UNIVERSITY OF CALIFORNIA SAN DIEGO

Combining Geometric and Topological Ideas with Machine Learning for Analyzing Complex Data

A dissertation submitted in partial satisfaction of the
requirements for the degree Doctor of Philosophy

in

Computer Science

by

Qi Zhao

Committee in charge:

Professor Yusu Wang, Chair
Professor Kamalika Chaudhuri, Co-Chair
Professor Yoav Freund
Professor Arya Mazumdar
Professor Hao Su

2022

The Dissertation of Qi Zhao is approved, and it is acceptable in quality and form for publication on microfilm and electronically.

University of California San Diego

2022

DEDICATION

Dedicated to my family for their love and support.

EPIGRAPH

Machines will be capable,
within twenty years,
of doing any work that a man can do.

Herbert Simon

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to acknowledge Professor Yusu Wang for her support as the chair of my committee and Professor Kamalika Chaudhuri as the co-chair of my committee. In the past years, Professor Wang supports my research and gives me valuable instructions and encouragement.

I would also like to acknowledge all members in our lab, Dingkang, Lucas, Tianqi, Minghao, Chen and Sam. They helped me in research and life. I am fortunate to have these friends.

Chapter 3, in full, is a reprint of the material as it appears in Learning Metric for Persistence-based Summaries and Applications for Graph Classification, 2019. Zhao, Qi; Wang, Yusu. Conference on Neural Information Processing System (NeurIPS), 2019. The dissertation author was the primary investigator and author of this paper.

Chapter 4, in part, is a reprint of the material as it appears in Persistence Enhanced Graph Neural Network, 2020. Zhao, Qi; Ye, Ze; Chen, Chao; Wang, Yusu. International Conference on Artificial Intelligence and Statistics (AISTATS), 2020. The dissertation author was the primary investigator and author of this paper.

Chapter 4, in part, is a reprint of the material as it appears in Prediction of Carbon Nanostructure Mechanical Properties and Role of Defects Using Machine Learning, 2021. Winetrout, Jordan; Zhao, Qi; Xu, Yanxun; Heinz, Hendrik; Wang, Yusu, arXiv, 2021. The dissertation author was the primary investigator and author of this paper.

Chapter 5, in full, is a reprint of the material as it appears in NN-Baker: A Neural-network Infused Algorithmic Framework for Optimization Problems on Geometric Intersection Graphs, 2021. McCarty, Evan; Zhao, Qi; Sidiropoulos, Anastasios; Wang, Yusu. Conference on Neural Information Processing System (NeurIPS), 2021. The dissertation author was the primary investigator and author of this paper.

Chapter 6, in full, has been submitted for publication of the material as it may appear in NN-Steiner: A Mixed Neural-algorithmic Approach for Minimum Rectilinear Steiner Tree Problem, 2022. Zhao, Qi; Wang, Yusu; Kahng, Andrew; Sidiropoulos, Anastasios. Interna-

tional Conference on Computer-Aided Design, 2022. The dissertation author was the primary investigator and author of this paper.

VITA

2016	B. S. in Mathematics , Zhejiang University
2016–2020	Master and Ph. D. in Computer Science and Engineering, Ohio State University
2020–2022	Ph. D. in Computer Science, University of California San Diego

PUBLICATIONS

"Learning Metric for Persistence-based Summaries and Applications for Graph Classification", **Qi Zhao**, Yusu Wang, Conference on Neural Information Processing System (NeurIPS), 2019

"Persistence Enhanced Graph Neural Network", **Qi Zhao**, Ze Ye, Chao Chen, Yusu Wang, International Conference on Artificial Intelligence and Statistics (AISTATS), 2020

"NN-Baker: A Neural-network Infused Algorithmic Framework for Optimization Problems on Geometric Intersection Graphs", Evan McCarty, **Qi Zhao**, Anastasios Sidiropoulos, Yusu Wang, Conference on Neural Information Processing System (NeurIPS), 2021

"Prediction of Carbon Nanostructure Mechanical Properties and Role of Defects Using Machine Learning", Jordan Winetrout, **Qi Zhao**, Yanxun Xu, Hendrik Heinz, Yusu Wang, arXiv, 2021

"NN-Steiner: A Mixed Neural-algorithmic Approach for Minimum Rectilinear Steiner Tree Problem", **Qi Zhao**, Yusu Wang, Andrew Kahng, Anastasios Sidiropoulos, 2022

ABSTRACT OF THE DISSERTATION

Combining Geometric and Topological Ideas with Machine Learning for Analyzing Complex Data

by

Qi Zhao

Doctor of Philosophy in Computer Science

University of California San Diego, 2022

Professor Yusu Wang, Chair
Professor Kamalika Chaudhuri, Co-Chair

Recently there has been an increasing number of learning problems arising in complex data domains, like graph-structured data, such as social networks and knowledge graphs. Various machine learning approaches have been developed to handle different kinds of data like PointNet for point clouds and graph neural networks for graph-structured data. At the same time, to tame the complexity in data, geometry and topology form natural platforms. In particular, computational geometry and the emerging field of topological data analysis (TDA) have been effective at capturing hidden structure and shape features from data, as well as providing efficient algorithms for them. In this thesis, we aim to integrate geometric ideas and topological concepts

with machine learning pipelines to further augment their power and performance.

In the first part of the thesis, we show how topological ideas, in particular, the so-called persistent homology, can help with analyzing graph-structured data. Persistent homology provides a flexible and versatile way to summarize complex shapes (including graphs) by a simple multiscale summary. In the first line of work, we designed an effective metric learning approach for persistence-based summaries of graphs, and showed how this improved graph classification tasks. In the second line, we show how topological summaries can be used to further improve graph neural networks' performance.

In the second part, we investigate how geometric algorithmic ideas can be combined with neural networks (NNs) to tackle hard optimization problems in geometric setting. In recent years, NNs have shown promise in helping solve combinatorial optimization problems. However, such approaches often tend to be ad hoc. Instead of solving problems in a completely data-driven manner using NNs, we propose to design mixed algorithmic-NN frameworks, where NNs are used as components within an algorithmic framework. In particular, this allows us to leverage elegant algorithmic ideas for problems in geometric setting to develop learning-based frameworks solving optimization problems efficiently in practice, but also with theoretical guarantees. We demonstrate our proposed mixed algorithmic-NN frameworks over two problems: Maximum-independent set (and several other graph problems) for intersection graphs in Euclidean setting, and computing (rectilinear) Minimum Steiner Tree for points in Euclidean setting.

Chapter 1

Introduction

1.1 Problem Statement

Machine learning and neural networks (NNs) have proved their increasing power in various challenging tasks and achieved remarkable success in many applications, such as image classification, semantic segmentation and object detection [HZRS16, Gir15, HLvdMW17, RDGF16, RFB15, LSD15, WJQ+17]. Besides image based tasks, they also demonstrate their great performance in natural language processing tasks like text analysis and machine translation [SVL14, ZZL15, VSP+17, JSL+17, KT19]. In these tasks, the data fed into the machine learning pipelines usually has a nice structure: points in Euclidean space, an image (grid-like) structure or a linear sequence. In such scenarios, it makes sense to design local filters or kernels to group representations from a neighborhood of pixels, or use a recurrent architecture to process sequential words embeddings, in an automatic way.

However, in recent years, problems in complex data domains have attracted great attention in many applications. For example, there are many problems arising in graph structured data, such as social networks analysis, chemical components structures research and transportation systems design. These problems require new machine learning approaches which not only can process features of nodes or point clouds, but also respect or leverage complex structure of objects. Different machine learning architectures are proposed in order to handle these problems. For example, graph kernels and embeddings [SSL+11, KGW16, YV15, PARS14, GL16] are

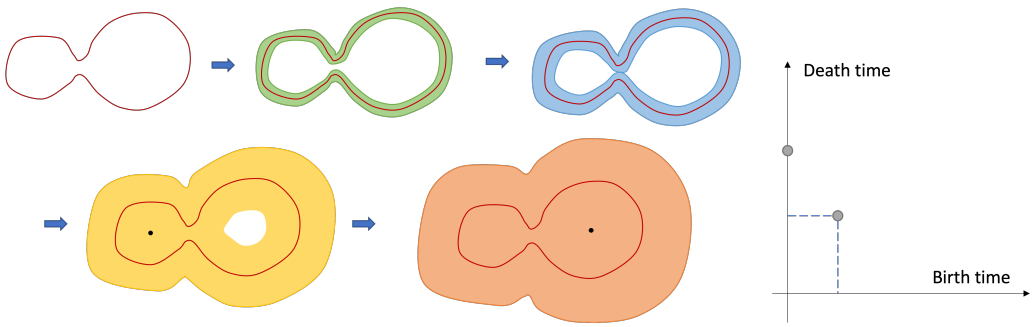


Figure 1.1. When we use persistent homology to describe this curve, we increase its width gradually. As the width increases, two holes appear and then disappear. Their appearing and disappearing moments recorded as persistence points in the persistence diagram.

proposed to process graph-structured data, PointNet [QSMG17] and variants are developed to process *a collection of* points data directly, and Geodesic Convolution Networks [MBBV15] are taken to handle 3-D shapes.

In particular, graph neural networks (GNNs) are successful generalizations of NNs in the graph domain [SGT+09, KW17, HYL17, XHLJ19, GSR+17, VCC+18]. A common way for GNNs to process node features and graph topology is to replace the local kernels performing convolutions in grid-structured data by message passing between nodes in graphs [GSR+17]. Simply speaking, nodes receive messages, such as nodes features and edge features, from specific neighborhoods. Then they update their features based on received messages. With carefully designed message functions and update functions, GNNs can learn meaningful representations of nodes or graphs for a wide range of tasks over graphs, like graph classifications, node classifications, link predictions, etc. It has proven to be an effective and powerful tool for analyzing graph-structured data, which are ubiquitous across many application domains in science and engineering.

At the same time, in a somewhat orthogonal direction of these development, the field of computational geometry and the rapidly-growing field of topological data analysis (TDA) [DW22, EH10] can provide effective tools to extract hidden structure and shape information from complex data. For example, the so-called persistent homology [ELZ02], which is a modern

much more powerful extension of the classical notion of homology groups, provides us a flexible way to encode essential features hidden in spaces and functions. It has the ability to produce a concise and multiscale feature representation for various kinds of complex data, including graphs, point clouds, shapes, and so on, in a unified framework. Intuitively, the classical homology can identify different dimensional "holes" in data like connected components in 0-dimension, loops in 1-dimension, and voids in 2-dimension. Persistent homology instead tracks the creation and disappearance of topological features through the "evolution" of a domain of interest (see Figure 1.1); thereby capturing multiscale features simultaneously. We will introduce persistent homology in detail in Chapter 2. These well-founded mathematical theories and new developed computational tools can help us to understand, discover, and model complex features hidden in data.

Given the prevalence of complex types of data in the modern data era, and given the potential power of geometric algorithms and topological methods in help tame that complexity, the **overarching question** that drives my thesis is the following:

> *Geometric algorithms and topological quantities are effective at capturing hidden structure and features in data, as well as providing efficient algorithms for them. How can we combine and integrate such ideas and methods with modern machine learning pipelines to further augment and enhance power and performance of these pipelines?*

1.2 Contributions

Guided by the overarching question above, this thesis makes contributions along the following two fronts:

- Part I: Injecting topolgoical persistence ideas to machine learning pipelines; and
- Part II: Integrating geometric algorithms ideas to machine learning architectures and develop mixed algorithmic-NN frameworks for optimization problems.

Part I: Injecting topological persistence ideas to machine learning.

As we mentioned earlier, persistent homology gives us a way to generate the so-called persistence diagram (PD) as a multiscale topological summary of features for an input object (e.g., for a graph, a surface model, or a point cloud). The PD consists of a multi-set of persistence points in the plane, where the x- and y-coordinates of each point capture the *birth* and the *death* of some topological features as one scans through the domain of interests via a certain perspective (called a filtration) – we will see details later in Chapter 2. For example, in Figure 1.1 (b), the two persistence points record the birth and death of the two holes as we gradually thicken the curve of interests. In the first part of my thesis, we will investigate how PDs topological summaries can help us with machine learning tasks to handle in particular graph type of data.

In particular, in Chapter 3, leveraging the power of PD summaries, we aim to use the space of PD as feature space to carry out machine learning tasks (e.g., clustering and classification). To do this, we need a metric structure on the space of PDs. Unfortunately, the classical notion of distances for PDs lack nice structure (e.g., inner-product structure), and cannot be easily integrated to machine learning methods. Thus in the past few years, there have been several methods developed to vectorize PD in order to better facilitate their use in machine learning pipelines [Bub15, AEK$^+$17, RHBK15, KFH18]. In these approaches, when computing the distance or kernel between persistence summaries, the importance (weight) of different persistence features are often pre-determined. Commonly, they are treated equally or reweighted by using "persistence" as importance. However, as recognized by recent studies, topological features with high persistence doesn't necessarily mean high importance in the downstream machine learning tasks. Hence we propose to learn a weight function for persistence points in feature space in a data driven way. Specifically, we propose a weighted-kernel, WKPI, for persistence summaries, which we prove to be positive semi-definite and stable to perturbations. In this kernel, we use a weight function to encode the importance of different locations in the persistence diagram. We then learn the weight function from a certain function class by a metric learning problem for persistence summaries. We formulate the metric-learning as an optimization

problem over a specifically designed cost function. Given a set of objects, we first learn a WKPI-kernel as described before, and then use the learned WKPI to further classify objects. We apply our approach in graph classification tasks. Our new persistence summaries based approach outperforms or achieves similar performance to existing state-of-the-art approaches, such as graph kernels and graph neural networks, on a wide range of benchmark datasets, validating the effectiveness of persistence summaries over describing graphs.

In Chapter 4, we show how we leverage persistence summaries to enhance the performance of graph neural networks (GNNs) in node classification tasks. In standard GNNs, each node aggregates messages, which are often weighted uniformly or by the node's degree, from its neighborhood and it then updates its representation based on aggregated messages [KW17, HYL17, XHLJ19]. This procedure is defined as message passing [GSR$^+$17]. Message passing GNNs have received tremendous attention and success in the past few years in various applications. However, it is also known that they have limitations in terms of what they can capture. For example, it is known that using simple local information as initial node features, the message passing GNNs cannot capture graph information such as size of (shortest) cycles. To this end, persistent homology can be effective at capturing local/global structural information of graphs. Hence we propose to use persistent homology to establish initial node features for message passing neural networks. Furthermore, we observe that the local topology within a graph has influence on message passing. Another observation is that persistence summaries extracted from subgraphs are stable to certain perturbations and can capture sufficient information to differentiate local topological structures in a graph. Based on these observations, we develop a novel graph network architecture, persistence enhanced graph network (PEGN), using persistence summaries in a data-driven manner. Specifically, we train a separate network to reweight messages between nodes based on the input persistent homology information extracted from neighborhood of nodes. This empowers the graph convolution and message passing in network to be adaptive with different local structures. Our algorithm achieves better performance than existing GNNs that only use node-feature-based attention in node classification tasks. This

confirms the power of persistence homology as advanced local structure information in graph learning.

Part II: Integrating geometric algorithms with designs of neural network architectures and developing mixed algorithmic-NN frameworks.

We focus on combinatorial optimizatoin problems, such as maximum independent sets, Steiner minimum tree problems and so on, which have important practical applications. These optimization problems however are hard to solve or even to approximation in the general setting. Recently there has been a range of approaches developed to use neural networks to help solve such optimization problems in a data-driven manner [BLP20]; such as the various GNN architectures developed to learn graph representations and solve graph optimization problems in an end to end manner [BPL$^+$17, LCK18, DKZ$^+$17, ASS20]. There have also been several recent lines of work to tackle the mixed integer linear programming problems [KLBS$^+$16, GCF$^+$19, GGK$^+$20] Despite the tremendous amount of progress in this direction, the practical performances still have much space for improvement; the neural networks can often be used in a rather ad hoc manner; and theoretical understanding is limited. One question is whether a machine learning pipeline can solve a combinatorial optimization problem exactly or approximately with theoretical guarantee. [SYK19, Lou20] connect GNNs with distributed local algorithm [Ang80, ÅFP$^+$09] and show GNNs' capacity on approximating some graph combinatorial optimization problems like maximum independent set. However, their capacity is limited to the special family of constant-degree graphs.

At the same time, in theoretical computer science, many elegant approximation algorithms or fixed parameter tractable algorithms have been developed for such hard problems in special settings, especially in the *geometric setting* that we are interested. Such algorithms often rely on deep insights of the mathematical structures behind data, and combine that with algorithmic paradigms such as specialized divide-and-conquer frameworks. A good example in this direction is Arora's PTAS (polynomial time approximation scheme) algorithms [Aro98] for a range of optimization problems in the Euclidean setting. Unfortunately, many such algorithms still have

not found their way to practices, and remain only of theoretical interests, as their time complexity, while being polynomial, can still be high (and often depends on some parameters exponentially).

In this second part of this thesis, we investigate how to integrate such beautiful algorithmic ideas with NN architectures. In particular, the high level idea is that, we will still leverage the algorithmic framework developed, while use NNs for certain costly components contained inside in a suitable manner. As we will see below: for two algorithmic paradigms we will consider, our mixed algorithmic-NN frameworks have the benefit of both worlds: (1) On one hand, the costly component is now replaced by a learnable NN, which significantly reduces the running time on test data. (2) On the other hand, the problems are decomposed via the algorithmic ideas, so that we are in fact learning an algorithmic component that will be applied to problems of *bounded size*! This not only means that we only need to train a fixed size NN over data of bounded size, this also ultimately gives our mixed algorithmic-NN framework the capacity to solve the problem at hand for input of arbitrary size.

In particular, in Chapter 5, as a warmup, we will introduce our first attempt in this direction. Our NN-Baker aims to infuse the so-called Baker's technique with NN architectures to solve optimization problems such as maximum independent set or minimum vertex cover, in the geometric setting. That is, we assume that our input graph is the intersection graph of a set of d-dimensional balls in the Euclidean space $\mathbb{R}^d$. We are inspired by the so-called Baker's technique [Bak94] and propose a partition based approximation algorithm, Baker-paradigm. We prove it gives a bi-criteria approximation with running time linear in the size of input point set, but exponential to other parameters. Baker-paradigm decomposes the problem into small sub-problems of fixed size independent of size of input point set, and the final solution is obtained by merging solutions of these sub-problems. For the family of such fixed-size sub-problems, we also design neural networks with universal approximation guarantees to solve them. By replacing the sub-problem component in Baker-paradigm with neural networks, we obtain a mixed algorithmic-NN framework which we call NN-Baker. NN-Baker has capacity to produce a bi-criteria approximation of MIS in near linear time. We design two instantiations, CNN-

Baker and GNN-Baker, according to the neural network we infuse into Baker-paradigm. In experiments, when the problem size increases, the performance of neural network baselines, like TGS [LCK18], LwD [ASS20] and Erdős-GNN [KL20], decrease. Our GNN-Baker can significantly improve their performance as the problem size increases. Part of the reason is our GNN-Baker only needs to train and validate the GNN component on small graphs of bounded size.

Our work in Chapter 5 shows promise of mixed algorithmic-NN framework. It however, leverages a rather simple algorithmic framework (the Baker's technique), which intuitively has a flat partition of input problem into a set of smaller instances. In Chapter 6, we will consider a more sophisticated algorithmic framework: Arora's PTAS [Aro98] for geometric optimization problems in the Euclidean setting. We will specifically consider the problem of minimum (rectilinear) Steiner tree problem, due to the practical importance of this problem in chip design. Arora's algorithm makes quadtree decomposition over the input instance into hierarchical sub-problems according to their geometric distribution and solves sub-problems level by level in a dynamic programming way. It has been proved to have a constant approximation ratio. However, this algorithm is also impractical due to the computational expense. In NN-Steiner, we simulate this dynamic programming procedure by neural networks to make the framework more efficient. In specific, we develop a bi-directional framework and generate graphs for sub-problems at different levels. We then design GNN architectures which receive and process knowledge from the previous level sub-problems and output knowledge for the those at the next level. In the forward direction, GNNs are used to learn potential Steiner points' embeddings, while in the backward direction, they learn potential Steiner points' distributions. NN-Steiner demonstrates its better generalization ability in experiments than both state-of-the-art approximation algorithms and machine learning based baselines. Our two algorithmic-NN frameworks open a new door to tackle hard combinatorial optimization problems by infusing neural networks as components into algorithmic structure leveraging geometric ideas and setting.

Chapter 2

Background

2.1 Persistent homology

Persistent homology (PH) is a powerful tool to capture topological features from objects like point clouds, shapes, graphs, etc. We first give an informal description of persistent homology in this section. See [EH10] for more detailed exposition on the subject.

Suppose we are given a shape X. Imagine we inspect X through a *filtration of* X, which is a sequence of growing subsets of X: $X_1 \subseteq X_2 \subseteq \cdots \subseteq X_n = X$. As we scan X, sometimes a new feature appears in X_i, and sometimes an existing feature disappears upon entering X_j. Using the topological object called homology classes to describe these features (intuitively components, independent loops, voids, and their high dimensional counter-parts), the birth and death of topological features can be captured by the *persistent homology*, in the form of a *persistence diagram* DgX. Specifically, for each dimension k, Dg$_k X$ consists of a multi-set of points in the plane (which we call the *birth-death plane* $\mathbb{R}^2$): each point (b,d) in it, called a *persistence-point*, indicates that a certain k-dimensional homological feature is created upon entering X_b and destroyed upon entering X_d. Besides persistence diagram, we also add the points on the diagonal to the persistence diagram, each with infinite multiplicity. In the remainder of the thesis, we often omit the dimension k for simplicity: when multiple dimensions are used for persistence features, we will apply our construction to each dimension and concatenate the resulting vector representations.

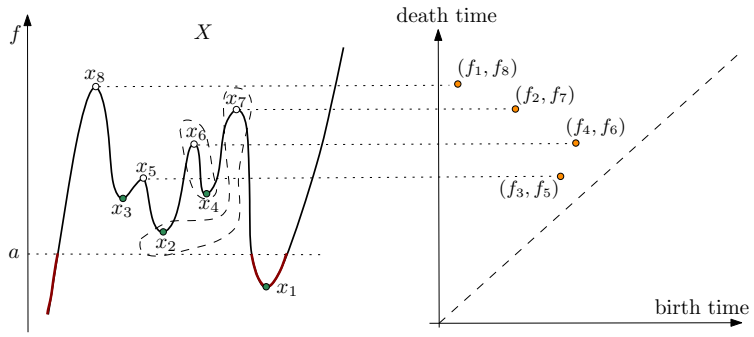


Figure 2.1. Sweep the curve in increasing f-values, at certain moments 0-th homological features (connected components) are created or destroyed. For example, a component is created when passing x_4 and killed when passing x_6, giving rise to persistence-point (f_4, f_6) in persistence diagram.

A common way to obtain a meaningful filtration of X is via the *sublevel-set filtration* induced by a *descriptor function* f on X. More specifically, given a function $f : X \to \mathbb{R}$, let $X_{\leq a} := \{x \in X \mid f(x) \leq a\}$ be its *sublevel-set at a*. Let $a_1 < a_2 < \cdots < a_n$ be n real values. The sublevel-set filtration w.r.t. f is: $X_{\leq a_1} \subseteq X_{\leq a_2} \subseteq \cdots \subseteq X_{\leq a_n}$; and its persistence diagram is denoted by $\mathrm{Dg} f$. Each persistence-point $p = (a_i, a_j) \in \mathrm{Dg} f$ indicates the function values when some topological features are created (when entering $X_{\leq a_i}$) and destroyed (in $X_{\leq a_j}$), and the *persistence* of this feature is its life-time $\mathrm{pers}(p) = |a_j - a_i|$. See Figure 2.1 for a simple example where $X = \mathbb{R}$. If one sweeps X top-down in decreasing function values, one gets the persistence diagram induced by the super-levelset filtration of X w.r.t. f in an analogous way. Finally, if one tracks the change of topological features in the *levelset* $f^{-1}(a)$, one obtains the so-called *levelset zigzag persistence* [CdSM09] (which contains the information captured by the *extended persistence* [CSEH09]).

Persistent homology for point clouds

When X is a point cloud, one commonly used approach to obtain persistence summaries is constructing a so-called *Vietoris-Rips filtration*. Here, a space is modeled by a *simplicial complex* spanned by a vertex set V: Roughly speaking, a k-dimensional simplex is the k-dimensional generalization of vertices (0-D), edges (1-D) and triangles (2-D simplices). A simplicial complex

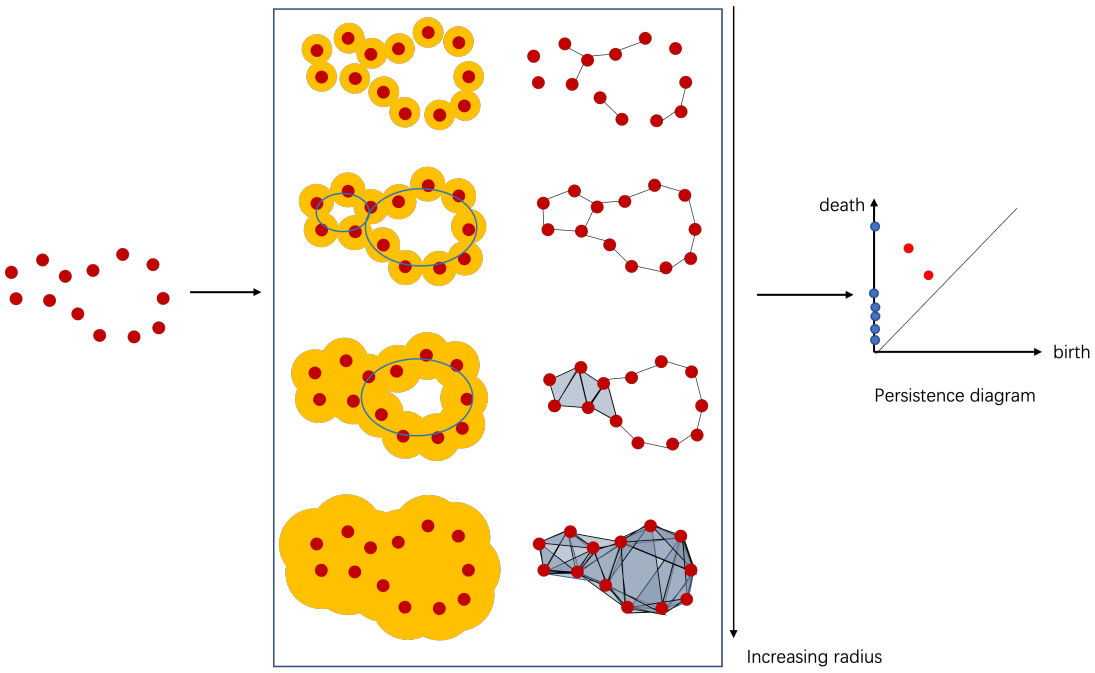


Figure 2.2. Vietoris-Rips filtration on point clouds. (Left): a set of input points in 2D. (Middle-left): a ball centered at each point in the 2D space with different scales (radius). (Middle-right): the corresponding Vietoris-Rips complexes as radius increases. (Right): the persistence diagram.

is then simply a union of simplices with the condition that if a simplex is contained in this complex, then any of its face will also be in the complex.

Given a set of point $V \subseteq \mathbb{R}^n$, the Vietoris-Rips complex at scale r consists of all simplices with diameter less than r:

$$\mathrm{VR}_r(V) = \{\sigma \subset V \mid \forall u, v \in \sigma, ||u - v|| \leq r\} \tag{2.1}$$

In particular, $\mathrm{VR}_0(V) = \{\{u\} || u \in V\}$, and $\mathrm{VR}_\infty(V)$ consist of all simplices spanned by vertices in V. By increasing r from 0 to ∞, we obtain a filtration $\mathrm{VR}_0(V) \subseteq \mathrm{VR}_{r_1}(V) \subseteq \mathrm{VR}_{r_2}(V) \subseteq ... \subseteq \mathrm{VR}_\infty(V)$ $(0 \leq r_1 \leq r_2 \leq ...)$. See Figure 2.2 for a 2D example for illustration: The left is a set of input points in 2D. In the middle-left, we create a ball centered at each point in the 2D space, and the union of balls (shown in orange) can be considered as a thickening of the space captured by input points at different scales (radius). In the middle-right, the corresponding

Vietoris-Rips complexes as radius increases. An edge is added whenever two balls intersect, and whenever there are three edges among three vertices, a triangle is added. In general, if the balls around k points all have pairwise intersection, then the $(k-1)$-simplex spanned by these k points is added into the complex. The sequence of Vietoris-Rips complexes as the radius increases gives rise to the Vietoris-Rips filtration. The persistence diagram shown in the right encodes the birth and death of topological features of different dimensions through this filtration. Blue dots corresponds to dimension-0 homological features, which captures the birth and death of connected components. It turns out these blue dots capture the single-linkage hierarchical clustering information. Red dots correspond to the birth and death of 1-dimensional homological features, which are the creation and death of holes in this 2D example. Intuitively, there are two holes created during this course, where the persistence (i.e, the life time as computed by deathtime - birthtime) of the left hole is smaller than that of the right hole. In general, higher-dimensional persistence diagrams will capture the creation and death of higher-dimensional voids.

Persistent homology in graph setting

Given a graph $G = (V,E)$, we can view it as a 1-dimensional simplicial complex. A (descriptor) function f defined on V or E will then induce a filtration as well as its persistence diagram summary. In particular, suppose $f : V \to \mathbb{R}$ is defined on the node set of G (e.g, the degree function). Then we can extend f to edges E of G by setting $f(u,v) = \max\{f(u), f(v)\}$, and the sublevel-set at a is defined as $G_{\leq a} := \{\sigma \in V \cup E \mid f(\sigma) \leq a\}$. Similarly, if we are given $f : E \to \mathbb{R}$, then we can extend f to V by setting $f(u) = \min_{u \in e, e \in E} f(e)$. As we sweep G via the sublevel-set filtration of f, connected components in the swept subgraphs will be created and merged, and new cycles will be created. The formal events are encoded in the 0-dimensional persistence diagram $\mathrm{Dg}_0 f$. The the 1-dimensional features (cycles), however, we note that cycles created will never be killed, as they are present in the total space $X = G$. To this end, we use the so-called *extended persistence* introduced in [CdSM09] which can record information of cycles.

An example of the persistence diagram induced by a function on graph is given in Figure 2.3. In this example, the descriptor function $f : V = \{u_1, \ldots, u_{10}\} \to \mathbb{R}$ is the shortest path distance function to the base point u_1; that is, for any u_i, $f(u_i) = -d_G(u_i, u_1)$ where d_G denotes the shortest path metric on G. Points in the 0-D persistence diagram and the 1-D extended persistence diagram are shown in Figure 2.3 (c). For example, there are three independent loops in this graph, and a specific basis (the so-called "thinnest" system of loops) are captured in the 1-D extended persistence diagram (giving rise to three persistence points).

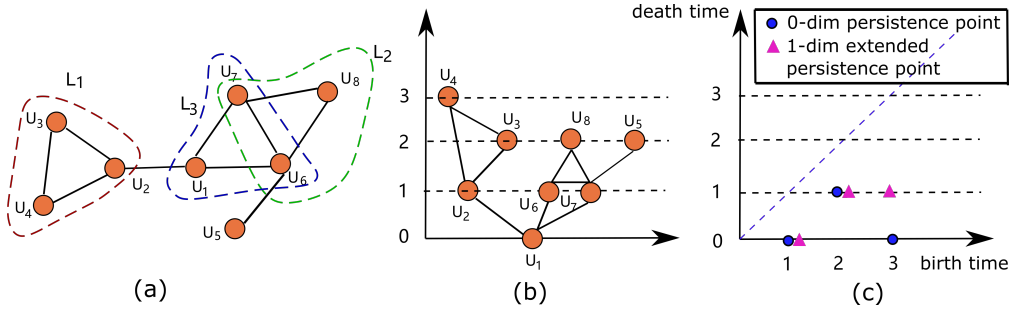


Figure 2.3. (a). A graph whose all edges have weight 1, other than edge (u_2, u_4) with weight $w(u_2, u_4) = 2$. (b). Re-plot the graph. Height of each node u_i equals to descriptor function value $f(u_i)$. (c) 0-D persistence diagram induced by the superlevel-set filtration and 1D extended persistence diagram.

Metrics for Persistence Diagrams.

Two most common ways to measure distances between persistence diagrams are the so-called bottleneck distance and the p-th Wasserstein distance. Both of these distances have been well studied in the literature, including stability results under these distances (e.g, [CSEH07, CSEHM10, CdSGO16]) and efficient implementations [KMN17, KMN18]. In this thesis, we focus on the Wassertein distance, which we define below. Recall that all points on the diagonal of the plane belongs to the persistence diagram with infinite multiplicity.

Definition 2.1.1. *Let A_1 and A_2 be two persistence diagrams. The p-th Wasserstein distance*

between them is defined as:

$$d_{W,p}(A_1,A_2)) = \inf_{\gamma:A_1\to A_2}[\sum_{x\in A_1}||x-\gamma(x)||^p]^{1/p} \tag{2.2}$$

where the infimum is over all bijections $\gamma : A_1 \to A_2$ and the summation is over all points in A_1.

2.2 Graph neural networks

With the increasing demands for deep learning techniques over graph-structured data, graph neural networks have been developed in recent years for various graph learning tasks and widely applied in fields like social networks, knowledge graphs, chemical molecules, etc.

As the earlier studies generalizing neural networks in graphs, [GMS05, SGT+09] take recurrent neural networks. They learn nodes' embeddings by propagating neighbor information iteratively until reaching a stable fixed point. Encouraged by the power and immense success of convolutional neural networks on images and texts, different efforts are made to generalize "convolution" over graphs. Spectral graph convolutional neural networks [BZSL14, DBV16] apply convolution operations to the spectral domain or the frequency domain of the input graph. A bottleneck of spectral graph convolution is it suffers the over-smoothing phenomenon and its performance decreases with the increasing number of hidden convolution layers.

Spatial neural networks take a different way to perform convolution over graphs, which is commonly formulated as message passing [GSR+17]. In specific, a node in a graph iteratively receives information from its neighborhood and update its representation or features. These node and edge representation information transferred between vertices are called messages. A transformation of the messages and updated representations can be learned through training. This message passing scheme can be formulated in a more explicit manner. Given an undirected graph $G = (V,E)$ where V is the node set and E is the edge set, the input to a GNN are node features h_u^0 and edge features e_{uv} for every $u \in V$ and $(u,v) \in E$, then in the t-th hidden layer of a

message passing GNN, the forward convolution consists of two functions:

$$\begin{aligned} \text{AGGREGATE} \quad & m_u^{t+1} = f^t(h_u^t, e_{uv}, \{h_v^t | v \in N(u)\}) \\ \text{UPDATE} \quad & h_u^{t+1} = g^t(m_u^{t+1}, h_u^t) \end{aligned} \tag{2.3}$$

where $N(u)$ is the neighborhood of node u, $f(\cdot)$ and $g(\cdot)$ are message aggregation and update function respectively. Finally, if the task is on the graph level, there is a readout function $r(\cdot)$ in the final layer mapping node representations to a graph representation

$$h_G = r(\{h_u^T | u \in V\}) \tag{2.4}$$

where the T-th layer is the final layer.

Some popular spatial GNNs are GCN [KW17], GraphSAGE [HYL17], GIN [XHLJ19], GAT [VCC+18], etc. These 1-hop GNNs have the same expressive power as the 1-dimensional Weisfeiler-Leman (1-WL) graph isomorphism test [WL68] in terms of distinguishing non-isomorphic graphs. [Lou20, SYK19] present that message passing GNNs are shown to be a universal approximator under sufficient conditions on depth, width, nodes features, while the expressive power of GNNs is limited by their width and depth, as they are equivalent to LOCAL [Ang80, Lin92, NS95], a classical model used in the study the distributed algorithms that is itself Turing universal.

GNNs introduced above only consider nodes and edges, which may not be sufficient to handle the complex graph-structured data. Higher-order GNNs are developed to capture higher-order substructures and their connections, i.e subgraphs consisting of at least 3 nodes, beyond nodes and edges, so that we have another informative features. For example, $(k+1)$-LGNN [MBHSL18] and k-FGNN [MBHSL19] can approximate any equivariant function less powerful than equivariant $(k+1)$-WL.

Chapter 3

Metric Learning for Persistence-based Summaries and Applications to Graph Classification

3.1 Introduction

In recent years a new data analysis methodology based on a topological tool called persistent homology has started to attract momentum. The persistent homology is one of the most important developments in the field of topological data analysis, and there have been fundamental developments both on the theoretical front (e.g, [ELZ02, CZ09, CCSG$^+$09, CdS10, CdSGO16, BCNK18]), and on algorithms / implementations (e.g, [She12, BKRW14, CBGY14, DSW16, KS17, Bau16]). As we already seen in Chapter 2, intuitively, on the high level, given a domain X with a function $f : X \to \mathbb{R}$ on it, the persistent homology summarizes "features" of X across multiple scales simultaneously in a single summary called the *persistence diagram* (see the second picture in Figure 3.1). A persistence diagram consists of a multiset of points in the plane, where each point $p = (b,d)$ intuitively corresponds to the birth-time (b) and death-time (d) of some (topological) features of X w.r.t. f. Hence it provides a concise representation of X, capturing *multi-scale features* of it simultaneously. Furthermore, the persistent homology framework can be applied to complex data (e.g, 3D shapes, or graphs), and different summaries could be constructed by putting different descriptor functions on input data.

