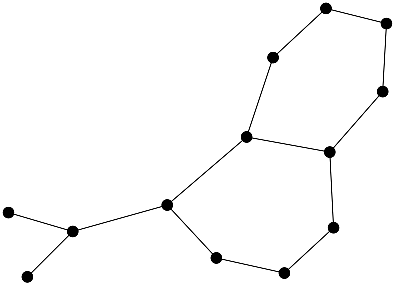
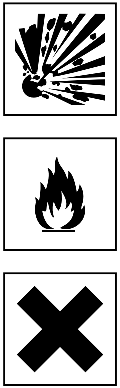
Topology-Based Graph Representation Learning

Bastian Rieck (@Pseudomanifold)

Oxford ML Summer School Unconference

What is graph classification?





Potential labels

How to represent graphs?

- Two graphs G and G' can have a *different* number of vertices.
- Hence, we require a *vectorised representation* $f\colon \mathcal{G} \to \mathbb{R}^d$ of graphs.
- Such a representation f needs to be *permutation-invariant*.

Graph neural networks in a nutshell

Learning aggregation schemes

- Learn node representations h_v based on aggregated attributes a_v.
- Aggregate them over neighbourhoods.
- Iteration k contains information up to k hops away.
- Repeat procedure K times.

$$a_v^{(k)} := \mathtt{aggregate}^{(k)}\left(\left\{h_u^{(k-1)} \mid u \in \mathcal{N}_{\mathrm{G}}(v)\right\}\right)$$

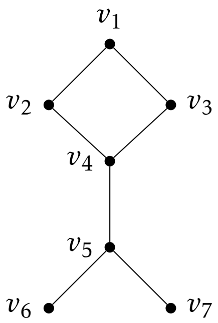
$$h_v^{(k)} := \mathtt{combine}^{(k)}\left(h_v^{(k-1)}, a_v^{(k)}\right)$$

$$h_{\mathrm{G}} := \mathtt{readout}\left(\left\{h_v^{(K)} \mid v \in \mathcal{V}_{\mathrm{G}}\right\}\right)$$

This terminology follows K. Xu, W. Hu, J. Leskovec and S. Jegelka, 'How Powerful are Graph Neural Networks?', *ICLR*, 2019.

Message passing in graphs

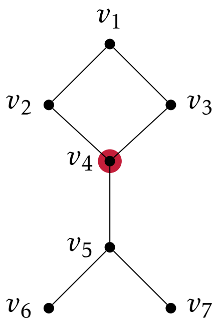
Example



Here, $v_i \in \mathbb{R}^d$ is a d-dimensional attribute vector (use one-hot encoding for labels).

Message passing in graphs

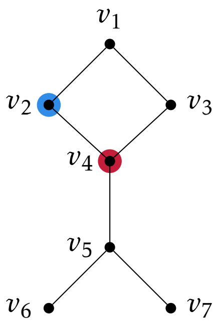
Example



Here, $v_i \in \mathbb{R}^d$ is a d-dimensional attribute vector (use one-hot encoding for labels).

Message passing in graphs

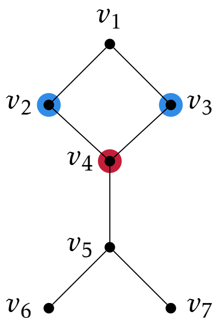
Example



Here, $v_i \in \mathbb{R}^d$ is a d-dimensional attribute vector (use one-hot encoding for labels).

Message passing in graphs

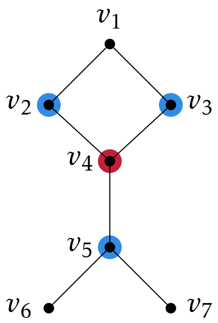
Example



Here, $v_i \in \mathbb{R}^d$ is a d-dimensional attribute vector (use one-hot encoding for labels).

Message passing in graphs

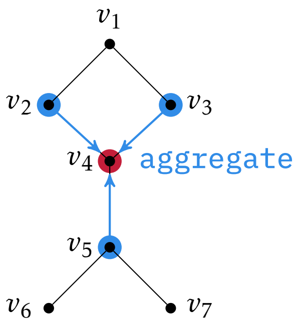
Example



Here, $v_i \in \mathbb{R}^d$ is a d-dimensional attribute vector (use one-hot encoding for labels).

Message passing in graphs

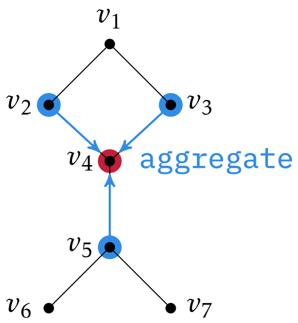
Example



Here, $v_i \in \mathbb{R}^d$ is a d-dimensional attribute vector (use one-hot encoding for labels).

Message passing in graphs

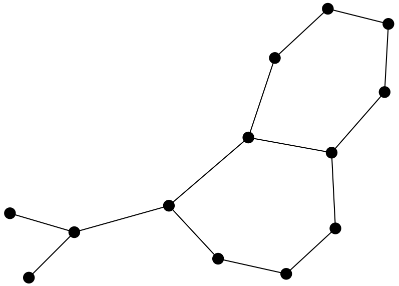
Example



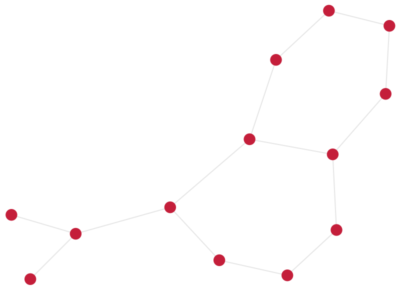
Here, $v_i \in \mathbb{R}^d$ is a d-dimensional attribute vector (use one-hot encoding for labels).

