Imperial College London

Imperial College of Science,
Technology and Medicine

Department of Computing

Neural function approximation on graphs: shape modelling, graph discrimination & compression

Giorgos Bouritsas

Supervisor: Michael Bronstein
Co-supervisor: Stefanos Zafeiriou

Submitted in partial fulfilment of the requirements for the degree of
Doctor of Philosophy in Computing

London, February 2023

Statement of Originality

I declare that this thesis and all research materials included are a product of my own work, except where explicitly indicated otherwise. All ideas originating from the work of others are appropriately referenced and acknowledged in full, according to standard academic and research practices.

Abstract

Graphs serve as a versatile mathematical abstraction of real-world phenomena in numerous scientific disciplines. This thesis is part of the Geometric Deep Learning subject area, a family of learning paradigms, that capitalise on the increasing volume of non-Euclidean data so as to solve real-world tasks in a data-driven manner. In particular, we focus on the topic of graph function approximation using neural networks, which lies at the heart of many relevant methods.

In the first part of the thesis, we contribute to the understanding and design of Graph Neural Networks (GNNs). Initially, we investigate the problem of learning on signals supported on a fixed graph. We show that treating graph signals as general graph spaces is restrictive and conventional GNNs have limited expressivity. Instead, we expose a more enlightening perspective by drawing parallels between graph signals and signals on Euclidean grids, such as images and audio. Accordingly, we propose a permutation-sensitive GNN based on an operator analogous to shifts in grids and instantiate it on 3D meshes for shape modelling (*Spiral Convolutions*).

Following, we focus on learning on general graph spaces and in particular on functions that are invariant to graph isomorphism. We identify a fundamental trade-off between invariance, expressivity and computational complexity, which we address with a symmetry-breaking mechanism based on substructure encodings (*Graph Substructure Networks*). Substructures are shown to be a powerful tool that provably improves expressivity while controlling computational complexity, and a useful inductive bias in network science and chemistry.

In the second part of the thesis, we discuss the problem of graph compression, where we analyse the information-theoretic principles and the connections with graph generative models. We show that another inevitable trade-off surfaces, now between computational complexity and compression quality, due to graph isomorphism. We propose a substructure-based dictionary coder - *Partition and Code* (PnC) - with theoretical guarantees that can be adapted to different graph distributions by estimating its parameters from observations. Additionally, contrary to the majority of neural compressors, PnC is parameter and sample efficient and is therefore of wide practical relevance. Finally, within this framework, substructures are further illustrated as a decisive archetype for learning problems on graph spaces.

Acknowledgements

The last 4.5 years were nothing short of an intense yet rich and transformative experience. I perceive the PhD as an "adventure" that touches on multiple dimensions of one's life and goes well beyond the margins of scientific research. The course of my personal adventure has been influenced by several people and to them I am indebted.

I am deeply thankful to my supervisor Michael Bronstein and co-supervisor Stefanos Zafeiriou, first and foremost for giving me the opportunity to embark on this journey. Michael gave me access to a wide spectrum of scientific disciplines and a plethora of opportunities to travel, meet new people and collaborate, opportunities that I would barely imagine before starting. At the same time, the freedom I was given to explore and expand my research interests, was crucial towards growing as an independent researcher.

To Stefanos, I owe deep gratitude for mentoring, supporting and looking after me. His encouragement and guidance to pursue my research aspirations, the wake-up calls he gave me when I doubted myself and the decisive solutions he provided were instrumental in the completion of this PhD. In addition, the collaborations he helped me set up with other members of the lab were very important to broaden my research agenda.

I can't stress enough how grateful I am to the senior collaborators and mentors I was lucky to have through these years: to Andreas Loukas, for providing me with the opportunity to do my research visit to EPFL, which deeply influenced me and shaped me as a scientist. He taught me how to be rigorous and meticulous, how to ask the right questions, but also not to be afraid to take risks and pursue curiosity-driven research.

Moreover, to the Structured Intelligence team and especially Charlie Nash and Pete Battaglia who gave me the opportunity to intern with them and obtain this unique experience of doing research at Deepmind. Through this, I got exposed to many new personal challenges and cultivated my interest in a fascinating research area that I was hoping to explore since the early days of my PhD.

I am indebted to Yannis Panagakis, for mentoring me and inspiring me to develop an independent and original way of thinking and a holistic approach to science. Moreover, I thank him for his trust in me and his willingness to explore together new research directions in the future.

I owe a massive thanks to my dear friend and collaborator Fabrizio Frasca. For consistently being my research buddy from our first and exploratory days in London, for "rescuing" me

whenever I was in trouble or I was simply absent-minded, for being supportive during the hard days and sincerely believing in me (and the list goes on). I learnt many lessons from him and he has been a true inspiration. Also, to Nikos Karalias, for being a great co-worker and for all the fun we had while I was at EPFL, where we were lucky enough to work from the office during the pandemic. In addition, I thank both for our wonderful brainstorming sessions, a highlight of my PhD and a constant reminder of the reasons why I love this profession, and for always being available to give me their insights and feedback (this thesis is a particularly good example).

A big shout-out goes to the people I've made friends with and collaborated at Imperial: Grigoris, Stelios (x2), Alexandros, Ntinos, Stathis, Rolandos, Markos, Guada, Josh, Mehdi, Jean, Sergey, and everyone that passed from the legendary office 351 and created a warm and fun atmosphere.

Throughout these years, I have been lucky enough to have many friends in Athens, London and occasionally various places around the world, that filled my daily life with fun and worry-free moments and gave me invaluable advice and support. Giannis P., Anastasis, Kostas, Orestis, Sotiris, Stefanos, Nikolas (x2), Maria, Kostantis, Giannis G., Aggelos, Petros T., Ilias, Petros P., Giorgos V., Giannis D., Alexia, Alexandros and many others: thank you with all my heart.

Fotini, thank you for all your support, patience and understanding, for being a constant companion through a pandemic and across five countries, for the moments of invaluable relaxation and happiness, for everything we experienced together, and generally, for simply being there.

I am deeply grateful to my family, my parents Panagiotis and Georgia and my sister Dimitra for constantly supporting me through this endeavour and giving me the courage to continue, and to my two little nephews Chloe and Hermes for being the main source of joy and positive energy during the last weeks of writing this thesis.

Concluding, doing a PhD in such a fast-paced field during a pandemic was far from an easy task. Nevertheless, reflecting at the end of this journey, I find myself determined to be part of and contribute to the academic environment as much as possible, despite (or in some cases also owing to) the hard moments that sometimes shook my faith. All the people I mentioned above, played a role in this outcome, and for that, I owe them special thanks.

Finally, to you, the reader, I want to seize this opportunity and wish you to enjoy reading this thesis, as much as I did while I was writing it.

Giorgos Bouritsas,

Athens & London, February 2023

Contents

List of Tables

List of Figures

Table 1: Table of notations

Category	Notation	Definition
learning setup	$\mathscr{X}, \mathscr{Y}$	input/output set of a function approx. problem
	$f^* : \mathscr{X} \to \mathscr{Y}$	target (ground truth) function
	$h : \mathscr{X} \to \mathscr{Y}$	hypothesis
	$\mathscr{H} \subseteq \mathscr{Y}^{\mathscr{X}}$	hypothesis class
	p	distribution over $\mathscr{X}$
	$\mathfrak{D} \in \left(\mathscr{X} \times \mathscr{Y}\right)^{\lvert\mathfrak{D}\rvert}$	training set (unsupervised case: $\in \mathscr{X}^{\lvert\mathfrak{D}\rvert}$)
	$\mathscr{A} : \bigcup_{m=1}^{\infty} \left(\mathscr{X} \times \mathscr{Y}\right)^m \to \mathscr{H}$	learning algorithm
	$\mathscr{L}(h, f^*, p),\ \hat{\mathscr{L}}(h, \mathfrak{D})$	true loss and empirical loss functions
structure	$\mathscr{T} \subseteq 2^{\mathcal{X}}$	topology on $\mathcal{X}$
	$d_{\mathcal{X}} : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_{\geq 0}$	metric on $\mathcal{X}$
	$\mathcal{N}(x), \mathcal{N}_\rho(x) \subseteq \mathscr{T}$	neighbourhood of x (arbitrary/ρ-radius)
	$x_1 \simeq x_2$	equivalence relation: $(x_1, x_2) \in \mathcal{R}_{\mathcal{X}} \subseteq \mathcal{X} \times \mathcal{X}$
	$g \in \mathcal{G}$	group element and group notation
	$\mathcal{X}/\mathcal{G}$	quotient of the (left) group action $\cdot : \mathcal{G} \times \mathcal{X} \to \mathcal{X}$
	$\mathrm{Orb}_{\mathcal{X}/\simeq}(x)$ or $\mathrm{Orb}_{\mathcal{X}/\mathcal{G}}(x)$	orbit/equivalence class of x
	$S(n)$	permutation group of n elements
graphs	$\mathfrak{G}$	a space (universe) of graphs
	$G = \left(\mathcal{V}_G, \mathcal{E}_G, \mathbf{u}_{\mathcal{V}_G}, \mathbf{u}_{\mathcal{E}_G}\right)$	graph w/ vertex set, edge set, vertex signal, edge signal
	$\mathbf{A}(G) \in \{0,1\}^{\mathcal{V}_G \times \mathcal{V}_G}$	adjacency matrix of G w/ $\mathbf{A}(G)(i,j) = 1$ iff $(i,j) \in \mathcal{E}_G$
	$d_G : \mathcal{V}_G \times \mathcal{V}_G \to \mathbb{R}_{\geq 0}$	(shortest-path) metric on G
	G, H	common symbols for graphs
	$G \simeq H$	isomorphic graphs
	$\mathfrak{G}/\simeq$	quotient of the isomorphism relation
	$\mathrm{Aut}(G)$	automorphism group of graph G
	P_n, C_n, K_n, S_n	path, cycle, complete and star graph of n vertices
	$H \subseteq G$	H is a subgraph of G
	$\mathcal{S}(G)$	the set of all the subgraphs of G
	$\mathbf{D}(G), \mathbf{L}(G)$	degree and laplacian matrix of G
signals	$u : \mathcal{X} \to \hat{\mathcal{X}}$	discrete or continuous signal
	$\mathcal{U} \subseteq \hat{\mathcal{X}}^{\mathcal{X}}$	function/signal space
	$\mathbf{u} \in \mathbb{R}^{\lvert\mathcal{X}\rvert \times d}$	matrix notation for discrete signal $u : \mathcal{X} \to \mathbb{R}^d$
	$\tau : \mathcal{X} \to \mathcal{X}$	shift function
	$\mathbf{S} \in \mathbb{R}^{\lvert\mathcal{X}\rvert \times \lvert\mathcal{X}\rvert}$	shift operator matrix (discrete signals)
	$\mathscr{F}_{\mathbf{S}}(\mathbf{u})$	Fourier Transform of signal $\mathbf{u}$ using the shift operator $\mathbf{S}$
probability inf. theory	$p_x(x), q_x(x)$	true and modelled p.m.f/p.d.f. of $x \in \mathscr{X}$
	$\mathbb{P}, \mathbb{E}, \mathbb{H}$	probability, expectation and entropy respectively
	$\mathrm{L}(x; h)$ or $\mathrm{L}(x; q)$	description length of code h or implied by distr. q
	$\mathrm{H}(\theta)$	binary entropy of a Bernoulli r.v. with probability θ
	$\mathbb{B}$	alphabet (e.g. binary $\{0,1\}$)
misc.	$f(n) = O\left(g\left(n\right)\right)$	big-O: $\exists c > 0, n_0$, s.t. $\lvert f(n)\rvert \leq cg(n),\ \forall n \geq n_0$
	$f(n) = \Omega\left(g\left(n\right)\right)$	big-Omega: $\exists c > 0, n_0$, s.t. $f(n) \geq cg(n),\ \forall n \geq n_0$
	$f(n) = \Theta\left(g\left(n\right)\right)$	big-Theta: $\exists c_1, c_2 > 0, n_0$, s.t. $c_1 g(n) \leq f(n) \leq c_2 g(n),\ \forall n \geq n_0$
	f, ϕ, ψ	arbitrary function symbols
	$\boldsymbol{\theta}, \boldsymbol{\varphi}$	arbitrary parameter symbols
	$\mathbf{z}$	latent representations
	$[K]$	$\{1, \ldots, \mathrm{K}\}$
	$2^{\mathcal{X}}$	the powerset of the set $\mathcal{X}$
	$\mathbf{1}_{\mathcal{X}'}$	indicator vector/function: $\mathbf{1}_{\mathcal{X}'}(x) = 1$ iff $x \in \mathcal{X}' \subseteq X$
	$\wr x_1, \ldots, x_n \int$	multiset notation

1 Introduction

1.1 The bigger picture: Geometric Deep Learning

"GRAPHS *are omnipresent*". Though this phrase might sound like hyperbole, graphs have a remarkable generality that permits the representation of numerous real-world phenomena. Enlisting the diverse application domains is by now arguably a cliché: From statistical physics, particle physics and chemistry to electrical engineering, civil engineering, social sciences and epidemiology, any system that can be perceived as a collection of entities that interact (e.g. via a physical force or information exchange, causal relationship, or in general if the behaviour/state of an entity affects that of another) or are linked (e.g. due to geometric proximity, or by physical connections such as a cable or a synapse) can be mathematically represented as a graph. At the same time, their manipulation, such as the computation of various properties, poses significant computational challenges and gives rise to unsolved mysteries in theoretical computer science. In this antithesis lies their beauty.

The theoretical and empirical investigation of graphs, both from a purely mathematical standpoint (in the fields of graph theory, group theory, combinatorics, theoretical computer science, network science and others) and from an application-related perspective, is a scientific endeavour with a long history dating back to at least the 18th century and Euler [Euler, 1736]. The first modern attempts to solve real-world graph-related tasks were dominated by handcrafted graph descriptors, such as molecular fingerprints [Cereto-Massagué et al., 2015, Capecchi et al., 2020], filter design for graph signals [Shuman et al., 2013, Ortega et al., 2018] and manually designed random graph models used to describe the behaviour of real-world graphs [Erdös and Rényi, 1959, Gilbert, 1959, Holland et al., 1983, Albert and Barabási, 2002].

Following this progress, the ground was laid for data-driven methods to emerge. As with most application domains of *machine learning (ML)*, the first successful and widely-adopted methods were based on kernel machines and are known as graph kernels (for a comprehensive review see [Kriege et al., 2020, Nikolentzos et al., 2021]). Not so long after, the sweeping wave of *deep learning* (DL) entrained the field of graph ML. Broadly speaking, graphs belong to (or constitute themselves) mathematical spaces of *non-Euclidean geometry*, similarly to a plethora of other mathematical objects of practical interest, such as hypergraphs, sets, manifolds and groups. Parallel research on neural networks designed for these sub-categories gradually converged to related ideas that formed the field that is now widely known as *Geometric Deep Learning* [Bronstein et al., 2017, Battaglia et al., 2018, Bronstein et al., 2021].

The aforementioned field concerns all data-driven algorithms employed to solve tasks where non-Euclidean data are involved, which, in the vast majority of cases, amounts to predicting one or more unknown functions. Formally, in *statistical learning theory* this problem is known as *function approximation* and refers to the process of approximating a function f^*, called the *ground truth* or *target*, from an input set $\mathscr{X}$ to an output set $\mathscr{Y}$ with another function h, called the *hypothesis*. The latter usually belongs to a set of candidates $\mathscr{H}$, called the *model* or the *hypothesis class* and in ML we are interested in methods using only a finite amount of samples (*the training set*) from $\mathscr{X} \times \mathscr{Y}$ in order to select the hypothesis h. The algorithm that selects a hypothesis, i.e. the map from the training data to the hypothesis, is called the *learning algorithm* (see section 2.2 for a more detailed discussion).

When the hypothesis class is parametrised by a neural network, this process is commonly referred to as *neural function approximation*. Contrary to the Euclidean case, modelling, manipulating, approximating and estimating functions on non-Euclidean data, and graphs in particular is still not well understood. This thesis seeks to deepen our understanding of the aforementioned topic. To achieve this, we will first examine the basic characteristics of the problems considered through the lens of the mathematical structure with which the non-Euclidean data of interest are endowed (section 1.2). Based on this, we will subsequently provide a categorisation of this topic's diverse problem landscape (section 1.3) and the fundamental principles that should be followed when designing ML- and neural network-based solutions (section 1.4). Therein, we will also discuss concepts and principles that apply to general ML problems and go beyond the scope of the problems that we consider, but are useful in order to find connections and obtain a complete picture of the field. Then, in chapters 3, 4 and 5, we will present our contributions to

three important problems which belong to the categories of *graph signals* (shape modelling) and *general graph spaces* (graph discrimination and graph compression).

1.2 Endowing the data with structure

In Geometric Machine/Deep Learning, we are especially interested in the study of the *structure* (the features) with which the input $\mathscr{X}$ and output set $\mathscr{Y}$ are equipped and the ways this affects learning. The vast majority of progress in machine learning has been made for Euclidean data, i.e. data residing in Euclidean vector spaces (vector spaces equipped with the inner product operation). Euclidean spaces are equipped with rich structure; in fact, they possess each one of the examples of structure that we enlist below, among others. In addition, many mathematical tools (calculus, linear algebra, etc.) have been developed and are well-understood for Euclidean spaces, while many popular application domains, such as image and audio processing, concern data that reside in Euclidean spaces, which partly explains the speed with which many machine learning achievements have been materialised. On the other hand, it is not uncommon that our data may reside in spaces with weaker structure or in spaces where the Euclidean space axioms are violated (non-Euclidean spaces).

Usually, structure can be directly inferred from the nature of the data, while sometimes assumptions are made as design choices. In both cases, structure gives rise to a restriction of the way we model the unknown task, i.e. to a certain *inductive bias* - see section 1.4.1. In fact, without structure, it is impossible to make assumptions, and in turn without assumptions, learning is impossible. Let us examine a few examples.[1] A set $\mathcal{X}$ may be endowed with:

- *Topology.* Intuitively a topology equips the set with a notion of a neighbourhood, allowing statements such as "is $x_1 \in \mathcal{X}$ close to $x_2 \in \mathcal{X}$?". Strictly speaking, a topology is a family of subsets of $\mathcal{X}$ respecting certain axioms, where each subset containing $x \in \mathcal{X}$ is a neighbourhood of x. This is particularly important for machine learning since it allows us to define *local* functions, i.e. by hypothesising that the target corresponding to each point is a function of its neighbours. A less obvious consequence of topology is that it allows us to define continuous functions, an almost universal assumption when learning on vector

[1] To ease exposition, we opted for a mainly intuitive explanation of the mathematical concepts involved (more analytic explanations can be found in many resources such as [Foldes, 1994, Corry, 2003]) and we will introduce strict definitions wherever necessary.

spaces or manifolds.

- *Metric.* A metric is a distance function $d_{\mathcal{X}} : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_{\geq 0}$ respecting certain axioms. It is a stronger notion of structure, allowing us not only to define neighbourhoods but also a quantitative measure of "closeness" between the points of a set, and to make statements such as "how close is $x_1 \in \mathcal{X}$ to $x_2 \in \mathcal{X}$?". This may give rise to stronger assumptions, e.g. local functions that depend on the distances from the neighbours, or functions that have a certain degree of smoothness (e.g. *Lipschitz* functions $h : \mathcal{X} \to \mathcal{Y}$ for which it holds that $\exists\ \lambda > 0$, such that $d_{\mathcal{X}}(x_1, x_2) \leq \lambda\ d_{\mathcal{Y}}\big(h(x_1), h(x_2)\big),\ \forall x_1, x_2 \in \mathcal{X}$, where $d_{\mathcal{X}}(\cdot,\cdot), d_{\mathcal{Y}}(\cdot,\cdot)$ are the metrics on the input and on the output space respectively).

- *Order.* This allows us to do comparisons between points in the set (partial, when only certain pairs are comparable, or total, when everything is comparable). It gives rise to important assumptions, such as *monotonicity* (e.g. increasing functions: $x_1 \leq x_2 \Rightarrow h(x_1) \leq h(x_2)$), or causality (e.g. $h(x) = h\big(\{x' \in \mathcal{X} \mid x' \leq x\}\big)$). Particular examples of interest are natural language, where words in a document are totally ordered, and trees, where vertices are partially ordered, giving rise to a *hierarchy*.

- *Equivalence relations.* Intuitively this allows us to define *equivalent* points in the set based on a certain property. They may be defined by transformations[2] on the points of the set (e.g. the equivalence relation that is induced by rotations on a vector space, i.e. if a vector is a rotation of another then they are equivalent); see section 2.3 for a strict definition. They give rise to *symmetries* and allow us to define functions that are *invariant* (equivalent points have the same output) or *equivariant* (a transformation applied to an input point will result in an equally transformed output) - see definition 1.1.

- *Arithmetic operations and identities*, such as multiplication and addition, and associativity and commutativity respectively, which, together with $\mathcal{X}$ constitute an *algebraic structure*. Operations and identities are necessary in order to define vector spaces and are the basis of linear algebra on which neural networks mostly rely.

Note that some of the above examples may give rise to others (e.g. a topology is induced from a metric) and a set may be endowed with a combination thereof. We can define all the above for Euclidean vector spaces, and in fact, there are multiple different definitions per category,

[2] usually these belong to a *group*, a set endowed with another type of interesting structure - see section 2.3.

but this is not always possible for many data domains. For example, on a graph, we can define a metric, e.g. *the shortest path distance*, and therefore a topology, and equivalence relations, e.g. vertices that are mapped to each other via *automorphisms* (see section 2.4 for definitions), but in general it is not possible to define an ordering of the vertices or operations such as addition. On a set of strings (for natural language processing), we can define a metric, e.g. the string edit distance, an order, e.g. alphabetic, and equivalence relations, e.g. synonyms, but again it is unclear how to define arithmetic operations.

The absence of algebraic structure on many non-Euclidean spaces is what makes most machine-learning algorithms, and especially neural networks, directly inapplicable since their majority is defined on finite-dimensional vector spaces.[3] To address this, the common strategy is to represent the data as vectors and then restrict the model by making assumptions based on the assigned structure. Before diving into the specific case of graphs that this thesis is devoted to, in the next two sections, we will make a brief overview of two important steps one needs to follow: identifying the structure of the data (section 1.3) and then designing the ML algorithm by respecting certain universal principles (section 1.4).

1.3 Function approximation on non-Euclidean spaces: problem categorisation

Designing a learning algorithm for non-Euclidean data requires first understanding their nature, i.e. their structure, as described in section 1.2. In this section, we provide a categorisation of the learning problems. Identifying the category a problem belongs to gives us the ability to choose the appropriate framework to tackle it. In our categorisation, we identify three major families, to which either the input or the output set of the target function might belong. In every case, $\mathscr{X}$ (or $\mathscr{Y}$) is a set of singletons/points and is equipped with various features, that turn it into a (geometric) space, including at least a topology $\mathscr{T} \subseteq 2^{\mathscr{X}}$. The families are: *single space*, *function space* and *space of spaces*.

Single space $\mathcal{X}$. This is the most general case. We focus on examples where the singletons of the said space do not have any added structure themselves. Notable examples are:

[3]Two notable exceptions are nearest neighbours and kernels that can be applied to non-Euclidean data as long as some notion of similarity/distance between pairs of points can be defined.

- *Real coordinate space* $\mathbb{R}^d$ (e.g. $f^* : \mathbb{R}^{d_{\text{in}}} \to \mathbb{R}^{d_{\text{out}}}$). This is the most widely explored topic. There is a multitude of ways to model functions on/to such spaces, ranging from polynomials to neural networks. Strikingly though, even in this case, although some design principles are better understood (e.g. the expressive power, where well-known universal function approximation results apply - see 1.4.3), others are still enigmatic for the ML community (e.g. how do some inductive biases affect generalisation? - see sections 1.4.1 and 1.4.2).

- *Fixed manifold* (e.g. figure 1.1 left - $f^* : \mathcal{M} \to \mathbb{R}^{d_{\text{out}}}$, where $\mathcal{M}$ is a manifold). Here we encounter cases where the manifold is known, e.g. reasoning about a signal on a 3D surface from limited observations. It is also possible that the manifold is unknown. In fact, the *manifold hypothesis* [Schölkopf et al., 1998, Tenenbaum et al., 2000, Roweis and Saul, 2000, Belkin and Niyogi, 2003, Donoho and Grimes, 2003, Cayton, 2005, Fefferman et al., 2016] asserts that most high-dimensional real-world data reside in a low-dimensional manifold, therefore many classical ML problems fall under this category. Important progress in this subtopic has been made for manifolds of constant curvature, primarily *hyperbolic* and secondarily *spherical* spaces, either in the output (e.g. hyperbolic embeddings) [Wilson et al., 2014, Liu et al., 2017a, Nickel and Kiela, 2017, Chamberlain et al., 2017, Nickel and Kiela, 2018, Sala et al., 2018], or in the input as well (e.g. hyperbolic neural nets) [Ganea et al., 2018, Liu et al., 2019, Chami et al., 2019, Bachmann et al., 2020].

- *Fixed graph* (e.g. figure 1.2 left - $f^* : \mathcal{V} \to \mathbb{R}^{d_{\text{out}}}$, where $\mathcal{V}$ the vertex set of a graph $G = (\mathcal{V}, \mathcal{E})$). The input/output set here is the vertex set of a graph and the topology can be defined in multiple ways, e.g. the one induced by the shortest-path metric on the graph. Sometimes, it might be convenient to use alternative definitions of the set/topology pair, e.g. via the *line graph*, i.e. the graph representing the adjacency of edges. Typically the problems whose input set falls under this category are known in the ML literature as *semi-supervised*, or *transductive* learning on graphs and they deal with mapping each vertex to a target value from a limited amount of vertex observations (e.g. determining the subject area of a paper in a citation network, where each vertex is a paper and only a subset of them is labelled). Typical ML solutions are provided by graph embeddings [Perozzi et al., 2014, Grover and Leskovec, 2016, Abu-El-Haija et al., 2018] or Graph Neural Networks (GNNs) [Kipf and Welling, 2017, Hamilton et al., 2017, Velickovic et al., 2018] (see section 2.6.3), to name a few. Other similar problems include more exotic

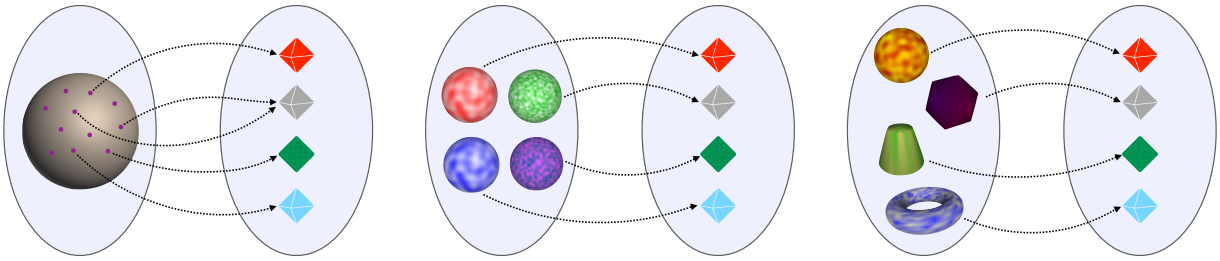


Figure 1.1: Problem categorisation (manifolds). The input space might be a (from left to right): single space (sphere), function space (signals on a sphere), space of spaces (space of manifolds).

discrete spaces, such as hypergraphs, simplicial complexes, CW complexes, etc.

Function space $\mathcal{U} = \{u : \mathcal{X} \to \hat{\mathcal{X}}\} \subseteq \hat{\mathcal{X}}^{\mathcal{X}}$. Typically, the co-domain of the functions u is the space of real numbers or a real-coordinate space $\hat{\mathcal{X}} = \mathbb{R}^d$, and the functions u are often called *signals*. The functions on signals are sometimes called *functionals*, when the output space is the space of real numbers or a real coordinate space, e.g. $f^* : \left(\mathbb{R}^{d_{\text{in}}}\right)^{\mathcal{X}} \to \mathbb{R}^{d_{\text{out}}}$, or *operators*, when the output space is also a space of signals, e.g. $f^* : \left(\mathbb{R}^{d_{\text{in}}}\right)^{\mathcal{X}} \to \left(\mathbb{R}^{d_{\text{out}}}\right)^{\mathcal{X}}$. We consider cases where the domain and co-domain $\mathcal{X}, \hat{\mathcal{X}}$ are fixed. For example:

- *Functions on vector spaces or manifolds* (e.g. figure 1.1 middle - $f^* : \left(\mathbb{R}^{d_{\text{in}}}\right)^{\mathcal{M}} \to \mathbb{R}^{d_{\text{out}}}$, where $\mathcal{M}$ is a manifold). Here we encounter problems where one wishes to model a function on/to an entire signal, such as mapping the initial conditions of a partial differential equation (PDE) over a Euclidean/non-Euclidean space to its solution, e.g. simulating weather conditions over the 3D surface of the earth. It is important to mention here that in case the said space is the input space, it might not be possible to obtain a succinct representation (that can be stored in a computer) of each signal, so in practice, we might encounter only a finite number of signal evaluations on its domain $\mathcal{X}$. This problem has recently started to gain traction in the neural network realm [Lu et al., 2021, Kovachki et al., 2021b, Dupont et al., 2022], both theoretically and practically, in order to solve PDEs and represent implicit functions, such as 3D shapes.

- *Functions on graphs and other discrete spaces* (e.g. figure 1.2 middle - $f^* : \left(\mathbb{R}^{d_{\text{in}}}\right)^{\mathcal{V}} \to \mathbb{R}^{d_{\text{out}}}$, where $\mathcal{V}$ the vertex set of a graph $G = (\mathcal{V}, \mathcal{E})$). Perhaps the most notable case here is that of *images*, the data domain that has gathered more attention than any other in ML. In particular, every image can be defined as a function from an input grid to $\mathbb{R}^d$ (where d is

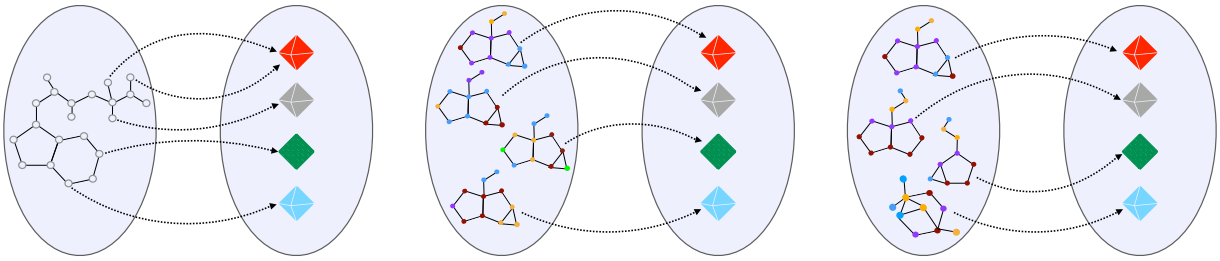


Figure 1.2: Problem categorisation (graphs). The input space might be a (from left to right): single space (the graph vertices), function space (signals on the graph vertices), space of spaces (space of graphs).

the called the number of channels), and the neural networks used in this setup are the well-known Convolutional Neural Networks (CNNs) [Fukushima and Miyake, 1982, LeCun et al., 1998, LeCun et al., 1989, Krizhevsky et al., 2017]. Also, an example of particular interest is that of functions on the vertices of a graph (*graph signals*). This problem has been largely addressed by methods originating in the field of *Graph Signal Processing (GSP)* (see section 2.5), such as graph wavelets [Coifman and Maggioni, 2006, Gavish et al., 2010, Rustamov and Guibas, 2013], and GNNs with similar philosophy [Bruna et al., 2014, Defferrard et al., 2016]. It is also the subject of investigation in the first publication contained in this thesis and we will extensively discuss it in chapter 3, where we will see that most GSP-based approaches have limited expressive power.

An attractive side note is that, when $\mathcal{U}$ refers to the output space, a particular case is *function approximation/learning per se*, since it requires finding a mapping from a dataset to a function (the learning algorithm $\mathscr{A} : \bigcup_{m=1}^{\infty} \left(\mathscr{X} \times \mathscr{Y}\right)^m \to \mathscr{Y}^{\mathscr{X}}$). This perspective has been studied by several works in the field of *meta-learning*, where a set of training datasets is used to determine (meta-learn) the learning algorithm [Mishra et al., 2018, Santoro et al., 2016, Garnelo et al., 2018], or some of its components (e.g. the parameter initialisation [Finn et al., 2017, Li et al., 2017b], the optimiser [Andrychowicz et al., 2016, Ravi and Larochelle, 2017], the hyperparameters [Franceschi et al., 2018] etc.). Please refer to [Hospedales et al., 2021] for a more detailed discussion.

Space of spaces $\mathfrak{X}$. In this case, we consider spaces where each singleton is a space itself, i.e. a set of points with some added structure. Adding structure to $\mathfrak{X}$, e.g. defining a topology, a metric, or an equivalence relation, is of particular importance in order to model functions in

this subtopic. For example:

- *Space of manifolds* (e.g. figure 1.1 right - $f^* : \mathfrak{M} \to \mathbb{R}^{d_{\text{out}}}$, where $\mathfrak{M} = \{\mathcal{M}_1, \mathcal{M}_2, \dots\}$ is a space of manifolds). Many 3D computer vision problems fall under this category, e.g. shape classification, shape segmentation, shape synthesis etc. Note though, that it is typically hard to represent 3D data as manifolds and other representations are preferred (e.g. point clouds, meshes, implicit functions), which explains why other perspectives (and corresponding neural nets) are chosen to deal with these problems. A more elaborate discussion is postponed until section 3.4.1 (in addition, in section 3.4.2 we apply one of the proposed algorithms to the particular case of deformable shapes). The equivalence relations that usually play a central role here are transformations between manifolds (e.g. rigid: rotations, translations and reflections) and designing invariant/equivariant networks is an active area of research [Thomas et al., 2018, Kondor, 2018, Shen et al., 2020, Satorras et al., 2021, Geiger and Smidt, 2022]

- *Space of sets* (e.g. $f^* : \mathfrak{P} \to \mathbb{R}^{d_{\text{out}}}$, where $\mathfrak{P} = \{\mathcal{P}_1, \mathcal{P}_2, \dots\}$ is a space of sets $\mathcal{P}_i = \{p_{i1}, p_{i2}, \dots\}$, with p_{ij}: singleton). In the most general case, each singleton is a set with no added structure. Then, the equivalence relation that guides the design of neural networks is the one arising from permutations of the elements of the set [Zaheer et al., 2017, Qi et al., 2017a]. However, in many cases (mostly in 3D data) a topology or metric can be defined allowing the definition of local architectures [Qi et al., 2017b, Thomas et al., 2019]. In addition, the equivalence relations mentioned in the previous subtopic might be observed here.

- *Space of graphs* (e.g. figure 1.2 right - $f^* : \mathfrak{G} \to \mathbb{R}^{d_{\text{out}}}$, where $\mathfrak{G} = \{G_1, G_2, \dots\}$ is a space of graphs $G_i = (\mathcal{V}_{G_i}, \mathcal{E}_{G_i})$). Frequently, in the literature, this setup is informally described by the term *inductive learning* on graphs and includes graph classification/regression tasks, where one needs to determine a mapping from a graph to a target value (e.g. predicting unknown properties of a molecule). The equivalence relation that governs the problems here is the one arising from *Graph Isomorphism (GI)* (see section 2.4 for a strict definition). In addition, graphs are equipped with a topology and a metric, which allows us to define local functions, which have been shown to be appropriate models for many real-world tasks. We investigate such problems in the 2nd and 3rd publications (chapters 4 and 5), where we thoroughly discuss the corresponding neural networks and the implications of

GI on their design.

Finally, note that the function space case is subsumed under the space of spaces case,[4] which in turn is is subsumed under the single space case. This partly explains why frequently the same neural networks are used in more than one setups. However, it is more appropriate to examine each problem through the lens of the most specific possible setup, in order to make the most of the problem structure (e.g. the fixed domain in the function space case or the internal topology of each singleton in the space of spaces case) with regards to the design principles that we will discuss in the next section.

1.4 Function approximation on non-Euclidean spaces: design principles

In this section we will give a general overview of the design principles we must adhere to when designing a learning system. These apply to all the categories we discussed in the previous section 1.3, but as we will see in the next sections, their theoretical understanding is in some cases limited. Relying on empirical observations and intuition is therefore a frequent phenomenon in (deep) learning.

The main principles are the following: (1) *Inductive biases*, i.e. what kind of assumptions about the true function are we making? These assumptions are present in all the components of the learning system, e.g. they might be *explicit* in the hypothesis class, by restricting it to functions that have an assumed property, or *implicit* in the learning algorithm, affecting the way it selects the hypothesis from a set of plausible options. (2) *Expressive power*, i.e. how to design a model that fits our assumptions? How powerful is our model in approximating functions with the assumed characteristics? (3) *Computational complexity*, i.e. the amount of compute needed to evaluate the selected hypothesis on an arbitrary input, and its scaling with the size of the input/output (whenever this is variable). Also, an important topic is the amount of computation needed for the learning algorithm in order to select the hypothesis.

[4]Each function u can be represented as a set $\{(x, u(x)) \mid x \in \mathcal{X}\}$, known as the *graph of the function*, which is a topological space, sometimes a manifold.

1.4.1 Explicit inductive biases

We first discuss the assumptions that are built in our hypothesis class, i.e. that apply to all the candidate hypotheses by construction. We will illustrate them in a consistent way with the types of structure mentioned in section 1.2. This is not an exhaustive list but contains the majority of assumptions that we usually encounter in learning systems. In general, a hypothesis might be a composition of functions, and the inductive biases that we will describe might apply to any of those functions or their composition as well.

Invariance/Equivariance

Perhaps the most ubiquitous assumption of learning systems is that of *invariance*. Informally, a function is called invariant to a set of transformations (which form *a group*) if its value remains unaffected when applying any of these transformations on the input and equivariant when the value of the function is transformed accordingly. The concept of transforming a point in the input or the output set is formalised by the notion of *group action*, which is described in detail in section 2.3. Formally:

Definition 1.1 (Invariant/Equivariant function)**.** *Let $\mathcal{X}$ be a set of points. Let $\mathcal{R}_{\mathcal{X}} \subseteq \mathcal{X} \times \mathcal{X}$ be an equivalence relation on $\mathcal{X}$ and we will write $x_1 \simeq_{\mathcal{R}_{\mathcal{X}}} x_2$ whenever $(x_1, x_2) \in \mathcal{R}_{\mathcal{X}}$. We will say that a function $h : \mathcal{X} \to \mathcal{Y}$ is invariant under $\mathcal{R}_{\mathcal{X}}$ if $x_1 \simeq_{\mathcal{R}_{\mathcal{X}}} x_2$ implies that $h(x_1) = h(x_2)$. Equivalently, let $\mathcal{G}$ be a group acting on $\mathcal{X}$, and denote the (left) group action with $\cdot : \mathcal{G} \times \mathcal{X} \to \mathcal{X}$. We will say that a function is invariant under the group action iff $h(g \cdot x) = h(x), \forall g \in \mathcal{G}, x \in \mathcal{X}$. Moreover, if the group also acts on $\mathcal{Y}$, we will say that a function is equivariant under the group action iff $h(g \cdot x) = g \cdot h(x), \forall g \in \mathcal{G}, x \in \mathcal{X}$.*

Sometimes invariance/equivariance are reasonable assumptions that we make about the target function, while in other cases they directly arise from the nature of the data, and failing to ensure that our model enjoys this property will lead to unreliable results. Some well-known transformations that we usually encounter in real-world data are the following:

Infinite groups, i.e. groups with an underlying set of infinite elements. The most useful examples are subgroups of the *general linear* group $GL(d, \mathbb{C})$, i.e. the set of all invertible matrices in $\mathbb{C}^{d \times d}$ (where $\mathbb{C}$ the field of complex numbers), such as the *Euclidean group* $E(d)$, which contains all isometric transformations of a Euclidean d-dimensional space, including translations

$T(d)$ and rotations and reflections $O(d)$ - the *orthogonal group* (the group containing only the rotations is called the *special orthogonal group* and is denoted with $SO(d)$).

Finite groups, i.e. groups with an underlying set of a finite number of elements. All finite groups are subgroups of the *symmetric group* $S(d)$, containing all permutations of d elements (*Cayley's theorem*). This group and subgroups thereof will be the ones that we will encounter in this thesis, in chapters 3, 4 and 5. In particular, *shifts* of various kinds, the discrete analogue of translations, form subgroups of $S(d)$ (see chapter 3). Moreover, when dealing with function approximation problems on/to a space of graphs, we want our functions to be invariant/equivariant to isomorphism (see section 2.4), which can be defined via the symmetric group acting on the vertex and edge sets of a graph.

In the early deep learning years, this inductive bias was usually implicit, and its existence was encouraged by *data augmentation*, i.e. by enlarging the training set with datapoints manually transformed by randomly sampled transformations from the relevant group. However, this not only increased the computational complexity of the learning algorithm but also could not provide guarantees for invariance/equivariance. As a remedy to that, it is therefore important to design invariance/equivariance as an explicit inductive bias, i.e. to guarantee that all the hypotheses in the hypothesis class will have this property, whenever a symmetry is known. This is not only because it guarantees the correctness and reliability of our model, but also because (1) it reduces considerably the size of the hypothesis class, and therefore improves the convergence of learning algorithms, and (2) because it makes our learning algorithm more sample efficient as has been shown in various works [Shawe-Taylor, 1991, Shawetaylor, 1995, Sokolic et al., 2017, Lyle et al., 2020, Sannai et al., 2021, Zhu et al., 2021, Behboodi et al., 2022]. However, embedding symmetries into the hypothesis class is generally a non-trivial task (it is currently an active area of research). At the same time, it is not always clear how to select an appropriate invariant/equivariant hypothesis class with desirable expressive power and computational complexity properties, while, as a matter of fact, sometimes leads to a fundamental trade-off (see chapters 4, 5).

Locality

We continue our discussion with the *locality* assumption, another classical convention in ML. Informally, a function is described as local if its value on a point of the input set depends not

only on the point itself but also on its *neighbours*. Formally:

Definition 1.2 (Local function)**.** *Let $\mathcal{X}$ be a set of points and $\mathscr{T} \subseteq 2^{\mathcal{X}}$ a topology on $\mathcal{X}$. Let $\mathcal{N}(x)$ be a neighbourhood of x, i.e. any $\mathcal{N}(x) \in \mathscr{T}$, such that $x \in \mathcal{N}(x)$. Then, a function $f_{local} : \mathcal{X} \to \mathcal{Y}$ will be called local if it can be expressed as follows:*

$$f_{\text{local}}(x) \coloneqq \psi\left(x,\, \{(x, x') \mid x' \in \mathcal{N}\left(x\right)\}\right),\ \forall x \in \mathcal{X}. \tag{1.1}$$

The following observations can be made. First off, locality can be defined even for sets with weak mathematical structure, since its definition only requires a topology. Second, the definition is quite flexible and the exact form of a local function will depend on our choices of neighbourhoods $\mathcal{N}(x)$ and the way that each neighbour will be involved in the computation. For example, in a metric space, one might define a neighbourhood as a metric ball with a certain radius ρ: $\mathcal{N}_\rho(x) = \{x' \in \mathcal{X} \mid d_{\mathcal{X}}(x, x') \leq \rho\}$. Moreover, neighbours might be treated equally based on various equivalence relations, e.g. their distance from x (by the term "treated equally", we informally imply that the function f_{local} will be e.g. invariant to permutations of equivalent neighbours). The definition also allows for multiple levels of computation, e.g. based on different values of ρ, which is a common design pattern of CNNs and GNNs.

In the **single space** case, locality assumptions are either implicit or explicit. For example, in the single vector space setup that we described in section 1.3, the neighbourhoods can be derived with algebraic operations on x that neural networks can express, and therefore rarely this assumption is explicitly made. On the other hand, in many non-Euclidean setups, this is not the case and therefore locality needs to be explicit in the hypothesis. For example, in the case of a single graph, the hypothesis might be a variation of a GNN, which amounts to a local function or a composition of local functions, where the neighbourhood might be defined based on the graph metric.

In the **function space** and **space of spaces** setups, locality assumptions are usually explicit as well, but now locality refers to the internal topology of each point. For example, a classical formulation for hypotheses on a space of spaces is: $h(\mathcal{X}) = \phi\Big(\big\{f_{\text{local}}\left(x\right) \mid x \in \mathcal{X}\big\}\Big)$, $\forall\ \mathcal{X} \in \mathfrak{X}$, where the function ϕ is known as the *readout* (this is the rationale in the design of GNNs - see section 2.6.3). Similarly, for hypotheses on a space of functions, the formula reads: $h(u) = \phi\Big(\big\{f_{\text{local}}\big(x, u\left(x\right)\big) \mid x \in \mathcal{X}\big\}\Big)$, $\forall\ u \in \mathcal{U} \subseteq \hat{\mathcal{X}}^{\mathcal{X}}$, but now usually the neighbourhoods used by f_{local} are defined based on distances from the input point x, i.e.

$$f_{\text{local}}(x) \coloneqq \psi\Big(\big(x, u(x)\big),\ \big\{\big(x, u(x), x', u(x')\big) \mid x' \in \mathcal{N}(x)\big\}\Big).$$

Local functions are common in real-world scenarios and this observation has been widely used as an intuitive motivation for putting this assumption forward. As a matter of fact, locality has been adopted by the signal processing community even before the deep learning era, where signals defined over various domains were processed with *local filters* (e.g. for denoising and edge detection in images). In graphs, as we will discuss in chapter 4 many real-world tasks are compositions of local functions over the graph, such as functions that depend on *graph substructures*.

1.4.2 Implicit inductive biases

In the section that follows we discuss an important yet not well-understood concept that should be always taken into account when designing neural networks in general and helps us create a more holistic image of the behaviour of our models. Since the following discussion is not crucial to understand our contributions, a reader familiar with the topic may safely skip it.

Undoubtedly the progress in deep learning methods is dominated by approaches that explicitly restrict the hypothesis class. However, even with these restrictions, the hypothesis class is still sufficiently large, including many functions that can achieve equally small *empirical error* (error on the training set) with the target function, but will be poor approximations, i.e. their *generalisation error* will be large. However, it is well-known that deep learning systems have achieved remarkable generalisation performance in a plethora of artificial intelligence tasks.

This has been a puzzling phenomenon for researchers for many years and still remains largely unexplained. The predominant belief attributes it to *implicit inductive biases* and asserts that there exists an interplay between the hypothesis class (architecture) and the learning algorithm, which favours certain functions over others. The former includes the explicit inductive biases, as well as a set of *architecture hyperparameters*, such as the depth (number of layers), width (dimension of intermediate representations) and activation functions. The latter is usually *not* the Empirical Risk Minimisation (ERM) algorithm (see section 2.2), which selects the hypothesis with the smallest empirical error, as this is usually a non-convex optimisation problem, which in general is NP-hard. Instead, in deep learning, the learning algorithm is typically a variation of *gradient descent (GD)*[5] paired with a particular initialisation (an initial guess) of the predicted

[5]more precisely, *stochastic gradient descent (SGD)*, i.e. a randomised variation that estimates the true

hypothesis and a set of *optimisation hyperparameters*, such as the learning rate, the number of optimisation iterations and other heuristics that schedule training (e.g. learning rate scheduler, early stopping, etc.).

Most of the research on implicit inductive biases has been focused on the first problem setup - functions on/to a **single space** - and even more so on the particular case of a single vector space (or a single known and convenient manifold, such as the unit sphere [Bubeck and Sellke, 2021]). The existing results so far are mainly theoretical investigations of simplified settings [Jacot et al., 2018, Du et al., 2019d, Allen-Zhu et al., 2019] or empirical. Nevertheless, they have provided valuable insights that have contributed to our understanding of the behaviour of neural networks and detecting their failure cases.

We will now discuss certain implicit inductive biases that are theorised to be instrumental in the behaviour of neural networks. A general term that has been proposed is that of the *simplicity bias* [Pérez et al., 2019a], i.e. describing the intuitive theory that neural networks trained with (S)GD "learn simple functions", or more precisely "learn simple patterns first during training". To quantify simplicity, some authors have proposed looking into the (distribution of) the distances of the training data from the decision boundaries, i.e. the margins, for classification setups [Arpit et al., 2017, Jiang et al., 2019], claiming that neural networks select hypotheses with large margins, or the Fourier transform of the learned hypothesis for ReLU networks, claiming that neural networks prefer low-frequency hypotheses (or learn low-frequency components first) - the *spectral bias* [Rahaman et al., 2019]. A similar postulate has been put forward for the single space setup where the input space is a graph, and it is widely known as the *homophily* assumption, which asserts that GNNs, under certain conditions, prefer functions where nearby vertices tend to have the same or similar target values [Zhu et al., 2020, Ma et al., 2022]. An interesting insight that can be inferred from all the above observations is that (under certain conditions) neural networks prefer functions that tend to have small *rates of change*, at least close to the support of the data distribution; or more precisely they tend to have no larger rates of change close to the training data, than the ones required to fit them. However, a rigorous statement still remains beyond our current understanding.

All the above coincide with our knowledge of many real-world tasks, that are known to be *simple* [Schmidhuber, 1997, Lin et al., 2017] or *homophilic* [McPherson et al., 2001]. At the same time, they potentially pinpoint a limitation of neural networks in approximating *high-frequency* or

gradient from subsets (batches) of the training data.

heterophilic functions, a limitation which has motivated the development of new architectures [Sitzmann et al., 2020b, Tancik et al., 2020, Zheng et al., 2022].

To conclude our discussion, it is important to mention here that there are many other factors that implicitly affect the selection of the hypothesis (and in turn the ability to generalise), that are of equal importance when designing a learning system. In particular, designers tend to select models that are "easy/fast to train" , while typically the hyperparameters are chosen based on a *validation set*. This gives us access to a (usually unbiased) estimate of the generalisation error, which is a powerful heuristic for model selection.

Finally, it goes without saying that the *training dataset* is particularly impactful with regards to generalisation (the larger the dataset, the more restricted the set of hypotheses that can fit it), acting just like an inductive bias itself. Many models, only recently published, that have been trained with immense amounts of data, have achieved unprecedented generalisation performance in many tasks [Brown et al., 2020, Ramesh et al., 2022, Poole et al., 2022, Ho et al., 2022], while, frequently, similar performance has been observed for different architectures. It is therefore probably only a slight exaggeration to say that the aforementioned inductive biases pale in comparison with the power of data[6] [Sutton, 2019]. Having said that, as we will see in section 1.4.4, larger datasets severely increase the computational complexity of the learning system, in terms of time and space complexity (increased number of iterations and parameters) and the data collection costs, while the access to them is limited. Thus, it becomes evident that this inductive bias is a privilege only for the few. Therefore, there is still a pressing need to design (explicit or implicit) inductive biases in the architecture and the learning algorithm, in order not only to be able to "democratise" learning systems at smaller scales but also to solve tasks where the collection of large datasets is impossible, e.g. in the physical sciences and biomedicine.