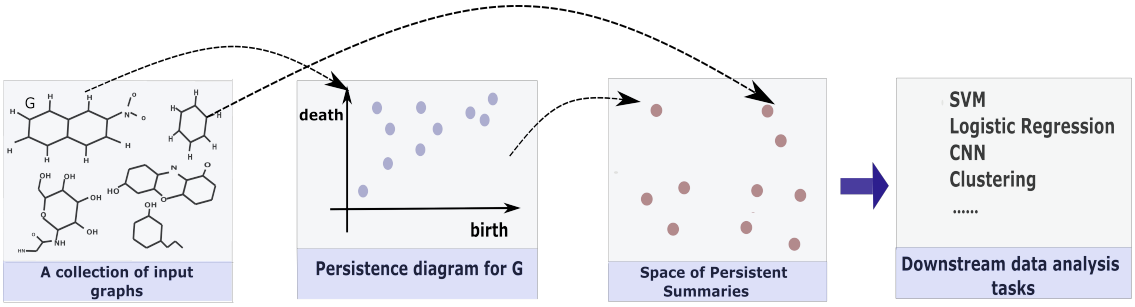


Figure 3.1. A persistence-based data analysis framework.

Due to these reasons, a new persistence-based feature vectorization and data analysis framework (Figure 3.1) has become popular. Specifically, given a collection of objects, say a set of graphs modeling chemical compounds, one can first convert each shape to a persistence-based representation. The input data can now be viewed as a set of points in a persistence-based feature space. Equipping this space with appropriate distance or kernel, one can then perform downstream data analysis tasks (e.g, clustering).

The original distances for persistence diagram summaries unfortunately do not lend themselves easily to machine learning tasks. Hence in the last few years, starting from the persistence landscape [Bub15], there have been a series of methods developed to map a persistence diagram to a vector representation to facilitate machine learning tools [RHBK15, AEK$^+$17, KFH18, CCO17, LMBB19].

In these approaches, when computing the distance or kernel between persistence summaries, the importance (weight) of different persistence features are often *pre-determined*. In persistence images [AEK$^+$17] and PWGK [KFH18], the importance of having a **weight-function** for the birth-death plane (containing the persistence points) has been emphasized and explicitly included in the formulation of their kernels. However, before using these kernels, the weight-function needs to be pre-set.

On the other hand, as recognized by [HKNU17], the choice of the weight-function should depend on the nature of the specific type of data at hand. For example, for the persistence diagrams computed from atomic configurations of molecules, features with small persistence

could capture the local packing patterns which are of utmost importance and thus should be given a larger weight; while in many other scenarios, small persistence leads to noise with low importance. However, in general researchers performing data analysis tasks may not have such prior insights on input data. Thus it is natural and highly desirable to *learn* a best weight-function from labelled data.

Main contributions of Chapter 3.

We study the problem of learning an appropriate metric (kernel) for persistence summaries from labelled data, and apply the learnt kernel to the challenging graph classification task.

(1) Metric learning for persistence summaries: We propose a new weighted-kernel (called *WKPI*), for persistence summaries based on persistence images representations. Our WKPI kernel is positive semi-definite and its induced distance is stable. A weight-function used in this kernel directly encodes the importance of different locations in the persistence diagram. We next model the metric learning problem for persistence summaries as the problem of learning (the parameters of) this *weight-function* from a certain function class. In particular, the metric-learning is formulated as an optimization problem over a specific cost function we propose. This cost function has a simple matrix view which helps both conceptually clarify its meaning and simplify the implementation of its optimization.

(2) Graph classification application: Given a set of objects with class labels, we first learn a best WKPI-kernel as described above, and then use the learned WKPI to further classify objects. We implemented this *WKPI-classification framework*, and apply it to a range of graph data sets. Graph classification is an important problem, and there has been a large literature on developing effective graph representations (e.g, [HK09, NST$^+$09, BRTH15, KGW16, SSL$^+$11, XJWL15, NPGK12], including the very recent persistent-homology enhanced WL-kernel [RBB19]), and graph neural networks (e.g, graph neural networks [YV15, NAK16, XHLJ19, VZ17, LMBB19, KSP$^+$18]) to classify graphs. The problem is challenging as graph data are less structured. We perform our WKPI-classification framework on various benchmark graph data sets as well as new

neuron-cell data sets. Our learnt WKPI performs consistently better than other persistence-based kernels. Most importantly, when compared with existing state-of-the-art graph classification frameworks, our framework shows similar or (sometimes significantly) better performance in almost all cases than the *best results* by existing approaches.

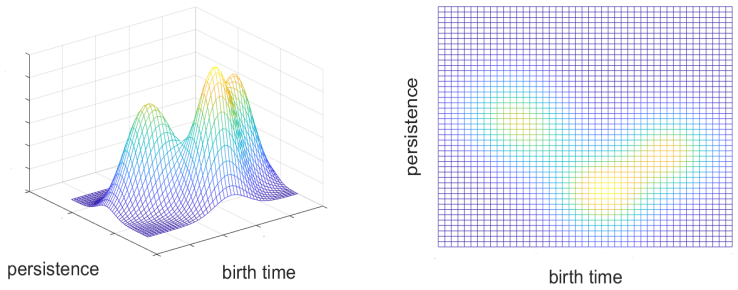
We note that [HKNU17] is the first to recognize the importance of using labelled data to learn a task-optimal representation of topological signatures. They developed an end-to-end deep neural network for this purpose, using a novel and elegant design of the input layer to implicitly learn a task-specific representation. Very recently, in a parallel and independent development of our work, Carrière et al. [CCI$^+$19] built an interesting new neural network based on the DeepSet architecture [ZKR$^+$17], which can achieve an end-to-end learning for multiple persistence representations *in a unified manner*. Compared to these developments, we instead explicitly formulate the metric-learning problem for persistence-summaries, and decouple the metric-learning (which can also be viewed as representation-learning) component from the downstream data analysis tasks. Also as shown in Section 3.4, our WKPI-classification framework (using SVM) achieves better results on graph classification datasets.

Outline of Chapter 3.

This chapter is organized as follows. In Section 3.2, we introduce some concepts and mathematical tools in the field of persistent homology. In Section 3.3, we present our weighted kernel, WKPI, and the metric learning framework. We then show experiments on neuron-cell classifications and graph classifications comparing to other topological based approaches, graph kernels and graph neural networks in Section 3.4. Finally, we make a conclusion in Section 3.5.

3.2 Persistence-based framework

We have introduced how we can construct a filtration and extract topological information from an object in Chapter 2. We also show what kinds of information can be encoded by persistent diagrams. Now given a collection of shapes Ξ, we can compute a persistence diagram $\mathrm{Dg}X$ for



(a) **(b)**

Figure 3.2. (a) shows the graph of a persistence surface (where z-axis is the function ρ_A), and (b) is its corresponding persistence image.

each $X \in \Xi$, which maps the set Ξ to a set of points in the space of persistence diagrams. There are natural distances defined for persistence diagrams, including the bottleneck distance and the Wasserstein distance, both of which have been well studied (e.g, stability under them [CSEH07, CSEHM10, CdSGO16]) with efficient implementations available [KMN17, KMN18]. However, to facilitate downstream machine learning tasks, it is desirable to further map the persistence diagrams to another "vector" representation. Below we introduce one such representation, called the persistence images [AEK$^+$17], as our new kernel is based on it.

Persistence images.

Let A be a persistence diagram (containing a multiset of persistence-points). Following [AEK$^+$17], set $T : \mathbb{R}^2 \to \mathbb{R}^2$ to be the linear transformation[1] where for each $(x,y) \in \mathbb{R}^2$, $T(x,y) = (x, y-x)$. Let $T(A)$ be the transformed diagram of A. Let $\phi_u : \mathbb{R}^2 \to \mathbb{R}$ be a differentiable probability distribution with mean $u \in \mathbb{R}^2$ (e.g, the normalized Gaussian where for any $z \in \mathbb{R}^2$, $\phi_u(z) = \frac{1}{2\pi\tau^2} e^{-\frac{\|z-u\|^2}{2\tau^2}}$).

Definition 3.2.1 ([AEK$^+$17])**.** *Let $\alpha : \mathbb{R}^2 \to \mathbb{R}$ be a non-negative weight-function for the persistence plane $\mathbb{R}^2$. Given a persistence diagram A, its* persistence surface $\rho_A : \mathbb{R}^2 \to \mathbb{R}$ *(w.r.t. α) is defined as: for any $z \in \mathbb{R}^2$, $\rho_A(z) = \sum_{u \in T(A)} \alpha(u)\phi_u(z)$.*

[1] In fact, we can define our kernel without transforming the persistence diagram. We use the transformation simply to follow the same convention as persistence images.

The persistence image is a discretization of the persistence surface. Specifically, fix a grid on a rectangular region in the plane with a collection $\mathscr{P}$ of N rectangles (pixels). The persistence image *for a diagram A is* $\mathrm{PI}_A = \{\ \mathrm{PI}[\mathbf{p}]\ \}_{\mathbf{p}\in\mathscr{P}}$ *consists of N numbers (i.e, a vector in* $\mathbb{R}^N$*), one for each pixel* $\mathbf{p}$ *in the grid* $\mathscr{P}$ *with* $\mathrm{PI}[\mathbf{p}] := \iint_{\mathbf{p}} \rho_A\, dydx$.

Figure 3.2 shows an example of persistence surface and persistence image from generated from a persistence diagram.

3.3 Metric learning frameworks

Suppose we are given a set of n objects Ξ (sampled from a hidden data space $\mathscr{S}$), classified into k classes. We want to use these labelled data to learn a good distance for (persistence image representations of) objects from Ξ which hopefully is more appropriate at classifying objects in the data space $\mathscr{S}$. To do so, below we propose a new persistence-based kernel for persistence images, and then formulate an optimization problem to learn the best weight-function so as to obtain a good distance metric for Ξ (and data space $\mathscr{S}$).

3.3.1 Weighted persistence image kernel (WKPI)

From now on, we fix the grid $\mathscr{P}$ (of size N) to generate persistence images (so a persistence image is a vector in $\mathbb{R}^N$). Let p_s be the center of the s-th pixel $\mathbf{p}_s$ in $\mathscr{P}$, for $s \in \{1,2,\cdots,N\}$. We now propose a new kernel for persistence images. A *weight-function* refers to a non-negative real-valued function on $\mathbb{R}^2$.

Definition 3.3.1. *Let* $\omega : \mathbb{R}^2 \to \mathbb{R}$ *be a weight-function. Given two persistence images* PI *and* PI′*, the* (ω-)weighted persistence image kernel (WKPI) *is defined as:* $k_w(\mathrm{PI},\mathrm{PI}') := \sum_{s=1}^{N} \omega(p_s) e^{-\frac{(\mathrm{PI}(s)-\mathrm{PI}'(s))^2}{2\sigma^2}}$.

<u>*Remark 0:*</u> We could use the persistence surfaces (instead of persistence images) to define the kernel (with the summation replaced by an integral). Since for computational purpose, one still needs to approximate the integral in the kernel via some discretization, we choose to present our

work using persistence images directly. Our Lemma 3.3.2 and Theorem 3.3.4 still hold (with slightly different stability bound) if we use the kernel defined for persistence surfaces.

Remark 1: One can choose the weight-function from different function classes. Two popular choices are: mixture of m 2D Gaussians; and degree-d polynomials on two variables.

Remark 2: There are other natural choices for defining a weighted kernel for persistence images. For example, we could use $k(\mathrm{PI},\mathrm{PI}') = \sum_{s=1}^{N} e^{-\frac{\omega(p_s)(\mathrm{PI}(s)-\mathrm{PI}'(s))^2}{2\sigma^2}}$, which we refer this as *altWKPI*. Alternatively, one could use the weight function used in PWGK kernel [KFH18] directly. Indeed, we have implemented all these choices, and our experiments show that our WKPI kernel leads to better results than these choices for almost all datasets (see Appendix Section A.2). In addition, note that PWGK kernel [KFH18] contains cross terms $\omega(x)\cdot\omega(y)$ in its formulation, meaning that there are quadratic number of terms (w.r.t the number of persistence points) to calculate the kernel, making it more expensive to compute and learn for complex objects (e.g, for the neuron data set, a single neuron tree could produce a persistence diagrams with hundreds of persistence points).

Lemma 3.3.2. *The WKPI kernel is positive semi-definite.*

The rather simple proof of the above lemma is in Appendix Section A.1.1. By Lemma 3.3.2, the WKPI kernel gives rise to a Hilbert space. We can now introduce the WKPI-distance, which is the *pseudo-metric* induced by the inner product on this Hilbert space.

Definition 3.3.3. *Given two persistence diagrams A and B, let* PI_A *and* PI_B *be their corresponding persistence images. Given a weight-function* $\omega : \mathbb{R}^2 \to \mathbb{R}$*, the* ($\omega$-weighted) WKPI-distance *is:*

$$\mathrm{D}_\omega(A,B) := \sqrt{k_w(\mathrm{PI}_A,\mathrm{PI}_A) + k_w(\mathrm{PI}_B,\mathrm{PI}_B) - 2k_w(\mathrm{PI}_A,\mathrm{PI}_B)}.$$

Stability of WKPI-distance.

Given two persistence diagrams A and B, two traditional distances between them are the bottleneck distance $d_B(A,B)$ and the p-th Wasserstein distance $d_{W,p}(A,B)$. Stability of these

two distances w.r.t. changes of input objects or functions defined on them have been studied [CSEH07, CSEHM10, CdSGO16]. Similar to the stability study on persistence images, below we prove WKPI-distance is stable w.r.t. small perturbation in persistence diagrams as measured by $d_{W,1}$. (Very informally, view two persistence diagrams A and B as two (appropriate) measures (with special care taken to the diagonals), and $d_{W,1}(A,B)$ is roughly the "earth-mover" distance between them to convert the measure corresponding to A to that for B.)

To simplify the presentation of Theorem 3.3.4, we use *unweighted persistence images w.r.t. Gaussian*, meaning in Definition 3.2.1, (1) the weight function α is the constant function $\alpha = 1$; and (2) the distribution ϕ_u is the Gaussian $\phi_u(z) = \frac{1}{2\pi\tau^2} e^{-\frac{\|z-u\|^2}{2\tau^2}}$. (Our result below can be extended to the case where ϕ_u is not Gaussian.) The proof of the theorem below follows from results of [AEK+17] and can be found in Appendix Section A.1.2.

Theorem 3.3.4. *Given a weight-function $\omega : \mathbb{R}^2 \to \mathbb{R}$, set $c_w = \|\omega\|_\infty = \sup_{z\in\mathbb{R}^2} \omega(z)$. Given two persistence diagrams A and B, with corresponding persistence images PI_A and PI_B, we have that: $\mathrm{D}_\omega(A,B) \leq \sqrt{\frac{20c_w}{\pi}} \cdot \frac{1}{\sigma\cdot\tau} \cdot d_{W,1}(A,B)$, where σ is the width of the Gaussian used to define our WKPI kernel (Def. 3.3.1), and τ is that for the Gaussian ϕ_u to define persistence images (Def. 3.2.1).*

Remark 3: We can obtain a more general bound for the case where the distribution ϕ_u is not Gaussian. Furthermore, we can obtain a similar bound when our WKPI-kernel and its induced WKPI-distance is defined using *persistence surfaces* instead of *persistence images*.

3.3.2 Optimization problem for metric-learning

Suppose we are given a collection of objects $\Xi = \{X_1, \ldots, X_n\}$ (sampled from some hidden data space $\mathscr{S}$), already classified (labeled) to k classes $\mathscr{C}_1, \ldots, \mathscr{C}_k$. In what follows, we say that $i \in \mathscr{C}_j$ if X_i has class-label j. We first compute the persistence diagram A_i for each object $X_i \in \Xi$. (The precise filtration we use to do so will depend on the specific type of objects. Later in Section 3.4, we will describe filtrations used for graph data). Let $\{A_1, \ldots, A_n\}$ be the

resulting set of persistence diagrams. Given a weight-function ω, its induced WKPI-distance between A_i and A_j can also be thought of as a distance for the original objects X_i and X_j; that is, we can set $\mathrm{D}_\omega(X_i,X_j) := \mathrm{D}_\omega(A_i,A_j)$. Our goal is to learn a good distance metric for the data space $\mathscr{S}$ (where Ξ are sampled from) from the labels. We will formulate this as learning a best weight-function ω^* so that its induced WKPI-distance fits the class-labels of X_i's best. Specifically, for any $t \in \{1,2,\cdots,k\}$, set:

$$cost_\omega(t,t) = \sum_{i,j\in\mathscr{C}_t} \mathrm{D}_\omega{}^2(A_i,A_j); \quad \text{and} \quad cost_\omega(t,\cdot) = \sum_{i\in\mathscr{C}_t, j\in\{1,2,\cdots,n\}} \mathrm{D}_\omega{}^2(A_i,A_j).$$

Intuitively, $cost_\omega(t,t)$ is the total in-class (square) distances for $\mathscr{C}_t$; while $cost_\omega(t,\cdot)$ is the total distance from objects in class $\mathscr{C}_t$ to all objects in Ξ. A good metric should lead to relatively smaller distance between objects from the same class, but larger distance between objects from different classes. We thus propose the following optimization problem, which is related to k-way spectral clustering where the distance for an edge (A_i,A_j) is $D^2_\omega(A_i,A_j)$:

Definition 3.3.5 (Optimization problem)**.** *Given a weight-function $\omega : \mathbb{R}^2 \to \mathbb{R}$, the* total-cost *of its induced WKPI-distance over Ξ is defined as: $TC(\omega) := \sum_{t=1}^{k} \frac{cost(t,t)}{cost(t,\cdot)}$. The* optimal distance problem *aims to find the best weight-function ω^* from a certain function class $\mathscr{F}$ so that the total-cost is minimized; that is: $TC^* = \min_{\omega\in\mathscr{F}} TC(\omega)$;*
and $\omega^* = \mathrm{argmin}_{\omega\in\mathscr{F}} TC(\omega)$.

Matrix view of optimization problem.

We observe that our cost function can be re-formulated into a matrix form. This provides us with a perspective from the Laplacian matrix of certain graphs to understand the cost function, and helps to simplify the implementation of our optimization problem, as several programming languages popular in machine learning (e.g Python and Matlab) handle matrix operations more efficiently (than using loops). More precisely, recall our input is a set Ξ of n objects with labels

from k classes. We set up the following matrices:

$$L = G - \Lambda; \quad \Lambda = \left[\Lambda_{ij}\right]_{n\times n}, \text{ where } \Lambda_{ij} = \mathrm{D}_\omega{}^2(A_i, A_j) \text{ for } i,j \in \{1,2,\cdots,n\};$$

$$G = \left[g_{ij}\right]_{n\times n}, \quad \text{where } g_{ij} = \begin{cases} \sum_{\ell=1}^{n} \Lambda_{i\ell} & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}$$

$$H = \left[h_{ti}\right]_{k\times n} \quad \text{where } h_{ti} = \begin{cases} \frac{1}{\sqrt{cost_\omega(t,\cdot)}} & i \in \mathscr{C}_t \\ 0 & otherwise \end{cases}$$

Viewing Λ as distance matrix of objects $\{X_1,\ldots,X_n\}$, L is then its Laplacian matrix. We have the following main theorem, which essentially is similar to the trace-minimization view of k-way spectral clustering (see e.g, Section 6.5 of [KCS11]). The proof for our specific setting is in Appendix A.1.3.

Theorem 3.3.6. *The total-cost can also be represented by $TC(\omega) = k - \mathrm{Tr}(HLH^T)$, where $\mathrm{Tr}(\cdot)$ is the trace of a matrix. Furthermore, $HGH^T = \mathbf{I}$, where $\mathbf{I}$ is the $k \times k$ identity matrix.*

Note that all matrices, L, G, Λ, and H, are dependent on the (parameters of) weight-function ω, and in the following corollary of Theorem 3.3.6, we use the subscript of ω to emphasize this dependence.

Corollary 3.3.7. *The Optimal distance problem is equivalent to*

$$\min_{\omega} \left(k - \mathrm{Tr}(H_\omega L_\omega H_\omega^T)\right), \quad \textit{subject to } \; H_\omega G_\omega H_\omega^T = \mathbf{I}.$$

Solving the optimization problem.

In our implementation, we use (stochastic) gradient descent to find a (locally) optimal weight-function ω^* for the minization problem. Specifically, given a collection of objects Ξ with labels from k classes, we first compute their persistence diagrams via appropriate filtrations, and obtain a resulting set of persistence diagrams $\{A_1,\ldots,A_n\}$. We then aim to find the best

parameters for the weight-function ω^* to minimize $Tr(HLH^T) = \sum_{t=1}^{k} h_t L h_t^T$ subject to $HGH^T = I$ (via Corollary 3.3.7). For example, assume that the weight-function ω is from the class $\mathscr{F}$ of mixture of m number of 2D non-negatively weighted (spherical) Gaussians. Each weight-function $\omega : \mathbb{R}^2 \to \mathbb{R} \in \mathscr{F}$ is thus determined by $4m$ parameters $\{x_r, y_r, \sigma_r, w_r \mid r \in \{1, 2, \cdots, m\}\}$ with $\omega(z) = w_r e^{-\frac{(z_x - x_r)^2 + (z_y - y_r)^2}{\sigma_r^2}}$. We then use (stochastic) gradient decent to find the best parameters to minimize $Tr(HLH^T)$ subject to $HGH^T = I$. Note that the set of persistence diagrams / images will be fixed through the optimization process.

From the proof of Theorem 3.3.6 (in Appendix A.1.3), it turns out that condition $HGH^T = \mathbf{I}$ is satisfied as long as the multiplicative weight w_r of each Gaussian in the mixture is non-negative. Hence during the gradient descent, we only need to make sure that this holds [2]. It is easy to write out the gradient of $TC(\omega)$ w.r.t. each parameter $\{x_r, y_r, \sigma_r, w_r \mid r \in \{1, 2, \cdots, m\}\}$ **in matrix form**. For example, $\frac{\partial TC(\omega)}{\partial x_r} = -(\sum_{t=1}^{k} \frac{\partial h_t}{\partial x_r} L h_t^T + h_t \frac{\partial L}{\partial x_r} h_t^T + h_t L \frac{\partial h_t^T}{\partial x_r})$; where $h_t = \left[h_{t1}, h_{t2}, ..., h_{tn}\right]$ is the t-th row vector of H. While this does not improve the asymptotic complexity of computing the gradient (compared to using the formulation of cost function in Definition 3.3.5), these matrix operations can be implemented much more efficiently than using loops in languages such as Python and Matlab. For large data sets, we use stochastic gradient decent, by sampling a subset of $s << n$ number of input persistence images, and compute the matrices H, D, L, G as well as the cost using the subsampled data points. The time complexity of one iteration in updating parameters is $O(s^2 N)$, where N is the size of a persistence image (recall, each persistence image is a vector in $\mathbb{R}^N$). In our implementation, we use Armijo-Goldstein line search scheme to update the parameters in each (stochastic) gradient decent step. The optimization procedure terminates when the cost function converges or the number of iterations exceeds a threshold. Overall, the time complexity of our optimization procedure is $O(\mathrm{R} s^2 N)$ where R is the number of iterations, s is the minibatch size, and N is the size (# pixels) of a single persistence image.

[2] In our implementation, we add a penalty term $\sum_{r=1}^{m} \frac{c}{exp(w_r)}$ to total-cost $k - Tr(HLH^T)$, to achieve this in a "soft" manner.

3.4 Experiments

We show the effectiveness of our metric-learning framework and the use of the learned metric via graph classification applications. In particular, given a set of graphs $\Xi = \{G_1, \ldots, G_n\}$ coming from k classes, we first compute the unweighted persistence images A_i for each graph G_i, and apply the framework from Section 3.3.1 to learn the "best" weight-function $\omega^* : \mathbb{R}^2 \to \mathbb{R}$ using these persistence images $\{A_1, \ldots, A_n\}$ and their labels. We then perform graph classification using kernel-SVM with the learned ω^*-WKPI kernel. We refer to this framework as *WKPI-classification* framework. We show two sets of experiments. Section 3.4.1 shows that our learned WKPI kernel significantly outperforms existing persistence-based representations. In Section 3.4.2, we compare the performance of WKPI-classification framework with various state-of-the-art methods for the graph classification task over a range of data sets. More details / results can be found in Appendix Section A.2.

Setup for our WKPI-based framework.

In all our experiments, we assume that the weight-function comes from the class $\mathscr{F}$ of mixture of m 2D non-negatively weighted Gaussians as described in the end of Section 3.3.2. We take m and the width σ in our WKPI kernel as hyperparameters: Specifically, we search among $m \in \{3,4,5,6,7,8\}$ and $\sigma \in \{0.001, 0.01, 0.1, 1, 10, 100\}$. The $10*10$-fold nested cross validation are applied to evaluate our algorithm: There are 10 folds in outer loop for evaluation of the model with selected hyperparameters and 10 folds in inner loop for hyperparameter tuning. We then repeat this process 10 times (although the results are extremely close whether repeating 10 times or not). Our optimization procedure terminates when the change of the cost function remains $\leq 10^{-4}$ or the iteration number exceeds 2000.

One important question is to initialize the centers of the Gaussians in our mixture. There are three strategies that we consider. (1) We simply sample m centers in the domain of persistence images randomly. (2) We collect all points in the persistence diagrams $\{A_1, \ldots, A_n\}$ derived from the training data Ξ, and perform a k-means algorithm to identify m means. (3) We perform a

Table 3.1. Classification accuracy on neuron dataset. Our results are WKPI-km and WKPI-kc.

Datasets	Existing approaches			Alternative metric learning	
	PWGK	SW	PI-PL	altWKPI	trainPWGK
NEURON-BINARY	80.5±0.4	85.3±0.7	83.7±0.3	82.1±2.1	84.6±2.4
NEURON-MULTI	45.1±0.3	57.6±0.6	44.2±0.3	54.3±2.3	49.7±2.4
Average	62.80	71.45	63.95	68.20	67.15
Datasets	Our WKPI framework				
	WKPI-km			WKPI-kc	
NEURON-BINARY	**89.6 ±2.2**			86.4±2.4	
NEURON-MULTI	56.6±2.7			**59.3±2.3**	
Average	**73.10**			72.85	

k-center algorithm to those points to identify m centers. Strategies (2) and (3) usually outperform strategy (1). Thus in what follows we only report results from using k-means and k-centers as initialization, referred to as *WKPI-kM* and *WKPI-kC*, respectively.

3.4.1 Comparison with other persistence-based methods

We compare our methods with state-of-the-art persistence-based representations, including the Persistence Weighted Gaussian Kernel (PWGK) [KFH18], original Persistence Image (PI) [AEK$^+$17], and Sliced Wasserstein (SW) Kernel [CCO17]. Furthermore, as mentioned in *Remark 2* after Definition 3.3.1, we can learn weight functions in PWGK by the optimizing the same cost function (via replacing our WKPI-distance with the one computed from PWGK kernel); and we refer to this as trainPWGK. We can also use an alternative kernel for persistence images as described in *Remark 2*, and then optimize the same cost function using distance computed from this kernel; we refer to this as altWKPI. We will compare our methods both with existing approaches, as well as with these two alternative metric-learning approaches (trainPWGK and altWKPI).

Description of neuron datasets.

Neuron cells have natural tree morphology (see Figure 3.3 (a) for an example), rooted at the cell body (soma), with dentrite and axon branching out. Furthermore, this tree morphology

is important in understanding neurons. Hence it is common in the field of neuronscience to model a neuron as a (geometric) tree (see Figure 3.3 (b) for an example downloaded from NeuroMorpho.Org[ADH07]).

Our NEURON-BINARY dataset consists of 1126 neuron trees classified into two (primary) classes: *interneuron* and *principal neurons* (data partly from the Blue Brain Project [MMR+15] and downloaded from http://neuromorpho.org/). The second NEURON-MULTI dataset is a refinement of the 459 interneuron class into four (secondary) classes: *basket-large, basket-nest, neuglia* and *martino*.

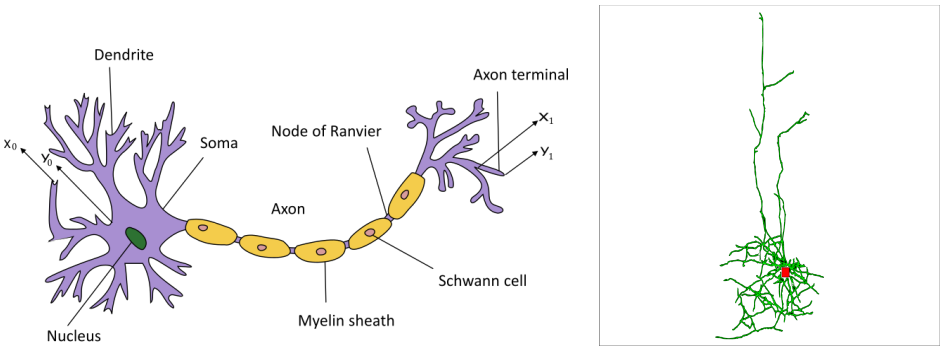


Figure 3.3. (a) An neuron cell (downloaded from Wikipedia)and (b) an example of a neuron tree (downloaded from NeuroMorpho.Org).

Generation of persistence diagrams.

Given a neuron tree T, following [LWA+17], we use the descriptor function $f : T \to \mathbb{R}$ where $f(x)$ is the geodesic distance from x to the root of T along the tree. To differentiate the dendrite and axon part of a neuron cell, we further negate the function value if a point x is in the dendrite. We then use the union of persistence diagrams A_T induced by both the sublevel-set and superlevel-set filtrations w.r.t. f. Under these filtrations, intuitively, each point (b,d) in the birth-death plane $\mathbb{R}^2$ corresponds to the creation and death of certain branch feature for the input neuron tree. The set of persistence diagrams obtained this way (one for each neuron tree) is the input to our WKPI-classification framework.

Results on neuron datasets.

Neuron-Binary dataset consists of 1126 neuron trees from two classes; while **Neuron-Multi** contains 459 neurons from four classes. As the number of trees is not large, we use all training data to compute the gradients in the optimization process instead of mini-batch sampling. Persistence images are both needed for the methodology of [AEK+17] and as input for our WKPI-distance, and its resolution is fixed at roughly 40×40 (see Appendix Section A.2.1 for details). For persistence image (PI) approach of [AEK+17], we experimented both with the unweighted persistence images (PI-CONST), and one, denoted by (PI-PL), where the weight function $\alpha : \mathbb{R}^2 \to \mathbb{R}$ is a simple piecewise-linear (PL) function adapted from what's proposed in [AEK+17]; see Appendix Section A.2.1 for details. Since PI-PL performs better than PI-CONST on both datasets, Table 3.1 only shows the results of PI-PL. The classification accuracy of various methods is given in Table 3.1. Our results are consistently better than other topology-based approaches, as well as alternative metric-learning approaches; not only for the neuron datasets as in Table 3.1, but also for graph benchmark datasets shown in Table A.2 of Appendix Section A.2.2, and often by a large margin. In Appendix Section A.2.2, we also show the heatmaps indicating the learned weight function $\omega : \mathbb{R}^2 \to \mathbb{R}$.

3.4.2 Graph classification task

We use a range of benchmark datasets: (1) several datasets on graphs derived from small chemical compounds or protein molecules: **NCI1** and **NCI109** [SSL+11], **PTC** [HKKS01], **PROTEIN** [BOS+05], **DD** [DD03] and **MUTAG** [DLdCD+91]; (2) two datasets on graphs representing the response relations between users in Reddit: **REDDIT-5K** and **REDDIT-12K** [YV15]; and (3) two datasets on IMDB networks of actors/actresses: **IMDB-BINARY**, and **IMDB-MULTI**. Their statistics are shown in Table 3.2. See Appendix Section 2.2 for descriptions of these datasets.

Many graph classification methods have been proposed in the literature, with different methods performing better on different datasets. Thus we include multiple approaches to

Table 3.2. Statistics of the benchmark graph datasets

Dataset	#classes	#graphs	average #nodes	average #edges
NCI1	2	4110	29.87	32.30
NCI109	2	4127	29.68	31.96
PTC	2	344	14.29	14.69
PROTEIN	2	1113	39.06	72.82
DD	2	1178	284.32	715.66
IMDB-BINARY	2	1000	19.77	96.53
IMDB-MULTI	3	1500	13.00	65.94
REDDIT-5K	5	4999	508.82	594.87
REDDIT-12K	11	12929	391.41	456.89

compare with, to include state-of-the-art results on different datasets: Weisfeiler-Lehman kernel (WL)[SSL$^+$11], Weisfeiler-Lehman optimal assignment kernel (WL-OA)[KGW16], Deep Graphlet kernel (DGK)[YV15], the very recent persistent Weisfeiler-Lehman kernel (P-WL-UC) [RBB19], Sliced Wasserstein kernel [CCO17], and Persistence Fisher kernel[LY18]; two graph neural networks: PATCHYSAN (PSCN) [NAK16] and Graph Isomorphism Network (GIN)[XHLJ19]. We also compare trainPWGKand altWKPIas we introduced above.

Classification results.

To generate persistence summaries, we need a meaningful descriptor function on input graphs. We consider two choices: (a) the *Ricci-curvature function* $f_c : G \to \mathbb{R}$, where $f_c(x)$ is the discrete Ricci curvature for graphs as introduced in [LLY11]; and (b) *Jaccard-index function* $f_J : G \to \mathbb{R}$ which measures edge similarities in a graph. See Appendix Section A.2.2 for details. Graph classification results are in Table 3.3: Ricci curvature function is used for the small chemical compounds datasets (NCI1, NCI9, PTC and MUTAG), while Jaccard function is used for proteins datasets (PROTEIN and DD) and the social/IMDB networks (IMDB's and REDDIT's). Results of previous methods are taken from their respective papers. Comparisons with **more methods** (including with other topology-based methods such as SW [CCO17]) are in Appendix Section A.2.2. We rerun the two best performing approaches GIN and RetGK using

Table 3.3. Graph classification accuracy.

Dataset	Graph kernel and GNN approaches					
	RetGK	WL	DGK	P-WL-UC	PSCN	GIN
NCI1	84.5	85.4	80.3	85.6	76.3	82.7
NCI109	-	84.5	80.3	85.1	-	-
PTC	62.5	55.4	60.1	63.5	62.3	66.6
PROTEIN	75.8	71.2	75.7	75.9	75.0	76.2
DD	81.6	78.6	-	78.5	76.2	-
MUTAG	**9**0.3	84.4	87.4	85.2	89.0	90.0
IMDB-BINARY	71.9	70.8	67.0	73.0	71.0	75.1
IMDB-MULTI	47.7	49.8	44.6	-	45.2	**5**2.3
REDDIT-5K	56.1	51.2	41.3	-	49.1	57.5
REDDIT-12K	**4**8.7	32.6	32.2	-	41.3	-
Dataset	Existing TDA approaches		Alternative metric learning		Our approaches	
	SW	PF	trainPWGK	altWKPI	WKPI-kM	WKPI-kC
NCI1	80.1	81.7	76.5	77.4	**87.5**	84.5
NCI109	75.5	78.5	77.2	81.2	85.9	**87.4**
PTC	**64.5**	62.4	62.5	64.2	61.7	68.1
PROTEIN	76.4	75.2	74.8	75.1	**78.5**	75.2
DD	78.9	79.4	76.4	72.5	**82.0**	80.3
MUTAG	87.1	85.6	86.4	88.5	85.8	88.3
IMDB-BINARY	69.6	71.2	71.8	67.3	70.7	**75.4**
IMDB-MULTI	48.7	48.6	45.8	45.3	46.4	49.5
REDDIT-5K	53.8	56.2	53.5	54.7	59.1	**59.5**
REDDIT-12K	48.3	47.6	43.7	42.1	47.4	48.4

the exactly same nested cross validation setup as ours. The results are also in Appendix Section A.2.2, which are similar to those in Table 3.3. Except for **MUTAG** and **IMDB-MULTI**, the performances of our WKPI-framework are similar or better than **the best of other methods**. Our WKPI-framework performs well on both chemical graphs and social graphs, while some of the earlier work tend to work well on one type of the graphs. Furthermore, note that the chemical / molecular graphs usually have attributes associated with them. Some existing methods use these attributes in their classification [YV15, NAK16, ZWX$^+$18]. Our results however are obtained **purely based on graph structure** without using any attributes.

3.5 Conclusion

This chapter introduces a new weighted-kernel for persistence images (WKPI), together with a metric-learning framework to learn the best weight-function for WKPI-kernel from labelled data. We apply the learned WKPI-kernel to the task of graph classification, and show that our new framework achieves similar or better results than the best results among a range of previous approaches.