Repeat this process multiple times and update the vertex representations accordingly. Use a `readout` function to obtain a graph-level representation.

Topological features in graphs

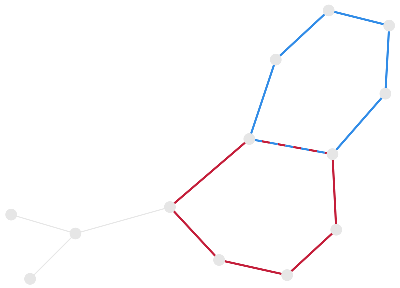


Topological features in graphs



β_0: Connected components

Topological features in graphs

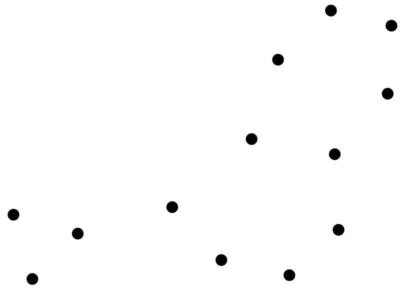


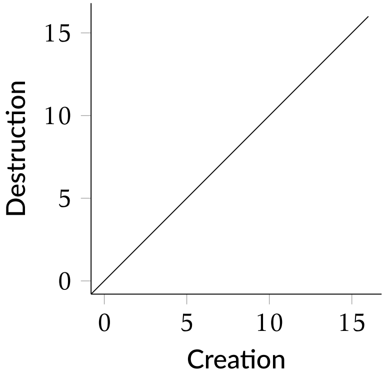
β_1: Cycles

Persistent homology

Intuition

Suppose we have *weights* on the edges. If we add them in ascending order of their weight, we can watch as topological features of the graph change! Summarise such features in a *persistence diagram*.

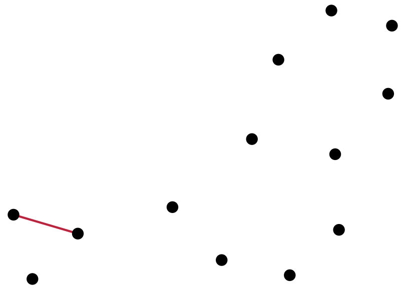


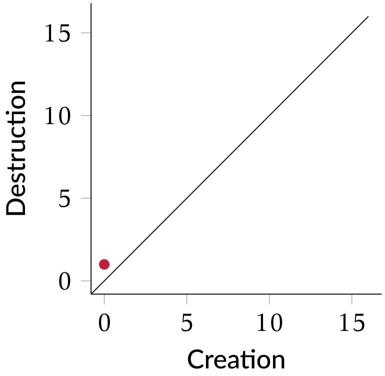


Persistent homology

Intuition

Suppose we have *weights* on the edges. If we add them in ascending order of their weight, we can watch as topological features of the graph change! Summarise such features in a *persistence diagram*.

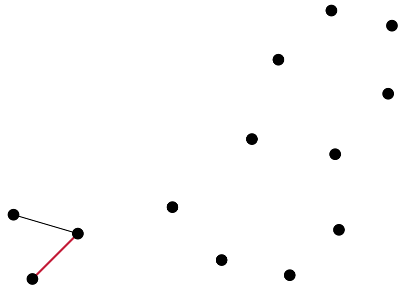


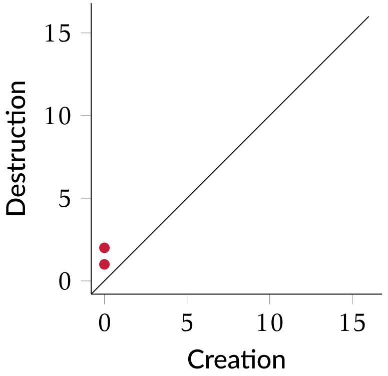


Persistent homology

Intuition

Suppose we have *weights* on the edges. If we add them in ascending order of their weight, we can watch as topological features of the graph change! Summarise such features in a *persistence diagram*.

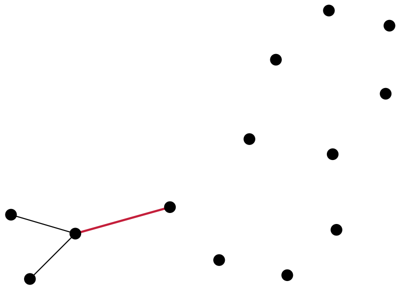


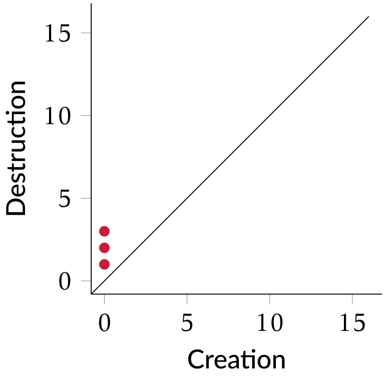


Persistent homology

Intuition

Suppose we have *weights* on the edges. If we add them in ascending order of their weight, we can watch as topological features of the graph change! Summarise such features in a *persistence diagram*.

