaluating the similarity using a kernel method, such as an SVM, which is an option for future work.

Complexity. The complexity of computing a filtration curve largely hinges on the complexity of the graph descriptor function. If we assume constant complexity for a graph descriptor function (which applies to a label histogram, for instance), the complexity of calculating a filtration curve is dominated by the complexity of sorting all filtration thresholds in a graph, i.e., $O(m \log m)$, with m denoting the number of edges in the graph.

Table 1: Summary statistics of the datasets.

Dataset	#Graphs	Avg. $\lvert V \rvert$	Avg. $\lvert E \rvert$	# Classes	Class Ratio
BZR_MD	306	21.30	225.06	2	0.51/0.49
COX2_MD	303	26.28	335.12	2	0.51/.049
DHFR_MD	393	23.87	283.02	2	0.68/0.32
ER_MD	446	21.33	234.85	2	0.59/0.41
BZR	405	35.75	38.35	2	0.79/.021
COX2	467	41.22	43.45	2	0.78/0.22
DHFR	756	42.23	44.54	2	0.61/0.39
PROTEINS	1113	39.06	72.82	2	0.60/0.40
IMDB-BINARY	1000	19.77	96.53	2	0.50/0.50
IMDB-MULTI	1500	13.00	65.94	3	balanced
REDDIT-BINARY	2000	429.63	497.75	2	0.50/0.50
REDDIT-5K	4999	508.52	594.87	5	balanced
MUTAG	188	17.93	19.79	2	0.66/0.34

Limitations. The main limitation of our approach lies in the choice of filtration for the graph. While we describe two filtrations for labeled and unlabeled graphs, it would also be possible to learn the filtration directly. Previous work [17] demonstrated this by using graph descriptors from computational topology. In the future, we aim to extend this approach to learn filtrations that are tailored to be useful for generalized graph descriptors.

4 EXPERIMENTS

Having introduced the details of our method, we now aim to evaluate its classification performance, runtime, and potential for even better scalability. We will address the following questions.

(1) Do filtrations curves perform well compared to SOTA alternatives?
(2) Is our runtime competitive with other fast SOTA methods, such as the Weisfeiler–Lehman relabeling scheme?
(3) Can we achieve similar results by reducing our representation, thereby "coarsening" the curve?

4.1 Datasets

For labeled graphs, we primarily consider benchmark datasets that have native edge weights: BZR_MD, COX2_MD, DHFR_MD, and ER_MD [20, 33]. These are datasets containing molecules, where the nodes represent atoms, and the edge weights correspond to the distance between the atoms, generated from the 3D coordinates of the atoms. Thus, these datasets are fully-connected graphs. Since there are few datasets available with edge weights, we show how to extend our approach to labeled graphs without edge weights by defining a weight function. For this, we use the datasets BZR, COX2, DHFR, and PROTEINS [6, 33], since they all have continuous node attributes. Following recent work which showed that this was beneficial [34], we use the Euclidean distance between the node attributes to define the edge weight on existing edges. The first three are again molecules, whereas PROTEINS is a dataset of proteins.

For unlabeled and unweighted graphs, we consider the social network datasets IMDB-BINARY, IMDB-MULTI, REDDIT-BINARY and REDDIT-MULTI-5K. Additionally, we treat MUTAG, another molecule dataset, as if it were an unlabeled dataset, removing all

Table 2: Classification accuracies on labeled datasets using the node label histogram filtration curve (FC-V) vs. state-of-the-art graph kernels and GNNs, and L_2 distance of each method to the winning method on each dataset. We report the average of 10 runs of 10-fold CV, with the best result in bold. OOT means the kernel matrix did not finish computing within 120 hours.