In our current framework, only a single descriptor function of each input object is used to derive a persistence-based representation. It will be interesting to extend our framework to leverage multiple descriptor functions (so as to capture different types of information) effectively. Recent work on multidimensional persistence would be useful in this effort. Another interesting question is to study how to incorporate categorical attributes associated to graph nodes effectively. Real-valued attributed can be used as a descriptor function to generate persistence-based summaries. But the handling of categorical attributes via topological summary is much more challenging, especially when there is no (prior-known) correlation between these attributes (e.g, the attribute is simply a number from $\{1,2,\cdots,s\}$, coming from s categories. The indices of these categories may carry no meaning).

This Chapter 3, in full, is a reprint of the material as it appears in Learning Metric for

Persistence-based Summaries and Applications for Graph Classification, 2019. Zhao, Qi; Wang, Yusu. Conference on Neural Information Processing System (NeurIPS), 2019. The dissertation author was the primary investigator and author of this paper.

Chapter 4

Persistence Enhanced Graph Neural Network

4.1 Introduction

Deep learning methods have achieved immense success in different domains such as computer vision and natural language processing [GBC16]. While deep neural networks have shown strong performance on image or text data, their learning power is yet to be fully exploited on graph-structured data. At the same time, data with a latent graph structure is ubiquitous in modern data science. It is highly desirable to develop deep learning techniques that best suite graph structures, such as social network, knowledge network, brain connectivity network, etc.

Earlier works on Graph Neural Networks (GNNs) [GMS05, SGT+09] use recursive networks. Those GNNs process the graph using a set of neurons, each corresponding to a node in the graph. The neurons update nodes representation and exchange information from linked neighbor nodes iteratively until reaching equilibrium.

Inspired by the power of convolutional networks on image and text data, different ideas have been proposed to implement the "convolution" on graph structures. There are two main directions, spectral convolutions and spatial convolutions. Spectral convolutional networks [BZSL14, DBV16] apply convolutions to the spectral domain or the frequency domain of the input graph. These methods tend to be efficient, but are highly graph-dependent.

In a more explicit manner, spatial convolutional networks implement convolutions on

graphs. The feature representation of each node is iteratively updated by aggregating information from immediate neighbors [HYL17, XHLJ19] or a receptive field determined by special methods [NAK16]. A transformation of the information from neighbors - either linear or non-linear - can be learned through training. The node representation information transferred between vertices are called *messages*. [VCC+18] used self-attention mechanism to further refine the messages based on local information, namely, features of source and target nodes.

In spatial convolutions, it is essential to have shared filter parameters across different parts of the graph. However, it has been observed that the filters should be adaptive to different local graph structures. In particular, node degrees have been used to reweight the messages or as additional features of node representations [KW17, MBM+17]. This way, messages relevant to hub nodes with high degrees will be different from messages between normal nodes. However, node degree is only the simplest graph structural property. There are much richer and advanced structural information that should be exploited in order to develop structure-adaptive convolutional filters.

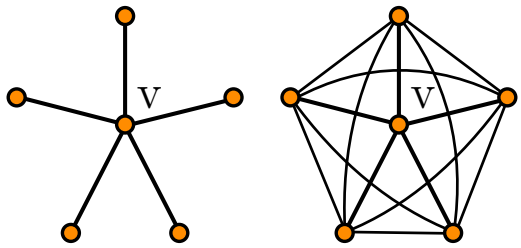


Figure 4.1. A node v is critical in a tree-structured neighborhood with no loops (left). But it is dispensable in a clique-structured neighborhood with 10 triangular loops containing v (right).

In this chapter, we propose a novel and principled approach to maximally leverage the structural information in spatial graph convolution. In particular, for a node of the graph, we are interested in the *loopiness* of its neighborhood, i.e., how well its neighbors are inter-connected. Such structural property measures how critical the center node, v, is in information flow. At one extreme, if the neighborhood forms a loop-free tree, v is indispensable as any information

between these neighbors has to pass v locally.[1] At another extreme, a clique neighborhood with N_v neighbors has $\binom{N_v}{2} = N_v(N_v-1)/2$ many triangular loops that contain v. In such case, any two neighbors can easily exchange information without going through v. See Figure 4.1 for illustrations. Such loopiness measure of a neighborhood should be fully exploited in graph learning, as it measures the information transmission efficiency of each node.

However, a direct cycle-counting within the neighborhood of v is insufficient. One major challenge is how to generalize to the cases when a graph is endowed with a metric, i.e., edges are annotated with different weights. These edge weights may come as part of the input information, e.g., frequency of communication between nodes in a social network, distance between nodes in a traffic network, etc. In other cases, these edge weights may be derived directly from the graph structural information. We need a robust 'cycle-counting' measure which should be stable to such graph metrics, i.e., does not change much when a metric is slightly perturbed.

To this end, we will use a principled mathematical tool, *persistent homology*, as a novel structural information measurement for graph convolutional networks. Persistent homology [ELZ02, EH10], as a modern adaption of the classic algebraic topology, can measure topological information carried in a metric space. As we introduced in Chapter 2, in graph context, persistent homology not only counts the number of loops, but also measures the saliency of all loops in view of a given metric. The result of [CSEH07, CSEHM10] shows that the structural information captured in persistent homology is stable with regard to certain perturbation of the underlying metric.

Main contributions of Chapter 4.

Our contribution in this chapter is two-fold.

A graph neural network architecture leveraging persistent homology information: We propose *Persistence Enhanced Graph Network* (PEGN), a novel network architecture for graphs, that uses persistent homology information in a data-driven manner. Based on the input persistent

[1] The neighbors may communicate through other parts of the graph, but at a higher expense (longer route).

homology information of different neighborhoods, we train a separate network to reweight messages between nodes. This empowers the spatial graph convolution to be highly adaptive with regard to different local structures.

Node classification application: We validate our proposed network on a broad spectrum of synthetic and real world graph datasets. Our method outperforms existing methods that only use node-feature-based attention or node degree information. This confirms the power of advanced structural information, i.e., persistent homology, in graph learning.

To the best of our knowledge, *we are the first to exploit advanced topological information beyond node degrees in spatial convolutional graph networks*. Note that spectral convolution methods, which run convolutions on the spectral or Fourier space, indirectly use advanced structural information. But for these methods, it is much harder to control the convolution with regard to different local structures as we do.

More on related Work.

Persistent homology plays a crucial role in topological data analysis, which extracts topological information from various geometric objects such as shapes, images or point clouds. Much progress has been made both on the theoretical front (e.g, [ELZ02, CZ09, CCSG$^+$09, CdS10, CdSGO16]) and the computational efficiency (e.g, [She12, BKRW14, CBGY14, DSW16, KS17, Bau16]). The extracted topological information has been used as powerful features in various contexts [HKNU17, AEK$^+$17, CCO17, KFH18]. Advanced topological methods have been developed for image segmentation [WCW$^+$17, HLSC19], clustering [NQWC17] and regularization of classifiers [CNBW19].

For graphs, persistent homology information has been used as a global structural signature for whole graph classification [RBB19, LWA$^+$17, ZW19]. But no existing methods use such information as a local structural information to improve the adaptability of graph convolutional networks.

Outline of Chapter 4.

This chapter is organized as follows. We first introduce what information a persistence diagram of a subgraph can capture from a local view in Section 4.2. Inspired by these information, we propose the Persistence Enhanced Graph Network (PEGN). We present details of the network architecture in Section 4.3. Then in Section 4.4, we show that our approach has achieved or matched the state-of-the-art across various graph benchmark datasets, in particular on larger and denser graphs. Finally, we introduce an application of graph neural networks enhanced by geometry and topology on material discovery in Section 4.5.

4.2 Persistence diagrams from subgraphs

In this section, we discuss what kinds of meaningful information we can capture from a specific kind of persistence diagrams generated from subgraphs. Given a graph $G(V,E)$, a subgraph G_u can be picked around a node $u \in V$ by selecting a set of nodes closed to u. One method is to pick its $q-hop$ neighbourhood ($q \geq 1$) and another is to apply a random walk starting from u. The filtration function f defined over node set included in G_u is the minimum geodesic distance d_{ux} between a node x and u. Naturally, a persistence diagram can be computed as illustrated in Chapter 3. The condition of edges is a little bit more complicated. The subgraph G_{uv} around an edge (u,v) can be constructed from the union or intersection of G_u and G_v, and the filtration function f over node set V_{uv} of G_{uv} can be defined as

$$\begin{aligned} f(x) &= min\{d_{xu}, d_{xv}\}, \quad or \\ f(x) &= max\{d_{xu}, d_{xv}\} \end{aligned} \tag{4.1}$$

where $x \in G_{uv}$. Or more simply, $f(x)$ can be defined as d_{xu} or d_{xv} directly.

In Chapter 2, we have introduced global topological information extracted by persistent homology from graphs. In this section, we focus on the local homological features within subgraphs. Define a level-r graph node to node u in graph G if its distance to u is r. From $\mathrm{Dg}^q G_u$,

the persistence diagram generated from q-hop neighborhood subgraph of node u as illustrated above, we can recover numerous local topology information of u as indicated in the Appendix of [TW19]:

- Degree of u, number of triangles incident on u, clustering coefficient at u.
- For any $r \leq q$, the number of level-r nodes to u.
- For any $r < q$, the number of crossing edges from level-r to level-$(r+1)$.
- For any $r \leq q$, the number of edges among level-r nodes.
- For any $r \leq q$, the persistence diagram $\mathrm{Dg}^r G_u$.
- The sequence of lengths of shortest system of loops passing through u.
- ...

Simply speaking, this subgraph persistence diagram provides us sufficient local topology information about a node and how it interacts with nodes within its q-hop neighborhood. This observation encourages us to develop a graph neural network leveraging subgraph persistence diagrams.

4.3 Persistence enhanced graph network

A Persistence Enhanced Graph Network (PEGN) is a spatial GNN. As we introduced in Chapter 2, the convolution can be viewed as a message passing framework. Messages are passed between nodes in order to update their feature representation. After a fixed number of iterations, the feature representation of each node is used for classification or other tasks. Here we focus on a node classification task, in which node representations are used to predict labels of all nodes. In a graph classification task, these node representations can be aggregated to a graph representation and be fed into a classifier.

To effectively incorporate structural information into the framework, we propose a separate network, Persistence Image Network (PIN), converting persistent homology information into message reweighting vectors. These vectors are used to reweight messages and to effectively improve the graph convolution performance. See Figure 4.2 for the architecture of our network.

We first describe details of PEGN, such as how messages are computed and passed between nodes, and how the messages are reweighted. Next, we show how persistent homology information are converted into message reweighting vectors by PIN.

4.3.1 PEGN: Message reweighting graph convolution

A graph neural network with L layers updates node feature representations for L times. Together with the input node features, we have $L+1$ node feature representations, $H^\ell = [\vec{h}_1^\ell, \vec{h}_2^\ell, \cdots, \vec{h}_N^\ell]$, $\vec{h}_n^\ell \in \mathbb{R}^{d_\ell}$, $\ell = 0, \cdots, L$. Here N is the number of nodes in the graph. d_ℓ is the feature dimension for the ℓ-th layer representation. We denote by H^0 the input node features. H^L is the final layer feature representations and will be used for prediction.

A convolutional layer generates the ℓ-th layer representation using the $(\ell-1)$-th layer representation. To compute the representation of node u, $\vec{h}_u^\ell$, we use the previous layer representations of u and its immediate neighbors, $\overline{\mathcal{N}}(u) = \{u\} \cup \mathcal{N}(u)$. These representations are transformed using a transformation matrix W^ℓ and are aggregated, formally,

$$\vec{h}_u^\ell = \sigma\left(\sum_{v \in \overline{\mathcal{N}}(u)} W^\ell \vec{h}_v^{\ell-1}\right). \tag{4.2}$$

The transformation matrix W^ℓ for the ℓ-th layer is learned in training. The additional function σ is the nonlinear transformation, e.g., ReLU. The transformed representation $W^\ell \vec{h}_v^{\ell-1}$ is the message passed from node v to node u. Note that all messages share a same transformation matrix W^ℓ.

To incorporate structural information, we introduce additional message reweighting vectors, $\tau_{v\to u}^\ell$. Reweighting vectors have the same length as the number of channels of a message.

They are different for different edges, depending on the structural information associated with the edge. Formally, we write the representation updating equation as

$$\vec{h}_u^{\ell} = \sigma\left(\sum_{v\in\overline{\mathcal{N}}(u)} \mathrm{diag}(\tau_{v\to u}^{\ell}) W^{\ell}\vec{h}_v^{\ell-1}\right). \tag{4.3}$$

Node degree and self-attention mechanism have been applied to reweight messages. However, the structural information is much richer than simply node degree.

We propose to use persistent homology describing topology of local neighborhood graphs of u and v to generate the reweighting vector $\tau_{v\to u}^{\ell}$. It remains to explain how persistent homology information, namely, persistence diagrams can be transformed into the reweighting vector using our Persistence Image Network (PIN). See Figure 4.3 for the architecture. First, it converts a persistence diagram into a fixed-length vector, called the persistence image. Second, it converts persistence images of u and v into a reweighting vector using a multilayer perceptron (MLP).

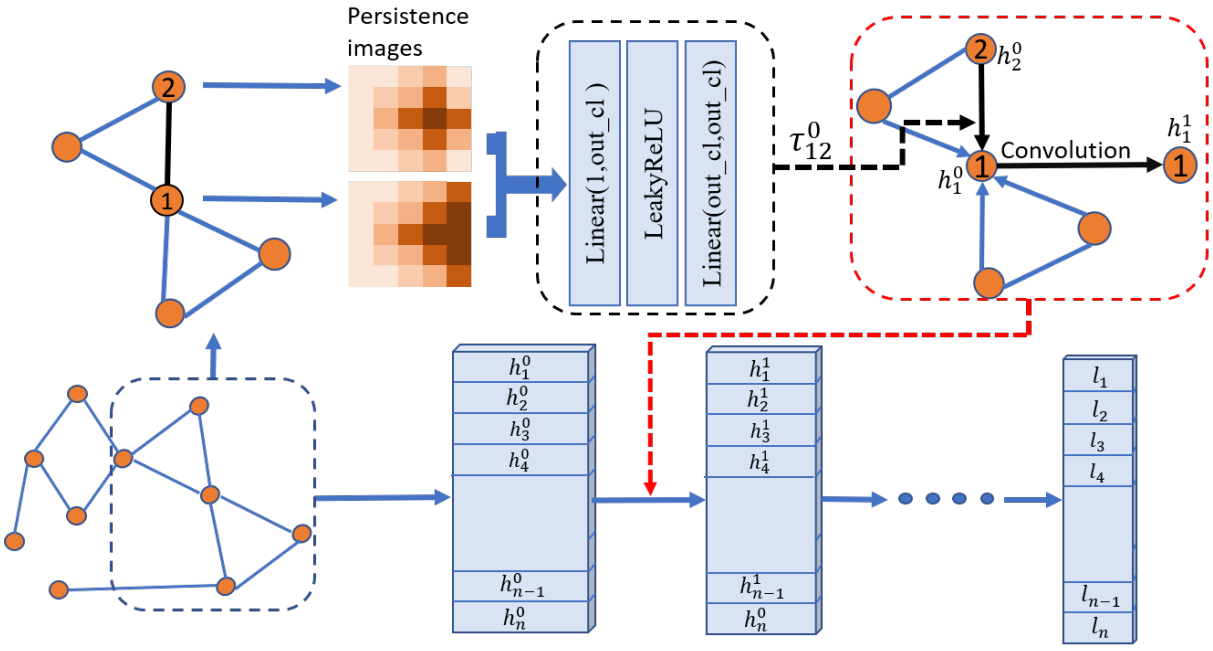


Figure 4.2. The framework of Persistence Enhanced Graph Network

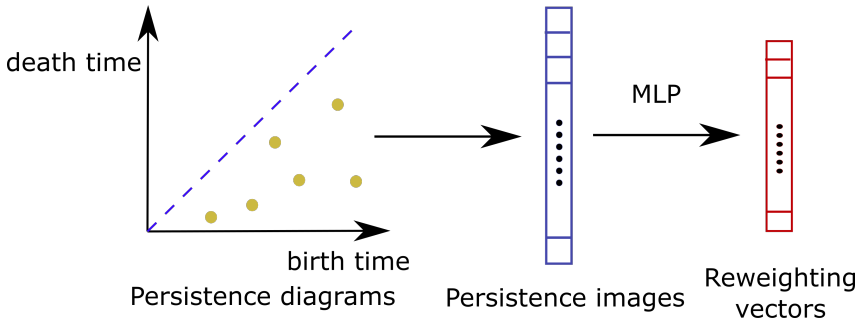


Figure 4.3. The framework of Persistence Image Network

4.3.2 Persistence images from subgraphs

We describe how the demanded persistence images are generated in this section. We have introduced how we can generate a persistence diagram for a q-hop neighborhood subgraph for each node and what kinds of information we can discover from it in Section 4.2.

Next, we need to vectorize the persistence diagrams extracted from neighborhoods to facilitate the neural networks frameworks we will introduce later. Recall what we introduced in Chapter 3, the persistence image is a stable vectorization of a persistence diagram to perturbations under 1-Wasserstein distance $d_{W,1}$. If we extract topological features from edge-wise subgraph G_{uv}, denote its corresponding persistence images as $PI(u,v)$. If we extract those from node-wise subgraph G_u, then the persistence image is denoted as $PI(u)$. Define the persistence information vector of any edge (u,v) as

$$\psi_{uv} = \begin{cases} PI(u,v) & \text{in edge-wise subgraph} \\ (PI(u), PI(v)) & \text{in node-wise subgraph} \end{cases} \tag{4.4}$$

where $(\cdot,\cdot)$ is the simple concatenation.

4.3.3 How persistence images improve GNNs

A common way to utilize the neighborhood subgraph persistence image $PI(u)$ in GNN is to take it as part of node features of u. In other words, we can take the neighborhood topology

information as the feature of a node itself. However, as we indicated in Section 4.1, these local topology in subgraphs have notable influence on interactions and message passing between nodes. Hence we focus on exploring how we can augment message passing between nodes by local persistence summaries instead of taking them as node features.

Next we present how ψ_{uv} acts on the reweighting parameter τ_{uv}^k in graph convolution processes and promotes the performance of GNNs. See Figure 4.2 for an illustration of our framework.

Start from the setup that τ_{uv}^k is a scalar, there is a function $f^k : \mathbb{R}^l \to \mathbb{R}$ mapping ψ_{uv} to τ_{uv}^k where l is the dimension of persistence information vectors. It is desirable that f^k is a learnable parametric function with the idea of data-driven mechanism. As the multi-layer perceptrons (MLPs) are universal function approximators shown by Cybenko's theorem [Cyb89] and easily carried into more complicated neural network models, we adopt MLPs to approximate the mapping function f^k. Notice the aggregations of messages are probably large arbitrarily, softmax functions are supposed to be applied for normalization after the MLPs processing. More exactly, denote the MLP and softmax function in the k_{th} layer as g^k and s^k respectively, the function approximating τ_{uv}^k is

$$\begin{aligned} f^k(\psi_{uv}) &= s^k(g^k(\psi_{uv})) \\ &= \frac{exp(g^k(\psi_{uv}))}{\sum_{x \in \mathcal{N}(u)} exp(g^k(\psi_{ux}))} \end{aligned} \tag{4.5}$$

Recall that in the k_{th} layer, the messages passing through edges are F^k dimensional, it is intuitive that a scalar form of τ_{uv}^k cannot exert all effects of persistence information vector ψ_{uv}. In other words, it is not necessary that reweighting parameters for different dimensions of features keep uniform. Instead a reweighting vector in F^k dimensions is cast to function $f^k : \mathbb{R}^l \to \mathbb{R}^{F^k}$.

$$f^k(\psi_{uv}) = \mathbf{S}^k(\mathbf{g}^k(\psi_{uv})) \tag{4.6}$$

If we view each dimension of ψ_{uv} as a channel, then $\mathbf{S}^k$ is channel-wise softmax function and $\mathbf{g}^k$ is an MLP outputting a F^k-dim vector. Notice that τ_{uv}^k is a F^k-dimensional vector rather than a scalar. Look back the messages aggregation and features update function (4.3), embed the persistence reweighting function into it, the convolution layer of PEGN is

$$\begin{aligned}\vec{h}_u^k &= \sigma_{k-1}(\sum_{v\in\bar{\mathscr{N}}(u)} diag(\tau_{uv}^{k-1})W^{k-1}\vec{h}_v^{k-1}) \\ &= \sigma_{k-1}(\sum_{v\in\bar{\mathscr{N}}(u)} diag(\mathbf{S}^{k-1}(\mathbf{g}^{k-1}(\psi_{uv})))W^{k-1}\vec{h}_v^{k-1})\end{aligned} \tag{4.7}$$

where $diag(\tau_{uv}^{k-1})$ is the diagonal matrix whose main diagonal entries are elements of τ_{uv}^{k-1}.

From layers above, we see the reweighting parameters τ_{uv}^k can even be a higher dimensional matrix, say $\mathbb{R}^{F^k}\times\mathbb{R}^{F^k}$ dimensions. However, the increasing of learnable parameters without significant mathematical meaning is not necessarily positive. In our experiments, the F^k-dimension vector formed τ_{uv}^k works well sufficiently.

4.4 Experiments

We have compared our Persistence Enhanced Graph Network against a couple of popular and strong baselines across two categories of graph benckmark datasets. In this section, we introduce these datasets and baselines. We summarize our experimental setup and discuss results. We show that our model has achieved or matched the state-of-the-arts in these node classification benckmarks.

Datasets.

We evaluate our graph network on three standard and widely used citation network benchmarks: Cora, Citeseer and Pubmed [SNB$^+$08]. We also use graph benchmarks such as Coauthor-CS, Coauthor-Physics, Amazon-Computers and Amazon-Photo [SMBG18]. The detailed introduction of these datasets are listed Appendix B. Pubmed, Coauthor and Amazon are larger and denser. Their detailed statistics are shown in Table 4.1. Some of them consist of over

10,000 nodes and 200,000 edges with higher average node degrees. They are more challenging in node classification tasks.

Table 4.1. Statistics of experimental benchmark datasets

	#Classes	#Features	#Nodes	#Edges	Edge density	Label rate
Cora	7	1433	2708	5429	0.0014	0.036
Citeseer	6	3703	3327	4732	0.0008	0.052
Pubmed	3	500	19717	44338	0.0002	0.003
Coauthor-CS	15	6805	18333	81894	0.0005	0.016
Coauthor-Physics	5	8415	34493	247962	0.0005	0.003
Amazon-Computers	10	767	13381	245779	0.0027	0.015
Amazon-Photo	8	745	7487	119043	0.0042	0.021

Experimental setup.

We compare our Persistence Enhanced Graph Network with a list of baselines: GCN [KW17] reweighting messages according to node degrees, GraphSAGE [HYL17] with mean aggregation of messages received within a sampled neighbourhood which works well in large graphs, MoNet [MBM$^+$17], Graph U-Net [GJ19], GAT [VCC$^+$18] adopting self-attention methods to reweight node features, and WLCN [MRF$^+$19] also taking subgraph structures information. In addition, we provide the node classification performance for all datasets of MLP which does not incorporate any graph structure information.

In the practical experiments, as the graph is unweighted, we apply Ollivier's Ricci curvature [NQWC17] as the weight function in graphs, and construct subgraphs by picking $1-hop$ and $2-hop$ neighbourhoods around each node. Denote them as **PEGN-RC-1** and **PEGN-RC-2** respectively. All persistence images in the experiments are 25-dimension. A two-layer graph network model is evaluated. The first layer makes a linear transformation over the input node representations, and then reweights the output feature vectors by pairing with a vector computed from three layer MLP. The non-linear function σ_0 is an exponential linear unit function. The second layer has the same structure with the first layer except the dimension of output feature vector is the number of classes, namely the second layer is for classification.

Table 4.2. Classification Accuracies on Benchmark Datasets

Method	Cora	Citeseer	PubMed	Coauthor CS	Coauthor Physics	Amazon Computer	Amazon Photo
MLP	58.2	59.1	70.0±2.1	88.3±0.7	88.9±1.1	44.9±5.8	69.6±3.8
MoNet	81.7	71.2	78.6±2.3	90.8±0.6	92.5±0.9	83.5±2.2	91.2±1.3
GraphSAGE	79.2	71.2	77.4±2.2	91.3±2.8	93.0±0.8	82.4±1.8	91.4±1.3
U-Net	82.5	72.0	78.9	92.7	94.0	86.0	91.9
WLCN	78.9	67.4	78.1	89.1	90.7	67.6	82.1
GCN	81.5	70.9	79.0±0.3	91.1±0.5	92.8±1.0	82.6±2.4	91.2±1.2
GAT	**83.0**	**72.5**	79.0±0.3	90.5±0.6	92.5±0.9	78.0±19.0	85.1±20.3
PEGN-RC-2	82.7	71.9	**79.4**±0.7	**92.9**±0.3	94.1±0.3	84.2±1	91.7±0.5
PEGN-RC-1	82.6	71.7	78.8±0.5	92.7±0.3	**94.2**±0.2	**86.3**±0.6	**92.5**±0.4

The train-validation-test split policy is exactly the same as that of GCN and GAT in [KW17, VCC+18]. Train graph networks with 20 nodes from each class, validate the algorithms on 500 nodes and test them on 1000 nodes. Models are initialized by Glorot initialization and the cross-entropy losses are minimized by Adam SGD optimizier with learning rate $r = 5e-3$. Additionally, we use L_2 regularization with $\lambda = 5e-4$ and early stopping based on the validation accuracy within 200 epochs.

The average classification accuracy and standard deviation are reported in Table 4.2 coming from 50 runs. We also collect the performances of baselines from [MBM+17, SMBG18, VCC+18].

Persistence Enhanced Graph Network is comparable with the state-of-the-art in the two small datasets, Cora and Citeseer, and outperforms baselines in other datasets, the larger and denser ones. Notice that although the performances of PEGN-RC-1 are slightly worse than PEGN-RC-2 on several datasets, it achieves higher average accuracy on dense Amazon graphs. Compared to GraphSAGE and U-Net aggregating information from multi-hop neighborhood and WLCN also taking subgraph structures information, it is fair to claim advanced topological structure information captured by persistence images indeed improve the performance of GNN in general, as the architecture of PEGN without reweighting mechanism is similar to that in GCN.

We also provide the experiment results taking Jaccard index as the weight function in Appendix Section B.1.2, which is similar to PEGN-RC. What's more, although multi-hop neighbourhood catches topological features from a wider range, it is not necessarily as distinguished as 1-hop neighbourhood in dense graphs. In particular, almost all of the 2-hop neighbourhood subgraphs in Amazon-Computers have more than 3000 nodes, which are considerably large parts of the entire graph. In other words, the topological features in 2-hop neighbourhood are not "local" sufficiently. Hence, multi-hop messages converge fast and are averaged out over the whole graph and produce similar node representations.

4.5 Application on material discovery

4.5.1 Introduction to carbon nanotubes data

Carbon fiber is among the most promising engineering materials for the 21st century due to superior tensile strength and modulus relative to low weight. It is extensively used in the automotive, aviation, and aerospace industry. Carbon fiber, for example, in the form of high-strength yarn, is composed of carbon nanotubes (CNTs), graphitic layers, and sometimes polymer binders [MRM$^+$12, KWPB12]. One of the major challenges in the research about CNTs data is finding the relationship between the current mechanical properties, like tensile modulus and tensile strength, and their structure features.

Tensile modulus and tensile strength of a CNT bundle can be obtained from precise reactive molecular dynamics simulations as shown in Figure 4.4. The simulations used the reactive Interface Force Field (IFF-R), which quantitatively reproduces pi-pi stacking, surface and interfacial energies, Young's moduli and tensile strength of CNTs, graphene, and graphite in agreement with experiments, and can also be utilized for solvent and polymer interfaces [PJG$^+$19]. These tensile properties are important in applications because they show how the shape of a CNT bundle changes under certain thermodynamics conditions. The structure features have potential influence on tensile properties including atoms density, structure defects such as

pristine, missing atoms, discontinuities or fracture defects, and complex configurations such as deformed and rearranged CNT lattices.

In this section, we apply our graph neural network framework, HS-GNN, augmented by local topological information for the task predicting CNTs' tensile modulus and strength with their structures.

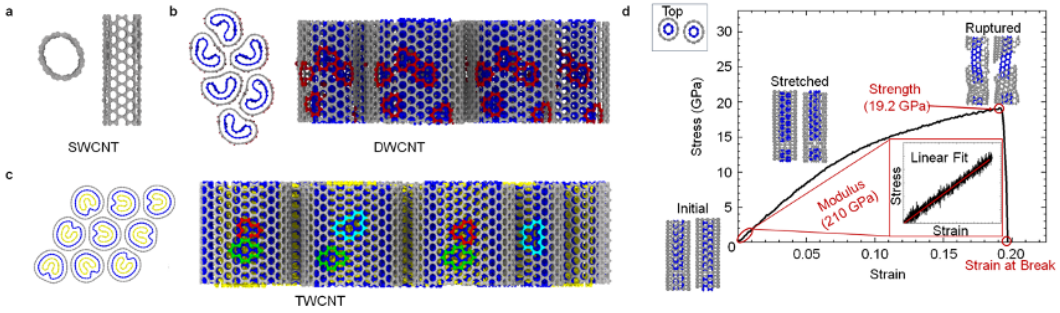


Figure 4.4. (a), (b), (c) are CNT structures with 1, 2, 3 walls. (d) shows how we obtain tensile modulus and strength from the shape of a CNT changes with increasing stress.

4.5.2 Neural network pipeline

The high level framework of our machine learning pipeline, called HS-GNN (Hierarchical Spatial Graph Neural Networks), for mechanical property predictions is shown in Figure 4.5. The input to the HS-GNN is the initial atomic structure $\mathtt{S}$ of a CNT configuration. Novel features of our HS-GNN architecture include: a heterogeneous graph representation of input, a hierarchical neural network model to capture large-scale interaction among different parts of the CNT bundle, and the injection of local geometric and topological features to better encode the shape of CNT bundle into the neural network via attention mechanism. These key components are briefly described in the following.

Heterogeneous Graph.

It is common to use a *bond-graph* $G_{bond\mathtt{S}}$ to represent $\mathtt{S}$, where each node corresponds to an atom, and there is an edge between two nodes if the corresponding atoms form a covalent bond. We go beyond the bond-graph and generate a heterogeneous graph $G_{\mathtt{S}}$ with multiple types

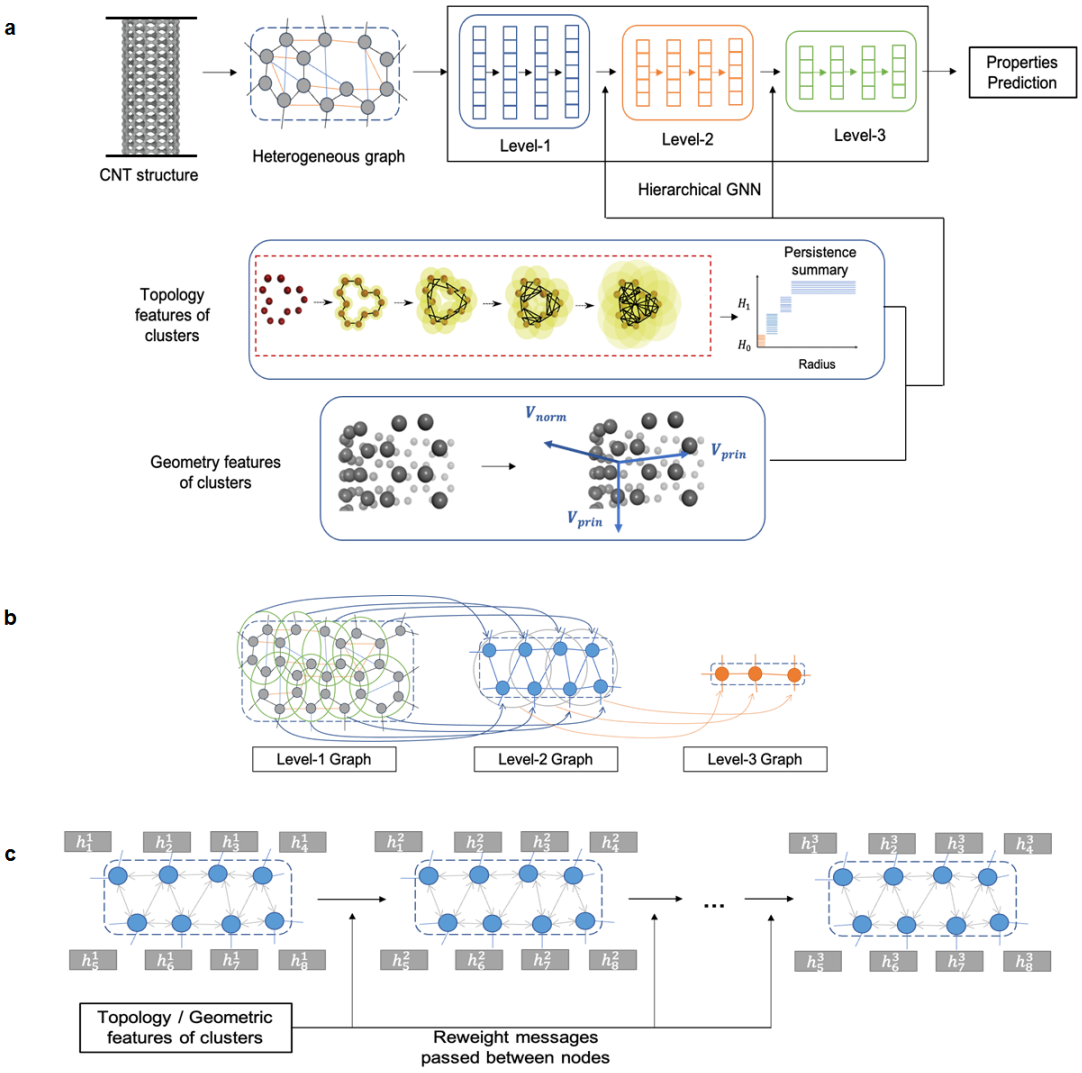


Figure 4.5. HS-GNN applies message passing, where persistence summaries and geometric features are used. (a) The pipeline to predict mechanical properties of CNTs. (b) Construction of the hierarchical graph series. (c) Illustration of the message-passing workflow of GNN in Level-2 and Level-3.

of edges among nodes from the initial atomic structure S. In particular, we add edges between nodes if corresponding atoms are spatially close (i.e, atoms from different nanotubes within 6Å distance) or if the so-called effective resistance distance between these two nodes is small. Effective resistance distance between two nodes is a geometric property measuring connectivity between these two nodes. We also connect nodes whose corresponding atoms form a dihedral angle, the angle of two planes formed by four sequentially bonded atoms rotated about a central bond. This heterogeneous graph is then fed to a hierarchical GNN.

Hierarchical GNN Model.

It is known that GNNs tend to have an oversmoothing issue [LHW18] and often cannot go very deep, thus limiting the aggregation of information from long range interactions. To address this issue, we use a hierarchical GNN that process the input graph $G_{\mathtt{S}}$ at multiple resolutions (three in our algorithm, Figure 4.5 a, b). Different from earlier hierarchical GNN approaches [YYM$^+$18, HLL$^+$19], HS-GNN finds hierarchies by spatial geometry information instead of graph topology or nodes features. The lowest level L_1 operates on input graph $G_{\mathtt{S}}$; while in a higher level L_i, each node corresponds to a cluster of nodes of level L_{i-1} and we can call such a node a *super-node*. The feature vector associated to each super-node is obtained by a pooling layer on level L_{i-1} graphs. Within each resolution level L_i, we use several GNN aggregation layers, which we refer to as *micro-layers* (Figure 3 c). In the end, a last pooling layer is applied so as to obtain graph-level prediction. See Figure 4.5 and Appendix Section B.2 for more details.

Encoding Local Geometry and Topology.

The shape of nanotubes and the spatial relation among neighboring tubes/sheets impact the final property. To encode local geometry, we use Principle Component Analysis (PCA) to capture a "curvature"-like quantity for each resolution level $L_i (i > 1)$. To encode local interactions among spatially close atoms, from potentially different nanotubes, we use the persistent homology to characterize the spatial distribution of points within each cluster in each level.

Node Features and Edge Attention.

The above information is incorporated into our HS-GNN via *node features* and via the *edge attention mechanisms.* Given any graph node v_a corresponding to atom a, our initial node feature vector $\mu^{(0)}(v_a)$ in level L_1 includes the degree of v_a in the bond graph, measuring local defects, the 3D coordinates of atom a, as well as random generated features. For the higher level GNNs, the persistence summary of a cluster is included in the features of the corresponding super-node to encode the shape of the configuration formed by all atoms in each super-node. Node features are transformed and aggregated through the neural networks. The edge-attention mechanism intuitively allows one to compute the "differential" of information at two endpoints v_a, v_b of an edge (v_a, v_b) (v_a and v_b will be super-nodes/clusters in $L_j(j > 1)$ level HS-GNN). This information is used to weight this connection (edge) (v_a, v_b) when aggregating information for node v_a from its neighbors. We use the two major principal vectors and the norm vector in local Principal Component Analysis (PCA) of points in each cluster to incorporate the edge-attention in level L_j of the HS-GNN.

4.5.3 Experiments

Dataset

Using the data set of 1159 tensile simulations of CNT bundles and graphitic structures, we trained and test the HS-GNN with split ratio 9:1. The distribution of tensile strengths within the training set is between 0-120 GPa, and the distribution of tensile moduli is between 0-1000 GPa. Few CNT bundles had strength and modulus values at the upper and lower limits, which correspond to pristine or completely fractured structures respectively.