1.4.3 Expressive Power

Having established our assumptions, the next design principle that we need to adopt is to make sure that our hypothesis class is sufficiently flexible, i.e. that it contains a sufficiently large set of hypotheses. We will say that the larger the hypothesis class, the larger its *expressive power* (or *expressivity* - the two terms will be used interchangeably).

[6] Strikingly, this is a phenomenon which still cannot be explained by classical learning theory results, since the curse of dimensionality demands exponentially large datasets with the input dimension.

Frequently, we are interested in hypothesis classes that have the *universal approximation property (UAP)*, i.e. that can approximate any target function in a function family that agrees with our assumptions, up to arbitrary precision in any (compact) subset of our input set $\mathscr{X}$.[7] It is important to note here that the UAP is purely existential, i.e. it does not give us guidance on how to construct the approximator for a known target function, or when we only have access to a finite amount of samples from it, or about the hardness of approximation. Nevertheless, it is an important result that informs us about the possibility of approximation or lack thereof.

The UAP is a concept that has been deeply studied in the field of approximation theory for functions on and to real coordinate spaces $\mathbb{R}^d$. Its study dates back to at least the 19th century when Weierstrass [Weierstrass, 1885] showed that any real-valued continuous function on a closed interval of the real numbers can be approximated by a *polynomial*. Importantly, this result was generalised by Stone [Stone, 1937, Stone, 1948] in the *Stone–Weierstrass theorem* giving sufficient conditions for a family of real-valued functions to have the UAP on more general topological spaces.

In the neural network realm, many results have been derived for **real coordinate spaces** starting from the seminal works of Cybenko [Cybenko, 1989] and Hornik [Hornik et al., 1989, Hornik, 1991]. These and follow-up papers, e.g. [Pinkus, 1999, Yarotsky, 2017, Lu et al., 2017, Kidger and Lyons, 2020, Maiorov and Pinkus, 1999, Park et al., 2021, Sonoda and Murata, 2017], showed that neural networks with at least two layers and non-polynomial activation functions can approximate any continuous function $\mathbb{R}^{d_{\text{in}}} \to \mathbb{R}^{d_{\text{out}}}$ on any compact subset of $\mathbb{R}^{d_{\text{in}}}$ up to arbitrary precision. In fact, the UAP provided strong arguments in favour of neural networks as learning paradigms in general and has been widely used to justify their popularity. More general results for arbitrary topological spaces were shown in [Kratsios and Bilokopytov, 2020, Kratsios and Papon, 2022], with implications for input/output spaces which are smooth manifolds.

Regarding the **function space** setup, it is helpful for our understanding, to rewrite the definition of a function space with finite domain $\mathcal{X}$ as $\hat{\mathcal{X}}^{|\mathcal{X}|}$. This implies that any sufficient conditions for universality in this product space also apply to the function space. For example, when $\hat{\mathcal{X}} = \mathbb{R}^{d_{\hat{\mathcal{X}}}}$, then the product space contains $|\mathcal{X}|$ concatenated vectors of dimension $d_{\hat{\mathcal{X}}}$, i.e. $\mathcal{U} = \mathbb{R}^{|\mathcal{X}| \cdot d_{\hat{\mathcal{X}}}}$, and therefore all the known universal approximation theorems for real coordinate spaces apply. This

[7] A more general notion, that of *Turing completeness* has been also explored for Recurrent Neural Networks (RNNs) [Siegelmann and Sontag, 1992, Schäfer and Zimmermann, 2006], Transformers [Pérez et al., 2019b, Yun et al., 2020] and GNNs [Loukas, 2020].

observation is of importance for many tasks in ML, such as learning on images and audio of fixed resolution, as well as learning on graph signals, since it tells us that we can use conventional neural networks (such as MLPs) without compromising on universal approximation.

Nevertheless, this approach does not enforce any restrictions to the hypothesis class, which as we will see in section 1.4.4 leads to an increase in both space complexity (increased number of parameters) and time complexity (increased neural network inference time and potentially learning algorithm iterations), but typically also in generalisation error. This motivates the study of domain-specific architectures w.r.t. their approximation capacity. Such results have been provided for Convolutional Neural Networks (CNNs), which have been shown in [Yarotsky, 2022, Zhou, 2020] to be universal when mapping discrete signals to real vectors.

Separate treatment is needed when the signal domain $\mathcal{X}$ is infinite, e.g. when $\mathcal{X} = \mathbb{R}^{d_\mathcal{X}}$. Although less well-known, many theoretical results have been derived in this case, regarding more general topological domains $\mathcal{X}$ and function spaces, showing the universality of certain architectures, most notably the DeepONet and Neural Operators of different types [Chen and Chen, 1993, Chen and Chen, 1995a, Chen and Chen, 1995b, Mhaskar and Hahm, 1997, Rossi and Conan-Guez, 2005, Lu et al., 2021, Lanthaler et al., 2022, Bhattacharya et al., 2021, Kovachki et al., 2021b, Kovachki et al., 2021a].

Finally, there has been significant progress in analysing function families on/to spaces equipped with a certain symmetry/equivalence relation, which also applies to the setup of **space of spaces** $\mathfrak{X}$. It might be tempting to approach this problem with general function approximators on $\mathfrak{X}$, e.g. in some cases $\mathfrak{X}$ is a subset of a real coordinate space (e.g. the space of all graphs with n vertices can be perceived as a subset of $\mathbb{R}^{n^2}$ - see section 4.2), where universal approximation is well-understood), but as we discussed in section 1.4.1, this hinders both optimisation and generalisation. It is therefore necessary to study approximators that are by construction invariant/equivariant.

A rather general strategy is that of *symmetrisation*, where invariant functions are defined, for every point $\mathcal{X} \in \mathfrak{X}$, as the average over the outputs of a (possibly) asymmetric function f_{asym} evaluated on the points in the equivalence class of $\mathcal{X}$. Formally, for invariance to the action of a group $\mathcal{G}$ we have: $h(\mathcal{X}) = \frac{1}{|\mathcal{G}|}\sum_{g\in\mathcal{G}} f_{\text{asym}}(g \cdot \mathcal{X})$.[8] Several universal approximation theorems have been derived for this class of functions, e.g. see [Yarotsky, 2022], under the condition that

[8] for equivariance the corresponding defintion reads $h(\mathcal{X}) = \frac{1}{|\mathcal{G}|}\sum_{g\in\mathcal{G}} g^{-1} f_{\text{asym}}(g \cdot \mathcal{X})$.

f_{asym} belongs to a universal function approximator on $\mathfrak{X}$. Nevertheless, this is practical only when the group is finite and relatively small, which prevents us from constructing approximators for many interesting symmetries (e.g. the size of a permutation group is $|S(n)| = n!$), while for infinite groups, such as rotations, the averaging operator is an integral and can be only estimated with sampling.

To address this, the predominant approach in recent years is constructing hypothesis classes, that contain *intrinsically* invariant/equivariant functions. Sufficient conditions are once again provided in [Yarotsky, 2022] using the theory of *symmetric polynomials*, while particular architectures have been analysed w.r.t the UAP [Segol and Lipman, 2020, Bogatskiy et al., 2020, Dym and Maron, 2021] and [Villar et al., 2021] provided sufficient conditions for many symmetries that arise from physical laws.

We make a separate allusion to the permutation symmetry of graphs. In particular, although sufficient conditions have been provided for universal function approximation on graphs [Loukas, 2020, Dasoulas et al., 2020, Murphy et al., 2019, Brüel Gabrielsson, 2020, Keriven and Peyré, 2019, Maron et al., 2019c], the resulting architectures are either not permutation invariant/equivariant or are computationally intractable. This is not surprising, since as shown in [Chen et al., 2019], universal approximation of invariant/equivariant graph functions is equivalent to solving graph isomorphism, which so far is not known to be solvable in polynomial time (the best algorithm runs in quasipolynomial time and it is due to Babai [Babai, 2016] - see also section 2.4). In fact, an important trade-off between computational complexity, expressivity and permutation invariance has been observed in this domain, with graph isomorphism being considered largely responsible: the hypothesis classes that have been proposed are either overly flexible (certain isomorphic graphs are mapped to different outputs) or overly restrictive (certain non-isomorphic graphs are mapped to the same output), sacrificing permutation invariance or expressivity respectively. Typically, the degree of their flexibility/restrictions is tightly related to their computational complexity. We will extensively discuss this topic in chapter 4, where we will also derive a universality result for a specific class of graph functions and a practical architecture with improved expressivity compared to vanilla GNNs.

1.4.4 Computational Complexity

The final universal design principle that we should be paying attention to is that of *computational complexity*. Although it is a concept that is less frequently discussed in ML publications, it was probably the most decisive factor that impeded deep learning progress in the late 20th century; neural networks were too expensive to train and the functions they were converging to after only a handful of iterations where not even sufficiently good in terms of their empirical error on the training data. However, the advent of modern hardware (mainly GPUs and new technology, such as TPUs), made massively parallel computation possible, which unlocked the potential of neural networks.

The time complexity of a learning system, regarding its training phase, can be summarised with the following scaling law: $O\left(f\left(d_{\text{in}}, d_{\text{out}}\right) \cdot |\mathfrak{D}| \cdot I\right)$, where $f\left(d_{\text{in}}, d_{\text{out}}\right)$ is typically the complexity of the computation needed to update the parameters of the hypothesis (e.g. the backward pass - gradient computation), based on each sample in the dataset, $|\mathfrak{D}|$ are the number of samples and I the number of epochs. In many cases f is linear (e.g. MLPs and CNNs with constant depth and width) and even if d_{in} and d_{out} are large, the computation can be parallelised (perhaps with the exception of RNNs and in general autoregressive models). Similarly, the computation can be parallelised across samples, which is done via mini-batching and stochastic estimation of gradients, which together with many improvements in weight initialisation schemes, optimisation algorithms, architecture modifications (e.g. residual connections, normalisation layers etc.) and hyperparameter heuristics, also allow for a reduction in the number of iterations I.

Of equal importance is space complexity. Modern neural networks are heavily overparametrised creating the need for large memory capacity. However, modern hardware has also significantly improved in that respect, allowing us to empirically observe phenomena (e.g. *double descent* [Belkin et al., 2019, Nakkiran et al., 2020]) that could not be explained by measuring model complexity with the number of parameters and corroborated the arguments in favour of alternative measures [Rasmussen and Ghahramani, 2000]. Overall, although training neural networks is still a notoriously laborious and slow procedure, all the above have allowed significant reductions in their computational complexity, making them more accessible and providing room for wider experimentation.

However, as the community started to focus on more complex tasks, more intricate architectures became necessary with f scaling worse than linearly w.r.t. the input and output dimensions

(e.g. attention-based neural nets, such as transformers, have quadratic complexity), which brought the issue of computational complexity to the front once more. In addition to that, *the computational complexity of a problem is inescapable*: if it belongs to one of certain complexity classes, such as NP, then there will be instances of it that we most likely won't be able to solve with architectures that run in polynomial, let alone in linear, time. Instead, our goal should be to learn appropriate heuristics, akin to many combinatorial optimisation solvers, that can approximate the ground truth function well enough with high probability, using samples from a given distribution of instances. In many cases, learning such heuristics is indeed possible, e.g. several problems, such as graph isomorphism, are hard because of certain instances that are "artificial", while real-world distributions contain instances for which the problem can be solved far more efficiently (see section 4.2).

This discussion is of particular relevance to this thesis. As we saw in section 1.1, there is a peculiarity in the structure of graphs that makes many problems defined on them computationally cumbersome. This includes universal isomorphism-invariant function approximation (section 1.4.3), which is unknown if it is solvable in polynomial time, and unlabelled graph compression (chapter 5), which at optimality is a specific case of *isomorphism-injective & isomorphism-invariant function approximation*. Other invariant problems such as *subgraph isomorphism* (i.e. deciding if a graph contains a subgraph isomorphic to another given graph) and the *clique decision problem* (i.e. deciding if a graph contains a k-clique) are known to be *NP-complete*. These problems are also of practical importance since subgraphs play a crucial role in many real-world tasks (see chapter 4).

At the same time, many graph problems can be approached with equivariant function approximation, e.g. consider functions $\mathfrak{X} \to \bigcup_{n=1}^{\infty} \mathcal{Y}^n$, where $\mathcal{Y} = \{0, 1, \ldots, |\mathcal{Y}|\}$, i.e. the output is a label vector, such as a zero-one vector, where each one of the n values corresponds to a vertex of the graph and indicates if it belongs to a target set or not. In this category, we find problems that are computationally hard, e.g. the *graph k-colouring* with $k \geq 3$, which is NP-complete (requires colouring the vertices of the graph with k colours, such that no adjacent vertices have the same colour), but sometimes also hard to approximate, e.g. the *maximum independent set* and the *vertex cover*, which are NP-hard (require finding the maximum vertex set with non-adjacent vertices and finding a vertex set such that all edges are incident to at least one of the selected vertices, respectively). In the field of combinatorial optimisation, many real-world problems are variations of the above.

It is now clear that computational complexity is a crucial factor that needs to be taken into account when designing graph ML algorithms. It is imperative for the designer to make the necessary compromises between computational complexity and expressive power (i.e. even if our hypothesis class is not universal, it might be sufficient to approximate a function well enough on a given real-world distribution). In this thesis, we will also empirically examine the interplay between computational complexity and generalisation. In chapters 4 and 5 we will see cases where larger computational complexity does not improve or sometimes worsens, generalisation (interestingly note that this is at odds with the empirical evidence about *overparametrisation* that we mentioned above). However, this interplay still remains beyond our current understanding.

1.5 Thesis outline & summary of problems considered

The rest of the thesis is organised as follows:

Chapter 2 introduces mathematical definitions and notations that will be necessary across all the chapters in the thesis and provides a brief overview of the machine learning terminology and the background knowledge on graph theory. Additionally, it contains the necessary preliminaries regarding relevant methods that have been used for the analysis and processing of graphs in a non-data-driven manner, mainly spanning the field of graph signal processing. Finally, it introduces the reader to Graph Neural Networks and their ancestors, the Weisfeiler-Leman graph isomorphism tests.

Chapter 3 describes our contribution to the problem of **neural function approximation on graph signals**. In particular, as per the categorisation of section 1.3, this chapter is concerned with learning on signal spaces, where the signal domain is the vertex set of a fixed graph and the co-domain is a real-coordinate space. In section 3.2 we will discuss how the design principles of section 1.4 apply in this context, and we will recall that it is easy to design function approximators that hold the UAP (as we saw in section 1.4.3) but at the expense of weak inductive biases and high computational complexity. However, addressing this requires defining an equivalence relation akin to shifts on Euclidean grids, which is a long-standing research question in Graph Signal Processing - GSP (see section 2.5). Based on this observation, we will define an analogous shift operator for graphs, that retains a fundamental property of conventional shifts, i.e. that of being permutations of the domain of the function (see section

3.3.1) and we will showcase a particular instantiation for mesh signals (*Spiral Convolutional Networks* - see section 3.3.2).

In section 3.3.3 we will show that our shift operator has expressivity advantages compared to conventional graph shift operators introduced in the field of GSP (*anisotropy vs isotropy*) and computational advantages compared to the so-called patch-operator based and attention-based GNNs, while we will illustrate connections with GNNs that are sensitive to graph isomorphism (*GI-sensitive/permutation-sensitive*). Finally, section 3.4.1 discusses our application of interest, shape modelling, and related topics, while section 3.5, provides an empirical comparison against other GNNs and traditional statistical methods for shape modelling. Appendix A provides implementation details and supportive experimental results. This work was originally presented in [Bouritsas et al., 2019]. A preliminary version can be found in [Bokhnyak et al., 2019].

In **Chapter 4** we shift our focus to **neural function approximation on general graph spaces**. This problem pertains to the setup where our input space is a space of spaces, where each singleton is a graph, possibly paired with a graph signal. In section 4.2 we will discuss the most common case of target functions that are invariant to graph isomorphism (*GI-invariant*). We will examine the famous trade-off in the GNN community, between expressive power, computational complexity and GI-invariance and its implications in the computation of graph properties with neural networks, either optimally (which relates to expressive power) or approximately (which relates to generalisation).

In section 4.3, we present *Graph Substructure Networks*, a GNN equipped with substructure encodings, that is GI-invariant and provably more expressive than vanilla GNNs (section 4.4), while the unavoidable increase in the computational complexity is only incurred during a preprocessing step (section 4.5). Section 4.6 discusses practical considerations regarding the substructures of interest, while section 4.7 provides an overview of the vast literature on the topic of GNN expressivity, including works that succeeded ours and further advanced the research on substructures as a GNN inductive bias. Section 4.8 concludes the chapter by comparing the empirical generalisation error of our method on a battery of graph analysis tasks (biological, chemical and social networks) and with relevant ablation studies. Omitted proofs, computational considerations and experimental details are relegated to the Appendix B. This work was originally presented in [Bouritsas et al., 2022]. A preliminary version can be found in [Bouritsas et al., 2020].

In **Chapter 5** we investigate **lossless graph compression**, another problem of function approximation on general graph spaces. Our interest in this problem was fuelled by two factors. First, by the practical considerations of data compression and the scarcity of approaches based on information-theoretic principles. Second, by the intimate connections between compression and generative models (or random graph models in network science terminology) and the theoretical challenges that arose in this investigation. In section 5.3 we show that compressing graphs without information loss amounts to approximating a **GI-injective function** (one that evaluates to different outputs for non-isomorphic graphs) and **estimating a graph distribution**. In the case of unlabelled graphs, optimality requires the former function to also be GI-invariant, which turns out to imply another fundamental trade-off between compression quality and computational complexity.

Section 5.4 presents our compression framework (*Partition and Code*). We initially observe that the distribution estimator should be also compressible as its parameters need to be also stored along with the data, which is a fact that is frequently ignored in neural compression. To address this, we depart from overparametrised neural estimators and propose a dictionary-based estimator: initially we partition the graph and map the resulting subgraphs to the elements of an adaptive dictionary (section 5.4.1), and subsequently, we encode each component of the resulting decomposition using a parameter-efficient distribution (section 5.4.2). The partitioning algorithm, the dictionary as well as the distribution are jointly optimised w.r.t. the total compression gains (i.e. storage cost of both the data and the estimator - see sections 5.4.3 and 5.6). Our method builds on the assumption that most real-world graphs consist of a small number of substructures of small size (for which graph isomorphism can be solved efficiently) repeated in high frequencies (*network motifs*). We formalise this theoretically in section 5.5 and empirically in section 5.8 on diverse domains showcasing increased compression gains compared to various competitive graph compressors. As a byproduct, our algorithm extracts substructures of high frequency, that can be further used for graph analysis and interpretation or as features for downstream supervised learning tasks. Finally, Appendix C provides omitted proofs, technical algorithmic and implementation details and additional experiments. This work was originally presented in [Bouritsas et al., 2021].

In the final **Chapter 6**, we summarise the contributions of the thesis and the collective conclusions that can be drawn from this research, as well as its impact and the future directions that we envision in this scientific field.

1.6 List of publications

Related Publications. The following articles are the base of this thesis and a separate chapter is dedicated to each one of them.

- **Giorgos Bouritsas***, Sergiy Bokhnyak*, Stylianos Ploumpis, Michael M. Bronstein, and Stefanos Zafeiriou. "Neural 3d morphable models: Spiral convolutional networks for 3d shape representation learning and generation." *IEEE/CVF International Conference on Computer Vision (ICCV), 2019 & International Conference on Learning Representations Workshops (ICLRW), 2019.* [Bouritsas et al., 2019, Bokhnyak et al., 2019]. Discussed in Chapter 3. The code is available at https://github.com/gbouritsas/neural3DMM.

- **Giorgos Bouritsas**, Fabrizio Frasca, Stefanos Zafeiriou, and Michael M. Bronstein. "Improving graph neural network expressivity via subgraph isomorphism counting." *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI), 2022 & International Conference on Machine Learning Workshops (ICMLW), 2020.* [Bouritsas et al., 2022, Bouritsas et al., 2020]. Discussed in Chapter 4. The code is available at https://github.com/gbouritsas/GSN.

- **Giorgos Bouritsas**, Andreas Loukas, Nikolaos Karalias, and Michael M. Bronstein. "Partition and Code: Learning how to compress graphs." *Advances in Neural Information Processing Systems (NeurIPS), 2021.* [Bouritsas et al., 2021]. Discussed in Chapter 5. The code is available at https://github.com/gbouritsas/PnC.

Other Publications The following articles were produced as part of the author's PhD research and are the output of collaborations on topics related to the ones presented in the thesis.

- Grigorios G. Chrysos, Stylianos Moschoglou, **Giorgos Bouritsas**, Yannis Panagakis, Jiankang Deng, and Stefanos Zafeiriou. "Π-nets: Deep Polynomial Neural Networks." *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2020.* [Chrysos et al., 2020].

- Rolandos Alexandros Potamias, Jiali Zheng, Stylianos Ploumpis, **Giorgos Bouritsas**, Evangelos Ververas, and Stefanos Zafeiriou. "Learning to Generate Customized Dynamic 3D Facial Expressions." *European Conference on Computer Vision (ECCV),*

2020. [Potamias et al., 2020].

- Soha Sadat Mahdi, Nele Nauwelaers, Philip Joris, **Giorgos Bouritsas**, Shunwang Gong, Sergiy Bokhnyak, Susan Walsh, Mark D. Shriver, Michael M. Bronstein, and Peter Claes. "3D Facial Matching by Spiral Convolutional Metric Learning and a Biometric Fusion-Net of Demographic Properties." *IEEE International Conference on Pattern Recognition (ICPR), 2020.* [Mahdi et al., 2021a].

- Grigorios G. Chrysos, Stylianos Moschoglou, **Giorgos Bouritsas**, Jiankang Deng, Yannis Panagakis, and Stefanos Zafeiriou. "Deep polynomial neural networks." *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI), 2021.* [Chrysos et al., 2021].

- Soha Sadat Mahdi, Nele Nauwelaers, Philip Joris, **Giorgos Bouritsas**, Shunwang Gong, Susan Walsh, Mark D. Shriver, Michael M. Bronstein, and Peter Claes. "Matching 3D Facial Shape to Demographic Properties by Geometric Metric Learning: A Part-Based Approach." *IEEE Transactions on Biometrics, Behavior, and Identity Science (TBIOM), 2021.* [Mahdi et al., 2021b].

- Soha Sadat Mahdi, Harold Matthews, Nele Nauwelaers, Michiel Vanneste, Shunwang Gong, **Giorgos Bouritsas**, Gareth S. Baynam, Peter Hammond, Richard Spritz, Ophir D. Klein, Benedikt Hallgrímsson, Hilde Peeters, Michael M. Bronstein, and Peter Claes. "Multi-Scale Part-Based Syndrome Classification of 3D Facial Images." *IEEE Access, 2022.* [Mahdi et al., 2022a].

- Rolandos Alexandros Potamias, **Giorgos Bouritsas**, and Stefanos Zafeiriou. "Revisiting point cloud simplification: A learnable feature preserving approach." *European Conference on Computer Vision (ECCV), 2022.* [Potamias et al., 2022].

- Soha Sadat Mahdi, Harold Matthews, Michiel Vanneste, Nele Nauwelaers, Shunwang Gong, **Giorgos Bouritsas**, Gareth S. Baynam, Peter Hammond, Richard Spritz, Ophir D. Klein, Benedikt Hallgrímsson, Hilde Peeters, and Peter Claes. "A 3D Clinical Face Phenotype Space of Genetic Syndromes using a Triplet-Based Singular Geometric Autoencoder". *bioRxiv, 2022 & under review.* [Mahdi et al., 2022b].

- Eleni Batziou, **Giorgos Bouritsas**, Alexandros Tsigonias-Dimitriadis. "Iterative combinatorial auctions: an improved approach using reinforcement learning". *under review, 2023.* [Batziou et al., 2023].

2

Background and Preliminaries

2.1 Notation

Below we summarise the main notation conventions that will be used throughout the thesis. We use bold letters for vectors $\mathbf{x}$ and matrices $\mathbf{X}$, calligraphic letters for sets $\mathcal{X}$ and fraktur font for sets of sets $\mathfrak{X}$. We use normal font for scalars x and whenever the structure of a point in a set is not specified. Functions are also denoted using normal font, e.g. $f : \mathcal{X} \to \mathcal{Y}$, either with lower case or upper case letters. Occasionally, we will use script style for special concepts that will be repeatedly used to distinguish them from others, e.g. for the input and output sets of a function approximation problem we write $\mathscr{X}, \mathscr{Y}$ respectively.

2.2 Machine learning nomenclature

At the heart of most (if not all) machine learning algorithms lies the ability to model, manipulate and evaluate functions of various kinds. To see this it is necessary to establish the general machine learning nomenclature that we will be using in this thesis. We will take advantage of this section to introduce several notations and a common mathematical and conceptual language that will be used. The reader familiar with the conventional machine learning concepts can safely skip this section.

ML components. Every learning setup consists of the following components:

- The input set $\mathscr{X}$, i.e. a set of possible inputs. This set is usually equipped with certain structure/features that turn it into a geometric space.

- The output set $\mathscr{Y}$, i.e. a set of possible outputs. Similarly to the input space, it might be also equipped with a certain structure, e.g. $\mathscr{Y} = \mathbb{R}^d$ for regression problems, but it might also be an unstructured set, e.g. a labelling in the case of classification.

- The *task* f^* that we wish to solve, which is a mapping between inputs and outputs: $f^* : \mathscr{X} \to \mathscr{Y}$. In the *supervised learning* setup the term *ground-truth* function is most commonly used. We will use these two terms interchangeably.

- The *hypothesis class* $\mathscr{H}$. A common scenario is when we don't know how to solve the task with a sequence of algorithmic steps that will provably yield the correct answer for every possible input x, or doing so is computationally intractable. To this end, we seek to predict an *approximate* solution h. The set of all candidate solutions comprises $\mathscr{H}$.

- The *data distribution* p over $\mathscr{X}$,[1]

- The *experience* we collect, which typically comes in the form of a *training dataset* $\mathfrak{D} = \{s_i\}_{i=1}^{|\mathfrak{D}|}$. This component is what discerns machine learning algorithms from other algorithmic solutions. In particular, the approximate solution here is selected in a *data-driven* manner, i.e. using the experience we collect, instead of e.g. a heuristic. In several machine learning problems, the experience comes with the presence of *supervision*, e.g. some knowledge of the ground-truth function. For example, in *supervised learning* each datapoint is a tuple of input and output points $s_i = (x_i, y_i)$, where $y_i = f^*(x_i)$. In other problems, collectively known as *unsupervised learning*, the experience contains no knowledge of the task, i.e. $s_i = x_i$. A central assumption in ML is that the datapoints x_i are *independently and identically distributed (i.i.d.)* and their distribution is p. We will be making this assumption throughout this thesis as well.

- The learning algorithm $\mathscr{A}$, which is responsible for mapping the experience to a hypothesis i.e. $h = \mathscr{A}(\mathfrak{D})$. In some cases, the learning algorithm returns more than one hypothesis (*model ensemble*), or a distribution over hypotheses (*bayesian learning*). These methods are not discussed in this thesis (but the algorithms for graph function approximation presented here seamlessly apply to these settings as well).

[1] It is common in various textbooks to define p over $\mathscr{X} \times \mathscr{Y}$. This is useful when the mapping between inputs and outputs is non-deterministic. However, this can be reformulated by defining the input space as $\mathscr{X} \times \mathscr{Y}$, the output space as $\mathbb{R}$ and the task as a distribution over $\mathscr{X} \times \mathscr{Y}$.

- The *true loss/risk function* $\mathscr{L} : \mathscr{H} \times \mathcal{F} \times \mathcal{P} \to \mathbb{R}$, where $\mathcal{F}$ the set of all possible tasks, $\mathcal{P}$ the set of all possible distributions over $\mathscr{X}$. The loss function measures the quality of approximation of the selected hypothesis.

Learning objective. The objective of the *learner* (loosely this consists of all the algorithms and models used to infer h) is to find the hypothesis h that minimises $\mathscr{L}(h, f, p)$. But since f, and usually p, are unknown, exactly evaluating the true loss is impossible. Instead, we typically minimise an estimation $\hat{\mathscr{L}}(h, \mathfrak{D})$, commonly known as the *empirical risk*. In this case the learning algorithm becomes $\mathscr{A}(\mathfrak{D}) = \operatorname{argmin}_{h \in \mathscr{H}} \hat{\mathscr{L}}(h, \mathfrak{D})$, which is widely known as *Empirical Risk Minimisation (ERM)*. Designing a learning algorithm that yields a hypothesis, such that the true risk is close to the empirical risk with high probability (w.h.p.), is the cornerstone of ML and highlights its intimate relation with statistics. For example, in supervised learning typically we have $\mathscr{L}(h, f^*, p) = \mathbb{E}_{x \sim p}\left[L(h(x), f^*(x))\right]$, where L is the pointwise loss function, and the empirical risk is $\hat{\mathscr{L}}(h, \mathfrak{D}) = \frac{1}{|\mathfrak{D}|} \sum_{s_i = (x_i, y_i) \in \mathfrak{D}} L\left(h(x_i), y_i\right)$.

2.3 Mathematical structure

Definition 2.1 (Equivalence relation)**.** *Let $\mathcal{X}$ be a set and $\mathcal{R}_{\mathcal{X}} \subseteq \mathcal{X} \times \mathcal{X}$ a binary relation on $\mathcal{X}$, i.e. a set of ordered tuples (x_1, x_2), with $x_1, x_2 \in \mathcal{X}$. We will say that $\mathcal{R}_{\mathcal{X}}$ is an equivalence relation if the following hold $\forall\, x_1, x_2, x_3 \in \mathcal{X}$:*

- $(x_1, x_1) \in \mathcal{R}_{\mathcal{X}}$ *(reflexivity),*
- $(x_1, x_2) \in \mathcal{R}_{\mathcal{X}} \Leftrightarrow (x_2, x_1) \in \mathcal{R}_{\mathcal{X}}$ *(symmetry),*
- $(x_1, x_2) \in \mathcal{R}_{\mathcal{X}}$ and $(x_2, x_3) \in \mathcal{R}_{\mathcal{X}} \Rightarrow (x_1, x_3) \in \mathcal{R}_{\mathcal{X}}$ *(transitivity).*

If $\mathcal{R}_{\mathcal{X}}$ is an equivalence relation, will use the notation $x_1 \simeq x_2$ whenever $(x_1, x_2) \in \mathcal{R}_{\mathcal{X}}$.

An important property of equivalence relations is that they split the set $\mathcal{X}$ into a partition of disjoint subsets, where each cell of the partition is called an *equivalence class*. The equivalence class of a point $x_1 \in \mathcal{X}$ is also called the *orbit* of x_1 and is denoted with $\mathrm{Orb}_{\mathcal{X}/\simeq}(x_1) = \{x_2 \in \mathcal{X} \mid x_2 \simeq x_1\}$, while the set of all orbits/equivalence classes is called the *quotient* of the equivalence relation and is denoted with $\mathcal{X}/\simeq = \{\mathrm{Orb}_{\mathcal{X}/\simeq}(x) \mid x \in \mathcal{X}\}$.

Definition 2.2 (Group). *A group is a set $\mathcal{G}$ together with a binary operation $\cdot : \mathcal{G} \times \mathcal{G} \to \mathcal{G}$, that satisfies the following axioms:*

- $(g_1 \cdot g_2) \cdot g_3 = g_1 \cdot (g_2 \cdot g_3)$, $\forall\, g_1, g_2, g_3 \in \mathcal{G}$ *(associativity),*
- $\exists\ e \in \mathcal{G}$ *which is unique and it is called the identity element, such that* $g \cdot e = e \cdot g = g$, $\forall\, g \in \mathcal{G}$ *(identity element),*
- $\forall\ g \in \mathcal{G}$, *there exists a unique element in* $\mathcal{G}$, *denoted as* g^{-1} *and called the inverse of* g, *such that* $g \cdot g^{-1} = g^{-1} \cdot g = e$ *(inverse element).*

Groups can be used to define equivalence relations. In particular, we can use groups to define transformations of a set $\mathcal{X}$, via the notion of *group action*, i.e. a function that maps the combination of a group element and a set element to another set element. The left group action is a function $\mathcal{G} \times \mathcal{X} \to \mathcal{X}$, usually denoted with $\cdot$ as well, which needs to satisfy the following axioms: $e \cdot x = x$ and $g_2 \cdot (g_1 \cdot x) = (g_2 \cdot g_1) \cdot x$. Similarly for the right group action $\mathcal{X} \times \mathcal{G} \to \mathcal{X}$. Using this definition, we can define for each group, the set $\{(x, g \cdot x) \mid x \in \mathcal{X}, g \in \mathcal{G}\}$, which one can easily verify is an equivalence relation. Using a similar notation as before, we define the *orbits* of $\mathcal{G}$ as $\mathrm{Orb}_{\mathcal{X}/\mathcal{G}}(x) = \{g \cdot x \mid g \in \mathcal{G}\}$ and the *quotient* of the group action as $\mathcal{X}/\mathcal{G} = \{\mathrm{Orb}_{\mathcal{X}/\mathcal{G}}(x) \mid x \in \mathcal{X}\}$. Inversely, for each equivalence relation $\mathcal{R}_{\mathcal{X}}$, there exists a *group* $\mathcal{G}$ of transformations acting on $\mathcal{X}$, whose orbits are the cells of the partition induced by $\mathcal{R}_{\mathcal{X}}$.

2.4 Graph theory

In this section, we will introduce notation and definitions of some graph-theoretic concepts that will be useful throughout the thesis. We typically use the symbol G to denote a graph, with $G = \big(\mathcal{V}_G, \mathcal{E}_G, \mathbf{u}_{\mathcal{V}_G}, \mathbf{u}_{\mathcal{E}_G}\big)$, where $\mathcal{V}_G \subset \mathbb{N}$ is the *vertex set* and $\mathcal{E}_G \subseteq \mathcal{V}_G \times \mathcal{V}_G$ is the *edge set* i.e. a set of vertex connections/relations. Graphs may be represented in multiple alternative ways, one of which, is the *adjacency matrix*, that will be denoted as $\mathbf{A}(G) \in \{0,1\}^{\mathcal{V}_G \times \mathcal{V}_G}$ and it is defined as $\mathbf{A}(G)(i,j) = 1$ iff $(i,j) \in \mathcal{E}_G$. A graph might be *undirected* ($\mathbf{A}(G) = \mathbf{A}(G)^\top$) or directed and may allow or prohibit the existence of *self-loops* ($\mathbf{A}(G)(i,i) = 0, \forall i \in \mathcal{V}_G$).

Optionally, one might equip the vertices with a signal on the vertices, i.e. a function from the vertex set to a real-coordinate space $\mathbb{R}^d$, which can be written in matrix notation as $\mathbf{u}_{\mathcal{V}_G} \in \mathbb{R}^{\mathcal{V}_G \times d_v}$, or also with a signal on the edges, i.e. similarly $\mathbf{u}_{\mathcal{E}_G} \in \mathbb{R}^{\mathcal{V}_G \times \mathcal{V}_G \times d_e}$. In some cases,

one encounters the term *attribute* or *feature* instead of signal, usually when the co-domain of the signal is a discrete space, such as $\mathbb{N}^d$. We will be using these terms interchangeably. Edge-wise signals can be used to define *weighted graphs*, i.e. when a weight is assigned to each edge, where a convenient compact representation is the weighted adjacency matrix $\mathbf{A}(G) \in \mathbb{R}^{\mathcal{V}_G \times \mathcal{V}_G \times d}$. For brevity, whenever it is clear from the context we will be dropping the subscript/function argument G from all the definitions above.

Topology and metric. The edge set, or equivalently the adjacency matrix, allows the definition of a metric and therefore a topology on the vertex set $\mathcal{V}_G$. A popular option is the *shortest-path distance*, also known as *geodesic distance*. In particular, let $\mathcal{P}_G^L(i,j)$ be the set of all vertex tuples that are walks of length L in G, i.e.

$$\mathcal{P}_G^L(i,j) = \{\mathbf{i} = (i_1, i_2, \ldots, i_L) \in \mathcal{V}_G^L \mid i_1 = i,\ i_L = j,\ \big(i_k, i_{k+1}\big) \in \mathcal{E}, \forall\, k \in [L-1]\}. \tag{2.1}$$

Then the shortest-path distance is defined as:

$$d_G(i,j) = \min_{L \in [|\mathcal{V}_G|]} d_G^L(i,j) \quad \text{with} \quad d_G^L(i,j) = \begin{cases} L, & \mathcal{P}_G^L \neq \emptyset \\ +\infty, & \text{else.} \end{cases} \tag{2.2}$$

This is a proper metric, for undirected graphs with no self-loops and an adjacency matrix with positive values. For a directed matrix it is a *quasi-metric*, since it is not necessarily symmetric. Even if d_G is not a metric, it allows us to define neighbourhoods as $\mathcal{N}_\rho(i) = \{j \in \mathcal{V} \mid d_G(i,j) \leq \rho\}$. Usually, when we write $\mathcal{N}(i)$ we will be implying the immediate neighbourhood $\mathcal{N}_1(i)$.

Isomorphism and Automorphism. In general, graphs are objects with no inherent ordering. This means that the vertices of the graph can be ordered arbitrarily (and their edges accordingly) and all the graphs resulting from different orderings are considered equivalent. A formal notion that summarises the above is the concept of *isomorphism*, defined as follows:

Definition 2.3 (Isomoprhism)**.** *Let $G = \big(\mathcal{V}_G, \mathcal{E}_G\big)$ and $H = \big(\mathcal{V}_H, \mathcal{E}_H\big)$ be two graphs. We will say that G and H are isomorphic, if there exists an adjacency preserving bijective mapping $f : \mathcal{V}_G \leftrightarrow \mathcal{V}_H$, i.e. if f is one-to-one and onto (permutation) and if $\forall i,j \in \mathcal{V}_G$ the following holds: $(i,j) \in \mathcal{E}_G \Leftrightarrow \big(f(i), f(j)\big) \in \mathcal{E}_H$. Then, the mapping f is called an isomorphism between G and H and we write $G \simeq H$.*

We will also denote the set of all adjacency preserving bijections between G and H with $\mathrm{Iso}(G, H)$ (which is an empty set when $G \not\simeq H$). In adjacency matrix notation the above definition reads: there exists a permutation matrix $\mathbf{P} \in S(n) = \{\mathbf{P} \in \{0,1\}^{n\times n} \mid \mathbf{P}\mathbf{1}_{[n]} = \mathbf{1}_{[n]}, \mathbf{1}_{[n]}\mathbf{P} = \mathbf{1}_{[n]}\})$, where $n = |\mathcal{V}_G|$ such that $\mathbf{P}\mathbf{A}(G)\mathbf{P}^\top = \mathbf{A}(H)$. When graph signals are present, the bijection should be such that the signal values are also preserved, i.e. $\mathbf{P}\mathbf{u}_{\mathcal{V}_G} = \mathbf{u}_{\mathcal{V}_H}$ and $\mathbf{P}\mathbf{u}_{\mathcal{E}_G}\mathbf{P}^\top = \mathbf{u}_{\mathcal{E}_H}$ (where the matrix multiplication is applied to each channel/signal dimension). Let $\mathfrak{G}$ be the set of all possible graphs. It is not hard to see that isomorphism is an equivalence relation on $\mathfrak{G}$ (since permutations form a group). Therefore, isomorphism partitions $\mathfrak{G}$ into equivalence classes (sometimes also referred to as *isomorphism classes*) denoted as $\mathrm{Orb}_{\mathfrak{G}/\simeq}(G) = \{H \in \mathfrak{G} \mid G \simeq H\}$.

An isomorphism that maps $G = (\mathcal{V}, \mathcal{E})$ onto itself, i.e. $\exists\, f : \mathcal{V} \leftrightarrow \mathcal{V}$ such that $(i,j) \in \mathcal{E} \Leftrightarrow \big(f(i), f(j)\big) \in \mathcal{E}$,[2] is called an *automorphism*. Obviously, all graphs have the *trivial automorphism*, i.e. the identity, so of interest are graphs that have non-trivial automorphisms. The set of all automorphisms of a graph, forms a group, the *automorphism group* of the graph, denoted as $\mathrm{Aut}(G)$, which acts on the vertex set $\mathcal{V}$. The automorphism group yields a partition of the vertices into orbits

$$\mathrm{Orb}_{\mathcal{V}}(i) = \{j \in \mathcal{V} \mid \exists\, f \in \mathrm{Aut}(G) \text{ with } f(i) = j\},\ \forall i \in \mathcal{V}, \tag{2.3}$$

where we off-loaded notation replacing $\mathcal{V}/\mathrm{Aut}(G)$ with $\mathcal{V}$. Intuitively, the automorphism group contains all the internal symmetries of the graph and allows us to group the graph vertices (or edges) based on on their *structural roles*, e.g. the endpoint vertices of a path, or all the vertices of a cycle (see below for definitions and Figure 4.1 in chapter 4 for an illustration). It is useful to define the edge partition into (directed) edge orbits that the automorphism group also yields:

$$\mathrm{Orb}_{\mathcal{E}}(i,j) = \{(k,\ell) \in \mathcal{E} \mid \exists\, f \in \mathrm{Aut}(G) \text{ with } f(i) = k, f(j) = \ell\},\ \forall (i,j) \in \mathcal{E}. \tag{2.4}$$

A simpler, and less discriminative definition is the one that partitions the edges into undirected edge orbits: $\mathrm{Orb}_{\mathcal{E}}(i,j) = \big\{\{k,\ell\} \in \mathcal{E} \mid \exists\, f \in \mathrm{Aut}(G) \text{ with } \{f(i), f(j)\} = \{k,\ell\}\big\},\ \forall \{i,j\} \in \mathcal{E}$ (which makes sense only for undirected graphs). *Minor detail*: there is an alternative way to define edge orbits; one can define edge isomorphisms by defining adjacency-preserving bijections on the edge sets, where edge adjacency means that two edges have a common endpoint. Subsequently, one can define edge automorphisms and the *edge automorphism* group, which

[2] in matrix notation: $\exists\, \mathbf{P} \in S(|\mathcal{V}|)$ such that $\mathbf{P}\mathbf{A}\mathbf{P}^\top = \mathbf{A}$

turns out that it is strictly larger than the group induced by the vertex automorphism group (induced edge automorphism group) only for 3 trivial cases [Whitney, 1932].

The Graph Isomorphism (GI) problem. The GI problem, i.e. the problem of deciding if two finite graphs are isomorphic, in its general form, is still a mystery in the field of computational complexity: while GI belongs in NP, it is still unknown if deciding if two graphs are isomorphic can be done in polynomial time [Lewis, 1983]. It is therefore unknown if GI belongs either in the P or in the NP-complete complexity classes, which makes it a good candidate for being characterised as *NP-intermediate* [Ladner, 1975] (assuming that P≠NP). Currently, the algorithm with the best worst-case complexity has been proposed by Babai in [Babai, 2016] and runs in quasi-polynomial (super-polynomial) time in the number of vertices.

Examples. Below we enlist some common examples of graphs that we will encounter. Denote the vertex set with $\mathcal{V} = [n]$. *Path graphs*, denoted with P_n, are isomorphic to a graph with edge set $\mathcal{E} = \bigcup_{i=1}^{n-1} \{i, i+1\}$ (we use set notation since we are referring to undirected graphs). *Cycle graphs*, denoted with C_n, are isomorphic to a graph with edge set $\mathcal{E} = \bigcup_{i=1}^{n} \{i, (i+1) \bmod |\mathcal{V}|\}$. *Complete graphs (cliques)*, denoted with K_n, are isomorphic to a graph with edge set $\mathcal{E} = \bigcup_{i=1,j}^{n} \{i, j\}$. *Star graphs*, denoted with S_n, are isomorphic to a graph with edge set $\mathcal{E} = \bigcup_{i=2}^{n} \{1, i\}$. Observe that S_1 is a single vertex graph and S_2 is a single edge graph.

Subgraphs. A subgraph $H = (\mathcal{V}_H, \mathcal{E}_H)$ of $G = (\mathcal{V}, \mathcal{E})$ is any graph with $\mathcal{V}_H \subseteq \mathcal{V}$ and $\mathcal{E}_H \subseteq \mathcal{E} \cap (\mathcal{V}_H \times \mathcal{V}_H)$ and we write $H \subseteq G$. When $\mathcal{E}_H$ includes all the edges of G with endpoints in $\mathcal{V}_H$, i.e., $\mathcal{E}_H = \mathcal{E} \cap \big(\mathcal{V}_H \times \mathcal{V}_H\big)$, the subgraph is said to be *induced*. In the literature, sometimes induced subgraphs are called *graphlets* [Pržulj et al., 2004, Pržulj, 2007]. Another popular term in the literature is that of *network motifs* [Milo et al., 2002], which is used to refer to statistically significant subgraphs, i.e. subgraphs that appear with a frequency larger than a given threshold compared to that produced by a random graph model. For brevity, we will use this term to refer to non-induced subgraphs (or more precisely, not necessarily induced subgraphs). Given some graph α, the *subgraph isomorphism* problem amounts to finding a subgraph H of G such that $H \simeq \alpha$. We will denote the set of all subgraphs in a graph G with $\mathcal{S}(G)$ (induced or not necessarily induced - this will be clear from the context).

2.5 Graph signal processing

The term *Graph Signal Processing (GSP)* refers to the subfield of classical signal processing that deals with the modification (filtering), analysis and synthesis of signals, the domain of which is the vertex set (or edge set) of a graph. The main difference with graph ML is that typically GSP methods are not data-driven and the functions used to process the graph signals are chosen by the designer based on application-specific desiderata. Similar to the relation between ML for computer vision and audio, and signal/image processing, graph ML has been heavily influenced by GSP, especially during its early years, and thus GSP can be considered one of its predecessors. Below, we introduce certain concepts from GSP that will be useful, particularly for chapter 3.

Graph Laplacian. Let $G = (\mathcal{V}, \mathcal{E})$ be a graph. The *Laplacian operator* is a linear operator that, when applied to a graph signal $\mathbf{u} \in \mathbb{R}^{\mathcal{V} \times d}$, performs at each point of the domain a weighted averaging of the signal over the point's neighbours, i.e. it smooths out the signal.

$$(\Delta \mathbf{u})(i) = \sum_{j \in \mathcal{N}_1(i)} w(i,j)(\mathbf{u}(j) - \mathbf{u}(i)), \tag{2.5}$$

where $w(i,j)$ denotes an edge weight that varies depending on the definition of the Laplacian. Its name originates from the fact that it is the discrete approximation of the negative continuous Laplacian (e.g. in $\mathbb{R}^3$: $-\nabla^2 = -(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2})$), using the finite difference method.

In matrix notation, one may define the *graph Laplacian matrix* as $\mathbf{L} \in \mathbb{R}^{\mathcal{V} \times \mathcal{V}}$ and express the operator with matrix multiplication $\Delta \mathbf{u} = \mathbf{L}\mathbf{u}$. The Laplacian matrix is usually defined as $\mathbf{L} = \mathbf{D} - \mathbf{A}$, where $\mathbf{D}$ is the diagonal degree matrix, i.e. $\mathbf{D}(i,i) = |\mathcal{N}_1(i)|$ and $\mathbf{D}(i,j) = 0$ if $(i,j) \notin \mathcal{E}$, and $\mathbf{A}$ the adjacency matrix. Sometimes other normalised alternatives are used, such as the random walk Laplacian $\mathbf{L}^{\text{rw}} = \mathbf{I} - \mathbf{D}^{-1}\mathbf{A}$, and the symmetrically normalised Laplacian $\mathbf{L}^{\text{sym}} = \mathbf{I} - \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$. In general, the Laplacian matrix is a matrix that needs to have the following structure:

$$\begin{cases} \mathbf{L}(i,j) < 0 & \text{if } i \neq j \text{ and } (i,j) \in \mathcal{E} \\ \mathbf{L}(i,j) = 0 & \text{if } i \neq j \text{ and } (i,j) \notin \mathcal{E} \end{cases} \tag{2.6}$$

Graph Shift Operators. In the field of GSP, the Laplacian has been widely used as an analogy to *shift operators* on grids. Generally, operators that perform weighted averaging over the neighbourhood of each vertex are referred to as *graph shift operators (GSO)* [Sandryhaila and Moura, 2013, Mateos et al., 2019, Gama et al., 2020, Dasoulas et al., 2021, Isufi et al., 2022]. A GSO is a linear operator, which can be represented as a matrix $\mathbf{S} \in \mathbb{R}^{\mathcal{V}\times\mathcal{V}}$ for which the following should hold:

$$\mathbf{S}(i,j) = 0, \text{ if } i \neq j \text{ and } (i,j) \notin \mathcal{E}. \tag{2.7}$$

This definition subsumes many linear operators on graph signals, such as the generalised Laplacian, the adjacency matrix, the Bethe Hessian [Saade et al., 2014] and others.

Graph Fourier Transform. In traditional signal processing, the *Fourier transform* is omnipresent in the analysis of continuous or discrete signals. As it is widely known the Fourier transform amounts to projecting the signal at hand to a set of *basis functions*, where the output of each projection is a *frequency component*. Relevant to our discussion is the *Discrete Fourier Transform (DFT)*, i.e. the projection of a discrete signal to a finite domain of frequencies. One interpretation of the basis functions (basis vectors in this case) is that they are the *eigenvectors of any 1D circular shift operator* (we state this fact without proof - the interested reader can refer to [Bronstein et al., 2021] pp. 37-38 for a detailed analysis).