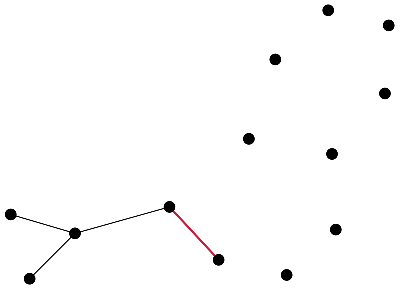


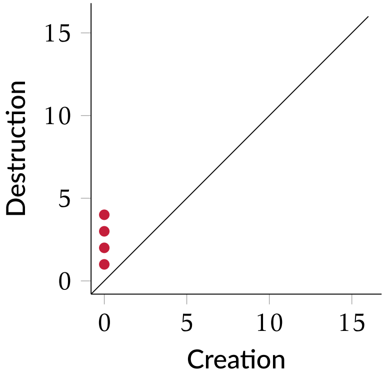


Persistent homology

Intuition

Suppose we have *weights* on the edges. If we add them in ascending order of their weight, we can watch as topological features of the graph change! Summarise such features in a *persistence diagram*.

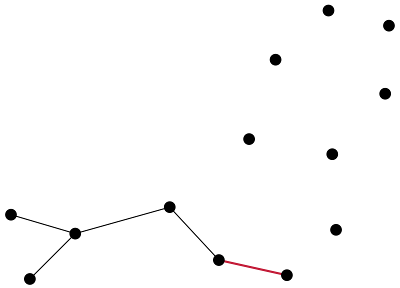


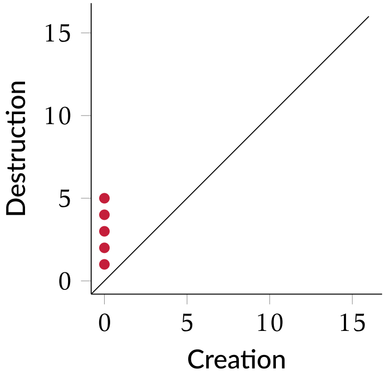


Persistent homology

Intuition

Suppose we have *weights* on the edges. If we add them in ascending order of their weight, we can watch as topological features of the graph change! Summarise such features in a *persistence diagram*.

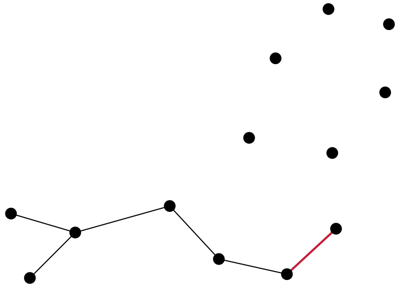


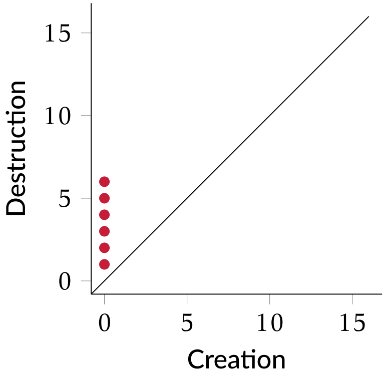


Persistent homology

Intuition

Suppose we have *weights* on the edges. If we add them in ascending order of their weight, we can watch as topological features of the graph change! Summarise such features in a *persistence diagram*.

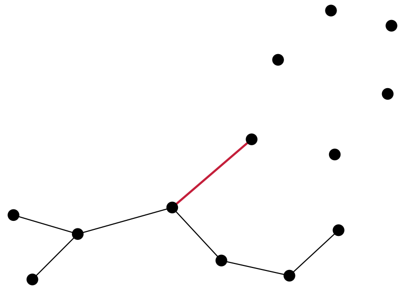


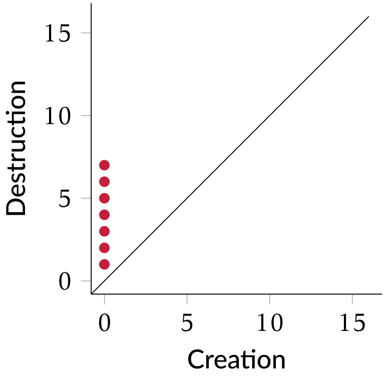


Persistent homology

Intuition

Suppose we have *weights* on the edges. If we add them in ascending order of their weight, we can watch as topological features of the graph change! Summarise such features in a *persistence diagram*.

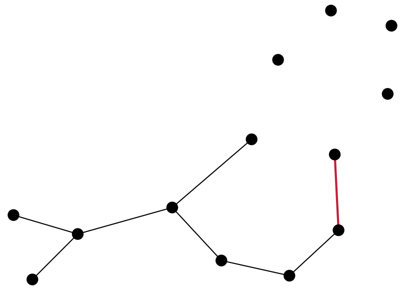


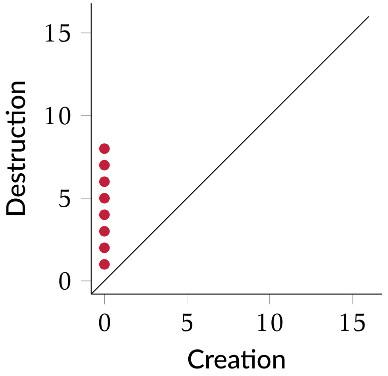


Persistent homology

Intuition

Suppose we have *weights* on the edges. If we add them in ascending order of their weight, we can watch as topological features of the graph change! Summarise such features in a *persistence diagram*.