	Native Edge Weights				Non-Native Edge Weights				L_2
Method	BZR_MD	COX2_MD	DHFR_MD	ER_MD	BZR	COX2	DHFR	PROTEINS	
CSM	**77.63 ± 1.29**	OOT	OOT	OOT	84.54 ± 0.65	79.78 ± 1.04	77.99 ± 0.96	OOT	0.38
HGK-SP	60.08 ± 0.88	59.92 ± 0.66	67.95 ± 0.00	59.42 ± 0.00	81.99 ± 0.30	78.16 ± 0.00	72.48 ± 0.65	74.53 ± 0.35	0.36
HGK-WL	52.64 ± 1.20	57.15 ± 1.20	66.08 ± 1.02	66.72 ± 1.28	81.42 ± 0.60	78.16 ± 0.00	75.35 ± 0.66	74.53 ± 0.35	0.37
MLG	51.46 ± 0.61	51.15 ± 0.00	67.95 ± 0.00	60.72 ± 0.69	**88.04 ± 0.70**	76.76 ± 0.87	**83.22 ± 0.94**	**75.55 ± 0.71**	0.42
WL	67.45 ± 1.40	60.07 ± 2.22	62.56 ± 1.51	70.35 ± 1.01	86.16 ± 0.97	79.67 ± 1.32	81.72 ± 0.80	73.06 ± 0.47	0.25
WL-OA	68.19 ± 1.09	62.37 ± 2.11	64.10 ± 1.70	70.96 ± 0.75	87.43 ± 0.81	**81.08 ± 0.89**	82.40 ± 0.97	73.50 ± 0.87	0.23
GNN	69.87 ± 1.29	66.05 ± 3.16	73.11 ± 1.59	75.38 ± 1.60	79.34 ± 2.43	76.53 ± 1.82	74.56 ± 1.44	70.31 ± 1.93	0.19
FC-V	75.61 ± 1.13	**73.41 ± 0.79**	**76.78 ± 0.69**	**82.51 ± 1.04**	85.61 ± 0.59	81.01 ± 0.88	81.43 ± 0.48	74.54 ± 0.48	**0.04**

node and edge information. Table 1 contains detailed summary statistics about the datasets.

4.2 Experimental Setup

We vectorize our curves and use a random forest for classification. For labeled datasets, we use the node label histogram curve (FC-V), and for the unlabeled datasets we use the Ricci (FC-Ricci, $\alpha = 0.5$), max degree (FC-Max Degree) and heat kernel signature (FC-HKS) filtration curves. We compare ourselves to the SOTA graph kernels (GKs), GNNs and topological methods. We include a selection of the SOTA and empirically strongest GKs for labeled graphs; our comparison partners are therefore comprised of the Subgraph Matching Kernel (CSM, $k \in \{3, 4, 5\}, c \in \{0.1, 0.5, 1.0\}$) [20], Shortest Path Hash Graph Kernel (HGK-SP) [25], Weisfeiler–Lehman Hash Graph Kernel (HGK-WL, $h \in \{0, \ldots, 7\}$) [25], Multiscale Laplacian Graph Kernel (MLG, $\eta, \gamma \in \{0.01, 0.1\}, r, l \in \{2, 3\}$) [19], Weisfeiler–Lehman (WL, $h \in \{0, \ldots, 7\}$) [30, 31], and Weisfeiler-Lehman Optimal Assignment (WL-OA, $h \in \{0, \ldots, 7\}$) [21] graph kernels. The C parameter of the SVM was tuned from $\{10^{-4}, \ldots, 10^{3}\}$.

For our GNN comparison, we use a message passing neural network that can incorporate edge attributes on the datasets with native edge weights [13, 32], and to a GIN-ϵ [38] on all others, which recently demonstrated its superior performance to the main deep learning approaches on graphs today, and thus can be considered the toughest method to beat. When node labels are not present, the nodes are given a one-hot label defined by the node degree. We used a 5-layer network and optimized over the hyperparameters used by the authors [38]: the number of epochs ∈ {1, ..., 350}, hidden units equal to 64, batch size ∈ {32, 128}, and dropout ∈ {0, 0.5}.

Finally, we also compared to two leading topological methods, PersLay [8] and Graph Filtration Learning (GFL) [17] on the unlabeled datasets, where these methods have shown good results. The results are those listed in the original paper, and thus are not compared on the same splits of the data as ours (thus listed in gray). Additionally, GFL reports results on a single run of 10-fold cross validation, whereas the rest of the results report on the average of 10 runs of 10-fold cross validation.

We perform a 10 times repeated 10-fold cross validation with a 5-fold inner cross validation to tune all hyperparameters. This means that all parameter tuning was performed on the training data only, which ensures an unbiased estimate of the generalization accuracy. Methods were allocated 128 GB of RAM and 120 hours for training. All experiments and comparison partners that ran use the same splits of the data; results reported from original authors are marked in gray in Tables 2 and 3.