One can follow the same rationale to define the *Graph Fourier Transform (GFT)*, using graph shift operators. In particular, the GSO/Laplacian matrix is usually diagonalisable, and therefore it admits an eigendecomposition, i.e. $\mathbf{S} = \mathbf{\Phi}\mathbf{\Lambda}\mathbf{\Phi}^{-1}$, where $\mathbf{\Phi} = [\boldsymbol{\phi}_1, \ldots, \boldsymbol{\phi}_{|\mathcal{V}|}]$, the eigenvector matrix with the eigenvectors $\boldsymbol{\phi}_i \in \mathbb{R}^{\mathcal{V}\times 1}$ as columns, and $\mathbf{\Lambda} = \text{diag}(\lambda_1, \ldots, \lambda_{|\mathcal{V}|})$ the diagonal matrix of the eigenvalues. When a real symmetric Laplacian matrix is chosen as GSO, it is always diagonalisable and the eigenvector matrix is orthogonal, i.e. $\mathbf{L} = \mathbf{\Phi}\mathbf{\Lambda}\mathbf{\Phi}^T$. Therefore, the i component of the GFT for the GSO $\mathbf{S}$ is $\mathscr{F}_{\mathbf{S}}\{\mathbf{u}\}(i) = \boldsymbol{\phi}_i^\top \mathbf{u}$, or in matrix form the GFT and the IGFT (inverse GFT) are as follows:

$$\hat{\mathbf{u}} = \mathscr{F}_{\mathbf{S}}\{\mathbf{u}\} = \mathbf{\Phi}^\top \mathbf{u} \quad \text{and} \quad \mathbf{u} = \mathscr{F}_{\mathbf{S}}^{-1}\{\hat{\mathbf{u}}\} = \mathbf{\Phi}\hat{\mathbf{u}} \tag{2.8}$$

2.6 Weisfeiler-Leman tests & Graph neural networks

2.6.1 Weisfeiler-Leman test

The *Weisfeiler-Leman graph-isomorphism test* [Weisfeiler and Leman, 1968], also known as *naive vertex classification*, *1-WL*, or just *WL*, is a fast heuristic to decide if two graphs are isomorphic or not. The WL test proceeds as follows: every vertex $i \in \mathcal{V}$ is initially assigned a *colour* $c_0(i)$ (a scalar value) that is later iteratively refined by aggregating information from its neighbours:

$$c_t(i) = \mathrm{HASH}\big(c_{t-1}(i),\ \wr c_{t-1}(j) \wr_{j \in \mathcal{N}_1(i)}\big), \tag{2.9}$$

where $\wr \cdot \wr$ denotes a multiset (a set that allows element repetitions). The WL algorithm terminates when the colours stop changing, and outputs a histogram of colours $\wr c_\infty(i) \wr_{i \in \mathcal{V}}$. Two graphs with different histograms are non-isomorphic; if the histograms are identical, the graphs are possibly, but not necessarily, isomorphic. Note that the neighbour aggregation in the WL test is a form of message passing, and as we will see in the next section 2.6.3, GNNs are the learnable analogue. As shown by the seminal works of [Xu et al., 2019, Morris et al., 2019] GNNs are at most as expressive as the WL test, i.e. the class of non-isomorphic graphs that can be distinguished by a GNN is a subset or equal to that of the WL test.

2.6.2 The WL hierarchy

WL has been generalised to higher-order variants that comprise the *WL hierarchy*. Following the terminology introduced in [Maron et al., 2019a], we describe the so-called *Folklore WL family (k-FWL)*. Note that, in the majority of papers on GNN expressivity [Morris et al., 2019, Maron et al., 2019a, Chen et al., 2020] another family of WL tests is discussed, under the terminology *k-WL* with expressive power equal to $(k-1)$-FWL. In contrast, in most graph theory papers on graph isomorphism [Cai et al., 1992, Fürer, 2017, Arvind et al., 2019] the k-WL term is used to describe the algorithms referred to as k-FWL in GNN papers. Here, we follow the k-FWL convention to align with the work mostly related to ours.

The k-FWL test assigns and updates the colours of k-tuples of vertices $\mathbf{i} = (i_1, i_2, \ldots, i_k) \in \mathcal{V}^k$ instead of single vertices $\in \mathcal{V}$. Initially, each tuple is assigned a colour $c_0(\mathbf{i})$ based on its *isomorphism type* which can be thought of as a generalisation of isomorphism that also preserves

the ordering of the vertices in the tuple:

Definition 2.4 (Isomorphism Types)**.** *Two k-tuples* $\mathbf{i}^a = (i_1^a, i_2^a, \dots, i_k^a)$, $\mathbf{i}^b = (i_1^b, i_2^b, \dots, i_k^b)$ *will have the same isomorphism type iff:*

- $\forall\, \ell, m \in [k], \quad i_\ell^a = i_m^a \Leftrightarrow i_\ell^b = i_m^b$
- $\forall\, \ell, m \in [k], \quad i_\ell^a \sim i_m^a \Leftrightarrow i_\ell^b \sim i_m^b$, *where* $\sim$ *means that the vertices are adjacent.*

Note that this is a stronger condition than isomorphism since the mapping between the vertices of the two tuples needs to preserve order. In case the graph is employed with edge and vertex features, these need to be preserved as well (see [Chen et al., 2020]) for the extended case).

The k-WL proceeds at each iteration by refining each colour as follows:

$$c_{t+1}(\mathbf{i}) = \mathrm{HASH}\Big(c_t(\mathbf{i}),\, \wr\big(c_t(\mathbf{i}_{j,1}), c_t(\mathbf{i}_{j,2}), \dots, c_t(\mathbf{i}_{j,k})\big)\int_{j\in\mathcal{V}}\Big), \tag{2.10}$$

where $\mathbf{i}_{j,\ell} = (i_1, i_2, \dots, i_{\ell-1}, j, i_{\ell+1}, \dots, i_k)$. The multiset $\wr\big(\mathbf{i}_{j,1}, \mathbf{i}_{j,2}, \dots, \mathbf{i}_{j,k}\big)\int_{j\in\mathcal{V}}$ can be perceived as a form of generalised neighbourhood. Observe that all possible tuples in the graph store information necessary for the updates, thus each k-tuple receives information *from the entire graph*, contrary to the *local* nature of the 1-WL test.

The expressive power of higher-order variants. The $(k+1)$-FWL test is strictly stronger than k-FWL, k-FWL is as strong as $(k+1)$-WL and 2-FWL is strictly stronger than the simple 1-WL test, where by stronger we mean that the class of graphs that are indistinguishable by the said variant is smaller. Due to this fact, as we will see in section 4.7, a series of works have proposed to improve the expressivity of GNNs by mimicking the WL hierarchy (this will be extensively discussed in chapter 4).

2.6.3 Graph neural networks

The term *Graph neural network (GNN)*, loosely speaking, collectively refers to all neural networks that are designed to operate on graph-structured data. As we saw in section 1.3, graph-structured data reside in various input/output spaces, e.g. plain vertices of a fixed graph, signals defined on a fixed graph, entire graphs (possibly paired with signals defined on their vertices/edges) and others. Despite this diversity, certain neural network paradigms (or at least their backbone)

are being used in multiple problems, which partially explains why the community has settled to a common terminology.

Due to the pervasiveness of graphs in many fields of mathematics, computer science, electrical engineering, physics and the life sciences, many methods for graph processing and inference have been proposed, frequently with different starting points. Strikingly, with GNNs our field has observed a convergence of these approaches into a common framework. In particular, following the early GNN prototypes [Sperduti and Starita, 1997, Gori et al., 2005, Scarselli et al., 2008], the first methods that became popular were proposed drawing inspiration from GSP [Bruna et al., 2014, Defferrard et al., 2016, Kipf and Welling, 2017] and geometry processing [Masci et al., 2015, Boscaini et al., 2016]. Simultaneously or shortly after, ideas originating from application domains, such as chemoinformatics [Duvenaud et al., 2015, Gilmer et al., 2017], social networks [Ying et al., 2018], protein analysis and design [Fout et al., 2017] and physics [Battaglia et al., 2016, Kipf et al., 2018] were incorporated in order to introduce suitable inductive biases for certain graph distributions. To understand and explain this vast landscape, ideas from graph theory [Xu et al., 2019, Chen et al., 2019, Chen et al., 2020], group theory and tensor methods [Kondor et al., 2018, Kondor and Trivedi, 2018, Maron et al., 2019b, Maron et al., 2019c, Maron et al., 2019a, Keriven and Peyré, 2019], as well as distributed computing [Sato et al., 2019, Loukas, 2020] were put forward.

The most common design paradigm for GNNs is *message passing* and the resulting architectures are known as Message Passing Neural Networks (MPNNs). In short, MPNNs are a composition of vertex-wise *local* functions, i.e. functions that, for each vertex, aggregate information from its neighbours. Optionally, when a graph-wise representation is required, MPNNs are complemented with one or more *global* functions, i.e. functions that aggregate information from all vertices. Formally, let $G = (\mathcal{V}, \mathcal{E}, \mathbf{u}_{\mathcal{V}}, \mathbf{u}_{\mathcal{E}})$ be a graph optionally equipped with a vertex and/or edge signal $\mathbf{u}_{\mathcal{V}} \in \mathbb{R}^{\mathcal{V} \times d_v}$, $\mathbf{u}_{\mathcal{E}} \in \mathbb{R}^{\mathcal{V} \times \mathcal{V} \times d_e}$. The general form of an MPNN can be defined as follows:

$$
\begin{aligned}
\mathbf{x}_0(i) &= \mathbf{u}_{\mathcal{V}}(i), \;\; i \in \mathcal{V} \\
\mathbf{m}_t(i) &= \mathrm{AGGR}_t\Big(\wr\big(\mathbf{x}_{t-1}(i), \mathbf{x}_{t-1}(j), \mathbf{u}_{\mathcal{E}}(i,j), \mathbf{w}_{\mathcal{E}}(i,j), \mathbf{w}_{\mathcal{V}}(i), \mathbf{w}_{\mathcal{V}}(j)\big)\wr_{j \in \mathcal{N}(i)}\Big), t \in [T], i \in \mathcal{V} \\
\mathbf{x}_t(i) &= \mathrm{UP}_t\Big(\mathbf{x}_{t-1}(i), \mathbf{w}_{\mathcal{V}}(i), \mathbf{m}_t(i)\Big), t \in [T], i \in \mathcal{V} \\
\mathbf{x}^G &= \mathrm{READ}\Big(\wr \mathbf{x}_T(i) \mid i \in \mathcal{V} \wr\Big).
\end{aligned}
\tag{2.11}
$$

T denotes the number of layers and UP_t are arbitrary functions mapping vectors to vectors (e.g. MLPs). The functions AGGR_t are multiset functions (e.g. DeepSets [Zaheer et al., 2017]). Usually, they are implemented as $\mathrm{AGGR}_t(\mathcal{X}) = \bigoplus_{x\in\mathcal{X}} \mathrm{MSG}_t(x)$, where MSG_t are also arbitrary functions mapping vectors to vectors and $\bigoplus$ is a permutation invariant operator, usually the summation, the element-wise maximum etc. The neighbourhood $\mathcal{N}(i)$ is usually the 1-hop neighbourhood $\mathcal{N}_1(i)$, while other alternatives include ρ-hop neighbourhoods $\mathcal{N}_\rho(i)$ or adding to the set the central vertex i, i.e. $\mathcal{N}_\rho(i) \cup \{i\}$. Finally, READ is a permutation invariant function, typically of the form $\mathrm{UP}\big(\bigoplus \mathrm{MSG}(\cdot)\big)$ as well, or just $\bigoplus$.

Finally, $\mathbf{w}_\mathcal{E} \in \mathbb{R}^{\mathcal{V}\times\mathcal{V}\times d_e^w}$, $\mathbf{w}_\mathcal{V} \in \mathbb{R}^{\mathcal{V}\times d_v^w}$ are edge and vertex weighting functions respectively (we will be also using the term features). In the literature, the weighting functions are also encountered under the term *positional encodings*. Throughout this manuscript we will see various examples, e.g. those arising in spectral filters by taking powers of the graph shift operator and those arising in patch operators (see section 3.3.3) and of course substructure encodings, that will be extensively discussed in chapter 4. Please refer to section 4.7.1 for a review of other popular examples. When the weighting functions are computed in a permutation equivariant way, the vertex-wise representations of GNNs are *permutation equivariant* and the graph-wise representation is *permutation invariant*, and therefore invariant to graph isomorphism. In many cases, permutation-sensitive positional encodings are chosen, which might be problematic in terms of generalisation, but beneficial in terms of expressive power (see sections 4.2 and 4.7.1).

To conclude, it will be useful to precisely state the time complexity of an MPNN. In particular, assume that the update functions are MLPs with maximum depth T_U, the aggregation function is of the form $= \bigoplus_{x\in\mathcal{X}} \mathrm{MLP}(x)$ and the readout function is of the form $= \mathrm{MLP}\big(\bigoplus_{x\in\mathcal{X}} x\big)$, where the MLPs have maximum depth T_M and T_R, respectively. Let d_{MLP} be the maximum width of all the MLPs. Then the time complexity can be written as:

$$O\Big(m\Big(d\big(2d+2d_v^w+d_e+d_e^w\big)+d^2\big(T_M-1\big)\Big)T_G+n\Big(d\big(2d+d_v^w\big)+d^2\big(T_U-1\big)\Big)T_G+d^2T_R\Big), \quad (2.12)$$

where $d = \max(d_v, d_{\mathrm{out}}, d_{\mathrm{MLP}})$, T_G, the number of message passing layers.

Part I

Graph Neural Networks: from graph signals to general graph spaces

3

Spiral Convolutional Networks & Neural 3D Morphable Models

3.1 Introduction

In the first of the three main chapters of the thesis we will discuss our first contribution which touches on two topics: (1) learning on a space of functions (signals) defined on a fixed graph, and (2) applications of geometric deep learning on 3D data. More precisely, in this setup, each datapoint is a function from $\mathcal{X}$ to another set, usually $\mathbb{R}^d$, and the signal domain $\mathcal{X}$ is a fixed graph embedded in the 3D space (a mesh), known as the *template*. We encounter such signals in 3D computer vision, biomedical imaging and computer graphics, where they might represent shape deformations texture, or various geometric and rendering-related signals on the mesh. In the tasks of interest, the shape signals might belong either in the input or in the output space and the most common examples are shape classification, shape segmentation, dimensionality reduction, shape synthesis etc.

Interestingly, the setup of signals on a fixed graph goes well-beyond 3D meshes and shape modelling. Similar problems can be found in bioinformatics where the fixed graph is a map of the human interactome with possible input signals of interest being drug-protein interactions and the task being a classification of each drug depending on its suitability to treat a disease [Gonzalez et al., 2021]. Another notable example is networked dynamical systems of fixed topology (e.g. from coupled oscillators to traffic flow networks and electrical grids), where the task is to predict the signal evolution or a property at a given timestep. Finally, a related application is surface Partial Differential Equations [Dziuk and Elliott, 2013], which are PDEs defined on a fixed surface and the input and output signals are the initial conditions and the

solution of the PDE respectively. When the surface is discredited into a fixed mesh, this problem belongs in the same category as the above.

In the chapter that follows, we will first study the principles of learning on fixed graph signals, consulting the design principles that we illustrated in section 1.4. We will see that although universal approximation can be achieved in more than one way, vanilla universal approximators may have high computational complexity and weak inductive biases, which in turn can result in poor optimisation convergence or/and poor generalisation. On the other hand, many methods in the literature, inspired by *Graph Signal Processing (GSP* (see section 2.5) have limited expressive power since they consist of *isotropic* operators. Although isotropy was considered by many as a necessary evil for operators on graph signals, in this work we will argue that this is not true when the graph is fixed, since vertices can be consistently identified across different signals. We, therefore, propose to break the assumption of local permutation invariance, by assigning a consistent ordering to the vertices. Going one step further, we provide a weight-sharing mechanism that reduces the required number of parameters, based on the assumption that operators acting locally can be re-used across different regions of the graph. Specifically for a fixed 3D mesh, we achieve this using a *spiral operator*, that locally orders the mesh vertices via a spiral scan. Our ordering-based GNN findings are corroborated by subsequent works that theoretically support local [Sato et al., 2019], as well as global [Loukas, 2020] orderings, proving their improved expressive power compared to vanilla locally permutation invariant GNNs.

3.2 Learning on signals supported on a fixed graph

Problem formulation. Let $G = (\mathcal{V}, \mathcal{E})$ be a graph and $\mathcal{U} = \{u : \mathcal{V} \to \mathbb{R}^d\}$ a function space (*signal space*), where the domain of each function is the vertex set of the graph (see figure 1.2 middle). We will be interested in both the case where $\mathcal{U}$ is the input set $\mathscr{X}$ of the function that we wish to approximate and the case where $\mathcal{U}$ is the output set $\mathscr{Y}$. These tasks are frequently called in the literature as *analysis* and *synthesis* respectively. Observe that this problem falls under the function space setup, as described in section 1.3, where the function domain is a non-euclidean space. To motivate the choices made in the proposed approach, i.e. w.r.t. the hypothesis class/architecture, we will focus on each one of the main design principles that we enlisted in section 1.4.

Let us begin with the **expressive power**.

Graph signals are vectors: UAP on real coordinate spaces.

As we saw in section 1.4.3, when the domain of functions in the function space is finite, as is the case with a finite graph, we can rewrite the function space as $\mathcal{U} = \hat{\mathcal{X}}^{|\mathcal{X}|} = \left(\mathbb{R}^d\right)^{|\mathcal{V}|} = \mathbb{R}^{d\cdot|\mathcal{V}|}$. Therefore, even though the existence of the underlying graph might create confusion and initially obfuscate the nature of the problem, one can straightforwardly reason about the sufficient conditions for a function approximator to hold the *UAP* on these problems: *any function approximator that holds the UAP on real coordinate spaces (e.g. polynomials, neural networks, etc.) holds the UAP on real coordinate signals defined on a fixed graph.*

This is a statement that remains underappreciated in this setup. However, before the deep learning era, it was not uncommon to treat graph signals as vectors. For example, as we will see in section 3.4.2, a common approach that was used in our application, i.e. shape modelling, was *Principal Component Analysis (PCA)*, which operates on vectors, completely discarding the underlying graph. As we will see in the experimental section 3.5, PCA is a strong competitor and has had widespread appreciation from the community due to its simplicity. It might not be a universal approximator, since it is a linear operator with orthogonality constraints, but the above discussion makes less surprising the fact that it works well for dimensionality reduction tasks, even when the underlying domain is a graph. Given the above, a reasonable next step from PCA is to simply employ a universal approximator for real coordinate spaces.

Inductive biases.

Nevertheless, such a hypothesis class is oblivious to the underlying graph structure and it, therefore, misses the opportunity to be restricted based on assumptions stemming from the underlying topology. This can be harmful for generalisation, especially in low-data regimes, such as biomedical applications or 3D computer vision, where dataset collection is costly and potentially subject to privacy restrictions. This is where the *explicit inductive biases* that we discussed in section 1.4.1 become useful.

First off, in analogy to euclidean underlying domains, such as grids, we will consider **local** functions. This is straightforward to implement since a graph has well-defined neighbourhoods, e.g. based on its shortest-path distance (section 2.4). Therefore, we can directly define

local functions using Eq. (1.1). The locality inductive bias, although potentially helpful for generalisation, does not combat the increased computational complexity problem, as we will see below. This issue is usually addressed by *weight sharing*, i.e. by re-using some parameters in the computational model of the architecture (e.g. across different locations in the input domain). Deciding how to share parameters is usually formally taken care of by **invariance/equivariance**. In particular, every symmetry in our domain enforces certain *constraints* into our hypothesis that in turn results in a reduction in the number of parameters.

However, contrary to locality, it is unclear a-priori what symmetries of fixed graph signals to consider and what are the right assumptions one should make. To obtain inspiration, researchers studied known symmetries in euclidean domains. The most well-known example is that of **shift invariance/equivariance**, which intuitively demands that if a signal is translated over the input domain, the output signal should remain unaffected/translated equally. This is a reasonable assumption for e.g. audio and image signals, where the absolute position of a signal within a grid is typically irrelevant to most downstream tasks.

On the other hand, it is unclear how to shift a graph signal, not to mention that it is not obvious if invariance/equivariance to shifts is desirable. In fact, equivariance to many *graph shift operators* (see section 2.5) results in isotropic operators, which not only have limited expressive power but in many domains (such as 3D meshes) they are known to be inappropriate inductive biases. For example, consider a fixed 3D mesh template and graph signals representing different textures. Shifting a texture graph signal using the *Laplacian*, the most popular graph shift operator, will result in a smoothed version of the original signal, and therefore applying a shift equivariant operator will result in an equally smoothed output. However, this is inappropriate for e.g. tasks such as deblurring where the target function amounts to a sharpened version of the input signal.

Computational complexity.

Therefore, constraining our discussion to particular transformations, such as graph shifts, is of questionable relevance. However, the issue of computational complexity that we discussed in section 1.4.4 persists. Let us inspect this in more detail.

Regarding global function approximators, (that might hold the UAP), even in the best scenario where the approximator is based on linear operators (matrix multiplications), we will have to

deal with at least $O(|\mathcal{V}|)$ number of parameters (assuming that the other matrix dimensions are constant w.r.t. the size of the input - this is the case of dimensionality reduction, e.g. PCA) or $O(|\mathcal{V}|^2)$ (assuming at least one matrix dimension comparable to the size of the input - this is the case of deriving vertex-wise or hierarchical representations).[1] The time complexity will be $O(|\mathcal{V}|^\omega)$, where $1 < \omega \leq 2$, an exponent arising from the size of the matrices and the matrix multiplication algorithm used.

Regarding local function approximators, we should separate our analysis into two cases. When mapping a graph signal to a single vector, effectively summarising the signal, locality means that the target value will be a combination of local functions, resulting in a number of parameters of at least $O\left(|\mathcal{V}|^2 \cdot \max_{v\in\mathcal{V}} |\mathcal{N}(v)|\right)$, where $\mathcal{N}(v)$ the neighbourhood of each vertex contributing into the local function. This is because we have $|\mathcal{V}|$ local functions with $O\left(\max_{v\in\mathcal{V}} |\mathcal{N}(v)|\right)$ parameters and then an aggregation function with $O(|\mathcal{V}|)$ parameters. Notice that this leads to an *increase* in space complexity compared to the $O(|\mathcal{V}|)$ required when computing a global summarisation function. The same holds when mapping a single vector to a graph signal, whereby we project the vector to an initial graph signal and then proceed with local operations.

When mapping a graph signal to another graph signal, locality means that the target value per vertex will be a local function, resulting in space complexity of at least $O(|\mathcal{V}| \cdot \max_{v\in\mathcal{V}} |\mathcal{N}(v))|$, since we have $|\mathcal{V}|$ local functions with $O\left(\max_{v\in\mathcal{V}} |\mathcal{N}(v)|\right)$ parameters. Notice, that this might also result in an *increase* in space complexity compared to the $O(|\mathcal{V}|)$ required when computing vertex-wise functions globally, but the other matrix dimensions are $O(1)$ (e.g. PCA). A *decrease* in space complexity is observed when in the global counterpart, intermediate matrix dimensions are $O(|\mathcal{V}|)$.

These scalings might look acceptable for small graphs but will quickly become unbearable when dealing with real-world large-scale graphs. In fact, this is the case in our application domain of interest, where deformable shape models are in general large-scale so as to capture high-resolution details, e.g. in 3D computer vision for rendering purposes, or in 3D biomedical imaging for accurate diagnosis. This is also the case in other potential domains of interest such as signals on the human interactome and traffic flow networks.

It is therefore imperative to shift our focus to a weight-sharing mechanism, even if this doesn't result from a strict invariance/equivariance constraint. With this in mind, in the next section,

[1] In both cases we assumed that the input/output signal dimensions are $O(1)$ with respect to the size of the graph.

we will discuss how we implement weight sharing based on *local vertex orderings*.

3.3 Spiral Convolutional Networks

3.3.1 Generalising convolutions to arbitrary countable domains

As we saw in the previous section 3.2, a universally applicable assumption for equivariance in graph signals does not exist. Therefore, it might be useful to take a step back, and examine alternative perspectives for weight-sharing (or more generally for function-sharing, in the context of non-linear operators).

As a warm-up let us examine the concept of 1D-convolution. Let $u : \mathcal{V} \to \mathbb{R}$ be a graph signal; for convenience assume for now that the co-domain of the input and output signals are $\mathbb{R}$. Let $\tau : \mathcal{V} \to \mathcal{V}$ be a *shift function*, e.g. a circular i right-shift $\tau_i(j) = (i + j) \mod |\mathcal{V}|$. It is more convenient to represent these functions as matrices, i.e. the graph signal with $\mathbf{u} \in \mathbb{R}^{|\mathcal{V}|}$ and the shift with a *shift operator* $\mathbf{S} \in \{0, 1\}^{|\mathcal{V}|\times|\mathcal{V}|}$, where $\mathbf{S}(i, j) = 1$ iff $\tau(i) = j$. When shift functions are bijections, i.e. *permutations*, the shift operator will be a doubly stochastic matrix.

The 1D-convolution between u and a filter $\boldsymbol{\theta} \in \mathbb{R}^{|\mathcal{V}|}$ reads:

$$h(\mathbf{u})\big(i\big) = (\mathbf{u}*\boldsymbol{\theta})(i) = \sum_{j=0}^{|\mathcal{V}|-1} \mathbf{u}\left(\tau_i\left(j\right)\right)\boldsymbol{\theta}(j) = \sum_{j,k=0}^{|\mathcal{V}|-1} \mathbf{S}_i(j,k)\mathbf{u}(k)\boldsymbol{\theta}(j) = (\mathbf{S}_i\mathbf{u})^T\boldsymbol{\theta} = \mathbf{u}^T\mathbf{S}_i^T\boldsymbol{\theta} = \boldsymbol{\theta}^\top\mathbf{S}_i\mathbf{u}, \tag{3.1}$$

where in this case $\mathbf{S}_i$ corresponds to a circular i-right shift. Intuitively, a convolution amounts to shifting either the filter or the signal, using a (potentially different for each vertex) shift operator. Regarding alternative shift operators, i-left shifts $\tau_i(j) = (j - i) \mod |\mathcal{V}|$, will result in an equivalent notion known as *correlation*, while non-circular shifts will result in convolutions in an extended domain (e.g. $\mathbb{Z}$), which is equivalent to doing zero-padding.

We can take this rationale one step further. By examining the derivation of Eq. (3.1) we can see that the only requirement for the shift operator is to be a *permutation matrix* or in other words a vertex *(re-)ordering*. More generally, for non-linear operators we can use a shared function f parametrised by a set of parameters $\boldsymbol{\theta}$ and obtain the output signal at each point i using shift operators as follows:

$$h(\mathbf{u})(i) = f_{\boldsymbol{\theta}}(\mathbf{S}_i\mathbf{u}),\ \ \mathbf{u} \in \mathbb{R}^{|\mathcal{V}|\times d_{\text{in}}}, h(\mathbf{u}) \in \mathbb{R}^{|\mathcal{V}|\times d_{\text{out}}} \tag{3.2}$$

where we also generalised the definition to allow for vector-valued signals. Further notice that we can implement Eq. (3.2) to any countable domain (finite or infinite) since it is always possible to define bijections.

Now, the question that arises for arbitrary domains is *how to choose the shift operator of each vertex i*, which intuitively is equivalent to asking *how to transport the filter/function across different parts of the domain.* For regular grids, the shift operators arise by the shift equivariance requirement (in fact it can be shown that convolutions are the only linear operators that possess the shift equivariance property). If no such assumption can be made, the shift operators can be selected arbitrarily, but different choices will potentially affect both the expressive power and the generalisation of the model.

Sparsity. In order to further reduce the number of parameters we use operators that are a function of only at most L elements in the re-ordered signal, using a set of vertex indices $\mathcal{V}_i \subset \mathcal{V},\ |\mathcal{V}_i| \leq L$. Denote with $\mathbf{1}_{\mathcal{V}_i}$ an indicator vector with $\mathbf{1}_{\mathcal{V}_i}(j) = 1$ iff $j \in \mathcal{V}_i$, else 0. Then our *sparse* filters are defined as follows:

$$h(\mathbf{u})(i) = \boldsymbol{\theta}^\top \mathbf{S}_i \text{diag}(\mathbf{1}_{\mathcal{V}_i})\mathbf{u} \ \text{ and } \ h(\mathbf{u})(i) = f_{\boldsymbol{\theta}}\Big(\mathbf{S}_i \text{diag}(\mathbf{1}_{\mathcal{V}_i})\mathbf{u}\Big) \tag{3.3}$$

3.3.2 Spiral (shift) operators

In this work we choose an intuitive mechanism, specifically considering the topology of meshes. In particular, we opt for a vertex re-ordering using spiral trajectories inspired by [Lim et al., 2018]. For the following discussion, we assume a triangular mesh $M = (\mathcal{V}, \mathcal{E}, \mathcal{F})$, which is an *undirected* graph, which additionally to the vertex and edge set $\mathcal{V}$ and $\mathcal{E}$, is employed with a set of *faces*, i.e. connected vertex triples: $\mathcal{F} \subseteq \{\{i_1, i_2, i_3\} \mid i_1, i_2, i_3 \in \mathcal{V} \text{ s.t. } (i_\ell, i_\kappa) \in \mathcal{E},\ \forall \ell, \kappa \in \{1, 2, 3\}\}$. Additionally, the mesh is assumed to be a proper discretisation of a continuous surface, i.e. a manifold, which enforces additional constraints to the graph structure, mainly *that each edge must belong to one or two faces* (or equivalently, each pair of connected vertices should have one or two mutual neighbours), i.e.

$$1 \leq \big|\{i_a, i_b, i_c\} \in \mathcal{F} \mid \{i_a, i_b, i_c\} \cap \{i_\ell, i_\kappa\} = \{i_\ell, i_\kappa\}\big| \leq 2,\ \forall (i_\ell, i_\kappa) \in \mathcal{E}. \tag{3.4}$$

A spiral trajectory around a vertex i is a vertex re-ordering $\tau_i : \mathcal{V} \to \mathcal{V}$, that is defined by sequentially traversing each ρ-radius *ring* $\mathcal{R}_\rho(i) = \{j \in \mathcal{V} \mid d_G(i,j) = \rho\}$ around i. Algorithmically it is defined as follows:

Step 1. The initial vertex is set to i, $\tau_i(0) = i$.

Step 2. Fix two degrees of freedom: (1) the *starting point*, i.e. the second vertex as an arbitrary neighbour of i, $\tau_i(1) = i_1$, with $(i, i_1) \in \mathcal{E}$[2] and (2) the *spiral orientation* via selecting the third vertex as one of the two possible mutual neighbours, $\tau_i(2) = i_2$, with $\{(i, i_2), (i_1, i_2)\} \subset \mathcal{E}$ (recall the manifold assumption for the mesh).

Step 3. Inductively re-order all the remaining vertices in $\mathcal{R}_1(i)$ such that $\tau_i(\ell) = i_\ell$, with $\{(i, i_\ell), (i_{\ell-1}, i_\ell)\} \subset \mathcal{E}$, with $\ell \in \{3, \dots, r_1\}$, where $r_1 = |\mathcal{R}_1(i)|$.

Step 4. Set the first vertex of the 2-ring as $\tau_i(r_1 + 1) = i_{r_1+1}$, with $\{(i_1, i_{r_1+1}), (i_{r_1}, i_{r_1+1})\} \subset \mathcal{E}$ and the second vertex as $\tau_i(r_1 + 2) = i_{r_1+2}$, with $\{(i_1, i_{r_1+2}), (i_{r_1+1}, i_{r+2})\} \subset \mathcal{E}$.

Step 5. Repeat analogously from Step 3 for each ring with $2 \leq \rho$ (modify the reference vertices in Steps 3,4 using iteratively the ordered vertices in $\mathcal{R}_{\rho-1}(i)$) until all vertices are visited.

Compactly, a spiral re-ordering (in vector notation) is as follows:

$$\boldsymbol{\tau}_i = [i, \mathcal{R}_1^1(i), \mathcal{R}_1^2(i), \dots, |\mathcal{V}|], \tag{3.5}$$

where $\mathcal{R}_\rho^\ell(i)$ denotes the ℓ-th element in the ρ-ring (trivially, $\mathcal{R}_0^1(i) = i$). When the manifold assumption holds, steps 3, 4, and 5 are unambiguous for closed surfaces (for open surfaces please refer to Appendix A.1) and thus there is exactly one possible option for the subsequent vertex at each step. In figure 3.1, we illustrate a spiral ordering on a mesh (left) and a 2D grid (using a triangular meshing).

Locality. To conclude, we introduce a locality inductive bias, by considering only the first L elements of each spiral/vertex re-ordering. By definition of the spiral shift operator, these elements will be at most ρ-hops far from the vertex i, where ρ depends on L and the maximum degree of the mesh. Then our network layers follow from the weight-sharing mechanisms of Eq. (3.3). Alternatively, one may choose ρ and define L as $\max_{i \in \mathcal{V}} |\mathcal{N}_\rho(i)|$. Then, following the notation of Eq. (3.3) we have that $\mathcal{V}_i = \mathcal{N}_\rho(i)$.

[2] In our method, we chose the starting point by a heuristic, i.e. by fixing a reference vertex i^* on the mesh and choosing the starting point to be the neighbour with the shortest geodesic path to i^*. However, in practice, we did not observe any difference in experimental performance by selecting different starting points.

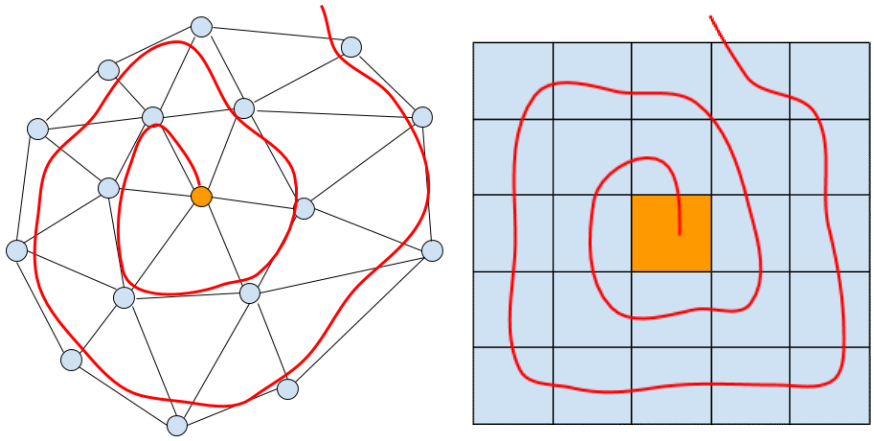


Figure 3.1: Spiral ordering on a mesh and an image patch

3.3.3 Analysis and Comparisons

In the following section, we will compare Spiral Convolutions to other GNNs in order to highlight their advantages in our learning setup and illustrate connections with architectures that have been proposed in the literature.

Comparison to [Lim et al., 2018]. First off, we focus our discussion on the original implementation of spiral trajectories on meshes, as proposed by [Lim et al., 2018]. In a nutshell, the main and instrumental difference is that in the aforementioned paper, the spiral shift operators *are not fixed*. In particular, for each vertex, the authors choose the spiral starting point (and therefore the entire shift operator) at random for every mesh signal during training and inference.

Firstly, selecting different shift operators per signal "breaks" the correspondence between signals that the common underlying domain provides us. Thus, it is unlikely that the weight sharing of Eq. (3.1) or (3.2) will be effective in approximating the complex function that will arise. Secondly, sampling shift operators can be perceived as a form of data augmentation that implicitly encourages invariance to the choice of the shift operator (it is an implicit form of *symmetrisation*). This choice biases the learning algorithm to converge to *locally permutation invariant/isotropic* solutions, which are restrictive (more on this in the next comparison) and it is precisely what we want to avoid with spiral convolutions. A minor difference is the use of a recurrent neural network in [Lim et al., 2018], which results in higher computational complexity and more laborious optimisation.

Comparison to spectral filters. Spectral convolutional operators developed in the seminal works of [Bruna et al., 2014], [Defferrard et al., 2016] and [Kipf and Welling, 2017] were among the first neural networks for graphs and to date they constitute strong baselines across multiple tasks. In fact, they have also been used in the application domain that we are concerned with in this work, i.e. deformable shape models, via *mesh autoencoders* [Ranjan et al., 2018]. However, we will show that they are actually over-restrictive in this case since they suffer from the fact that they are inherently *isotropic*.

Spectral filters rely on the *Laplacian operator*. Recall from section 2.5, that the Laplacian is a weighted averaging operator that is applied locally over each vertex neighbourhood and that in Graph Signal Processing, the Laplacian is a particular case of the so-called *graph shift operators*, i.e. linear operators that have been defined in analogy to shifts on 1D signals. Several methods have defined parametric linear operators on graph signals that are equivariant to these definitions of graph shifts. [Bruna et al., 2014] define spectral filters using the *Graph Fourier Transform* and the *convolution theorem*, which asserts that convolution in the original domain is equivalent to multiplication in the frequency domain.

In particular, let $\mathbf{L}$ be a graph shift operator, such as the Laplacian. The following discussion will be made using the symbol $\mathbf{L}$ to denote graph shift operators instead of $\mathbf{S}$, in order to avoid notation clash with the spiral shift operators that we defined in the previous section. Also, note that we will discuss one-dimensional signals $\in \mathbb{R}^{\mathcal{V}\times 1}$, but our analyses easily carry over to multi-dimensional signals $\in \mathbb{R}^{\mathcal{V}\times d}$. Assume that the graph shift operator is diagonalisable as $\mathbf{L} = \mathbf{\Phi}\mathbf{\Lambda}\mathbf{\Phi}^{-1}$.The authors of [Bruna et al., 2014] define the linear operator as follows:

$$h(\mathbf{u}) = \mathscr{F}_{\mathbf{L}}^{-1}\Big\{\boldsymbol{\theta} \odot \mathscr{F}_{\mathbf{L}}\{\mathbf{u}\}\Big\} = \mathscr{F}_{\mathbf{L}}^{-1}\Big\{\mathrm{diag}(\boldsymbol{\theta})\mathbf{\Phi}^{-1}\mathbf{u}\Big\} = \mathbf{\Phi}\mathrm{diag}(\boldsymbol{\theta})\mathbf{\Phi}^{-1}\mathbf{u}, \tag{3.6}$$

where $\boldsymbol{\theta} \in \mathbb{R}^{|\mathcal{V}|\times 1}$ the parameters of the linear operator defined directly in the frequency domain and $\odot$ the Hadamard (elementwise) product. As this operator lacks locality, subsequent works, such as [Defferrard et al., 2016, Kipf and Welling, 2017] further restricted it by setting the diagonal parameter matrix as a polynomial of degree k of the eigenvalue matrix:

$$\mathrm{diag}(\boldsymbol{\theta}) = \sum_{k=0}^{K} \theta_k \mathbf{\Lambda}^k, \tag{3.7}$$

which in turn results in a linear operator that is a polynomial of the GSO:

$$h(\mathbf{u}) = \mathbf{\Phi}\text{diag}(\boldsymbol{\theta})\mathbf{\Phi}^{-1}\mathbf{u} = \sum_{k=0}^{K} \theta_k \mathbf{\Phi}\mathbf{\Lambda}^k\mathbf{\Phi}^{-1}\mathbf{u} = \sum_{k=0}^{K} \theta_k \mathbf{L}^k\mathbf{u} \tag{3.8}$$

$$h(\mathbf{u})(i) = \boldsymbol{\theta}^\top \mathbf{W}_i\mathbf{u},\ \ \mathbf{W}_i = \begin{bmatrix} \mathbf{1}_{\{i\}} \\ \mathbf{L}(i,:) \\ \vdots \\ \mathbf{L}^K(i,:) \end{bmatrix} \tag{3.9}$$

It is easy to show, that by the definition of the GSO in (2.7) we have that if $d_G(i,j) > k$, then $\mathbf{L}^k(i,j) = 0$ [Defferrard et al., 2016, Hammond et al., 2011], and therefore the same holds for $\sum_{k=0}^{K} \theta_k \mathbf{L}^k$, which in turn implies that that linear operator of (3.8) is a local function that depends on K-hop neighbourhoods. To construct neural networks based on spectral filters, it is common to interleave linear operators as the ones in Equations (3.6) and (3.8) with non-linearities $\sigma(\cdot)$ that act vertex-wise on the signal.

Even though spectral filters are very popular and have essentially paved the path for the introduction of general GNNs and message passing, in many cases they can be very restrictive. For example,

Remark 3.1. *Let* $\mathbf{L}$ *be a GSO for which it holds that* $\exists\ i,j \in \mathcal{V}$ *such that* $\mathbf{L}^k(i,:) = \mathbf{L}^k(j,:)$, *for* $1 \leq k \leq K$. *Then, if* h *is a* K*-th order spectral filter as in* (3.8), *and* $\mathbf{u}(i) = \mathbf{u}(j)$ *we have that* $h(\mathbf{u})(i) = h(\mathbf{u})(j)$. *The result carries over to a spectral GNN, i.e. multiple spectral filters interleaved with vertex-wise non-linearities.*

This is obvious since we will have that $\mathbf{W}_i\mathbf{u} = \mathbf{W}_j\mathbf{u}$, and since $h(\mathbf{u})(i) = h(\mathbf{u})(j)$ the result inductively holds for subsequent layers. Note that the Laplacian assumptions of this remark are true for *automorphic vertices* (that are in the same orbit of the automorphism group $\text{Orb}_{\mathcal{V}}(i) = \text{Orb}_{\mathcal{V}}(j)$), i.e. pairs of vertices that are connected to the exact same vertices and are thus completely symmetric (see section 2.4).

Remark 3.2. *Let* $\mathbf{L}$ *be a GSO for which it holds that* $\exists\ i \in \mathcal{V}$ *such that* $\mathbf{L}^k(i,j) = \mathbf{L}^k(i,\ell)$, $\forall j,\ell \in \mathcal{V}$ *with* $j,\ell \in \mathcal{N}_k(i)$ *for* $1 \leq k \leq K$. *Then,* K*-th order spectral filters as in* (3.8) *are locally permutation invariant, i.e.* ***isotropic****. The results carry over to the case of multiple spectral filters interleaved with vertex-wise non-linearities.*

To see this define a permutation $\tau_i : \mathcal{V} \to \mathcal{V}$, with corresponding permutation operator $\mathbf{S}_i$ that permutes vertices based on their distance from the vertex i, i.e. $\tau_i(j) = \ell$ implies that

$d_G(i,j) = d_G(i,\ell)$. Now we have the following:

$$h(\mathbf{S}_i\mathbf{u})(i) = \sigma\Big(\sum_{k=0}^{K}\theta_k\mathbf{L}^k\mathbf{S}_i\mathbf{u}\Big)(i) = \sigma\Big(\sum_{k=0}^{K}\theta_k\sum_{j\in\mathcal{N}_k(i)}\mathbf{L}^k(i,j)\mathbf{u}\big(\tau(j)\big)\Big)$$
$$= \sigma\Big(\sum_{k=0}^{K}\theta_k\sum_{j\in\mathcal{N}_k(i)}\mathbf{L}^k\big(i,\tau(j)\big)\mathbf{u}\big(\tau(j)\big)\Big) = \sigma\Big(\sum_{k=0}^{K}\theta_k\sum_{j\in\mathcal{N}_k(i)}\mathbf{L}^k(i,j)\mathbf{u}(j)\Big) = h(\mathbf{u})(i).$$

This phenomenon is observed for example in 1st order spectral filters when using most GSOs based on the adjacency matrix (including the Laplacian), or for spectral filters of arbitrary order when working with specific categories of graphs, such as *strongly regular* graphs (see chapter 4 and [Haemers, 2000]).

Local permutation invariance is also a property of general MPNNs, such as [Xu et al., 2019]. While a necessary evil when dealing with problems on a space of graphs, where no canonical ordering can be defined, spectral filters and general MPNNs are rather weak when dealing with signals on a fixed graph. This is even more important when dealing with 3D surfaces, where the local permutation invariance property amounts to *local rotation-invariance*. On the other hand, spiral convolutional filters take advantage of the fact that on a fixed graph one can define an ordering of the neighbours of each vertex and consistently repeat it across all input signals. Consequently, our filters are in general anisotropic (unless isotropic filters are learned by the learning algorithm).

To illustrate this visually, in figure 3.2 we show an *impulse response* using a Dirac function centred on a vertex on the forehead of our template. We compare the output of a selected spectral filter (and on an arbitrary output channel) learned using the architecture of [Ranjan et al., 2018] (where the Chebyshev polynomials of [Defferrard et al., 2016] are used) vs a selected spiral convolution filter with $K = 1$. As expected, spectral filters diffuse the signal isotropically, a phenomenon which can be explained similarly to remark 3.2, i.e. due to the regularity of the connectivity of the mesh which results in having equal values of the Laplacian for all neighbouring vertices with equal distance from the vertex where the Dirac is placed.

In the general case, spectral filters may counteract local permutation invariance by using anisotropic graph shift operators (e.g. as in the case of global spectral filters defined via the graph Fourier transform), or by increasing the size of the receptive field, which may result in

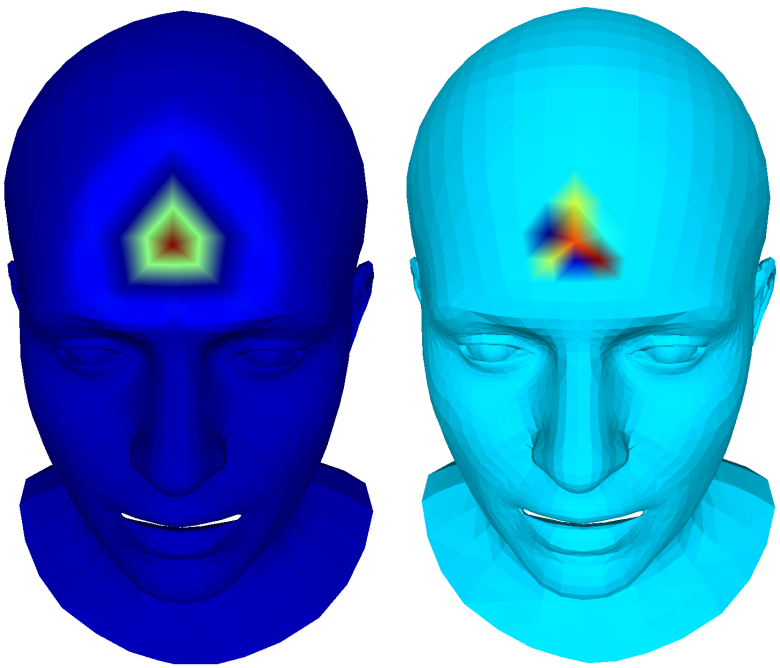


Figure 3.2: Illustration of the representations produced by ChebNet vs Spiral convolutions

an indirect symmetry breaking for most graphs based on the graph connectivity. However, even in this case, there is only one free/learnable parameter per hop k, which will unavoidably result in other restrictions in the hypothesis class, similar to remark 3.2 (although less intuitive or mathematically "clean"). It is therefore reasonable to directly break the local permutation invariance, as we do with spiral convolutions, and learn anisotropic operators, regardless of the connectivity, that do not necessarily require larger receptive fields.

We conclude our discussion with another interesting remark, which states that linear spectral operators are equivariant to linear transformations of an input signal with the corresponding GSO and its powers. Formally:

Remark 3.3. *Let* $\mathbf{L}$ *be a GSO as defined in* (2.7)*. Define the corresponding graph* k*-shifts as* $\mathbf{L}^k$ *and linear spectral filters* $h : \mathbb{R}^{|\mathcal{V}|} \to \mathbb{R}^{|\mathcal{V}|}$ *as in Eq.* (3.6)*. Then, it holds that* $h(\mathbf{L}^k\mathbf{u}) = \mathbf{L}^k h(\mathbf{u})$*.*

The verification of this remark is straightforward:

$$\begin{aligned} h(\mathbf{L}^k\mathbf{u}) &= \mathbf{\Phi}\mathrm{diag}(\boldsymbol{\theta})\mathbf{\Phi}^{-1}\mathbf{L}^k\mathbf{u} = \mathbf{\Phi}\mathrm{diag}(\boldsymbol{\theta})\mathbf{\Phi}^{-1}\mathbf{\Phi}\mathbf{\Lambda}^k\mathbf{\Phi}^{-1}\mathbf{u} = \mathbf{\Phi}\mathrm{diag}(\boldsymbol{\theta})\mathbf{\Lambda}^k\mathbf{\Phi}^{-1}\mathbf{u} \\ &= \mathbf{\Phi}\mathbf{\Lambda}^k\mathrm{diag}(\boldsymbol{\theta})\mathbf{\Phi}^{-1}\mathbf{u} = \mathbf{\Phi}\mathbf{\Lambda}^k\mathbf{\Phi}^{-1}\mathbf{\Phi}\mathrm{diag}(\boldsymbol{\theta})\mathbf{\Phi}^{-1}\mathbf{u} = \mathbf{L}^k\mathbf{\Phi}\mathrm{diag}(\boldsymbol{\theta})\mathbf{\Phi}^{-1}\mathbf{u} = \mathbf{L}^k h(\mathbf{u}). \end{aligned}$$

This result also applies to polynomial spectral filters, since they are a specific case. The remark, intuitively tells us that spectral filters are *equivariant to smoothing*, which as we saw in the

section 3.2 is harmful to certain tasks. Generalising the result to non-linear spectral filters is not straightforward, but typically in the analysis of neural nets, the behaviour of the linear counterparts provides us with valuable information for the behaviour in the non-linear regime.

Comparison to patch operators. Another related family of models of interest is that of *geometric neural networks* based on *patch operators*. In particular, by the first term, we refer to neural nets whose input space is a space of signals on manifolds, and by the second we refer to operators analogous to our vertex wise shift-operators, but in a continuous form, which, when defined locally, amount to extracting a subset of the surface and mapping it to a *chart*.

Although manifolds lack a global coordinate system, they locally resemble a euclidean space around each point. Based on this fact, the first *intrinsic* mesh/manifold convolutional architectures such as GeodesicCNN [Masci et al., 2015], ACNN [Boscaini et al., 2016] or MoNet [Monti et al., 2017], as well as more sophisticated recent ones [Zhou et al., 2020, Sun et al., 2020, de Haan et al., 2021, Weiler et al., 2021] are based on a construction of a local system of coordinates around each point $x \in \mathcal{M}$ of the manifold. In particular, the authors of these papers define a mapping $P_x : \{\mathcal{M} \to \mathbb{R}\} \to \{\mathcal{M}' \to \mathbb{R}\}$,[3] named as **patch operator**, that converts the signal on a local neighbourhood of the manifold to a signal on another manifold $\mathcal{M}'$, usually a euclidean space e.g. for a 3D surface $\mathcal{M}'$ can be $\mathbb{R}^2$, leveraging on the locally-euclidean definition of manifolds.