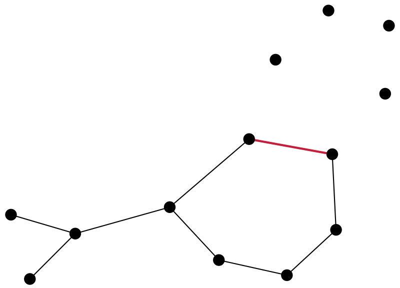


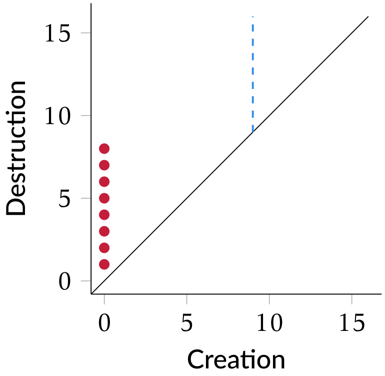


Persistent homology

Intuition

Suppose we have *weights* on the edges. If we add them in ascending order of their weight, we can watch as topological features of the graph change! Summarise such features in a *persistence diagram*.

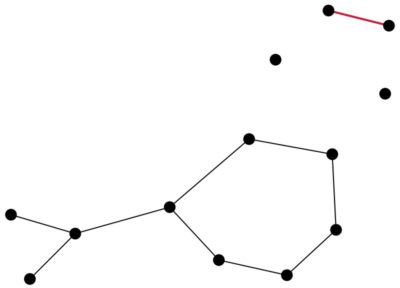


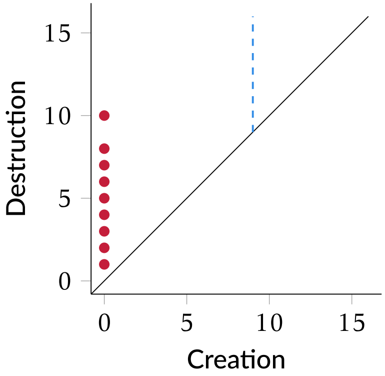


Persistent homology

Intuition

Suppose we have *weights* on the edges. If we add them in ascending order of their weight, we can watch as topological features of the graph change! Summarise such features in a *persistence diagram*.

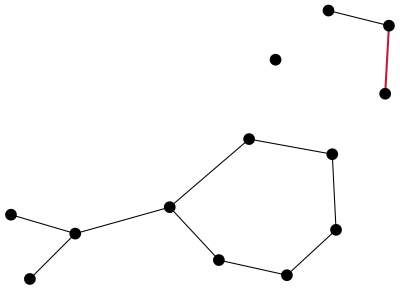


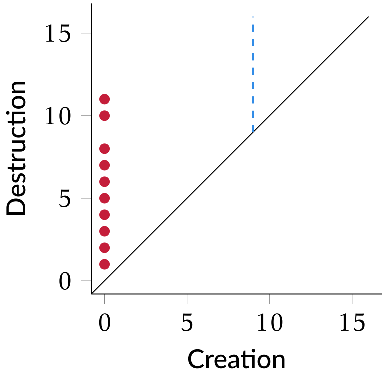


Persistent homology

Intuition

Suppose we have *weights* on the edges. If we add them in ascending order of their weight, we can watch as topological features of the graph change! Summarise such features in a *persistence diagram*.

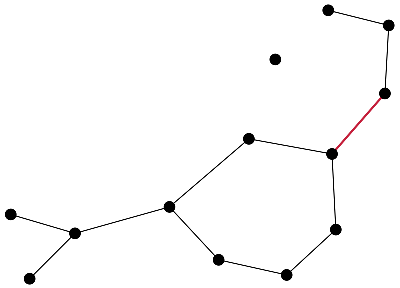


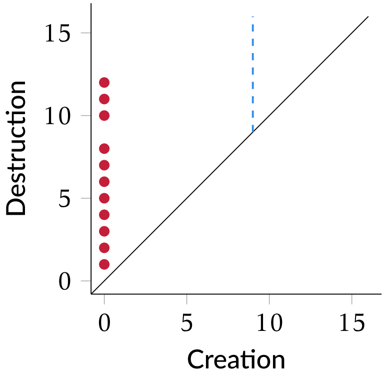


Persistent homology

Intuition

Suppose we have *weights* on the edges. If we add them in ascending order of their weight, we can watch as topological features of the graph change! Summarise such features in a *persistence diagram*.

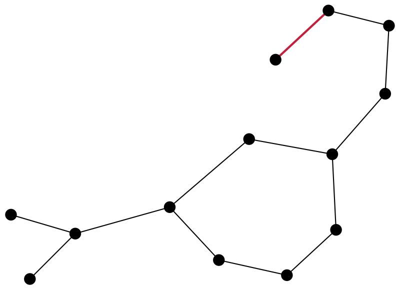


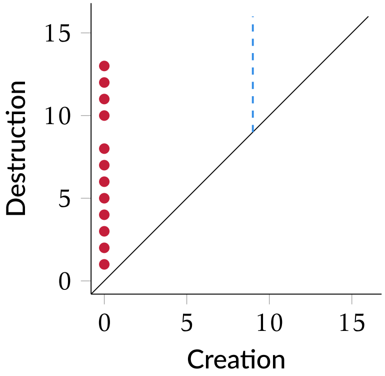


Persistent homology

Intuition

Suppose we have *weights* on the edges. If we add them in ascending order of their weight, we can watch as topological features of the graph change! Summarise such features in a *persistence diagram*.