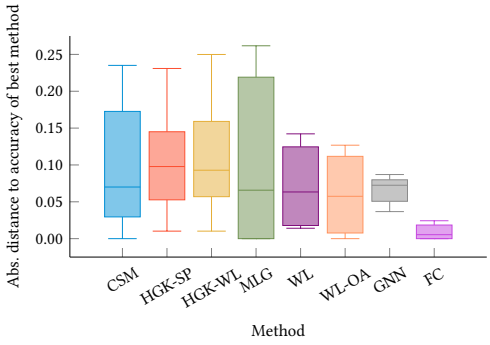
4.2.1 The Importance of a Proper Training. We would like to briefly highlight two important but underdiscussed aspects of running a proper training procedure for graph classification. Proper training mandates using a double cross validation scheme when the method requires that one chooses hyperparameters and best practices dictate running *repeated* 10-fold cross validation. While the training setup described above is commonplace in most graph kernel literature, it is less consistent in some of the more recent approaches, and can have a non-trivial effect on the results.

Cross validation can be used for *model selection* or for *model assessment*, but when it is used for both, it is necessary to have a double cross validation scheme (i.e., with both an inner and outer cross validation) in order to have an accurate assessment of the model performance. Several recent approaches, in particular neural network based approaches, tend to do a single cross validation for model selection, and then use that for model assessment as well. Practically speaking, this means that one uses the same split of the data to simultaneously chose the parameters of the model and estimate the model's generalization performance, leading to an overly optimistic estimate of the generalization accuracy. This issue is described thoroughly in Errica et al. [12].

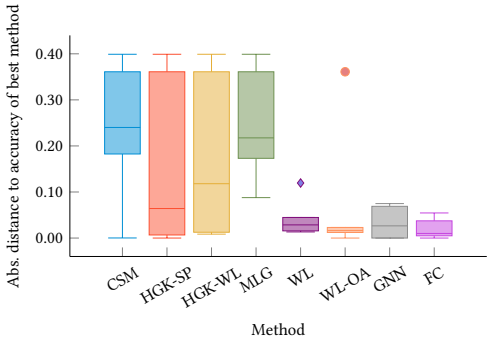
The second point is that the best practice is to repeat 10-fold cross validation (the standard is 10 times), to reduce the effect of having a lucky split—a lottery ticket—of the data. Since the benchmark datasets in graph classification are often quite small, the variability can be quite large: for example, previous work of ours had results as high as 80% on IMDB-BINARY when considering just a single run of 10-fold cross validation. However, the results were not reflective of performance when repeated 10 times, which reduced the results to around 73%. It is possible to see this effect in the standard deviations,

Table 3: Classification accuracies on the unlabeled datasets using the Ricci (FC-Ricci), max degree (FC-Max Degree), and heat kernel signature (FC-HKS) filtration curves vs. SOTA competitor methods, and the L_2 distance of each method to the best performing method. Gray results signify results from the original paper. The best result among methods that were trained on the same splits is in bold. GFL results are reported on a single run of 10-fold CV; all other methods report the average of 10 runs of 10-fold CV. OOM means "out of memory" (from an allocation of 128 GB RAM), and OOT means it did not compute in 120 hours. Our method ignores the node labels of MUTAG; the comparison methods use them.

Method	IMDB-BINARY	IMDB-MULTI	REDDIT-BINARY	REDDIT-MULTI	MUTAG	L_2
CSM	OOT	OOT	OOT	OOT	87.29 ± 1.25	0.62
HGK-SP	73.34 ± 0.47	**51.58 ± 0.42**	OOM	OOM	80.90 ± 0.48	0.54
HGK-WL	72.75 ± 1.02	50.73 ± 0.63	OOM	OOM	75.51 ± 1.34	0.55
MLG	52.56 ± 0.42	34.27 ± 0.33	OOM	OOM	78.53 ± 2.25	0.61
WL	71.15 ± 0.47	50.25 ± 0.72	77.95 ± 0.60	51.63 ± 0.37	85.75 ± 1.96	0.13
WL-OA	**74.01 ± 0.66**	49.95 ± 0.46	87.60 ± 0.33	OOM	86.10 ± 1.95	0.36
GNN	66.53 ± 2.33	48.93 ± 0.88	89.9 ± 1.9	56.1 ± 1.6	80.42 ± 2.07	0.11
PersLay	71.2	48.8	N/A	55.6	89.8	–
GFL	74.5 ± 4.6	49.7 ± 2.9	90.2 ± 2.8	55.7 ± 2.9	N/A	–
FC-Ricci	73.01 ± 0.65	46.13 ± 0.35	**89.41 ± 0.24**	**52.36 ± 0.37**	**87.31 ± 0.66**	**0.07**
FC-Max Degree	67.55 ± 0.53	43.45 ± 0.36	85.97 ± 0.15	45.91 ± 0.22	73.79 ± 1.00	0.20
FC-HKS	73.84 ± 0.36	46.80 ± 0.37	86.78 ± 0.25	52.15 ± 2.18	84.93 ± 0.90	**0.07**