An interesting aspect is also computing time. Using a server with 24 CPU cores and 1 RTX A6000 GPU, the HS-GNN requires about 420 minutes to compute the persistence summaries and PCA information of CNT bundles in the entire training set to train the model. The prediction (inference) process to obtain the predicted tensile strength and modulus for one bundled CNT nanostructure takes only about 2 minutes. The IFF-R simulation of the full

stress-strain curve for the same bundled CNT nanostructure takes about 30 minutes using 24 CPU cores. The improvement in speed is at least a factor of 10, and could be as high as 100 or more if the MD simulations were performed at lower strain rates.

Results

The ML pipeline and predictions are a first attempt at efficiently predicting mechanical properties of carbon nanostructures. We compared the performance of the HS-GNN to 6 other baseline machine learning models and linear regression (LR) based on global features of CNT bundles, PointNet [QSMG17] and RS-CNN [LFXP19] on point cloud formation CNT bundles, and three GNN models, GCN [KW17], MPNN [GSR$^+$17] and DimeNet [KGG20]. Compared to graph based pipelines, the two point cloud based approaches may loss important information like chemical bonds between nodes. Among these baselines, DimeNet has been considered to be state-of-the-art in property predictions of molecular structures.

Our ML method using the HS-GNN was tested in 3 ways: HS-GNN-A excludes any tensile property predictions as input features, HS-GNN-B uses the predicted strength to predict modulus, and HS-GNN-C uses the predicted strength and a pretrained model to predict modulus. See Table 4.3, HS-GNN significantly outperforms all 6 baselines in predicting tensile strength and modulus of CNT bundles. The HS-GNN is 2 to 5 times more accurate including larger structures not included in the training set. Uncertainties under approximately 10% can be considered of quantitative value relative to experimental measurements, which often have similar errors. In particular, the Mean Squared Errors (MSE) of our new HS-GNNs are only around half of those for the best prior method DimeNet, i.e, the errors of DimeNet almost double the errors of our proposed methods. What's more, we also test HS-GNN and these baselines on 16 large instances containing more than 20000 atoms. Our approach also achieves better generalization performance.

Table 4.3. MSE in prediction of strength and modulus of CNT by different machine learning pipelines. Prediction errors for larger structures (L) not included in the data set are also given.

Prediction error	LR	PointNet	RS-CNN	GCN	MPNN	DimeNet
Strength (%)	21.3	15.2	11.6	14.5	12.0	8.6
Modulus (%)	37.4	30.0	20.6	31.2	19.7	16.4
Strength (L) (%)	41.5	32.3	24.5	35.0	27.2	20.9
Modulus (L) (%)	42.1	37.4	30.5	40.9	34.8	23.5
	HS-GNN-A	HS-GNN-B	HS-GNN-C			
Strength (%)	4.2	4.2	4.2			
Modulus (%)	8.8	8.4	7.8			
Strength (L) (%)	12.7	12.7	12.7			
Modulus (L) (%)	17.6	15.8	15.3			

Further Sensitivity Analysis by Ablation.

Besides the GNN-based baselines (e.g, dimeNet, GCN, and MPNN) and our new HS-GNN models, we also evaluate two other GNN setups containing only partial components from our new HS-GNN models to evaluate the sensitivity. In particular, in the first setup, we remove the topological and geometric information from our HS-GNN-A model: This is to study the effect of adding hierarchies in our architectures. In this case, the prediction error for test datasets for strength and modulus increases to 7.4% and 14.6%, respectively. In the second setup, we use a non-hierarchical GNN setup, but with topological and geometric information used as node features and edge attention. The goal here is to study the effect of adding hierarchical levels in the neural network model. The prediction error for test datasets for strength and modulus then increase to 10.7% and 19.5%, respectively. Compared to performances of GCN (with prediction error 14.5% and 31.2%, respectively), we note that both hierarchical design (to capture long range interactions among atoms) and local topological and geometric information can reduce the prediction error significantly. Compared to the performance of our HS-GNN model with prediction errors 4.2% and 8.8%, respectively (HS-GNN-A), we note that the two strategies are somewhat complementary and using both reduced the prediction error to approximately half.

4.6 Conclusion

In this chapter, we present Persistence Enhanced Graph Network, a novel architecture leveraging topological structure information with persistence images, a stable vectorized representation of persistence diagrams. Experiments show the scheme that passing messages are reweighted in accordance with topological features achieves or matches state-of-the-art across node classification benchmarks. One potential improvement comes from more elaborately designed subgraphs around nodes or edges and filtration functions. Moreover, extending Persistence Enhanced Graph Network to graph classification tasks would be an interesting research direction. In the end, we also apply our idea of augmenting GNNs by local topological information on material discovery task. Experiments show that with the help of persistence summaries over local sub-structures, GNNs can predict carbon nanotubes' tensile properties with higher accuracy based on their micro-structures.

This Chapter 4, in part, is a reprint of the material as it appears in Persistence Enhanced Graph Neural Network, 2020. Zhao, Qi; Ye, Ze; Chen, Chao; Wang, Yusu. International Conference on Artificial Intelligence and Statistics (AISTATS), 2020. The dissertation author was the primary investigator and author of this paper.

This Chapter 4, in part, is a reprint of the material as it appears in Prediction of Carbon Nanostructure Mechanical Properties and Role of Defects Using Machine Learning, 2021. Winetrout, Jordan; Zhao, Qi; Xu, Yanxun; Heinz, Hendrik; Wang, Yusu, arXiv, 2021. The dissertation author was the primary investigator and author of this paper.

Chapter 5

NN-Baker: An Algorithmic NN Framework for Optimization Problems on Geometric Intersection Graphs

5.1 Introduction

Many tasks in science and engineering can be naturally modeled by combinatorial optimization problems over graphs, such as maximum independent set, minimum vertex cover, minimum multi-way cut, maximum clique, and so on. These problems are often NP-hard. Hence there has been great effort devoted to developing efficient approximation algorithms. However, many such problems are hard to approximate in the general setting as well: for example, it is known that the maximum independent set problem is NP-hard, and it is even NP-hard to approximate within $n^{1-\varepsilon}$ on n-vertex graphs, for any fixed $\varepsilon > 0$ [Hås99].

On the other hand, many of these hard problems admit PTAS in the *geometric setting* when the graphs are assumed to be induced by points in fixed-dimensional Euclidean space. Here, a *PTAS (polynomial-time approximation scheme)* is a polynomial time algorithm which for any fixed $\varepsilon > 0$, can approximate a maximization problem within a factor of $(1+\varepsilon)$ (or within a factor of $(1-\varepsilon)$ for a minimization problem). One most prominent example is the travelling salesman problem (TSP), where in general there is no PTAS available (and currently the best known approximation algorithm achieves a factor of $3/2-\alpha$, for some $\alpha > 10^{-36}$ [KKG20]). However

a PTAS, more specifically in this case a near linear time $(1+\varepsilon)$-approximation algorithm, was developed for the Euclidean TSP problem in a ground breaking work by Arora [Aro98]. The maximum independent set problem is known to remain NP-hard in the geometric setting [FPT81], but it is known that it also admits a PTAS [Cha03, EJS05, CHP12].

Nevertheless, for problems of large size, even these PTAS are still not practical, especially since many of them involve large constants that are **exponential** in some fixed parameter (e.g, having a term like $n^{\frac{1}{\varepsilon}}$ where $(1+\varepsilon)$ is the approximation factor). In applications, practitioners often rely on handcrafted heuristic search strategies to find high-quality solutions.

Recently, there has been a surge on using machine learning (ML) for combinatorial optimization problems; see the review paper by Bengio *et al.* [BLP20] which also provides a nice categorization of different ways that a ML component may contribute to the combinatorial optimization problems. Earlier such approaches focus on several graph combinatorial optimization problems, including TSP, minimum vertex cover and maximum independent sets; see e.g., [VFJ15, BPL$^+$17, DKZ$^+$17, LCK18, SYK19, BCFL20, KL20, ASS20]. More recently, there has been a range of approaches developed to tackle more general mixed integer linear programming problems (MILP); e.g, [KLBS$^+$16, GCF$^+$19, FWDT20, GGK$^+$20, NBG$^+$20]. See Section 5.1 for more detailed description of some related work.

In general, an ML framework can be trained to solve an optimization problem in *an end-to-end manner* (e.g, [BPL$^+$17, DKZ$^+$17, LCK18, KL20, ASS20]). Alternatively, many recent approaches use ML module as a component *within a specific algorithmic framework* to help make hopefully better (either in terms of quality or efficiency) decisions during the execution of this algorithm: such as using imitation learning or reinforcement learning to learn a good policy to decide which variable to branch in a branch-and-bound algorithm [GCF$^+$19, GGK$^+$20]. Despite the tremendous progress in combining ML with optimization problems, theoretical understanding remains limited: Does a proposed ML pipeline have the expressiveness capacity to solve the combinatorial optimization problem exactly or approximately? What is a suitable model for input data distribution to talk about generalization?

Consider the capacity question: While neural networks are known to have many universal approximation results for real or complex valued continuous functions (or some other special families of functions), e.g, [Cyb89, Pin99, SGT$^+$08, PMB18], combinatorial optimization problems, say finding the maximum independent set, cannot be easily modeled to fit into such function approximation frameworks. Nevertheless, a very interesting recent work [SYK19] shows that for graph combinatorial optimization problems, the so-called vector-vector consistent graph neural networks (VV$_C$-GNNs), can solve the same family of problems as the distributed local algorithms in the so-called port-numbering models. Leveraging the literature in distributed local algorithms [Ang80, ÅFP$^+$09], this leads to several positive results on the capacity of GNNs for approximating minimum vertex cover or maximum matching problems with certain constant factors, however **only for graphs with bounded degrees** – intuitively, the depth of the GNN will depend on the bound Δ on the maximum node degree of input graphs. Unfortunately, the connection to distributed local algorithms also leads to negative results: roughly speaking, these constant factor approximations for the special family of **constant-degree graphs** are the best that a GNN can do for these graph optimization problems. See [SYK19] for details.

Main contributions of Chapter 5.

The aforementioned results (on the capacity of GNNs) are for the case where GNNs are used to solve an optimization problem *in an end-to-end manner*, and other than the bound on max-degree, the input graphs are abstract graphs. In this chapter, we advocate the study of using ML for optimization problems in the *geometric setting* where (graph) optimization problems are induced by points in a fixed dimensional Euclidean space $\mathbb{R}^d$. This is a rather common scenario in practice, such as solving TSP in road networks spanned by cities in $\mathbb{R}^2$, or solving maximum independent set in a communication network spanned by sensor nodes in $\mathbb{R}^2/\mathbb{R}^3$ (i.e, the unit-ball model that we will introduce in Section 5.2.1). Such graphs can also be the result of an embedding of an input arbitrary graph into a certain latent space. At the same time, the geometric setting brings special structures to the problem at hand, which an algorithm and also

ML can then leverage.

In particular, using the maximum independent set (MIS) problem as an example, we first propose what we call *the Baker-paradigm* in Section 5.2, which is an approximation framework for geometric optimization problems inspired by Baker's work in [Bak94]. We show that the Baker-paradigm gives a bi-criteria approximation for the MIS in the Euclidean setting (Theorem 5.2.1). The running time is only *linear* in the size of input point set, but *exponential* in terms of the parameters. This framework is general and can be extended to several other geometric optimization problems (Section 5.2.3).

A key advantage of our Baker-paradigm is that it decomposes the problem into (at most a linear number of) small sub-problems of fixed size (independent of size of input graphs). For the family of such fixed-size sub-problems, we can now design neural networks with universal approximation guarantees to solve them. Using such a neural network to replace (Step-2) of our Baker-paradigm, we then obtain a mixed algorithmic-ML framework, which we refer to as NN-Baker, that has the capacity to produce a bi-criteria approximation of MIS within *any* constant factor in near linear time (i.e, a bi-criteria PTAS); see Section 5.3.1 and Theorem 5.3.2. Note that while Theorem 5.2.1 already gives a near-linear time bi-criteria PTAS for MIS, the constant involved in the time complexity is exponential in the approximation parameters, making it inefficient in practice. In contrast, the NN-Baker will replace the costly component by a neural network component, and only call this neural network at most n times. Other than calls to neural networks, the time needed is $\Theta(n)$ where the constant contains only terms polynomial in the approximation paramters. The resulting mixed algorithmic-ML framework is very efficient, as we show in Section 5.4.

In Section 5.3.2, we provide an instantiation of the NN component for our NN-Baker based on GNN. We present a range of experimental results in Section 5.4 to show the effectiveness the GNN-Baker frameworks. Note that our NN-Baker can be used together with other SOA NN framework to solve combinatorial optimization problems and to further improve them (sometimes significantly). For example, we deploy different SOA GNNs for graph optimization

problems to instantiate (Step-2) in NN-Baker, including TGS of [LCK18] in the supervised setting, LwD of [ASS20] in the reinforcement learning setting, as well as Erdős-GNN [KL20] in the unsupervised setting. We show that as the problem size increases, the performance of each original GNN decreases (the GNN is always trained and tested on problems of similar sizes). However, using GNN-Baker significantly improves the performance of the original GNN as the problem size increases – This is partly because, independent of the problem size, in (Step 2) our GNN-Baker only needs to train and test the GNN component on a small graph (of bounded size). Thus the trained GNN can adapt to the problem structure much better and require much fewer training samples.

Our NN-Baker is, to our best knowledge, the first (bi-criteria) PTAS for a combinatorial problem for an ML-based approach (in terms of expressiveness). The recent line of work of using a ML component (e.g a GNN trained by imitation learning) to make branching decisions within the branch-and-bound algorithmic framework [GCF+19, GGK+20] may solve the exact problem *given enough running time*. However, the number of times the algorithm calls the NN component may be exponential in the input size. Instead, our NN-Baker framework calls the neural network (which has a bounded size) only a linear number of times. Our approach can open new directions to design NN-infused algorithmic frameworks with theoretical guarantees, by for example, leveraging divide and conquer paradigm and replacing certain algorithmic components by neural components.

More on related work.

The idea of using neural networks to tackle optimization problems traces back to the 1980's. One of the most important frameworks in this direction is the Hopfield Neural Network (HNN) [Hop82, HT85]. In particular, the HNN is a feedback neural network with pre-specified weights (whose assignments depend on the optimization problem at hand), which encodes a dynamic system whose associated energy function characterize the optimization problem to be solved. To use it to solve an optimization problem, one starts with an initial state, and iterates till

convergence.

As branch-and-bound (B&B) has been proven to be a powerful framework in solving optimization problems, especially in MILP (mixed integer-linear programing) problems, researchers proposed different machine learning algorithms to boost B&B. [KLBS$^+$16] provided a list of features of variables and designed a learning-to-rank model on selecting branching variables. [GCF$^+$19, GGK$^+$20, NBG$^+$20] developed GNN approaches to learn the policy of choosing branching variables after formulating MILP problems as graphs by imitation learning or reinforcement learning. Besides MILP problems, GNNs are also applied on graph combinatorial optimization problems. [VFJ15] encodes the input graphs as sequences and takes an attention RNN to process the sequences. It can be used to compute convex hulls or solve problems like TSP. [BPL$^+$17, DKZ$^+$17, DCL$^+$18, PAL$^+$19, ASS20] takes reinforcement learning on graphs to solve routing problems like TSP, and other problems like maximum independent set. [LCK18] solves graph theory problems by supervised learning setup after solving a set of cases as training set by existing solvers, while [KL20] introduces unsupervised approaches by designing loss function and training setup based on objective functions and variables constraints. In addition, there are works [SYK19, XLZ$^+$20, Lou20] that study the power of GNN on solving different kinds of combinatorial optimization problems.

Outline of Chapter 5.

This chapter is organized as follows. We first introduce the Baker-paradigm solving problems like MIS on intersection graphs in Section 5.2. We then present the algorithmic-NN framework, NN-Baker, and its instantiation in Section 5.3. In Section 5.4, we show experiments of our NN-Baker and other GNN baselines in solving MIS problems for different dimensional data. Finally, we make conclusions about our NN-Baker in Section 5.5.

5.2 The Baker-paradigm

We now propose a framework to obtain approximation algorithms for certain geometric optimization problems based on the work of Baker [Bak94]. Baker's technique has been applied successfully to a plethora of optimization problems on planar graphs, including Maximum Independent Set, Minimum Vertex Cover, Minimum (Edge) Dominating Set, Maximum Triangle Matching, Maximum H-Matching, TSP, and many others. Furthermore, the technique has been adapted to the geometric setting (see [Aro03] for a survey). From the geometric setting, the most relevant work to ours is [HM85], where the authors obtain approximation algorithms for maximum independent set and minimum vertex cover on unit disk graphs. We will show in the next section how neural networks can be used to obtain efficient algorithms within this framework. We begin with some definitions.

5.2.1 Preliminaries

We consider the geometric setting: i.e., combinatorial optimization problems on geometric intersection graphs. We present our methods for the case when the input is a **unit-ball graph**, which is the intersection graph of unit balls in $\mathbb{R}^d$. Specifically, graph nodes correspond to a set of balls of unit radius in $\mathbb{R}^d$, and two nodes are connected by an edge iff the two balls intersect. Our approach can be extended to intersection graphs of several other geometric objects such as unit hypercubes, ellipsoids of bounded aspect ratio, and so on. For the sake of succinctness, we focus on the case of unit balls.

Approximations and bi-criteria relaxations.

The algorithms we will develop are bi-criteria approximations. The precise definition of a bi-criteria optimization problem depends on the space of feasible solutions. For concreteness, let us focus on the *d-dimensional Maximum Independent Set of Unit Balls problem* (denoted by ***d*-MIS**), for some fixed dimension $d \in \mathbb{N}$: The input to the d-MIS problem is a set of points $X \subset \mathbb{R}^d$, which corresponds to the set of centers of unit balls. Let $G_X = (X, E)$ denote the intersection

graph spanned by points in X, where $(x,x') \in E$ if the unit balls $\mathsf{ball}(x,1)$ and $\mathsf{ball}(x',1)$ intersect (meaning that $\|x-x'\| \leq 2$). The goal is to find the maximum independent set of G_X, which is equivalent to finding some maximum cardinality subset of disjoint balls centered at X. Let $\mathsf{OPT}(X)$ denote the size of a maximum independent set for G_X. For any $\alpha > 0$, an algorithm is an *α-approximation for d-*MIS if on any input $X \subset \mathbb{R}^d$, it outputs some independent set $Y \subseteq X$, with $\mathsf{OPT}(X)/\alpha \leq |Y| \leq \mathsf{OPT}(X)$.

The bi-criteria version of the problem is defined as follows. Let $\varepsilon > 0$. We say that some $Y \subseteq X$ is $(1+\varepsilon)$-*independent* if the balls of radius $1/(1+\varepsilon)$ centered at the points in Y are disjoint; that is, for all $p,q \in Y$, we have $\|p-q\|_2 > 2/(1+\varepsilon)$. We denote the size of the maximum $(1+\varepsilon)$-independent subset of X by $\mathsf{OPT}_{1+\varepsilon}(X)$. For any $\alpha \geq 1$, $\beta \geq 1$, We say that an algorithm is (α,β)-*bi-criteria approximation* if on any input $X \subset \mathbb{R}^d$, outputs some β-independent set $Y \subseteq X$, with $\mathsf{OPT}(X)/\alpha \leq |Y| \leq \mathsf{OPT}_\beta(X)$.

Randomization.

The algorithms we present for d-MIS are randomized, and thus the size of the output is a random variable. We use the following standard extensions of the above definitions in this setting. We say that a randomized algorithm is α-approximation in expectation if on any input X it outputs a solution Y with $\mathsf{OPT}(X)/\alpha \leq \mathbf{E}[|Y|] \leq \mathsf{OPT}(X)$. We say that a randomized algorithm is (α,β)-bi-criteria approximation in expectation if on any input X it outputs a solution Y with $\mathsf{OPT}(X)/\alpha \leq \mathbf{E}[|Y|] \leq \mathsf{OPT}_\beta(X)$.

5.2.2 Baker's paradigm for d-MIS

In this section, we describe our Baker-paradigm to obtain an approximation algorithm for d-MIS. We will later see that the same method can be extended to several other optimization probelms on geometric intersection graphs. Let $\varepsilon > 0$ be arbitrarily small but fixed. The algorithm proceeds in the following steps: See Figure 5.1 for an illustration.

Algorithm Baker-MIS: The input is a set of points $X \subset \mathbb{R}^d$, with $|X| = n$, where each

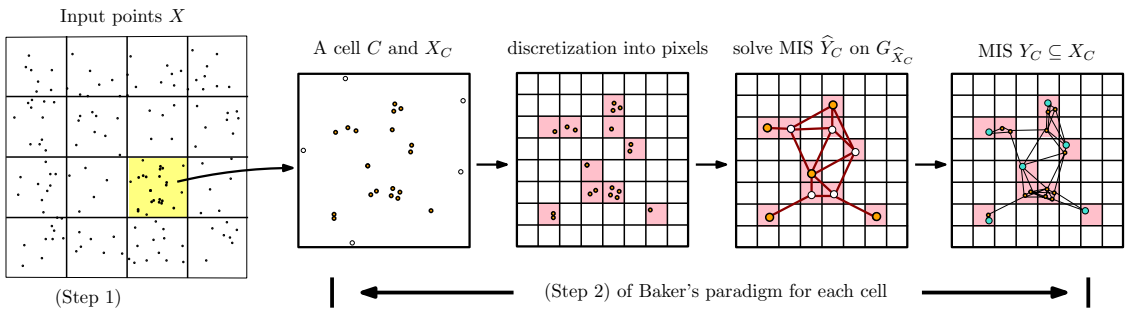


Figure 5.1. Illustration of Baker-paradigm in 2D. First we put a randomly shifted grid on input points X. Consider a cell C, we then solve sub-problems in it after a second level partition and got solution Y_{C}. In (Step 3), the union of all Y_{C} for all cells is returned as the final MIS.

point in X is the center of a unit ball. The algorithm has three steps.

Step 1: The randomly shifted grid.

Let Γ be an axis-parallel hyper-grid, where each cell is a d-D axis-parallel hypercube of side-length $\mathsf{r} = \frac{2d}{\varepsilon}$. Specifically, let $\Gamma = \bigcup_{i=1}^{d} \bigcup_{j\in\mathbb{Z}} \{h_i + j\frac{2d}{\varepsilon} e_i\}$, where $e_1, \ldots, e_d$ is the standard orthonormal basis, and for $i \in [1,d]$, h_i is the (d-1)-D hyperplane that passes through the origin and is orthogonal to e_i. Pick $\tau \in [0, 2d/\varepsilon)^d$ uniformly at random. Let $\Gamma + \tau$ denote the grid obtained by shifting Γ by the vector τ.

Step 2: Bi-criteria solution for the problem locally on each cell.

Now given a cell C of $\Gamma + \tau$, for any $P \subset \mathbb{R}^d$, let $P_{\mathsf{C}} = P \cap \mathsf{C}$ denote the restriction of P within C. Let X' be the set of centers of the unit balls in X that intersect the shifted grid $\Gamma + \tau$ (i.e., these are points within distance-1 to the hyperplanes (or gridlines for the 2D case) in the grid). Let $\delta > 0$, and set $\delta' = \delta/\sqrt{d}$. We partition **each cell** C of $\Gamma + \tau$ to a d-dimensional grid of pixels, where each pixel is a d-dimensional hypercube of side length δ'. We snap each point in $X \setminus X'$ to the corner of the pixel containing it that is closest to the origin, thus obtaining the set $\widehat{X}$, ignoring multiplicities; that is

$$\widehat{X} = \bigcup_{(p_1,\ldots,p_d)\in X\setminus X'} \{(\delta' \lfloor \frac{p_1}{\delta'} \rfloor, \ldots, \delta' \lfloor \frac{p_d}{\delta'} \rfloor)\}.$$

Now for each cell C of $\Gamma+\tau$, $\widehat{X}_{\mathsf{C}}$ (the restriction of $\widehat{X}$ to C) consists of the bottom-left corners of those pixels containing some points in $X_{\mathsf{C}} \setminus X'_{\mathsf{C}}$. Let $G_{\widehat{X}_{\mathsf{C}}}$ be the intersection graph of the radius-$(1-\delta)$ balls with centers in $\widehat{X}_{\mathsf{C}}$. Note that the size of $\widehat{X}_{\mathsf{C}}$ is at most $(\frac{2d}{\varepsilon\delta'})^d$, and thus the graph $G_{\widehat{X}_{\mathsf{C}}}$ is of bounded size. Furthermore, note that any independent set of $G_{\widehat{X}_{\mathsf{C}}}$ can be of size at most $s = V_d(\frac{2d}{\varepsilon})^d$ (due to a simple packing argument), where V_d denotes the volume of the d-dimensional ball of radius $(1-\delta)$. We can then compute the maximum independent set $\widehat{Y}_{\mathsf{C}} \subseteq \widehat{X}_{\mathsf{C}}$ in $G_{\widehat{X}_{\mathsf{C}}}$ by a brute-force enumeration of all subsets of $\widehat{X}_{\mathsf{C}}$ of cardinality at most s, and returning the maximum cardinality subset that is independent in $G_{\widehat{X}_{\mathsf{C}}}$. Finally, we compute $Y_{\mathsf{C}} \subseteq X_{\mathsf{C}}$ by mapping each point $\widehat{p} \in \widehat{Y}_{\mathsf{C}}$ to an arbitrary point $p \in X_{\mathsf{C}} \setminus X'_{\mathsf{C}}$ that lies in the pixel that $\widehat{p}$ represents.

Step 3: The final solution.

The final solution is $Y = \bigcup_{\mathsf{C}} Y_{\mathsf{C}}$, the union of MIS returned within all non-empty cells.

The proof of the following theorem can be found in Appendix Section C.1.1.

Theorem 5.2.1. *Let $\varepsilon, \delta > 0$. The algorithm Baker-*MIS *is a $(1+\Theta(\varepsilon), 1+\Theta(\delta))$-bi-criteria approximation in expectation for* MIS. *On input a set of size n, the algorithm runs in time $(1/(\varepsilon\delta))^{(d/\varepsilon)^{O(d)}} n$.*

Recall that our algorithm removes those points X' within distance 1 to the grid. Intuitively, we need randomly shifted grid so that this X' does not contain too many points from an optimal maximum independent set in expectation. Note that in practice, we can further improve the quality of the output: Specifically, currently, all points within distance 1 to the shifted grid $\Gamma+\tau$ (i.e, $X' \subseteq X$) are removed. The reason is to ensure that solutions (max-independent sets) of neighboring cells do not conflict each other. We can add some of those points from X' back to the solution Y as long as they do not cause conflict (i.e, within distance 2) to any points in Y. We can do so in a greedy manner in practice to obtain an even better solution. Our theorem above holds for this greedily improved solution as it remains an independent set and is a superset of Y.

Removing the Bi-Criteria Condition.

We remark that directly using the original Baker idea one can in fact obtain a $(1+\varepsilon)$-approximation for MIS (instead of a bi-criteria approximation), by not having a second-level discretization in each cell in (Step 2). This is effectively a randomized version of the algorithm from [HM85]. The standard details can be found in Appendix Section C.1.2. As shown in the following theorem, the price to pay is that the dependency of time complexity on n increases from previous n (i.e, linear) to $n^{(1/\varepsilon)^{O(d)}}$, which is significant.

Theorem 5.2.2. *We can modify Baker-paradigm to provide a* $(1+\Theta(\varepsilon))$*-approximation in expectation for* MIS. *On input a set of size n, this modified algorithm runs in time* $n^{(1/\varepsilon)^{O(d)}}$.

5.2.3 Other graph optimization problems

We now briefly discuss how the above general framework can be extended to other problems on unit ball graphs, with only minor modifications. We describe some representative such problems.

Minimum vertex cover. In the Minimum Vertex Cover (MVC) problem, we are given a graph G and the goal is to find a minimum cardinality set $U \subseteq V(G)$, such that all edges in G have at least one endpoint in U. In the case were G is a unit ball graph in $\mathbb{R}^d$, this problem can be solved by modifying the Baker paradigm as follows. In Step 2, we enlarge each cell C of $\Gamma+\tau$ by increasing its side length to $2+2d/\varepsilon$. Thus, any two adjacent cells have an intersection of width 2, and some points in X may fall in multiple cells. By the linearity of expectation, it follows that the expected number of points in any optimal solution that fall in multiple cells (counting multiplicities) is at most $\varepsilon|\mathsf{OPT}(X)|$. Therefore, by solving the problem independently on each cell and taking the union of all the solutions we obtain a $(1+\varepsilon)$-approximate solution for the initial problem X. Discretizing each cell into further pixels gives rise to a more efficient, but bi-criteria approximation.

Maximum Acyclic Subgraph, Planar Subgraph, and $\mathscr{F}$-Minor Free Subgraph. In the Maximum Acyclic Subgraph problem we are given a graph G and the goal is to compute a

subgraph of G with a maximum number of vertices that is acyclic. It follows by the linearity of expectation, that the expected number of balls in any optimal solution that intersect the randomly shifted grid is at most $\varepsilon|\mathsf{OPT}(X)|$. Thus, solving the problem on each cell and taking the union of all the acyclic subgraphs found, results in a $(1+\varepsilon)$-approximate optimal acyclic subgraph of the input. (Similar to MVC, discretizing each cell into further pixels gives rise to a more efficient, but bi-criteria approximation.) The exact same argument works also for the Maximum Planar Subgraph problem, where the goal is to find a subgraph with a maximum number of vertices that is planar. Finally, the same argument extends to the case of the more general Maximum $\mathscr{F}$-Minor Free Subgraph problem, where the goal is to find a subgraph with a maximum number of vertices that does not contain as a minor any of the graphs in a fixed family $\mathscr{F}$. We note that this problem generalizes the Maximum Acyclic Subgraph problem (when $\mathscr{F}$ contains the triangle graph) and the Maximum Planar Subgraph problem (when $\mathscr{F}$ contains K_5 and $K_{3,3}$).

5.3 A NN-Baker framework

5.3.1 Infusing neural network inside the Baker-paradigm

Instead of solving (Step 2) of Baker-paradigm in a brute-force manner, we can replace it by a neural network, and we refer to the resulting generic paradigm as *NN-Baker*. Roughly speaking, we will replace the exact computation of a MIS in Step 2 of algorithm Baker-MIS by a neural network. More specifically, consider the following:

Step 2'.

We follow the same notations as in Step 2 of algorithm Baker-MIS. For each cell C of the grid $\Gamma+\tau$, we proceed as follows. Recall $\widehat{X}_{\mathsf{C}}$ is a set of corners of all non-empty pixels (i.e, containing some point from $X \setminus X'$) in C. Let W_{C} be the set of all pixels in C, and let $\mathscr{P}_{\mathsf{C}}$ be the powerset of W_{C}. Let $f_{\text{MIS}} : \mathscr{P}_{\mathsf{C}} \to \mathscr{P}_{\mathsf{C}}$ be such that for all $Z \in \mathscr{P}_{\mathsf{C}}$, $f_{\text{MIS}}(Z)$ is some optimal solution to the MIS problem on input Z w.r.t. radius $(1-\delta)$; that is, f_{MIS} maps an instance to an optimal MIS solution for the intersection graph formed by radius $(1-\delta)$

balls. Since every point in $X_C \setminus X'_C$ is at distance at most δ from some point in $\widehat{X}_C$, it follows that $f_{\text{MIS}}(\widehat{X}_C)$ is a $(1, 1+\Theta(\delta))$-bi-criteria solution for the set of points in $X_C \setminus X'_C$ (see the argument in the proof of Theorem 5.2.1 in Appendix). We can view f_{MIS} as a mapping between indicator vectors of subsets of W_C; i.e. $f_{\text{MIS}} : \{0,1\}^k \to \{0,1\}^k$, where $k = |\mathscr{P}_C| = 2^{(2d/(\varepsilon\delta))^d}$. Let $\widehat{f}_{\text{MIS}} : [0,1]^k \to [0,1]^k$ be any continuous extension of f_{MIS}. We then approximate $\widehat{f}_{\text{MIS}}$ by a function $g_{\mathscr{N}} : [0,1]^k \to [0,1]^k$ as computed by a neural network $\mathscr{N}$. We round coordinate-wise the output of g_{MIS} to a vector in $\{0,1\}^k$, by setting every value greater than $1/2$ to 1, and all other values to 0. We thus obtain the indicator vector of some $Y_C \subset W_C$. Alternatively, we can produce a discrete solution by the following greedy strategy: We sort the points in W_C in non-increasing order of their values in the vector $g_{\mathscr{N}}(\widehat{X}_C)$, and we take $\widehat{Y}_C$ to be a maximal prefix of this sorted order that forms an $(1+\Theta(\delta))$-independent set in $G_{\widehat{X}_C}$. Finally, we compute $Y_C \subseteq X_C \setminus X'_C$ by mapping each point $\widehat{p} \in \widehat{Y}_C$ to any point $p \in X_C$ within the pixel represented by $\widehat{p}$.

Universal-Baker.

We now give a theoretical instantiation of NN-Baker using the neural network obtained by the following universal approximation result.

Theorem 5.3.1 (Cybenko [Cyb89])**.** *Let σ be any continuous sigmoidal function. Let $m \in \mathbb{N}$, and let $\mathscr{C}$ be a compact subset of $\mathbb{R}^m$. Let $f : \mathscr{C} \to \mathbb{R}$ be a continuous function, and $\gamma > 0$. Then, there exists $N \in \mathbb{N}$, $a_1, \ldots, a_N \in \mathbb{R}$, $y_1, \ldots, y_N \in \mathbb{R}^m$, and $\theta_1, \ldots, \theta_N \in \mathbb{R}$, such that the function $g : \mathscr{C} \to \mathbb{R}$, with $g(x) = \sum_{i=1}^{N} a_i \sigma(y_i^T x + \theta_i)$, satisfies $\sup_{x \in \mathscr{C}} |g(x) - f(x)| < \gamma$.*

Using the neural network given by the above result, we can then argue that our NN-Baker has the capacity (expressiveness) of solving MIS problem (for unit-ball graphs) in a bi-criteria approximation. The simple proof of this theorem can be found in Appendix Section C.1.3.

Theorem 5.3.2. *Let $\varepsilon, \delta > 0$. There exists $N = N(\varepsilon, \delta, d)$, such that the following holds. Suppose that the function $g_{\mathscr{N}^*}$ in Step 2' of the NN-Baker framework is computed by the neural network $\mathscr{N}^*$ given by Theorem 5.3.1, with a single hidden layer of size N. Then, the resulting algorithm*

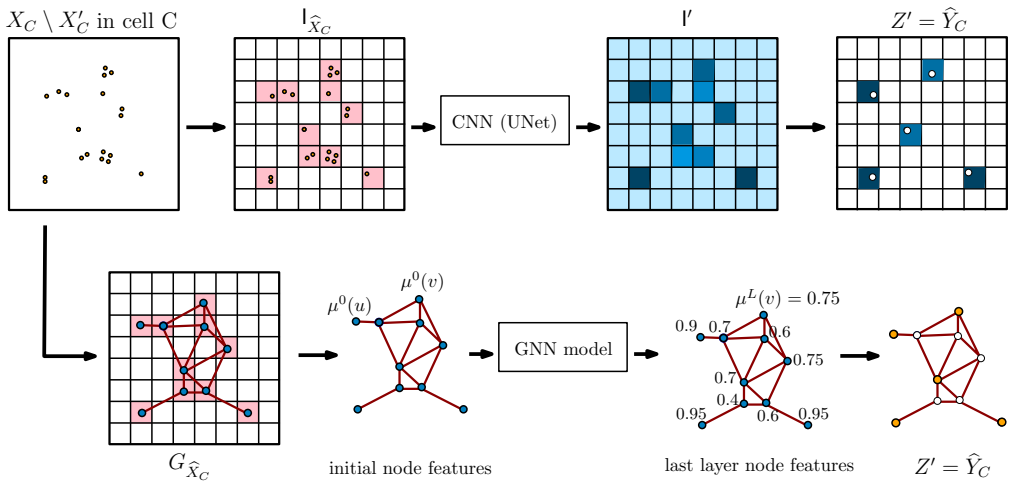


Figure 5.2. Given the set of points $X_{\mathsf{C}} \setminus X'_{\mathsf{C}}$ contained in cell C, the top row shows the processing of it by a CNN component as in CNN-Baker, while the bottom row shows it for a GNN component as in GNN-Baker. $\widehat{Y}_{\mathsf{C}}$ in the figure corresponds to the set of pixels containing points from Y_{C}.

is a $(1+\Theta(\varepsilon), 1+\Theta(\delta))$*-bi-criteria approximation in expectation for* d-MIS*, and it will call this (same) neural network at most n times where n is the number of points generating the input graph.*

Remark 5.3.1. *A similar statement to Theorem 5.3.2 holds for the Vertex Cover problem, yielding a* $(1+\Theta(\varepsilon), 1+\Theta(\delta))$*-bi-criteria approximation using the modifications discussed in Section 5.2.3.*

5.3.2 Instantiation of NN-Baker

Above we introduce a generic NN-Baker framework and a theoretical instantiation. We now provide a specific practical instantiations of this framework: GNN-Baker where (Step 2') of NN-Baker (or equivalently, (Step 2) of our Baker-paradigm) is implemented by a specific GNN. We provide details below. Specifically, recall that the input is a set of points $X \subset \mathbb{R}^d$, and for a randomly shifted grid $\Gamma + \tau$, let us focus on a specific grid cell $\mathsf{C} \in \Gamma + \tau$. Recall that in NN-Baker, given C and $\widehat{X}_{\mathsf{C}}$, we will use a neural network to compute a subset of $\widehat{Y}_{\mathsf{C}}$ which ideally approximates a maximum independent set of the (unit-ball) geometric intersection graph $G_{\widehat{X}_{\mathsf{C}}}$ spanned by $\widehat{X}_{\mathsf{C}}$. The set $\widehat{Y}_{\mathsf{C}}$ (of pixels) is further relaxed to a subset $Y_{\mathsf{C}} \subset X_{\mathsf{C}}$ as an independent set solution within cell C.