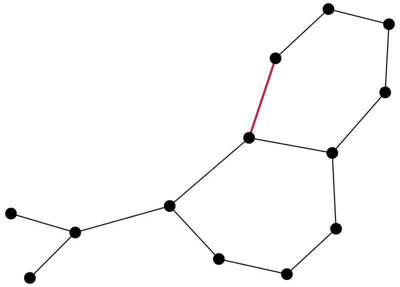


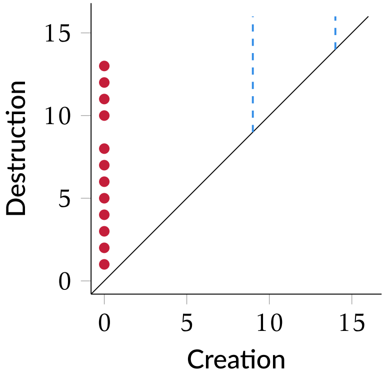


Persistent homology

Intuition

Suppose we have *weights* on the edges. If we add them in ascending order of their weight, we can watch as topological features of the graph change! Summarise such features in a *persistence diagram*.

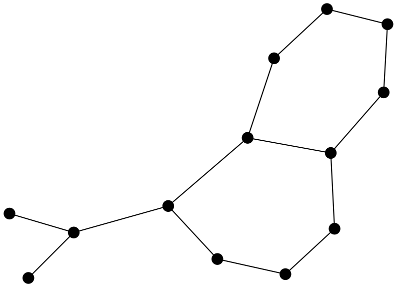


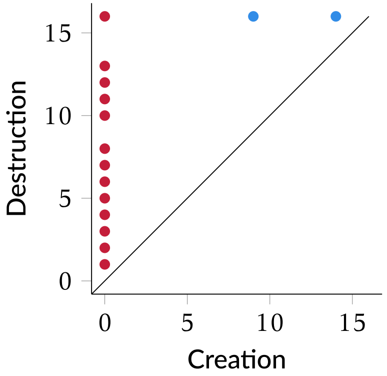


Persistent homology

Intuition

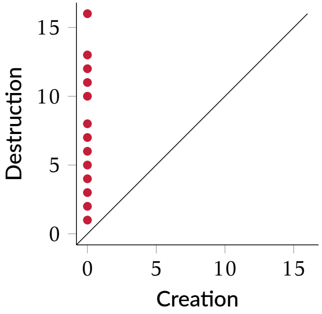
Suppose we have *weights* on the edges. If we add them in ascending order of their weight, we can watch as topological features of the graph change! Summarise such features in a *persistence diagram*.

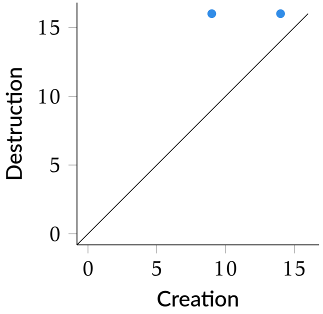




Some formal properties

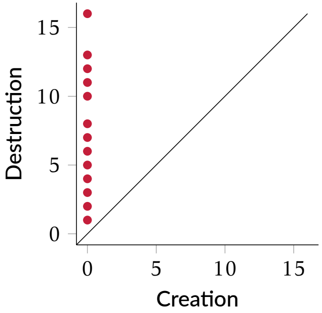
Persistent homology assigns a graph G with a function $f : \mathrm{G} \to \mathbb{R}$ a set of *persistence diagrams*, describing the topological features of G, as 'measured' via f.

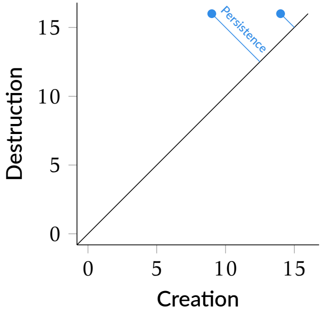




Some formal properties

Persistent homology assigns a graph G with a function $f : \mathrm{G} \to \mathbb{R}$ a set of *persistence diagrams*, describing the topological features of G, as ‘measured’ via f.





$$\mathrm{pers}(c, d) := |d - c|$$

Status quo

- Graphs are topological objects.
- But GNNs are *incapable* of recognising certain topological structures!
- What can we gain when imbuing them with knowledge about the topology?

Topological layers for graph classification

TOGL

arXiv:2102.07835v2 [cs.LG] 3 Jun 2021

Topological Graph Neural Networks

Max Horn[1, 2, *] **Edward De Brouwer**[3, *] **Michael Moor**[1, 2] **Yves Moreau**[3]
Bastian Rieck[1, 2, *, †] **Karsten Borgwardt**[1, 2, †]

[1]Department of Biosystems Science and Engineering, ETH Zurich, 4058 Basel, Switzerland
[2]SIB Swiss Institute of Bioinformatics, Switzerland
[3]ESAT-STADIUS, KU LEUVEN, 3001 Leuven, Belgium
[*]These authors contributed equally.
[†]These authors jointly supervised this work.