(a) Labeled graphs



(b) Unlabeled graphs

Figure 3: Boxplot depicting the absolute differences of each method to the best performing method on each dataset; lower values are better. For labeled graphs, FC is FC-V (upper graph); for unlabeled it is FC-Ricci (lower graph).

which are particularly high when methods report on a single run. Figure 1 in Errica et al. [12] provides a useful depiction of the magnitude of this effect on different datasets.

We highlight these two points to address any surprise that the reader may have in seeing that the GNN results are lower than what the original authors report or what they may have expected, sometimes by several percentage points. This is explained by the fact that we perform the proper inner cross validation to select the parameters (model selection), and use an outer cross validation to report the generalization accuracy (model assessment), and furthermore we repeat our 10-fold cross validation 10 times. Our results are consistent with those found by Errica et al. [12], and represent a more accurate generalization accuracy of the approaches.

4.3 Classification Performance

To address question (1) of whether filtration curves perform well compared to state-of-the-art alternatives, we provide a comparison of the results of our approach and competing methods on labeled (Table 2) and unlabeled (Table 3) datasets. The filtration curve shows consistently strong performance: it beats all comparison methods on several datasets and is competitive with the best method when it is not first. To assess the general performance of different methods, we report the L_2 distance of each method to the winning method of each dataset. OOT/OOM values are replaced with the value of a classifier that predicts the majority class. This allows us to assess whether a method is consistently strong, or is merely able to outperform other methods on specific datasets since the L_2 distance rewards methods that are close to the winning method on all datasets, and penalizes strong deviations from the best method on single datasets. Here we see that our method shines, having the lowest L_2 distance on both labeled and unlabeled graphs. Figure 3 shows this from another lens; it is a boxplot of the absolute distances of each method to the best performing methods (lower is better). Having a large range in the

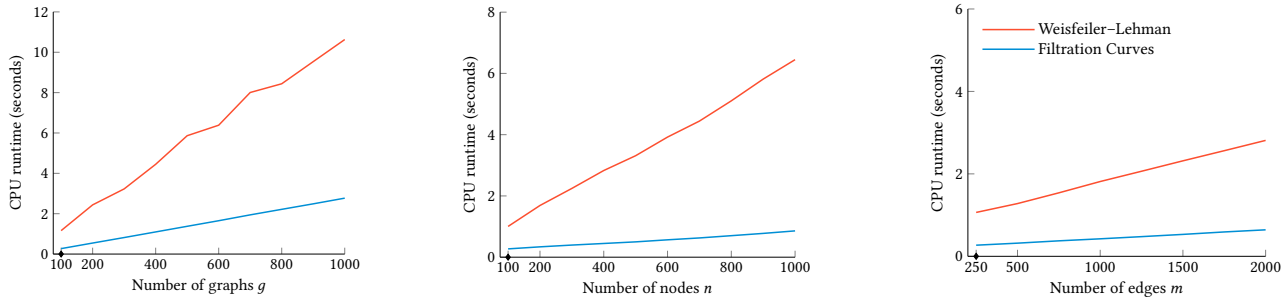


Figure 4: A comparison of CPU runtimes of building a filtration curve (here, FC-Max Degree) versus the Weisfeiler–Lehman relabeling scheme, on synthetic graphs. In each plot, only one variable is varied ($g \in \{100, 200, \ldots, 1000\}$, $n \in \{100, 200, \ldots, 1000\}$ and $m \in \{250, 500, \ldots, 2000\}$); the default values are $g = 100$, $n = 100$, and $m = 250$ (shown as diamonds on the x-axis).

boxplot indicates that a method performs badly on some datasets; in contrast, the filtration curves have a very small range, showing its consistent performance. This is particularly notable given how much simpler our approach is relative to the other methods: FC-V has no parameters; FC-Ricci has one.