GNN-Baker.

In GNN-Baker, we instead directly use the unit-ball graph $G_{\widehat{X}_C}$ as input, with the initial node feature $\mu^0(v)$. At the last L-th layer, the node feature $\mu^L(v)$ gives the likelihood that v is in the maximum independent set. We then retrieve an independent set Y_C by a greedy approach introduced in Paragraph 5.3.2. For the specific choice of the GNN architecture, we can use any of the existing models, such as GCN [KW17], GraphSAGE [HYL17], GIN [XHLJ19], GAT [VCC$^+$18], and VV$_C$-GNN [SYK19]. In our later experiments, we will use TGS [LCK18] and LwD [ASS20] in our GNN-Baker framework, as these are state-of-the-art (SOA) approaches specifically designed for graph combinatorial problems. (Our experiments show that using a general purpose GNN has much worse performance than TGS and LwD.) In particular, TGS, a supervised learning approach, takes GCN to process reduced graphs and use a tree search approach to label nodes whether they are in an independent set or not. LwD is reinforcement-learning based, and designs a policy network and value network on each MIS problem state with GraphSAGE architecture. We will also use the Erdős-GNN [KL20], designed for optimization problems in the *unsupervised setting*. Erdős-GNN takes multiple GIN layers followed by a GAT layer to learn graphs' distribution. It designs a differentiable loss function based on expectations of optimization problems objective functions and a probabilistic penalty function.

Besides GNN-Baker, we can also implement NN-Baker in Step 2' by a CNN and develop a resulting CNN-Baker. One can find details of CNN-Baker in our paper [MZSW21]. However, GNN-Baker has a better flexibility to extend to high dimensional cases.

Retrieve method.

In (Step 2') of NN-Baker, we describe an alternative greedy approach to convert this likelihood map to a subset of points $Y_C \subset X_C \setminus X'_C$ as the output MIS. However, in our implementation for both CNN-Baker and GNN-Baker, we will make a slight modification: In particular, note that the output from (Step 2') will be a $(1+\Theta(\delta))$-independent subset of X_C. In practice, we would like to guarantee that we output a valid MIS for X_C (i.e, any two points in our output

Y_{C} should be at least distance 2 apart). We thus follow the greedy approach as outlined in (Step 2') but with a small modification: We sort all pixels in $Z = \widehat{X}_{\mathsf{C}}$ in decreasing order of their pixel values, and assume $Z = \{z_1, \ldots, z_\ell\}$ is this sorted list. We then inspect them one-by-one in order. At the beginning, initialize an output set Y' to be empty. Then in the i-th iteration, let y_i be any point from $X_{\mathsf{C}} \setminus X'_{\mathsf{C}}$ in pixel z_i. If y_i is independent to all points in Y' (i.e, it is more than 2 away for all points in Y'), add y_i to Y'; and otherwise, do nothing. In the end after ℓ iterations, we obtain a set of $Y' \subseteq X_{\mathsf{C}} \setminus X'_{\mathsf{C}}$ points (not pixels) which is guaranteed to be an independent set for the unit-ball graph spanned by points in X_{C}. Set $Y_{\mathsf{C}} = Y'$ and return it as the independent set for this call C. This will further guarantee that the final output $\bigcup_{\mathsf{C}} Y_{\mathsf{C}}$ computed by our CNN-Baker will be a valid MIS for the unit-ball graph spanned by input points X.

5.4 Experimental results

We present results for d-MIS here. We also report the generalization results of NN-Baker and its performance on solving minimum vertex cover problems in this section. More results can be found in Appendix Section C.2. We consider 5 families of unit-ball graphs: (2D-dense) consists of a set of graphs, each with points distributed uniformly on a 2D rectangular region with around 40K – 50K points. (2D-sparse) consists of a set of graphs, each with 40k – 50k points distributed uniformly on a 2D rectangular region four times larger than the dense region . (2D-Gaussian) consists of graphs each spanned by points sampled from a Gaussian-mixture distribution with 40K – 50K points over the same region size as the dense region. (3D) consists of graphs each spanned by points sampled from a 3D region. (Torus-4D) consists of graphs each spanned by points sampled from a torus embedded in $\mathbb{R}^4$ (see Appendix Section C.2 for details).

First, we note both CNN-Baker and GNN-Baker return valid MIS for the input unit-ball intersection graphs (as detailed in the implementation of CNN-Baker). To report accuracy, we will need ground truth solutions. However, computing exact solutions for all our test cases is computationally intractable. Instead, we use the output of a SOA solver KaMIS [LSS$^+$17] as the

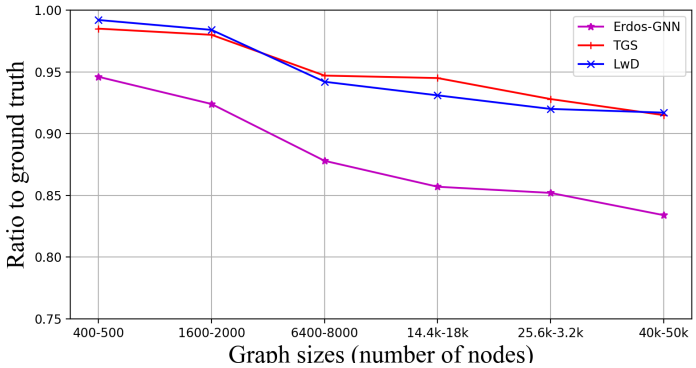


Figure 5.3. Results of Erdos-GNN, TGS and LwD on MIS problems with different sizes of graphs in 2D setting. We report the ratio to ground truth computed by KaMIS.

ground truth solutions, and report the average ratio of MIS obtained over this ground-truth MIS sizes as a metric for accuracy (so the larger this ratio is the better). To validate the accuracy of KaMIS on our training and test data, we compared against an exact solution [XN17] for 20 test sets and found that the accuracy of KaMIS exceeded 99.9% of the optimal solution in all cases. On the other hand, regardless the accuracy of KaMIS, since we are taking the ratio as the metric for accuracy, higher ratio is always better.

Before we show results of CNN-Baker and GNN-Baker, we first show how the SOA GNN-based approaches, TGS [LCK18], LwD [ASS20] and Erdős-GNN [KL20], all of which are specifically designed for graph optimization problems (see discussions in Section 5.3.2), perform as the size of graph increases. Here for each target size, 1000 graphs spanned by points sampled from the same dense distribution in 2D space are used for training, then tested on 100 graphs of roughly the same target size. As shown in Figure 5.3, the accuracy of these approaches decreases as the size of (geometric) graphs increases.

NN-Baker setup. For each of the five setups, we train on 1000 graphs and test on 200 graphs each containing between 40k - 50k points. However, inside our NN-Baker framework, the input domain is partitioned into cells of side-length 12.8, and each cell is further partitioned into 128×128 pixels (each with side-length $= 0.1$). In other words, each cell can be viewed a 128x128 image. This means that the training set for the NN component involved contains

Table 5.1. The ratio of MIS results from different GNNs, GNN-Baker approaches and K-Baker approach to ground truth. (Larger values are better.)

	UNetBaker	Erdős	ErdősBaker	TGS	TGSBaker	LwD	LwDBaker
2D-dense	0.915	0.834	0.923	0.915	0.936	0.917	**0.955**
2D-sparse	0.919	0.822	0.917	0.909	**0.931**	0.908	0.926
2DGaussian	0.917	0.769	0.848	0.905	**0.927**	0.911	0.925
3D	-	0.856	0.930	0.924	0.948	0.902	**0.954**
Torus-4D	-	0.812	0.926	0.923	**0.937**	0.910	**0.937**

only small graphs restricted to such cells. In the end, each training graph consists of around 400 points for the cases of (2D-dense), (3D) and(Torus-4D), and around 100 to 125 points for the case of (2D-sparse). The size of each small training graph for Gaussian case is non-uniform. For CNN-Baker, we apply a UNet which reduces the input 128x128 image to an 8x8 image after 4 down-scaling layers, each of which consists of two convolutional layers, a dropout layer and a max pooling layer. From the 8x8 image, there are then 4 upsampling/concatenation layers to bring the size back to 128x128. This model is denoted by UNet-Baker in Table 5.1. For GNN-Baker, as mentioned in Section 5.3.2, we test TGS-Baker, LwD-Baker, and Erdős-Baker. For each individual neural network involved, we use the same training setup and hyperparameters as those in their official implementations. For LwD and Erdős-GNN, we use node degree as initial node features, while for TGS, we simply use a constant vector. The accuracy of these different methods over the 5 families of graphs are shown in Table 5.1. The number of parameters for these GNN-Bakers range from 50K to 600K, while the UNet-Baker uses around 80M parameters.

As shown in Table 5.1, using Baker framework consistently improves the performance of these SOA neural networks on geometric MIS problems. The improvement over the unsupervised Erdős-GNN (i.e, Erdős-Baker vs. Erdős-GNN) is particularly significant. We remark that we have also experimented with using a simple multi-layer fully connected NN (i.e., a multi-layer version of the NN used in Theorem 5.3.2) to instantiate our NN-Baker, and the performance (under similar number of parameters as our UNet-Baker) is much worse, between 70%-80%.

To show the generalization power of the NN-Baker models, we tested our models on data

with different distributions than the data was trained. For these, we used the data same three 2D distributions as our other results. We observe that accuracy decreases but still improves over the non-Baker version. We report the ratio of MIS results from these generalized models to ground truth in Table 5.2.

Table 5.2. The ratio of MIS results from generalized models to ground truth

Train	Test	UNetBaker	ErdősBaker	TGSBaker	LwDBaker
2D-dense	2D-sparse	0.910	0.901	0.925	0.914
	2DGaussian	0.912	0.832	0.912	0.915
2D-sparse	2D-dense	0.915	0.908	0.926	0.936
	2DGaussian	0.917	0.825	0.919	0.908

Timing. To show that the NN-Baker is more efficient compared to a traditional Baker paradigm, we compare the runtime of our models against a non-neural network based approach. For this, we use KaMIS as our solver for each cell, and call the resulting framework **K-Baker**. In Table 5.3, we show the average time (in seconds) taken to solve a problem with 40k-50k points. We show the runtime of K-Baker set to achieve similar performances to UNetBaker in the table. If we set a similar runtime to our UNetBaker, then performances of K-Baker on dataset 2D-dense, 2D-sparse and 2DGaussian are 0.753, 0.672 and 0.674, which are much poorer than our NN-Baker.

Table 5.3. Average solve times of KaMIS, NN-Baker and K-Baker (seconds)

	KaMIS	K-Baker	UNetBaker	ErdősBaker	TGSBaker	LwDBaker
2D-dense	3385.1	415.19	18.06	32.36	15.72	44.55
2D-sparse	2468.8	402.57	59.97	68.95	33.74	81.06
2DGaussian	3166.7	420.83	58.05	47.45	28.38	62.59

MVC problems. We also evaluate the performance of CNN-Baker and GNN-Baker on solving minimum vertex cover (MVC) problems. The training and test dataset is the same as what we take in MIS problems, and the ground truth is computed by KaMIS. We report the ratio of MVC results from different approaches to ground truth in Table 5.4. Since this is a

minimization problem, results closer to 1.0 are more optimal.

Table 5.4. The ratio of MVC results from different approaches to ground truth.

	UNetBaker	Erdős	ErdősBaker	TGS	TGSBaker	LwD	LwDBaker
2D-dense	1.210	1.141	1.066	1.072	1.054	1.071	1.038
2D-sparse	1.301	1.531	1.248	1.271	1.206	1.274	1.221
2DGaussian	1.234	1.203	1.133	1.084	1.064	1.078	1.066

5.5 Conclusion

The advancement in neural network architectures and their potential to adapt to the structure and input distribution of a problem in a data-driven manner, have brought new ways to tackle traditionally challenging tasks, such as graph optimization problems. In this chapter, we advocate two points of view: (1) Problems in geometric settings can provide structures that both algorithms and neural networks can leverage; for example, they can help to decompose problems into local versions of bounded size and thus lead to more effective NN components. (2) Infusing NN + learning into an algorithmic paradigm can lead to a more powerful framework for hard problems, potentially with theoretical guarantees. While the latter is a view that has already attracted momentum in recent years, our work provides new perspectives (e.g, the decomposition into bounded-size sub-problems) together with some theoretical guarantees, and we show that the resulting method is indeed more powerful empirically too. Our present algorithms currently apply to only geometric intersection graphs. Nevertheless, we believe that such ideas go beyond the geometric setting which we hope to explore in the future, such as to frameworks to obtain algorithms for graphs with bounded tree-width. Indeed, the algorithms and theoretical computer science community has developed many beautiful algorithmic paradigms that may be suitable to be infused with NN+ML ideas.

This Chapter 5, in full, is a reprint of the material as it appears in NN-Baker: A Neural-network Infused Algorithmic Framework for Optimization Problems on Geometric Intersection Graphs, 2021. McCarty, Evan; Zhao, Qi; Sidiropoulos, Anastasios; Wang, Yusu. Conference on

Neural Information Processing System (NeurIPS), 2021. The dissertation author was the primary investigator and author of this paper.

Chapter 6

NN-Steiner: Algorithmic NN Framework for Rectilinear Steiner Minimum Tree

6.1 Introduction

Given a set of points V in $\mathbb{R}^d$, a Steiner tree spanning V is a tree whose vertex set is V together with a set of additional points $S \subset \mathbb{R}^d$ called *Steiner points*. A *rectilinear Steiner tree* is a Steiner tree where all edges are axis-parallel. Given V, the *rectilinear Steiner minimum tree (RSMT)* problem aims to compute the rectilinear Steiner tree spanning V with smallest possible cost (defined as the total length of all edges in the tree). The (rectilinear) Steiner minimum tree problem has important applications in Chip design. However, while its formulation is similar to that of the classical problem of minimum spanning tree, allowing the use of Steiner points makes the problem computationally much harder: Indeed, the RSMT problem is NP-hard to solve. There are simple algorithms to approximate this problem within a constant factor (e.g., factor 3/2 [Hwa76, KR92] and 5/4 [BFK$^+$94]). Theoretically the best known approximation algorithm for RSMT in fixed-dimensional Euclidean space is obtained via the PTAS (polynomial-time approximation scheme) proposed by Arora [Aro98] (which can provide $(1+\varepsilon)$-approximation for a range of optimization problems, such as TSP, in addition to RSMT). Unfortunately, while this algorithm runs in polynomial time, the time complexity depends exponentially on $\frac{1}{\varepsilon}$ and has not yet found its way to practice. In practice especially in chip design, a range of heurstic strategies to balance the tradeoff between time complexity and practical performance.

On the other hand, as we have introduced in Chapter 5, with the recent success of deep neural networks in many applications, there has been a surge on using neural networks to help tackle combinatorial optimization problems [BLP20, KLBS$^+$16, LCK18, SLB$^+$18, GCF$^+$19, SYK19], such as to solve travelling salesman problems (TSP) or some other routing related problems with reinforcement learning [VFJ15, BPL$^+$17, DCL$^+$18, PAL$^+$19]. Very recently, [LCY21] developed the first neural-network based approach for RSMT by finding the so-called rectilinear edge sequences using reinforcement learning. [CKL$^+$22] designed a reinforcement learning framework to solve obstacle avoiding Steiner tree problem.

While there has been some progress in this direction, the significant challenges we mentioned in Chapter 5 remain : Neural networks are often used in an ad hoc manner and our theoretical understanding of the resulting framework is limited. For example, one question is whether a proposed machine learning pipeline has the expressive capacity to solve RSMT problem exactly or approximately. In general, how does the architecture design reflects or leverages the mathematical structure behind RSMT at hand?

One idea to further inject theoretical justification to a principled design of neural approaches for these problems is by leveraging the vast literature on (approximation) algorithms developed as we presented in Chapter 5. In particular, instead of using a neural network to solve RSMT in an end-to-end manner, one can instead use neural components within a high-level algorithmic framework. We have introduced examples in this direction, the line of work using NNs to learn better variable selection decisions within a branch-and-bound (B&B) framework to solve, say, MILP (mixed integer-linear programming) problems [GCF$^+$19, GGK$^+$20, NBG$^+$20]. We also developed this mixed algorithmic-NN framework further in Chapter 5 to solve problems such as maximum independent set in the geometric setting, by first using the Baker's technique to decompose input problems to small instances of *bounded sizes*, and then train a single neural network to solve these instances.

In this chapter, we follow the direction initialized in Chapter 5 and develop an effective mixed neural-algorithmic framework for solving RSMT in $\mathbb{R}^d$. (We will use $\mathbb{R}^2$ to describe the

algorithm below, but it can be extended to $\mathbb{R}^d$ for a constant value of d.) In particular, we develop NN-Steiner, which is a mixed neural-algorithmic framework that leverages the ideas behind Arora's PTAS for RSMT [Aro98]. At a high level, Arora's PTAS partitions the input domain into squares in a hierarchical manner via a shifted quadtree, then solves the problem via a bottom-up dynamic programming (DP) procedure. One key result of [Aro98] is that for each square, the DP step only involves assembling partial solutions (of bounded number) from the four child-square of the present square. In practice, this DP step, while can be implemented in polynomial time, is not practical. In Section 6.3.2, we develop a bi-directional neural-algorithmic approach to simulate the DP. See Figure 6.5 for a high-level illustration. The costly DP step will be replaced by a single neural network component which outputs a learned embedding of partial solutions. Another neural network component will simulate the backward retrieval of Steiner points in a top-down manner. A key advantage is that all the neural network components involved only need to process instances of bounded size, no matter how large the input problem size is.

Main contributions of Chapter 6.

We show that our NN-Steiner achieves the best of both worlds: On the theoretical front, we show in Section 6.3.1 that this framework has the capacity to produce an approximate solution for RSMT using only neural networks of bounded complexity. On the practical front, it uses neural networks to replace a key but also most costly component in Arora's PTAS, thereby leading to an efficient architecture. Furthermore, since the neural component only needs to handle instances of fixed size, the training is more effective. It is also important to note that as the neural component is learning an algorithmic component which remains the same no matter how big the input problem size is, once trained, the entire NN-Steiner generalizes well to problems of much larger sizes which we demonstrate in Section 6.4. Indeed, extensive experimental results in Section 6.4 show that our NN-Steiner achieve better performance than various NN-based or non-NN-based SOA methods.

In summary, we propose NN-Steiner, a novel neural-algorithmic framework to solve the

RSMT problem leveraging the algorithmic idea of Arora's PTAS (which is the theoretically best approximation algorithm for RSMT). This, to our best knowledge, is the first neural architecture of bounded size that has the capacity to approximately solve RSMT. More importantly, the algorithmic alignment of our NN-Steiner also leads to better practical performance than existing SOA methods, especially for large datasets (of more than 1000 points in practice). Finally, we note that the methodology behind our NN-Steiner framework can be extended to handle obstacle avoiding RSMT which we aim to explore as a next step.

Outline of Chapter 6.

This chapter is organized as follows. We first introduce RSMT problems and the Arora's algorithm solving them approximately in Section 6.2. We then present the algorithmic-NN framework, NN-Steiner, and its approximated instantiation in Section 6.3. In Section 6.4, we show experimental results of our NN-Steiner and approximation algorithms or other machine learning baselines. Finally, we make conclusions about our NN-Steiner in Section 6.5.

6.2 Preliminaries

We now introduce the RSMT problem. We will then briefly describe Arora's PTAS for RSMT [Aro98]. For simplicity of presentation, we will assume input points lie in $\mathbb{R}^2$; the definitions and Arora's algorithm can both be extended to $\mathbb{R}^d$.

First, given a set of points P in the plane, a spanning tree T for P is a tree connecting P by a set of line segments (tree edges). If we use L_1 distance (also called rectilinear distance) to measure the distance between two nodes as weights of the edges in the tree, then the resulting tree is called a *rectilinear tree*. Equivalently, we can think that we connect two points $p_1 = (a,b)$ and $p_2 = (c,d)$ by a monotone path where each edge is axis parallel. Then the rectilinear distance between p_1 and p_2 is the (Euclidean) length of such a path; see Figure 6.1. The cost of a rectilinear tree T, denoted by $\texttt{cost}(T)$, is the total weights of all edges.

Definition 6.2.1 (RSMT)**.** *Given a set of points $V \subset \mathbb{R}^2$, the rectilinear Steiner minimum tree*

(RSMT) for V is a rectilinear tree T^* with vertex set $V \cup S$ with minimum cost among all possible choices of S (which could empty). The set S is referred to as Steiner points.

See Figure 6.1 for some simple examples.

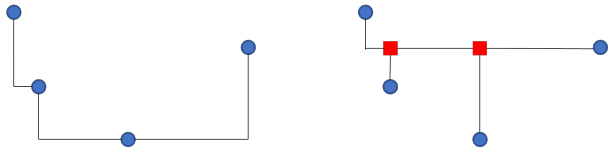


Figure 6.1. (Left) a rectilinear tree spanned by blue points. (Right) a rectilinear Steiner tree, where red points are Steiner points.

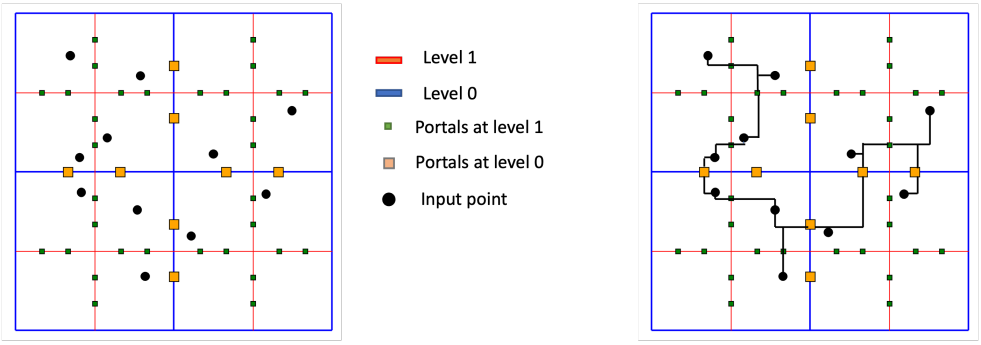
Arora's PTAS

We now briefly describe the high-level idea behind Arora's polynomial time approximation algorithm, which we refer to as *Arora-PTAS* from now on. First, for simplicity of exposition, we assume that the input points $V \subset \mathbb{R}$ have integral coordinates, and that the diameter of point set V is $O(n)$ with $n = |V|$. (A perturbation process is given in [Aro98] to round input points so that this assumption holds without changing the theoretical guarantee of the algorithm.) W.o.l.g suppose input points V are contained in a bounding box of size length $L = O(n)$.

Step 1: Construct a shifted quartree.

Randomly pick $0 < a, b < L$, and then translate the point set by vector (a, b) in the plane. We then compute a quadtree after the shift, where the splitting of quadtree cells terminate the first time the side-length becomes less than 1. In other words, the quadtree T is a tree where each internal node has degree 4. The root has level 0, and is associated with a square of side length L. Any node $v \in T$ at level i is associated with a square (quadtree cell) $\square_v$ of size (i.e., side length) $\frac{L}{2^i}$. Splitting the horizontal and vertical sides in the middle decomposes $\square_v$ into four child-quadtree cells (squares) each of size $\frac{L}{2^{i+1}}$, corresponding to the four children nodes of $v \in T$ at level $i+1$. As all points have integer coordinates, each leaf cell can contain at most 1 point. As the bounding box side length is $L = O(n)$, the height (max-level) of quadtree is thus

$h_T = O(\log L)$, the total number of nodes in T (and thus the number of cells across all levels) is bounded by $O(n \log L)$.



(a) A quadtree decomposition of the input points

(b) A $(2,1)$-light rectilinear Steiner tree

Figure 6.2. (a) shows a two-level quadtree over the input points (black dots). Each side of quadtree cell has 2 portals. (b) gives an example of a $(2,1)$-light rectilinear Steiner tree.

We will now consider a special families of (rectilinear) Steinter trees, which intuitively can cross any quadtree cell (at any level) only a bounded number of times.

Definition 6.2.2. *Let m, r be positive integers. An m-regular set of portals for a shifted quadtree is a set of points on the sides of cells in the quadtree, such that each cell (square) has a portal at each of its 4 corners and m other equally-spaced portals on each of its four sides. A Steiner tree is (m,r)-light if it crosses each edge of each square at most r times and always at a portal.*

See Figure 6.2 for an example of quadtree decomposition and a (m,r)-light rectilinear Steiner tree.

Step 2: Dynamic programming (DP)

It is shown in [Aro98] that there exists (m,r)-light rectilinear Steiner trees whose cost approximate the optimal cost of RSMT. Hence our goal now is to compute such a good (m,r)-light rectilinear Steiner tree. In particular, Arora's proposed to use a dynamic programming to construct it in a bottom-up manner. We sketch the idea here.

We process all quadtree cells in bottom-up manner. For a fixed quadtree cell A, consider a (m,r)-light Steiner tree T restricted to A: this will give rise to some Steiner forest T_{A} in A

which can exit this cell only via portals on its four sides. In particular, the portion of the Steiner tree outside this cell A can be solved independently as long as we know the following *portal-configuration*: (i) the set of *exiting-portals* on the side of this cell that will be used by T (which connect points outside A with those inside), and (ii) how these exiting-portals are connected by trees in the Steiner forest T_{A}. In other words, a portal-configuration is the "boundary" of the Steiner forest T_{A} on the side of A.

Let Ξ_{A} be the set of portal-configurations for a cell A, and set $D = |\Xi|$. It is easy to show that $D \leq (4m+4)^{4r}\mathsf{Bell}(4r) \leq (4m+4)^{8r}$, where $\mathsf{Bell}(k)$ is the so-called *Bell number* that gives the number of possible partitions of a set of cardinality k. Our goal now is to compute, for each portal configuration $\sigma \in \Xi$, the minimum cost $cost(\sigma)$ of any rectilinear Steiner-forest within A that gives rise to this boundary condition. Overall, assuming an arbitrary but fixed order of portal-configurations in $\Xi = \{\sigma_1, \ldots, \sigma_D\}$, the costs of all portal-configurations can then be represented by a vector $\vec{\mathsf{C}}_{\mathsf{A}} \in \mathbb{R}^D$, where $\vec{\mathsf{C}}_{\mathsf{A}}[i] = cost(\sigma_i)$. We call $\vec{\mathsf{C}}_{\mathsf{A}}$ the *cost-vector for* A, which stores costs of all possible partial solutions for points of P contained within cell A. We now describe the DP algorithm to compute this cost-vector for all cells in a bottom-up manner in decreasing order of levels. Given $\sigma \in \Xi$, we also abuse the notation and use $\vec{\mathsf{C}}_{\mathsf{A}}[\sigma]$ to denote $\vec{\mathsf{C}}_{\mathsf{A}}[i]$ if $\sigma = \sigma_i$.

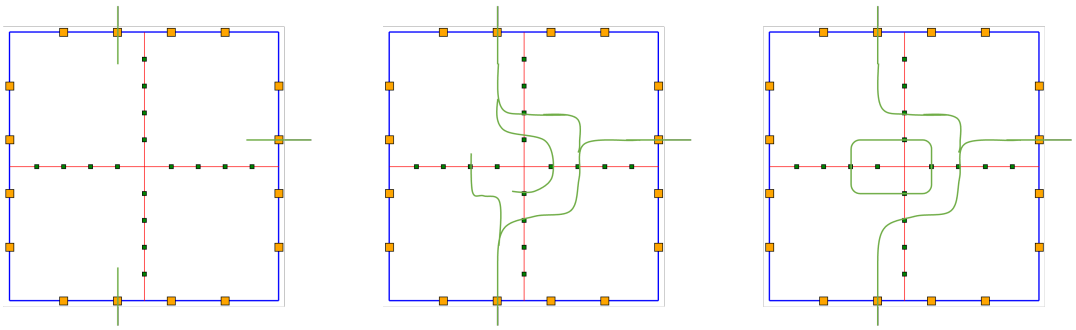
Base case: A **is a leaf cell.** In this case, there is at most one point p from P contained in A. Since only portals can be used as Steiner points, we can thus simply enumerate all possible ways p can be connected to the chosen portals in the portal-configuration.

Inductive step: A **is not a leaf cell.** The four child-cells $\mathsf{A}_1, \ldots, \mathsf{A}_4$ of A are from the level below A and thus by inductive hypothesis we have already computed the the cost-vectors $\vec{\mathsf{C}}_{\mathsf{A}_k}$s for each of them, for $k = 1, \ldots, 4$. Consider any portal-configuration $\sigma \in \Xi$. We simply need to enumerate all choices of portal-configurations $\tau_1, \ldots, \tau_4$ for child-cell $\mathsf{A}_1, \ldots, \mathsf{A}_4$ respectively that are *consistent with* σ, meaning that their portals along common sides are the same, and the connected components of portals from 4 child-cells don't form cycles (thus is still induced by a

valid Steiner forest). We then have that

$$cost(\sigma) = \min_{\tau_1,\ldots,\tau_4 \text{ consistent with } \sigma} \vec{C}_{A_1}[\tau_1] + \vec{C}_{A_2}[\tau_2] + \vec{C}_{A_3}[\tau_3] + \vec{C}_{A_4}[\tau_4]. \tag{6.1}$$

The choices of $\tau_1, \ldots, \tau_4$ that give rise to the minimum cost $cost(\sigma)$ above is called the set of *child-cell portal configurations generating* σ. See Figure 6.3 for an example of the inductive step.



(a) **(b)** **(c)**

Figure 6.3. To compute cost of (4,2)-Steiner tree with the 3 chosen portals in figure (a), we first solve sub-problems in the 4 child-cells. Specifically, we find (4,2)-Steiner forests with all portals combination in child-cells consistent with interfaces as shown in figure (b). Figure (c) is an invalid example.

Final construction of approximate RSMT. At the end of DP, after we compute the cost-vector for the root cell (which contains all input points P inside), we identify the portal-configuration $\sigma^* \in \Xi$ that has the lowest cost. To obtain the corresponding rectilinear Steiner tree whose cost is $\texttt{cost}(\sigma^*)$, we need to now perform a top-down back-propagation: in particular, from σ^* for the root cell we can then retrieve the set of child-cell portal-configurations $\tau_1^*, \ldots, \tau_4^*$ (from the four children of the root cell) generating σ^*. Repeat this process till we reach all leaf cells. Let S denote the union of all portals chosen in these optimal portal-configurations for all cells; these will serve as the Steiner points that we need to add. In other words, we now simply compute an optimal rectilinear minimum spanning tree T^* for the point set $P \cup S$, which can be done in $O(m \log m)$ time where $m = |P \cup S|$. The following theorem guarantees the T^*

constructed above indeed is an approximate RSMT. See Appendix D for the proof of this theorem following from [Aro98].

Theorem 6.2.3. Structure Theorem. *If the shifts* $0 \leq a, b < L$ *are chosen uniformly randomly in range* $[0, L]$*, then with probability at least* $1/2$*, the rectilinear Steiner tree* T^* *computed above has a cost that is at most* $\left(1 + \frac{8}{r} + O(\frac{4 \log L}{m})\right)\mathsf{OPT}$*, where* OPT *is the cost of the optimal RSMT.*

6.3 NN-Steiner

Although the running time of Arora's algorithm is polynomial in theory, its computation is very expensive in practice as we need to find all possible combinations of *r* portals from *m* portals on each side of square. Instead of enumerating all portal-configurations in dynamic programming in a brute-force manner, we propose an algorithmic-NN framework, NN-Steiner, infusing neural networks (NNs) into Arora's algorithm, to help us select Steiner points from the set of portals to build the output Steiner tree. In particular, in Section 6.3.1, we first show that in fact, the key components within the DP algorithm can be simulated exactly (not approximated) by certain neural networks. However, such neural networks are not efficient either. Then in Section 6.3.2, we show a practical instantiation of NN-Steiner and we will demonstrate its practical performance in Section 6.4.

6.3.1 Theoretical NN-Steiner to simulate Arora's PTAS

We can simulate key components in the the dynamic programming step of Arora's PTAS. In particular, we will use four neural networks: $\mathsf{NN}_{\mathsf{base}}$ and $\mathsf{NN}_{\mathsf{DP}}$ to implement the base case and inductive step of the DP, respectively, to compute encoding of cost-vectors for all quadtree cells in a bottom-up manner, and then $\mathsf{NN}_{\mathsf{top}}$ and $\mathsf{NN}_{\mathsf{retrieve}}$ to obtain the top level optimal portal-configuration, as well as backtrack that choice to all cells in the tree to retrieve optimal portal-configurations from all cells in a top down manner. Once portals used in all cells are retrieved, we use the union of them as the set of Steiner points and compute the

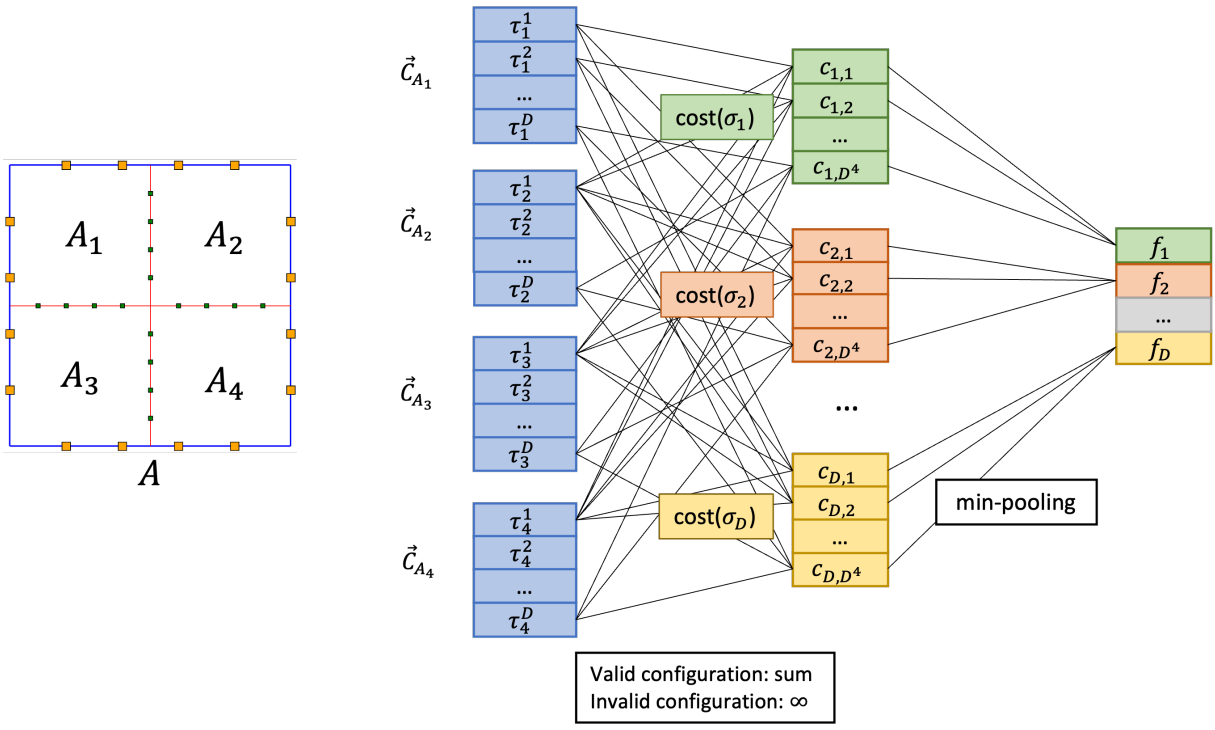


Figure 6.4. A neural network $\mathsf{NN}_{\mathsf{DP}}$ simulating function f_{DP} for dynamic programming.

minimum rectilinear spanning tree spanned by original points and these Steiner points as the output rectilinear Steiner tree.

It turns out that there exist designs and parameters of these four neural networks so that they simulate Arora's DP algorithm exactly; and also these NNs are each of only **bounded size** independent of n (only depending on parameters m and r, which we set to be large constant in practice). More specifically, we describe how to construct the neural network $\mathsf{NN}_{\mathsf{DP}}$ to simulate one inductive step in the DP algorithm.