For a manifold signal u, a linear patch operator is defined as $P_x(u)(z) = \int_{x' \in \mathcal{M}} w_z(x, x')u(x')dx'$, $\forall x \in \mathcal{M}, z \in \mathcal{M}'$, where $w_z : \mathcal{M} \times \mathcal{M} \to \mathbb{R}$ is a (potentially parametric) weighting function. Then, convolution is defined as an inner product on the new space as follows: $\hat{u}(x) = \int_{z \in \mathcal{M}'} \theta(z)\Big(P_x(u)(z)\Big)dz$, where $\theta : \mathcal{M}' \to \mathbb{R}$. The definition of patch operators can be generalised to arbitrary input domains, including discrete ones such as graphs and meshes. In analogy to the above, a discrete domain linear patch operator $P_i : \{\mathcal{V} \to \mathbb{R}\} \to \{\mathcal{V}' \to \mathbb{R}\}$ becomes:

$$P_i(\mathbf{u})(z) = \sum_{j \in \mathcal{V}} w_z(i, j)\mathbf{u}(j), \ \forall i \in \mathcal{V}, z \in \mathcal{V}', \tag{3.10}$$

and the patch operator-based convolution:

$$\hat{\mathbf{u}}(i) = \sum_{z \in \mathcal{V}'} \boldsymbol{\theta}(z)\Big(P_i(\mathbf{u})(z)\Big) = \boldsymbol{\theta}^\top \mathbf{W}_i \mathbf{u}, \tag{3.11}$$

[3]For notation convenience, w.l.o.g. we again limit our discussion to one-dimensional signals.

where we wrote the patch operator in matrix notation as $P_i(\mathbf{u}) = \mathbf{W}_i\mathbf{u}$, $\mathbf{W}_i \in \mathbb{R}^{|\mathcal{V}'|\times|\mathcal{V}|}$. Equation (3.11) bears a striking resemblance with Equations (3.1) and (3.9) and illustrates the analogy between shift operators, spectral operators (graph shift operators) and patch operators, which are all special cases of the *GNN weighting functions* we defined in section 2.6.3. The main differences are that (1) conventional shift operators are typically permutations, as is the case with spiral shifts, and (2) conventional shift operators and graph shift operators do not contain learnable parameters while patch operators might do. Moreover, notice the differences in the dimensions of the operators: shifts have $|\mathcal{V}|$ output dimensions and locality is provided with an indicator matrix multiplied to the shift, graph shifts have $K+1$ output dimensions which directly reflect the size of the neighbourhood (K), while patch operators may have an arbitrary output dimension reflecting the size of the new space, but not necessarily locality in the original space.

Although patch operators can be very flexible, they may either result in a large number of parameters and hardness of optimisation/generalisation due to insufficient weight sharing (if $\mathbf{W}_i$ are freely parametrised), or may require to be carefully parametrised, with a manually chosen system of local coordinates. It is thus expected that these methods are more suitable in the case that our input space contains multiple manifolds (or multiple graphs, when projecting to local coordinate systems is possible). Similar conclusions can be inferred for non-linear alternatives that are akin to attention mechanisms [Verma et al., 2018, Velickovic et al., 2018], i.e. where the weighting coefficients are $w_z\big(\mathbf{u}(i), \mathbf{u}(j)\big)$ instead of $w_z(i,j)$ resulting in an operator which can be compactly written (with a slight abuse of notation) as

$$\hat{\mathbf{u}}(i) = \boldsymbol{\theta}^\top \mathbf{W}_i(\mathbf{u})\mathbf{u}, \quad \text{where} \quad \mathbf{W}_i(\mathbf{u})(z,j) = w_z\big(\mathbf{u}(i), \mathbf{u}(j)\big). \tag{3.12}$$

Permutation-sensitive GNNs. Message-passing neural networks until a few years ago, were almost universally implemented using permutation invariant functions, both at the aggregation and at the readout level (see 2.6.3). However, subsequent works, published after the paper that we discuss here, proved that permutation-sensitive GNNs have improved expressive power, although at the expense of generalisation when dealing with graph spaces, where invariance to isomorphism is necessary. We encounter two cases: *local identifiers* $\mathbf{w}_\mathcal{E} \in \mathbb{R}^{\mathcal{V}\times\mathcal{V}\times d_e^w}$ and *global identifiers* $\mathbf{w}_\mathcal{V} \in \mathbb{R}^{\mathcal{V}\times d_v^w}$, as defined in Eq. (2.11). The most relevant theoretical result to the former case is that of [Sato et al., 2019], where an architecture named *Consistent Port Number*

GNN - CPNGNN is proposed and it is shown to be strictly more expressive than permutation invariant GNNs while regarding the latter case several papers [Dasoulas et al., 2020, Loukas, 2020, Sato et al., 2021, Abboud et al., 2021] have proven their universality, as we have already mentioned.

Most relevant to our work is that of [Sato et al., 2019]. Their architecture has a similar rationale to GNNs, i.e. for each vertex, information is aggregated from its neighbours, but the aggregation is performed in a permutation-sensitive manner. First, the neighbours of vertex i are ordered, based on an ordering mechanism that the authors call consistent port numbering, and each neighbour j is assigned an integer value $\in \left[|\mathcal{N}_1(j)|\right]$, which is called a "port".[4] Then, ordered vertices, along with their port numbers are concatenated and given as input to a permutation-sensitive function.

Interestingly, this method is very similar to our ordering-based GNN and spiral convolutions. In particular, when considering only immediate neighbours $\in \mathcal{N}_1(i)$, the only difference between the two methods is that CPNGNNs use port numbers as part of the aggregation. Intuitively, port numbers allow neighbours to identify each other, a feature that we speculate might be of importance when the graph structure is relevant to the task, e.g. when solving combinatorial optimisation problems such as those studied by [Sato et al., 2019], but less relevant when the graph is fixed, as in our case. However, the expressive power of an ordering-based GNN that uses vertex-wise shift operators, instead of port numbers, has not been fully characterised yet.

3.4 Application: 3D Deformable Shapes

3.4.1 3D deep learning

Given the success of deep learning in computer vision, speech recognition, and natural language processing in the past decade, it was a natural consequence that 3D geometric data would follow. In fact, the rapid progress of 3D acquisition technology has made possible the collection of large real-world datasets, while 3D simulation and CAD (computer-aided design) environments have been widely used to create large repositories of artificial, frequently highly realistic, 3D data. This has allowed the use of data-driven methods in a variety of tasks in the fields of

[4] To facilitate understanding we deliberately omit some technical details that have to do with the *consistency* of the poor numbering/ordering procedure. The interested reader may refer to [Sato et al., 2019] for more details.

geometry processing, computer graphics and animation, 3D computer vision and 3D biomedical imaging. Characteristic examples are *shape analysis* problems, such as classification and retrieval, segmentation, object detection, scene understanding, shape matching/alignment, etc. and *shape synthesis* problems, such as 3D reconstruction, shape optimisation and style transfer, animation etc.

One of the key challenges in the processing of 3D data, and in particular 3D surfaces is that they are continuous in nature. Therefore, multiple different strategies have been implemented in order to store and represent them on a computer. In the early years of 3D deep learning, researchers were mainly focusing on *euclidean* representations of 3D data (e.g. voxels and UV maps), using traditional methods that were already present in their ML toolbox. However, as the field gained more popularity and solid geometric deep learning techniques were developed, other representations (e.g. point clouds, meshes and implicit surfaces) became the centre of attention, due to their improved abilities in approximating the continuous surface. At the time of writing this thesis, the research in this field is ongoing and fruitful and machine learning has become an organic component of many 3D processing pipelines.

3D deep learning literature. Before diving into the details of the 3D representation of interest to us in section 3.4.2, let us provide a brief overview of popular 3D deep learning methods provided for various representations. **Volumetric CNNs** were among the first 3D deep learning methods proposed for 3D voxels, for e.g. classification, retrieval, single image reconstruction and voxel synthesis [Wu et al., 2015, Maturana and Scherer, 2015, Qi et al., 2016, Girdhar et al., 2016, Sharma et al., 2016, Wu et al., 2016, Riegler et al., 2017] . Among the key drawbacks of volumetric methods are that due to the uniformity of the grid, high resolution and detailed representations come at the expense of high computational complexity and a high level of redundancy. **Point clouds** are a lightweight alternative and several methods exist for synthesis and analysis tasks [Qi et al., 2017a, Qi et al., 2017a, Fan et al., 2017, Xu et al., 2018b, Hua et al., 2018, Yang et al., 2018, Achlioptas et al., 2018, Wu et al., 2019, Thomas et al., 2019, Wang et al., 2019], while as we saw in sections 1.3 and 1.4.3, recently more sophisticated architectures that enforce invariance/equivariance to certain euclidean transformations have been proposed [Thomas et al., 2018, Fuchs et al., 2020, Shen et al., 2020]. Despite their compactness, point clouds also face a computational complexity-representation accuracy trade-off and they are less popular than meshes for realistic and high-quality 3D representation and rendering.

Implicit surfaces parameterised by neural networks (also known as *implicit neural representations* or *neural fields*) have gained immense popularity in the last few years, mainly due to their ability to accurately and naturally represent a continuous surface. Currently, they are the preferred method for storage, representation, rendering and synthesis of surfaces [Park et al., 2019, Mescheder et al., 2019, Chen and Zhang, 2019, Sitzmann et al., 2019, Atzmon and Lipman, 2020, Gropp et al., 2020, Sitzmann et al., 2020b, Tancik et al., 2020, Niemeyer et al., 2020, Sitzmann et al., 2020a, Mildenhall et al., 2021, Tancik et al., 2021, Yu et al., 2021] , while recently there have been attempts to use them also for analysis tasks [Dupont et al., 2022]. One of the main challenges for implicit functions is incorporating important inductive biases, such as symmetries and locality, and therefore generalisation to unseen surfaces is sometimes poor. This is currently an active field of research and various publications are trying to address this issue.

Finally, **meshes** are accurate and rich representations of continuous surfaces, but they are yet to become as popular as other representations, due to their intricate structure that makes them less "neural-network-friendly". The topology induced by the underlying graph of a mesh motivates the employment of local architectures, similar to GNNs. However, function approximation problems in this domain have special characteristics that can be summarised with the following challenges: (1) *Symmetries*, e.g. (a) *permutation invariance* (as with general graphs and point clouds), (b) *invariance to euclidean transformations* and (c) *discretisation invariance* (as with point clouds), i.e. our hypothesis should be invariant (or at least robust) to different discretisations of the same continuous surface. (2) *Mesh synthesis* problems are significantly harder compared to other surface representations, as one needs to generate the topology of the graph, apart from the 3D coordinates of the vertices, which introduces inherent trafe-offs between computational complexity, generalisation and permutation invariance (see section 5.7 for a discussion). At the same time, the 3D structure of meshes provides us with an opportunity: (3) *Expressivity*: the 3D positions of the vertices can be used for symmetry-breaking (see chapter 4), and provide a way out of the expressive power limitations of vanilla GNNs.

In this category, we find spectral GNNs applied to meshes, typically of fixed topology, [Bruna et al., 2014, Defferrard et al., 2016], or of arbitrary topology with the help of *functional maps* [Yi et al., 2017, Litany et al., 2017] specialised GNNs variants for meshes (of arbitrary topology) [Kostrikov et al., 2017, Feng et al., 2019, Hanocka et al., 2019, Milano et al., 2020, Smirnov and Solomon, 2021], including local-charting/patch-based ones [Masci et al., 2015, Boscaini et al., 2016, Monti et al., 2017, Poulenard and Ovsjanikov, 2018, Verma et al., 2018, Fey et al.,

2018, Zhou et al., 2020, Wiersma et al., 2020, Sun et al., 2020, de Haan et al., 2021, Weiler et al., 2021], mappings to flat domains (followed by the application of e.g. a conventional CNN) [Sinha et al., 2016, Maron et al., 2017, Haim et al., 2019] and surface heat diffusion [Sharp et al., 2022], which was shown to be robust to different discretisations.

3.4.2 Deformable shapes

Fortunately, most of the above challenges are irrelevant when dealing with *meshes of fixed topology*, which is the application domain of interest in this work and one of the most widespread paradigms of signals on a fixed graph. Even though one might be wary of the limitations of this category, in fact, these meshes, also known as *deformable shape models*, encompass a large variety of mesh distributions that are encountered in real-world problems, such as the modelling of faces [Blanz et al., 1999], human and animal bodies [Loper et al., 2015, Zuffi et al., 2017], hands [Romero et al., 2017] etc. In a nutshell, these shape categories contain shapes that are *topologically equivalent*, i.e. every pair of shapes in the category are *homeomorphic*, [5] which informally means that they have the same topological properties (e.g. number of holes, number of connected components, etc.). It is thus common and convenient in differential geometry, geometry processing and 3D computer vision, to work with *deformations* instead of shapes per se. In particular, as we have previously mentioned, typically one identifies a reference shape, a *template* and aligns to it every other shape in the category, a procedure called *shape registration* [Besl and McKay, 1992, Amberg et al., 2007, Tam et al., 2013]. When the template is discretised into a mesh, this procedure allows us to obtain a mapping from each template vertex to a point in the surface of each shape in the category. Therefore, shapes can now be defined as *deformation signals* on the template, while other shape signals can be transferred to the template as well.

Now, clearly, most of the challenges we mentioned in section 3.4.1 are no longer present. Permutation and discretisation invariances are no longer a problem and mesh synthesis reduces to synthesising only 3D vertex positions (or other signals thereon). Invariance to euclidean transformations can be guaranteed by restricting the hypothesis class, using an equivariant network, or by transforming each mesh to a *canonical pose*, which renders invariance unnecessary. The latter is common for deformable shapes, where translations are taken care of by *centroid*

[5] Formally, a homeomorphism is a continuous bijection with continuous inverse between two topological spaces.

subtraction and rotations and reflections by alignment to the template with the *Procrustes algorithm* [Gower, 1975, Goodall, 1991]. Improved expressivity with regards to vanilla GNNs is the topic of this work as we extensively discussed in section 3.3.

Statistical shape models. To date, the gold standard in modelling deformable shapes are *statistical shape models*, the origins of which date back to the seminal works of [Yuille et al., 1992, Cootes et al., 1995, Cootes et al., 2001], and are largely based on PCA. Initially, one of their main application domain was 3D biomedical imaging and biometrics, where they are still actively being used to model organs, skeletal structures, tissues, etc. [Heimann and Meinzer, 2009], but gradually they started becoming popular in the fields of 3D computer vision and computer graphics for mesh dimensionality reduction, representation learning, 3D reconstruction from a single image, 3D animation and for many synthesis and analysis tasks involving 3D avatars. Perhaps the most notable and popular example is the famous 3D Morphable Model (3DMM) of [Blanz et al., 1999] for faces. Regarding facial identity, the Large Scale Face Model (LSFM) [Booth et al., 2018] was proposed, while a large scale model of the entire head was proposed in [Ploumpis et al., 2019]. Regarding facial expression, similar methods have been presented in [Cao et al., 2014, Li et al., 2017a]. Additionally, the most well-known models for human bodies and hands, are the SCAPE [Anguelov et al., 2005] and SMPL [Loper et al., 2015] models, and the MANO [Romero et al., 2017] model respectively.

Below we briefly explain the method that lies at the heart of most statistical shape models, i.e. Principal Component Analysis (PCA) [Hotelling, 1933]. In a nutshell, PCA performs a change of basis on the data by calculating an orthonormal basis, such that the largest percentage of the variance of the data lies in only a few dimensions. This can be mathematically written as an optimisation program which can be solved in closed form and the solution can be found by performing a singular value decomposition (SVD) on the data matrix $\mathbf{U} \in \mathbb{R}^{|\mathfrak{D}| \times |\mathcal{V}| \cdot d}$, where $|\mathfrak{D}| = \{\mathbf{u}_1, \ldots, \mathbf{u}_{|\mathfrak{D}|}\}$ the training dataset. In particular, let $\mathbf{U} - \bar{\mathbf{U}} = \mathbf{V}\mathbf{\Sigma}\mathbf{\Theta}^T$ be the SVD of the centred data matrix, where $\bar{\mathbf{U}} = \frac{1}{|\mathfrak{D}|}\mathbf{1}_{[|\mathfrak{D}|]}\mathbf{1}_{[|\mathfrak{D}|]}^\top\mathbf{U}$ the column-wise average (e.g. mean shape). Then, each datapoint is transformed as follows:

$$\mathbf{z} = h_{\text{enc}}(\mathbf{u}) = \mathbf{\Theta}_L^T(\mathbf{u} - \bar{\mathbf{u}}), \tag{3.13}$$

where $\mathbf{\Theta}_L = \Big[\mathbf{\Theta}(:,1); \cdots ; \mathbf{\Theta}(:,L)\Big]$ is the truncated projection matrix, which means that we

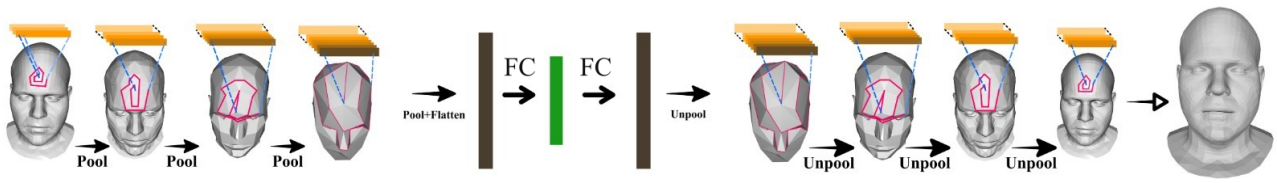


Figure 3.3: Illustration of our Neural3DMM architecture

keep only the first L principal components, effectively reducing the dimension of our data. The reconstruction from the PCA latent vector $\mathbf{z}$ is:

$$\hat{\mathbf{u}} = h_{\text{dec}}(\mathbf{z}) = \bar{\mathbf{u}} + \mathbf{\Theta}_L \mathbf{z}. \tag{3.14}$$

Keeping all principal components we obtain a lossless reconstruction of the data since $\mathbf{\Theta}^T$ is orthogonal by the definition of SVD. Observe that this method is linear, global and it only allows for signal-wise representations (unless we perform *local PCA* on vertex neighbourhoods). Also, it is important to reiterate that the model parameters (the parameters of matrix $\mathbf{\Theta} \in \mathbb{R}^{L \times |\mathcal{V}| \cdot d}$) scale with the number of vertices, while the time complexity of computing the SVD is $O\Big(|\mathfrak{D}||\mathcal{V}|d \cdot \min(|\mathfrak{D}|, |\mathcal{V}|d)\Big)$ which makes the method impractical for large meshes.

3.4.3 Neural 3D Morphable Models

Autoencoders. Following the same rationale with Morphable models, [Ranjan et al., 2018] defined a *non-linear morphable model*, where the encoder h_{enc} and decoder h_{dec} functions are neural networks. In particular, the authors, propose an architecture in the spirit of classical *deep convolutional autoencoders*, named *COMA*, by composing GNN layers (that produce vertex-wise representations), with mesh pooling layers in the encoder, and mesh unpooling layers in the decoder, that allow a considerable reduction in the number of parameters. During pooling only a subset of the vertices is retained, the selection of which is based on a popular mesh coarsening technique [Garland and Heckbert, 1997], while for unpooling the representations of the vertices that were discarded are computed as a weighted average of their neighbours using the barycentric coordinates of the closest triangle (please refer to [Ranjan et al., 2018] for more details). The optimisation objective function is the empirical mean of the reconstruction error: $\text{argmin}_{\boldsymbol{\theta},\boldsymbol{\varphi}} \frac{1}{|\mathfrak{D}|} \sum_{\mathbf{u}_i \in \mathfrak{D}} \|h^{\text{DEC}}_{\boldsymbol{\varphi}}\big(h^{\text{ENC}}_{\boldsymbol{\theta}}(\mathbf{u}_i)\big) - \mathbf{u}_i\|_p$, where $\|\cdot\|_p$ is usually the L_1 norm.

To ensure a fair comparison, in the experimental section we retain the same architecture

as in COMA and compare different non-linear operators, e.g. spectral and patch-based, as discussed in section 3.3.3, to the proposed spiral convolutions. The evaluation is done w.r.t. the reconstruction error of the model, given the dimension of the latent space, which indirectly tests the dimensionality reduction/loss compression capabilities of a model. Our architecture is dubbed *Neural 3D Morphable Models*, as a reference to the classical morphable models that we already discussed. An illustration of the architecture can be found in Fig 3.3.

Generative adversarial networks. In addition to the above, we tested our model on the task of (unconditional) mesh synthesis, i.e. learning to sample from the true mesh distribution. In particular, we develop a Generative adversarial network (GAN), trained with a distribution matching objective (1-Wasserstein distance minimisation [Arjovsky et al., 2017] - we solve the dual problem and we use the gradient penalty method [Gulrajani et al., 2017] as a surrogate to the 1-Lipschitz constraint). The generator and discriminator networks, share the same architecture with the decoder and the encoder of the autoencoder respectively. Note that due to the high resolution of the datasets considered, this is a particularly challenging statistical learning problem, and inductive biases are crucial to restrict the hypothesis class distributions.

3.5 Results

In this section, we showcase the effectiveness of our proposed method on a variety of shape datasets of fixed mesh topology. We conduct a series of ablation studies in order to compare our operator to other GNNs, by using the same autoencoder architecture. We experimented with two autoencoder variants, one architecture that follows the implementation details of [Ranjan et al., 2018] to ensure a fair comparison (*simple Neural3DMM*), and another one with increased parameter count (larger layer widths) that provides a further boost in performance (*large Neural3DMM*).

First off, we compare spiral convolutions to spectral filters (*ChebNet* [Defferrard et al., 2016]), where we observe improved performance on both the training and the test set, which is evidence for increased expressive power and generalisation. Moreover, we discuss the advantages of our method compared to patch operators and attention-based GNNs. Finally, we show the importance of the consistency of the ordering by comparing our method to different variants of the method proposed in [Lim et al., 2018]. Furthermore, we quantitatively show that our

method can yield better representations (in the sense of the reconstruction error) than the linear 3DMM and COMA, while maintaining a small parameter count and frequently allowing a more compact latent representation. Finally, we proceed with a qualitative evaluation (1) of the latent space of the autoencoder, by generating novel examples through vector space arithmetic, and (2) of the distribution learnt by the GAN, by synthesising high-resolution human faces. The mesh signals that are studied in this work are chosen as *the standardised deformations from the mean shape*, i.e. for every vertex we subtract its mean position and divide with the standard deviation, where the mean and the std are computed on the training set.

3.5.1 Datasets

COMA. The facial expression dataset from Ranjan et al. [Ranjan et al., 2018] consists of 20K+ 3D scans (5023 vertices) of twelve unique identities performing twelve types of extreme facial expressions. We used the same data split as in [Ranjan et al., 2018], i.e. 18,000+ datapoints in the training set, 100 in the validation and 2050 in the test set.

DFAUST. The dynamic human body shape dataset from Bogo et al. [Bogo et al., 2017], consists of 40K+ 3D scans (6890 vertices) of ten unique identities performing actions such as leg and arm raises, jumps, etc. We randomly split the data into 34,5K+ training datapoints, 500 for validation and 5K for testing.

MeIn3D. The 3D large-scale facial identity dataset from Booth et al. [Booth et al., 2016], consists of more than 10,000 distinct identity scans with 28K vertices which cover a wide range of gender ethnicity and age. For the subsequent experiments, the MeIn3D dataset was randomly split within demographic constraints to ensure gender, ethnic and age diversity, into 9K train and 1K test meshes.

For the quantitative experiments of sections 3.5.2 and 3.5.3 we report the estimated generalisation error on the test set, measuring the ability of the model to represent novel shapes from the same distribution as it was trained on. The metric is the euclidean distance in the 3D space (in millimetres) between corresponding input and reconstructed vertex positions, averaged over the vertices and the samples in the test set.

3.5.2 Operator comparisons

Isotropic vs Anisotropic Convolutions

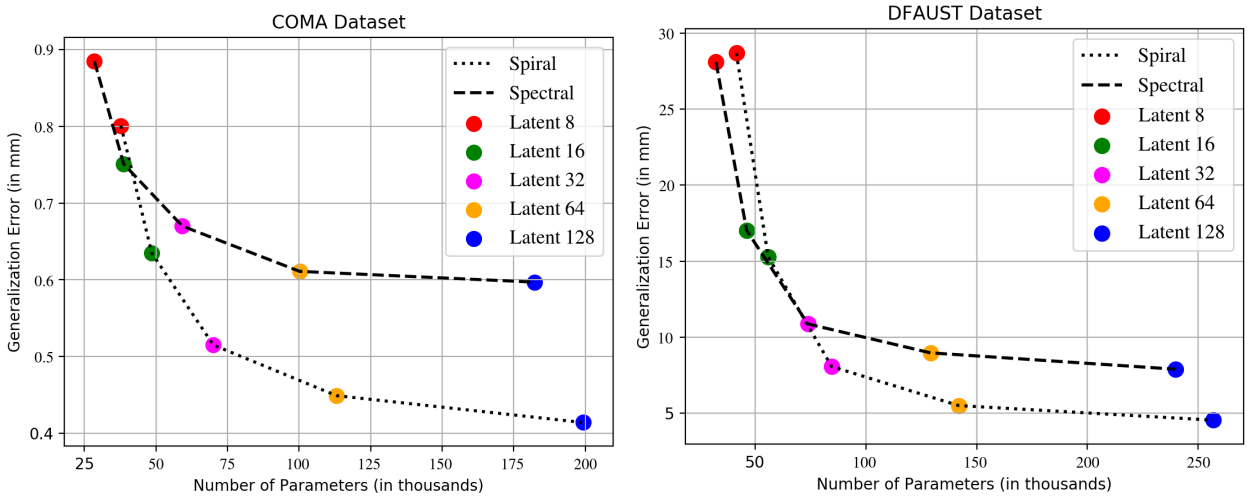


Figure 3.4: SpiralNet vs ChebNet filters

In this experiment, we compared the generalisation reconstruction error of SpiralNet against ChebNet, using the architecture of [Ranjan et al., 2018], by varying the dimension of the latent vector d_{out}. For both datasets, as clearly illustrated in Fig 3.4, spiral convolution-based autoencoders consistently outperform the counterpart for every latent dimension considered, in accordance with the analysis made in section 3.3.3. Additionally, by increasing the latent dimensions, our model's performance increases at a higher rate than its counterpart, whose performance tends to saturate early on. Notice that the number of parameters has the same scaling law as the latent size grows, but the spiral model makes better use of the added parameters. Note that the number of parameters in our case is slightly larger due to the fact that the immediate neighbours ($\rho = 1$), that determine the size of the spiral, range from 7 to 10, while the polynomials used in [Ranjan et al., 2018] go up to the 6th power of the Laplacian.

Spiral vs Attention based Convolutions

	GAT		FeastNet		MoNet		Ours
kernels	9	25	9	25	9	25	-
error	0,762	0,732	0,750	0,623	0,708	**0,583**	**0,635**
params	50K	101K	49K	98K	48K	95K	**48K**
time	12,77	15,37	9,04	9,66	10,55	10,96	**8,18**

Table 3.1: Spiral shift operators vs patch and attention-based operators.

In this experiment, we compare our method with certain state-of-the-art patch operator-based and attention-based GNNs. First, **MoNet** is the patch-operator-based model of [Monti et al., 2017], where the weighting functions (also known as heads or kernels) $\boldsymbol{w}_z(i,j)$ are Gaussian kernels defined on a pseudo-coordinate space (here we display the best-obtained results when choosing the pseudo-coordinates to be local cartesian on the fixed mesh template). The parameters of the gaussian kernels are learnable. **FeastNet** [Verma et al., 2018] and **Graph Attention** [Velickovic et al., 2018] are attention-based alternatives, where the weighting functions are learnable functions of the signal, i.e. $\boldsymbol{w}_z\big(\mathbf{u}(i),\mathbf{u}(j)\big)$. In both cases, the weighting functions are learnable softmax kernels, with minimal differences (linear and translation invariant transformation succeeded by a softmax with kernel-wise normalisation and neighbour-wise averaging in the former, and non-linear transformation succeeded by a softmax with neighbour-wise normalisation and kernel-wise averaging in the latter).

In Table 3.1, we provide results on the COMA dataset, using the simple Neural3DMM architecture with latent dimension 16. We choose the number of kernels/heads/weighting functions $|\mathcal{V}'|$ to be either 9 (equal to the size of the spiral in our method, for a fair comparison) or 25 (as in [Monti et al., 2017], to showcase the effect of a heavier parametrisation). With regards to the same order of magnitude of the number of parameters, our method outperforms its counterparts, while compared to heavier parametrisations our results are comparable or slightly worse. This shows that spiral operators (or more generally shift operators) can make more efficient use of the available learnable parameters, thus being a lightweight alternative to attention-based methods without sacrificing performance. Also, its formulation allows for fast computation (observe the per mesh inference time in ms - on a GeForce RTX 2080 Ti GPU - in Table 3.1).

Comparison to Lim et al. [Lim et al., 2018]

Ordering	random (mesh & epoch)	random (mesh)	random (epoch)	fixed
LSTM	0.888 [Lim et al., 2018]	0.880	0,996	0.792
Linear	0.829	0.825	0.951	**0.635 (Ours)**

Table 3.2: Importance of the ordering consistency.

In order to showcase how the operator behaves when the ordering is not consistent, we perform experiments under four scenarios: the original formulation of [Lim et al., 2018], where each spiral is randomly oriented for every mesh and every training/inference iteration - *random (mesh & epoch)*; choosing different orderings for every mesh, but keeping them fixed across

training/inference iterations - *random (mesh)*; choosing the same ordering across all the meshes, but sampling a new ordering at every training/inference iteration - *random (epoch)*; and fixed ordering - *Ours*. We compare the LSTM-based non-linear layer of [Lim et al., 2018] as in Eq. (3.2) and a linear layer as in Eq. (3.1). The experimental setting and architecture are the same as in the previous experiment. Observe that fixed orderings achieve significantly improved performance compared to inconsistent/stochastic ones, which substantiates the benefits of our approach. It is also interesting to notice, that even the best-performing inconsistent ordering (i.e. fixed per iteration but inconsistent across meshes) is comparable to ChebNet (see figure 3.4), which hints that inconsistent orderings may result in a collapse to isotropic operators.

3.5.3 Mesh autoencoders: quantitative results

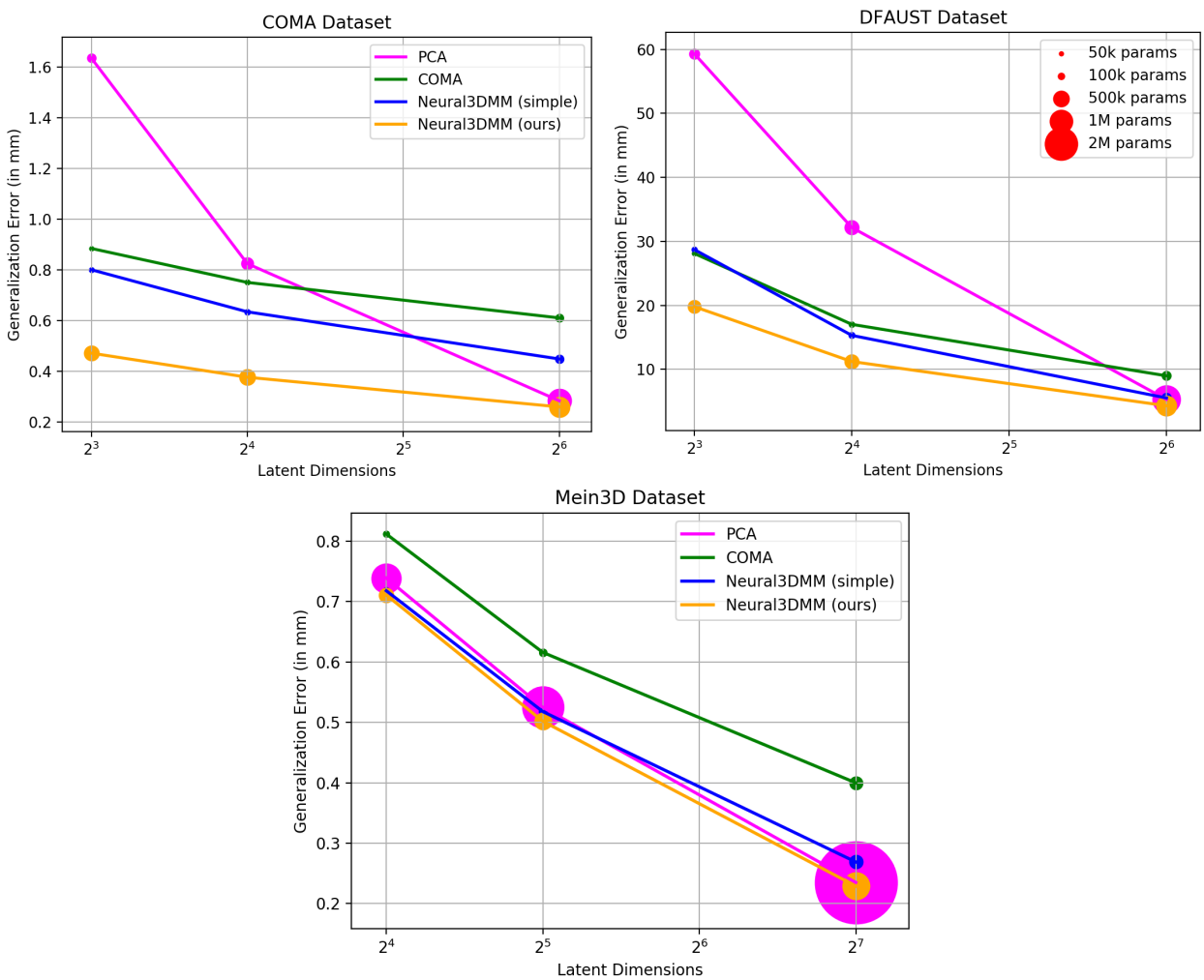


Figure 3.5: Quantitative evaluation of Neural3DMM against the baselines, w.r.t. generalisation reconstruction error and # of parameters.

In this section, we compare the following methods for different dimensions of the latent space: **PCA**, the classical 3D Morphable Model of [Blanz et al., 1999], **COMA**, the ChebNet-based Mesh Autoencoder of [Ranjan et al., 2018], **Neural3DMM (small)**, our spiral convolution autoencoder with the same architecture as in COMA, **Neural3DMM (ours)**, our spiral convolution autoencoderwith a larger parameter count (see Appendix A.1 for details). The dimensions of the latent vectors were chosen such that the principal components in PCA capture certain levels of signal variance (roughly 85%, 95% and 99% of the total variance).

As can be seen from the graphs in Fig 3.5, our Neural3DMM achieves smaller generalisation errors in every case it was tested on. For the COMA and DFAUST datasets, both GNN-based architectures outperform PCA for small latent sizes. In Fig 3.6 we compare example reconstructions of samples from the test set (latent dimension equal to 16). It is clearly visible that PCA prioritises body shape over pose resulting in the misplacement of body parts (for example see the right leg of the woman on the leftmost column). On the contrary, COMA places the vertices in approximately correct locations, but struggles to recover the fine details of the shape leading to non-smooth reconstructions and various artefacts and deformities; our model on the other hand seemingly balances these two difficult tasks resulting in quality reconstructions that preserve pose and shape.

Compared to [Ranjan et al., 2018], it is again apparent that our spiral-based autoencoder has increased capacity, which when paired with a wider architecture, makes our larger Neural3DMM outperform the other methods by a considerably large margin in terms of both generalisation and lossy compression (measured by the dimension of the latent space). Despite the fact that for higher dimensions, PCA can capture more than 99% of the total variance, thus making it a tough-to-beat baseline, our larger model still manages to outperform it. The main advantage here is the substantially smaller number of parameters of which we make use. This is clearly seen in the comparison for the MeIn3D dataset, where the large vertex count makes global methods such as PCA impractical. It is necessary to mention here, that larger latent space sizes are not necessarily desirable for an autoencoder because they might lead to poor representations that won't generalise well to downstream tasks.

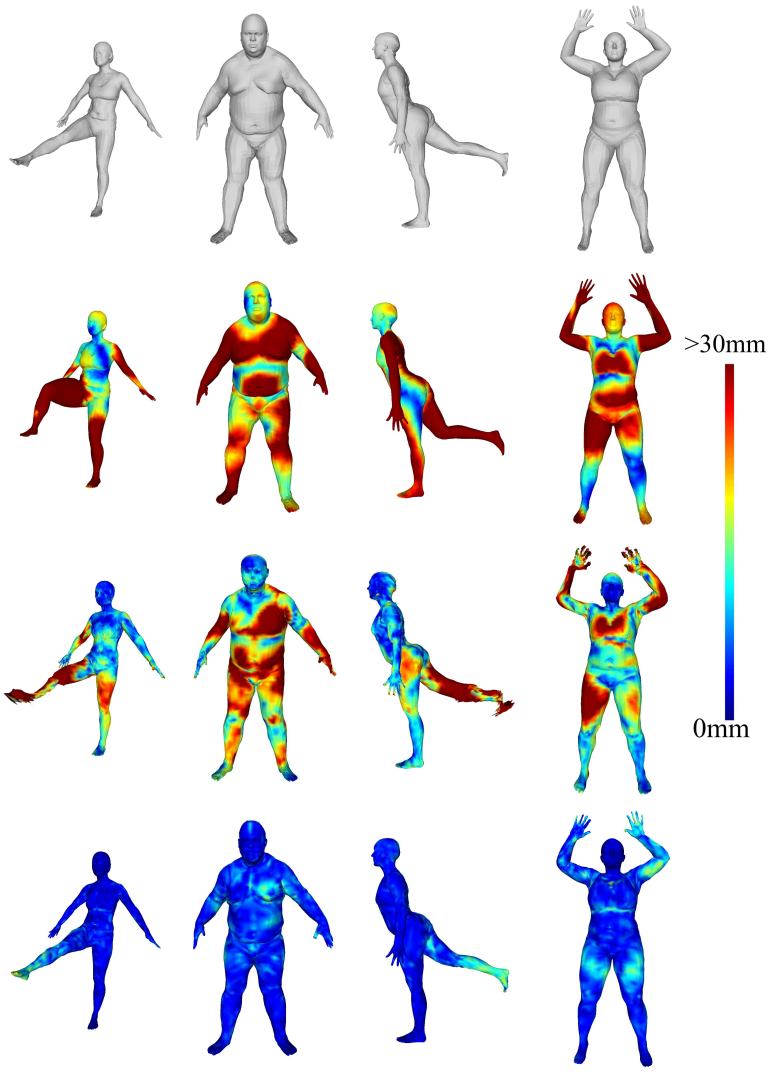
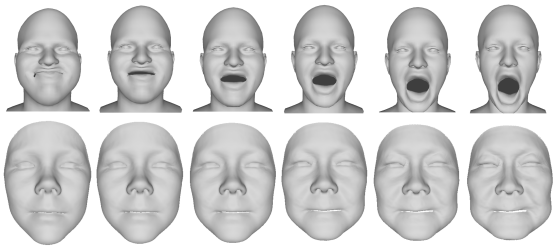


Figure 3.6: Colour coding of the per-vertex euclidean error of the reconstructions produced by PCA (2nd), COMA (3rd), and Neural3DMM (bottom). The top row shows the ground truth shapes.

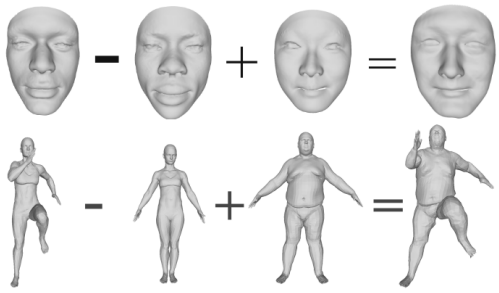
3.5.4 Qualitative results

Interpolation Fig 3.7a: We choose two sufficiently different samples $\mathbf{u_1}$ and $\mathbf{u_2}$ from our test set, encode them in their latent representations $\mathbf{z_1}$ and $\mathbf{z_2}$ and then produce intermediate encodings by sampling the line that connects them, i.e. $\mathbf{z}(t) = t\mathbf{z_1} + (1-t)\mathbf{z_2}$, where $t \in (0,1)$, and then decoding the latent vectors.

Shape Analogies Fig 3.7b: We choose three meshes $\mathbf{u}_1$, $\mathbf{u}_2$, $\mathbf{u}_3$, and synthesise a new one $\mathbf{u}_4$ such that it linearly satisfies the analogy $\mathbf{u}_1{:}\mathbf{u}_2{::}\mathbf{u}_3{:}\mathbf{u}_4$ as in [Mikolov et al., 2013], i.e. $h_{\text{enc}}(\mathbf{u}_2) - h_{\text{enc}}(\mathbf{u}_1) = h_{\text{enc}}(\mathbf{u}_4) - h_{\text{enc}}(\mathbf{u}_3)$, where we then solve for $h_{\text{enc}}(\mathbf{u}_4)$ and decode it. This way we transfer a specific characteristic using meshes from our dataset (e.g. gender transfer in the top row and pose transfer in the second).



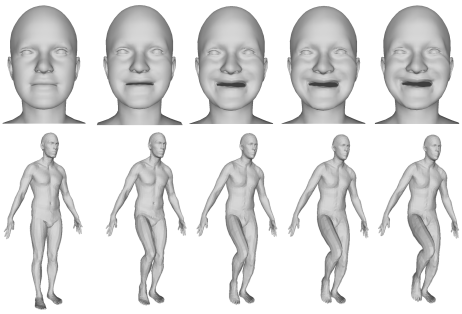
(a) Interpolations between the rightmost and leftmost shape (expressions and identities).



(b) Analogies in MeIn3D and DFAUST.

Figure 3.7: Interpolation & analogies

Extrapolation Fig 3.8a: Similarly, we decode latent representations that reside on the line defined by $\mathbf{z_1}$ and $\mathbf{z_2}$, but outside the respective line segment, i.e. $\mathbf{z} = t\mathbf{z_1} + (1-t)\mathbf{z_2}$, where $t \in (-\infty, 0) \cup (1, +\infty)$. We choose $\mathbf{z_1}$ to be our neutral expression for COMA and neutral pose for DFAUST, in order to showcase the exaggeration of a specific characteristic on the shape.



(a) Extrapolation using the two leftmost shapes (neutral and randomly selected expression/pose).

(b) Synthesied identities from our 3D GAN

Figure 3.8: Extrapolation & mesh synthesis

Face synthesis with GANs Finally, in figure 3.8b, we sampled several 3D faces using the generator of a GAN, as described in section 3.4.3. Notice that they are realistic, and following the statistics of the dataset, span a large proportion of the real distribution of human faces, in terms of ethnicity, gender and age. Compared to the most popular approach for synthesising faces, i.e. the 3DMM, our model learns to produce finer details on the facial surface compared to the samples produced by its counterpart, which are typically overly smooth. We direct the reader to Appendix A.2 to compare with samples drawn from the 3DMM's latent space.

4

Graph Substructure Networks

4.1 Introduction

In this chapter, we shift our attention to a broader and arguably more challenging function approximation problem, that of learning on a general space of graphs. This topic spans several application domains, including chemistry and materials science (molecular and material property prediction, drug discovery and molecular dynamics simulations [Wieder et al., 2020, Noé et al., 2020, Reiser et al., 2022]), bioinformatics and network neuroscience (predicting properties of biological [Muzio et al., 2021] or brain networks [Bessadok et al., 2021]), computational social sciences (social network analysis [Tan et al., 2019]), physics (e.g. classification and simulation of many-particle systems [Shlomi et al., 2020]) and (3D) geometry processing anc computer vision (analysis of 3D meshes of arbitrary topology - see section 3.4.1). Here, we focus on analysis tasks, i.e. where the input set $\mathscr{X}$ is a set of graphs and the output set $\mathscr{Y}$ is typically a real-coordinate space. Synthesis problems have different challenges and deserve a separate investigation (more on this in chapter 5), but as we will see in both cases the foundational underlying reason for the challenges in both setups, is that of *graph symmetries* and in particular (not surprisingly) the concept of *graph isomorphism (GI)*.

We will start again by discussing the design principles that one should follow, giving particular emphasis on the *expressive power*. We will see that, contrary to most problems discussed in section 1.4.3, in this problem setup, it is currently unknown if there exists a hypothesis class that holds the *universal approximation property (UAP)* and is at the same time *efficiently computable* (i.e. in polynomial time). In fact, the expressive power of several GNN has been accurately characterised, starting from the vanilla, locally-permutation invariant ones, i.e. those

defined by Eq. (2.11) excluding the weighting functions $\mathbf{w}_{\mathcal{V}}, \mathbf{w}_{\mathcal{E}}$, which have been shown to be equivalent to the *Weisfeiler-Leman* algorithm [Xu et al., 2019, Morris et al., 2019], revealing important limitations in computing certain graph properties.

The realisation of these facts has created a heated interest and a surge of publications on the topic of the expressive power of GNNs. However, as we will see, so far, there is no universally acceptable solution. This is mainly due to the fact that it is not obvious when an improvement in expressive power will also benefit *generalisation*, i.e. the objective that we actually desire to optimise for. In addition, there seems to be a fundamental tension between expressive power and either invariance/equivariance to GI (which is known to be linked with generalisation) or computational complexity, and typically an increase in the former comes with a sacrifice in one of the latter. Therefore, expressive power cannot be studied independently from the other design principles and the designer should aim to strike a balance.

The present work draws inspiration from two observations. First, from its predecessor, presented in the previous chapter 3, where we saw that isotropy is a limiting property for filters applied to graph signals. This concept was later formalised for learning on a space of graphs [Sato et al., 2019, Loukas, 2020], where it was shown that *local GNNs* improve their expressive power when endowed with symmetry-breaking mechanisms. Consequently, GNNs can retain their locality inductive bias, and their $O(|\mathcal{E}| + |\mathcal{V}|)$ (linear w.r.t. the number of edges and vertices) computational complexity, yet one might need to give up invariance/equivariance. Second, from the striking limitation of GNNs to detect and count substructures [Arvind et al., 2019, Chen et al., 2020], an ability that as we will in see in section 4.7 is crucial for generalisation in a multitude of applications, including molecular graphs, where e.g. functional groups and rings are related to chemical properties, and biological or social networks, where cliques and other substructures are related to the emergence of community structure.

Therefore, two major questions arise when designing GNN architectures: (a) Can we improve the expressive power of local GNNs, using a *symmetry-breaking mechanism (anisotropic)* that does not sacrifice *invariance/equivariance to isomorphism*? (b) Can we make GNNs aware of the *structural properties* of the graph?

In this work, we attempt to provide an answer to the above simultaneously. In brief, we propose a symmetry-breaking mechanism based on *substructure encodings*, i.e. vertex-wise or edge-wise features (or weighting functions as in Eq. (2.11)) that encode the membership (or absence

thereof) of each vertex/edge in certain substructures (*subgraph counts*) belonging to a predefined dictionary. Despite its simplicity, this modification straightforwardly brings *provable* expressive power improvements to GNNs for the vast majority of dictionaries (see section 4.4) and retains invariance/equivariance, thus addressing (a), while (b) is addressed by construction. At the same time, we achieve a controllable increase in computational complexity, since this is relegated to a preprocessing subgraph enumeration step, while the computational complexity of the GNN per se remains unaffected. Finally, by selecting the substructure dictionary, one can provide the model with different inductive biases, based on the graph distribution at hand. This will be illustrated in section 4.8, where apart from experimentally validating the improved expressive power in terms of solving hard graph isomorphism problems, we will also show that choosing substructures based on domain-specific knowledge, achieves a consistent improvement in the empirical generalisation error on several real-world benchmarks, ranging from molecular graphs to biological and social networks.

4.2 Learning on graph spaces

Problem formulation. Let $\mathfrak{G}$ be a graph space, i.e. a set of graphs $G = (\mathcal{V}, \mathcal{E}, \mathbf{u}_{\mathcal{V}}, \mathbf{u}_{\mathcal{E}})$, defined as in section 2.4. We are interested in analysis tasks, i.e. when $\mathfrak{G}$ is the input set of of the function that we wish to approximate, and typically the output set is $\mathbb{R}^{d_{\text{out}}}$ or a finite set of classes $\{0, 1, \ldots, |\mathscr{Y}|\}$. In particular, we will consider *GI-invariant target functions* $g : \mathfrak{G} \to \mathscr{Y}$, where $f^*(G_1) = f^*(G_2), \forall\, G_1 \simeq G_2$. Observe that this problem falls under the *space of spaces* setup, as described in section 1.3 (see figure 1.2 right), where each singleton is a non-euclidean space. To discuss functions in this setup, it is sometimes convenient to work with an alternative representation, i.e. that of the adjacency matrix $\mathbf{A} \in \{0, 1\}^{|\mathcal{V}| \times |\mathcal{V}|}$.

Warm-up: Treating graph spaces as vector spaces. Is this a good idea?

As a prelude, we will start with the simplest attempt to approximate functions in this setup. First, we will assume that $\mathfrak{G}$ contains graphs with a bounded number of vertices, $|\mathcal{V}| \leq |\mathcal{V}_{\max}|$. Then, one can represent each graph with the tuple $\big(|\mathcal{V}|, \mathbf{A}^{\text{ext}}, \mathbf{u}_{\mathcal{V}}^{\text{ext}}, \mathbf{u}_{\mathcal{E}}^{\text{ext}}\big)$, where we zero-padded the adjacency and the signals (e.g. $\mathbf{A}^{\text{ext}}(i, j) = \mathbf{A}(i, j),\ i, j \in \mathcal{V}$ and $\mathbf{A}^{\text{ext}}(i, j) = 0, i \notin \mathcal{V},$ or $j \notin \mathcal{V}$) and included in the tuple the size of the graph to distinguish zero-padded tuples from

tuples with isolated nodes or zero signals. Then, $\mathfrak{G}$ can be rewritten as a product space $\mathfrak{G} \subseteq [|\mathcal{V}_{\max}|] \times \{0,1\}^{|\mathcal{V}_{\max}|\times|\mathcal{V}_{\max}|} \times \mathbb{R}^{|\mathcal{V}_{\max}|\times d_v} \times \mathbb{R}^{|\mathcal{V}_{\max}|\times|\mathcal{V}_{\max}|\times d_e} \subset \mathbb{R}^{1+|\mathcal{V}_{\max}|^2+d_v|\mathcal{V}_{\max}|+d_e|\mathcal{V}_{\max}|^2}$. Therefore, one can naively attempt to approach this problem by using a function approximator that holds the UAP on real-coordinate spaces.

But, why is this probably a bad idea? First, its computational complexity is going to be unbearable even for medium-sized graphs: even for approximators based on linear operators, we will need at least $O(|\mathcal{V}_{\max}|^2)$ number of parameters (assuming, e.g. a graph-level task, and the other input dimensions, as well as the intermediate parameter matrix dimensions, being constant w.r.t. the size of the input), or worse $O(|\mathcal{V}_{\max}|^4)$ (assuming e.g. a vertex-level task, with no-dimensionality reduction by the intermediate matrices). Also, the time complexity will be $O(|\mathcal{V}_{\max}|^\omega)$, with $2 \leq \omega \leq 4$.

Second, obviously, this approach has poor inductive biases as it completely ignores the structure of the underlying space, and most importantly the symmetries induced by GI. In particular, the learning algorithm should converge to a function that is invariant/equivariant to GI, which is a quest unlikely to be achieved even with a massive amount of training data or data augmentation, even for medium-sized graphs (e.g. the generalisation bound of [Sokolic et al., 2017] informs us that a GI-sensitive neural network might need $O(|\mathcal{V}_{\max}|!)$ more training data to achieve the same generalisation error with a GI-invariant one). But, even in this case, we will still be unable to guarantee GI-invariance/equivariance, which is a crucial requirement for the reliability of the learning system in most cases. Finally, other inductive biases, such as locality are also ignored, while the upper bound to the vertex count will force us to use a possibly unreasonably large amount of computation even for small graphs, not to mention that the bounded vertex count assumption might be violated.