Abstract

Graph neural networks (GNNs) are a powerful architecture for tackling graph learning tasks, yet have been shown to be oblivious to eminent substructures, such as cycles. We present TOGL, a novel layer that incorporates global topological information of a graph using persistent homology. TOGL can be easily integrated into *any type* of GNN and is strictly more expressive in terms of the Weisfeiler–Lehman test of isomorphism. Augmenting GNNs with our layer leads to beneficial predictive performance for graph and node classification tasks, both on synthetic data sets, which can be classified by humans using their topology but not by ordinary GNNs, and on real-world data.

1. Introduction

Graphs are a natural description of structured data sets in many domains, including bioinformatics, image processing, and social network analysis. Numerous methods address graph learning problems such as graph classification or node classification. *Graph neural networks* (GNNs) describe a flexible set of architectures for graph learning tasks and have seen many successful applications over recent years [50]. At their core, many GNNs are based on iterative message passing schemes. Since these schemes are collating information over the neighbours of every node, GNNs cannot necessarily capture certain simple topological structures in graphs, such as cycles [8]. These structures, however, are relevant for certain applications, such as the analysis of molecular graphs, whose classification necessitates knowledge about connectivity information [29, 46].

By contrast, methods based on topological features, commonly summarised under the term of *topological data analysis* (TDA), have shown promising results in machine learning tasks. Focusing on coarse structures—such as the presence or absence of cycles—they can be used to provide multi-scale representations that capture the shape of complex structured and unstructured data sets. In this paper, we propose a Topological Graph Layer (TOGL) that can be easily integrated into any GNN to make it 'topology-aware'. We thus obtain a generic way to augment existing GNNs and increase their expressivity in graph learning tasks. Figure 1 provides a motivational example that showcases the potential benefits of using topological information: high predictive performance is reached *earlier* for a *smaller* number of layers.

1

Max Horn
@ExpectationMax

Edward De Brouwer
@EdwardOnBrew

Michael Moor
@Michael_D_Moor

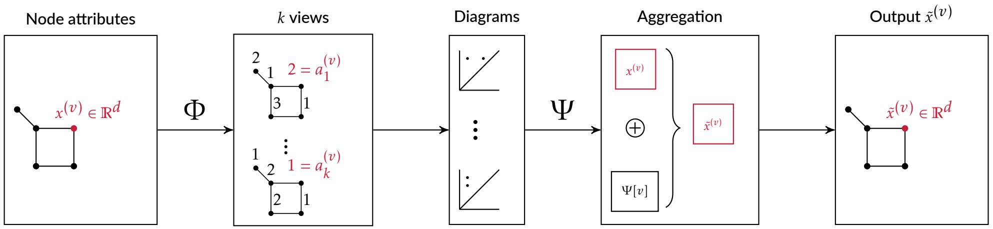
Yves Moreau

Karsten Borgwardt
@kmborgwardt

M. Horn†, E. De Brouwer†, M. Moor, Y. Moreau, **B. Rieck**†‡ and K. Borgwardt‡, 'Topological Graph Neural Networks', Preprint, 2021, arXiv: `2102.07835 [cs.LG]`

Topological graph neural networks

Overview



- Use a node map $\Phi\colon \mathbb{R}^d \to \mathbb{R}^k$ to create k different filtrations of the graph.
- Use a coordinatisation function Ψ to create *compatible* representations of the node attributes.

Expressivity of a GNN

Typical GNN architectures are *no more expressive* than the Weisfeiler–Lehman test for graph isomorphism, commonly abbreviated as WL$[1]$.[1]

Theorem

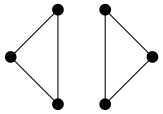
Persistent homology is *at least* as expressive as WL$[1]$, i.e. if the WL$[1]$ label sequences for two graphs G and G' diverge, there exists an injective filtration f such that the corresponding persistence diagrams $\mathcal{D}_0$ and $\mathcal{D}_0'$ are not equal.

[1] K. Xu, W. Hu, J. Leskovec and S. Jegelka, 'How Powerful are Graph Neural Networks?', *ICLR*, 2019.

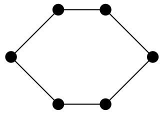
Expressivity of a GNN

There's more!

There are non-isomorphic graphs that WL$[1]$ cannot distinguish, but persistent homology can:



G



G′

We have $\beta_0(\mathrm{G}) = \beta_1(\mathrm{G}) = 2$, because G consists of two connected components and two cycles, whereas $\beta_0(\mathrm{G}') = \beta_1(\mathrm{G}') = 1$ as G' only consists of one connected component and one cycle.

Experiments

- Take GCN architecture with 4 convolutional layers (GCN-4).
- Replace second layer by TOGL.
- Use ‘static’ variant that ‘fakes’ topological calculations as an ablation.

Advantage

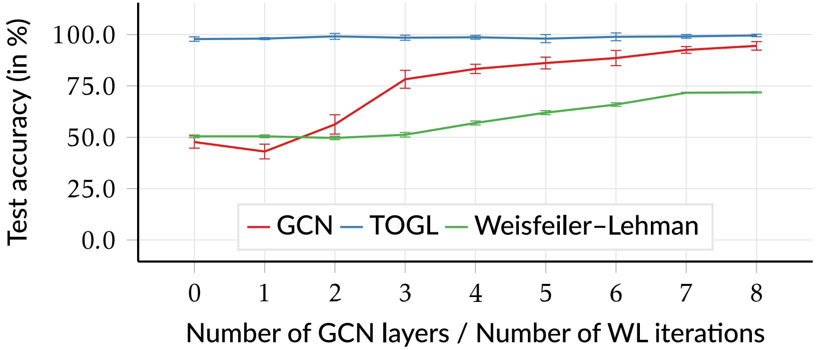
Architectures have approximately the same number of parameters; we are therefore comparing ‘apples and apples.’