First, recall that the inductive step of the DP algorithm can be rewritten as applying a function $f_{DP} : (\mathbb{R}^D)^4 \to \mathbb{R}^D$. In particular, for any quadtree cell A with child-cells $\mathsf{A}_1, \ldots, \mathsf{A}_4$, the input to f_{DP} will be the four cost-vectors $(\vec{\mathsf{C}}_{\mathsf{A}_1}, \vec{\mathsf{C}}_{\mathsf{A}_2}, \vec{\mathsf{C}}_{\mathsf{A}_3}, \vec{\mathsf{C}}_{\mathsf{A}_4}) \in (\mathbb{R}^D)^4$, and the output is the cost-vector $\vec{\mathsf{C}}_{\mathsf{A}} \in \mathbb{R}^D$.

Note that Eqn (6.1) gives how to compute each entry in output vector $f_{DP}(\vec{\mathsf{C}}_{\mathsf{A}_1}, \ldots, \vec{\mathsf{C}}_{\mathsf{A}_4})$. In other words, f_{DP} has a simple form, which can be modeled as a certain linear functions

followed by a min-pooling, which can be easily simulated by a neural network as shown in Figure 6.4. In particular, to compute the $\vec{\mathsf{C}}_{\mathsf{A}}[i]$ (corresponding to the cost of portal-configuration σ_i), each neuron c_ℓ takes in a set of four portal-configurations $\tau_j \in \mathsf{A}_j$, $j = 1,\ldots,4$, that are consistent with σ_i, and it simply does a sum operation. Then the output will take a min-pooling of all values at c_ℓs. Given that there are at most D^4 number of such four portal-configurations for each σ_i, the entire model has complexity $\Theta(D^5)$ (where recall $D \leq (4m+4)^{8r}$ is independent of the input point set size n). In other words, there is a $\mathsf{NN}_{\mathsf{DP}}$ of bounded complexity that simulate the DP step exactly. We summarize with the following theorem for the existence of NNs to implement Arora's PTAS:

Theorem 6.3.1. *There exist four neural networks, each of only bounded size depending only on m and r, that can simulate the dynamic programming algorithm of Arora's PTAS, such that the resulting algorithm can find an* $\big(1+\frac{8}{r}+O(\frac{4\log L}{m})\big)$*-approximate rectilinear Steiner tree (i.e., its cost is at most* $\big(1+\frac{8}{r}+O(\frac{4\log L}{m})\big)\mathsf{OPT}$*). The entire framework will call these neural networks only* $O(n\log L) = O(n\log n)$ *times.*

6.3.2 Practical Instantiation of NN-Steiner

We introduce our NN-Steiner framework and show that in theory, there exists simple NNs that can exactly simulate key components in Arora's dynamic programming. That theoretical instantiation however is not practical as it essentially is still explicitly encoding the exponential number of portal-configurations (exponential in m, r, but not n). In what follows, we present a practical instantiation of this framework in which the four aforementioned neural networks are implemented by variants of graph neural networks (GNNs). See Figure 6.5 for a high-level overview of our NN-Steiner pipeline.

Forward pass.

The forward processing involves two neural networks $\mathsf{GNN}_{\mathsf{base}}$ and $\mathsf{GNN}_{\mathsf{DP}}$, and call the $\mathsf{GNN}_{\mathsf{DP}}$ in bottom-up manner to compute implicit encoding of costs of possible portal-configurations.

Base case neural network $\mathsf{GNN}_{\mathsf{base}}$.

At the leaf level of Arora's algorithm, each cell contains at most 1 point. In practice we terminate the quadtree decomposition when a cell contains no more than k_b points, where k_b is a hyperparameter. Given a leaf cell A with $P_{\mathsf{A}} \subset P$ being the set of input points contained in A, ideally, the neural network $\mathsf{GNN}_{\mathsf{base}}$ should take some representations of $\mathsf{A}, P_{\mathsf{A}}$, as well as the set of $4m+4$ portals on the side of this cell as input, and output a d_c-dimensional vector (in $\mathbb{R}^{\mathsf{d}_c}$) as an implicit encoding of cost-vector $\vec{\mathsf{C}}_{\mathsf{A}} \in \mathbb{R}^D$; note $\mathsf{d}_c << D$ in practice.

However, to better encode the relative relations among portals, we will use a graph to encode the input entities, and instead of outputing a graph-level representation, we will use node-features which can be used for the later inductive step neural network $\mathsf{GNN}_{\mathsf{DP}}$.

More specifically, first, we solve RSMT T_{A} for points P_{A} within this leaf cell A – we can afford to do so as the size $|P_{\mathsf{A}}| \leq \mathsf{k}_b$ is supposed to be a small constant, and we use the state-of-the-art solver GeoSteiner [JWWZ18] to solve it. We then connect each portal on the side of A to its closet point in T_{A} to form a graph $\widehat{G}_{\mathsf{A}}$, which the graph neural network $\mathsf{GNN}_{\mathsf{base}}$ operates on. Specifically, we take a variant of GIN [XHLJ19] taking edge features into the message function as the $\mathsf{GNN}_{\mathsf{base}}$ in practice. The *relative coordinates* of all points (including portals) are used as part of the node features, which is generated by shifting the cell so that its left-bottom vertex is at origin $(0,0)$. The rectilinear distance between nodes are used as edge features. At the output layer, each graph node is associated with a $\mathbb{R}^{\mathsf{d}_c}$-vector as node features, and the collection of them over all portals form an implicit encoding of the cost-vector for all portal-configurations of this cell.

The advantage of terminating with k_b points in a cell is two-fold: (1) For a quadtree cell with very few points, it is harder to learn a meaningful encoding of portal-configurations. However, if k_b is small, then majority of cells in a quadtree will have few points inside (note that a complete degree-4 tree will have around 75% nodes at the leaf level!). Hence training on such collection of cells tend to provide bad supervision which affect the effectiveness of learning. (2) Furthermore, with larger k_b we have fewer number of cells to train on, which improves efficiency

too. On the other hand, as the k_b increases, the base case deviates further from Arora's PTAS, and that affects the performance of the framework too. In our experiments k_b is a hyperparameter (see Section 6.4 for its effect and choice).

DP inductive step neural network $\mathsf{GNN}_{\mathsf{DP}}$.

Next, we use another graph neural network $\mathsf{GNN}_{\mathsf{DP}}$ to simulate the function f_{DP} which as mentioned in Section 6.3.1 is equivalent to the DP step in Arora's PTAS. More specifically, given a cell A at level i, let $\mathsf{A}_1, \ldots, \mathsf{A}_4$ be its 4 child-cells at level $(i+1)$ in fixed order. Intuitively, $\mathsf{GNN}_{\mathsf{DP}}$ will take the current implicit encoding of cost-vectors for each child cell, and generate a implicit encoding for the parent cell A. Again, to better leverage relations among portals, instead of using MLP, we use a graph neural network, GIN, to implement $\mathsf{GNN}_{\mathsf{DP}}$. First, we construct a graph $\widehat{G}_{\mathsf{A}}$, where the node set is the set of portals at level $i+1$ and level i on sides of cell A, $\mathsf{A}_1, \ldots, \mathsf{A}_4$. We connect each portal to its neighbor portals along the cell sides. Besides, we connect each level $i+1$ portal on sides of A_i to all level i portals also appearing on sides of A_i.

The initial node features are setup as follows: portals at level i take their relative coordinates as node features. Portals at level $i+1$ have multiple d_c-dimensional node feature vectors from different child-cells. We input them into MLPs and take the output d_c-dimensional vector as initial node features of portals at level $i+1$.

Similar to the base NN $\mathsf{GNN}_{\mathsf{base}}$, at the output layer, each graph node has a d_c-dimensional node feature vector as an encoding for partial solutions for A. The **same** NN $\mathsf{GNN}_{\mathsf{DP}}$ will be applied once for every internal nodes of the quadtree in a bottom-up manner.

Backward pass.

As described above, the forward pass will apply the base NN $\mathsf{GNN}_{\mathsf{base}}$ to all leaf cells, and then $\mathsf{GNN}_{\mathsf{DP}}$ to all internal nodes, to simulate the bottom-up DP algorithm of Arora's PTAS. Now that we have the encoding of partial solutions for all cells, we use two more neural networks to simulate the backtracking stage of the DP algorithm.

Root-level retrieval $\mathsf{GNN}_{\mathsf{top}}$.

The goal of $\mathsf{GNN}_{\mathsf{top}}$ is to map the current encoding of cost-vector for the root cell A to a likelihood map on all portals $\rho_{\mathsf{A}} : \mathit{Portals}(\mathsf{A}) \to \mathbb{R}$, which encodes how likely a portal is chosen as a Steiner point; here, $\mathit{Portals}(\mathsf{A})$ denote the set of portals on the sides of A. This is easily achieved by a GIN over a graph $\widehat{G}'_{\mathsf{A}}$ connecting portals as follows: connecting each portal at the root level to its neighbor portals at the same level, and set the initial node features as those from the output after running $\mathsf{GNN}_{\mathsf{DP}}$ for this root cell. The output is a likelihood value $\rho_{\mathsf{A}}(p)$ for each portal p. Note that f_{A} can be thought of as the restriction of the *portal-likelihood map* $\rho : \mathscr{P} \to \mathbb{R}$ to only the set of portals $\mathit{Portals}(\mathsf{A})$; where $\mathscr{P}$ denotes the set of all portals across all levels.

Iterative backward retrieval step $\mathsf{GNN}_{\mathsf{retrieve}}$.

Next, using the portal-likelihood map ρ_{A}, we wish to induce portal-likelihood maps for portals from its children, and repeat this process till we compute the likelihood f for all portals. This is achieved by using a neural network $\mathsf{GNN}_{\mathsf{retrieve}}$ repeatedly to all internal quadtree nodes. In particular, for each node A with child-cells $\mathsf{A}_1, \ldots, \mathsf{A}_4$, suppose we have already computed the likelihood for portals on the side of this cell. $\mathsf{GNN}_{\mathsf{retrieve}}$ will operate on the same graph $\widehat{G}$ as constructed for $\mathsf{GNN}_{\mathsf{DP}}$. The difference from $\mathsf{GNN}_{\mathsf{DP}}$ is the design of node features. We use the likelihood of portals from the parent cell A as their node features, while the node features of portals from child-cells are their encodings computed in the forward pass. In the final output layer, all nodes has a single value as its feature, which is the likelihood value.

Retrieval of Steiner points.

After the downward pass, we have a portal-likelihood map $\rho : \mathscr{P} \to \mathbb{R}$ over all portals. We use an iterative algorithm (shown in Algorithm 1) to choose the final set of Steiner points from portals, as well as construct a rectilinear Steiner tree. Intuitively, since in general we do not have a good threshold for choosing portals, Algorithm 1 keeps lowering the threshold to choose Steiner points as long as the cost of the resulting rectilinear Steiner tree is still decreasing, and

stops till it starts to increase. We also point out that when we construct a rectilinear Steiner tree with picked portals, Steiner points with degree less than 3 are removed (as they are redundant).

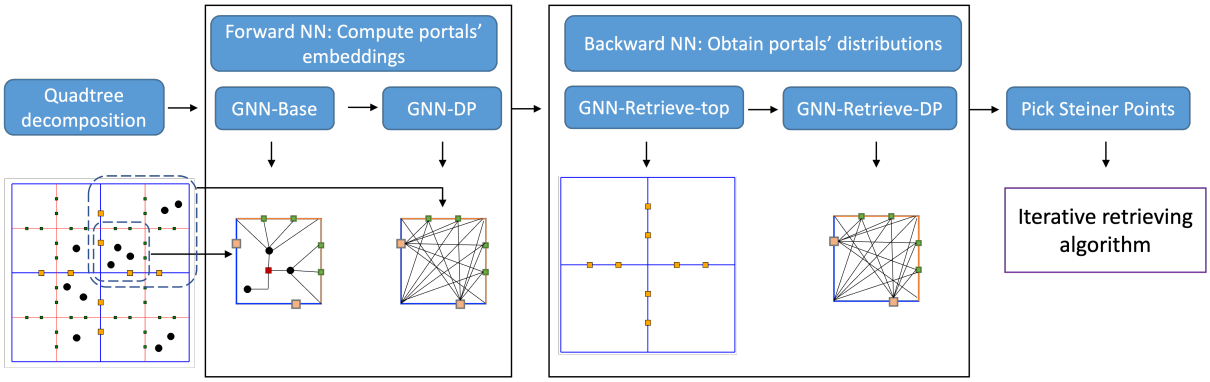


Figure 6.5. Pipeline of a NN-Steiner instantiation

Algorithm 1. Iterative retrieving algorithm

Input:

The set of portals with likelihood

Hyperparameters t_0 and δ_t

Output:

A rectilinear Steiner tree

Take all portals with likelihood $\geq t_0$ as Steiner points, construct rectilinear Steiner tree T_0, compute cost c_0

for $i = 1; i < t_0/\delta_t; i++$ **do**

$t_i = t_{i-1} - \delta_t$

Take all portals with likelihood $\geq t_i$ as Steiner points, construct rectilinear Steiner tree T_i, compute cost c_i

if $c_i > c_{i-1}$ **then**

Set $j^* = i - 1$, break the loop

end if

end for

return T_{j^*}

A further improvement in architecture.

In the instantiation of the four neural networks above, we note that information about input points were only directly used by $\mathsf{GNN}_{\mathsf{base}}$ for leaf cells. Later in $\mathsf{GNN}_{\mathsf{DP}}$, such information was only implicitly used as in encoding at portals. Here we hope add stronger supervision of points' spatial distributions more directly. Specifically, in $\mathsf{GNN}_{\mathsf{base}}$, we use another GNN over the

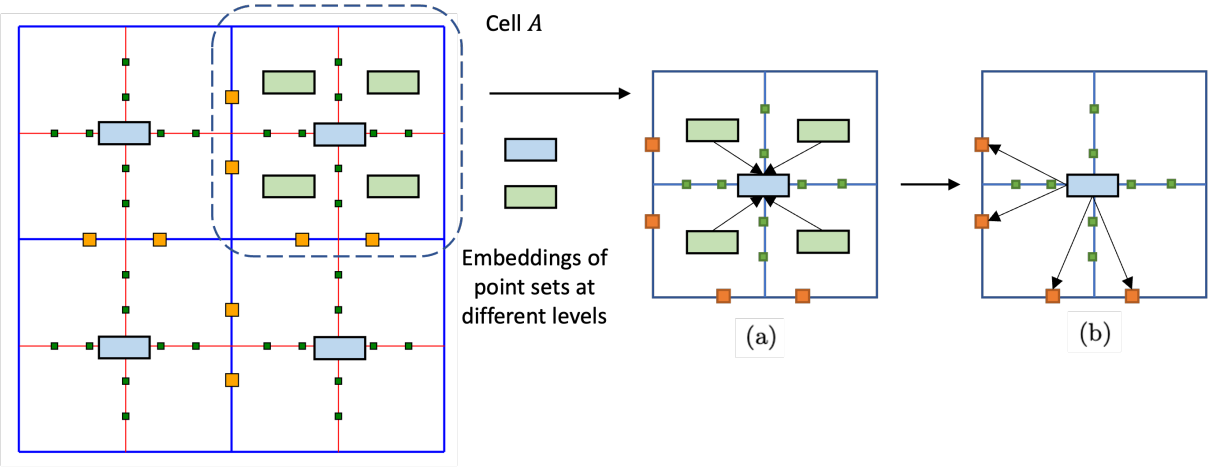


Figure 6.6. Besides portals' feature vector, we also pass points encoding vector in $\mathsf{GNN}_{\mathsf{DP}}$. In (a), the points encoding vector v_{A} of cell A obtained by infusing those of 4 child-cells of A with a neural network. In (b), v_{A} is used to update portals' feature vectors on sides of A.

computed RSMT of points (not portals) within this cell to obtain a fixed dimensional d_c'-vector as the encoding of points in this cell. Then in $\mathsf{GNN}_{\mathsf{DP}}$, when we process A with child-cells $\mathsf{A}_1, \ldots, \mathsf{A}_4$, each child cell already has a d_c'-vector as encoding of points inside. We first use another neural network (a simple MLP) to fuse those four d_c'-vectors into a single d_c'-vector v_{A} to encode points information in the parent cell A. This vector is used during the message passing stage of $\mathsf{GNN}_{\mathsf{DP}}$ to update the node features for each portal. In specific, we create a new node in graph $\widehat{G}_{\mathsf{A}}$ connecting to all portals on sides of A with initial node feature v_{A}. This node passes its feature to portals in $\mathsf{GNN}_{\mathsf{DP}}$. We refer to this setup with improvement as NN-Steiner-II. See Figure 6.6 for a simple illustration.

6.4 Experimental performance

In this section, we present experimental results of baselines and NN-Steiner on 2d RSMT. Point sets are sampled from a mixture of uniform and Gaussian distribution on the 2d-plane. We consider 4 different sizes of point sets: Dataset-i $(i = 1,2,3,4)$. The point sets in 4 datasets consists of about 500, 1500, 2500 or 5000 points respectively. Each dataset consists of 3000 point sets. See Appendix D for the point generation method in details.

In our experiments, we use the output of a state-of-the-art RSMT solver Geosteiner [JWWZ18] as ground truth solutions and report accuracies of different approaches based on this. In order to train our NN-Steiner framework, we need supervision about portals selection. Instead of collecting training labels by really solving RSMT with Arora-PTAS, we turn to the shifted-Groundtruth. Specifically, we first make quadtree decomposition and portals picking over the input point sets as what Arora-PTAS does. We then solve RSMT problems over point sets by Geosteiner and output ground truth solutions. The shifted-Groundtruth is obtained by shifting Steiner points found in ground truth to the portals most closed to them. We take the cross-entropy between the likelihood learned by NN-Steiner and shifted-Groundtruth as the training loss function. NN-Steiner models are initialized by Glorot initialization and trained with batch size 64 and 1000 epochs. The cross-entropy losses are minimized by Adam optimizer to train our model with learning rate of 3.0×10^{-4}. The learning rate will decay by 0.96 after each 100 epochs. To tune the two hyperparameters, the size of a base cell k_b and the number of portals on each side of a cell m, we explore our NN-Steiner for $\mathsf{k}_b \in \{20, 30, 40, 50, 60, 80, 100\}$ and $m \in \{10, 20, 30, 50, 70, 100\}$ by grid search. When testing, we set $t_0 = 0.6$ and $\delta_t = 0.05$ in the Steiner points iterative retrieving algorithm (Algorithm 1). We compare our NN-Steiner to two approximation algorithms, PD-II+HVW+DAS [ACH$^+$18] and FLUTE [CW07]. In PD-II+HVW+DAS, we set the hyperparameter $\alpha = 0$ to obtain a solution with minimum edge length. For FLUTE, we solve problems with 3 different settings, $A = 3$ (default setting), $A = 10$ and $A = 18$ (the most accurate setting). We also compare to a machine learning based approach, REST[LCY21], in which we set the transformation hyperparamter $T = 18$ to get the minimum cost.

We train and test REST, NN-Steiner and NN-Steiner-II on Dataset-i ($i = 1, 2, 3$) respectively and obtain their average accuracies by 10 folds cross validation. In order to measure the generalization performances of REST and our two NN-Steiner frameworks, we take their models trained on Dataset-3 and test their accuracies on Dataset-4. We report the accuracies and percentages of "wins" of baselines, NN-Steiner and NN-Steiner-II on the 4 datasets in Table 6.1. As

shown in the table, NN-Steiner-II achieve higher accuracies than both approximation algorithms and machine learning baselines in all datasets except Dataset-1 in which point sets have the smallest size. The empirical results indicate our NN-Steiner-II approach has better performance on large instances. The comparison between NN-Steiner and NN-Steiner-II indicates that the direct encoding of points distribution is significant in neural network framework. We report the average training time of REST and NN-Steiner-II on 2700 instances and average test time on 300 instances in Table 6.2. We also present the average running time of FLUTE-10 and FLUTE-18 on solving 300 instances in Table 6.2.

Table 6.1. Accuracy and winning rate of baselines and NN-Steiner frameworks

Methods	Dataset-1	Dataset-2	Dataset-3	Dataset-4
FL-3	1.109 /5.5%	1.132 /0.8%	1.163 /0.2%	1.161 /0.6%
FL-10	1.097 /37.4%	1.121 /8.6%	1.124 /6.1%	1.123 /9.5%
FL-18	1.095 /58.4%	1.114 /38.6%	1.116 /36.5%	1.119 /40.8%
PD-II	1.108 /8.3%	1.136 /0.4%	1.140 /0.7%	1.143 /1.2%
REST	1.102 /30.5%	1.119 /11.6%	1.122 /9.4%	1.135 /3.4%
NNSt	1.119 /0.0%	1.123 /6.2%	1.125 /3.6%	1.129 /5.2%
NNSt-II	1.115 /2.8%	1.110 /49.8%	1.111 /53.1%	1.114 /54.9%

Table 6.2. Training and test / running time (minutes)

Dataset		FL-10	FL-18	REST	NNSt-II
Dataset-1	Training	-	-	192.3	216.8
	Test (running)	4.2	11.5	1.4	1.6
Dataset-2	Training	-	-	211.4	253.6
	Test (running)	11.3	56.2	2.3	2.8
Dataset-3	Training	-	-	268.5	334.7
	Test (running)	24.6	97.2	3.5	5.4
Dataset-4	Test (running)	43.1	198.8	5.6	7.5

To further compare the generalization capability of REST and NN-Steiner framework, we train and test the machine learning models on different dataset, and report their performance in Table 6.3. Besides, we test REST and NN-Steiner-II models trained on Dataset-3 over

increasingly larger point sets to explore their performances on large instances. Each large test dataset consists of 300 point sets. We present the test results in Figure 6.7. As shown in Table 6.3 and Figure 6.7, NN-Steiner-II has consistently better generalization performance than REST in large point sets.

Table 6.3. Generalization performances (accuracy and winning rate) of REST and NN-Steiner-II

Training set	Test set	REST	NN-Steiner-II
Dataset-2	Dataset-1	1.098 /79.7%	1.113 /20.3%
	Dataset-3	1.136 /12.5%	1.115 /87.5%
	Dataset-4	1.144 /7.6%	1.118 /92.4%
Dataset-3	Dataset-1	1.096 /78.5%	1.110 /21.5%
	Dataset-2	1.115 /23.6%	1.103 /76.4%
	Dataset-4	1.135 /9.8%	1.114 /90.2%

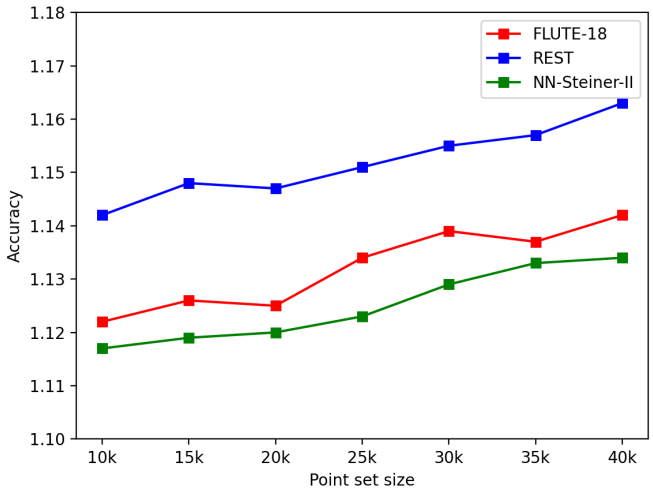
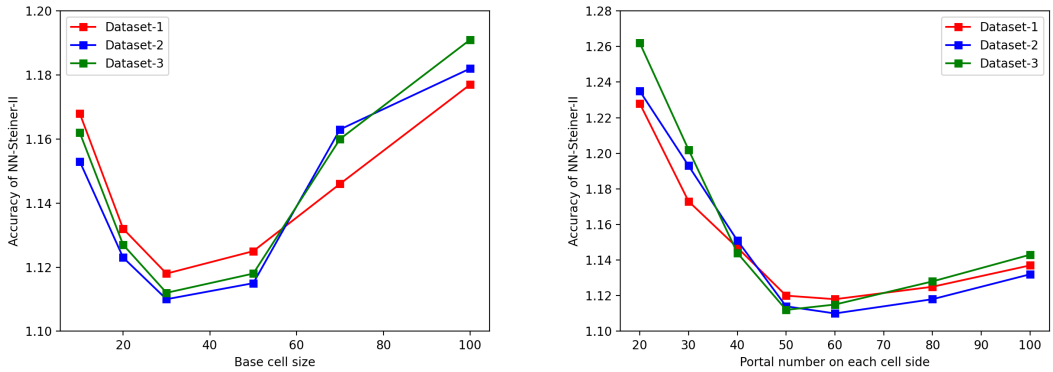


Figure 6.7. Performance of baselines and NN-Steiner-II on large instances.

Finally, we discuss the influence of two hyperparameters, base cell size k_b and portal number on each cell side m. We fix $m = 50$, train and test our NN-Steiner-II models with k_b increasing from 10 to 100. We also fix $k_b = 30$, and explore the performance of NN-Steiner-II with m ranging from 20 to 100. As shown in Figure 6.8, a small k_b is insufficient to learn a meaningful encoding of points distribution and portals configuration as we mentioned in Section



(a) **(b)**

Figure 6.8. Performance of NN-Steiner-II with different k_b and m.

6.3.2, while a large k_b increases the complexity of base case solving. Besides, it is not necessary that performance of NN-Steiner-II improves as m becomes larger, because a large m increases the complexity of models.

6.5 Conclusion

The development of solving combinatorial optimization problems and routing problems by neural networks require more theoretical understanding, especially the expressive capabilities of neural networks. Our NN-Steiner extends the scope of algorithmic-NN framework of NN-Baker to Steiner-like problems such that geometric setting of path-wise problems can also provide structures that both algorithms and neural networks can leverage. Specifically, RSMT can be decomposed into hierarchical sub-problems and solved by approximation algorithms with dynamic programming. Infusing neural networks into the algorithmic structure to simulate the dynamic programming leads to an efficient and effective framework with theoretical guarantees. Our NN-Steiner works on $\mathbb{R}^2$ RSMT problems well, but it has potential to be extended in higher dimensional Steiner minimum tree problems or obstacle avoiding Steiner minimum tree problems.

This Chapter 6, in full, has been submitted for publication of the material as it may

appear in NN-Steiner: A Mixed Neural-algorithmic Approach for Minimum Rectilinear Steiner Tree Problem, 2022. Zhao, Qi; Wang, Yusu; Kahng, Andrew; Sidiropoulos, Anastasios. International Conference on Computer-Aided Design, 2022. The dissertation author was the primary investigator and author of this paper.

Chapter 7

Conclusions and Future Work

Starting from our overarching question,

> *Geometric algorithms and topological quantities are effective at capturing hidden structure and features in data, as well as providing efficient algorithms for them. How can we combine and integrate such ideas and methods with modern machine learning pipelines to further augment and enhance power and performance of these pipelines?*

we present four different works augmenting machine learning pipelines with topological persistence ideas or integrating geometric algorithms with neural networks.

First, we proposed a new weighted-kernel for persistence-based summaries, WKPI, for classification tasks. We proved this kernel is positive semi-definite and its induced metric is stable to perturbations. We also designed a metric learning approach to learn the best weight function for WKPI-kernel from labelled data that can encode the importance of different locations in the persistence diagram. We applied our learned WKPI-kernel to the task of neuron-cell classification and graph classification. Our approach performs consistently better than other persistence-based kernels. Most importantly, comparing to graph kernels and graph neural networks baselines, our new framework achieves similar or better results among benchmarks from social networks and chemical components. Considering that our approach is purely based on graph structure without any attributes, these results are more meaningful.

Second, we presented a new graph neural network architecture, PEGN, leveraging topological structure information with persistence images. We provided the intuition about

the power of local persistence images from specific subgraphs, say neighborhood of nodes, on differentiating nodes in graphs from a theoretical view. Then we incorporated local topological information into message passing of graph neural networks in an attention manner. In our experiments, baselines include approaches using multi-hop neighborhood information or node-feature-based attention. Experimental results show that PEGN performs similarly or outperforms state-of-the-art graph neural networks in node classification tasks.

Third, we designed a novel algorithmic-NN framework, NN-Baker, solving combinatorial optimization problems in geometric setting. We showed that problems in geometric setting had structures that both approximation algorithms and neural networks can leverage, thus infusing neural networks into an algorithmic paradigm had the potential to lead to a more powerful frameworks for higher efficiency and theoretical guarantees. In specific, our NN-Baker was developed by infusing neural networks into Baker-paradigm, a partitioning based polynomial approximation scheme. We applied NN-Baker to solve maximum independent set problem, a classic combinatorial optimization problem known as NP-hard, on geometric intersection graphs. In experiments, our approach improves the performance of existing graph neural network based baselines on large instances by a large margin. Notice that our Baker-paradigm and NN-Baker can also solve some other combinatorial optimization problems in geometric setting.

Finally, we developed a second algorithmic-NN framework, NN-Steiner, solving rectilinear Steiner minimum tree problems. Similar to problems in geometric setting, problems connecting points on a plane like TSP and SMT also had a nice structure beneficial to both algorithms and neural networks. Specifically, we simulated the dynamic programming procedure in Arora's algorithm, a polynomial approximation algorithm, by a series of neural networks. Our approach outperforms both approximation algorithms and machine learning based baselines and has better generalization ability in experiments. This work is the first supervised learning approach solving RSMT problems and demonstrates that algorithmic-NN idea is not restricted in node-wise problems.

Future directions.

We first discuss future work in leveraging topological ideas. To design GNNs with stronger expressive power, we have to encode higher order information in message passing. We have discovered fruitful features with 0-dim and 1-dim extended persistence diagrams from graphs. In the future, one can investigate higher dimensional homological features spanning more than 2 nodes in graphs, and inject them into message passing. There has benn some extension of GNNs to high-order GNNs. However, the investigation has been limited. We would like to understand among which potential simplices messages should pass, and how the messages should be processed more effectively across them. Specifically, we can explore how we can learn constructing simplices from input graphs in a more intelligent and data-driven way according to their topological properties, which allows nodes or other simplices not connected by edges to transit features in a more meaningful way. This direction has a potential capacity to alleviate or even overcome the over-smoothing and un-reach problems of GNNs.

Then we discuss future work in solving geometric problems. We infused neural networks into Baker's paradigm and Arora's algorithm to solve node-wise and routing like problems respectively. Notice that in the past decades, there has been a huge literature in theoretical computer science field about approximation algorithms. One can explore the literature and find more powerful and efficient algorithmic structures for combinatorial optimization problems, problems in high dimensional space especially, and infuse neural networks into them. For example, our NN-Steiner and some other machine learning approaches focuses on solving routing like problems on 2-d planes. One can investigate an algorithmic-NN approach to solve those problems in 3-d or even higher dimensional spaces. There are also some other interesting problems like convex hull and Delaunay triangulation. Moreover, another interesting direction is to generalize this algorithmic-NN idea beyond geometry setting. For example, we would like to infuse neural networks into approximation algorithms solving MIS problems on general graphs.

Appendix A

Chapter 3: Appendix

A.1 Missing proofs

A.1.1 Proof of Lemma 3.3.2

Consider an arbitrary collection of n persistence images $\{\mathrm{PI}_1, \ldots, \mathrm{PI}_n\}$ (i.e, a collection of n vectors in $\mathbb{R}^N$). Set $K = [k_{ij}]_{n\times n}$ to be the $n \times n$ kernel matrix where $k_{ij} = k_w(\mathrm{PI}_i, \mathrm{PI}_j)$. Now given any vector $v = (v_1, v_2, ..., v_n)^T$, we have that:

$$\begin{aligned} v^T K v &= \sum_{i,j=1}^{n} v_i v_j k_{ij} \\ &= \sum_{i,j=1}^{n} v_i v_j \sum_{s=1}^{m} \omega(p_s) e^{-\frac{(\mathrm{PI}_i(s)-\mathrm{PI}_j(s))^2}{2\sigma^2}} \\ &= \sum_{s=1}^{m} \omega(p_s) \sum_{i,j=1}^{n} v_i v_j e^{-\frac{(\mathrm{PI}_i(s)-\mathrm{PI}_j(s))^2}{2\sigma^2}}. \end{aligned}$$

Because Gaussian kernel is positive semi-definite and the weight-function ω is non-negative, $v^T K v \geq 0$ for any $v \in \mathbb{R}^N$. Hence the WKPI kernel is positive semi-definite.

A.1.2 Proof of Theorem 3.3.4

By Definitions 3.3.1 and 3.3.3, combined with the fact that $1 - e^{-x} \leq x$ for any $x \in \mathbb{R}$, we have that:

$$\begin{aligned}
\mathrm{D}_\omega{}^2(A,B) &= k_w(\mathrm{PI}_A,\mathrm{PI}_A) + k_w(\mathrm{PI}_B,\mathrm{PI}_B) - 2k_w(\mathrm{PI}_A,\mathrm{PI}_B) \\
&= 2\sum_{s=1}^{N} \omega(p_s) - 2\sum_{s=1}^{n} \omega(p_s) e^{-\frac{(\mathrm{PI}_A(s)-\mathrm{PI}_B(s))^2}{\sigma^2}} \\
&= 2\sum_{s=1}^{N} \omega(p_s)(1 - e^{-\frac{(\mathrm{PI}_A(s)-\mathrm{PI}_B(s))^2}{\sigma^2}}) \\
&\leq 2c_w \sum_{s=1}^{N} (1 - e^{-\frac{(\mathrm{PI}_A(s)-\mathrm{PI}_B(s))^2}{\sigma^2}}) \\
&\leq 2\frac{c_w}{\sigma^2} \sum_{s=1}^{n} (\mathrm{PI}_A(s) - \mathrm{PI}_B(s))^2 \\
&\leq 2\frac{c_w}{\sigma^2} ||\mathrm{PI}_A - \mathrm{PI}_B||_2^2
\end{aligned}$$

Furthermore, by Theorem 10 of [AEK+17], when the distribution ϕ_u to in Definition 2.1 is the normalized Gaussian $\phi_u(z) = \frac{1}{2\pi\tau^2} e^{-\frac{\|z-u\|^2}{2\tau^2}}$, and the weight function $\alpha = 1$, we have that $\|\mathrm{PI}_A - \mathrm{PI}_B\|_2 \leq \sqrt{\frac{10}{\pi}} \cdot \frac{1}{\tau} \cdot d_{W,1}(A,B)$. (Intuitively, view two persistence diagrams A and B as two (appropriate) measures, and $d_{W,1}(A,B)$ is then the "earth-mover" distance between them so as to convert the measure corresponding to A to that for B, where the cost is measured by the total L_1-distance that all mass have to travel.) Combining this with the inequalities for $\mathrm{D}_\omega{}^2(A,B)$ above, the theorem then follows.

A.1.3 Proof of Theorem 3.3.6

We first show the following properties of matrix L which will be useful for the proof later.