The pivotal role of graph isomorphism: graph discrimination

Naturally, an alternative solution is to focus on hypothesis classes that can guarantee invariance/equivariance to GI (they are *GI-invariant/equivariant*), i.e. for invariance:

$$\mathscr{H} = \{h : \mathfrak{G} \to \mathscr{Y} \mid h(G_1) = h(G_2), \forall\, G_1, G_2 \in \mathfrak{G}, \text{ with } G_1 \simeq G_2\}. \tag{4.1}$$

To define GI-equivariance we need to be more specific about the output set $\mathscr{Y}$. A common setup is vertex-wise or edge-wise tasks, where the equivalent elements in the output set are defined using the same bijective mappings as the ones used to determine isomorphisms in the input set:

$$\mathscr{H} = \{h : \mathfrak{G} \to \mathscr{Y} \mid h\big(G_1\big)\big(f(i)\big) = h(G_2)(i), \forall G_1 \simeq G_2,\ i \in \mathcal{V}_{G_1},\ f \in \mathrm{Iso}(G_1, G_2)\}. \tag{4.2}$$

Now, if we assume that $\mathfrak{G}$ is the space of all graphs and we consider a hypothesis class $\mathscr{H}$ that holds the UAP, this automatically means that $\mathscr{H}$ can be used to solve GI (for certain specific hypothesis classes, the inverse also holds as shown in [Dasoulas et al., 2020, Chen et al., 2019], i.e. for certain hypothesis classes, if they can solve GI, they are also universal). Additionally, if $\mathscr{H}$ contains only functions that can be computed in polynomial time, this would imply that GI can be solved in polynomial time. Let us formalise this claim:

Proposition 4.1. *Let $\mathcal{H} = \{h : \mathfrak{G} \to \mathbb{R}^{d_{\text{out}}} \mid h(G_1) = h(G_2), \forall\, G_1, G_2 \in \mathfrak{G},\ \text{with } G_1 \simeq G_2\}$ be a hypothesis class that is a universal approximator of GI-invariant functions, where $\mathfrak{G}$ is the space of all graphs. Then, if all $h \in \mathscr{H}$ can be computed in $O(poly(|\mathcal{V}|))$ time, where $|\mathcal{V}|$ the number of vertices of the input graph, then GI can be decided in polynomial time.*

Proof. Let $g : \mathfrak{G} \to \mathbb{R}^{d_{\text{out}}}$ be a GI-invariant function, such that $\forall\, G_1 \not\simeq G_2$, it holds that $\|g(G_1) - g(G_2)\|_p > c > 0$, where $\|\cdot\|_p$ a p-norm in $\mathbb{R}^{d_{\text{out}}}$ and c a constant. Such functions will be called *GI-complete*. Since $\mathscr{H}$ is a universal GI-invariant function approximator, there exists a function $h^* \in \mathscr{H}$ that can approximate g with precision $\epsilon < c/2$ for all possible graph inputs. Then, h^* can be used to solve GI as follows:

If two graphs are isomorphic, then $\|h^*(G_1) - h^*(G_2)\|_p = 0$ since $\mathscr{H}$ contains only GI-invariant functions. If two graphs are non-isomorphic we will have that $\|h^*(G_1) - h^*(G_2)\|_p > 0$ since:

$$\begin{aligned}
\|h^*(G_1) - h^*(G_2)\|_p &= \|g(G_1) - g(G_2) + h^*(G_1) - g(G_1) + g(G_2) - h^*(G_2)\| \\
&\geq \Big| \|g(G_1) - g(G_2)\|_p - \|h^*(G) - g(G_1) + g(G_2) - h(G_2)\|_p \Big| \\
&\geq \|g(G_1) - g(G_2)\|_p - \|h^*(G) - g(G_1) + g(G_2) - h(G_2)\|_p \\
&\geq c - \|h^*(G_1) - g(G_2)\|_p - \|g(G_2) - h^*(G_2)\|_p \\
&\geq c - 2\epsilon > 0.
\end{aligned}$$

Therefore, since h^* can be computed in polynomial time, we get the desideratum. □

However, as we discussed in section 2.4, to date, it is unknown if a polynomial time algorithm for GI exists. Thus, in light of the current evidence in computational complexity theory, no GI-invariant & universal hypothesis class containing only hypotheses that can be computed in polynomial time is currently known to exist either. In fact, given the difficulty of solving certain instances of graph isomorphism, it is customary to characterise the expressive power of GNNs, based on their ability to distinguish non-isomorphic graphs of increasing difficulty, instead of their ability to approximate certain function classes.

Computational complexity vs expressive power vs GI invariance

Currently, there is strong evidence that there is a fundamental tradeoff between invariance to GI, expressive power and computational complexity and the three concepts cannot be studied separately. As we will see in detail in section 4.7, one may choose to focus on low computational complexity and preservation of GI invariance, as the vast majority of the early GNN research. This can be done, by either employing the *locality inductive bias*, which results in locally permutation invariant aggregation functions (see Eq. (2.11)) [Xu et al., 2019], or by first characterising the *linear GI invariant/equivariant operators* and then interleaving them with non-linearities [Maron et al., 2019b], in a similar fashion to shift-invariance/equivariance and convolutions (as discussed in chapter 3). It turns out that both methods result in equally expressive hypothesis classes [Xu et al., 2019, Morris et al., 2019, Geerts, 2020], which are at most as powerful as the WL algorithm, in terms of their ability to distinguish non-isomorphic graphs.

Therefore the immediate next step in order to improve expressivity was to either sacrifice GI invariance or computational complexity (as is also the case in the present work) in favour of the other two. In the former case, several papers [Murphy et al., 2019, Dasoulas et al., 2020, Loukas, 2020, Sato et al., 2021] independently proposed GI-sensitive neural networks as a means to improve expressive power. In the latter case, the first papers that traded computational complexity for expressivity were based on higher-order variants of the WL algorithm (known as the *WL hierarchy*, or simply *k-WL tests* - see section 2.6.2 for details) and equivalent neural networks were either implemented as direct analogues [Morris et al., 2019], with higher-order tensors [Maron et al., 2019b, Maron et al., 2019c, Keriven and Peyré, 2019], or with matrix

polynomials [Maron et al., 2019a, Chen et al., 2019, Puny et al., 2023].

GNNs employed with GI-sensitive global identifiers $\mathbf{w}_{\mathcal{V}}$, were shown to hold the UAP with as few as a single vertex reordering, but achieving GI-invariance / equivariance might require $O(|\mathcal{V}_{\max}|!)$ different reorderings. Analogously, higher-order GNNs are GI-invariant / equivariant regardless of their order, but the UAP comes with an unbearable price to pay in computational complexity, with the required tensors shown to be of order $\Omega(|\mathcal{V}|^{(|\mathcal{V}|-2)/2})$, for graphs of fixed size $|\mathcal{V}|$ [Maron et al., 2019c].

Graph properties

Going one step forward, one can argue that graph discrimination is an insufficient measure of the expressive power of GNNs and instead we should be examining their ability to capture certain properties that are either known or assumed to be intimately related to real-world tasks. Some examples are properties relevant to the *connectivity* of the graph (e.g. the number of connected components and the vertex connectivity), to *distances* and *paths* between its vertices (e.g. the diameter, the Wiener index and centrality measures), to its *community structure* (e.g. the clustering coefficient and the modularity), and to the presence (and the location) of various *substructures*, such as cliques and cycles.

Yet, vanilla GNNs, and sometimes other GNNs with improved expressivity, have been shown to be unable to compute some of the properties above [Loukas, 2020, Garg et al., 2020, Chen et al., 2020, Zhang et al., 2023], in the sense that they cannot distinguish pairs of graphs for which these properties are unequal. This should not come as a surprise to us[1] given the computational complexity of solving these problems, e.g. graph partitioning problems are typically NP-hard, subgraph counting in the general case has $O(|\mathcal{V}|^k)$ complexity, where k is the size of the subgraph etc.. Therefore it would be unreasonable to expect a GNN that runs in linear time to be able to solve them in the general case.

Generalisation

The missing element from the discussion above (arguably the elephant in the room) is generalisation. In fact, it is not uncommon for architectures that are solely designed with expressivity in mind to suffer from poor generalisation or tedious optimisation, as is the case of GI-sensitive

[1] perhaps with some exceptions as the ones presented in [Zhang et al., 2023].

GNNs with random identifiers [Abboud et al., 2021]. Besides, it has been observed that in many benchmarks almost all graph pairs can be distinguished by 1-WL [Zopf, 2022] and it is reasonable to assume that the graph pairs that are indistinguishable by various GNN architectures are unlikely to be encountered for most tasks ([Babai et al., 1980] asserts that almost all non-isomorphic graphs can be distinguished by the 1-WL, using the *Erdős-Renyi* random graph model). This hints that architectures that work well in practice do not necessarily do so because of improved expressivity.

We argue here that it is not the ability (or inability) of certain GNNs to distinguish graph pairs that allows them to or prevents them from solving tasks related to certain graph properties, but their "ease of learning". In particular, although two different GNN architectures might be able to discriminate all graph pairs in a given distribution, their success in approximating the true function well enough (and in turn in generalising well) will depend on the function that they will converge to when optimised with gradient-based methods on a finite dataset. Unfortunately, our theoretical understanding here is limited and, not different from conventional neural networks, the existing results are mainly generalisation error confidence bounds [Scarselli et al., 2018, Du et al., 2019b, Garg et al., 2020, Liao et al., 2021], which are frequently vacuous, and do not allow architectures to be directly compared.

Therefore, the arguments that can be put forward are mainly intuitive and/or empirical. A simple rule-of-thumb is to constructively examine the function that a GNN needs to converge to in order to compute a relevant graph property and assess its "simplicity" (see also section 1.4.2). For example, if it can be implemented as a composition of linear (or polynomial) functions (e.g. see the proof of Theorem 4.2), this provides positive evidence that a GNN will be able to learn it using an acceptable number of training samples (although this has not been completely theoretically explained so far, arguments under simplified assumptions can be found in the *algorithmic alignment* framework of [Xu et al., 2020]).

To conclude, we should mention that graph discrimination sometimes gives us hints about the ability to generalise (e.g. when designing more expressive architectures with a certain graph property in mind, as is the case of the present work) and perhaps for this reason, although it is not our end goal, to date it largely drives the research on GNN architectures. However, the fact that in-distribution graph discrimination can be achieved even with vanilla GNNs, begs us to wonder if better generalisation can be achieved without an increase in expressivity (or

equivalently a penalty in computational complexity or GI-invariance). After all, the premise of ML is to learn good and efficient heuristics and not to solve computationally challenging problems to optimality (see section 1.4.4 for a discussion). This, to date, remains a pertinent question in our field.

4.3 Graph Substructure Networks

In the following section, we describe our proposed approach to the graph space function approximation problem, coined as *Graph Substructure Network (GSN)*. Before diving into the details, we will re-iterate the design principle trade-off that we adopted: Improved expressive power, and preservation of GI-invariance, by doing sacrifices in computational complexity. In terms of inductive biases, we aimed at retaining the locality of classical GNNs and introduced a new one: that of **substructures**, which proved beneficial for generalisation, following evidence from related fields, and was later adopted by several methods proposed in the wider GNN community.

Subgraph counts as structural features

In order to preserve locality, we adopted the general GNN formulation and aimed at achieving our objectives via a local symmetry-breaking mechanism. Recall from equation (2.11) that this can be realised using vertex-wise or edge-wise weighting functions. Below we show how these are constructed.

First, we define a *graph dictionary* $\mathcal{D} = \{\alpha_1, \alpha_2, \ldots, \alpha_{|\mathcal{D}|}\}$, i.e. a finite set of (typically small) graphs that we will refer to as *atoms*, for example, cycles or complete graphs of given sizes. For each input graph $G = (\mathcal{V}, \mathcal{E}) \in \mathfrak{G}$ and for each dictionary graph $\alpha \in \mathcal{D}$, the algorithm proceeds as follows. First, we enumerate all the subgraphs in G that are isomorphic to α. Then, for each vertex in G we enumerate all its appearances in a subgraph isomorphic to α. More precisely, to make our feature more fine-grained/discriminative, for each vertex we enumerate all the times it gets mapped to a vertex that belongs in a particular orbit (*structural role*) in α. Finally, a *vertex-wise structural feature/weighting function/identifier* (the terms will be used interchangeably from now on) $\mathbf{w}_{\mathcal{V}}(i), i \in \mathcal{V}$ for each vertex is obtained by concatenating all the subgraph counts across dictionary graphs and orbits into a single vector.

Formally, let $\mathcal{S}(G)$ be the set of all the subgraphs of G (the definition applies to either induced or not necessarily induced subgraphs). Also, $\forall \alpha = (\mathcal{V}_\alpha, \mathcal{E}_\alpha) \in \mathcal{D}$, let $\mathrm{Aut}(\alpha)$ be the automorphism group of α. This yields a partition of the vertices into vertex orbits (vertex structural roles) $\mathrm{Orb}_{\mathcal{V}_\alpha}(v)$ as defined in Eq. 2.3. Let $\mathcal{V}_\alpha/\mathrm{Aut}(\alpha) = \{\mathcal{O}_1^{\mathcal{V}_\alpha}, \mathcal{O}_2^{\mathcal{V}_\alpha}, \ldots, \mathcal{O}_{|\mathcal{V}_\alpha/\mathrm{Aut}(\alpha)|}^{\mathcal{V}_\alpha}\}$ be the quotient of the automorphism group when acting on the vertex set. Then, we define the function $\texttt{vcount}(G, \alpha)$ that enumerates all the times a vertex orbit of an atom appears on a graph vertex, for all the vertices $i \in \mathcal{V}$ and all orbits $\mathcal{O}_v^{\mathcal{V}_\alpha} \in \mathcal{V}_\alpha/\mathrm{Aut}(\alpha)$, i.e. $\texttt{vcount}(G, \alpha) \in \mathbb{N}^{|\mathcal{V}| \times |\mathcal{V}_\alpha/\mathrm{Aut}(\alpha)|}$ with:

$$\begin{aligned} &\texttt{vcount}(G, \alpha)(i, v) := \Big|\big\{H \in \mathcal{S}(G) \mid \exists f \in \mathrm{Iso}(H, \alpha) \text{ s.t. } i \in \mathcal{V}_H,\ f(i) \in \mathcal{O}_v^{\mathcal{V}_\alpha}\big\}\Big| \\ &\mathbf{w}_{\mathcal{V}} = \big[\texttt{vcount}(G, \alpha_1); \ldots; \texttt{vcount}(G, \alpha_{|\mathcal{D}|})\big] \in \mathbb{N}^{|\mathcal{V}| \times d_v^w},\ d_v^w = \sum_{\alpha_i \in \mathcal{D}} |\mathcal{V}_{\alpha_i}/\mathrm{Aut}(\alpha_i)| \end{aligned} \tag{4.3}$$

Note that there exist $|\mathrm{Aut}(H)|$ different functions f that can map a subgraph H with α when they are isomorphic; any of those can be used to determine the orbit mapping of each vertex i. Similarly, according to Eq. 2.4, we define for each edge $e \in \mathcal{E}_\alpha$ the edge orbits (edge structural roles) $\mathrm{Orb}_{\mathcal{E}_\alpha}(e)$, and the quotient of the automorphism group when acting on the edge set $\mathcal{E}_\alpha/\mathrm{Aut}(\alpha) = \{\mathcal{O}_1^{\mathcal{E}_\alpha}, \mathcal{O}_2^{\mathcal{E}_\alpha}, \ldots, \mathcal{O}_{|\mathcal{E}_\alpha/\mathrm{Aut}(\alpha)|}^{\mathcal{E}_\alpha}\}$.[2] We now define *edge-wise structural features/weighting functions* $\mathbf{w}_{\mathcal{E}}(i, j)$, $(i, j) \in \mathcal{E}$ by counting occurrences of edge orbits using the function $\texttt{ecount}(G, \alpha)$ that enumerates all the times an edge orbit of an atom appears on a graph edge, i.e. $\texttt{ecount}(G, \alpha) \in \mathbb{N}^{|\mathcal{E}| \times |\mathcal{E}_\alpha/\mathrm{Aut}(\alpha)|}$ with:

$$\begin{aligned} &\texttt{ecount}(G, a)\big((i, j), e\big) = \Big|\big\{H \in \mathcal{S}(G) \mid \exists f \in \mathrm{Iso}(H, \alpha) \text{ s.t. } (i, j) \in \mathcal{E}_H,\ \big(f(i), f(j)\big) \in \mathcal{O}_e^{\mathcal{E}_\alpha}\big\}\Big| \\ &\mathbf{w}_{\mathcal{E}} = [\texttt{ecount}(G, \alpha_1); \ldots; \texttt{ecount}(G, \alpha_{|\mathcal{D}|})] \in \mathbb{N}^{|\mathcal{E}| \times d_e^w},\ d_e^w = \sum_{\alpha_i \in \mathcal{D}} |\mathcal{E}_{\alpha_i}/\mathrm{Aut}(\alpha_i)| \end{aligned} \tag{4.4}$$

An example illustration of vertex and edge structural features is provided in figure 4.1.

[2] Observe that we use the vertex automorphism group and the directed edge orbits definition, first due to its simplicity and second, because it is an important condition for the validity of Theorem 4.4). The edge automorphism group is more discriminative only for 3 trivial cases, so we do not expect differences in performance in the vast majority of the cases.

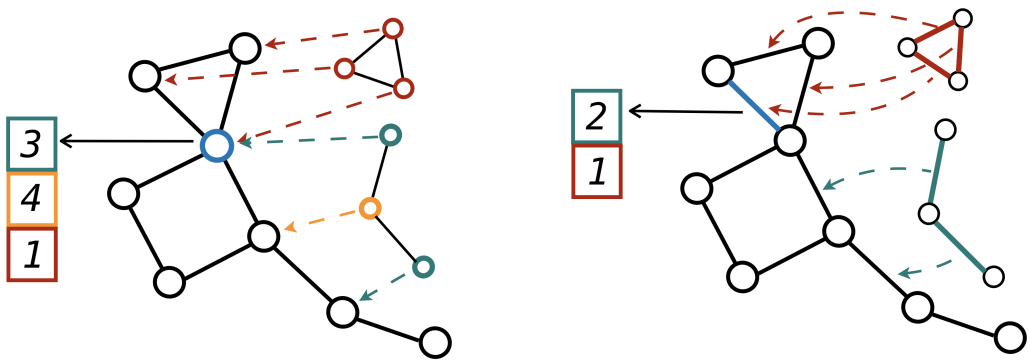


Figure 4.1: *Vertex* (left) and *edge* (right) structural features computed via induced subgraph counting for a 3-cycle C_3 and a 3-path P_3. Counts are reported for the blue vertex on the left and for the blue edge on the right. Different colours depict different orbits and for simplicity, we illustrate undirected edge orbits.

Substructure-aware message passing

Now we can directly use the general GNN formulation of Eq. (2.11), to define what we call *substructure-aware message passing*. In particular, using only vertex-wise structural features $\mathbf{w}_{\mathcal{V}}$ gives rise to vertex-wise *Graph Substructure Networks (GSN-v)* while using only edge-wise structural features $\mathbf{w}_{\mathcal{E}}$ gives rise to edge-wise *Graph Substructure Networks (GSN-e)*. These are analogous to global [Loukas, 2020, Dasoulas et al., 2020] and local [Sato et al., 2019] identifiers respectively, or absolute and relative positional encodings in language models [Shaw et al., 2018, Dai et al., 2019].

It is important to note here that contrary to identifier-based GNNs that obtain universality at the expense of GI-equivariance (since the identifiers are arbitrarily chosen with the sole requirement of being unique), GSNs are by construction GI-invariant/equivariant. This stems from the fact that the process generating our structural identifiers (i.e. subgraph isomorphism) is GI-invariant/equivariant itself (the proof is provided in the Appendix B.1.1).

4.4 How powerful are GSNs?

We now turn to the expressive power of GSNs in comparison to MPNNs and the WL tests, a key tool for the theoretical analysis of the expressivity of graph neural networks so far. Since GSN is a generalisation of MPNNs, it is easy to see that it is at least as powerful. Importantly, GSNs have the capacity to learn functions that traditional MPNNs cannot learn. The following observation derives directly from the analysis of the counting abilities of the 1-WL test [Arvind

et al., 2019] and its extension to MPNNs [Chen et al., 2020].

Theorem 4.2. *GSN is strictly more powerful than MPNN and the 1-WL test when one of the following holds:*

- *(1) For any input graph $G = (\mathcal{V}, \mathcal{E}) \in \mathfrak{G}$, the set of subgraphs $\mathcal{S}(G)$ contains* ***arbitrary subgraphs (motifs)****, i.e. $\mathcal{S}(G) = \{(\mathcal{V}_H, \mathcal{E}_H) \mid \mathcal{V}_H \subseteq \mathcal{V},\ \mathcal{E}_H \subseteq \mathcal{E} \cap \big(\mathcal{V}_H \times \mathcal{V}_H\big)$, and (2) $\exists\ \alpha \in \mathcal{D}$, which is not isomorphic to any of the star graphs, i.e. $\alpha \not\simeq S_k, \forall\ k \in \mathbb{N}, r \geq 1$. Alternatively,*

- *(1) For any input graph $G = (\mathcal{V}, \mathcal{E}) \in \mathfrak{G}$, the set of subgraphs $\mathcal{S}(G)$ contains* ***induced subgraphs (graphlets)****, i.e. $\mathcal{S}(G) = \{(\mathcal{V}_H, \mathcal{E}_H) \mid \mathcal{V}_H \subseteq \mathcal{V},\ \mathcal{E}_H = \mathcal{E} \cap \big(\mathcal{V}_H \times \mathcal{V}_H\big)$, and (2) $\exists\ \alpha \in \mathcal{D}$, which is not isomorphic to the single vertex or the single edge graphs, i.e. $\alpha \not\simeq S_1, \alpha \not\simeq S_2$.*

Proof. It is easy to see that the GSN hypothesis class contains MPNNs, and is thus at least as expressive. We can also show that GSN is at least as expressive as 1-WL by repurposing the proof of Theorem 3 in [Xu et al., 2019] (see Appendix B.1.2).

Given the first part of the proposition, in order to show that GSNs are strictly more expressive than the 1-WL test, it suffices to show that GSN can distinguish a pair of graphs that 1-WL deems isomorphic. Let α be a graph that 1-WL/MPNNs cannot count, then there exists a pair of graphs G_1, G_2 that 1-WL/MPNNs deem isomorphic, even though G_1, G_2 have a different number of subgraphs isomorphic to α. On the other hand, if $\alpha \in \mathcal{D}$, we can compute the number of isomorphic subgraphs by summing up the structural features across vertices/edges and orbits:

$$\big|\{H \in \mathcal{S}(G) \mid H \simeq \alpha\}\big| = \frac{1}{|\mathcal{V}_\alpha|} \sum_{i \in \mathcal{V}} \sum_{v \in \mathcal{V}_\alpha / \mathrm{Aut}(\alpha)} \mathbf{w}_{\mathcal{V}}(i, v) = \frac{1}{|\mathcal{E}_\alpha|} \sum_{(i,j) \in \mathcal{E}} \sum_{e \in \mathcal{E}_\alpha / \mathrm{Aut}(\alpha)} \mathbf{w}_{\mathcal{E}}(i, j, e). \quad (4.5)$$

Observe that both subgraph counting equations of (4.5) can be expressed by a GNN of Eq. (2.11). For example, for GSN-v we set $\mathbf{x}_T(i) = \frac{1}{|\mathcal{V}_\alpha|} \sum_{v \in \mathcal{V}_\alpha / \mathrm{Aut}(\alpha)} \mathbf{w}_{\mathcal{V}}(i, v)$, while for GSN-e we set $\mathbf{x}_T(i) = \frac{1}{|\mathcal{E}_\alpha|} \sum_{j \in \mathcal{N}(i)} \sum_{e \in \mathcal{E}_\alpha / \mathrm{Aut}(\alpha)|} \mathbf{w}_{\mathcal{E}}(i, j, e$. Then in both cases, we set the READ function to $\sum_{i \in \mathcal{V}} \mathbf{x}_T(i)$ (it is a permutation invariant function so it can be expressed), which gives us the desideratum. Therefore, GSN can obtain different representations $h(G_1), h(G_2)$ and deem G_1, G_2 as non-isomorphic. An example is depicted in figure 4.2 (left), where the two non-isomorphic graphs are distinguishable by GSN via e.g. cycle counting, but not by 1-WL.

To characterise the graphs for which GSN is more expressive than 1-WL/MPNNs, we can use the results of [Arvind et al., 2019], who showed that 1-WL, and consequently MPNNs, cannot count any connected subgraph apart from *forests of stars*, which include star graphs of any size (note that this contains single vertices and single edges). In addition, [Chen et al., 2020] showed that 1-WL, and consequently MPNNs, cannot count any induced connected subgraph with 3 or more vertices, i.e. any induced connected subgraph apart from single vertices and single edges. □

Universality. A natural question that emerges is what are the sufficient conditions under which GSN can solve GI. This would entail that GSN is a universal approximator of functions defined on graphs [Dasoulas et al., 2020, Chen et al., 2019] (assuming a universal function approximator in real-coordinate spaces after the READ function). To answer this, we can examine whether there exists a specific substructure collection that can completely characterise each graph. To date, we are not aware of any results in graph theory that can guarantee the reconstruction of a graph from a smaller collection of its subgraphs. However, the *Reconstruction Conjecture* [Kelly et al., 1957, Ulam, 1960] is the most closely related statement and is widely believed to be true, although still only proven for $n \leq 11$ [McKay, 1997]); it states that a graph with size $n \geq 3$ can be reconstructed from its vertex-deleted subgraphs. Consequently, (proof in the Appendix B.1.3):

Corollary 4.3. *Let $\mathfrak{G}$ be the set of all graphs of size $\leq n$ and the graph dictionary $\mathcal{D}$ be the set of all graphs of size $\leq n-1$. If the Reconstruction Conjecture holds, then GSN can distinguish all non-isomorphic graphs of size $\leq n$. Additionally, let $\mathscr{H} = \{h : \mathfrak{G} \to \mathcal{Y} \mid h = h_2 \circ h_1, h_1 \in \mathscr{H}_1, h_2 \in \mathscr{H}_2\}$ be a composite hypothesis class, where $\mathscr{H}_1 = \{h_1 : \mathfrak{G} \to \mathcal{Z}\}$ the GSN hypothesis class of GI-invariant functions with a dictionary defined as above, and $\mathscr{H}_2 = \{h : \mathcal{Z} \to \mathcal{Y}\}$ a hypothesis class that holds the UAP on functions from $\mathcal{Z}$ to $\mathcal{Y}$. Then, $\mathscr{H}$ holds the UAP on GI-invariant functions $\mathfrak{G} \to \mathcal{Y}$.*

GSN-v vs GSN-e. We can also compare the expressive power of the two proposed variants. A crucial observation that we make is that for each graph α in the dictionary, *the vertex structural identifiers can be reconstructed by the corresponding edge identifiers using linear operations*. Thus, we can show that for every GSN-v there exists a GSN-e that can simulate the behaviour of the former (proof in the Appendix B.1.4).

Theorem 4.4. *For a given subgraph collection $\mathcal{D}$, let $\mathscr{H}_{\text{GSN-}v}$ and $\mathscr{H}_{\text{GSN-}e}$ be the set of functions that can be expressed by a GSN-v and a GSN-e, respectively, with arbitrary depth and width. Then, it holds that $\mathscr{H}_{\text{GSN-}e} \supseteq \mathscr{H}_{\text{GSN-}v}$, or in other words GSN-e is at least as expressive as GSN-v.*

Comparison with higher-order WL tests. Finally, the expressive power of GSN can be compared to higher-order versions of the WL test. In particular, for each k-th order *Folklore WL test* in the hierarchy (see section 2.6.2), it is known that there exists a family of graphs that will make the test fail. These are known in the literature as k-isoregular graphs [Douglas, 2011], and the most well-known example is the *Strongly Regular* (SR) graph family, for $k = 2$:

Definition 4.5 (Strongly regular graph)**.** *A* SR(n,d,λ,μ)-graph *is a regular graph with n vertices and degree d, where every two adjacent vertices have always λ mutual neighbours, while every two non-adjacent vertices have always μ mutual neighbours.*

Proving that k-FWL test cannot be more expressive than GSN amounts to constructing a dictionary that allows GSN to distinguish certain pairs from the k-isoregular family. Indeed, this is the case for the 2-FWL test. Formally:

Proposition 4.6. *Let $\mathfrak{G}$ be the set of all graphs of size $\leq n$. There exist graphs α with size $|\mathcal{V}_\alpha| = O(1)$, i.e. independent of the maximum graph size n, such that a GSN with a dictionary $\mathcal{D} \supseteq \{\alpha\}$ is not less expressive than 2-FWL.*

We provide numerous counterexamples that prove this claim. figure 4.2 (right) provides a typical pair of SR graphs that can be distinguished with a 4-clique, while in section 4.8.1 this is extended to a large-scale study, where other constant size substructures (paths, cycles and cliques) can achieve similar results.

Remark. At the time of writing our paper, it was not clear if there exists a certain dictionary that results in GSNs that align with the WL hierarchy. Following our work, [Barceló et al., 2021] gave more in-depth results regarding the relations between the WL hierarchy and GSNs and settled this open question negatively. In particular, they showed that for every dictionary there exists a k such that k-WL is more expressive than the corresponding GSN while for certain dictionaries, GSN can distinguish graphs that $(k-1)$-WL deems isomorphic. However, there are also graphs that $(k-1)$-WL distinguishes, while GSN fails to do so. Nevertheless, we stress

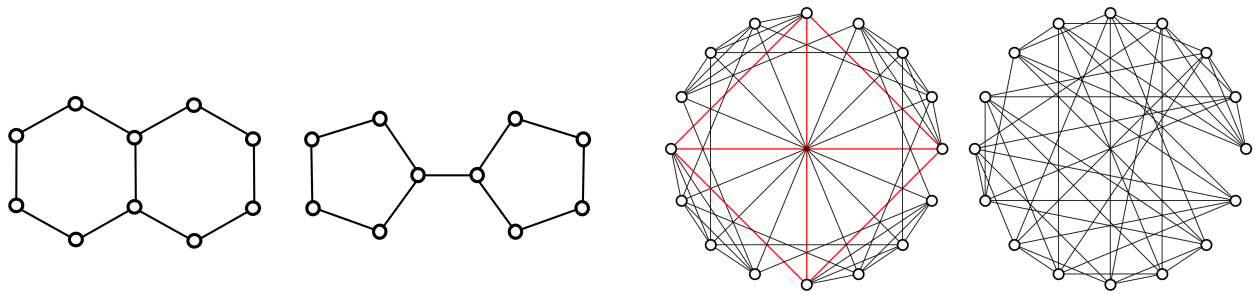


Figure 4.2: (Left) *Decalin* and *Bicyclopentyl*: Non-isomorphic molecular graphs than can be distinguished by GSN, but not the by the WL test [Sato, 2020] (vertices represent carbon atoms and edges represent chemical bonds). (Right) *Rook's 4x4 graph* and the *Shrikhande graph*: the smallest pair of strongly regular non-isomorphic graphs with the same parameters SR(16,6,2,2). GSN can distinguish them with 4-clique counts, while 2-FWL fails.

that such a connection is not necessary in order to design GNNs, since it does not take into account generalisation. In particular, despite the increase in expressivity, k-WL tests are not only more computationally involved, but they also process the graph in a non-local fashion. However, locality is presumed to be a strong inductive bias of GNNs, i.e. many real-world tasks on graph spaces are assumed to be compositions of local functions, an assumption that has been widely validated empirically by the excellent performance of MPNNs.

4.5 The computational complexity of GSNs

The complexity of GSN comprises two parts: (1) Preprocessing (substructure enumeration and assignment of features to vertices/edges) and (2) GNN evaluation (training/inference). The key appealing property is that GNN evaluation is linear w.r.t. the number of edges m and vertices n of an input graph, $O(m+n)$, similarly to a conventional MPNN (assuming constant depth, width and feature vector dimensions, including d_v^w, d_e^w). In particular, when using structural features (or any weighting function in general), we incur an increase in the multiplicative factors of m and n (which is linear in the dimensions of the structural features), compared to those of a vanilla MPNN (see Eq. (2.12)). The dimensions of the structural features $\mathbf{w}_{\mathcal{V}}, \mathbf{w}_{\mathcal{E}}$ are of size $O(|\mathcal{D}|k)$ and $O(|\mathcal{D}|k^2)$ respectively, and therefore when $|\mathcal{D}| = O(1)$ and $k = O(1)$, the GNN evaluation complexity remains linear in m and n. This is in contrast to the GNN evaluation computational complexity of higher-order methods, such as [Maron et al., 2019a, Morris et al., 2019] with $O(n^k)$ complexity, and [Vignac et al., 2020] with $O(n^2)$. This is also considerably smaller than the evaluation complexity of relational pooling [Murphy et al., 2019] which is $O(n!)$

Table 4.1: Summary of complexity bounds. GSN vs other GI-invariant/equivariant GNNs. $(n, m) = (|\mathcal{V}_G|, |\mathcal{E}_G|)$: # of vertices, # of edges (maximum across the dataset for the training complexity), $k = |\mathcal{V}_\alpha|$: # of atom vertices, $c(G, \alpha)$: # of occurrences of α in G, $a(G)$: arboricity of G, $\mathcal{D}$: dictionary, $\mathfrak{D}$: training set, I: # of epochs.

Model	GSN			MPNN	k-IGN & k-GNN	Symmetrisation
Preprocessing ($\texttt{enum}(G, \alpha)$)	Worst-case (brute-force)	n^k		—	—	—
	Special cases	α: connected, induced α: clique α: cycle G: planar, $k = O(1)$	$c(G, \alpha)nm$ $a(G)^{k-2}m$ $m + kc(G, \alpha)$ $n + c(G, \alpha)$			
Total training (dataset $\mathfrak{D}$)	$O\Big(I\lvert\mathfrak{D}\rvert(m+n) + \sum_{\alpha\in\mathcal{D}, G\in\mathfrak{D}} \texttt{enum}(G, \alpha)\Big)$			$O\big(I\lvert\mathfrak{D}\rvert(m+n)\big)$	$O\big(I\lvert\mathfrak{D}\rvert n^k\big)$	$O\big(I\lvert\mathfrak{D}\rvert n!\big)$
Total inference (per graph G)	$O\big(\sum_{\alpha\in\mathcal{D}} \texttt{enum}(G, \alpha) + m + n\big)$			$O\big(m+n\big)$	$O\big(n^k\big)$	$O\big(n!\big)$

in absence of approximations.

The worst-case complexity of subgraph enumeration for an arbitrary atom α of size k is $O\big(k^2 \frac{n!}{(n-k)!}\big) = O(n^k)$, for $k = O(1)$, which follows from the brute-force enumeration of all vertex tuples (without repetitions) of size k in the graph and then scanning each of them to verify adjacency preservation (by e.g. testing the induced adjacency matrices for equality with those of the atoms). However, this bound can be substantially improved for specific instances of the problem, taking into account the following: (1) The properties of the target graph, e.g., its sparsity. (2) The structure of the subgraph of interest, e.g., cycles and cliques. These substructure families are of particular importance in real-world graphs, social networks and molecules respectively, as will be discussed in the experimental section 4.8. (3) The number of occurrences of the patterns in the target graph (*output sensitive* algorithms), e.g., the algorithm in [Avis and Fukuda, 1996] enumerates connected induced subgraphs in $O\big(c(G, \alpha)nm\big)$ time, where $c(G, \alpha)$ the number of occurrences of the subgraph of interest α in G. In Appendix B.2.1 we provide an overview of these specialised algorithms and their complexities.

Table 4.1, provides a summary of the theoretical time complexity bounds of the preprocessing, including special cases, as well as the overall GSN training and inference. Additionally, in the same table we compare against other GI-invariant/equivariant GNN families: vanilla MPNNs, such as [Xu et al., 2019], higher-oreder GNNs, such as k-IGNs [Maron et al., 2019b] and k-GNNs [Morris et al., 2019] and symmetrisation without sampling, such as [Dasoulas et al., 2020, Murphy et al., 2019]. Regarding the total training time, in the vast majority of cases, the GNN evaluation term dominates since the number of epochs are typically large (e.g. compared to the size of the dictionary), and therefore the total training overhead of GSN (w.r.t. a vanilla MPNN) is usually

considerably smaller than its competitors. Additionally, typically in ML, the NN training is repeated multiple times during experimentation in order to optimise the hyperparameters of the model. Therefore, it is beneficial to maintain a low neural network complexity. Regarding the total inference time, the complexity of GSN varies a lot depending on the structure of the pattern and the target graph, as well as on the number of pattern occurrences in the target graph. For example, cycle enumeration in molecular graphs (which are planar and typically contain only a handful of cycles), or clique enumeration in very sparse graphs, will incur only an additional linear term in the overall inference complexity, rendering the total GSN inference time similar to vanilla MPNNs, and significantly smaller than its other competitors. However, clique enumeration in dense graphs will result in $O(n^k)$ complexity, increasing the inference time of GSN significantly compared to MPNNs and making it similar to higher-order GNNs.

Moreover, for both cases the bounds for higher-order GNNs and symmetrised GNNs are tight, whereas those of GSN might be overly pessimistic, since, apart from the specialised algorithms, arbitrary subgraph counting/enumeration is widely studied, and many general-purpose algorithms [Ullmann, 1976, Houbraken et al., 2014, Cordella et al., 2004, Carletti et al., 2017, Han et al., 2013, McCreesh and Prosser, 2015, Hocevar and Demsar, 2014] provide practical implementations that achieve significant speed-ups using heuristics. As a side note, approximate counting algorithms are also widely used, especially for counting frequent network motifs [Kashtan et al., 2004, Wernicke, 2005, Wernicke, 2006, Wernicke and Rasche, 2006], and can provide a considerable speed-up. Furthermore, recent neural approaches [Ying et al., 2020b, Ying et al., 2020a] provide fast approximate counting.

In our experiments, we used a general-purpose subgraph isomorphism algorithm. We benchmarked the VF2 algorithm [Cordella et al., 2004], and its recently improved version, the VF3 [Carletti et al., 2017] in our real-world networks and, as expected, in most of the cases, the preprocessing runtime was considerably smaller than that of the naive enumeration (see Fig. B.1, Appendix B.2.2, where we compare the empirical scaling w.r.t. n and k, to the $O(n^k)$ naive enumeration bound). In the Appendix B.2.2, we report the average and total preprocessing runtimes for both algorithms and various datasets and pattern sizes (Tables B.1 and B.2), while in the Appendix B.2.3, we contrast the preprocessing runtimes with those required in total for training/inference (Tables B.3 and B.4), experimentally validating the theoretical bounds of Table 4.1. As discussed above, specialised algorithms are expected to yield further improvements, however, this comparison is beyond the scope of our work.

4.6 Substructure Selection

Expressivity. The Reconstruction Conjecture provides a sufficient, albeit impractical condition for universality. This motivates us to analyse the constant size case $k = O(1)$ for practical scenarios, similar to the argument put forward for hard instances of graph isomorphism (Proposition 4.6). In particular, one can count only the most discriminative subgraphs, i.e. the ones that can achieve the maximum possible vertex disambiguation, similar to identifier-based approaches. Whenever these subgraph counts can provide a unique identification of the vertices, then universality will also hold (Corollary 3.1. in [Loukas, 2020]).

We conjecture, that in real-world scenarios the number of subgraphs needed for unique, or near-unique identification, are far fewer than those dictated by Corollary 4.3. This is consistent with our experimental findings, where we observed that certain small substructures such as paths and trees, significantly improve vertex disambiguation, compared to the initial vertex features (see figure 4.4 (left) and Table B.6 in the appendix). As expected this allows for better fitting of the training data, which validates our claim that GNN expressivity improves. In addition, [Barceló et al., 2021] provided an analysis showing in which cases adding an atom to an existing dictionary improves expressivity and when it remains unaffected.

Generalisation. However, none of the above claims can guarantee good generalisation to unseen data. For example, in figure 4.4, we observe that the test set performance does not follow the same trend as training performance when choosing substructures with strong vertex disambiguation. Aiming at better generalisation, it is desirable to make use of substructures for which there is prior knowledge of their importance in certain network distributions and have been observed to be intimately related to various properties. For example, graphlets have been extensively analysed in protein-protein interaction networks [Pržulj et al., 2004], triangles and cliques characterise the structure of ego-nets and social networks, in general, [Granovetter, 1982], simple cycles (rings) are central in molecular distributions, directed and temporal motifs have been shown to explain the working mechanisms of gene regulatory networks, biological neural networks, transportation networks and food webs [Milo et al., 2002, Paranjape et al., 2017, Benson et al., 2016]. In figure 4.4 (right), we showcase the importance of these inductive biases: a cycle-based GSN predicting molecular properties achieves a smaller generalisation gap compared to a traditional MPNN, while at the same time generalising better with fewer

training data.

Going forward from domain knowledge and various heuristics, such as motif frequencies or feature selection strategies, it would be desirable to design the dictionary in a data-driven manner. This is still an open problem that does not admit a straightforward solution due to its combinatorial nature. In chapter 5, we will describe a substructure selection algorithm, that learns the dictionary $\mathcal{D}$ using samples from the graph distribution at hand (training data) and optimising for a compression criterion. We will theoretically show, that optimal dictionaries have *low entropy*, i.e. they are small and contain frequent subgraphs. One could use this method as a heuristic (assuming that frequent subgraphs, will be relevant in the determination of real-world tasks). An even more reasonable and improved choice would be to learn the dictionary, using an algorithm similar to the one presented in chapter 5, optimising for the objective of interest on the training data (i.e. using the ERM learning algorithm). However, there are several challenges in this approach, such as non-differentiability and increased computational complexity during training, since subgraph enumeration cannot be precomputed in this case, and therefore we left this endeavour for future work.

4.7 Comparisons and Related Wrok

4.7.1 GNN expressivity

The following section provides a brief overview of the related work on GNN expressivity (in some cases enlisting already mentioned references for the sake of completeness of the timeline). Given the popularity of the topic in the GNN community, the literature is quite wide so inevitably some references might be missing, and most likely new works will be published in parallel with the publication of this thesis. However, we hope to provide the reader with the important takeaways and contextualise our work within this landscape. For a more comprehensive review, we refer the interested reader to the surveys of [Sato, 2020, Jegelka, 2022, Morris et al., 2021] and the excellent tutorial of [Frasca et al., 2022b]. In the experimental section, GSN is compared against a variety of these methods in real-world scenarios.

GI-sensitive GNNs & symmetrisation. With regard to the class of GI/permutation-sensitive GNNs, we have already discussed the cases of [Sato et al., 2019] and [Loukas, 2020], which

showed the connections between GNNs and distributed local algorithms [Angluin, 1980, Linial, 1992, Naor and Stockmeyer, 1993] and suggested *deterministic* local or global identifiers. The closest practical instantiation of this is the work of [Vignac et al., 2020], where the authors propose to propagate matrices of order equal to the size of the graph instead of vectors, at the expense of quadratic complexity. In the spirit of randomised algorithms, [Dasoulas et al., 2020] propose to use *stochastic* colourings/features in order to uniquely identify the vertices, where the space of possible options is reduced by first inspecting the vertex features, which is similar to [Murphy et al., 2019, Sato et al., 2021, Abboud et al., 2021] where a single sample from the random colour/feature distribution is used for each graph. It is worth observing that these methods are on expectation GI-invariant, which is a form of *symmetrisation*, as initially pinpointed by [Murphy et al., 2019] (see also [Puny et al., 2022]), and sampling amounts to a Monte Carlo estimate of the expected value (and its gradient during training). However, these estimators are typically high-variance, which is something that has not been convincingly addressed so far. In general, these approaches lack a principled permutation equivariant way to choose orderings/identifiers. To date, this is an open problem in graph theory called *graph canonicalisation* and it is at least as hard as solving graph isomorphism itself (see also section 5.3).

WL hierarchy. The seminal results in the theoretical analysis of the expressivity of GNNs [Xu et al., 2019] and k-GNNs [Morris et al., 2019] established that traditional message passing-based GNNs are at most as powerful as the 1-WL test, while [Chen et al., 2019, Dasoulas et al., 2020] showed that graph isomorphism is equivalent to universal GI-invariant function approximation. [Morris et al., 2019] first proposed a neural analogue to k-WL, which was later improved with more efficient alternatives in [Morris et al., 2020b, Morris et al., 2022]. [Kondor et al., 2018] and [Maron et al., 2019b] proposed higher-order (k-th order) Invariant Graph Networks (k-IGNs), which operate on k-th order tensors, while the latter fully characterised the space of linear (permutation) invariant/equivariant layers. Thereafter, a series of works characterised their expressive power w.r.t. universal approximation [Maron et al., 2019c, Ravanbakhsh, 2020, Keriven and Peyré, 2019] and equivalence with k-WL [Maron et al., 2019a, Geerts, 2020]. Additionally, a more efficient alternative, based on equivariant matrix polynomials, was proposed [Maron et al., 2019a], and was shown to be equivalent to k-FWL [Azizian and Lelarge, 2021] (see section 2.6.2 for the exact definition). The main drawbacks of these methods are the training and inference time complexity and memory requirements of $O(|\mathcal{V}|^k)$, where $k \geq 2$, the

super-exponential number of parameters (for linear IGNs) making them impractical, as well as their non-local nature possibly making them more prone to overfitting.

Symmetry-breaking & positional encodings. Concurrently and following our paper, other classes of weighting functions/positional encodings/symmetry-breaking mechanisms were proposed and frequently incorporated into Transformer architectures. A (non-exhaustive) list includes Laplacian eigenvectors [Dwivedi et al., 2020, Beaini et al., 2021, Dwivedi and Bresson, 2021, Kreuzer et al., 2021], distance-based encodings [Li et al., 2020, Ying et al., 2021], diffusion kernels [Mialon et al., 2021], heat kernels [Feldman et al., 2022], random walk-based encodings [Dwivedi et al., 2022, He et al., 2022] and Laplacian eigenmaps [Wang et al., 2022]. Most of these encodings probably improve expressivity, they have been associated with important graph properties and can be efficiently computed, but several of them are GI-sensitive with a recent work attempting to address this problem [Lim et al., 2023]. Finally, in another related paper [de Haan et al., 2020], in the spirit of symmetry breaking the authors propose to linearly transform each neighbouring message with a different weight matrix based on the local isomorphism class of the corresponding edge (similar to our definition of structural roles). However, as also noted by the authors, taking into account all possible local isomorphism classes leads to insufficient weight sharing and hence to overfitting.

Subgraph GNNs. Finally, following our work, several methods used subgraphs as an instrumental element of their GNN architectures. [Barceló et al., 2021] proposed *homomorphism* counts (mappings that allow vertex repetitions) instead of subgraph counts, showed that GSN can be expressed as such, and provided a series of theoretical results answering several of the theoretical questions we left open in this work. [Bodnar et al., 2021, Bodnar et al., 2021] showed that lifting graphs to a *simplicial* or a *cellular* complex and subsequently performing an appropriate WL algorithm (or a neural analogue) improves expressivity. Recently, there has been a surge in methods that, sometimes in parallel, proposed variations of the so-called *Subgraph GNNs* [You et al., 2021, Sandfelder et al., 2021, Thiede et al., 2021, Cotta et al., 2021, Zhang and Li, 2021, Papp et al., 2021, Bevilacqua et al., 2022, Zhao et al., 2022], where, in a nutshell, a graph is decomposed into a multiset of subgraphs using a certain predefined policy (e.g. vertex removal, edge removal, vertex marking, ego-net extraction, ego-net extraction w/ vertex marking). Several of these policies were compared against each other and against GSN

in [Papp and Wattenhofer, 2022]. Node-based policies were unified and extended in a general framework in [Frasca et al., 2022a], where it was also proved that they are at most as expressive as 3-IGNs and 3-WL. Finally, [Qian et al., 2022] also proposed a generalising framework (in addition to a data-driven method for subgraph selection) showing that the expressivity of Subgraph GNNs is upper bounded by (k+1)-FWL, but they are incomparable to k-FWL.

Quantifying expressivity. As we saw in section 4.2, solely quantifying the expressive power of GNNs in terms of their ability to distinguish non-isomorphic graphs does not provide the necessary granularity. As a result, there have been several efforts to analyse the power of k-WL tests in comparison to other graph properties (graph invariants) [Fürer, 2010, Fürer, 2017, Arvind et al., 2019, Dell et al., 2018]. [Garg et al., 2020]. [Loukas, 2020] showed lower bounds for the depth-width tradeoffs required to compute various graph properties and solve or approximate various combinatorial optimisation problems with global-identifier-based GNNs, similarly to [Sato et al., 2019], where impossibility results were derived for local-identifier-based GNNs w.r.t the capacity to approximate certain NP-hard optimisation problems. Vanilla GNNs were studied w.r.t. their ability to compute polynomials of the adjacency matrix [Dehmamy et al., 2019] and to count substructures [Chen et al., 2020], while [Garg et al., 2020] showed that some impossibility results (e.g. inability to compute shortest or longest cycle, diameter etc.) extend to other more expressive architectures. Very recently [Zhang et al., 2023] showed that many GNN architectures (including GSN for problems with graphs of unbounded size) cannot solve *vertex-biconnectivity* problems, even though their computational complexity is linear. Finally, [Fereydounian et al., 2022] provided a fine-grained analysis on the exact class of functions that GNNs can express.

4.7.2 Substructures in complex networks prior to the GNN era.

The idea of analysing complex networks based on small-scale topological characteristics dates back to the 1970s and the notion of triad census for directed graphs [Holland and Leinhardt, 1976]. The seminal paper of [Milo et al., 2002] coined the term *network motifs* as over-represented subgraph patterns that were shown to characterise certain functional properties of complex networks in systems biology. The closely related concept of *graphlets* [Pržulj et al., 2004, Pržulj, 2007, Milenković and Pržulj, 2008, Sarajlić et al., 2016], different from motifs in being induced

subgraphs, has been used to analyse the distribution of real-world networks and as a topological signature for network similarity. Our work is similar in spirit to the *graphlet degree vector* (GDV) [Pržulj, 2007], a vertex-wise descriptor based on graphlet counting.

Substructures have been also used in the context of ML. In particular, subgraph patterns have been used to define Graph Kernels (GKs) [Horváth et al., 2004, Shervashidze et al., 2009, Costa and De Grave, 2010, Kriege and Mutzel, 2012, Nguyen and Maehara, 2020], with the most prominent being the graphlet kernel [Shervashidze et al., 2009]. Motif-based vertex embeddings [Dareddy et al., 2019, Rossi et al., 2018] and diffusion operators [Monti et al., 2018, Sankar et al., 2019, Lee et al., 2019] that employ adjacency matrices weighted according to motif occurrences, have recently been proposed for graph representation learning. Our formulation provides a unifying framework for these methods and was the first to analyse their expressive power.