4.4 Runtime Analysis

Having confirmed the first question by finding that our method has the lowest L_2 distance of all methods considered, we investigated question (2): the runtime of our method compared to other methods. Section 3.2 described an attractive theoretical complexity, so we wanted to assess the empirical performance relative to other methods that can generate a feature vector representation of a graph. To do so, we compared ourselves with the Weisfeiler–Lehman (WL) relabeling scheme, which can also be used to generate a vector representation of a graph, and is the basis for the Weisfeiler-Lehman GK [30, 31]. We generated synthetic graphs with edge weights and compared the runtimes of generating a vector from a filtration curve vs. using the WL relabeling scheme across different numbers of graphs, nodes and edges in the dataset. In each experiment, we varied one parameter of the graph, with the default values for the number of graphs $g = 100$, the number of nodes $n = 100$, and the number of edges in each graph $m = 250$. Figure 4 shows the results of building a filtration curve using the max degree; our runtime is favorable even compared to already fast methods, such as WL, underscoring one of the key advantages of our approach.

4.5 Coarsening the Curve

Finally, while we found good results using the full curve, question (3) aimed to understand whether we can achieve comparable performance while "coarsening" the curve, to further improve the speed and space requirements of our method. As mentioned in Section 3.2, the complexity of building the filtration curve is $O(m \log m)$, where m is the number of edges in a graph, and we saw a fast empirical run time. Indeed, building the curve is done in a matter of seconds to minutes across all datasets, and we see a strong correlation between the average number of edges in a dataset and how long it takes. Once each graph is represented as a vector, the remaining runtime is that of the classifier, which is largely determined by the size of the vector representation and the size of the dataset.

In order to further speed up the classification step, we coarsened the curve by not considering all edges, but rather a summary of the curve. This amounts to shrinking the vector representation; in this case, we chose to represent each dataset as a vector in $\mathbb{R}^d$, with $d \in \{5, 10, 25, 50, 100\}$, to assess how much information is already captured with such a compressed representation. Figure 5 shows the results from testing the compressed representations on the labeled datasets and FC-V. Surprisingly, a 100-dimensional vector already captures most of the classification performance of the full representation, showing the potential for massive computational and storage speedups with little performance loss. We therefore concluded that we can coarsen the curve and achieve good results.

5 DISCUSSION & CONCLUSION

We propose an effective and efficient method for graph representation that can be applied to both labeled and unlabeled datasets, making use of the relevant attribute and structural information, respectively. In doing so, we showcase how to identify critical distinguishing information in a graph and achieve competitive predictive performance, while remaining completely parameter-free in the node label histogram and max degree curve. Our approach achieved the best classification performance across datasets, as measured

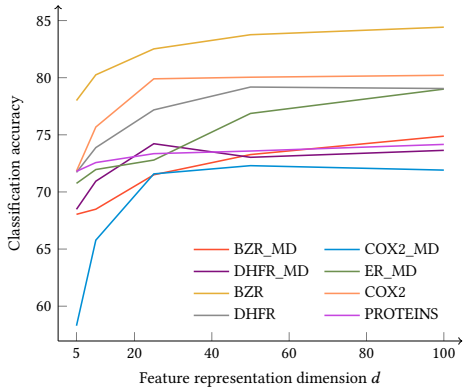


Figure 5: Classification performance on labeled datasets when coarsening FC-V, i.e., using feature vectors of varying dimension d to represent each curve.

by the L_2 distance to the best performing method, showing that our approach works consistently well on all datasets. While our current approach already exhibits strong performance, we note that it does so while only using relatively basic information about the graph in the filtration curves: the node labels and count of connected components respectively. Incorporating more sophisticated information is a natural next step, as is exploring the combination of curve descriptors with other graph representations, such as with the feature vector generated by the Weisfeiler–Lehman algorithm.

Furthermore, there is much to be explored in how to leverage the beneficial properties (described in Section 3.2) of representing a graph as a curve. Representing a graph using step functions can be used for a myriad of purposes, such as calculating the L_1 norm of the mean filtration curves, by class, and performing hypothesis testing to see if there is a significant difference between the two classes, as was suggested by Rieck et al. [28]. Preliminary experiments suggest that classification performance of our method is associated with statistically significant differences in the per-class filtration curve norms. Hypothesis testing appears to enable us to gauge the topological nature of a dataset. We plan on investigating potential applications of such properties as an extension of our work.

ACKNOWLEDGMENTS

This research was supported by the Alfried Krupp Prize for Young University Teachers of the Alfried Krupp von Bohlen und Halbach-Stiftung (K.B.).