Lemma A.1.1. *The matrix L is symmetric and positive semi-definite. Furthermore, for every*

vector $f \in \mathbb{R}^n$, we have

$$f^T L f = \frac{1}{2} \sum_{i,j=1}^{n} \Lambda_{ij} (f_i - f_j)^2 \tag{A.1}$$

Proof. By construction, it is easy to see that L is symmetric as matrices Λ and G are. The positive semi-definiteness follows from Eqn (A.1) which we prove now.

$$\begin{aligned}
f^T L f = f^T G f - f^T \Lambda f \ &= \sum_{i=1}^{n} f_i^2 g_{ii} - \sum_{i,j=1}^{n} f_i f_j \Lambda_{ij} \\
&= \frac{1}{2} \Big(\sum_{i=1}^{n} f_i^2 g_{ii} + \sum_{j=1}^{n} f_j^2 g_{jj} - \sum_{i,j=1}^{n} 2 f_i f_j \Lambda_{ij} \Big) \\
&= \frac{1}{2} \Big(\sum_{i=1}^{n} f_i^2 \sum_{j=1}^{n} \Lambda_{ij} + \sum_{j=1}^{n} f_j^2 \sum_{i=1}^{n} \Lambda_{ji} \\
&\qquad - \sum_{i,j=1}^{n} 2 f_i f_j \Lambda_{ij} \Big) \\
&= \frac{1}{2} \sum_{i,j=1}^{n} \Lambda_{ij} \cdot (f_i^2 + f_j^2 - 2 f_i f_j) \\
&= \frac{1}{2} \sum_{i,j=1}^{n} \Lambda_{ij} (f_i - f_j)^2
\end{aligned}$$

The lemma then follows. □

We now prove the statement in Theorem 3.3.6. Recall that the definition of various matrices, and that h_t's are the row vectors of matrix H. For simplicity, in the derivations below, we use $D(i,j)$ to denote the ω-induced WKPI-distance $\mathrm{D}_\omega(A_i, A_j)$ between persistence diagrams A_i and A_j. Applying Lemma A.1.1, we have:

$$\begin{aligned}
\mathrm{Tr}(HLH^T) = \sum_{t=1}^{k} (HLH^T)_{tt} \ &= \sum_{t=1}^{k} h_t L h_t^T \\
&= \sum_{t=1}^{k} \frac{1}{2} \cdot \sum_{j_1, j_2 = 1}^{n} D^2(j_1, j_2)(h_{t,j_1} - h_{t,j_2})^2 \\
&= \sum_{t=1}^{k} \frac{1}{2} \cdot \sum_{j_1, j_2 = 1}^{n} D^2(j_1, j_2)(h_{t,j_1}^2 + h_{t,j_2}^2 - 2 h_{t,j_1} h_{t,j_2}).
\end{aligned} \tag{A.2}$$

Now by definition of h_{ti}, it is non-zero only when $i \in \mathscr{C}_t$. Combined with Eqn (A.2), it then follows that:

$$\begin{aligned}
\mathrm{Tr}(HLH^T) &= \sum_{t=1}^{k} \frac{1}{2} \cdot \Big(\sum_{j_1 \in \mathscr{C}_t, j_2 \in [1,n]} \frac{D^2(j_1,j_2)}{cost_\omega(t,\cdot)} \\
&+ \sum_{j_1 \in [1,n], j_2 \in \mathscr{C}_t} \frac{D^2(j_1,j_2)}{cost_\omega(t,\cdot)} - 2 \sum_{j_1, j_2 \in \mathscr{C}_t} \frac{D^2(j_1,j_2)}{cost_\omega(t,\cdot)} \Big) \\
&= \sum_{t=1}^{k} \frac{1}{2} \Big(\sum_{j_1 \in \mathscr{C}_t, j_2 \notin \mathscr{C}_t} \frac{D^2(j_1,j_2)}{cost_\omega(t,\cdot)} \\
&+ \sum_{j_1 \notin \mathscr{C}_t, j_2 \in \mathscr{C}_t} \frac{D^2(j_1,j_2)}{cost_\omega(t,\cdot)} \Big) \\
&= \sum_{t=1}^{k} \sum_{j_1 \in A_t, j_2 \notin A_t} \frac{D^2(j_1,j_2)}{cost_\omega(t,\cdot)} \\
&= \sum_{t=1}^{k} \frac{cost_\omega(t,\cdot) - cost_\omega(t,t)}{cost_\omega(t,\cdot)} \\
&= k - TC(\omega)
\end{aligned}$$

This proves the first statement in Theorem 3.3.6. We now show that the matrix HGH^T is the $k \times k$ identity matrix $\mathbf{I}$. Specifically, first consider $s \neq t \in [1,k]$; we claim:

$$(HGH^T)_{st} = h_s G h_t^T = \sum_{j_1, j_2 = 1}^{n} h_{sj_1} G_{j_1 j_2} h_{t j_2} \quad = 0.$$

It equals to 0 because h_{sj_1} is non-zero only for $j_1 \in \mathscr{C}_s$, while $h_{t j_2}$ is non-zero only for $j_2 \in \mathscr{C}_t$. However, for such a pair of j_1 and j_2, obviously $j_1 \neq j_2$, which means that $G_{j_1 j_2} = 0$. Hence the sum is 0 for all possible j_1 and j_2's.

Now for the diagonal entries of the matrix HGH^T, we have that for any $t \in [1,k]$:

$$\begin{aligned}(HGH^T)_{tt} = h_t G h_t^T &= \sum_{j_1,j_2=1}^{n} h_{tj_1} G_{j_1,j_2} h_{tj_2} \\ &= \sum_{j_1,j_2 \in \mathscr{C}_t} \frac{G_{j_1 j_2}}{cost_{\omega}(t,\cdot)} \quad = \sum_{j_1 \in \mathscr{C}_t} \frac{G_{j_1 j_1}}{cost_{\omega}(t,\cdot)} \\ &= \sum_{j_1 \in \mathscr{C}_t} \frac{\sum_{\ell=1}^{n} D^2(j_1,\ell)}{cost_{\omega}(t,\cdot)} \\ &= \frac{\sum_{j_1 \in \mathscr{C}_t, \ell \in [1,n]} D^2(j_1,\ell)}{cost_{\omega}(t,\cdot)} \\ &= \frac{cost_{\omega}(t,\cdot)}{cost_{\omega}(t,\cdot)} \quad = 1.\end{aligned}$$

This finishes the proof that $HGH^T = \mathbf{I}$, and completes the proof of Theorem 3.3.6.

A.2 More details for experiments

A.2.1 More on neuron experiments

Setup for persistence images.

Persistence-images are both needed for the methodology of [AEK+17] and as input for our WKPI-distance. For each dataset, the persistence image for each object inside is computed within the rectangular bounding box of the points from all persistence diagrams of input trees. The y-direction is then discretized to 40 uniform intervals, while the x-direction is discretized accordingly so that each pixel is a square. In our experiments, the choice of discretization (to obtain persistence images) does not seem to have a significant effect on the final results. For persistence image (PI) approach of [AEK+17], we show results both for the unweighted persistence images (PI-CONST), and one, denoted by PI-PL, where the weight function $\alpha : \mathbb{R}^2 \to \mathbb{R}$ (for Definition 2.1) is the following piecewise-linear function (modified from one proposed by Adams et al. [AEK+17]) where b the largest persistence for any persistent-

point among all persistence diagrams.

$$\alpha(x,y) = \begin{cases} \frac{|y-x|}{b} & |y-x| < b \text{ and } y > 0 \\ \frac{|-y-x|}{b} & |-y-x| < b \text{ and } y < 0 \\ 1 & otherwise \end{cases} \tag{A.3}$$

Weight function learnt.

In Figure A.1 we show the heatmaps of the learned weight-function ω^* for both datasets. Interestingly, we note that the important branching features (points in the birth-death plane with high ω^* values) separating the two primary classes (i.e, for **Neuron-Binary** dataset) is different from those important for classifying neurons from one of the two primary classes (the interneuron class) into the four secondary classes (i.e, the **Neuron-Multi** dataset). Also high importance (weight) points may not have high persistence. In the future, it would be interesting to investigate whether the important branch features are also biochemically important.

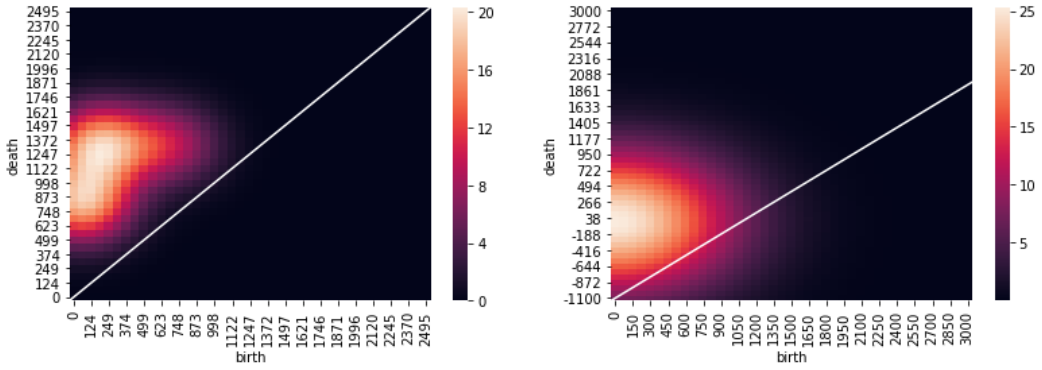


Figure A.1. Heatmaps of learned weight-function ω^* for Neuron-Binary (left) and Neuron-Multi (right) datasets. Each point in this plane indicates birth-death of some branching feature. Warmer color indicates higher ω^* value. x- and y-axies are birth / death time.

A.2.2 More on graph classification experiments

Benchmark datasets for graph classification.

Below we first give a brief description of the benchmark datasets we used in our experiments. These are collected from the literature.

NCI1 and **NCI109** [SSL$^+$11] consist of two balanced subsets of datasets of chemical compounds screened for activity against non-small cell lung cancer and ovarian cancer cell lines, respectively.

PTC [HKKS01] is a dataset of graph structures of chemical molecules from rats and mice which is designed for the predictive toxicology challenge 2000-2001.

DD [DD03] is a data set of 1178 protein structures. Each protein is represented by a graph, in which the nodes are amino acids and two nodes are connected by an edge if they are less than 6 Angstroms apart. They are classified according to whether they are enzymes or not.

PROTEINS [BOS$^+$05] contains graphs of protein. In each graph, a node represents a secondary structure element (SSE) within protein structure, i.e. helices, sheets and turns. Edges connect nodes if they are neighbours along amino acid sequence or neighbours in protein structure space. Every node is connected to its three nearest spatial neighbours.

MUTAG [DLdCD$^+$91] is a dataset collecting 188 mutagenic aromatic and heteroaromatic nitro compounds labelled according to whether they have a mutagenic effect on the Gramnegtive bacterium Salmonella typhimurium.

REDDIT-5K and **REDDIT-12K** [YV15] consist of graph representing the discussions on the online forum Reddit. In these datasets, nodes represent users and edges between two nodes represent whether one of these two users leave comments to the other or not. In REDDIT-5K, graphs are collected from 5 sub-forums, and they are labelled by to which sub-forums they belong. In REDDIT-12K, there are 11 sub-forums involved, and the labels are similar to those in REDDIT-5K.

IMDB-BINARY and **IMDB-MULTI** [YV15] are dataset consists of networks of 1000

actors or actresses who played roles in movies in IMDB. In each graph, a node represents an actor or actress, and an edge connects two nodes when they appear in the same movie. In IMDB-BINARY, graphs are classified into Action and Romance genres. In IMDB-MULTI, they are collected from three different genres: Comedy, Romance and Sci-Fi.

In our experiments, for REDDIT-12K dataset, due to the larger size of the dataset (with about 13K graphs), we deploy the EigenPro method ([MB17], code available at https://github.com/EigenPro/EigenPro-matlab), which is a preconditioned (stochastic) gradient descent iteration) to significantly improve the efficiency of kernel-SVM.

Persistence generation.

To generate persistence diagram summaries, we want to put a meaningful descriptor function on input graphs. We consider two choices in our experiments: (a) the *Ricci-curvature function* $f_c : G \to \mathbb{R}$, where $f_c(x)$ is a discrete Ricci curvature for graphs as introduced in [LLY11]; and (b) *Jaccard-index function* $f_J : G \to \mathbb{R}$.

Then Ollivier's Ricci curvature between two nodes u and v is

$$\kappa_{uv}^{\alpha} = 1 - W(m_u^{\alpha}, m_v^{\alpha})/d(u,v) \tag{A.4}$$

where $W(\cdot,\cdot)$ is Wasserstein distance between two measures and $d(u,v)$ is the distance between two nodes, and probability measure m_u^{α} around node u is defined as

$$m_x^{\alpha}(x) = \begin{cases} \alpha & x = u \\ (1-\alpha)/n_u & x \in \mathcal{N}(u) \\ 0 & \text{otherwise} \end{cases} \tag{A.5}$$

$n_u = |\mathcal{N}(u)|$ and α is a parameter within $[0,1]$. In our experiments, we set $\alpha = 0.5$.

In particular, the Jaccard-index of an edge $(u,v) \in G$ in the graph is defined as $\rho(u,v) = \frac{|NN(u) \cap NN(v)|}{|NN(u) \cup NN(v)|}$, where $NN(x)$ refers to the set of neighbors of node x in G. The Jaccard index

has been commonly used as a way to measure edge-similarity[1]. As in the case for neuron data sets, we take the union of the 0-th persistence diagrams induced by both the sublevel-set and the superlevel-set filtrations of the descriptor function f, and convert it to a persistence image as input to our WKPI-classification framework [2].

In all results reported in main text and in Table A.1, Ricci curvature function is used for the small chemical compounds data sets (NCI1, NCI9, PTC and MUTAG), while Jaccard function is used for the two proteins datasets (PROTEIN and DD) as well as the social/IMDB networks (IMDB's and REDDIT's). Both 0-dim and 1-dim extented persistence diagrams are employed. In general, we observe that Ricci curvature is more sensitive to accurate graph local structure, while Jaccard function is better for noisy graphs (with noisy edge). In Figure A.2, we show the heatmaps of the weight function before and after our metric learning for NCI1 and REDDIT-5K datasets. In particular, the left column shows the heatmaps of the initialized weight function, while the right column shows the heatmaps of the optimal weight function as learned by our algorithm.

Additional results.

Many graph classification methods have been proposed in the literature. We compare our results with a range of existing approaches, which includes state-of-the-art results on different datasets: six graph-kernel based approaches: RetGK[ZWX+18], FGSD[VZ17], Weisfeiler-Lehman kernel (WL)[SSL+11], Weisfeiler-Lehman optimal assignment kernel (WL-OA)[KGW16], Deep Graphlet kernel (DGK)[YV15], and the very recent persistent Weisfeiler-Lehman kernel (P-WL-UC)[3] [RBB19]; two graph neural networks: PATCHYSAN (PSCN) [NAK16], Graph Isomorphism Network (GIN)[XHLJ19]; as well as the topology-signature-based neural network (DL-TDA) [HKNU17].

[1] We modify our persistence algorithm slightly to handle the edge-valued Jaccard index function

[2] We expect that using the 0-th zigzag persistence diagrams will provide better results. However, we choose to use only 0-th standard persistence as it can be easily implemented to run in $O(n \log n)$ time using a simple union-find data structure.

[3] Note that results for three version of persistent WL kernels are reported in their paper. We take the one (P-WL-UC, with uniform node labels) that performs the best from their Table 1.

Table A.1. Classification accuracy on graphs. Our results are in columns WKPI-kM and WKPI-kC.

Dataset	Previous approaches							Our appraches	
	RetGK	WL	DGK	FGSD	PSCN	GIN	P-WL-UC	WKPI-kM	WKPI-kC
NCI1	84.5	85.4	80.3	79.8	76.3	82.7	85.6	**87.5**	84.5
NCI109	-	84.5	80.3	78.8	-	-	85.1	85.9	**87.4**
PTC	62.5	55.4	60.1	62.8	62.3	66.6	63.5	62.7	**68.1**
PROTEIN	75.8	71.2	75.7	72.4	75.0	76.2	75.9	**78.5**	75.2
DD	81.6	78.6	-	77.1	76.2	-	78.5	**82.0**	80.3
MUTAG	90.3	84.4	87.4	**92.1**	89	90	85.2	85.8	88.3
IMDB-BINARY	71.9	70.8	67.0	71.0	71.0	75.1	73.0	70.7	**75.4**
IMDB-MULTI	47.7	49.8	44.6	45.2	45.2	**52.3**	-	46.4	49.5
REDDIT-5K	56.1	51.2	41.3	47.8	49.1	57.5	-	59.1	**59.5**
REDDIT-12K	48.7	32.6	32.2	-	41.3	-	-	47.4	48.4
Average	-	66.39	-	-	-	-	-	69.99	71.66

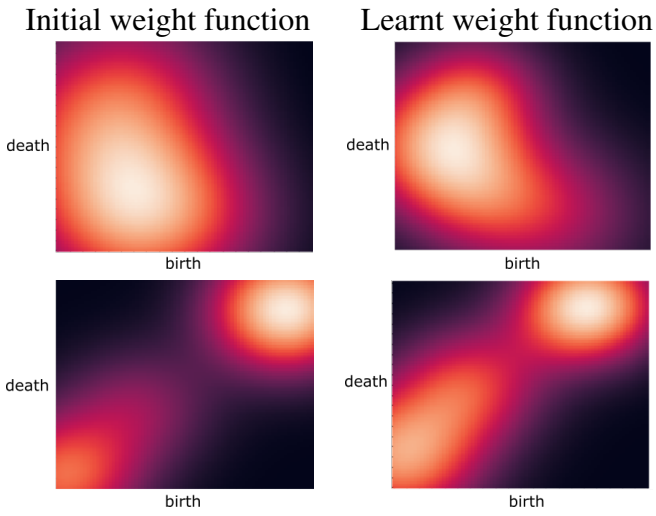


Figure A.2. Heatmap of initialized weight function (left column) and that of the learnt weight-function ω^* (right column). Top row shows results for NCI1 data set; while bottom row contains those for REDDIT-5K data set.

Additional results of comparing our results with more existing methods are given in Table A.1. The results of DL-TDA (topological signature based deep learning framework) [HKNU17] are not listed in Table A.1, as only the classification accuracy for REDDIT-5K (accuracy 54.5%) and REDDIT-12K (44.5%) are given in their paper (although their paper contains many more results on other objects, such as images). While also not listed in this table, we note that our results also outperform the newly independently proposed general neural network architecture for persistence representations reported in the very recent preprint [CCI$^+$19]. Comparison with other topological-based non-neural network approaches are given below.

Topological-based methods on graph data.

Here we compare our WKPI-framework with the performance of several state-of-the-art persistence-based classification frameworks, including: PWGK [KFH18], SW [CCO17], PI [AEK$^+$17] and PF [LY18].

Table A.2. Classification accuracy on graphs for topology-based methods.

Datasets	Existing TDA approaches					Our WKPI framework	
	PWGK	PI-CONST	PI-PL	SW	PF	deWKPI-kM	deWKPI-kC
NCI1	73.3	72.5	72.1	80.1	81.7	**87.2**	84.7
NCI109	71.5	74.3	73.1	75.5	78.5	85.5	**86.9**
PTC	62.2	61.3	64.2	**64.5**	62.4	61.1	64.3
PROTEIN	73.6	72.2	69.1	76.4	75.2	**77.4**	75.6
DD	75.2	74.2	76.8	78.9	79.4	**79.8**	79.1
MUTAG	82.0	85.2	83.5	87.1	85.6	85.5	**88.0**
IMDB-BINARY	66.8	65.5	69.7	69.6	71.2	70.6	**75.4**
IMDB-MULTI	43.4	42.5	46.4	48.7	48.6	47.1	**48.8**
REDDIT-5K	47.6	52.2	51.7	53.8	56.2	58.7	**59.3**
REDDIT-12K	38.5	43.3	45.7	**48.3**	47.6	45.2	44.5
Average	63.41	64.3	65.23	68.29	68.64	69.81	**70.66**

We use the same setup as our WKPI-framework to train these two metrics, and use their resulting kernels for SVM to classify the benchmark graph datasets. WKPI-framework outperforms the existing approaches and alternative metric learning methods on all datasets except **MUTAG**. WKPI-kM (i.e, WKPI-kmeans) and WKPI-kC (i.e, WKPI-kcenter) improve

Table A.3. Graph classification accuracy of GIN and RetGK on graph benchmarks with the same nested cross validation setup

	NCI1	NCI109	PTC	PROTEIN	DD
RetGK	84.5±0.2	84.8±0.2	62.9±1.6	75.4±0.6	81.6±0.4
GIN	82.4±1.6	86.5±1.5	67.8±6.5	76.7±2.6	81.1±2.5
	MUTAG	IMDB-BIN	IMDB-MULTI	Reddit5K	Reddit12K
RetGK	90.0±1.1	72.3±1.0	47.7±0.4	55.8 ±0.5	48.5± 0.2
GIN	89.0±7.5	75.6±5.3	52.4±3.1	57.2±1.5	47.9± 2.1

the accuracy by $3.9\% - 11.9\%$ and $5.4\% - 13.5\%$, respectively. Besides, we show results by another experimental setup. In 10-fold cross validation, choose m and σ leading to the smallest cost function value, then evaluate the classifier on the test set. Repeat this process 10 times. That is, m and σ are not the hyperparameters of the SVM classifiers, but are determined by the metrics learning. We refer to these two approaches as deWKPI-kM and deWKPI-kC in accordance with the initialization methods. The classification accuracy of all these methods are reported in Table A.2.

Appendix B

Chapter 4: Appendix

B.1 More details for experiments

B.1.1 More on datasets

Cora, Citeseer and Pubmed are citation graphs in which nodes represent documents and edges represent the undirected citation relations. Node features are elements of a bag-of-words representations of documents. In two Coauthor graphs, nodes represent authors which are connected by an edge if they jointly authored a paper. Node features are keywords for each author's papers, and node class labels are given by the authors' most active study fields. In two Amazon graphs, nodes represent goods and two nodes are connected if consumers frequently buy them together. Node features are bag-of-words encoded product reviews, and class labels indicate the product category.

B.1.2 More experiments results

Besides Ollivier Ricci curvature, we also make experiments by taking Jaccard index as the weight function for graphs and construct subgraphs by picking 1-hop and 2-hop neighbourhoods around each node. Denote them **PEGN-JI-1** and **PEGN-JI-2**, respectively. The results are reported in Table B.1.

Table B.1. Classification Accuracies on Benchmark Datasets

Method	Cora	Citeseer	PubMed	Coauthor CS	Coauthor Physics	Amazon Computer	Amazon Photo
PEGN-JI-2	82.5±0.5	71.7±0.6	78.7±0.6	92.7±0.3	94.1±0.3	84.0±1.0	92.2±0.5
PEGN-JI-1	82.4±0.5	71.7±0.5	78.5±0.6	92.7±0.3	94.1±0.2	86.1±0.6	92.7±0.4

Algorithm 2. δ-Net clustering

Input: A set of points $P = \{p_1, p_2, ..., p_n\}$, radius $\delta > 0$

Output: A δ-net Q of P, and a set of clusters Π where each point $q \in Q$ corresponding to a cluster in Π

$P' = \{\}$, $\Pi = \{\}$
while $P' \neq$ **do**
  Randomly pick point $p_i \in P \setminus P'$
  Obtain a cluster $C = \{p_j : ||p_i - p_j||_2 \leq \delta | p_j \in P\}$
  $P' = P' \cup C$, $\Pi = \Pi \cup \{C\}$
end while
Set a node set $V = \{\}$ and an edge set $E = \{\}$
for $C_i \in \Pi$ **do**
  Create a node v_i, add it to V
end for
for $(v_i, v_j) \in V \times V$ **do**
  if $i \neq j$ and C_i, C_j have overlapping points in P **then**
    Add edge $\{v_i, v_j\}$ to E
  end if
end for
Construct net Q with node set V and edge set E

B.2 More details for HS-GNN

δ-Net Clustering

See Algorithm 2 for the detailed δ-Net clustering algorithm used to find hierarchies in CNT bundles for HS-GNN.

Appendix C

Chapter 5: Appendix

C.1 Missing proofs

C.1.1 Proof of Theorem 5.2.1

In what follows, we use the notation $[k]$ to denote the set of integers $1, 2, \ldots, k$. Recall that each cell of our axis-parallel hyper-grid Γ has side-length $\frac{2d}{\varepsilon}$. Also recall that we put on a second-level more refined lattice (grid) which gives us pixels of side-length $\delta' = \delta/\sqrt{d}$. For simplicity of argument, we assume that $\frac{1}{\delta'}$ (i.e, the reciprocal of the pixel side-length) is an integer. This condition can be removed by slightly more careful analysis. In what follows, we will show that the output of Baker's paradigm, Y, is a $(1+2\varepsilon, 1+6\delta)$-bi-criteria approximation in expectation for d-MIS for $0 < \varepsilon, \delta < 1/3$, with the desired time complexity.

Now fix some optimal solution $Y^* \subseteq X$ for d-MIS over X. Let $Y' \subseteq Y^*$ be the subset of points in Y^* such that the unit balls centered at them intersect the shifted grid $\Gamma + \tau$; that is

$$Y' = \{p \in Y^* : \mathsf{ball}(p,1) \cap (\Gamma + \tau) \neq \emptyset\}.$$

Note that $Y' \subseteq X'$ as constructed in (Step 2) of Chapter 5. For any point $p \in X$, for any $i \in [d]$, let $\mathscr{E}_{p,i}$ be the event that $\mathsf{ball}(p,1)$ intersects some $(d-1)$-dimensional hyperplane of the shifted grid $\Gamma + \tau$, that is orthogonal to e_i. The event $\mathscr{E}_{p,i}$ occurs precisely when the i-th coordinate of τ falls within an interval of length 2 out of the side-length $\frac{2d}{\varepsilon}$ of a cell: This is

because it is equivalent to that we have a segment of length $2d/\varepsilon$ (along the i-th axis in direction e_i), and a point is within distance 1 to either endpoint of this segment. Hence the total probability is the same as a point to fall within an interval of length $1+1=2$ out of an interval of length $2d/\varepsilon$. Since τ is chosen uniformly at random, we get

$$\Pr[\mathscr{E}_{p,i}] = \frac{2}{2d/\varepsilon} = \varepsilon/d.$$

Let $\mathscr{E}_p$ be the event $p \in Y'$. By the union bound, we have

$$\Pr[\mathscr{E}_p] = \Pr[\bigcup_{i\in[d]} \mathscr{E}_{p,i}] \leq \sum_{i\in[d]} \Pr[\mathscr{E}_{p,i}] = \varepsilon.$$

Using linearity of expectation, the above implies

$$\mathbf{E}[|Y'|] = \sum_{p\in Y^*} \Pr[\mathscr{E}_p] \leq |Y^*| \cdot \varepsilon = \mathsf{OPT}(X) \cdot \varepsilon. \tag{C.1}$$

In other words, if we only consider points in $X \setminus X'$, then the size of the optimal solution of d-MIS for $X \setminus X'$ can only be at most $\mathsf{OPT}(X) \cdot \varepsilon$ less than $\mathsf{OPT}(X)$, that is, it is at least $(1-\varepsilon)\mathsf{OPT}(X)$.

Recall that $\widehat{X}$ is the "snapping" of of points in $X \setminus X'$ to the second-level lattice points: in particular, a point $p = (p_1, \ldots, p_d) \in X \setminus X'$ is mapped to the point $\widehat{p} = (\delta' \lfloor \frac{p_1}{\delta'} \rfloor, \ldots, \delta' \lfloor \frac{p_d}{\delta'} \rfloor)$ which intuitively is the (d-dimensional analog of the) left-bottom of the pixel (of side length δ') in the second-level lattice containing p. Let $G_{\widehat{X}}$ be the intersection graph spanned by balls centered points in $\widehat{X}$ but with radius $1-\delta$. First, note that as all points within 1 from cell-boundaries are removed[1], we have that each connected component of $G_{\widehat{X}}$ has to be contained inside some cell C of $\Gamma + \tau$. Hence to compute MIS (maximum independent set) for $G_{\widehat{X}}$, we can do so by computing an optimal MIS for $G_{\widehat{X}_\mathsf{C}}$, the restriction of $G_{\widehat{X}}$ within every cell C of $\Gamma + \tau$, and then the union of them over all cells is necessarily an MIS for $G_{\widehat{X}}$. In (Step 2), we compute $\widehat{Y}_\mathsf{C}$, which

[1] We removed all points in X within distance 1 from the cell boundaries. But since we assume that the $1/\delta'$, the reciprocal of the pixel side-length, is an integer, this statement about points in $\widehat{X}$ still holds.

is an MIS for $G_{\widehat{X}_{\mathsf{C}}}$. If we take the union of this for all cells, namely $\widehat{Y} = \bigcup_{\mathsf{C}} \widehat{Y}_{\mathsf{C}}$, then it is clear that $\widehat{Y}$ is an MIS for $G_{\widehat{X}}$.

Furthermore, for any cell C, as it is a d-dimensional hypercube of side-length $2d/\varepsilon$, we have that its volume is $(2d/\varepsilon)^d$. As any independent set in the cell necessarily has pairwise distance > 2, it follows that the maximum cardinality of any independent set in C is at most $s = (2d/\varepsilon)^d / V_d = V_d^{-1}(2d/\varepsilon)^d$, where V_d stands for the volume of a radiue $(1-\delta)$ ball in $\mathbb{R}^d$. Furthermore, there are only $M := (\frac{2d}{\varepsilon\delta'})^d = (\frac{2d\sqrt{d}}{\varepsilon\delta})^d$ number of pixels inside cell C, we can thus enumerate all possible independent sets for $\widehat{X}_{\mathsf{C}}$ in time $M^s = (\frac{1}{\varepsilon\delta})^{(d/\varepsilon)^{O(d)}}$ time. Since there are at most n cells of $\Gamma + \tau$ that contains non-empty $\widehat{X}_{\mathsf{C}}$, the total time to construct an MIS for $G_{\widehat{X}}$ is thus $(\frac{1}{\varepsilon\delta})^{(d/\varepsilon)^{O(d)}} n$ as claimed in Theorem 5.2.1.

Note that in (Step 2) of Baker's paradigm, after computing $\widehat{Y}_{\mathsf{C}}$, we need to transfer it to a subset $Y_{\mathsf{C}} \subseteq X_{\mathsf{C}} \subseteq X$ of original input points. In particular, we achieve this by mapping each point $\widehat{p} \in \widehat{Y}_{\mathsf{C}}$ to an arbitrary point $p = \pi(\widehat{p}) \in X_{\mathsf{C}}$ contained in the pixel that $\widehat{p}$ is the bottom-left corner of (this is a consequence of the construction of set $\widehat{X}_{\mathsf{C}}$, where we snap each point q in X_{C} to the left-bottom vertex of the pixel q lies in). Obviously, this map $\pi : \widehat{Y}_{\mathsf{C}} \to Y_{\mathsf{C}}$ is a bijection, and the distance $\|\widehat{p} - \pi(\widehat{p})\|_2 \leq \delta$. (Note that the diameter of a pixel in the cell C is δ as the side-length of this pixel is $\delta' = \delta/\sqrt{d}$.)

What remains is to prove that $Y = \bigcup_{\mathsf{C}} Y_{\mathsf{C}}$ as computed in (Step 3) of Baker's paradigm is indeed a bi-criteria approximation in expectation for the MIS of G_X, the unit-ball intersection graph spanned by input points X.

To this end, first, note that $\widehat{Y} = \bigcup_{\mathsf{C}} \widehat{Y}_{\mathsf{C}}$ is a maximum independent set for $G_{\widehat{X}}$ as argued earlier. We claim that $|\widehat{Y}| \geq |Y^* \setminus Y'|$. This is because that since $Y^* \setminus Y'$ is an independent set for the unit-ball graph spanned by points in $X \setminus X'$, we have that for any points $y, y' \in Y^* \setminus Y'$, $\|y - y'\|_2 > 2$. Now map y and y' to $\hat{y}$ and $\hat{y}'$, the respective left-bottom corner of the pixels they are contained in; note that $\hat{y}, \hat{y}' \in \widehat{X}$. By the triangle inequality, we have that $\|\hat{y} - \hat{y}'\|_2 \geq 2 - 2\delta$ as the diameter of each pixel is δ. This means that snapping all points in $Y^* \setminus Y'$ as such to points in $\widehat{X}$ gives rise to an independent set of $G_{\widehat{X}}$. This in turn implies that a maximum independent

set of $G_{\widehat{X}}$, namely $\widehat{Y}$, is at least as large as the set $Y^* \setminus Y'$; that is, $|\widehat{Y}| \geq |Y^* \setminus Y'|$. Combining this with Eqn (C.1), we then have that

$$\mathbf{E}[|Y|] = \mathbf{E}[|\widehat{Y}|] \geq |Y^*| - \mathbf{E}[|Y'|] \geq (1-\varepsilon)|Y^*| = (1-\varepsilon)\mathsf{OPT}(X) \geq \mathsf{OPT}/(1+2\varepsilon), \quad \text{(C.2)}$$

where the last inequality holds for any positive $\varepsilon < 1$. This establishes one side (lower-bound side) of the bi-criteria approximation.

We now consider the upper-bound in the bi-criteria approximation. Note that we have a bijection $\pi : \widehat{Y} \to Y$ which sends a point $\widehat{p} \in \widehat{Y}$ to a point $\pi(p)$ within δ distance. Combining this with the fact that $\widehat{Y}$ itself is an independent set for $G_{\widehat{X}}$ (i.e, any two points inside are at least distance $2-2\delta$ apart), we have that Y is an $(1+6\delta)$-independent set: This is because any two points in Y are at least $2-4\delta \geq 2/(1+6\delta)$ apart, as $1-2\delta \geq 1/(1+6\delta)$ holds for any positive $\delta < 1/3$. It then follows that $|Y| \leq \mathsf{OPT}_{1+\Theta(\delta)}(X)$.

Putting both sides (upper and lower bounds) together, we have that the set Y computed by our proposed Baker's paradigm is a $(1+\Theta(\varepsilon), 1+\Theta(\delta))$-bicriteria approximation of d-MIS for input point set X in expectation. Together with the time complexity bound computed earlier, this concludes the proof of Theorem 5.2.1.

C.1.2 Removing the bi-criteria condition and Theorem 5.2.2

We note that one can easily modify our algorithm Baker-MIS to obtain a $(1+\Theta(\varepsilon))$-approximation for MIS (instead of a bi-criteria approximation), by trading off a slower running time, similarly to [HM85]. We include the details here for completeness. The algorithm proceeds exactly as Baker-MIS, with the only difference being that Step 2 is replaced by the following:

Step 2": Solving the problem exactly locally on each cell. For each cell C of $\Gamma + \tau$, let X_{C} be the restriction of $X \setminus X'$ to cell C. Now in this modified step $2''$, we will work with X_{C} instead of working with the set $\widehat{X}_{\mathsf{C}}$, which is the snapping of set X_{C} to pixels in the cell. Let $G_{X_{\mathsf{C}}}$ be the intersection graph of unit balls centered at the points in X_{C}; that is

$V(G_{X_\mathsf{C}}) = X_\mathsf{C}$, and

$$E(G_{X_\mathsf{C}}) = \left\{ \{p,q\} \in \binom{X_\mathsf{C}}{2} : \|p-q\|_2 \leq 2 \right\}.$$

We compute the maximum independent set Y_C in G_{X_C} which we know has size at most $s = V_d^{-1}(2d/\varepsilon)^d$, where V_d denotes the volume of the d-dimensional unit ball. This can be done by enumerating all possible subsets of $X_\mathsf{C} = V(G_{X_\mathsf{C}})$ of size at most s, and taking the maximum cardinality such subset that is independent in G_C.

In (Step 3), we will return $Y = \bigcup_\mathsf{C} Y_\mathsf{C}$ as before. As seen in Theorem 2.2, the price to pay to obtain a standard $(1+\varepsilon)$-approximation is that the dependency of time complexity on n increases from previous n (i.e, linear) to $n^{(1/\varepsilon)^{O(d)}}$. This is because during the exhaustive enumeration to solve MIS for G_{X_C}, we have to take all subsets of X_C of size at most s. Since the cardinality of X_C could be n (say when all points in X happen to be inside a single cell C of the randomly shifted grid $\Gamma + \tau$), we thus needs $n^{(1/\varepsilon)^{O(d)}}$ time for this enumeration. The approximation guarantee follows the proof of Theorem 5.2.1, but as Y constructed is now a valid independent set for X, we do not have the relaxation of $(1+\Theta(\delta))$-independent set. Theorem 5.2.2 thus follows.

C.1.3 Proof of Theorem 5.3.2

By Theorem 5.3.1 stated in the main text, we can obtain a neural network $\mathscr{N}^*$ with a single hidden layer that computes a function $g_{\mathscr{N}^*} : [0,1]^k \to [0,1]^k$, such that

$$\sup_{x \in [0,1]^k} |g_{\mathscr{N}^*}(x) - f_{\text{MIS}}(x)| < 1/2,$$

where the hidden layer has size $N = N(\varepsilon, \delta, d)$. By rounding the output of $g_{\mathscr{N}^*}$, we obtain the indicator vector of a maximum-independent set $\widehat{Y}_\mathsf{C}$ for $\widehat{X}_\mathsf{C}$. The same holds for the greedy strategy to choose an output as described in (Step $2'$) of NN-Baker. The proof of Theorme 5.2.1 states

that Eqn (C.2) holds for $\widehat{Y} = \bigcup_{\mathsf{C}} \widehat{Y}_{\mathsf{C}}$.

Next, we map $\widehat{Y}_{\mathsf{C}}$ to Y_{C} as before by mapping each $\widehat{p} \in \widehat{Y}_{\mathsf{C}}$ to $\pi(p)$ on X_{C} within δ distance to $\widehat{p}$. Following the same argument as in the proof of Theorem 5.3.2, we know that the resulting $Y = \bigcup_{\mathsf{C}} Y_{\mathsf{C}}$ is a $(1+\Theta(\varepsilon), 1+\Theta(\delta))$-bicriteria approximation in expectation of d-MIS for the input points X. Finally, since there will be at most n cells containing at least one point from X, we know that we only need to call this neural network $\mathscr{N}^*$ at most n times. This completes the proof of Theorem 5.3.2.

C.2 More details for experiments

Hardware information

All baselines and NN-Baker models are trained and test on an AMD-EPYC-7452 CPU and a RTX-A6000 GPU.

Additional dataset information

2D-Gaussian dataset is a collection of geometric graphs generated from 40k - 50k points sampled from a 2D mixture of 5 Gaussian distribution. The centers of this 5 Gaussian distribution are [(64, 64), (32, 32), (32, 96), (96, 32), (96, 96)] and their standard deviance is 20.0. The input domain is partitioned into cells of side-length 12.8, and each cell is further partitioned into 128×128 pixels. 3D dataset is a collection of geometric graphs generated by points uniformly sampled from a 3D cube region with around 40k - 50k points. Each cell in the domain has side-length 5, and is partitioned into $50 \times 50 \times 50$ pixels. Torus-4D dataset is a collection of geometric graphs generated by points sampled from a 4D surface with around 40k - 50k points. The 4D surface is generated by two functions $f, g : [0,1]^2 \to \mathbb{R}^4$ that:

$$\begin{aligned} f(\alpha) = &\quad (r\sin\alpha, r\cos\alpha, 0, 0) \\ g(\beta) = &\quad (0, 0, r\sin\beta, r\cos\beta) \end{aligned} \tag{C.3}$$

where $r > 0$ is a constant, and we set $r = 20$ in experiments. Given a point set X uniformly sampled from $[0,1]^2$, we have a resulting point set $(f+g)(X) \subset \mathbb{R}^4$. Each cell in the 4D surface is mapped from a cell with side-length 0.1 in $[0,1]^2$ which is partitioned into 100×100 pixels.