4.8 Results

In the following section, we evaluate GSN in comparison to the state-of-the-art in a variety of datasets from different application domains. We are interested in practical scenarios where the dictionary $\mathcal{D}$, as well as the size of each $\alpha \in \mathcal{D}$, are kept small. Depending on the dataset domain we experimented with typical graph families (*cycles, paths, cliques and trees*) and maximum substructure size k (for each setting, the dictionary consists of all the substructures of the family with size $\leq k$).

We experimented with both induced subgraphs and not-necessarily-induced subgraphs (from now on referred to as graphlets and motifs respectively) and observed similar performance in most cases. To showcase that structural features can be used as an off-the-shelf strategy to boost GNN performance, we usually choose a base message passing architecture and minimally modify it into a GSN. Unless otherwise stated, the base architecture is a general-purpose MPNN as in Eq. (2.11) with MLPs used in the message and update functions. Additional implementation details can be found in Appendix B.3.

4.8.1 Graph Isomoprhim Testing

We tested the ability of GSNs to decide if two graphs are non-isomorphic on a collection of Strongly Regular graphs of size up to 35 vertices, attempting to disambiguate pairs with the same

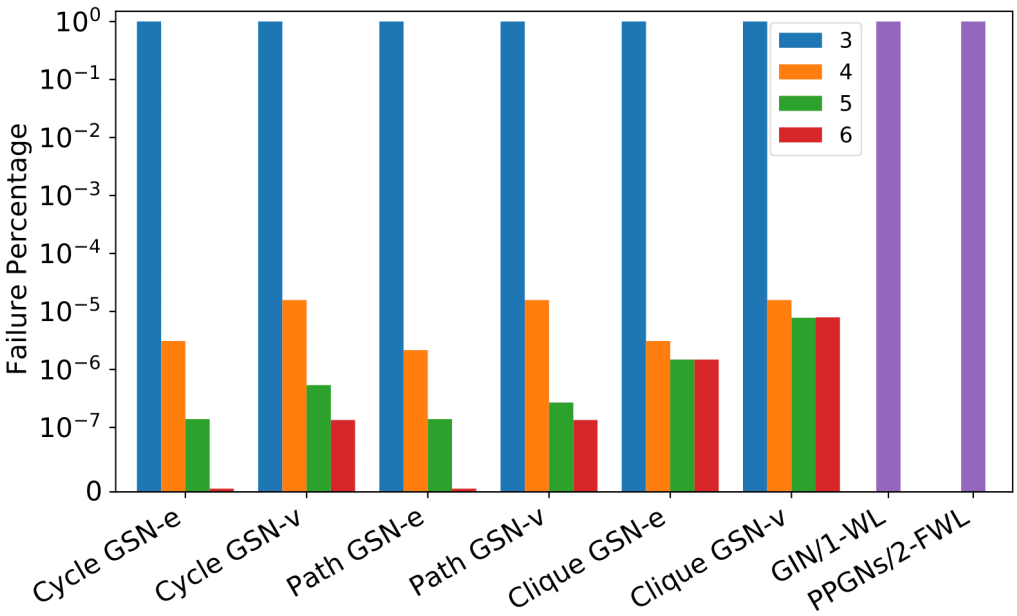


Figure 4.3: GI test for SR graphs (log scale, smaller values are better). Different colours indicate different substructure sizes.

number of vertices (for different sizes the problem becomes trivial). As we are only interested in the bias of the architecture itself, we use a *GSN with random weights* to compute graph-wise representations in $\mathbb{R}^{d_{\text{out}}}$. Two graphs are deemed isomorphic if the Euclidean distance of their representations is smaller than a predefined threshold ϵ. Figure 4.3 shows the failure percentage of our isomorphism test when using different graphlet substructures (*cycles, paths*, and *cliques*) of varying maximum size k. Interestingly, the number of failure cases of GSN decreases rapidly as we increase k; cycles and paths of maximum length $k = 6$ are enough to tell apart all the graphs in the dataset. Note that the performance of cliques saturates, possibly because the largest clique in our dataset has 5 vertices. Observe also the discrepancy between GSN-v and GSN-e. In particular, vertex-wise counts do not manage to distinguish all graphs, (although failing on only a few instances), which is in accordance with Theorem 4.4. Finally, 1-WL [Xu et al., 2019] and 2-FWL [Maron et al., 2019a] equivalent models demonstrate 100% failure, as expected from theory.

4.8.2 TUD benchmarks: graph classification

We evaluate GSN on datasets from the TUD benchmarks, a well-established family of datasets in the graph ML community. We use seven datasets of molecular, biological and social networks and compare GSN against various GNNs and Graph Kernels. The base architecture that we used

Table 4.2: Graph classification accuracy on TUD Datasets. First, Second, **Third** best methods are highlighted. For GSN, the best-performing dictionary is shown. *Graph Kernel methods.

Dataset	MUTAG	PTC	Proteins	NCI1	Collab	IMDB-B	IMDB-M
RWK* [Gärtner et al., 2003]	79.2±2.1	55.9±0.3	59.6±0.1	>3 days	N/A	N/A	N/A
GK* (k=3) [Shervashidze et al., 2009]	81.4±1.7	55.7±0.5	71.4±0.31	62.5±0.3	N/A	N/A	N/A
PK* [Neumann et al., 2016]	76.0±2.7	59.5±2.4	73.7±0.7	82.5±0.5	N/A	N/A	N/A
WL kernel* [Shervashidze et al., 2011]	90.4±5.7	59.9±4.3	75.0±3.1	**86.0±1.8**	78.9±1.9	73.8±3.9	50.9±3.8
GNTK* [Du et al., 2019a]	90.0±8.5	**67.9±6.9**	75.6±4.2	**84.2±1.5**	**83.6±1.0**	**76.9±3.6**	**52.8±4.6**
DCNN [Atwood and Towsley, 2016]	N/A	N/A	61.3±1.6	56.6±1.0	52.1±0.7	49.1±1.4	33.5±1.4
DGCNN [Zhang et al., 2018]	85.8±1.8	58.6±2.5	75.5±0.9	74.4±0.5	73.8±0.5	70.0±0.9	47.8±0.9
IGN [Maron et al., 2019b]	83.9±13.0	58.5±6.9	**76.6±5.5**	74.3±2.7	78.3±2.5	72.0±5.5	48.7±3.4
GIN [Xu et al., 2019]	89.4±5.6	64.6±7.0	76.2±2.8	82.7±1.7	80.2±1.9	75.1±5.1	52.3±2.8
PPGNs [Maron et al., 2019a]	**90.6±8.7**	66.2±6.6	**77.2±4.7**	83.2±1.1	81.4±1.4	73.0±5.8	50.5±3.6
Natural GN [de Haan et al., 2020]	89.4±1.60	66.8±1.79	71.7±1.04	82.7±1.35	N/A	74.8±2.01	51.3±1.50
WEGL [Kolouri et al., 2021]	N/A	**67.5±7.7**	76.5±4.2	N/A	80.6±2.0	75.4±5.0	52.3±2.9
GIN+GraphNorm [Cai et al., 2021]	**91.6 ± 6.5**	64.9 ± 7.5	**77.4 ± 4.9**	82.7 ± 1.7	80.2 ± 1.0	76.0 ± 3.7	N/A
GSN-e	**90.6±7.5** 6 (cycles)	**68.2±7.2** 6 (cycles)	**76.6±5.0** 4 (cliques)	**83.5± 2.3** 15 (cycles)	**85.5±1.2** 3 (triangles)	**77.8±3.3** 5 (cliques)	**54.3±3.3** 5 (cliques)
GSN-v	**92.2±7.5** 12 (cycles)	67.4±5.7 10 (cycles)	74.59±5.0 4 (cliques)	**83.5±2.0** 3 (triangles)	**82.7±1.5** 3 (triangles)	**76.8±2.0** 4 (cliques)	**52.6±3.6** 3 (triangles)

is GIN [Xu et al., 2019]. We follow the same evaluation protocol of [Xu et al., 2019], performing 10-fold cross-validation and then reporting the performance at the epoch with the best average accuracy across the 10 folds. Table 4.2 lists all the methods evaluated with the split of [Zhang et al., 2018]. We select our model by tuning architecture and optimisation hyperparameters and dictionary-related parameters, that is (i) maximum dictionary graph size k, (ii) motifs against graphlets. Following domain evidence, for networks with community structure (moderate and large clustering coefficient), such as social networks, we used *cliques*, while for molecular graphs we used *cycles*, due to the prominent role of ring structures that are known to strongly influence molecular properties. We report the best-performing architecture for both GSN-e and GSN-v variants. As can be seen, our model outperforms the majority of its competitors in most of the datasets, typically with a considerable margin from the base GNN architecture.

4.8.3 ZINC benchmark: regression on molecular graphs

We evaluate GSN on the task of regressing the "penalized water-octanol partition coefficient - logP" (see [Gómez-Bombarelli et al., 2018, Kusner et al., 2017, Jin et al., 2018a] for details) of molecules from the ZINC database [Irwin et al., 2012a, Dwivedi et al., 2020]. We use structural features obtained with cycle counting and report the result of the best-performing dictionary on the validation set. The data split is obtained from [Dwivedi et al., 2020] and the evaluation metric is the Mean Absolute Error (MAE). We compare against a variety of baselines, ranging from traditional message passing NNs to recent more expressive architectures [Corso et al.,

Table 4.3: *ZINC* dataset (graph property regression). Performance metric on the test set (mean absolute error).

Method	MAE	MAE (EF)
GCN [Kipf and Welling, 2017]	0.469±0.002	-
GIN [Xu et al., 2019]	0.408±0.008	-
GraphSage[Hamilton et al., 2017]	0.410±0.005	-
GAT [Velickovic et al., 2018]	0.463±0.002	-
MoNet[Monti et al., 2017]	0.407±0.007	-
GatedGCN [Bresson and Laurent, 2017]	0.422±0.006	0.363±0.009
MPNN	0.254±0.014	0.209±0.018
MPNN-r	0.322±0.026	0.279±0.023
PNA[Corso et al., 2020]	0.320±0.032	0.188±0.004
DGN[Beaini et al., 2021]	0.219±0.010	0.168±0.003
GNNML[Balcilar et al., 2021]	0.1612 ± 0.006	-
HIMP[Fey et al., 2020]	-	0.151±0.006
SMP[Vignac et al., 2020]	0.219±	0.138±
GSN	**0.140±0.006**	**0.115±0.012**

2020, Beaini et al., 2021, Vignac et al., 2020, Balcilar et al., 2021] and a molecular-specific one (HIMP) which is based on the junction-tree molecular decomposition [Fey et al., 2020]. As dictated by the evaluation protocol of [Dwivedi et al., 2020], the total number of parameters of the model is approximately 100K, which is achieved by selecting an appropriate network width. Wherever possible, we compare two variants, one that ignores edge features and one that takes them into account. In both cases, GSN outperforms all the baselines. For completeness, we also note that a GSN with 500K params attains **0.101 ± 0.010** MAE.

4.8.4 OGB benchmark: classification on large-scale graphs

Table 4.4: OGB property prediction: molecular graphs. Test and validation performance metrics. First, Second, **Third** best methods are highlighted

	ogbg-molhiv		**ogbg-molpcba**	
Method	Test ROC-AUC	Val. ROC-AUC	Test AP	Val. AP
GCN[Kipf and Welling, 2017]	0.7606 ± 0.0097	0.8204 ± 0.0141	0.2020 ± 0.0024	0.2059 ± 0.0033
HIMP [Fey et al., 2020]	0.7880 ± 0.0082	N/A	0.2739 ± 0.0017	N/A
PNA [Corso et al., 2020]	**0.7905 ± 0.0132**	**0.8519 ± 0.0099**	**0.2838 ± 0.0035**	**0.2926 ± 0.0026**
GIN [Xu et al., 2019]	0.7558 ± 0.0140	0.8232 ± 0.0090	0.2266 ± 0.0028	0.2305 ± 0.0027
GIN+VN [Xu et al., 2019]	0.7707 ± 0.0149	0.8479 ± 0.0068	0.2703 ± 0.0023	0.2798 ± 0.0025
DGN (eigenvectors) [Beaini et al., 2021]	**0.7970 ± 0.0097**	0.8470 ± 0.0047	**0.2885 ± 0.0030**	**0.2970 ± 0.0021**
GSN (GIN base)	0.7606 ± 0.0174	**0.8517 ± 0.0090**	0.2508 ± 0.0023	0.2542 ± 0.0021
GSN (GIN+VN base)	0.7799 ± 0.0100	**0.8658 ± 0.0084**	**0.2816 ± 0.0047**	**0.2913 ± 0.0017**
GSN (DGN base)	**0.8039 ± 0.0090**	0.8473 ± 0.0096	N/A	N/A

We use three graph property prediction datasets from the Open Graph Benchmark (OGB) [Hu et al., 2020]: `ogbg-molhiv` and `ogbg-molpcba` are molecular datasets where the task is

Table 4.5: OGB property prediction: protein-protein association networks (biological). Test and validation performance metrics. First, Second, **Third** best are highlighted.

Method	ogbg-ppa	
	Test Accuracy	Val. Accuracy
GCN[Kipf and Welling, 2017]	0.6839 ± 0.0084	0.6497 ± 0.0034
HIMP [Fey et al., 2020]	N/A	N/A
PNA [Corso et al., 2020]	N/A	N/A
GIN [Xu et al., 2019]	**0.6892 ± 0.0100**	**0.6562 ± 0.0107**
GIN+VN [Xu et al., 2019]	**0.7037 ± 0.0107**	**0.6678 ± 0.0105**
DGN (eigenvectors) [Beaini et al., 2021]	N/A	N/A
GSN (GIN base)	0.6877 ± 0.0022	0.6489 ± 0.0065
GSN (GIN+VN base)	**0.7119 ± 0.0120**	**0.6716 ± 0.0078**
GSN (DGN base)	N/A	N/A

graph-level binary classification. In `ogbg-molhiv` the task is to predict if a molecule inhibits HIV replication or not, while `ogbg-molpcba` has multiple tasks that we need to optimise for simultaneously. We also evaluate GSN on `ogbg-ppa`, a dataset of protein-protein association networks, where the task is to predict the taxonomic group of each graph (37-way classification) The underlying distribution of this dataset is significantly different from the other two since it contains larger and denser graphs with community structure.

In Tables 4.4 and 4.5, we compare against the following baselines: two vanilla GNNs (GCN [Kipf and Welling, 2017] and GIN [Xu et al., 2019]), two modern GNNs (PNA [Corso et al., 2020] and DGN [Beaini et al., 2021]) with provably increased expressivity and strong empirical performance, and HIMP, a molecule-specific GNN [Fey et al., 2020]. There is a plethora of other methods that have reported results in these datasets, sometimes outperforming GSN (see the OGB public leaderboard[3]). Note that many of them are either domain-specific or have intricate implementation details which are orthogonal to our work. Hence, an exhaustive comparison is beyond the scope of this experimental section. Instead, our goal is to highlight that GSN can be used as a plug-and-play method and boost the performance of a base architecture without any hyperparameter tuning (except for the dictionary selection, which in our case boils down to searching the maximum size k of the graphs in the family considered), rather than to achieve state-of-the-art results.

Following this rationale, we choose a base architecture and modify it into a GSN variant by introducing structural features in the aggregation function: cycle counts for the molecular datasets and triangle counts for `ogbg-ppa`. We use the following base architectures: (a) *GIN*

[3] https://ogb.stanford.edu/docs/leader_graphprop/

and (b) *GIN-VN*, a variation of GIN that takes edge features into account and is extended with a *virtual node*, i.e. an artificial vertex that is connected to all the other vertices in the graph. For `ogbg-molhiv` we also used (c) DGN [Beaini et al., 2021] as base architecture, a GNN that propagates messages in an anisotropic manner, and it is a particular instantiation of Eq. (2.11). The authors of DGN use weighting functions (they refer to them as graph vector fields) defined by the eigenvectors of the graph, which in this case are replaced by subgraph counts. More information can be found in the supplementary material.

Using the evaluators provided by the authors, we report the relevant metric at the epoch with the best validation performance (the maximum dictionary graph size is also chosen based on the validation set). By examining the results in Tables 4.4, 4.5 the following observations can be made, (a) GSN seamlessly improves the performance of the base architecture, both in the test set and the validation set, sometimes significantly, or, in only one case, is on par. (b) Cycles are a good inductive bias when learning on molecules, corroborating our results on the ZINC dataset, while the same holds for triangles in PPA networks. The latter agrees with our intuition that tasks defined on graphs with community structure correlate with the presence of triangles (or cliques), as was the case for social networks in the TU Datasets. (c) General purpose GNNs benefit from symmetry-breaking mechanisms, either in the form of eigenvectors (DGN) or in the form of substructures.

4.8.5 Ablation Studies

Comparison between substructure collections. In figure 4.4 (left), we compare the training and test error for different graph families (cycles, paths and trees – for each experiment we use all the graphs of size $\leq k$ in the family). Additionally, with regards to GSN-v, we measure the "uniqueness" of the features/identifiers each dictionary yields as follows: for each graph G in the dataset, we measure the percentage of unique vertex features $1 - \delta_G = \frac{1}{|\mathcal{V}|}\left|\left\{\left(\mathbf{u}_{\mathcal{V}}(i), \mathbf{w}_{\mathcal{V}}(i)\right) \mid i \in \mathcal{V}\right\}\right|$ (vertex attributes concatenated with vertex structural features) and then take its empirical mean over the training dataset, i.e. $1-\delta_{\mathfrak{D}} = 1-\frac{1}{|\mathfrak{D}|}\sum_{G\in\mathfrak{D}}\delta_G$, yielding the disambiguation score. The disambiguation scores for the different graph families are illustrated as horizontal bars in figure 4.4 (the exact values can be found in Appendix B.3, Table B.6).

The first thing to notice is that the training error is tightly related to the disambiguation score.

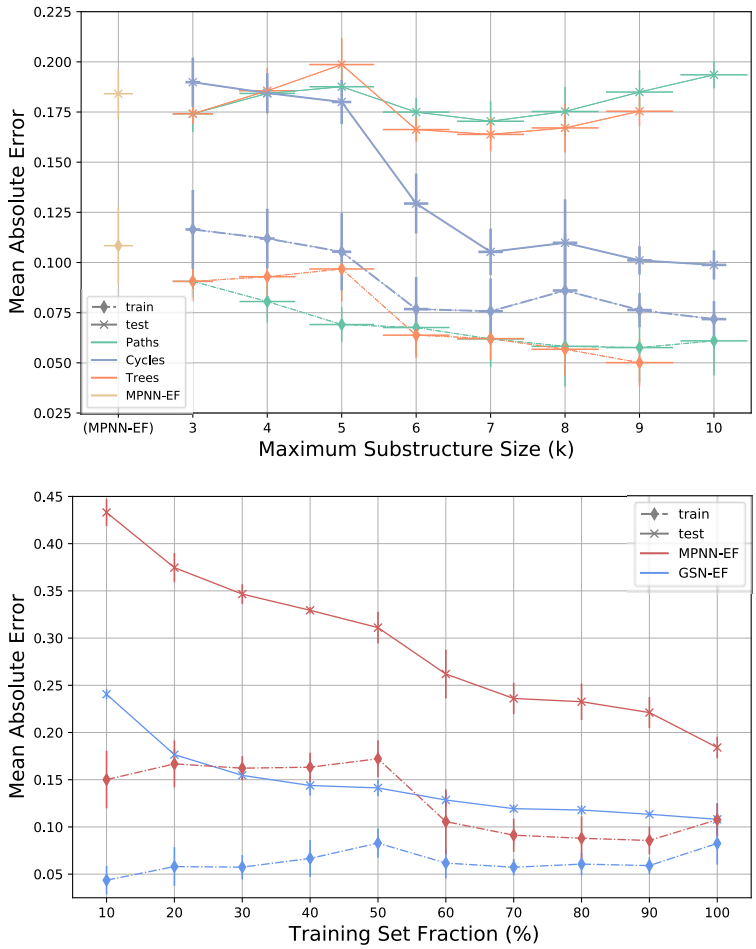


Figure 4.4: (Top) Train (dashed) and test (solid) MAEs for path-, tree- and cycle-GSN-*EF* as a function of the maximum dictionary graph size k. Vertical bars indicate standard deviation; horizontal bars depict disambiguation scores $\delta_{\mathfrak{D}}$. (Bottom) Train (dashed) and test (solid) MAEs for GSN-*EF*(blue) and MPNN-*EF*(red) as a function of the dataset fraction used for training.

As identifiers become more discriminative, the model gains expressive power. On the other hand, the test error is not guaranteed to decrease when the identifiers become more discriminative. For example, although cycles have smaller disambiguation scores, they manage to generalise much better than the other substructures, the performance of which is similar to the baseline architecture (MPNN with MLPs). This is also observed when comparing against [Sato et al., 2021] (*MPNN-r* method in Table 4.3), where, akin to unique identifiers, random features are used to improve the expressivity of GNN architectures. This approach also fails to improve the baseline architecture in terms of performance in the test set. This validates our intuition that unique identifiers can be hard to generalise when chosen in a non-GI equivariant way and motivates once more the importance of choosing the identifiers not only based on their discriminative power, but also in a way that allows incorporating the appropriate inductive

biases. Finally, we observe a substantial jump in performance when using GSN with cycles of size $k \geq 6$. This is not surprising, as cyclical patterns of such sizes (e.g. aromatic rings) are very common in organic molecules.

Sample efficiency. We repeat the experimental evaluation on ZINC using different fractions of the training set and compare the vanilla MPNN model against GSN. In figure 4.4 (right), we plot the training and test errors of both methods. Regarding the training error, GSN consistently performs better, following our theoretical analysis of its expressive power. More importantly, GSN manages to generalise much better even with a small fraction of the training dataset. Observe that GSN requires only 20% of the samples to achieve approximately the same test error that MPNN achieves when trained on the entire training set.

Table 4.6: Comparison between DeepSets and GSN using the same dictionary $\mathcal{D}$.

Dataset	DeepSets	# params	GSN	# params
MUTAG	**93.3±6.9**	3K	92.8±7.0	3K
PTC	66.4±6.7	2K	**68.2±7.2**	3K
Proteins	**77.8±4.2**	3K	**77.8±5.6**	3K
NCI1	80.3 ±2.4	10K	**83.5± 2.0**	10K
Collab	80.9 ±1.6	30K	**85.5±1.2**	52K
IMDB-B	77.1 ±3.7	51K	**77.8±3.3**	65K
IMDB-M	53.3 ±3.2	68K	**54.3±3.3**	66K
ZINC	0.288 ±0.003	366K	**0.108 ±0.018**	385K
ogbg-molhiv	77.34±1.46	3.4M	**77.99±1.00**	3.3M

Structural features: How important is message passing? Finally, we study the abilities of the structural features to solve the task at hand, when given as input to a graph-agnostic function approximator. In particular, to retain GI-invariance, we treat the structural features as a set and employ a hypothesis class that holds the UAP on set functions, i.e. *DeepSets* [Zaheer et al., 2017]. We use the same dictionary as the one that our best-performing GSN uses and, to ensure a fair evaluation, we perform the same hyperparameter search on both DeepSets and GSN (see Appendix B.3). Interestingly, as we show in Table 4.6, our baseline attains particularly strong performance across a variety of datasets and often outperforms other traditional message-passing baselines. This demonstrates the importance of substructures in a variety of graph-level real-world problems and motivates their use as a new inductive bias in GNN architectures. As expected, we observe that applying message passing, brings performance improvements in the vast majority of the cases, sometimes considerably, as in the ZINC dataset.

Part II

Graph Compression

5

Partition and Code: Learning how to compress graphs

5.1 Introduction

In the following and last chapter of our contributions, we will continue navigating the research line of function approximation in graph spaces. We will now focus on a topic at the intersection of machine learning and information theory for graph-structured data. We will study a *source coding* problem and in particular that of *lossless graph compression*. A reasonable question to ask is why we treat this as a function approximation problem and what makes compression special, among other particular cases.

The function design aspect. To answer the first part of the question, we will first establish the concept of *data compression*. In general, storage, communication, and creation of *information* (anything from symbols, audiovisual signals and text corpora to mathematical concepts, physical laws and even emotions and intentions qualify as a good example for the abstract notion of information) can be achieved by representing information with the help of an agreed *language*. Typically, a language consists of sequences of symbols that belong in a finite alphabet $\mathbb{B}$ and may need to respect a set of rules. For example, humans may communicate information using natural languages arising from various different known alphabets, such as the Latin and the Greek alphabet, while similarly, computers use arithmetic alphabets, most notably the binary one $\mathbb{B} = \{0, 1\}$. Importantly, representing information with a given language can be done in multiple different ways, i.e. with multiple different symbol sequences, also known as *codewords*, (e.g. this very text can be rewritten in infinitely many different ways in order to convey the same piece of information to the reader). In general, by data compression, we refer to procedures

that convert an original representation of a piece of information to a new one, whose length, also known as *description length* (the number of sequence symbols used) is smaller than that of the original one.

To formalise this mathematically, we can assume that each piece of information is taken from an input set $\mathscr{X}$ and our language is the output set $\mathscr{Y} = \mathbb{B}^* := \bigcup_{i=0}^{\infty} \mathbb{B}^i$. In data compression, we usually seek to determine a function $h_{\text{enc}} : \mathscr{X} \to \mathscr{Y}$, which is commonly known as *compressor*, *encoder*, or *source code* according to the terminology used in [Cover and Thomas, 2006]. Ground-truth information is rarely known a-priori (unless we seek to imitate the behaviour of a predefined compressor). Instead, the code is designed or optimised with respect to a specific objective, the general principle of which is to favour codes that lead to as small as possible description lengths and as much as possible information preserved. In *lossless compression*, which is the setup of interest in this thesis, no information can be lost and thus the code needs to be invertible, i.e. one needs to guarantee the existence of a function h_{dec} , such that $h_{\text{dec}}\big(h_{\text{enc}}(x)\big) = x, \ \forall\, x \in \mathscr{X}$. In this case, the code is called *non-singular*. The objective is to minimise the description lengths of all the points in $\mathscr{X}$. More generally, when the points that the compressor encounters are sampled from a distribution (or *probabilistic source* in information-theoretic terminology), the objective is to minimise the *expected* codeword length. In lossy compression, information loss (*distortion*) is allowed and thus the code needs not necessarily to be invertible, while the objective is formalised so as to find a balance between the minimisation of codeword lengths and distortion. Distortion is either defined as a distance from the original representation, as a perceptual distance or based on the ability to solve a downstream task [Dubois et al., 2021].

The function approximation/statistical aspect. In order to design a compressor it is necessary to make assumptions about the information we will encounter and in particular about its distribution. In fact, for lossless compression, the *source-coding theorem* [Shannon, 1948] dictates that the optimal achievable bit rate of a compressor is uniquely determined by the data distribution, and in particular by its *entropy* (see section 5.2). The assumptions can be made a-priori, i.e. by using prior knowledge about the data that will be encountered. However, this approach is too rigid and will inevitably lead to a suboptimal code if our assumptions are incorrect. Therefore, a more flexible alternative is to adapt our assumptions by using data observations. This is exactly where the function approximation aspect of the problem arises: one can approximate the distribution from observations and design an optimal compressor for

the estimated distribution (either subsequently on in an online fashion). In the information theory literature, this process is called *universal source coding* and notable examples are *adaptive arithmetic coding* and *Lempel-Ziv* compressors [Ziv and Lempel, 1977, Ziv and Lempel, 1978, Welch, 1984]. This illustrates the connection between compression and learning: optimally compressing data from an unknown distribution is equivalent to accurately estimating it.

The importance of graph compression. From a practical standpoint, it goes without saying that, given the ever-increasing amount of data that needs to be generated, stored, transmitted, processed, visualised, etc every day, data compression is essential in order to ensure a reduction in the storage and transmission costs and the time spent. Compression of euclidean data such as text, images, or video, has been deeply studied, and many algorithms are widely used in practice underpinning many modern technologies, such as web protocols and video streaming. However, graph-structured data, as well as other types of combinatorial data [Steinruecken, 2015], remain a notable exception. Graph data are becoming more prevalent, either in the form of single graphs of massive scale (e.g. web graphs, knowledge graphs or a graph of social network) or in the form of large graph datasets (e.g. molecular graphs, code represented as a graph) and therefore it becomes increasingly important to invent specialised and practical ways to encode them parsimoniously.

Nevertheless, the algorithms currently used to compress graphs are general-purpose (e.g. for sequential data) or tailored to *labelled* graphs, i.e. graphs with a particular ordering/labelling of their vertices, and fail to make the right assumptions about the underlying distributions, thus resulting in suboptimal codes (see section 5.7 for details). The most important assumption they miss is that in many cases we are not interested in the exact vertex ordering, and we can instead compress the graph as an *unlabelled* object (when the graphs are by construction unlabelled, e.g. molecules, or when the labelling is irrelevant, e.g. when we are only interested in querying on inferring graph-level information). Failing to make such an assumption can be shown to lead to a significant loss in the potential compression gains (see section 5.3 for an analysis).

Additionally, data compression is of theoretical interest since it allows us to mathematically reason about the randomness of our data (e.g. do they contain any patterns/regularities and what are they?) as well as their complexity (e.g. can we succinctly describe them?). In particular, data compression is intimately related to machine learning [MacKay, 2003] and *algorithmic information theory* [Solomonoff, 1960, Solomonoff, 1964, Solomonoff, 1964, Chaitin,

1969, Kolmogorov, 1998], which in the case of graphs, provides us insights into probabilistic notions, such as random graph models (generative models) for unlabelled graphs, and complexity notions, such as the *Kolmogorov complexity* of graphs. It is also worth mentioning that, as we will see in section 5.3, the constraints imposed by the principles of unlabelled graph compression on our hypothesis class, create fundamentally different design challenges compared to the usual graph function approximation problems, such as classification and regression.

Our contributions. This work serves a dual purpose. First, to explore the fundamental principles of unlabelled graph compression and second, to propose a practical algorithm that adheres to them. The main challenges that arise can be summarised as follows (see section 5.3 for details):

C1. Dealing with graph isomorphism (computational complexity). It can be shown that approaching an optimal bit rate requires mapping each graph to its equivalence class. Addressing this would imply a solution to graph isomorphism, and therefore optimality comes with the cost of high-computational complexity. This is the most fundamental challenge that discerns graph compression from other types of data.

C2. Likelihood estimation and model description length (compression vs generative models). An optimal encoder [Shannon, 1948] requires one to accurately estimate and evaluate the probabilities of all the possible outcomes of the underlying domain. This task reminds us of the objective of likelihood-based (deep) generative models (see section 5.7). For instance, autoregressive models, partition the data into parts to obtain a decomposition of the probability distribution: e.g. images are partitioned into pixels or patches [van den Oord et al., 2016, Mentzer et al., 2019], and text into characters or n-grams [Bell et al., 1989, Schmidhuber and Heil, 1996, Mahoney, 2000]. Generative models rely on two factors to accurately capture all the statistical dependencies in the data: a large amount of training data and a powerful, typically heavily parametrised model. However, in compression adaptivity using only a handful of observations is desired (especially in diverse domains, such as real-world networks), while large models result in a reduction of the overall compression gains. In particular, although frequently overlooked, the parameters of the model themselves need to be also stored and transmitted to the decoder along with the compressed data in order to recover the original representation. Thus, the objective that one should seek to optimise must account for the description length of the observations as well as that of the model parameters. This highlights an important difference

between compression and generative models, with vanilla neural networks being sometimes unsuitable, due to the fact that they are *overparameterised* and of non-variable description length.

The Partition and Code (PnC) framework. Our algorithmic solution to the aforementioned challenges is the *Partition and Code (PnC)* framework. PnC is a *dictionary coding* algorithm that consists of three main steps: (a) Initially, a graph is partitioned into non-overlapping subgraphs, which in turn yields a decomposition of the graph likelihood. (b) Subsequently, frequent subgraphs are mapped to a (learned) *dictionary* - see Fig. 5.1. (c) Finally, each component of the decomposition is *entropy coded* using a distribution estimator. Additionally, we propose a practical instantiation of the framework that allows optimisation of all three components (partitioning, dictionary, distribution estimator) with gradient-based methods (section 5.6).

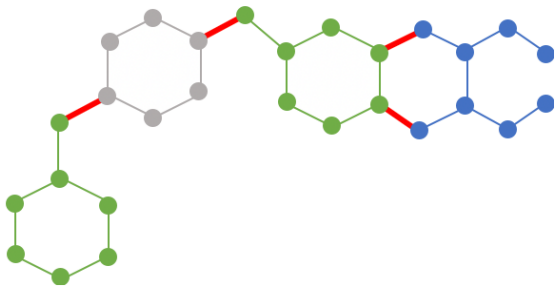


Figure 5.1: Illustration of the graph decomposition. The subgraph colours correspond to dictionary atoms a_1, a_2 and a_3. Cuts are denoted in red.

As we will thoroughly discuss, our framework attempts to find a trade-off between the aforementioned challenges: *C1*: Subgraphs are mapped isomorphically to the elements of the dictionary, thus leading to important compression gains, To ensure we can efficiently solve GI, we constrain the dictionary to only contain graphs of size up to a small constant. *C2*: Graph partitioning provides us with a decomposition of the distribution, which with the help of the dictionary is parameterised with only a handful of parameters that can be efficiently estimated. Additionally, *we use neural networks only to infer the decomposition of the distribution* (and not to predict the likelihood), and hence we can rely on overparameterisation without having to relay the NN parameters to the decoder. Finally, the description length of the dictionary (which accounts for the largest portion of the model parameters) can be optimised during training w.r.t. the total description length objective.

Our theoretical analysis reveals that PnC can significantly improve upon less sophisticated graph encoders and justifies the usefulness of both the "Partition" and the "Code" component. Specifically, we prove that under mild conditions on the underlying graph distribution, PnC

requires in expectation $\Theta(n^2)$ fewer bits than standard graph encodings, even if the latter are given access to an oracle that solves GI. Further, the dictionary induces additional savings of $\Theta(n)$ bits, with the gain being inversely proportional to the entropy of the distribution of the dictionary atoms. Thus, the more repetitive the patterns in the graph distribution are, the larger will be the compression benefits of PnC. These findings are corroborated by our empirical studies. We evaluate our framework on diverse real-world graph distributions and showcase compression gains with respect to both traditional graph compressors and more sophisticated specialised alternatives, including deep graph generative models.

5.2 Background: Information theory and graph encodings

Following the usual terminology in information theory and Minimum Description Length (MDL) theory [MacKay, 2003, Grünwald, 2007, Cover and Thomas, 2006], we assume an *observation space* $\mathscr{X}$, and a probability distribution $p_x(x)$ (or $p(x)$ - we will drop the subscript whenever the random variable is clear from the context), sometimes referred to as *probabilistic source*, producing samples from the observation space. We observe a dataset $\mathfrak{D} = \{x_1, x_2, \ldots, x_{|\mathfrak{D}|}\}$ of i.i.d. observations drawn from p. Note that, in the context of graphs, this setting is in contrast with most works on graph compression [Vigna and Boldi, 2004, Claude and Navarro, 2010, Lim et al., 2014, Dhulipala et al., 2016], where the target is to compress a *single* large network, such as a social network or a web graph.

Codes and distributions. Recall that a *source code* $h_{\text{enc}} : \mathscr{X} \to \mathbb{B}^*$ is a mapping from the observation space to a variable-length sequence of binary symbols and in lossless compression, we are interested in *non-sinular codes*, i.e. codes that are invertible. Of particular interest are the *uniquely decodable* codes, for which it holds that any concatenation of codewords can be uniquely mapped to a sequence of observations. This eliminates the need to encode delimiters between codewords. *Instantaneous* or *prefix codes* are a practical subcase, for which it holds that no codeword is a prefix of another codeword, and therefore sequences of codewords can be decoded in one pass. From now on, all the source codes that we will be dealing with will be prefix codes.

Typically we are only interested in the description length of a datapoint x under the code h_{enc}, i.e. the number of bits needed to encode the datapoint x, denoted with $\mathrm{L}(x; h_{\text{enc}})$, or

$\mathrm{L}(x)$ for brevity, rather than the particular instantiation of the code itself. An important property of uniquely decodable codes is the the well-known *Kraft–McMillan inequality* (l.h.s. in formula (5.1)):

$$\sum_{x\in\mathscr{X}} |\mathbb{B}|^{-\mathrm{L}(x;h_{\mathrm{enc}})} \leq 1 \quad \text{(code)} \iff q(x) = \frac{|\mathbb{B}|^{-\mathrm{L}(x;h_{\mathrm{enc}})}}{\sum_{x\in\mathscr{X}} |\mathbb{B}|^{-\mathrm{L}(x;h_{\mathrm{enc}})}} \quad \text{(distribution)}. \tag{5.1}$$

Conversely, for every set of natural numbers $\{\mathrm{L}(x;h_{\mathrm{enc}})\}_{x\in\mathscr{X}}$ that satisfies the l.h.s of Eq. (5.1), there exists a uniquely decodable code over $\mathscr{X}$ with these codeword lengths, as well as an *implicit distribution* $q(x)$ (r.h.s. of Eq. (5.1)). When the Kraft–McMillan inequality holds with strict inequality, we say that the code is *redundant*, while in the case of equality, we say that the code is *complete*. Using Eq. (5.1) and Gibbs' inequality we recover a variant of Shannon's source coding theorem. This asserts that the expected codeword length of any uniquely decodable code is lower bounded by the entropy of the true distribution (see [MacKay, 2003, Cover and Thomas, 2006]):

$$\mathbb{E}_{x\sim p(x)}[\mathrm{L}(x;h_{\mathrm{enc}})] \geq \mathbb{H}_{x\sim p(x)}[x] = \mathbb{E}_{x\sim p(x)}[-\log_{|\mathbb{B}|} p(x)], \tag{5.2}$$

with the equality attained when $\sum_{x\in\mathscr{X}} |\mathbb{B}|^{-\mathrm{L}(x;h_{\mathrm{enc}})} = 1$ and $\mathrm{L}(x;h_{\mathrm{enc}}) = -\log_{|\mathbb{B}|} p(x)$, i.e. when the implicit distribution is equal to the true one for all $x \in \mathscr{X}$. Since the latter does not guarantee that the codeword lengths will be natural numbers, one may set them as $\mathrm{L}(x;h_{\mathrm{enc}}) = \lceil -\log_{|\mathbb{B}|} q(x)\rceil$ (*Shannon–Fano code*), which satisfy the l.h.s. of Eq. (5.1) (since $\lceil -\log_{|\mathbb{B}|} q(x)\rceil \geq -\log_{|\mathbb{B}|} q(x)$), and therefore also constitute a uniquely decodable code. In this case, the expected description length becomes $\mathbb{E}_{x\sim p(x)}[\lceil -\log_{|\mathbb{B}|} q(x)\rceil] < \mathbb{E}_{x\sim p(x)}[-\log_{|\mathbb{B}|} q(x)] + 1$, and its optimal value is $\mathbb{H}_{x\sim p(x)}[x] + 1$.

Hence, instead of explicitly optimising for codes, we can optimise for distributions:

$$\min_q \mathbb{E}_{x\sim p(x)}[-\log_{|\mathbb{B}|} q(x)], \tag{5.3}$$

which, at optimality, is guaranteed in expectation to incur at most one extra symbol compared to the lower bound, using a properly selected code. The selection of a code given a distribution so as to approach the entropy lower bound as closely as possible is commonly known as *entropy coding*, with the most widespread paradigms being Huffman coding [Huffman, 1952], arithmetic coding [Pasco, 1976, Rissanen, 1976, Rissanen and Langdon, 1979, Witten et al., 1987] and

asymmetric numeral systems [Duda, 2009, Duda, 2013]. From now on, for brevity, we will assume that our alphabet is the binary one, and we will express the information content in bits by using a base-2 logarithm log instead of $\log_{|\mathbb{B}|}$.

Uniform codes. An important principle that we follow is that whenever we cannot make any assumptions about an underlying distribution, or modelling it is impractical or expensive, then the uniform distribution q^{unif} is chosen for encoding. The reason is that uniform distribution is *worst-case optimal* [Grünwald, 2007] in two ways: (1) over all the data, i.e. $\max_{x\in\mathscr{X}}[-\log q^{\text{unif}}(x)] = \min_q \max_{x\in\mathscr{X}}[-\log q(x)]$, and (2) over distributions, i.e. for any non-uniform unknown distribution p, there exists at least one distribution q such that $\mathbb{E}_{x\sim p}[-\log q(x)] > \mathbb{E}_{x\sim p}\log[-q^{\text{unif}}(x)]$.

Common graph encodings. Denote a labelled graph with $G = (\mathcal{V}, \mathcal{E}) \in \mathfrak{G}$, where $\mathfrak{G}$ is our observation space and let $n = |\mathcal{V}|, m = |\mathcal{E}|$. Many standard graph encodings are based on variations of the above principle of uniform encodings. For example, one can decompose the graph distribution as $q(G) = q(G \mid n)q(n)$ or $q(G) = q(G \mid n, m)q(m \mid n)q(n)$ and use a uniform encoding for at least of one the terms. Such are the following encodings we commonly encounter:

- *uniform*: $\mathrm{L}^{\text{unif}}(G \mid n) = n^2$, which is optimal for a uniform distribution over labelled graphs of n vertices. This can be implemented by storing the binary adjacency matrix.
- *edge list*: $\mathrm{L}^{\text{EL}}(G \mid n, m) = m \log n^2$, which is optimal for a uniform distribution over labelled graphs of n vertices and m edges, where multi-edges are allowed. This can be implemented by storing the pairs of connected vertices. Observe that, although commonly used in practice, this model is in fact always redundant for simple graphs.
- *Erdős–Rényi (ER)*: $\mathrm{L}^{\text{ER}}(G \mid n, m) = \log \binom{n^2}{m}$, which is optimal for the Erdős–Rényi model, i.e. a uniform distribution over *simple* labelled graphs of n vertices and m edges.

Uniform encodings can be also used for n and m, i.e. $\mathrm{L}(n) = \log(n_{\max} + 1)$, where $n_{\max}$ is an upper bound on the vertex count and $\mathrm{L}(m) = \log(n^2 + 1)$.[1] The latter, differently from the uniform encoding, is optimal when all edges counts are equally probable, and when combined with the ER model compresses more efficiently graphs that are either very sparse or very dense as the number of possible graphs with m edges is maximised when $m = n^2/2$. Note that all the

[1] An alternative is an *universal integer code*, which is optimal for monotonically decreasing distributions.

formulas above apply to directed graphs with self-loops allowed. To modify them for undirected graphs with or without self-loops, we simply replace n^2 with $n^2 - n$ or $\binom{n}{2}$ respectively.

We will collectively refer to any of these encodings as *null encodings*, paraphrasing the term *null model*, which is commonly used in network science to refer to random graph models that reflect a *null hypothesis*, i.e. a hypothesis that does not display any particular distinct patterns apart from satisfying a set of predefined constraints. In our framework, we will make use of a null model in order to encode low-probability subgraphs that cannot be mapped into the dictionary (see section 5.4.2), and in practice, we use the ER model for undirected graphs without self-loops, with uniform vertex and edge count probabilities, that reads:

$$\mathrm{L}^{\mathrm{null}}(G) := \log(n_{\max} + 1) + \log\left(\binom{n}{2} + 1\right) + \log\binom{\binom{n}{2}}{m}. \tag{5.4}$$

Binary entropy. An important notion that appears in our theoretical analysis is the *binary* entropy, i.e. the entropy of a Bernoulli variable with success probability θ. To distinguish it from the general notion of entropy we write $\mathrm{H}(\theta) = -\theta \log \theta - (1-\theta)\log(1-\theta)$. It holds that $0 \leq \mathrm{H}(\theta) \leq 1$, where the l.h.s equality is satisfied for $\theta \in \{0, 1\}$ and the r.h.s. for $\theta = 1/2$.

5.3 The principles of unlabelled graph compression

Problem formulation: Labelled vs unlabelled lossless graph compression Let $\mathfrak{G}$ be a graph space, i.e. a space of *labelled* graphs. In graph compression, we are tasked with designing a function $h_{\mathrm{enc}} : \mathfrak{G} \to \mathbb{B}^*$. Observe, that similarly to chapter 4, this is an *analysis* problem and falls under the *space of spaces* setup (see figure 1.2 right). In labelled lossless graph compression, each graph in $\mathfrak{G}$ is a distinct element, i.e. two isomorphic graphs $G_1 \simeq G_2$ are considered distinct, and therefore h_{enc} should be injective, i.e. $h_{\mathrm{enc}}(G_1) = h_{\mathrm{enc}}(G_2) \Rightarrow G_1 = G_2$, in order to guarantee the existence of a decoder function $h_{\mathrm{dec}} : \mathbb{B}^* \to \mathfrak{G}$,[2] for which it holds that $h_{\mathrm{dec}}\big(h_{\mathrm{enc}}(G)\big) = G,\ \forall G \in \mathfrak{G}$. In unlabelled lossless graph compression, two graphs G_1, G_2 are considered distinct if and only if they are not isomorphic, and therefore h_{enc} should be *isomorphism-injective (GI-injective)*, i.e. $h_{\mathrm{enc}}(G_1) = h_{\mathrm{enc}}(G_2) \Rightarrow G_1 \simeq G_2$. This is necessary in order to guarantee the existence of a decoder function $h_{\mathrm{dec}} : \mathbb{B}^* \to \mathfrak{G}$, for which it holds

[2]Technically speaking, the domain of h_{dec} should be $h_{\mathrm{enc}}[\mathfrak{G}]$, i.e. h_{dec} may be undefined for some codewords in $\mathbb{B}^*$. This detail was avoided for simplicity and with our definition, we imply an arbitrary extension of h_{dec} to the entire $\mathbb{B}^*$.

that $h_{\text{dec}}\big(h_{\text{enc}}(G)\big) \simeq G$, $\forall G \in \mathfrak{G}$, or in other words the decoder should output a graph isomorphic to the one originally encoded. Note that from the perspective of labelled graphs, the compression is lossy. Finally, as we will see in Theorem 5.1 any *optimal* unlabelled graph compressor is additionally required to be *isomorphism invariant (GI-invariant)*, i.e. $G_1 \simeq G_2 \Rightarrow h_{\text{enc}}(G_1) = h_{\text{enc}}(G_2)$.

How to design the encoder and decoder functions?

Labelled graph compression amounts to designing a uniquely- decodable code. As we saw in section 5.2, designing such codes is well understood, and in our case amounts to estimating the distribution of labelled graphs. However, this result does not straightforwardly apply to unlabelled graphs, since the encoder function is no longer injective. To overcome this we can design the encoder as a two-step process with a composition of functions $h_{\text{enc}} = h_{\text{d-enc}} \circ h_{\text{g-enc}}$.

- In the first step, graphs are mapped to an intermediate representation with a *GI-injective function* $h_{\text{g-enc}} : \mathfrak{G} \to \mathcal{Z}$
- In the second step, intermediate representations are mapped to codewords with an *injective function* $h_{\text{d-enc}} : \mathcal{Z} \to \mathbb{B}^*$.

We will call $h_{\text{g-enc}}$ as the **graph encoder**, and $h_{\text{d-enc}}$ as the **distribution encoder**. Then, the decoder will be $h_{\text{g-dec}} \circ h_{\text{d-dec}}$, where $h_{\text{d-dec}}$ is the inverse of $h_{\text{d-enc}}$ and $h_{\text{g-dec}}$ is a function for which it holds that $h_{\text{g-dec}}\big(h_{\text{g-enc}}(G)\big) \simeq G$. Now let us examine each of the two steps separately.

The graph encoder: Designing an optimal $h_{\text{g-enc}}$ is quite challenging: GI-invariant and GI-injective functions are known as *complete graph invariants* ([Dubois et al., 2021] use the term *maximal invariants*) and computing them is is at least as hard as graph isomorphism. Additionally, the decoder $h_{\text{g-dec}}$ needs to be manually designed (contrary to $h_{\text{d-dec}}$, as we will see below) and we need to make sure that it is efficiently computable. One possible solution is to solve a *graph canonicalisation* problem and define $h_{\text{g-enc}}$ as a function that returns the same *canonical form* for all graphs in an equivalence class, i.e. $\mathcal{Z} \subseteq \mathfrak{G}$ and $G \simeq h_{\text{g-enc}}(G)$. In this case, $h_{\text{g-dec}}$ can be the identity function, which is obviously efficiently computable. However, the problem of the high computational complexity of graph canonicalisation persists. There exist several heuristics for this that are either efficiently computable but are approximate and canonicalise graphs with high probability (e.g. the Weisfeiler-Leman algorithm [Weisfeiler and Leman, 1968] or

GNNs), or exact, but not efficiently computable for all graphs (e.g. the Nauty tool [McKay et al., 1981, McKay and Piperno, 2014]). However, similarly to graph isomorphism, to date there is no efficiently computable algorithm that returns a complete graph invariant/canonical graph form for all graphs.

To address this, we should resort to a necessary compromise. In compression, although it might seem surprising at first glance, having the property of GI-injectivity *is necessary*, while having the property of GI-invariance *is not*. In particular, a non-GI-injective $h_{\text{g-enc}}$ allows two non-isomorphic graphs to be mapped to the same intermediate representations and therefore to the same codewords. Then it will be impossible for the decoder $h_{\text{g-dec}}$ to reconstruct the isomorphism class of both of them. On the other hand, GI-invariance can be dropped. This will result in the possibility of having two isomorphic graphs mapped to different intermediate representations and therefore to different codewords. This does not prevent the decoder from mapping these representations to the same isomorphism class and thus it is possible to achieve zero information loss. Overall, dropping GI-invariance does not threaten the correctness of the compressor, but as we will see below leads to suboptimal bit rates.

The distribution encoder: Regarding the function $h_{\text{d-enc}}$, since it is injective, we can use the aforementioned information-theoretic results and design instead a distribution on intermediate representations $q_z : \mathcal{Z} \to [0, 1]$. The probabilities can be later translated into codewords using, e.g. arithmetic coding techniques, as in [Steinruecken, 2015]. The same procedure allows us to obtain the intermediate representation decoder $h_{\text{d-dec}} : \mathbb{B}^* \to \mathcal{Z}$ for free.

How to optimise the encoder-decoder?

To begin with, as we saw in section 5.2 optimising the distribution encoder amounts to estimating the probability distribution of the intermediate representations p_z. Now the objective of Eq (5.3) becomes:

$$\mathbb{E}_{z\sim p_z}[-\log q_z(z)] = \mathbb{E}_{G\sim p_G}[-\log q_z\big(h_{\text{g-enc}}(G)\big)] \tag{5.5}$$

and minimising the empirical objective amounts to the following problem:

$$\min_{q_z} \frac{1}{|\mathfrak{D}|} \sum_{G\in\mathfrak{D}} -\log q_z\big(h_{\text{g-enc}}(G)\big) \tag{5.6}$$

Additionally, it s possible to parametrise $h_{\text{g-enc}}$ as well and therefore Eq. (5.6) needs to be adjusted in order to also optimise for $h_{\text{g-enc}}$.