REFERENCES

[1] Kubilay Atasu and Thomas Mittelholzer. 2019. Linear-Complexity Data-Parallel Earth Mover's Distance Approximations. In *ICML*, Vol. 97. PMLR, 364–373.
[2] Serguei A. Barannikov. 1994. The Framed Morse Complex and its Invariants. *Advances in Soviet Mathematics* 21 (1994), 93–115.
[3] Eric Berry, Yen-Chi Chen, Jessi Cisewski-Kehe, and Brittany Terese Fasy. 2018. Functional Summaries of Persistence Diagrams. *arXiv e-prints*, Article arXiv:1804.01618 (2018), arXiv:1804.01618 pages. arXiv:1804.01618 [stat.ME]
[4] Karsten Borgwardt, Elisabetta Ghisu, Felipe Llinares-López, Leslie O'Bray, and Bastian Rieck. 2020. Graph Kernels: State-of-the-Art and Future Challenges. *Foundations and Trends® in Machine Learning* 13, 5-6 (2020), 531–712.
[5] Karsten Borgwardt and Hans-Peter Kriegel. 2005. Shortest-path kernels on graphs. In *Proceedings of the Fifth IEEE International Conference on Data Mining*. IEEE Computer Society, Washington, DC, USA, 74–81.
[6] Karsten M. Borgwardt, Cheng Soon Ong, Stefan Schönauer, S. V. N. Vishwanathan, Alex J. Smola, and Hans-Peter Kriegel. 2005. Protein function prediction via graph kernels. *Bioinformatics* 21, suppl 1 (2005), i47–i56.
[7] Horst Bunke and Kaspar Riesen. 2007. A Family of Novel Graph Kernels for Structural Pattern Recognition. In *Progress in Pattern Recognition, Image Analysis and Applications*. Springer, Heidelberg, Germany, 20–31.
[8] Mathieu Carrière, Frédéric Chazal, Yuichi Ike, Théo Lacombe, Martin Royer, and Yuhei Umeda. 2020. PersLay: A Neural Network Layer for Persistence Diagrams and New Graph Topological Signatures. In *Proceedings of the 23rd International Conference on Artificial Intelligence and Statistics*, Vol. 108. PMLR, 2786–2796.
[9] Deli Chen, Yankai Lin, Wei Li, Peng Li, Jie Zhou, and Xu Sun. 2020. Measuring and Relieving the Over-Smoothing Problem for Graph Neural Networks from the Topological View. *Proceedings of the AAAI Conference on Artificial Intelligence* 34, 4 (2020), 3438–3445.
[10] Ilya Chevyrev, Vidit Nanda, and Harald Oberhauser. 2020. Persistence Paths and Signature Features in Topological Data Analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 42, 1 (2020), 192–202.
[11] Herbert Edelsbrunner, David Letscher, and Afra Zomorodian. 2002. Topological Persistence and Simplification. *Discrete & Computational Geometry* 28, 4 (2002), 511–533.
[12] Federico Errica, Marco Podda, Davide Bacciu, and Alessio Micheli. 2020. A Fair Comparison of Graph Neural Networks for Graph Classification. In *ICLR*.
[13] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. 2017. Neural message passing for quantum chemistry. In *ICML*. PMLR, 1263–1272.
[14] Robert Glem, Andreas Bender, Catrin Hasselgren, Lars Carlsson, Scott Boyer, and James Smith. 2006. Circular fingerprints: Flexible molecular descriptors with applications from physical chemistry to ADME. *IDrugs : The Investigational Drugs Journal* 9 (2006), 199–204.
[15] Alexander Grigor'yan. 2009. *Heat Kernel and Analysis on Manifolds*. American Mathematical Society.
[16] Christoph Hofer, Roland Kwitt, Marc Niethammer, and Andreas Uhl. 2017. Deep Learning with Topological Signatures. In *NeurIPS*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.). Curran Associates, Inc., 1634–1644.
[17] Christoph D. Hofer, Florian Graf, Bastian Rieck, Marc Niethammer, and Roland Kwitt. 2020. Graph Filtration Learning. In *ICML*, Hal Daumé III and Aarti Singh (Eds.), Vol. 119. PMLR, 4314––4323.
[18] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR*.
[19] Risi Kondor and Horace Pan. 2016. The Multiscale Laplacian Graph Kernel. In *NeurIPS*, D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett (Eds.). Curran Associates, Inc., 2990–2998.