More experimental statistics

As a baseline comparison, we compared our CNN and GNN approaches to a standard feed forward NN. Both models were trained on the image segmentation problem of converting 128x128 images with the points as inputs to 128x128 images of points which should be in the independent set. This was combined with the Baker technique to produce the figures in Table C.1. For the "Small" neural network, this model contained two hidden layers and a total number of parameters equal to our UNet-Baker approach (76 million). The "Large" model also contains two hidden layers and a total number of parameters roughly equal to double that value (147 million). For this data, models were trained and tested on data from the same distribution.

Table C.1. Performance on MIS by fully connected models

	Small	Large
2D-dense	0.714	0.788
2D-sparse	0.789	0.898
2DGaussian	0.724	0.852

To report the variance of the architectures proposed, we trained ten models for each architecture from different random start weights. We report the standard deviations ($\times 10^{-3}$) of the ratios of MIS results from our models to ground truth in Table C.2.

Table C.2. The standard deviations of MIS results from different models ($\times 10^{-3}$).

	UNetBaker	Erdős	ErdősBaker	TGS	TGSBaker	LwD	LwDBaker
2D-dense	3.06	2.70	5.83	1.15	3.72	3.58	6.75
2D-sparse	2.93	2.73	8.25	1.12	3.49	3.61	8.53
2DGaussian	3.10	10.62	14.36	2.35	4.28	8.57	7.82
3D	-	3.53	5.85	1.82	2.95	5.42	8.63
Torus-4D	-	5.64	5.24	2.10	3.87	5.35	8.30

Post-processing is an important part of our implementations. After each of the cells are solved by the NN-Baker framework, we then add in points close to the boundaries that do not intersect any of the points already in the set. The percentage of points added for all methods is given in Table C.3.

Table C.3. Proportion of points added in post processing

	UNetBaker	ErdősBaker	TGSBaker	LwDBaker
2D-dense	0.085	0.080	0.025	0.032
2D-sparse	0.086	0.043	0.005	0.006
2DGaussian	0.082	0.042	0.025	0.016

Appendix D

Chapter 6: Appendix

D.1 Proof of Theorem 6.2.3

Theorem D.1.1. Structure Theorem. *If the size of the bounding box is L, let shifts* $0 \leq a,b < L$ *be picked randomly, then with probability at least* $1/2$*, the dissection with shift* (a,b) *has an associated Steiner tree of cost at most* $(1+\frac{8}{r}+O(\frac{4logL}{m}))-OPT$ *that is* $(m,r)-light.$

Proof.

Lemma D.1.2 (Patching Lemma)**.** *Let* ℓ *be any line segment of length s and* T *be a Steiner tree. The segment* ℓ *could be crossed by* T *an arbitrary number of times. We can easily modify* T *to a new Steiner tree that crosses the segment* ℓ *at most once while increasing the cost of the tree by at most s.*

Proof of Patching Lemma. Without loss of generality, assume the segment ℓ in Lemma A.1 is vertical. Then, the present lemma is shown by removing crossings one by one in a top-down manner, as follows. Let $z_1,\ldots,z_k$ be the set of intersection points between T and the line segment ℓ, sorted by decreasing y-coordinate. Consider z_1: Imagine "cutting" the tree T at z_1, which will break T into two connected components; one containing a "left copy" z_1^- of z_1 and the other containing a "right copy" z_1^+ of z_1. One of these two components, say the one connecting z_1^-, is disconnected to the components containing z_2: we thus simply connect z_1^- to the "left copy" z_2^- by a vertical edge. The resulting tree $T^{(1)}$ is still a valid Steiner tree. We

repeat this until we finish processing all crossing points other than the last one, z_k. Overall, the total length of extra (vertical) edges we add is at most s. In the end only one crossing point (i.e, z_k) remains.

Lemma D.1.3. *Let us grid the bounding box by putting a sequence of vertical and horizontal lines at unit distance from one another. If l is one of these lines and T is a Steiner tree, denote the number of times that T crosses l as $t(T,l)$.*

$$\sum_{l:vertical} t(T,l) + \sum_{l:horizontal} t(T,l) \leq 2\text{cost}(T) \tag{D.1}$$

This lemma is obvious as an edge of T that has length s contributes at most $O(s)$ to the left hand side.

Let T be the optimum Steiner tree and suppose (a,b) is picked randomly. Whenever the Steiner tree has intersections with a square side 'too many' times, we use Patching Lemma to reduce the number of crossings. This increases cost, and we upperbound it as as follows.

Suppose l has level i. It is touched by 2^{i+1} level $i+1$ squares, which partition it into 2^{i+1} segments of length $L/2^{i+1}$. For each $j > i$, line l is also touched by 2^j level j squares. We refer to the portion of l that lies in a level j square as a level j segment. Our goal is to reduce the number of crossings in each level i segment to r or less.

An overloaded segment of l is one which the Steiner tree crosses at least $r+1$ times. For every segment at level $logL-1$ that is overloaded, we apply Patching Lemma and reduce the crossings to 1. Then we proceed to level $logL-2$ and apply Pathcing Lemma to each overloaded segment. We continue this procedure until no segments are overloaded at level i. In the end, we move all crossings to portals.

Next let's compute the cost increase in this Steiner tree transformation. Imagine a procedure in which the tree transformation on vertical grid line l proceeds to level 0 until the

entire is not overloaded. Let $X_{l,j}(b)$ be a random variable denoting the number of overloaded level j segments encountered in this procedure. Note that $X_{l,j}(b)$ is determined by vertical shift b, which determines the location of the crossings on l. We claim for every b,

$$\sum_{j\geq 0} X_{l,j}(b) \leq \frac{t(T,l)}{r} \tag{D.2}$$

Because the optimum Steiner tree crossed grid line l only $t(T,l)$ times, and each application of Patching Lemma counted on the left hand side of Equation (D.2) replaces at least $r+1$ crossings by 1, thus eliminating r crossings each time.

Since a level j segment has length $L/2^j$, the cost of this transformation procedure is at most

$$\sum_{j\geq 1} X_{l,j}(b)\frac{L}{2^{j-1}} \tag{D.3}$$

by applying Pathcing Lemma. The actual cost increase in the tree transformation at l depends on the level of l, which is determined by horizontal shift a. When the level is i, the cost increase upperbounded by the terms of Equation D.3 corresponding to $j \geq i+1$: $\sum_{j\geq i+1} X_{l,j}(b)\frac{L}{2^{j-1}}$. Line l is at the level i with probability at most $2^{i+1}/L$. Thus we can compute the expectation of $Y_{l,a}$, the cost increase to l when the horizontal shift is a. For every vertical shift b:

$$\begin{aligned} E_a[Y_{l,a}] &\leq \sum_{i\geq 1} \frac{2^{i+1}}{L} \cdot \sum_{j\geq i+1} X_{l,j}(b)\frac{L}{2^{j-1}} \\ &= \sum_{j\geq 1} \frac{X_{l,j}(b)}{2^{j-1}} \sum_{i\leq j-1} 2^i \\ &= \sum_{j\geq 1} \frac{X_{l,j(b)}}{2^{j-1}}(2^j-1) \\ &\leq \sum_{j\geq 1} 2X_{l,j}(b) \leq \frac{2t(T,l)}{r} \end{aligned} \tag{D.4}$$

The expectation of cost increase for all lines is $E_a[\sum_l Y_{l,a}] = \sum_l \frac{2t(T,l)}{r}$. According to Lemma D.1.3, $E_a[\sum_l Y_{l,a}] \leq \frac{4\text{cost}(T)}{r}$.

Finally we need to compute the cost increase in moving crossings to their nearest portals. If line l has maximal level i, the distance between each of the $t(T,l)$ crossings and its nearest portal is at most $\frac{L}{2^{i+1}m}$. Instead of actually moving a crossing to a portal, we break the edge at the crossing and add two line segments (on each side of l) to it. Thus the expected increase for moving every crossing in l to its nearest portals is upper bounded

$$\sum_{i=1}^{logL} \frac{2^i}{L} t(T,l) \cdot \frac{L}{2^{i+1}m} \cdot 2 = \frac{t(T,l)logL}{m} \tag{D.5}$$

According to Lemma D.1.3, the total expected cost increase for all crossings in all lines are upper bounded

$$\sum_{l} \frac{t(T,l)logL}{m} \leq \frac{2logL}{m} \text{cost}(T) \tag{D.6}$$

Denote the Steiner tree obtained from T by $(a,b)-$shift, transformation and moving crossings as $T_{a,b,m,r}$, we have

$$E[\text{cost}(T_{a,b,m,r})] \leq (1 + \frac{4}{r} + \frac{2logL}{m})\text{cost}(T) \tag{D.7}$$

According to Markov's inequality, with probability at least $1/2$, the cost of the best $(m,r)-$light Steiner tree for the shifted dissection is at most $(1 + \frac{8}{r} + O(\frac{4logL}{m}))-$OPT.

□

D.2 More details for experiments

Points generation

Point set with different sizes are generated by Algorithm 3.

Algorithm 3. Point Generation

Input:

(Low, High): The range of number of points in one point set;
Width: Sample points from space $\{1,2,...,\text{Width}\}^2$;
Num: The number of point sets
σ: Parameter on points distribution

Output:

L: List of point sets
$L = []$
for $k = 1$ to Num **do**
  Randomly sample $n \in [\text{Low}, \text{High}]$
  for $i = 1$ to n **do**
    $S = \emptyset$
    Randomly pick $p \in \{0,1\}$
    if $p = 0$ **then**
      Sample (x,y) from $\{1,2,...,\text{Width}\}^2$ under uniform distribution
      Add (x,y) to S
    else
      Sample (x,y) from $[1,\text{Width}]^2$ under Gaussian distribution $\mathcal{N}((\text{M},\text{M}),\sigma^2 I)$ where $\text{M} = \lfloor \text{Width}/2 \rfloor$ and I is the identity matrix
      Take $(\bar{x},\bar{y})$ as the integer point nearest to (x,y)
      Add $(\bar{x},\bar{y})$ to S
    end if
  end for
  Append S to L
end for

Bibliography

[ACH+18] Charles J Alpert, Wing-Kai Chow, Kwangsoo Han, Andrew B Kahng, Zhuo Li, Derong Liu, and Sriram Venkatesh. Prim-dijkstra revisited: Achieving superior timing-driven routing trees. In *Proceedings of the 2018 International Symposium on Physical Design*, pages 10–17, 2018.

[ADH07] Giorgio A Ascoli, Duncan E Donohue, and Maryam Halavi. Neuromorpho. org: a central resource for neuronal morphologies. *Journal of Neuroscience*, 27(35):9247–9251, 2007.

[AEK+17] Henry Adams, Tegan Emerson, Michael Kirby, Rachel Neville, Chris Peterson, Patrick Shipman, Sofya Chepushtanova, Eric Hanson, Francis Motta, and Lori Ziegelmeier. Persistence images: a stable vector representation of persistent homology. *Journal of Machine Learning Research*, 18:218–252, 2017.

[ÅFP+09] Matti Åstrand, Patrik Floréen, Valentin Polishchuk, Joel Rybicki, Jukka Suomela, and Jara Uitto. A local 2-approximation algorithm for the vertex cover problem. In *International symposium on distributed computing*, pages 191–205. Springer, 2009.

[Ang80] Dana Angluin. Local and global properties in networks of processors. In *Proceedings of the twelfth annual ACM symposium on Theory of computing*, pages 82–93, 1980.

[Aro98] Sanjeev Arora. Polynomial time approximation schemes for euclidean traveling salesman and other geometric problems. *Journal of the ACM (JACM)*, 45(5):753–782, 1998.

[Aro03] Sanjeev Arora. Approximation schemes for np-hard geometric optimization problems: A survey. *Mathematical Programming*, 97(1):43–69, 2003.

[ASS20] Sungsoo Ahn, Younggyo Seo, and Jinwoo Shin. Learning what to defer for maximum independent sets. In *International Conference on Machine Learning*, pages 134–144. PMLR, 2020.

[Bak94] Brenda S Baker. Approximation algorithms for np-complete problems on planar graphs. *Journal of the ACM (JACM)*, 41(1):153–180, 1994.

[Bau16] Ulrich Bauer. Ripser. https://github.com/Ripser/ripser, 2016.

[BCFL20] Thomas Barrett, William Clements, Jakob Foerster, and Alex Lvovsky. Exploratory combinatorial optimization with reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 3243–3250, 2020.

[BCNK18] Sumit Bhatia, Bapi Chatterjee, Deepak Nathani, and Manohar Kaul. Understanding and predicting links in graphs: A persistent homology perspective. *arXiv preprint arXiv:1811.04049*, 2018.

[BFK+94] Piotr Berman, Ulrich Fößmeier, Marek Karpinski, Michael Kaufmann, and Alexander Zelikovsky. Approaching the 5/4—approximation for rectilinear steiner trees. In *European Symposium on Algorithms*, pages 60–71. Springer, 1994.

[BKRW14] Ulrich Bauer, Michael Kerber, Jan Reininghaus, and Hubert Wagner. Phat – persistent homology algorithms toolbox. In Hoon Hong and Chee Yap, editors, *Mathematical Software – ICMS 2014*, pages 137–143, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[BLP20] Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine learning for combinatorial optimization: a methodological tour d'horizon. *European Journal of Operational Research*, 2020.

[BOS+05] Karsten M Borgwardt, Cheng Soon Ong, Stefan Schönauer, SVN Vishwanathan, Alex J Smola, and Hans-Peter Kriegel. Protein function prediction via graph kernels. *Bioinformatics*, 21(suppl_1):i47–i56, 2005.

[BPL+17] Irwan Bello, Hieu Pham, Quoc V. Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning. *5th International conference on Learning Representations Workshop Track*, 2017.

[BRTH15] Lu Bai, Luca Rossi, Andrea Torsello, and Edwin R. Hancock. A quantum jensen-shannon graph kernel for unattributed graphs. *Pattern Recognition*, 48(2):344–355, 2015.

[Bub15] Peter Bubenik. Statistical topological data analysis using persistence landscapes. *Journal of Machine Learning Research*, 16(1):77–102, 2015.

[BZSL14] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. *International Conference on Learning Representations*, 2014.

[CBGY14] Maria Clément, Jean-Daniel Boissonnat, Marc Glisse, and Mariette Yvinec. The gudhi library: simplicial complexes and persistent homology. http://gudhi.gforge.inria.fr/python/latest/index.html, 2014.

[CCI+19] Mathieu Carriere, Frédéric Chazal, Yuichi Ike, Théo Lacombe, Martin Royer, and Yuhei Umeda. A general neural network architecture for persistence diagrams and graph classification. *arXiv preprint arXiv:1904.09378*, 2019.

[CCO17] Mathieu Carrière, Marco Cuturi, and Steve Oudot. Sliced Wasserstein kernel for persistence diagrams. *International Conference on Machine Learning*, pages 664–673, 2017.

[CCSG+09] Frédéric Chazal, David Cohen-Steiner, Marc Glisse, Leonidas J. Guibas, and Steve Oudot. Proximity of persistence modules and their diagrams. In *Proc. 25th ACM Sympos. on Comput. Geom.*, pages 237–246, 2009.

[CdS10] Gunnar Carlsson and Vin de Silva. Zigzag persistence. *Foundations of Computational Mathematics*, 10(4):367–405, 2010.

[CdSGO16] Frédéric Chazal, Vin de Silva, Marc Glisse, and Steve Oudot. *The structure and stability of persistence modules*. SpringerBriefs in Mathematics. Springer, 2016.

[CdSM09] Gunnar Carlsson, Vin de Silva, and Dmitriy Morozov. Zigzag persistent homology and real-valued functions. In *Proc. 25th Annu. ACM Sympos. Comput. Geom.*, pages 247–256, 2009.

[Cha03] Timothy M Chan. Polynomial-time approximation schemes for packing and piercing fat objects. *Journal of Algorithms*, 46(2):178–189, 2003.

[CHP12] Timothy M Chan and Sariel Har-Peled. Approximation algorithms for maximum independent set of pseudo-disks. *Discrete & Computational Geometry*, 48(2):373–392, 2012.

[CKL+22] Po-Yan Chen, Bing-Ting Ke, Tai-Cheng Lee, I-Ching Tsai, Tai-Wei Kung, Li-Yi Lin, En-Cheng Liu, Yun-Chih Chang, Yih-Lang Li, and Mango C-T Chao. A reinforcement learning agent for obstacle-avoiding rectilinear steiner tree construction. In *Proceedings of the 2022 International Symposium on Physical Design*, pages 107–115, 2022.

[CNBW19] Chao Chen, Xiuyan Ni, Qinxun Bai, and Yusu Wang. A topological regularizer for classifiers via persistent homology. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 2573–2582, 2019.

[CSEH07] David Cohen-Steiner, Herbert Edelsbrunner, and John Harer. Stability of persistence diagrams. *Discrete & Computational Geometry*, 37(1):103–120, 2007.

[CSEH09] David Cohen-Steiner, Herbert Edelsbrunner, and John Harer. Extending persistence using Poincaré and Lefschetz duality. *Foundations of Computational Mathematics*, 9(1):79–103, 2009.

[CSEHM10] David Cohen-Steiner, Herbert Edelsbrunner, John Harer, and Yuriy Mileyko. Lipschitz functions have Lp-stable persistence. *Foundations of computational mathematics*, 10(2):127–139, 2010.

[CW07] Chris Chu and Yiu-Chung Wong. Flute: Fast lookup table based rectilinear steiner minimal tree algorithm for vlsi design. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(1):70–83, 2007.

[Cyb89] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[CZ09] G. Carlsson and A. Zomorodian. The theory of multidimensional persistence. *Discrete & Computational Geometry*, 42(1):71–93, 2009.

[DBV16] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in neural information processing systems*, pages 3844–3852, 2016.

[DCL+18] Michel Deudon, Pierre Cournut, Alexandre Lacoste, Yossiri Adulyasak, and Louis-Martin Rousseau. Learning heuristics for the tsp by policy gradient. In *International conference on the integration of constraint programming, artificial intelligence, and operations research*, pages 170–181. Springer, 2018.

[DD03] Paul D Dobson and Andrew J Doig. Distinguishing enzyme structures from non-enzymes without alignments. *Journal of molecular biology*, 330(4):771–783, 2003.

[DKZ+17] Hanjun Dai, Elias B Khalil, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 6351–6361, 2017.

[DLdCD+91] Asim Kumar Debnath, Rosa L Lopez de Compadre, Gargi Debnath, Alan J Shusterman, and Corwin Hansch. Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity. *Journal of medicinal chemistry*, 34(2):786–797, 1991.

[DSW16] Tamal K. Dey, Dayu Shi, and Yusu Wang. Simba: An efficient tool for approximating Rips-filtration persistence via simplicial batch-collapse. In *24th Annual European Symposium on Algorithms (ESA 2016)*, volume 57 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 35:1–35:16, 2016.

[DW22] Tamal Krishna Dey and Yusu Wang. *Computational Topology for Data Analysis*. Cambridge University Press, 2022.

[EH10] Herbert Edelsbrunner and John Harer. *Computational Topology : an Introduction*. American Mathematical Society, 2010.

[EJS05] Thomas Erlebach, Klaus Jansen, and Eike Seidel. Polynomial-time approximation schemes for geometric intersection graphs. *SIAM Journal on Computing*, 34(6):1302–1323, 2005.

[ELZ02] H. Edelsbrunner, D. Letscher, and A. Zomorodian. Topological persistence and simplification. *Discrete Comput. Geom.*, 28:511–533, 2002.

[FPT81] Robert J Fowler, Michael S Paterson, and Steven L Tanimoto. Optimal packing and covering in the plane are np-complete. *Information processing letters*, 12(3):133–137, 1981.

[FWDT20] Aaron Ferber, Bryan Wilder, Bistra Dilkina, and Milind Tambe. Mipaal: Mixed integer program as a layer. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 1504–1511, 2020.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[GCF+19] Maxime Gasse, Didier Chételat, Nicola Ferroni, Laurent Charlin, and Andrea Lodi. Exact combinatorial optimization with graph convolutional neural networks. In *Advances in Neural Information Processing Systems 32*, 2019.

[GGK+20] Prateek Gupta, Maxime Gasse, Elias Khalil, Pawan Mudigonda, Andrea Lodi, and Yoshua Bengio. Hybrid models for learning to branch. *Advances in Neural Information Processing Systems*, 33, 2020.

[Gir15] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

[GJ19] Hongyang Gao and Shuiwang Ji. Graph u-nets. In *International Conference on Machine Learning*, pages 2083–2092, 2019.

[GL16] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864, 2016.

[GMS05] Marco Gori, Gabriele Monfardini, and Franco Scarselli. A new model for learning in graph domains. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 2, pages 729–734. IEEE, 2005.

[GSR+17] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1263–1272. JMLR. org, 2017.

[Hås99] Johan Håstad. Clique is hard to approximate within $n^{1-\varepsilon}$. *Acta Mathematica*, 182(1):105–142, 1999.

[HK09] Shohei Hido and Hisashi Kashima. A linear-time graph kernel. In *Data Mining, 2009. ICDM'09. Ninth IEEE International Conference on*, pages 179–188. IEEE, 2009.

[HKKS01] Christoph Helma, Ross D. King, Stefan Kramer, and Ashwin Srinivasan. The predictive toxicology challenge 2000–2001. *Bioinformatics*, 17(1):107–108, 2001.

[HKNU17] Christoph Hofer, Roland Kwitt, Marc Niethammer, and Andreas Uhl. Deep learning with topological signatures. In *Advances in Neural Information Processing Systems*, pages 1634–1644, 2017.

[HLL+19] Jingjia Huang, Zhangheng Li, Nannan Li, Shan Liu, and Ge Li. Attpool: Towards hierarchical feature representation in graph convolutional networks via attention mechanism. In *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, 2019.

[HLSC19] Xiaoling Hu, Fuxin Li, Dimitris Samaras, and Chao Chen. Topology-preserving deep image segmentation. In *Advances in Neural Information Processing Systems*, pages 5658–5669, 2019.

[HLvdMW17] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q Weinberger. Densenet: densely connected convolutional networks. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 30, pages 82–84, 2017.

[HM85] Dorit S Hochbaum and Wolfgang Maass. Approximation schemes for covering and packing problems in image processing and vlsi. *Journal of the ACM (JACM)*, 32(1):130–136, 1985.

[Hop82] John J Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8):2554–2558, 1982.

[HT85] John J Hopfield and David W Tank. "neural" computation of decisions in optimization problems. *Biological cybernetics*, 52(3):141–152, 1985.

[Hwa76] Frank K Hwang. On steiner minimal trees with rectilinear distance. *SIAM journal on Applied Mathematics*, 30(1):104–114, 1976.

[HYL17] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pages 1024–1034, 2017.

[HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[JSL+17] Melvin Johnson, Mike Schuster, Quoc V Le, Maxim Krikun, Yonghui Wu, Zhifeng Chen, Nikhil Thorat, Fernanda Viégas, Martin Wattenberg, and Greg Corrado. Google's multilingual neural machine translation system: Enabling zero-shot translation. *Transactions of the Association for Computational Linguistics*, 5:339–351, 2017.

[JWWZ18] Daniel Juhl, David M Warme, Pawel Winter, and Martin Zachariasen. The geosteiner software package for computing steiner trees in the plane: an updated computational study. *Mathematical Programming Computation*, 10(4):487–532, 2018.

[KCS11] Effrosini Kokiopoulou, Jie Chen, and Yousef Saad. Trace optimization and eigenproblems in dimension reduction methods. *Numerical Linear Algebra with Applications*, 18(3):565–602, 2011.

[KFH18] Genki Kusano, Kenji Fukumizu, and Yasuaki Hiraoka. Kernel method for persistence diagrams via kernel embedding and weight factor. *Journal of Machine Learning Research*, 18(189):1–41, 2018.

[KGG20] Johannes Klicpera, Janek Groß, and Stephan Günnemann. Directional message passing for molecular graphs. *arXiv preprint arXiv:2003.03123*, 2020.

[KGW16] Nils M. Kriege, Pierre Louis Giscard, and Richard C. Wilson. On valid optimal assignment kernels and applications to graph classification. In *Advances in Neural Information Processing Systems*, pages 1623–1631, 2016.

[KKG20] Anna R Karlin, Nathan Klein, and Shayan Oveis Gharan. A (slightly) improved approximation algorithm for metric tsp. *arXiv preprint arXiv:2007.01409*, 2020.

[KL20] Nikolaos Karalias and Andreas Loukas. Erdos goes neural: an unsupervised learning framework for combinatorial optimization on graphs. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, 2020.

[KLBS+16] Elias Khalil, Pierre Le Bodic, Le Song, George Nemhauser, and Bistra Dilkina. Learning to branch in mixed integer programming. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016.

[KMN17] Michael Kerber, Dmitriy Morozov, and Arnur Nigmetov. Geometry helps to compare persistence diagrams. *J. Exp. Algorithmics*, 22:1.4:1–1.4:20, September 2017.

[KMN18] Michael Kerber, Dmitriy Morozov, and Arnur Nigmetov. HERA: software to compute distances for persistence diagrams. https://bitbucket.org/grey_narn/hera, 2018.

[KR92] Andrew B Kahng and Gabriel Robins. A new class of iterative steiner tree heuristics with good performance. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 11(7):893–902, 1992.

[KS17] Michael Kerber and Hannah Schreiber. Barcodes of towers and a streaming algorithm for persistent homology. In *33rd International Symposium on Computational Geometry (SoCG 2017)*, page 57. Schloss Dagstuhl-Leibniz-Zentrum für Informatik GmbH, 2017.

[KSP+18] Risi Kondor, Hy Truong Son, Horace Pan, Brandon M. Anderson, and Shubhendu Trivedi. Covariant compositional networks for learning graphs. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Workshop Track Proceedings*, 2018.

[KT19] Jacob Devlin Ming-Wei Chang Kenton and Lee Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of NAACL-HLT*, pages 4171–4186, 2019.

[KW17] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *International Conference on Learning Representations*, 2017.

[KWPB12] S. P. Koenig, L. D. Wang, J. Pellegrino, and J. S. Bunch. Selective molecular sieving through porous graphene. *Nat. Nano.*, 7(11), 2012.

[LCK18] Zhuwen Li, Qifeng Chen, and Vladlen Koltun. Combinatorial optimization with graph convolutional networks and guided tree search. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 537–546, 2018.

[LCY21] Jinwei Liu, Gengjie Chen, and Evangeline FY Young. Rest: Constructing rectilinear steiner minimum tree via reinforcement learning. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pages 1135–1140. IEEE, 2021.

[LFXP19] Yongcheng Liu, Bin Fan, Shiming Xiang, and Chunhong Pan. Relation-shape convolutional neural network for point cloud analysis. In *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, 2019.

[LHW18] Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *AAAI Conf. Artif.*, 2018.

[Lin92] Nathan Linial. Locality in distributed graph algorithms. *SIAM Journal on computing*, 21(1):193–201, 1992.

[LLY11] Yong Lin, Linyuan Lu, and Shing-Tung Yau. Ricci curvature of graphs. *Tohoku Mathematical Journal, Second Series*, 63(4):605–627, 2011.

[LMBB19] Ron Levie, Federico Monti, Xavier Bresson, and Michael M. Bronstein. Cayleynets: Graph convolutional neural networks with complex rational spectral filters. *IEEE Trans. Signal Processing*, 67(1):97–109, 2019.

[Lou20] Andreas Loukas. What graph neural networks cannot learn: depth vs width. In *International Conference on Learning Representations*, number CONF, 2020.

[LSD15] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

[LSS+17] Sebastian Lamm, Peter Sanders, Christian Schulz, Darren Strash, and Renato F. Werneck. Finding near-optimal independent sets at scale. *J. Heuristics*, 23(4):207–229, 2017.

[LWA+17] Yanjie Li, Dingkang Wang, Giorgio A Ascoli, Partha Mitra, and Yusu Wang. Metrics for comparing neuronal tree shapes based on persistent homology. *PloS one*, 12(8):e0182184, 2017.

[LY18] Tam Le and Makoto Yamada. Persistence Fisher kernel: A Riemannian manifold kernel for persistence diagrams. In *Advances in Neural Information Processing Systems (NIPS)*, pages 10028–10039, 2018.

[MB17] Siyuan Ma and Mikhail Belkin. Diving into the shallows: a computational perspective on large-scale shallow learning. In *Advances in Neural Information Processing Systems*, pages 3778–3787, 2017.

[MBBV15] Jonathan Masci, Davide Boscaini, Michael Bronstein, and Pierre Vandergheynst. Geodesic convolutional neural networks on riemannian manifolds. In *Proceedings of the IEEE international conference on computer vision workshops*, pages 37–45, 2015.

[MBHSL18] Haggai Maron, Heli Ben-Hamu, Nadav Shamir, and Yaron Lipman. Invariant and equivariant graph networks. In *International Conference on Learning Representations*, 2018.

[MBHSL19] Haggai Maron, Heli Ben-Hamu, Hadar Serviansky, and Yaron Lipman. Provably powerful graph networks. *Advances in neural information processing systems*, 32, 2019.

[MBM+17] Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodola, Jan Svoboda, and Michael M Bronstein. Geometric deep learning on graphs and manifolds using mixture model cnns. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5115–5124, 2017.

[MMR+15] Henry Markram, Eilif Muller, Srikanth Ramaswamy, Michael W Reimann, Marwan Abdellah, Carlos Aguado Sanchez, Anastasia Ailamaki, Lidia Alonso-Nanclares, Nicolas Antille, and Selim Arsever. Reconstruction and simulation of neocortical microcircuitry. *Cell*, 163(2):456–492, 2015.

[MRF+19] Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4602–4609, 2019.

[MRM+12] Céline Merlet, Benjamin Rotenberg, Paul A. Madden, Pierre-Louis Taberna, Patrice Simon, Yury Gogotsi, and Mathieu Salanne. On the molecular origin of supercapacitance in nanoporous carbon electrodes. *Nat. Mater.*, 1(4), 2012.

[MZSW21] Evan McCarty, Qi Zhao, Anastasios Sidiropoulos, and Yusu Wang. Nn-baker: A neural-network infused algorithmic framework for optimization problems on geometric intersection graphs. *Advances in Neural Information Processing Systems*, 34, 2021.

[NAK16] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning convolutional neural networks for graphs. In *International conference on machine learning*, pages 2014–2023, 2016.

[NBG+20] Vinod Nair, Sergey Bartunov, Felix Gimeno, Ingrid von Glehn, Pawel Lichocki, Ivan Lobov, Brendan O'Donoghue, Nicolas Sonnerat, Christian Tjandraatmadja, and Pengming Wang. Solving mixed integer programs using neural networks. *arXiv preprint arXiv:2012.13349*, 2020.

[NPGK12] Marion Neumann, Novi Patricia, Roman Garnett, and Kristian Kersting. Efficient graph kernels by randomization. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 378–393. Springer, 2012.

[NQWC17] Xiuyan Ni, Novi Quadrianto, Yusu Wang, and Chao Chen. Composing tree graphical models with persistent homology features for clustering mixed-type data. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2622–2631. JMLR. org, 2017.

[NS95] Moni Naor and Larry Stockmeyer. What can be computed locally? *SIAM Journal on Computing*, 24(6):1259–1277, 1995.

[NST+09] Shervashidze Nino, Vishwanathan SVN, Perti Tobias, Mehlhorn Kurt, and Borgwardt Karsten. Efficient graphlet kernels for large graph comparison. *Artificial Intelligence and Statistics*, pages 488–495, 2009.

[PAL+19] Marcelo Prates, Pedro HC Avelar, Henrique Lemos, Luis C Lamb, and Moshe Y Vardi. Learning to solve np-complete problems: A graph neural network for

decision tsp. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4731–4738, 2019.

[PARS14] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710, 2014.

[Pin99] Allan Pinkus. Approximation theory of the mlp model in neural networks. *Acta Numerica*, 8:143–195, 1999.

[PJG+19] C. Pramanik, T. Jamil, J. R. Gissinger, D. Guittet, P. J. Arias-Monje, S. Kumar, and H. Heinz. Polyacrylonitrile interactions with carbon nanotubes in solution: Conformations and binding as a function of solvent, temperature, and concentration. *Adv. Funct. Mater.*, 29, 2019.

[PMB18] Jorge Pérez, Javier Marinković, and Pablo Barceló. On the turing completeness of modern neural network architectures. In *International Conference on Learning Representations*, 2018.

[QSMG17] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, 2017.

[RBB19] Bastian Rieck, Christian Bock, and Karsten Borgwardt. A persistent weisfeiler-lehman procedure for graph classification. *International Conference on Machine Learning*, 2019.

[RDGF16] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[RFB15] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[RHBK15] Jan Reininghaus, Stefan Huber, Ulrich Bauer, and Roland Kwitt. A stable multi-scale kernel for topological machine learning. In *Computer Vision & Pattern Recognition*, pages 4741–4748, 2015.

[SGT+08] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. Computational capabilities of graph neural networks. *IEEE Transactions on Neural Networks*, 20(1):81–102, 2008.

[SGT+09] F Scarselli, M Gori, Ah Chung Tsoi, M Hagenbuchner, and G Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 1(20):61–80, 2009.

[She12] D. Sheehy. Linear-size approximations to the Vietoris-Rips filtration. In *Proc. 28th. Annu. Sympos. Comput. Geom.*, pages 239–248, 2012.

[SLB+18] Daniel Selsam, Matthew Lamm, B Benedikt, Percy Liang, Leonardo de Moura, and David L Dill. Learning a sat solver from single-bit supervision. In *International Conference on Learning Representations*, 2018.

[SMBG18] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation. *arXiv preprint arXiv:1811.05868*, 2018.

[SNB+08] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI magazine*, 29(3):93–93, 2008.

[SSL+11] Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. Weisfeiler-Lehman graph kernels. *Journal of Machine Learning Research*, 12:2539–2561, 2011.

[SVL14] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, 27, 2014.

[SYK19] Ryoma Sato, Makoto Yamada, and Hisashi Kashima. Approximation ratios of graph neural networks for combinatorial problems. *Advances in Neural Information Processing Systems*, 32, 2019.

[TW19] Minghao Tian and Yusu Wang. A limit theorem for the 1 st betti number of layer-1 subgraphs in random graphs. *arXiv preprint arXiv:1911.00585*, 2019.

[VCC+18] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *International Conference on Learning Representations*, 2018.

[VFJ15] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems-Volume 2*, pages 2692–2700, 2015.

[VSP+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[VZ17] Saurabh Verma and Zhi-Li Zhang. Hunt for the unique, stable, sparse and fast feature learning on graphs. *Advances in Neural Information Proceeding Systems*, pages 88–98, 2017.

[WCW+17] Pengxiang Wu, Chao Chen, Yusu Wang, Shaoting Zhang, Changhe Yuan, Zhen Qian, Dimitris Metaxas, and Leon Axel. Optimal topological cycles and their application in cardiac trabeculae restoration. In *International Conference on Information Processing in Medical Imaging*, pages 80–92. Springer, 2017.

[WJQ+17] Fei Wang, Mengqing Jiang, Chen Qian, Shuo Yang, Cheng Li, Honggang Zhang, Xiaogang Wang, and Xiaoou Tang. Residual attention network for image classification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3156–3164, 2017.

[WL68] Boris Weisfeiler and Andrei Leman. The reduction of a graph to canonical form and the algebra which appears therein. *NTI, Series*, 2(9):12–16, 1968.

[XHLJ19] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *International Conference on Learning Representations*, 2019.

[XJWL15] Lixiang Xu, Xie Jin, Xiaofeng Wang, and Bin Luo. A mixed Weisfeiler-Lehman graph kernel. In *International Workshop on Graph-based Representations in Pattern Recognition*, pages 242–251, 2015.

[XLZ+20] Keylu Xu, Jingling Li, Mozhi Zhang, Simon S Du, Ken-ichi Kawarabayashi, and Stefanie Jegelka. What can neural networks reason about? *International Conference on Learning Representations*, 2020.

[XN17] Mingyu Xiao and Hiroshi Nagamochi. Exact algorithms for maximum independent set. *Information and Computation*, 255:126–146, 2017.

[YV15] Pinar Yanardag and SVN Vishwanathan. Deep graph kernels. *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1365–1374, 2015.

[YYM+18] Rex Ying, Jiaxuan You, Christopher Morris, Xiang Ren, William L Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. In *Int. Conf. Mach. Learn. ICML*, 2018.

[ZKR+17] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Ruslan R Salakhutdinov, and Alexander J Smola. Deep sets. In *Advances in neural information processing systems*, pages 3391–3401, 2017.

[ZW19] Qi Zhao and Yusu Wang. Learning metrics for persistence-based summaries and applications for graph classification. In *Advances in Neural Information Processing Systems*, pages 9855–9866, 2019.

[ZWX+18] Zhen Zhang, Mianzhi Wang, Yijian Xiang, Yan Huang, and Arye Nehorai. Retgk: Graph kernels based on return probabilities of random walks. In *Advances in Neural Information Processing Systems*, pages 3968–3978, 2018.

[ZZL15] Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. *Advances in neural information processing systems*, 28, 2015.