An additional aspect that we need to consider, is that when the encoders are parametric, i.e. $q_z(\cdot;\varphi)$ and $h_{\text{g-enc}}(\cdot;\vartheta)$, the decoders $h_{\text{d-dec}}$ and $h_{\text{g-dec}}$ might also depend on φ and ϑ respectively. For $h_{\text{d-dec}}$ this is always the case when we use entropy coding since both the encoder and the decoder need to maintain the estimator of the distribution in order to match codewords to data. For $h_{\text{g-dec}}$, sometimes we can define it in a non-parametric way (as in the case of graph canonicalisation or graph partitioning - e.g. see Appendix C.2.4) independently of the parameters of the encoder. In case the decoders do depend on the encoder parameters φ and ϑ, these need to be stored and transmitted along with the data. Although this is frequently ignored in the field of neural compression, *it is of paramount importance when the number of parameters, or more precisely their description length* $\mathrm{L}\big((\varphi,\vartheta)\big)$*, is large.* In particular, the empirical objective becomes:

$$\min_{\varphi,\vartheta} \frac{1}{|\mathfrak{D}|}\Bigg(\sum_{G\in\mathfrak{D}} -\log q_z\big(h_{\text{g-enc}}(G;\vartheta);\varphi\big) + \mathrm{L}\big((\varphi,\vartheta)\big)\Bigg), \tag{5.7}$$

which is also known as the two-part code *Minimum Description Length (MDL)* principle. This is similar to the *Occam's razor* principle and is reminiscent of the bias-variance or empirical error-model complexity trade-off. Classical learning theory relates model complexity to generalisation, i.e. the larger the model complexity the larger the worst-case deviation of the empirical error from the true error. However, in modern deep learning, this relation is challenged and estimators modelled with *overparameterised* neural networks often showcase improved generalisation. In fact, overparameterisation is commonly argued to be one of the decisive factors for the successful optimisation and generalisation of NNs with gradient-based methods [Neyshabur et al., 2018, Du et al., 2019d, Arora et al., 2019, Allen-Zhu et al., 2019, Allen-Zhu et al., 2019, Bubeck and Sellke, 2021, Belkin et al., 2019, Nakkiran et al., 2020]. This is a pertinent issue for neural compression as the description length of these models might even end up being larger than that of the data, unless they undergo a process for *model compression* themselves (see Appendix C.3.2). Moreover, it is unclear how to optimise Eq. (5.7) when φ represents the parameters of a neural network since their description length is usually fixed.

The benefits of unlabelled graph compression

It is worth explaining here why it is beneficial to compress graphs as unlabelled objects when the vertex labelling is uninteresting. In particular, we can assume that the labelled graphs we encounter in practice are sampled as follows. First, an unlabelled graph is sampled from a distribution p_S on equivalence classes (i.e. on the quotient $\mathfrak{G}/\simeq$). Following, a vertex labelling is assigned by sampling uniformly at random from all possible vertex labellings, inducing a labelled graph distribution $p_G(G) = \frac{p_S(S)}{|S|}$, $\forall S \in \mathfrak{G}/\simeq, \forall G \in S$. It is not hard to show that the entropy (and as a result the expected description length of an optimal compressor) of the random variable of unlabelled graphs will be asymptotically smaller than the entropy of the random variable of labelled graphs by $O\big(\mathbb{E}[\log n!]\big)$ bits, where n the number of vertices. This result can be found in [Choi and Szpankowski, 2012] and is recovered using rate-distortion theory for invariant tasks in [Dubois et al., 2021].

When doing a compromise on GI-invariance, our optimal compressor will incur a penalty in its expected description length compared to the optimal compressor that uses a GI-invariant and GI-injective $h_{\text{g-enc}}$. We quantitatively characterise this penalty which is described by the following theorem:

Theorem 5.1. *Consider a distribution p_S over unlabelled graphs $S \in \mathfrak{G}/\simeq$ and a distribution p_G over labelled graphs $G \in \mathfrak{G}$ such that $p_G(G) = \frac{p_S(S)}{|S|}$, $\forall S \in \mathfrak{G}/\simeq, \forall G \in S$. Denote the orbit of a graph G under the isomorphism equivalence relation with $\mathrm{Orb}(G) = \mathrm{Orb}_{\mathfrak{G}/\simeq}(G) = \{G' \in \mathfrak{G} \mid G' \simeq G\}$. Further, consider a compression scheme for unlabelled graphs that first represents a labelled graph in an intermediate representation using a GI-injective graph encoder $h_{g\text{-}enc} : \mathfrak{G} \to \mathcal{Z}$ and then losslessly compresses intermediate representations using entropy coding with a distribution model $q_z : \mathcal{Z} \to [0, 1]$. Denote with p_z the pushforward measure $h_{g\text{-}enc}\#p_G$. Then, the following holds for the optimal compressor of intermediate representations:*

$$\min_{q_z} \mathbb{E}_{z\sim p_z}\big[-\log q_z(z)\big] - \mathbb{H}_{S\sim p_S}[S] = \mathbb{E}_{G\sim p_G}\left[\log\left(\frac{\big|\mathrm{Orb}(G)\big|}{\big|\{G' \in \mathfrak{G} \mid h_{g\text{-}enc}(G) = h_{g\text{-}enc}(G')\}\big|}\right)\right]. \tag{5.8}$$

Intuitively, this tells us that the larger the number of isomorphic graphs that are mapped to the same intermediate representation (on expectation) the smaller the penalty that we will incur in the optimal expected description length. The penalty is zero when $h_{\text{g-enc}}$ is GI-invariant and maximum - $O\big(\mathbb{E}[\log n!]\big)$ - when $h_{\text{g-enc}}$ is injective, i.e. we perform labelled graph compression,

which recovers the result of [Choi and Szpankowski, 2012]. Please refer to Appendix C.1.1 for the proof.

5.4 The Partition & Code (PnC) framework

To showcase the rationale behind our compressor, we will start by discussing the design choices that apply to both labelled and unlabelled graph compression and subsequently, we will specify the extra steps, that are specific to unlabelled graph compression, and will be taken to further reduce the bit rate. First, we will investigate the challenge *C2* that is common to both setups. In neural compression, it is customary to estimate the probability distribution of the data using deep generative models. However, as already discussed, typically a large amount of data is required to obtain good estimates, while the true objective demands a joint optimisation of the description length of the model parameters, along with the description length of the data, a goal that is currently unknown how to achieve when the model is designed as a neural network.

Dictionary coding

To address this, we will shift our attention from deep generative models to other parametric distributions that are more amenable to our true objective. In particular, we draw inspiration from traditional adaptive compressors, and in particular, from dictionary coding [Ziv and Lempel, 1977, Ziv and Lempel, 1978, Welch, 1984, Deutsch, 1996] which underlies some of the currently most popular compression protocols used in practice, such as `zip`, `gzip`, `pdf`, `GIF`, `PNG` and others, and tokenisation methods (e.g. byte-pair encoding [Gage, 1994]) used by large language models, such as GPT-3 [Brown et al., 2020].

Instead of strictly modelling all possible dependencies in the graph distribution (which would entail a heavy parametrisation), dictionary coding identifies frequent patterns of variable size that dominate the data observations. Subsequently, one can parametrically model only the dependencies between dictionary patterns (the *atoms*) and use a generic non-parametric model for the remainder of the information, which, with high probability, will result in only a small increase in the overall bit rate. The description length of the model parameters in this case is dominated by the description length of the dictionary itself. The advantage compared to other adaptive or neural compressors, is that the dictionary is not only adaptive/learnable but

has also a variable description length (depending on the atoms that it contains) which we can optimise.

Partitioning

To adapt this idea for graph compression, it remains to define the nature of the atoms that will be stored in the dictionary. In text compression and image compression, the dictionary atoms are strings and image patches respectively. An immediate analogue for labelled graphs is to define dictionary atoms as *sub-matrices* of the adjacency matrix. An obvious disadvantage of this is that forming a dictionary with sub-matrices of arbitrarily labelled graphs might result in atoms of low frequencies (in other words to a high entropy dictionary) and therefore might not be amenable to dictionary coding.

To address this, vertex re-ordering (see section 5.7) methods are usually employed in order to transform the adjacency matrix into a matrix with sub-matrices that are more likely to be repeatable, which, when paired with traditional dictionary coding algorithms, achieve significant compression gains.[3] This idea is also inspired by traditional compression algorithms, and in particular, the *Burrows–Wheeler transform* [Burrows and Wheeler, 1994] that re-orders the symbols of a sequence in order to make it more amenable to encoders, such as the *move-to-front transform* [Ryabko, 1980, Bentley et al., 1986] and *run-length-encoding*, that take advantage of contiguous repeated elements in a sequence.

Pairing vertex-reordering with sequence-based dictionary encoding is similar to *graph partitioning*, where now instead of sequences, the dictionary elements can be (labelled) *subgraphs* (main-diagonal blocks of the adjacency matrix) and *cuts*, i.e. edges between the vertices of the subgraphs (off-diagonal blocks). This is the approach we take in this work, where our dictionary contains subgraphs as atoms. Cuts can be also stored in a dictionary, but since they are less likely to be repeatable, in our case they are encoded using a generic non-parametric encoding. It is important to mention here, that many real-world networks display *community-structure*, i.e. their adjacency matrices are close to block-diagonal. This allows us to attain important compression gains by only using a partitioning that uncovers these communities (without the help of a dictionary) and encodes them with a non-parametric model, suitable for this assumption. This is theoretically supported by Theorem 5.2.a.

[3] In the case of labelled graphs, the re-ordering may also be transmitted so as to reconstruct the original graph.

Unlabelled subgraphs

To address challenge *C1*, an ideal graph encoder should ensure that the above steps are performed in a GI-invariant way. In particular, every pair of isomorphic graphs should admit the same vertex re-ordering and partitioning. Since this cannot be done in an efficiently computable way, we propose an alternative that strikes a balance between GI-invariance and computational complexity. In particular, instead of mapping labelled subgraphs to dictionary atoms, i.e. by comparing their adjacency matrices for equality, we restrict the dictionary to contain only non-isomorphic atoms (of arbitrary but fixed vertex ordering) and map unlabelled subgraphs via graph isomorphism testing. This has a dual purpose. First, it reduces the number of possible vertex re-orderings that can result in the same representation, and second, it reduces the bit rate by construction (see Theorem 5.3). The reduction in computational complexity is guaranteed by bounding the maximum size of the dictionary atoms to a small number $k = O(1)$, i.e. constant w.r.t. the size of the graphs to be compressed, for which GI is efficiently computable.

In summary, our pipeline consists of three main modules: a *partitioning* module and a *dictionary* module, which collectively constitute the graph encoder, and a *distribution encoder* module. (a) The partitioning module is responsible for decomposing the graph into disjoint subgraphs and cross-subgraph edges (or *cuts*). (b) The dictionary is a small collection of subgraphs (*atoms*) that are recurrent in the graph distribution. The dictionary module maps the partitioned subgraphs to atoms in the dictionary, allowing us to represent the graph as a collection of atom indices, non-dictionary subgraphs, and cuts. (c) This representation is given as input to the distribution encoder that calculates its probability according to an estimator. Finally, the probability is translated into a bitstream using entropy coding.

Remark. A minor detail that it is appropriate to mention is that the exact construction of $h_{\text{g-enc}}$ and $h_{\text{g-dec}}$ when using a partitioning algorithm and a dictionary which maps subgraphs isomorphically entails some technical steps that relate to the ordering of the vertices of each subgraph. To avoid overloading the main text with unnecessary details we relegate the discussion on the exact way to construct these functions to the Appendix C.2.4 and will compromise with a slight abuse of notation in the following section where we will identify $h_{\text{g-enc}}$ with the partitioning algorithm PART.

5.4.1 Graph encoder

Partitioning. Consider a graph $G = (\mathcal{V}, \mathcal{E})$. The first step of PnC is to employ a partitioning algorithm $\mathsf{PART}(\cdot; \boldsymbol{\theta})$, where $\boldsymbol{\theta}$ denote the optional parameters of the algorithm, to decompose the graph into b *disjoint* and *induced* subgraphs:

$$\mathsf{PART}(G; \boldsymbol{\theta}) = (\mathcal{H}, C) \quad \text{and} \quad \mathcal{H} = \{H_1, H_2, \cdots, H_b\}, \tag{5.9}$$

The subgraphs that a partitioning yields should have the following properties: (1) $H_i = \{\mathcal{V}_{H_i}, \mathcal{E}_{H_i}\}$ with $\mathcal{V}_{H_i} \subseteq \mathcal{V}$ and $\mathcal{E}_{H_i} = \mathcal{E} \bigcap \mathcal{V}_{H_i} \times \mathcal{V}_{H_i}$, i.e. the subgraphs are induced, (2) $\mathcal{V}_{H_i} \cap \mathcal{V}_{H_j} = \emptyset$, $\forall i, j \in [b], i \neq j$, $\mathcal{V} = \bigcup_{i=1}^{b} \mathcal{V}_{H_i}$, i.e. the subgraph vertex sets partition $\mathcal{V}$ and (3) $C = \{\mathcal{V}_C, \mathcal{E}_C\}$ is a b-partite graph containing all the *cuts* of the partitioning, i.e. $\mathcal{V}_C = \mathcal{V}, \mathcal{E}_C = \mathcal{E} - \bigcup_i \mathcal{E}_{H_i}$.

Dictionary. Partitioning-based compression methods, such as [Peixoto, 2019], encode the block adjacency matrix that the partitioning yields using a non-parametric distribution encoder. Although this is very effective for networks with community structure, it relies on a strong assumption, while it does not explicitly deal with graph isomorphism. Instead we propose to exploit regularities in the output of PART using dictionary coding and store the most commonly occurring subgraphs. Concretely, we define a dictionary $\mathcal{D}$ to be a collection of non-isomorphic graphs (usually connected), called atoms, from some graph universe $\mathfrak{U}$:

$$\mathcal{D} = \{\alpha_1, \alpha_2, \ldots, \alpha_{|\mathcal{D}|}\}, \quad \text{where} \quad \alpha_i = (\mathcal{V}_{\alpha_i}, \mathcal{E}_{\alpha_i}) \in \mathfrak{U}, \text{ and } \alpha_i \not\simeq \alpha_j, \forall;\ i \neq j. \tag{5.10}$$

As mentioned above, the dictionary mapping is implemented via graph isomorphism using the following function $\psi_{\mathcal{D}}(\cdot)$:

$$\psi_{\mathcal{D}}(H) = \begin{cases} i, & \text{iff } \exists\ \alpha_i \in \mathcal{D} \text{ s.t. } \alpha_i \simeq H \\ 0, & \text{else,} \end{cases} \tag{5.11}$$

which is a function since by the definition of the dictionary there will be at most one atom isomorphic to each graph. Finally, our graph encoder function becomes $h_{\text{g-enc}}(G; \vartheta) := \mathsf{PART}\big(G; (\boldsymbol{\theta}, \mathcal{D})\big)) = (\mathcal{H}_{\text{dict}}, \mathcal{H}_{\text{null}}, C)$, where

$$\mathcal{H}_{\text{dict}} = \wr \psi_{\mathcal{D}}(H) \mid H \in \mathcal{H}, \exists \alpha \in \mathcal{D} \text{ s.t. } \alpha \simeq H \int \text{ and } \mathcal{H}_{\text{null}} = \{H \in \mathcal{H} \mid \forall \alpha \in \mathcal{D},\ \alpha \not\simeq H\}. \tag{5.12}$$

Note that it is unnecessary to preserve the labelling of the vertices of the non-dictionary subgraphs in $\mathcal{H}_{\text{null}}$ and those of the cut matrix C, and we can instead encode them using block adjacency matrices (here we slightly abuse notation for simplicity). The order with which the subgraph block matrices are encoded can be arbitrary but should correspond to that of the cut block matrices to ensure unique decodability. However, the atom indices of dictionary subgraphs $\mathcal{H}_{\text{dict}}$ can be encoded as a multiset since the dictionary provides us with a way to obtain a canonical index ordering. All the above are explained in detail in Appendix C.2.4.

The reason for a dual subgraph encoding. The choice for the dual encoding into dictionary and non-dictionary subgraphs is made for the following reasons: (a) It allows PART to choose non-dictionary atoms. This is crucial to our approach, since constraining the partitioning to specific isomorphism classes would significantly complicate optimisation. (b) Further, it enables us to maintain a small dictionary and avoid storing subgraphs that will be infrequent in the distribution. Instead, we give the possibility to encode them with a non-parametric model whenever this leads to larger compression gains.

5.4.2 Distribution encoder

The last step entails compressing G by encoding the output of PART using a probability distribution, i.e. $q\big(h_{\text{g-enc}}(G;\vartheta)\big) = q\big(\mathsf{PART}\big(G;(\boldsymbol{\theta},\mathcal{D})\big)\big)$. The distribution is decomposed as follows:

$$\begin{aligned} q\big(\mathsf{PART}\big(G;(\boldsymbol{\theta},\mathcal{D})\big)\big) &= q(\mathcal{H}_{\text{dict}},\mathcal{H}_{\text{null}},C) \\ &= q(\mathcal{H}_{\text{dict}},\mathcal{H}_{\text{null}})q(C \mid \mathcal{H}_{\text{dict}},\mathcal{H}_{\text{null}}) \\ &= q(b_{\text{dict}},b)q(\mathcal{H}_{\text{dict}},\mathcal{H}_{\text{null}} \mid b_{\text{dict}},b)q(C \mid \mathcal{H}_{\text{dict}},\mathcal{H}_{\text{null}}) \\ &= q(b_{\text{dict}},b)q(\mathcal{H}_{\text{dict}} \mid b_{\text{dict}},b)q(\mathcal{H}_{\text{null}} \mid \mathcal{H}_{\text{dict}},b_{\text{null}})q(C \mid \mathcal{H}_{\text{dict}},\mathcal{H}_{\text{null}}), \end{aligned} \tag{5.13}$$

where $b, b_{\text{dict}}, b_{\text{null}}$ are the total number of subgraphs, the number of dictionary subgraphs and the number of non-dictionary subgraphs in the partition respectively. This corresponds to a description length $\mathrm{L}\left(\mathsf{PART}\big(G;(\boldsymbol{\theta},\mathcal{D})\big);q\right) = -\log q\big(\mathsf{PART}\big(G;(\boldsymbol{\theta},\mathcal{D})\big)$. Further, we may parameterise q with a vector of parameters $\boldsymbol{\varphi}$.

Number of subgraphs. First, we encode b along with b_{dict} (which uniquely determine b_{null}). We use a *categorical* distribution $q(b)$ for b and a *binomial* distribution for b_{dict} conditioning on the

former. The binomial is parameterised by the probability of a "successful" outcome δ, where $1-\delta = q(H \in \mathcal{D})$ is the probability of an arbitrary subgraph belonging in the dictionary, and the outcomes are assumed to be independent:

$$q(b_{\text{dict}}, b; \boldsymbol{\varphi}) = \text{Binomial}(b_{\text{dict}} \mid b; \boldsymbol{\varphi}) q(b) = \binom{b}{b_{\text{dict}}} (1-\delta)^{b_{\text{dict}}} \delta^{b-b_{\text{dict}}} q(b), \tag{5.14}$$

Dictionary subgraphs. The dictionary subgraphs are encoded via their corresponding indices into the graph dictionary. As we mentioned above, these indices form a multiset (intuitively, a histogram). We use a *multinomial* distribution to encode them, by first conditioning on the number of dictionary subgraphs b_{dict}. As a generalisation of the binomial, the multinomial distribution is parametrised by the probabilities of each possible outcome, i.e. the probability of the appearance of each atom, denoted with $q(\alpha)$, and the outcomes are again assumed to be independent:

$$q(\mathcal{H}_{\text{dict}} \mid b_{\text{dict}}, b; \boldsymbol{\varphi}) = q(\mathcal{H}_{\text{dict}} \mid b_{\text{dict}}; \boldsymbol{\varphi}) = \text{Multinomial}(b_1, \ldots, b_{|\mathcal{D}|} \mid b_{\text{dict}}; \boldsymbol{\varphi}) = b_{\text{dict}}! \prod_{\alpha \in \mathcal{D}} \frac{q(\alpha)^{b_\alpha}}{b_\alpha!}, \tag{5.15}$$

where $b_i = \left|\{\psi_{\mathcal{D}}(H) \in \mathcal{H}_{\text{dict}} \mid \psi_{\mathcal{D}}(H) = i\}\right|$ and $\sum_{i \in |\mathcal{D}|} b_i = b_{\text{dict}}$.

Non-Dictionary subgraphs. For the non-dictionary subgraphs, we encode their adjacency matrices independently according to the non-parametric null model of Eq. (5.4):

$$q(\mathcal{H}_{\text{null}} \mid \mathcal{H}_{\text{dict}}, b_{\text{null}}; \boldsymbol{\varphi}) = q(\mathcal{H}_{\text{null}} \mid b_{\text{null}}; \boldsymbol{\varphi}) = \prod_{H \in \mathcal{H}_{\text{null}}} q^{\text{null}}(H). \tag{5.16}$$

Cuts. We encode the cuts conditioned on the subgraphs, also using a non-parametric null model for multi-partite unidrected graphs similar to [Peixoto, 2013]. Denote the vertex count of subgraph H_i as k_i. Further denote with $\boldsymbol{m}_c = \{m_{1,1}, m_{1,2}, \ldots, m_{b-1,b}\}$ the vector containing the number of edges between each subgraph pair i, j and $m_c = \sum_{i<j}^{b} m_{ij}$. The b-partite graph C containing the cuts will be encoded hierarchically, i.e. first we encode the total edge count m_c, then the pairwise counts $\boldsymbol{m}_c$ and finally, for each subgraph pair, we independently encode the arrangement of the edges. For each of these cases, a uniform encoding is chosen, following the rationale mentioned in section 5.2. Hence, calculating the length of the encoding boils down to

enumerating possible outcomes:

$$
\begin{aligned}
q(C \mid \mathcal{H}; \boldsymbol{\varphi}) &= q(C, m_c, \boldsymbol{m}_c \mid \mathcal{H}; \boldsymbol{\varphi}) = q(m_c \mid \mathcal{H}; \boldsymbol{\varphi}) q(\boldsymbol{m}_c \mid m_c, \mathcal{H}; \boldsymbol{\varphi}) q(C \mid \boldsymbol{m}_c, m_c, \mathcal{H}; \boldsymbol{\varphi}) \\
&= \left(\left(1 + \sum_{j>i}^{b} k_i k_j\right) \cdot \binom{b(b-1)/2 + m_c - 1}{m_c} \cdot \prod_{j>i}^{b} \binom{k_i k_j}{m_{ij}} \right)^{-1}
\end{aligned}
\tag{5.17}
$$

We make the following remarks: (a) The encoding is the same regardless of the isomorphism class of the subgraphs, and the only dependence arises from their number, as well as their vertex counts. (b) Small cuts are prioritised, thus the encoding has an inductive bias towards distinct clusters in the graph. (c) Our cut encoding bears resemblance to those used in non-parametric Bayesian inference for SBMs (e.g., see the section C.2.1 on the baselines and [Peixoto, 2019] for a detailed analysis of a variety of probabilistic models), although a central difference is that in these works the encodings also take the vertex ordering into account.

Alternative parametrisations. Observe that when parametrising our distribution we implicitly make several assumptions (more precisely our estimator will be optimal should these assumptions hold) that may be weakened. For example, one may assume that dictionary subgraphs are not sampled independently and model $q(b_{\text{dict}} \mid b; \boldsymbol{\varphi})$ with a categorical distribution for each b and $q(b_1, \ldots, b_{|\mathcal{D}|} \mid b_{\text{dict}}; \boldsymbol{\varphi})$ with an autoregressive model. Moreover, the non-dictionary subgraphs can be modelled with a generative model for a sequence of binary matrices, e.g. a doubly autoregressive (one for the adjacency and one for the sequence) model, conditioned on a representation of the histogram $\mathcal{H}_{\text{dict}}$ and the scalar b_{null}. This type of model can be also used to model the cuts, again conditioning on a representation of $\mathcal{H}_{\text{dict}}$, as well as on a representation of $\mathcal{H}_{\text{null}}$. It is evident though, that all these improvements will significantly increase the description length $\mathrm{L}(\boldsymbol{\varphi})$, which is non-variable and cannot be optimised. Our current distribution estimator learnable parameter set is $\boldsymbol{\varphi} = \left(\delta, \{q(b)\}_{b=b_{\min}}^{b_{\max}}, \{q(\alpha)\}_{\alpha \in \mathcal{D}}\right)$ which enjoys a small description length by construction, negligible compared to the description length of the data or the dictionary.

5.4.3 Selecting a hypothesis by minimising the total description length

In order to optimise the two-part MDL objective of Eq. (5.7), we need to define the description length of the model parameters. First off, we examine the parameters of the graph encoder $\vartheta = (\boldsymbol{\theta}, \mathcal{D})$. Importantly, the parameters of the partitioning algorithm $\boldsymbol{\theta}$ do not need to

be transmitted to the decoder, since the graph decoder $h_{\text{g-dec}}$ is non-parametric and can be defined independently of the parameters of the encoder (please see Appendix C.2.4 for the exact definition of the graph decoder). Therefore, PART can be safely overparameterised (e.g. with a neural network) and can do all the "heavy-lifting" by selecting subgraphs that will be convenient for dictionary coding.

Regarding the dictionary, there are two viable choices: (a) If the universe is small enough to be efficiently enumerable and we assume that the adjacency graphs of the universe are public (accessible to both the encoder and the decoder), then storing the dictionary amounts to storing the indices of the atoms in the universe. In this case, a simple non-parametric encoder can perform a uniform encoding of the size of the dictionary $|\mathcal{D}|$ and then a uniform encoding of the set of $|\mathcal{D}|$ indices: $\mathrm{L}(\mathcal{D}) = \log|\mathfrak{U}| + \log\binom{|\mathfrak{U}|}{|\mathcal{D}|}$. (b) On the other hand, when $\mathfrak{U}$ is too large to enumerate or non-public, the adjacency matrices of the atoms can be stored one by one, as if independently sampled from the null-model given in (5.4):

$$\mathrm{L}(\mathcal{D}) = \sum_{\alpha_i \in \mathcal{D}} \mathrm{L}^{\text{null}}(\alpha_i). \tag{5.18}$$

The latter is the method that we adopt in our implementation since it relies on fewer assumptions. Finally, the parameters of the distribution encoder $\varphi = \boldsymbol{\varphi}$ need to be transmitted to the decoder as well, but their description length is not variable (see the discussion above), thus it will not be included in the optimisation objective. In practice, we use a fixed length 16-bit encoding for each of the learnable parameters.

Putting everything together, in order to encode a graph dataset $\mathfrak{D}$ sampled i.i.d. from $\mathfrak{G}$ we minimise the total description length

$$\min_{\boldsymbol{\theta},\mathcal{D},\boldsymbol{\varphi}} \sum_{G \in \mathfrak{D}} \mathrm{L}\left(\mathsf{PART}\big(G; (\boldsymbol{\theta}, \mathcal{D})\big); \boldsymbol{\varphi}\right) + \mathrm{L}(\mathcal{D}) \tag{5.19}$$

5.5 Theoretical Analysis: the compression gains of PnC

The following section performs a comparative analysis of the description length growth rate of various graph compressors. We theoretically compare PnC against two strong baselines whose description length can be derived in closed form: (a) Partitioning-only encoding for unlabelled graphs with code length L^{part}. Here, a graph is decomposed into subgraphs and cuts,

but no dictionary coding is performed and the distribution of subgraphs and cuts is modelled with a null model, i.e. $\mathrm{L}^{\mathrm{part}}(G) = \mathrm{L}^{\mathrm{null}}(\mathcal{H}) + \mathrm{L}^{\mathrm{null}}(C|\mathcal{H})$ (see Eq. (C.7) in the Appendix for the exact formula). (b) Encodings that do not rely on partitioning and a decomposition of the graph distribution but model the distribution with a null model directly on the space of graphs. Our results hold for both the baselines that use a null model on labelled, but also, most importantly, on unlabelled graphs, such as the Erdős-Renyi model for unlabelled graphs of n vertices: $\mathrm{L}^{\text{ER-S}}(G \mid n, m) = \log |\mathfrak{G}_{n,m}/\simeq|$ where $\mathfrak{G}_{n,m}/\simeq$ is the set of all unlabelled graphs with n vertices and m edges. $\mathrm{L}^{\text{ER-S}}(G)$ serves as a lower bound to typical encodings such as that of Eq. (5.4), but can be impractical to implement due to the complexity of GI. The analysis of additional baselines, the exact assumptions and formulas for the compared encodings, as well as all proofs, can be found in the Appendices C.1.2-C.1.4.

Our main theorem shows that, under mild conditions on the underlying graph distribution, the expected description lengths of the compared encodings are totally ordered:

Theorem 5.2. *Consider a distribution p over labelled graphs with n vertices and a partitioning algorithm that decomposes a graph into b blocks of $k = O(1)$ vertices. Then asymptotically with n the following hold for an optimal PnC compressor:*

$$\mathbb{E}_{G\sim p}[\mathrm{L}^{\mathrm{PnC}}(G)] \overset{(1b)}{\lesssim} \mathbb{E}_{G\sim p}[\mathrm{L}^{\mathrm{part}}(G)] \overset{(1a)}{\lesssim} \mathbb{E}_{G\sim p}[\mathrm{L}^{\text{ER-S}}(G)] \tag{5.20}$$

under the following conditions:

(1a) $\bar{\mathrm{H}}_m - \frac{\log(k^2+1)}{k^2}) - \bar{\mathrm{H}}_{m_{ij}} > 0$, *where* $\bar{\mathrm{H}}_{m_{ij}} := \frac{1}{b^2-b}\sum_{i\neq j}^{b} \mathbb{E}_{G\sim p}[\mathrm{H}(\frac{m_{ij}}{k^2})]$ *and* $\bar{\mathrm{H}}_m := \mathbb{E}_{G\sim p}[\mathrm{H}(\frac{m}{n^2})]$ *are the expected binary entropy of the cut size m_{ij} between subgraphs i and j (averaged over all subgraph pairs) and that of the total number of edges m, respectively.*

(1b) $1 - \delta_{\mathrm{true}} > 0$ *and* $\log|\mathcal{D}| < \log(k^2+1) + k^2 \frac{\bar{\mathrm{H}}_{m_i} - \delta_{\mathrm{true}}\bar{\mathrm{H}}^{\mathrm{null}}_{m_i}}{1-\delta_{\mathrm{true}}}$, *where* $\bar{\mathrm{H}}_{m_i} := \frac{1}{b}\sum_{i=1}^{b} \mathbb{E}_{G\sim p}[\mathrm{H}(\frac{m_i}{k^2})]$ *is the expected binary entropy of the subgraph edges m_i averaged over all subgraphs,* $\bar{\mathrm{H}}^{\mathrm{null}}_{m_i}$ *is the analogue when averaging over non-dictionary subgraphs,* $\delta_{\mathrm{true}} := \frac{\mathbb{E}_{G\sim p}[b_{\mathrm{null}}]}{b}$ *is the expected percentage of non-dictionary subgraphs and $|\mathcal{D}|$ is the size of the dictionary.*

The compression gains are:

$$\mathbb{E}_{G\sim p}[\mathrm{L}^{\mathrm{Part}}(G)] \lesssim \mathbb{E}_{G\sim p}[\mathrm{L}^{\text{ER-S}}(G)] - \Theta(n^2) \text{ and } \mathbb{E}_{G\sim p}[\mathrm{L}^{\mathrm{PnC}}(G)] \lesssim \mathbb{E}_{G\sim p}[\mathrm{L}^{\mathrm{part}}(G)] - \Theta(n) \tag{5.21}$$

Theorem 5.2 provides insights into the compressibility of certain graph distributions given their structural characteristics. In particular, we can make the following remarks:

Condition (1a) can be satisfied even for very small values of k as long as the graphs possess community structure. Perhaps counter-intuitively, when $k = O(1)$ we can satisfy the condition even if the communities have $O(n)$ size by splitting them into smaller subgraphs. This is possible because, in contrast to the majority of graph partitioning objectives that are based on minimum cuts, the compression objective attains its minimum when the cuts have "low entropy". Since communities that are tightly internally connected have large cuts, $\bar{\mathrm{H}}_{m_{ij}}$ and the code length will be kept small. This is a key observation that strongly motivates the use of partitioning for graph compression.

Condition (1b) provides an upper bound to the size of the dictionary, which can be easily satisfied for moderately small values of k as long as δ_{true} is not close to 1, i.e. we don't fall back to a partitioning-based compressor. Observe that the smaller the value of δ_{true}, the larger the upper bound for the size of the dictionary. More importantly, the dependence of the compression gain on the entropy $\mathbb{H}(\mathcal{D})$, reveals that dictionary atoms should be frequent subgraphs in the distribution, confirming our intuition. The bounds also show that, since the probabilities of the atoms are estimated from the data, PnC does not need to make assumptions about the inner structure of the subgraphs and can adapt to general distributions.

We lastly provide theoretical evidence on the importance of encoding dictionary subgraphs as isomorphism classes instead of adjacency matrices, which is related to challenge *C1*. Theorem 5.3 shows that, if isomorphism is not taken into account, the extra number of bits that we will incur will grow linearly with the number of vertices. Formally, (proof in Appendix C.1.5):

Theorem 5.3. *Let p be a graph distribution on labelled graphs of n vertices that is invariant to isomorphism, i.e., $p(G') = p(G)$ if $G \simeq G$. Moreover, consider any algorithm that partitions G in b subgraphs of k vertices. Denote by $\mathrm{L}^{\text{PnC-G}}$ and $\mathrm{L}^{\text{PnC-S}}$ the description length of an optimal PnC compressor that maps labelled subgraphs to dictionary atoms by comparing their adjacency matrices, and that of an optimal PnC compressor that maps unlabelled subgraphs to dictionary atoms via graph isomorphism testing, respectively. The following holds:*

$$\mathbb{E}_{G\sim p}[\mathrm{L}^{\text{PnC-S}}(G)] \approx \mathbb{E}_{G\sim p}[\mathrm{L}^{\text{PnC-G}}(G)] - n(1-\delta_{true})\log k, \tag{5.22}$$

under the condition that almost all graphs in the dictionary are rigid [4] *and* $\delta_{\text{true}} := \frac{\mathbb{E}_{G\sim p}[b_{\text{null}}]}{b}$.

Importantly, the compression gains implied by the theorem hold independently of the size of the dictionary, applying e.g. also when the dictionary is equal to the universe and contains all graphs of size k (which amounts to the traditional partitioning baselines). We refer the reader to section 5.8, where both key theoretical findings are reflected in the experimental results.

5.6 Optimisation and learning algorithms

We turn our focus to learning algorithms for the optimisation of the MDL objective (5.19). The following sections explain how each parametric component of PnC is learned.

Distribution encoder parameters $\boldsymbol{\varphi}$. The distribution encoder is parametrised as follows: $q(\alpha_i)$ and $q(b)$ are parametrised by real-valued learnable variables that are converted into categorical distributions over the dictionary atoms and the number of vertices respectively, using a softmax function. Similarly, δ is parametrised by a real-valued learnable variable converted to a probability via the sigmoid function.

Dictionary $\mathcal{D}$. Let $\mathfrak{U} = \{\alpha_1, \alpha_2, \dots \alpha_{|\mathfrak{U}|}\}$ be a practically enumerable universe and define $\boldsymbol{x} = (x_1, x_2, \dots, x_{|\mathfrak{U}|})$ as

$$x_i = \begin{cases} 1 & \text{if } \alpha_i \in \mathcal{D} \\ 0 & \text{otherwise.} \end{cases}$$

Thus, x_i indicates whether $\mathcal{D}$ contains subgraph α_i. Now, optimising for the dictionary is equivalent to finding the binary assignments for $\boldsymbol{x}$ that minimise (5.19). To circumvent the combinatorial nature of this problem, we apply the continuous relaxation $\hat{x}_i = \sigma(\xi_i), \forall i \in \{0, 1, \dots, |\mathfrak{U}|\}$, where σ is the sigmoid function, ξ_i are learned continuous variables, and $\hat{x}_i \in [0, 1]$ a fractional alternative to x_i. Appendix C.2.2 shows how (5.19) can be re-written using $\boldsymbol{x}$ and optimised by using the surrogate gradient w.r.t. $\hat{\boldsymbol{x}}$.

It is important to note that, since most subgraphs α_i will be never encountered in the graph distribution, we build the universe adaptively during training, by progressively adding the different graphs that the partitioning algorithm yields. Our universe contains subgraphs of size up to $k = O(1)$, in order to ensure that the isomorphism testing can be efficiently computed.

[4] A rigid graph has only the trivial automorphism.

Parametric graph partitioning algorithm. Finding the graph partitions that minimise (5.19) in principle requires searching in the space of partitioning algorithms. Instead, we constrain this space via a differentiable parametrisation that allows us to perform gradient-based optimisation. Currently, learning to partition is an open problem, as to the extent of our knowledge known neural approaches do not guarantee that the subgraphs are connected [Karalias and Loukas, 2020] or/and require a fixed number of clusters [Wilder et al., 2019, Nazi et al., 2019, Bianchi et al., 2020].

Our *Neural Partitioning* is a randomised algorithm parametrised with a GNN. When applied to a graph, the GNN outputs a random $(\mathcal{H}, C)$ together with a corresponding probability $p^{\text{GNN}}(\mathcal{H}, C \mid G; \boldsymbol{\theta})$ and training is performed by estimating the gradients w.t.t. $\boldsymbol{\theta}$ with REINFORCE [Williams, 1992]. Our algorithm proceeds by iteratively sampling (and removing) subgraphs from the graph until it becomes empty. At each step t we select a subgraph H_t, by first sampling its vertex count k_t, and subsequently sampling at most k_t vertices. To guarantee connectivity, we also sample the vertices iteratively and mask out the probabilities outside the pre-selected vertices' neighbourhoods. The complexity of the algorithm is $O(n)$, where n the number of the vertices of the graph. Please refer to Appendix C.2.3 for an in-depth explanation of the algorithm and relevant implementation details. We stress that we mainly consider this algorithm as a proof of concept that we ablate against other non-parametric partitioning algorithms. A plethora of solutions can be explored in a parametric setting and we anticipate further work in this direction in the future.

Final MDL objective. Given a dataset $\mathfrak{D}$, we train all components by minimising the description length:

$$\mathrm{L}(\mathfrak{D}) = \sum_{G \in \mathfrak{D}} \mathbb{E}_{(\mathcal{H},C) \sim p^{\text{GNN}}(\mathcal{H},C \mid G;\boldsymbol{\theta})} [\mathrm{L}\left((\mathcal{H}, C); \boldsymbol{x}, \boldsymbol{\varphi}\right)] + \mathrm{L}(\mathcal{D}; \boldsymbol{x}). \tag{5.23}$$

Taking the expectation over the GNN output we calculate the gradients as follows:

$$\begin{aligned}
\nabla_{\boldsymbol{\varphi}} L(\mathfrak{D}) &= \sum_{G \in \mathfrak{D}} \mathbb{E}\left[\nabla_{\boldsymbol{\varphi}} \mathrm{L}\left((\mathcal{H}, C); \boldsymbol{x}, \boldsymbol{\varphi}\right)\right] \\
\nabla_{\hat{\boldsymbol{x}}} L(\mathfrak{D}) &= \sum_{G \in \mathfrak{D}} \mathbb{E}\left[\nabla_{\hat{\boldsymbol{x}}} \mathrm{L}\left((\mathcal{H}, C); \hat{\boldsymbol{x}}, \boldsymbol{\varphi}\right)\right] + \nabla_{\hat{\boldsymbol{x}}} \mathrm{L}(D; \hat{\boldsymbol{x}}), \\
\nabla_{\boldsymbol{\theta}} L(\mathfrak{D}) &= \sum_{G \in \mathfrak{D}} \mathbb{E}\left[\mathrm{L}\left((\mathcal{H}, C); \boldsymbol{x}, \boldsymbol{\varphi}\right) \nabla_{\boldsymbol{\theta}} \ln p^{\text{GNN}}(\mathcal{H}, C \mid G; \boldsymbol{\theta})\right]
\end{aligned}$$

5.7 Related Work

Engineered codecs. The majority of graph compressors are not explicitly designed as probabilistic models, but rely on handcrafted encodings optimised to take advantage of domain-specific properties of e.g., WebGraphs [Vigna and Boldi, 2004], social networks [Dhulipala et al., 2016, Apostolico and Drovandi, 2009, Boldi et al., 2011], and biological networks [Peshkin, 2007]. A common idea in these approaches is *vertex re-ordering* [Vigna and Boldi, 2004, Lim et al., 2014, Dhulipala et al., 2016, Apostolico and Drovandi, 2009, Boldi et al., 2011, Chierichetti et al., 2009, Claude and Navarro, 2010, Rossi and Zhou, 2018], where the adjacency matrix is permuted in such a way that makes it "compression-friendly" for mainstream compressors of sequences, such as gzip. The algorithms identifying the re-orderings are usually based on heuristics taking advantage of specific network properties, e.g., community structure. Another recurrent idea is to detect or use predefined *frequent substructures* (e.g., cliques) to represent more efficiently different parts of the graph via grammar rules [Maneth and Peternek, 2016]. Most of these approaches implicitly make rigid assumptions about the underlying graph distribution and are not adaptive, but strive to find a balance between compression ratios and fast operations on the compressed graphs. Thus, despite their practical importance, they are less relevant to our work. A comprehensive survey can be found in [Besta and Hoefler, 2018].

Theory-driven approaches. Several works have contributed to the foundations of the information content and the complexity of graphs [Naor, 1990, Choi and Szpankowski, 2012, Trucco, 1956a, Trucco, 1956b, Körner, 1973, Turán, 1984, Dehmer, 2008, Dehmer et al., 2009, Mowshowitz and Dehmer, 2012]. However, few works have attempted to model the underlying graph distribution. Perhaps the most outstanding progress has been made for graphs modelled by the Stochastic Block Model (SBM) [Holland et al., 1983, Rosvall and Bergstrom, 2008, Karrer and Newman, 2011, Peixoto, 2012, Peixoto, 2013, Peixoto, 2017, Peixoto, 2019], where community structure is prevalent. Although originally invented for clustering and network analysis purposes, these approaches can be seamlessly used for compression since they are random graph models with explicit likelihood computation. In fact, as we argue in this work, virtually any graph clustering algorithm can be used successfully for compression, by defining codewords corresponding to a community-based random graph model. However, as our experiments confirm, such approaches are less effective at compressing graphs that do not contain clusters.

Likelihood-based neural approaches. Any generative model that can provide likelihood

estimates in a finite sample space can be used for lossless compression. As a result, a plethora of likelihood-based neural compressors have been recently invented, ranging from autoregressive models for text [Schmidhuber and Heil, 1996, Mahoney, 2000, Cox, 2016, Goyal et al., 2019] and images [van den Oord et al., 2016, Mentzer et al., 2019] to latent variable models [Townsend et al., 2019, Kingma et al., 2019, Townsend et al., 2020, Ruan et al., 2021, Severo et al., 2022] (paired with bits-back coding [Wallace, 1990, Hinton and Van Camp, 1993] and Asymmetric Numeral Systems - ANS [Duda, 2013]), normalising flows [Ho et al., 2019, Hoogeboom et al., 2019, Tran et al., 2019, van den Berg et al., 2021] and most recently, diffusion-based generative models [Kingma et al., 2021, Hoogeboom et al., 2022]. However, the vast majority of current graph generators lack the necessary theoretical properties an effective graph compressor should have: they compute probabilities on labelled graphs instead of isomorphism classes by resorting to a heuristic ordering [Jin et al., 2018b, Li et al., 2018, You et al., 2018, Liu et al., 2018, Liao et al., 2019, Dai et al., 2020, Shi et al., 2020, Luo et al., 2021] (in general this will be suboptimal unless we canonicalise the graph/solve graph isomorphism, while different orderings will have non-zero probabilities, hence we will incur compression losses), and/or do not provide a likelihood [Cao and Kipf, 2018, Yang et al., 2019, Bojchevski et al., 2018, Niu et al., 2020, Liu et al., 2021]. Recently, [Dubois et al., 2021] developed a relevant framework for invariant compression using rate-distortion theory, but its instantiation and optimisation cannot guarantee that non-isomorphic graphs will not be mapped to the same codeword.

Generative models are usually parameter inefficient while compressing them (especially during training) is a challenging problem [Bird et al., 2021]. Note that this is significantly different than compressing neural classifiers since the capacity to infer the likelihood up to high precision needs to be retained. Therefore, these approaches can have diminishing (or even negative) returns when the dataset size is not large enough. In contrast, by optimising the total description length, we design a compressor that is practical even for small datasets, while the learned dictionary makes our compressor interpretable.

Other related work includes using compression objectives paired with heuristic algorithms for downstream tasks, such as motif finding [Ketkar et al., 2005, Bloem and de Rooij, 2020] and graph summarisation [Koutra et al., 2014, LeFevre and Terzi, 2010], lossy compression/coarsening [Nourbakhsh et al., 2015, Loukas and Vandergheynst, 2018, Loukas, 2019, Garg and Jaakkola, 2019, Dong and Sawin, 2020, Jin et al., 2020], and graph dictionary learning in the context of sparse coding [Zhang et al., 2012, Vincent-Cuaz et al., 2021].

5.8 Results

We evaluate our framework in a variety of datasets: small molecules, proteins and social networks [Yanardag and Vishwanathan, 2015, Irwin et al., 2012b, Morris et al., 2020a, Borgwardt et al., 2005, Helma et al., 2001]. Across all methods, we assume an optimal entropy coder that attains the entropy lower bound (this is often realistic since modern entropy coders asymptotically approach it) to evaluate the expressive power of each model independently of the coder. We measure the description length of the data as their negative log-likelihood (NLL) under each probabilistic model, as well as the total description length by adding the cost of the parameters which need to be transmitted to the decoder (see Appendix C.4 for details).

Baselines. We aim to assess representative approaches across the entire spectrum of graph probabilistic models, i.e., from generic uninformative non-parametric distributions to overparameterised neural compressors. We consider the following types of compressors: (a) *Null models.* We select the *uniform*, the *edge list* model and the *Erdős-Renyi* model (see section 5.2). (b) *Partitioning-based:* Non-parametric methods that aim at grouping vertices in tightly-connected clusters. They can be used for any type of sparse matrix [Chakrabarti et al., 2004] and are based on the assumption that there exists a hidden community structure in the graph. The partioning algorithms used are *SBM fitting* [Peixoto, 2012, Peixoto, 2013, Peixoto, 2017, Peixoto, 2019], the *Louvain* algorithm [Blondel et al., 2008] and *Label Propagation* clustering[Raghavan et al., 2007]. The encoding we use to encode the clusters corresponds exactly to the SBM assumptions, hence the partitioning-based results are always superior for this approach. (c) *Likelihood-based neural compressors.* As with any likelihood-based model, graph generative models can be transformed into graph compressors. We evaluate the original GraphRNN [You et al., 2018] and GRAN [Liao et al., 2019] networks, as well as smaller instantiations that have undergone model compression using the Lottery Ticket Hypothesis algorithm [Frankle and Carbin, 2019].

Results. Tables 5.1 and 5.2 report the compression quality of each method measured in terms of the average number of bits required to store each edge in a dataset (bpe). We present four variants of PnC, differing on the type of partitioning algorithm used [Raghavan et al., 2007, Blondel et al., 2008, Peixoto, 2019]. We report separately the cost of compressing the data as well as the total cost (including the parameters). Several observations can be made with regard to the baselines:

First off, *off-the-shelf likelihood-based neural approaches are poor compressors due to failing to*

Table 5.1: Average bits per edge (bpe) for molecular datasets. First, Second, Third

Method type	Graph type	Small Molecules								
	Dataset name	MUTAG			PTC			ZINC		
		data	total	params	data	total	params	data	total	params
Null	Uniform (raw adjac.)	-	8.44	-	-	17.43	-	-	10.90	-
	Edge list	-	7.97	-	-	9.38	-	-	8.60	-
	Erdős-Renyi	-	4.78	-	-	5.67	-	-	5.15	-
Partitioning (non-parametric)	SBM-Bayes	-	4.62	-	-	5.12	-	-	4.75	-
	Louvain	-	4.80	-	-	5.27	-	-	4.77	-
	PropClust	-	4.92	-	-	5.40	-	-	4.85	-
Neural (likelihood)	GraphRNN	1.33	3338.21	388K	1.57	1394.59	389K	1.62	43,16	388K
	GRAN	0.81	12557.75	1460K	2.18	5269.82	1470K	1.30	157.7	1461K
	GraphRNN (pruned)	1.95	12.39	1.08K	2.16	6.71	1.10K	1.79	**2.02**	1.90K
	GRAN (pruned)	2.59	24.56	2.23K	4.31	14.00	2.36K	3.26	3.47	1.69K
PnC	PnC + SBM	3.81	4.11	49	4.38	4.79	155	3.34	3.45	594
	PnC + Louvain	2.20	**2.51**	47	2.68	**3.14**	166	1.96	**1.99**	196
	PnC + PropClust	2.42	**3.03**	63	3.38	**4.02**	178	2.20	2.35	726
	PnC + Neural Part.	2.17±0.02	**2.45±0.02**	46±1	2.63±0.26	**2.97±0.14**	143±31	2.01±0.02	**2.07±0.03**	384±105

Table 5.2: Average bits per edge (bpe) for social and protein datasets. First, Second, Third

Method type	Graph type	Biology			Social Networks					
	Dataset name	PROTEINS			IMDB-B			IMDB-M		
		data	total	params	data	total	params	data	total	params
Null	Uniform (raw adjac.)	-	24.71	-	-	2.52	-	-	1.83	-
	Edge list	-	10.92		-	8.29	-	-	7.74	-
	Erdős-Renyi	-	5.46	-	-	1.94	-	-	1.32	-
Partitioning (non-parametric)	SBM-Bayes	-	3.98	-	-	**0.80**	-	-	**0.60**	-
	Louvain	-	3.95	-	-	1.22	-	-	0.88	-
	PropClust	-	4.11	-	-	1.99	-	-	1.37	-
Neural (likelihood)	GraphRNN	2.03	156.99	392K	1.03	132.27	395K	0.72	127.84	392K
	GRAN	1.51	607.96	1545K	0.26	488.88	1473K	0.17	475.13	1467K
	GraphRNN (pruned)	2.63	3.76	2.56K	1.43	1.92	1.28K	0.91	1.39	1.28k
	GRAN (pruned)	4.28	5.11	1.78K	0.84	1.75	2.38K	0.55	1.41	2.31K
PnC	PnC + SBM	3.26	**3.60**	896	0.50	**0.54**	198	0.38	**0.38**	157
	PnC + Louvain	3.34	**3.58**	854	0.96	**1.02**	202	0.66	**0.70**	141
	PnC + PropClust	3.42	3.68	866	1.45	1.64	241	0.93	1.04	178
	PnC + Neural Part.	3.34±0.25	**3.51±0.23**	717±61	1.00±0.04	1.05±0.04	186±25	0.66±0.05	0.72±0.05	178±14

address challenge C2. These models exhibit an unfavourable trade-off between the data and model complexity, often requiring significantly more bpe than the null models. Although model compression techniques can alleviate this tradeoff (especially for larger datasets, e.g., pruned GraphRNN on ZINC), in most of the cases the compression ratios required to outperform PnC are significantly higher than the best that have been reported in the literature (See Table C.4 in the Appendix). Perhaps more importantly, it is unclear how to optimise the model description length during training (one of the few exceptions is [Havasi et al., 2019]) and usually model compression might be tedious and is based on heuristics [Han et al., 2016, Louizos et al., 2017, Han et al., 2015, Hinton et al., 2015]. See Appendix C.3.2 for more details and additional experiments.