[20] Nils Kriege and Petra Mutzel. 2012. Subgraph Matching Kernels for Attributed Graphs. In *ICML*.
[21] Nils M. Kriege, Pierre-Louis Giscard, and Richard C. Wilson. 2016. On Valid Optimal Assignment Kernels and Applications to Graph Classification. In *NeurIPS*. 1623–1631.
[22] Nils M. Kriege, Fredrik D. Johansson, and Christopher Morris. 2020. A Survey on Graph Kernels. *Applied Network Science* 5, 1 (2020), 6.
[23] Yong Lin, Linyuan Lu, and Shing-Tung Yau. 2011. Ricci curvature of graphs. *Tohoku Mathematical Journal, Second Series* 63, 4 (2011), 605–627.
[24] H. Ling and K. Okada. 2007. An Efficient Earth Mover's Distance Algorithm for Robust Histogram Comparison. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 29, 5 (2007), 840–853.
[25] Christopher Morris, Nils M. Kriege, Kristian Kersting, and Petra Mutzel. 2016. Faster Kernels for Graphs with Continuous Attributes via Hashing. In *Proceedings of the 16th IEEE International Conference on Data Mining*. 1095–1100.
[26] T. Ramraj and R. Prabhakar. 2015. Frequent Subgraph Mining Algorithms – A Survey. *Procedia Computer Science* 47 (2015), 197–204. Graph Algorithms, High Performance Implementations and Its Applications (ICGHIA 2014).
[27] Bastian Rieck, Christian Bock, and Karsten Borgwardt. 2019. A Persistent Weisfeiler–Lehman Procedure for Graph Classification. In *ICML*. PMLR, 5448–5458.
[28] Bastian Rieck, Filip Sadlo, and Heike Leitte. 2020. Topological Machine Learning with Persistence Indicator Functions. In *Topological Methods in Data Analysis and Visualization V*, Hamish Carr, Issei Fujishiro, Filip Sadlo, and Shigeo Takahashi (Eds.). Springer, 87–101.
[29] Kaspar Riesen. 2015. Graph Edit Distance. In *Structural Pattern Recognition with Graph Edit Distance: Approximation Algorithms and Applications*. Springer, Chapter 2, 29–44.
[30] Nino Shervashidze and Karsten Borgwardt. 2009. Fast subtree kernels on graphs. In *NeurIPS*. 1660–1668.
[31] N. Shervashidze, P. Schweitzer, E. Jan van Leeuwen, K. Mehlhorn, and K. M. Borgwardt. 2011. Weisfeiler–Lehman Graph Kernels. *Journal of Machine Learning Research* 12 (2011), 2539–2561.
[32] Martin Simonovsky and Nikos Komodakis. 2017. Dynamic edge-conditioned filters in convolutional neural networks on graphs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 3693–3702.
[33] Jeffrey J. Sutherland, Lee A. O'Brien, and Donald F. Weaver. 2003. Spline-fitting with a genetic algorithm: a method for developing classification structure-activity relationships. *Journal of Chemical Information and Computer Sciences* 43, 6 (2003), 1906–1915.
[34] Matteo Togninalli, Elisabetta Ghisu, Felipe Llinares-López, Bastian Rieck, and Karsten Borgwardt. 2019. Wasserstein Weisfeiler–Lehman Graph Kernels. In *NeurIPS*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, and R. Garnett (Eds.). Curran Associates, Inc., 6436–6446.
[35] Anton Tsitsulin, Davide Mottin, Panagiotis Karras, Alexander Bronstein, and Emmanuel Müller. 2018. NetLSD: Hearing the Shape of a Graph. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2347–2356.
[36] Yuhei Umeda. 2017. Time Series Classification via Topological Data Analysis. *Transactions of the Japanese Society for Artificial Intelligence* 32, 3 (2017), D–G72_1–12.
[37] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. 2021. A Comprehensive Survey on Graph Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems* 32, 1 (2021), 4–24.
[38] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2019. How Powerful are Graph Neural Networks?. In *ICLR*.
[39] Ze Ye, Kin Sum Liu, Tengfei Ma, Jie Gao, and Chao Chen. 2020. Curvature Graph Network. In *ICLR*.
[40] Qi Zhao and Yusu Wang. 2019. Learning metrics for persistence-based summaries and applications for graph classification. In *NeurIPS*. 9855–9866.