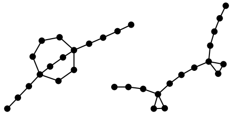
Plan

1. Assess expressivity on synthetic data sets.
2. Assess predictive performance on data sets without node features.
3. Assess predictive performance on benchmark data sets.

Expressivity

`Necklaces` data set





Classifying graphs/nodes based on structural features alone

Existing data sets tend to 'leak' information into node attributes, thus decreasing the utility of topological features. Hence, we replaced all node features by random ones.

	Graph classification				*Node classification*	
Method	`DD`	`ENZYMES`	`MNIST`	`PROTEINS`	`Cluster`	`Pattern`
GAT-4	63.3±3.7	21.7±2.9	63.2±10.4	67.5±2.6	16.7±0.0	58.3±8.8
GIN-4	**75.6**±**2.8**	21.3±6.5	83.4± 0.9	**74.6**±**3.1**	16.4±0.1	84.8±0.0
GCN-4 (*baseline*)	68.0±3.6	22.0±3.3	76.2± 0.5	68.8±2.8	16.7±0.0	85.6±0.0
TopoGNN-3-1	75.1±2.1	**30.3**±**6.5**	**84.8**± **0.4**	73.8±4.3	**16.8**±**0.0**	**86.7**±**0.0**
TopoGNN-3-1 (static)	68.0±2.4	23.7±5.4	82.9± 0.0	71.2±5.1	**16.8**±**0.0**	85.8±0.0

Classifying benchmark data sets

While we improve baseline classification performance, the best performance is *not* driven by the availability of topological structures!

	Graph classification							*Node classification*	
Method	CIFAR-10	DD	ENZYMES	MNIST	PROTEINS-full	IMDB-B	REDDIT-B	CLUSTER	PATTERN
GAT-4	64.2±0.4	**75.9±3.8**	**68.5±5.2**	95.5±0.2	76.3±2.4	—	—	57.7±0.3	75.8±1.8
GATED-GCN-4	**67.3±0.3**	72.9±2.1	65.7±4.9	97.3±0.1	**76.4±2.9**	—	—	60.4±0.4	84.5±0.1
GIN-4	55.5±1.5	71.9±3.9	65.3±6.8	96.5±0.3	74.1±3.4	72.9±4.7	89.8±2.2	58.4±0.2	85.6
WL	—	77.7±2.0	54.3±0.9	—	73.1±0.5	71.2±0.5	78.0±0.6	—	—
WL-OA	—	77.8±1.2	58.9±0.9	—	73.5±0.9	**74.0±0.7**	87.6±0.3	—	—
GCN-4 (*baseline*)	54.2±1.5	72.8±4.1	65.8±4.6	90.0±0.3	76.1±2.4	68.6±4.9	**92.8±1.7**	57.0±0.9	85.5±0.4
TopoGNN-3-1	61.7±1.0	73.2±4.7	53.0±9.2	**95.5±0.2**	76.0±3.9	72.0±2.3	89.4±2.2	60.4±0.2	**86.6±0.1**
TopoGNN-3-1 (static)	62.1±0.5	71.0±2.8	49.8±7.0	95.4±0.1	75.7±3.6	72.8±5.4	92.1±1.6	**60.5±0.2**	85.6±0.1

Where do we go from here?

- ‘If all you have is a hammer, everything looks like a nail.’ Our data sets may actually *stymie* progress in GNN research because their classification does not necessarily require structural information.
- Nevertheless, higher-order structures (such as cliques) can be crucial in discerning between different graphs.
- Would an integration into GIN architectures be smarter?
- Can we state conditions under which we are *guaranteed to learn* an appropriate filtration function?
- What do we gain from learning a filtration function?

Filtration curves

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

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

Leslie O'Bray
@leslieobray

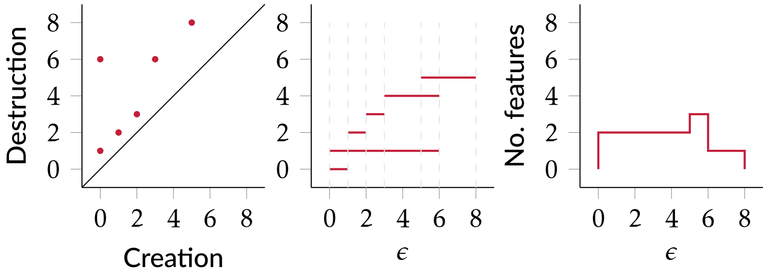
Karsten Borgwardt
@kmborgwardt

L. O'Bray[†], **B. Rieck**[†] and K. Borgwardt, 'Filtration Curves for Graph Representation', *KDD*, New York, NY, USA, 2021, pp. 1267–1275

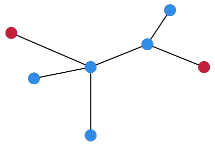
Filtration curves

Motivation

- Given a filtration of graphs, we can easily obtain a persistence diagram.
- Persistence diagrams can be conveniently represented by *Betti curves*.
- What if we use a more general descriptor function here?

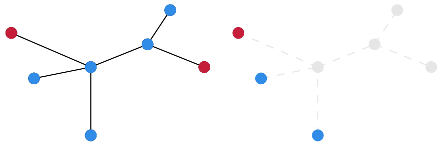


Example



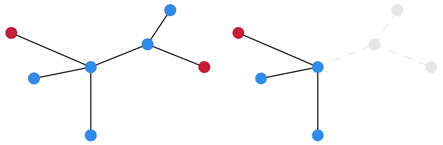
In this example, we use the *node label histogram* as a descriptor function.

Example



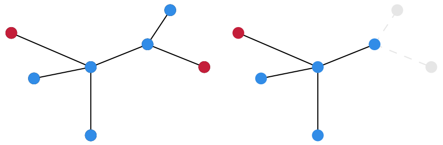
In this example, we use the *node label histogram* as a descriptor function.

Example



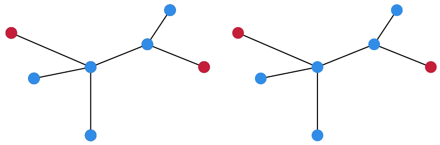
In this example, we use the *node label histogram* as a descriptor function.

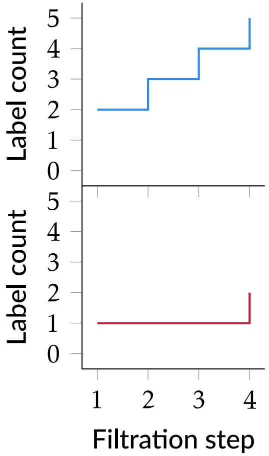
Example



In this example, we use the *node label histogram* as a descriptor function.

Example





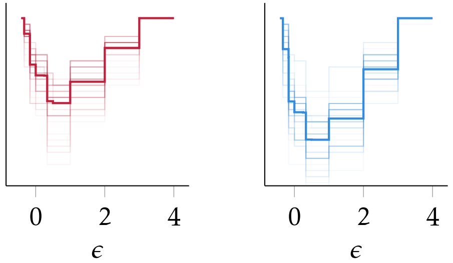
In this example, we use the *node label histogram* as a descriptor function.

General idea

- ✰ Pick function to induce a graph filtration $\mathrm{G}_1 \subseteq \mathrm{G}_2 \cdots \subseteq \mathrm{G}_k = \mathrm{G}$.
- ✰ Pick descriptor function $f\colon \mathcal{G} \to \mathbb{R}^d$.
- ✰ Evaluate f alongside the filtration.
- ✰ This turns a graph G into a high-dimensional *path* via $\mathcal{P}(\mathrm{G}) := \bigoplus_{i=1}^{k} f(\mathrm{G}_i)$.
- ✰ The path $\mathcal{P}(\mathrm{G}) \in \mathbb{R}^{k \times d}$ carries multi-scale information about G.

Properties

As generalised Betti curves, filtration curves 'inherit' a lot of their properties.[2] For instance, the *mean* filtration curve is well-defined and may be used for hypothesis testing.



[2]**B. Rieck**, F. Sadlo and H. Leitte, 'Topological Machine Learning with Persistence Indicator Functions', *Topological Methods in Data Analysis and Visualization V*, Cham, Switzerland, 2020, pp. 87–101.

Choices, choices, choices...

Filtration functions

- Native edge weights
- Degree function
- Ollivier–Ricci curvature
- Heat kernel signature

Descriptor functions

- Node label histogram
- Number of connected components

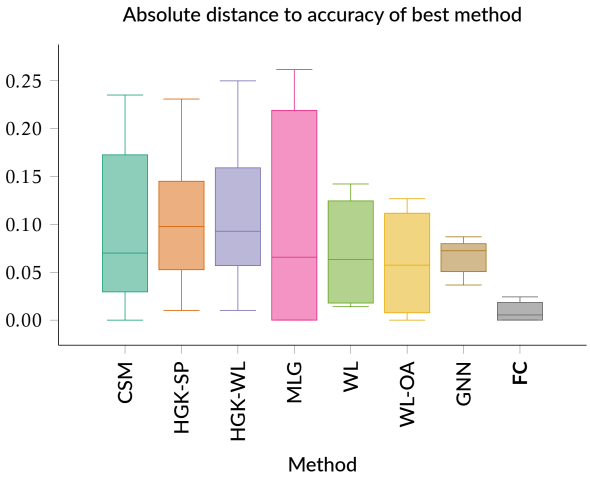
All filtration functions are ‘shallow’ for now—we are not learning a task-specific filtration.

Experiments

Is this competitive?

	Native edge weights				*Non-native edge weights*			
Method	BZR_MD	COX2_MD	DHFR_MD	ER_MD	BZR	COX2	DHFR	PROTEINS
CSM	**77.63±1.29**	—	—	—	84.54±0.65	79.78±1.04	77.99±0.96	—
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
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
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**
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
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
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
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

How good is our overall performance?



Lessons learned

- ✩ Filtration curves, even based on simple descriptors, are surprisingly competitive.
- ✩ The multi-scale aspects of TDA can be translated to other domains!
- ✩ Extensions based on learned filtrations are possible.
- ✩ We need better data sets that contain structural information.