In addition, as expected from Theorem 5.2.a, *non-parametric clustering algorithms work well*

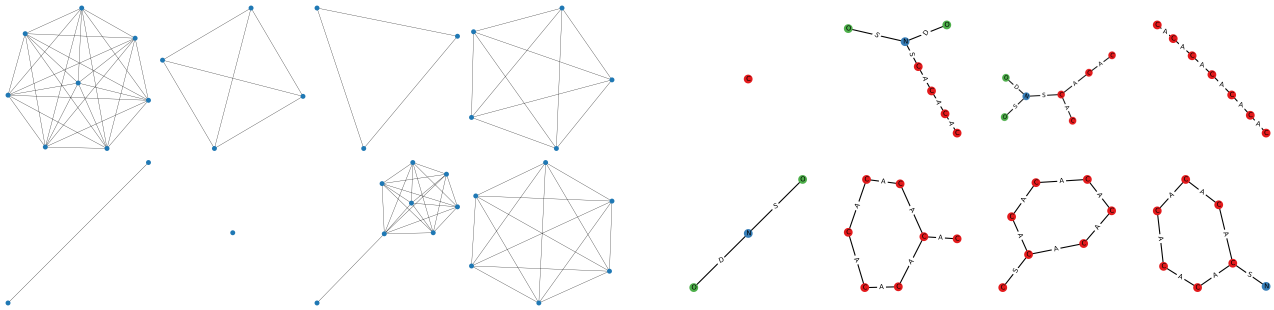


Figure 5.2: PnC + Neural Part. - Most probable graphs in the IMDB-B dataset (left) and the attributed MUTAG dataset (right).

when the dataset has a strong community structure, but are not a good choice for datasets with repetitive substructures. For instance, the best clustering algorithm requires 2.4× more bpe for ZINC than the best PnC. *PnC variants achieve the best compression in all datasets considered.* This verifies Theorem 5.2.b, since the learned dictionaries are relatively small, and confirms our hypothesis that our framework is sufficiently flexible to account for the particularities of each dataset. As seen, neural partitioning performs in every case better, or on par with the combination of PnC with the Louvain algorithm. However, in the social network datasets, the combination of PnC with SBM achieves the best performance. This occurs because these networks fit nicely with the SBM inductive bias (which as a matter of fact is exactly that of low-entropy cuts), and most importantly, due to the fact that the clusters recovered by the SBM are small and repetitive, which makes them ideal for the PnC framework. We also observe that there is room for improvement for the neural partitioning variant, and hypothesise that a more powerful parametrised algorithm can be designed. Since learnable partitioning is still an open problem, we leave this research direction to future work.

Fig. 5.2 shows the most likely dictionary atoms for the IMDB-B and the MUTAG dataset (also including attributes - Appendix C.3.3 provides additional experiments). Observe that cliques or near-cliques and typical molecular substructures, such as carbon cycles and junctions are recovered for social networks and molecules respectively. This clearly highlights the connection between compression and pattern mining and provides evidence for potential applications of our framework.

6 Discussion

In the following and final chapter of the thesis, we will summarise our contributions to the fields of graph machine learning and geometric deep learning and we will pinpoint the crux of the problems we encountered and the main axes that have driven this research. Additionally, we will review the impact that our research has had both in terms of applications and in terms of the related ideas that emerged after our work. Finally, we will discuss potential ideas that we anticipate to push forward our understanding of the problems considered.

6.1 Main contributions & takeaways

Summary of main results. The unifying element that bridges our contributions is the problem of approximating functions on graphs. In its most general form, addressing it amounts to providing an answer to the following question: Consider a target function $f^* : \mathscr{X} \to \mathscr{Y}$, where at least one of $\mathscr{X}, \mathscr{Y}$ is a set of graphs attributed with vertex and/or edge signals, i.e. $\mathfrak{G} \subseteq \{(\mathcal{V}, \mathcal{E}, \mathbf{u}_{\mathcal{V}}, \mathbf{u}_{\mathcal{E}}) \mid \mathcal{V} \subset \mathbb{N}, \mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}, \mathbf{u}_{\mathcal{V}} \in \mathbb{R}^{\mathcal{V} \times d_v}, \mathbf{u}_{\mathcal{E}} \in \mathbb{R}^{\mathcal{E} \times d_e}\}$ and a distribution p on $\mathscr{X}$. *How can we design a function h that approximates f^* up to a desired precision with high probability, using only a finite sample of data?*. In section 1.3 we provided formal specifications of this concept and identified different problems that can be encapsulated therein, while in section 1.4 we presented the common principles that the designer should follow to succeed in the desired task.

In chapter 3 we analysed the first popular instance of the problem, the case of a *fixed graph topology* $(\mathcal{V}, \mathcal{E})$, which finds applications in fields ranging from network science (network analysis and network dynamic signals) to biomedical imaging, computer graphics and 3D computer

vision (shape modelling), where the latter is the one we focus on. We saw that treating the problem with many established graph function approximation methodologies is limiting in terms of the functions that can be expressed, while on the other end, treating it as a function approximation problem on real coordinate spaces comes with an important cost in computational complexity and with weaker inductive biases. To address the above, we drew inspiration from shift-equivariance and convolutions on grids and proposed a *permutation-sensitive graph neural network (GNN)* that overcomes the expressivity issues of vanilla permutation-invariant GNNs (isotropy), while being computationally efficient and maintaining the desirable inductive bias of locality.

In the following chapters 4 and 5, we moved to the general problem formulation, where the datapoints are of variable graph topology, with the applications spanning multiple scientific fields, including chemistry, biology and social sciences, and considered analysis problems, i.e. when the input space is a space of graphs and the output space is typically a subset of a real coordinate space. In chapter 4 we analysed the problem of approximating *GI-invariant functions*, i.e. functions that always evaluate to the same output for every pair of isomorphic graphs, and saw that this constraint unless allowed to be violated, imposes a fundamental tradeoff between expressivity and computational complexity. To address this, we proposed to harness the power of *graph substructures* in the form of symmetry-breaking weighting functions incorporated into vanilla message-passing GNNs and benefited from provable improvements in expressivity, controllable increase in computational complexity, as well as a strong inductive bias that empirically improved generalisation.

Finally, in chapter 5, we analysed the problems of graph compression, an important, yet under-explored topic in information theory, and saw that it is a particular case of an approximation problem of *GI-injective functions*, i.e. functions that always evaluate to different outputs for every pair of non-isomorphic graphs. In addition, we showed that when compressing unlabelled graphs, optimality imposes an additional constraint, that of GI-invariance, which revealed another tradeoff, now between compression quality and computational complexity. To address this, first, we reformulated the problem as regularised distribution estimation and then we proposed an efficient (w.r.t. both time complexity and model size) estimator based on *dictionary coding* with theoretically provable and empirical compression gains.

Conclusions. The following collective conclusions can be derived from the research questions we investigated.

- The problem of function approximation on graphs is diverse and different instances thereof will lead to fundamentally different constraints for the target function, as well as the designed hypothesis. Although the constraints of certain instances of the problem entail inevitable trade-offs (see below), others frequently lead to great simplifications and give us more room to approach optimality in an efficient way. Except for signals on a fixed graph, other notable examples for which this is true, are the space of asymmetric graphs, which we know can be efficiently canonicalised, or molecular graphs, which are frequently equipped with a canonical ordering (canonical SMILES [Weininger et al., 1989]).

- Graph isomorphism has a multifaceted influence on general graph function approximation problems. Whenever our target function is not known to belong to any special instance, isomorphic graphs will be generally considered indiscernible. Although at first glance, graph discrimination seemed to be of mainly theoretical interest, it turned out to reveal fundamental limits in universal graph function approximation, but also in optimally computing graph properties that are known to be of practical relevance in real-world tasks.

- Due to the above, GI-invariance, computational complexity and expressivity are tightly linked. Obtaining improvements in one of the three dimensions typically requires trading one of the other two, and to date, it is unknown what is the optimal trade-off (the *Pareto front*) and how to attain it. This will require answering deep questions in graph theory, such as "Can we completely characterise the set of graphs that can be distinguished in linear time"?

- Nevertheless, the above discussion excludes two important desiderata of function approximation. Recall the definition: the target function needs to be *approximated up to a desired precision* and *with high probability*. In other words, it is acceptable for our approximator to make small mistakes on data that are rarely encountered. Considering that the fundamental trade-offs above arise from graphs that, under reasonable assumptions, will have low probability [Babai et al., 1980], we should not be disheartened.

- The misalignment between our true desiderata and the concepts that are currently deeply investigated, calls for a (at least partial) paradigm shift. In particular, the fact that the

vast majority of the graphs in the datasets we experiment with, are WL-distinguishable, implies that, in this context, graph discrimination can be achieved in linear time. It is, therefore, necessary to examine the behaviour of our function approximators in these regimes and w.r.t. our desiderata, i.e. generalisation. For example, under the condition of a particular distribution of graphs (that can be discriminated in linear time, e.g. asymmetric graphs), what are the fundamental limits of computing or approximating certain graph properties, and how do they compare with the unconditional case? How can we benefit from learning, i.e. will these limits be also reflected in sample complexity?

- A parallel line can be drawn between graph classification/regression and graph compression/graph generative models. We saw that in practical problems where GI-invariance is necessary (the former case), linear time algorithms can achieve graph discrimination with high probability. Similarly, in practical problems where GI-injectivity is necessary (the latter case), linear time algorithms can achieve GI-invariance [Babai and Kucera, 1979] with high probability. Therefore, similar questions to the ones above can be asked here, where now we are interested in the generalisation of such algorithms in distribution estimation problems.

- Finally, in the works presented in this thesis, our choices are driven by expressivity considerations and the constraints of the target function, but also by task-dependent inductive biases (anisotropy in chapter 3, substructures in chapters 4, 5 and community structure in chapter 5). Although in terms of generalisation, these are supported by intuitive and empirical arguments, their success (or lack thereof, potentially in other cases), advocates their theoretical investigation, in line with the aforementioned paradigm shift.

6.2 Impact & the road ahead

Learning on graph signals & spiral convolutions Our findings discussed in chapter 3 have been corroborated by subsequent works that used variations of spiral convolutions for various shape modelling tasks, both for shape analysis and synthesis. Some indicative examples include *3D computer vision and graphics applications*, such as predicting shape correspondences and facial expression classification [Gong et al., 2019], hand pose estimation [Kulon et al., 2020], prediction of human-scene interaction [Hassan et al., 2021], body and cloth reconstruction from

single-images, [Jiang et al., 2020] and facial expression animation [Potamias et al., 2020]. Other examples in the field of *3D medical imaging* include, genetic syndrome diagnosis [O'Sullivan et al., 2022, Mahdi et al., 2022a], demographics prediction from facial structure [Mahdi et al., 2021a, Mahdi et al., 2021b, Mahdi et al., 2022a], ear cartilage prediction for reconstructive surgery [Sullivan et al., 2020] and analysis of brain morphology of Alzheimer's disease patients [Azcona et al., 2021]. A subsequent publication showed improved empirical performance using soft permutations [Gao et al., 2022], while in [Chrysos et al., 2020, Chrysos et al., 2021] we showed significant improvements using polynomial approximators to instantiate Eq. (3.2).

Regarding learning on signals supported on a fixed graph, we anticipate further research beyond shape modelling and more sophisticated weight-sharing mechanisms, building on the framework of permutation-sensitive GNNs. In particular, capitalising on the universal approximation results of [Loukas, 2020] and in absence of the necessity to preserve GI-invariance, the challenging question that remains in this topic is how to design unique vertex identifiers that are flexible and can generalise well. Several of the positional encoding methods, such as the ones discussed in section 4.7.1, might prove appropriate for this objective.

Finally, regarding meshes of arbitrary topology, we already saw in section 3.4.1 that important progress has already been achieved. At the same time, several methods have been recently proposed for geometric graphs, such as 3D conformations of molecules [Satorras et al., 2021, Schütt et al., 2021, Klicpera et al., 2020] and followed by relevant theoretical arguments [Joshi et al., 2022]. These results provide solutions to similar challenges as the ones that we encounter when learning on meshes, such as permutation invariance and invariance to euclidean transformations. However, discretisation invariance is a particularly challenging question that current research has not been able to rigorously answer so far. Considering that meshes are simple depictions (to be precise, lossy representations) of reality, it is natural to believe that discretisation invariance should be studied through the lens of the underlying continuous surface, which can also simplify synthesis tasks (e.g. by synthesising the parameters of a continuous function). The large body of literature on implicit surfaces that we saw in section 3.4.1, as well as the promising results of continuous neural PDE solvers/continuous simulators [Kovachki et al., 2021b] provides us with positive indications of this direction.

Learning on graph spaces & graph substructure networks As we discussed in section 4.7.1, following our work, the concept of subgraphs was studied in several papers, as a means of

improving the expressivity of GNNs (see paragraph "Subgraph GNNs"), establishing them as a popular research theme. In addition, observing the empirical performance of the architectures proposed in many of these works, subgraphs are verified to be an appropriate inductive bias for many real-world graph-level analysis tasks (e.g. for molecular property prediction substructure-oriented methods are among the best performing), while recently they have been also used to improve the performance of graph generative models [Vignac et al., 2022, Boget et al., 2022]. Moreover, our work posed important theoretical research questions (e.g. regarding substructure selection, comparisons with k-WL tests and exact expressivity quantification) that sparked the interest of fellow researchers in the community and were partially addressed by future works [Barceló et al., 2021, Papp and Wattenhofer, 2022]. Overall, GSN currently serves as a strong baseline and an important reference point in the field of GNN expressivity.

As we saw in section 4.2 substructure-related information is one of the many categories of graph properties that are known to be closely related to real-world tasks, while others (e.g. distance encodings, or community-related information, implicitly conveyed by spectral encodings) have also been incorporated into modern GNNs and demonstrated strong empirical generalisation. This naturally poses the question if such information can be discovered in a data-driven manner (as is customary in many ML application domains). Having established the inability of vanilla, and sometimes modern, GNNs to optimally compute these properties, we envision that future research endeavours will attempt to push our knowledge forward w.r.t. the ability to do distribution-dependent approximations. As we saw in the previous section, this entails first theoretically investigating the limitations of the current architectures, but in the context of practical operating regimes.

Additionally, by departing from the strict requirements of expressivity and graph discrimination, we envision new algorithmic approaches to emerge and possibly achieve better invariance-complexity-generalisation trade-offs. Some ideas include algorithms with potentially high worst-case complexity, but variable runtime (in line with most practical solvers of graph-related problems), or with variable GI-invariance, e.g. with guaranteed invariance for easy instances, such as asymmetric graphs, but with potentially GI-sensitive output for hard instances. In this endeavour, it will be once again helpful to draw inspiration from other sub-fields of theoretical computer science, such as distributed computing and combinatorial optimisation (in particular approximation and randomised algorithms), since many of the problems we encounter when designing GNNs are disguised versions of problems that have re-appeared in the past.

Graph compression & graph generative models. Furthermore, we anticipate further practical advancements in the topic of graph compression. Our work marked the first step in the topic of neural compression of (unlabelled) graphs and there are still deep questions that wait to be answered, such as how to achieve better trade-offs between GI-invariance and computational complexity. Additionally, in the process of developing this work, we uncovered two other fundamental problems: First, that of neural partitioning, which is a hard combinatorial optimisation problem, that currently has not received as much attention as others in the same category (TSP, SAT, maximum clique etc.).

Second, as we have extensively discussed in chapter 5 and section 5.3, we faced a universal problem in neural compression: that of minimising the description length of the parametric estimator. In our case, we addressed it by using an estimator with a variable description length (due to the dictionary), but at the expense of expressivity and in turn, potentially, optimality. Given the remarkable success of neural generative models as distribution estimators, in the future we expect more algorithms for model compression in the same spirit with [Havasi et al., 2019] so that neural networks can be assigned a variable and optimisable with gradient-based methods description length. Moreover, we expect similar attempts to be proposed for (neural) distribution estimators in single massive graphs, instead of sets of graphs as in our case, where scalability is a pertinent issue.

Finally, we hope that the understanding we obtained by investigating graph compression, w.r.t. random graph models can influence the research of graph generative models. In particular, initially in the literature, GI-sensitive generative models were proposed, with this being considered as a disadvantage, while, lately, methods for GI-invariant models were successfully implemented. However, to achieve the latter in a computationally efficient way, GNNs are used, and these inevitably come with the expressivity (and possibly generalisation) limits that we have seen. In our work, we argue that if the vertex ordering is irrelevant, as with most benchmarks we see in practice (e.g. molecule generation), using a distribution estimator on labelled graphs is going to be wasteful in compression terms, even if the estimator is GI-invariant, and instead a distribution on isomorphism classes should be estimated. This calls to possibly re-think GI-sensitive generative models, or to be more precise, models that GI-invariance holds with high probability, which is a trade-off reminiscent of the ones we have to make in the GNN expressivity/graph analysis literature. Given the above, we expect that the answers provided to the fundamental questions of the two topics may converge in the near future.

Bibliography

[Abboud et al., 2021] Abboud, R., Ceylan, İ. İ., Grohe, M., and Lukasiewicz, T. (2021). The surprising power of graph neural networks with random node initialization. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021, Virtual Event / Montreal, Canada, 19-27 August 2021*, pages 2112–2118. ijcai.org. 80, 102, 114

[Abu-El-Haija et al., 2018] Abu-El-Haija, S., Perozzi, B., Al-Rfou, R., and Alemi, A. A. (2018). Watch your step: Learning node embeddings via graph attention. *Advances in Neural Information Processing Systems (NeurIPS)*, 31:9198–9208. 28

[Achlioptas et al., 2018] Achlioptas, P., Diamanti, O., Mitliagkas, I., and Guibas, L. (2018). Learning representations and generative models for 3d point clouds. In *International Conference on Machine Learning (ICML)*, volume 80 of *Proceedings of Machine Learning Research*, pages 40–49. PMLR. 81

[Albert and Barabási, 2002] Albert, R. and Barabási, A.-L. (2002). Statistical mechanics of complex networks. *Reviews of modern physics*, 74(1):47. 23

[Allen-Zhu et al., 2019] Allen-Zhu, Z., Li, Y., and Liang, Y. (2019). Learning and generalization in overparameterized neural networks, going beyond two layers. *Advances in neural information processing systems*, 32. 138

[Allen-Zhu et al., 2019] Allen-Zhu, Z., Li, Y., and Song, Z. (2019). A convergence theory for deep learning via over-parameterization. In *International Conference on Machine Learning, (ICML)*, volume 97 of *Proceedings of Machine Learning Research*, pages 242–252. PMLR. 37, 138, 229

[Alon et al., 1997] Alon, N., Yuster, R., and Zwick, U. (1997). Finding and counting given length cycles. *Algorithmica*, 17(3):209–223. 220

[Amberg et al., 2007] Amberg, B., Romdhani, S., and Vetter, T. (2007). Optimal step nonrigid icp algorithms for surface registration. In *2007 IEEE conference on computer vision and pattern recognition*, pages 1–8. IEEE. 83

[Andrychowicz et al., 2016] Andrychowicz, M., Denil, M., Gomez, S., Hoffman, M. W., Pfau, D., Schaul, T., Shillingford, B., and De Freitas, N. (2016). Learning to learn by gradient descent by gradient descent. *Advances in Neural Information Processing Systems (NIPS)*, 29. 30

[Angluin, 1980] Angluin, D. (1980). Local and global properties in networks of processors. In *ACM Symposium on Theory of Computing (STOC)*, pages 82–93. ACM. 114

[Anguelov et al., 2005] Anguelov, D., Srinivasan, P., Koller, D., Thrun, S., Rodgers, J., and Davis, J. (2005). Scape: shape completion and animation of people. *ACM Transactions on Graphics (TOG)*, 24(3):408–416. 84

[Apostolico and Drovandi, 2009] Apostolico, A. and Drovandi, G. (2009). Graph compression by bfs. *Algorithms*, 2(3):1031–1044. 152

[Arjovsky et al., 2017] Arjovsky, M., Chintala, S., and Bottou, L. (2017). Wasserstein generative adversarial networks. In *International Conference on Machine Learning (ICML*, volume 70 of *Proceedings of Machine Learning Research*, pages 214–223. PMLR. 86

[Arora et al., 2019] Arora, S., Du, S., Hu, W., Li, Z., and Wang, R. (2019). Fine-grained analysis of optimization and generalization for overparameterized two-layer neural networks. In *International Conference on Machine Learning (ICML)*, Proceedings of Machine Learning Research, pages 322–332. PMLR. 138

[Arpit et al., 2017] Arpit, D., Jastrzębski, S., Ballas, N., Krueger, D., Bengio, E., Kanwal, M. S., Maharaj, T., Fischer, A., Courville, A., Bengio, Y., et al. (2017). A closer look at memorization in deep networks. In *International Conference on Machine Learning (ICML)*, volume 70 of *Proceedings of Machine Learning Research*, pages 233–242. PMLR. 37

[Arvind et al., 2019] Arvind, V., Fuhlbrück, F., Köbler, J., and Verbitsky, O. (2019). On weisfeiler-leman invariance: Subgraph counts and related graph properties. In *Fundamentals of Computation Theory (FCT)*, volume 11651 of *Lecture Notes in Computer Science*, pages 111–125. Springer. 58, 96, 106, 107, 116, 227

[Atwood and Towsley, 2016] Atwood, J. and Towsley, D. (2016). Diffusion-convolutional neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, volume 29, pages 1993–2001. 119

[Atzmon and Lipman, 2020] Atzmon, M. and Lipman, Y. (2020). Sal: Sign agnostic learning of shapes from raw data. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2565–2574. 82

[Avis and Fukuda, 1996] Avis, D. and Fukuda, K. (1996). Reverse search for enumeration. *Discrete applied mathematics*, 65(1-3):21–46. 110

[Azcona et al., 2021] Azcona, E. A., Besson, P., Wu, Y., Kurani, A. S., Bandt, S. K., Parrish, T. B., Katsaggelos, A. K., Initiative, A. D. N., et al. (2021). Analyzing brain morphology in alzheimer's disease using discriminative and generative spiral networks. *bioRxiv*. 161

[Azizian and Lelarge, 2021] Azizian, W. and Lelarge, M. (2021). Expressive power of invariant and equivariant graph neural networks. In *9th International Conference on Learning Representations (ICLR)*. OpenReview.net. 114

[Babai, 2016] Babai, L. (2016). Graph isomorphism in quasipolynomial time. In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, pages 684–697. 41, 55

[Babai et al., 1980] Babai, L., Erdos, P., and Selkow, S. M. (1980). Random graph isomorphism. *SIAM Journal on computing*, 9(3):628–635. 102, 159

[Babai and Kucera, 1979] Babai, L. and Kucera, L. (1979). Canonical labelling of graphs in linear average time. In *20th Annual Symposium on Foundations of Computer Science (sfcs 1979)*, pages 39–46. IEEE. 160

[Bachmann et al., 2020] Bachmann, G., Bécigneul, G., and Ganea, O. (2020). Constant curvature graph convolutional networks. In *International Conference on Machine Learning (ICML)*, volume 119 of *Proceedings of Machine Learning Research*, pages 486–496. PMLR. 28

[Balcilar et al., 2021] Balcilar, M., Héroux, P., Gaüzère, B., Vasseur, P., Adam, S., and Honeine, P. (2021). Breaking the limits of message passing graph neural networks. In *International Conference on Machine Learning, (ICML)*, volume 139 of *Proceedings of Machine Learning Research*, pages 599–608. PMLR. 120

[Barceló et al., 2021] Barceló, P., Geerts, F., Reutter, J., and Ryschkov, M. (2021). Graph neural networks with local graph parameters. *Advances in Neural Information Processing Systems (NeurIPS)*, 34:25280–25293. 108, 112, 115, 162

[Battaglia et al., 2016] Battaglia, P., Pascanu, R., Lai, M., Rezende, D. J., et al. (2016). Interaction networks for learning about objects, relations and physics. In *Advances in Neural Information Processing systems (NIPS)*, volume 29, pages 4502–4510. 60

[Battaglia et al., 2018] Battaglia, P. W., Hamrick, J. B., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V. F., Malinowski, M., Tacchetti, A., Raposo, D., Santoro, A., Faulkner, R., Gülçehre, Ç., Song, H. F., Ballard, A. J., Gilmer, J., Dahl, G. E., Vaswani, A., Allen, K. R., Nash, C., Langston, V., Dyer, C., Heess, N., Wierstra, D., Kohli, P., Botvinick, M. M., Vinyals, O., Li, Y., and Pascanu, R. (2018). Relational inductive biases, deep learning, and graph networks. *CoRR*, abs/1806.01261. 24

[Batziou et al., 2023] Batziou, E., Bouritsas, G., and Tsigonias-Dimitriadis, A. (2023). Iterative combinatorial auctions: an improved approach using reinforcement learning. under review. 48

[Beaini et al., 2021] Beaini, D., Passaro, S., Létourneau, V., Hamilton, W., Corso, G., and Liò, P. (2021). Directional graph networks. In *International Conference on Machine Learning (ICML)*, volume 83 of *Proceedings of Machine Learning Research*, pages 748–758. PMLR. 115, 120, 121, 122, 231

[Behboodi et al., 2022] Behboodi, A., Cesa, G., and Cohen, T. (2022). A pac-bayesian generalization bound for equivariant networks. *CoRR*, abs/2210.13150. 34

[Belkin et al., 2019] Belkin, M., Hsu, D., Ma, S., and Mandal, S. (2019). Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences*, 116(32):15849–15854. 42, 138

[Belkin and Niyogi, 2003] Belkin, M. and Niyogi, P. (2003). Laplacian eigenmaps for dimensionality reduction and data representation. *Neural computation*, 15(6):1373–1396. 28

[Bell et al., 1989] Bell, T., Witten, I. H., and Cleary, J. G. (1989). Modeling for text compression. *ACM Computing Surveys (CSUR)*, 21(4):557–591. 130

[Benson et al., 2016] Benson, A. R., Gleich, D. F., and Leskovec, J. (2016). Higher-order organization of complex networks. *Science*, 353(6295):163–166. 112

[Bentley et al., 1986] Bentley, J. L., Sleator, D. D., Tarjan, R. E., and Wei, V. K. (1986). A locally adaptive data compression scheme. *Communications of the ACM*, 29(4):320–330. 141

[Besl and McKay, 1992] Besl, P. J. and McKay, N. D. (1992). A method for registration of 3-d shapes. *IEEE Trans. Pattern Anal. Mach. Intell.*, 14(2):239–256. 83

[Bessadok et al., 2021] Bessadok, A., Mahjoub, M. A., and Rekik, I. (2021). Graph neural networks in network neuroscience. *CoRR*, abs/2106.03535. 95

[Besta and Hoefler, 2018] Besta, M. and Hoefler, T. (2018). Survey and taxonomy of lossless graph compression and space-efficient graph representations. *CoRR*, abs/1806.01799. 152

[Bevilacqua et al., 2022] Bevilacqua, B., Frasca, F., Lim, D., Srinivasan, B., Cai, C., Balamurugan, G., Bronstein, M. M., and Maron, H. (2022). Equivariant subgraph aggregation networks. In *10th International Conference on Learning Representations (ICLR)*. OpenReview.net. 115

[Bhattacharya et al., 2021] Bhattacharya, K., Hosseini, B., Kovachki, N. B., and Stuart, A. M. (2021). Model Reduction And Neural Networks For Parametric PDEs. *The SMAI Journal of computational mathematics*, 7:121–157. 40

[Bianchi et al., 2020] Bianchi, F. M., Grattarola, D., and Alippi, C. (2020). Spectral clustering with graph neural networks for graph pooling. In *International Conference on Machine Learning, (ICML)*, volume 119 of *Proceedings of Machine Learning Research*, pages 874–883. PMLR. 151

[Biewald, 2020] Biewald, L. (2020). Experiment tracking with weights and biases. Software available from wandb.com. 226, 259

[Bird et al., 2021] Bird, T., Kingma, F. H., and Barber, D. (2021). Reducing the computational cost of deep generative models with binary neural networks. In *9th International Conference on Learning Representations (ICLR)*. OpenReview.net. 153

[Birmelé et al., 2013] Birmelé, E., Ferreira, R. A., Grossi, R., Marino, A., Pisanti, N., Rizzi, R., and Sacomoto, G. (2013). Optimal listing of cycles and st-paths in undirected graphs. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 1884–1896. SIAM. 220

[Björklund et al., 2014] Björklund, A., Pagh, R., Williams, V. V., and Zwick, U. (2014). Listing triangles. In *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, volume 8572 of *Lecture Notes in Computer Science*, pages 223–234. Springer. 220

[Blanz et al., 1999] Blanz, V., Vetter, T., et al. (1999). A morphable model for the synthesis of 3d faces. In *SIGGRAPH*. 83, 84, 91

[Bloem and de Rooij, 2020] Bloem, P. and de Rooij, S. (2020). Large-scale network motif analysis using compression. *Data Mining and Knowledge Discovery*, 34(5):1421–1453. 153

[Blondel et al., 2008] Blondel, V. D., Guillaume, J.-L., Lambiotte, R., and Lefebvre, E. (2008). Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment*, 2008(10):P10008. 154, 249

[Bodnar et al., 2021] Bodnar, C., Frasca, F., Wang, Y., Otter, N., Montufar, G. F., Lio, P., and Bronstein, M. (2021). Weisfeiler and lehman go topological: Message passing simplicial networks. In *International Conference on Machine Learning (ICML)*, volume 83 of *Proceedings of Machine Learning Research*, pages 1026–1037. PMLR. 115

[Bogatskiy et al., 2020] Bogatskiy, A., Anderson, B., Offermann, J., Roussi, M., Miller, D., and Kondor, R. (2020). Lorentz group equivariant neural network for particle physics. In *International Conference on Machine Learning (ICML)*, volume 119 of *Proceedings of Machine Learning Research*, pages 992–1002. PMLR. 41

[Boget et al., 2022] Boget, Y., Gregorova, M., and Kalousis, A. (2022). Granngan: Graph annotation generative adversarial networks. *CoRR*, abs/2212.00449. 162

[Bogo et al., 2017] Bogo, F., Romero, J., Pons-Moll, G., and Black, M. J. (2017). Dynamic FAUST: Registering human bodies in motion. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 87

[Bojchevski et al., 2018] Bojchevski, A., Shchur, O., Zügner, D., and Günnemann, S. (2018). Netgan: Generating graphs via random walks. In *International Conference on Machine Learning (ICML)*, volume 80 of *Proceedings of Machine Learning Research*, pages 609–618. PMLR. 153

[Bokhnyak et al., 2019] Bokhnyak, S., Bronstein, M., Bouritsas, G., and Zafeiriou, S. (2019). Learning to represent & generate meshes with spiral convolutions. *7th International Conference on Learning Representations Workshop (ICLRW) on Learning on Graphs and Manifolds*. 45, 47

[Boldi et al., 2011] Boldi, P., Rosa, M., Santini, M., and Vigna, S. (2011). Layered label propagation: A multiresolution coordinate-free ordering for compressing social networks. In *Proceedings of the 20th international conference on World Wide Web*, pages 587–596. 152

[Bonald et al., 2020] Bonald, T., de Lara, N., Lutz, Q., and Charpentier, B. (2020). Scikit-network: Graph analysis in python. *Journal of Machine Learning Research*, 21(185):1–6. 259

[Booth et al., 2018] Booth, J., Roussos, A., Ponniah, A., Dunaway, D., and Zafeiriou, S. (2018). Large scale 3d morphable models. *International Journal of Computer Vision (IJCV)*. 84

[Booth et al., 2016] Booth, J., Roussos, A., Zafeiriou, S., Ponniah, A., and Dunaway, D. (2016). A 3d morphable model learnt from 10,000 faces. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 87

[Borgwardt et al., 2005] Borgwardt, K. M., Ong, C. S., Schönauer, S., Vishwanathan, S., Smola, A. J., and Kriegel, H.-P. (2005). Protein function prediction via graph kernels. *Bioinformatics*, 21(suppl_1):i47–i56. 154, 258

[Boscaini et al., 2016] Boscaini, D., Masci, J., Rodolà, E., and Bronstein, M. (2016). Learning shape correspondence with anisotropic convolutional neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, volume 29, pages 3189–3197. 60, 78, 83

[Bouritsas et al., 2019] Bouritsas, G., Bokhnyak, S., Ploumpis, S., Bronstein, M., and Zafeiriou, S. (2019). Neural 3d morphable models: Spiral convolutional networks for 3d shape representation learning and generation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 7213–7222. 45, 47

[Bouritsas et al., 2020] Bouritsas, G., Frasca, F., Zafeiriou, S. P., and Bronstein, M. (2020). Improving graph neural network expressivity via subgraph isomorphism counting. *International Conference on Machine Learning Workshop (ICMLW) on Graph Representation Learning and Beyond (GRL+)*. 45, 47

[Bouritsas et al., 2022] Bouritsas, G., Frasca, F., Zafeiriou, S. P., and Bronstein, M. (2022). Improving graph neural network expressivity via subgraph isomorphism counting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 45, 47, 249

[Bouritsas et al., 2021] Bouritsas, G., Loukas, A., Karalias, N., and Bronstein, M. (2021). Partition and code: learning how to compress graphs. *Advances in Neural Information Processing Systems(NeurIPS)*, 34:18603–18619. 46, 47

[Bresson and Laurent, 2017] Bresson, X. and Laurent, T. (2017). Residual gated graph convnets. *CoRR*, abs/1711.07553. 120

[Bron and Kerbosch, 1973] Bron, C. and Kerbosch, J. (1973). Finding all cliques of an undirected graph (algorithm 457). *Commun. ACM*, 16(9):575–576. 220

[Bronstein et al., 2021] Bronstein, M. M., Bruna, J., Cohen, T., and Velickovic, P. (2021). Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. *CoRR*, abs/2104.13478. 24, 57

[Bronstein et al., 2017] Bronstein, M. M., Bruna, J., LeCun, Y., Szlam, A., and Vandergheynst, P. (2017). Geometric deep learning: Going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42. 24

[Brown et al., 2020] Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. (2020). Language models are few-shot learners. In *Advances in Neural Information Processing Systems (NeurIPS)*. 38, 140

[Brüel Gabrielsson, 2020] Brüel Gabrielsson, R. (2020). Universal function approximation on graphs. *Advances in Neural Information Processing Systems (NeurIPS)*, 33:19762–19772. 41

[Bruna et al., 2014] Bruna, J., Zaremba, W., Szlam, A., and LeCun, Y. (2014). Spectral networks and locally connected networks on graphs. In Bengio, Y. and LeCun, Y., editors, *2nd International Conference on Learning Representations (ICLR)*. OpenReview.net. 30, 60, 74, 82

[Bubeck and Sellke, 2021] Bubeck, S. and Sellke, M. (2021). A universal law of robustness via isoperimetry. *Advances in Neural Information Processing Systems*, 34:28811–28822. 37, 138

[Burrows and Wheeler, 1994] Burrows, M. and Wheeler, D. (1994). A block-sorting lossless data compression algorithm. In *Digital SRC Research Report*. Citeseer. 141

[Cai et al., 1992] Cai, J., Fürer, M., and Immerman, N. (1992). An optimal lower bound on the number of variables for graph identifications. *Combinatorica*, 12(4):389–410. 58, 227

[Cai et al., 2021] Cai, T., Luo, S., Xu, K., He, D., Liu, T., and Wang, L. (2021). Graphnorm: A principled approach to accelerating graph neural network training. In *International Conference on Machine Learning, (ICML)*, volume 139 of *Proceedings of Machine Learning Research*, pages 1204–1215. PMLR. 119

[Cao et al., 2014] Cao, C., Weng, Y., Zhou, S., Tong, Y., and Zhou, K. (2014). Facewarehouse: A 3d facial expression database for visual computing. *IEEE Transactions on Visualization and Computer Graphics*. 84

[Cao and Kipf, 2018] Cao, N. D. and Kipf, T. (2018). Molgan: An implicit generative model for small molecular graphs. *CoRR*, abs/1805.11973. 153

[Capecchi et al., 2020] Capecchi, A., Probst, D., and Reymond, J.-L. (2020). One molecular fingerprint to rule them all: drugs, biomolecules, and the metabolome. *Journal of cheminformatics*, 12(1):1–15. 23

[Carletti et al., 2017] Carletti, V., Foggia, P., Saggese, A., and Vento, M. (2017). Introducing VF3: A new algorithm for subgraph isomorphism. In *Graph-Based Representations in Pattern Recognition - 11th IAPR-TC-15 International Workshop, GbRPR 2017, Anacapri, Italy, May 16-18, 2017, Proceedings*, volume 10310 of *Lecture Notes in Computer Science*, pages 128–139. 111

[Cayton, 2005] Cayton, L. (2005). Algorithms for manifold learning. *Univ. of California at San Diego Tech. Rep*, 12(1-17):1. 28

[Cereto-Massagué et al., 2015] Cereto-Massagué, A., Ojeda, M. J., Valls, C., Mulero, M., Garcia-Vallvé, S., and Pujadas, G. (2015). Molecular fingerprint similarity search in virtual screening. *Methods*, 71:58–63. 23

[Chaitin, 1969] Chaitin, G. J. (1969). On the simplicity and speed of programs for computing infinite sets of natural numbers. *Journal of the ACM (JACM)*, 16(3):407–422. 130

[Chakrabarti et al., 2004] Chakrabarti, D., Papadimitriou, S., Modha, D. S., and Faloutsos, C. (2004). Fully automatic cross-associations. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 79–88. 154

[Chamberlain et al., 2017] Chamberlain, B., Deisenroth, M., and Clough, J. (2017). Neural embeddings of graphs in hyperbolic space. In *Proceedings of the 13th International Workshop on Mining and Learning with Graphs (MLG)*. 28

[Chami et al., 2019] Chami, I., Ying, Z., Ré, C., and Leskovec, J. (2019). Hyperbolic graph convolutional neural networks. *Advances in Neural Information Processing Systems (NeurIPS)*, 32:4869–4880. 28

[Chen and Chen, 1993] Chen, T. and Chen, H. (1993). Approximations of continuous functionals by neural networks with application to dynamic systems. *IEEE Transactions on Neural networks*, 4(6):910–918. 40

[Chen and Chen, 1995a] Chen, T. and Chen, H. (1995a). Approximation capability to functions of several variables, nonlinear functionals, and operators by radial basis function neural networks. *IEEE Transactions on Neural Networks*, 6(4):904–910. 40

[Chen and Chen, 1995b] Chen, T. and Chen, H. (1995b). Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems. *IEEE Transactions on Neural Networks*, 6(4):911–917. 40

[Chen et al., 2020] Chen, Z., Chen, L., Villar, S., and Bruna, J. (2020). Can graph neural networks count substructures? In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33. 58, 59, 60, 96, 101, 106, 107, 116, 248

[Chen et al., 2019] Chen, Z., Villar, S., Chen, L., and Bruna, J. (2019). On the equivalence between graph isomorphism testing and function approximation with gnns. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 32, pages 15868–15876. 41, 60, 99, 101, 107, 114, 217

[Chen and Zhang, 2019] Chen, Z. and Zhang, H. (2019). Learning implicit fields for generative shape modeling. *CVPR*. 82

[Chiba and Nishizeki, 1985] Chiba, N. and Nishizeki, T. (1985). Arboricity and subgraph listing algorithms. *SIAM J. Comput.*, 14(1):210–223. 220

[Chierichetti et al., 2009] Chierichetti, F., Kumar, R., Lattanzi, S., Mitzenmacher, M., Panconesi, A., and Raghavan, P. (2009). On compressing social networks. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 219–228. 152

[Choi and Szpankowski, 2012] Choi, Y. and Szpankowski, W. (2012). Compression of graphical structures: Fundamental limits, algorithms, and experiments. *IEEE Transactions on Information Theory*, 58(2):620–638. 139, 140, 152, 236, 245

[Chrysos et al., 2021] Chrysos, G. G., Moschoglou, S., Bouritsas, G., Deng, J., Panagakis, Y., and Zafeiriou, S. (2021). Deep polynomial neural networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(8):4021–4034. 48, 161

[Chrysos et al., 2020] Chrysos, G. G., Moschoglou, S., Bouritsas, G., Panagakis, Y., Deng, J., and Zafeiriou, S. (2020). P-nets: Deep polynomial neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7325–7335. 47, 161

[Claude and Navarro, 2010] Claude, F. and Navarro, G. (2010). Fast and compact web graph representations. *ACM Transactions on the Web (TWEB)*, 4(4):1–31. 132, 152

[Clevert et al., 2016] Clevert, D., Unterthiner, T., and Hochreiter, S. (2016). Fast and accurate deep network learning by exponential linear units (elus). In *4th International Conference on Learning Representations (ICLR)*. OpenReview.net. 212

[Coifman and Maggioni, 2006] Coifman, R. R. and Maggioni, M. (2006). Diffusion wavelets. *Applied and computational harmonic analysis*, 21(1):53–94. 30

[Cootes et al., 2001] Cootes, T. F., Edwards, G. J., and Taylor, C. J. (2001). Active appearance models. *IEEE Transactions on pattern analysis and machine intelligence*, 23(6):681–685. 84

[Cootes et al., 1995] Cootes, T. F., Taylor, C. J., Cooper, D. H., and Graham, J. (1995). Active shape models-their training and application. *Computer vision and image understanding*, 61(1):38–59. 84

[Cordella et al., 2004] Cordella, L. P., Foggia, P., Sansone, C., and Vento, M. (2004). A (sub)graph isomorphism algorithm for matching large graphs. *IEEE Trans. Pattern Anal. Mach. Intell.*, 26(10):1367–1372. 111

[Corry, 2003] Corry, L. (2003). *Modern algebra and the rise of mathematical structures.* Springer Science & Business Media. 25

[Corso et al., 2020] Corso, G., Cavalleri, L., Beaini, D., Liò, P., and Veličković, P. (2020). Principal neighbourhood aggregation for graph nets. In *Advances in Neural Information Processing Systems (NeurIPS).* 120, 121

[Costa and De Grave, 2010] Costa, F. and De Grave, K. (2010). Fast neighborhood subgraph pairwise distance kernel. In *International Conference on Machine Learning (ICML)*, pages 255–262. Omnipress. 117

[Cotta et al., 2021] Cotta, L., Morris, C., and Ribeiro, B. (2021). Reconstruction for powerful graph representations. *Advances in Neural Information Processing Systems (NeurIPS)*, 34:1713–1726. 115

[Cover and Thomas, 2006] Cover, T. M. and Thomas, J. A. (2006). *Elements of information theory (2. ed.).* Wiley. 128, 132, 133

[Cox, 2016] Cox, D. (2016). Syntactically informed text compression with recurrent neural networks. *CoRR*, abs/1608.02893. 153

[Cybenko, 1989] Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314. 39

[Dai et al., 2020] Dai, H., Nazi, A., Li, Y., Dai, B., and Schuurmans, D. (2020). Scalable deep generative modeling for sparse graphs. In *International Conference on Machine Learning, (ICML)*, volume 119 of *Proceedings of Machine Learning Research*, pages 2302–2312. PMLR. 153

[Dai et al., 2019] Dai, Z., Yang, Z., Yang, Y., Carbonell, J. G., Le, Q. V., and Salakhutdinov, R. (2019). Transformer-xl: Attentive language models beyond a fixed-length context. In *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Volume 1: Long Papers*, pages 2978–2988. Association for Computational Linguistics. 105

[Danisch et al., 2018] Danisch, M., Balalau, O., and Sozio, M. (2018). Listing k-cliques in sparse real-world graphs. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web, WWW 2018, Lyon, France, April 23-27, 2018*, pages 589–598. ACM. 220

[Dareddy et al., 2019] Dareddy, M. R., Das, M., and Yang, H. (2019). motif2vec: Motif aware node representation learning for heterogeneous networks. In *2019 IEEE International Conference on Big Data (Big Data)*, pages 1052–1059. 117

[Dasoulas et al., 2021] Dasoulas, G., Lutzeyer, J. F., and Vazirgiannis, M. (2021). Learning parametrised graph shift operators. In *9th International Conference on Learning Representations*

(ICLR). OpenReview.net. 57

[Dasoulas et al., 2020] Dasoulas, G., Santos, L. D., Scaman, K., and Virmaux, A. (2020). Coloring graph neural networks for node disambiguation. In Bessiere, C., editor, *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, pages 2126–2132. ijcai.org. 41, 80, 99, 100, 105, 107, 110, 114, 217

[de Haan et al., 2020] de Haan, P., Cohen, T. S., and Welling, M. (2020). Natural graph networks. *Advances in Neural Information Processing Systems (NeurIPS)*, 33:3636–3646. 115, 119

[de Haan et al., 2021] de Haan, P., Weiler, M., Cohen, T., and Welling, M. (2021). Gauge equivariant mesh cnns: Anisotropic convolutions on geometric graphs. In *9th International Conference on Learning Representations (ICLR)*. OpenReview.net. 78, 83

[Debnath et al., 1991] Debnath, A. K., Lopez de Compadre, R. L., Debnath, G., Shusterman, A. J., and Hansch, C. (1991). 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. 258

[Defferrard et al., 2016] Defferrard, M., Bresson, X., and Vandergheynst, P. (2016). Convolutional neural networks on graphs with fast localized spectral filtering. *Advances in Neural Information Processing Systems (NIPS)*, 29:3837–3845. 30, 60, 74, 75, 76, 82, 86

[Dehmamy et al., 2019] Dehmamy, N., Barabási, A.-L., and Yu, R. (2019). Understanding the representation power of graph neural networks in learning graph topology. *Advances in Neural Information Processing Systems*, 32:15387–15397. 116

[Dehmer, 2008] Dehmer, M. (2008). Information-theoretic concepts for the analysis of complex networks. *Applied Artificial Intelligence*, 22(7-8):684–706. 152

[Dehmer et al., 2009] Dehmer, M., Barbarini, N., Varmuza, K., and Graber, A. (2009). A large scale analysis of information-theoretic network complexity measures using chemical structures. *PLoS One*, 4(12):e8057. 152

[Dell et al., 2018] Dell, H., Grohe, M., and Rattan, G. (2018). Lovász meets weisfeiler and leman. In *International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 107 of *LIPIcs*, pages 40:1–40:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik. 116

[Deutsch, 1996] Deutsch, P. (1996). Deflate compressed data format specification version 1.3. Technical report. 140

[Dhulipala et al., 2016] Dhulipala, L., Kabiljo, I., Karrer, B., Ottaviano, G., Pupyrev, S., and Shalita, A. (2016). Compressing graphs and indexes with recursive graph bisection. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1535–1544. 132, 152

[Dobson and Doig, 2003] Dobson, P. D. and Doig, A. J. (2003). Distinguishing enzyme structures from non-enzymes without alignments. *Journal of molecular biology*, 330(4):771–783. 258

[Dong and Sawin, 2020] Dong, Y. and Sawin, W. (2020). Copt: Coordinated optimal transport on graphs. *Advances in Neural Information Processing Systems (NeurIPS)*, 33. 153

[Donoho and Grimes, 2003] Donoho, D. L. and Grimes, C. (2003). Hessian eigenmaps: Locally linear embedding techniques for high-dimensional data. *Proceedings of the National Academy of Sciences*, 100(10):5591–5596. 28

[Dorn, 2010] Dorn, F. (2010). Planar subgraph isomorphism revisited. In *27th International Symposium on Theoretical Aspects of Computer Science, STACS 2010, March 4-6, 2010, Nancy, France*, volume 5 of *LIPIcs*, pages 263–274. Schloss Dagstuhl - Leibniz-Zentrum für Informatik. 221

[Douglas, 2011] Douglas, B. L. (2011). The weisfeiler-lehman method and graph isomorphism testing. *arXiv preprint arXiv:1101.5211*. 108

[Du et al., 2019a] Du, S. S., Hou, K., Salakhutdinov, R., Póczos, B., Wang, R., and Xu, K. (2019a). Graph neural tangent kernel: Fusing graph neural networks with graph kernels. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 32, pages 5724–5734. 119, 229

[Du et al., 2019b] Du, S. S., Hou, K., Salakhutdinov, R. R., Poczos, B., Wang, R., and Xu, K. (2019b). Graph neural tangent kernel: Fusing graph neural networks with graph kernels. *Advances in Neural Information Processing Systems (NeurIPS)*, 32. 102

[Du et al., 2019c] Du, S. S., Lee, J. D., Li, H., Wang, L., and Zhai, X. (2019c). Gradient descent finds global minima of deep neural networks. In *International Conference on Machine Learning (ICML)*, volume 97 of *Proceedings of Machine Learning Research*, pages 1675–1685. PMLR. 229

[Du et al., 2019d] Du, S. S., Zhai, X., Póczos, B., and Singh, A. (2019d). Gradient descent provably optimizes over-parameterized neural networks. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net. 37, 138

[Dubois et al., 2021] Dubois, Y., Bloem-Reddy, B., Ullrich, K., and Maddison, C. J. (2021). Lossy compression for lossless prediction. *Advances in Neural Information Processing Systems*, 34:14014–14028. 128, 136, 139, 153

[Duda, 2009] Duda, J. (2009). Asymmetric numeral systems. *CoRR*, abs/0902.0271. 134

[Duda, 2013] Duda, J. (2013). Asymmetric numeral systems: entropy coding combining speed of huffman coding with compression rate of arithmetic coding. *CoRR*, abs/1311.2540. 134, 153, 261

[Dupont et al., 2022] Dupont, E., Kim, H., Eslami, S. A., Rezende, D. J., and Rosenbaum, D. (2022). From data to functa: Your data point is a function and you can treat it like one. In *International Conference on Machine Learning (ICML)*, volume 162 of *Proceedings of Machine Learning Research*, pages 5694–5725. PMLR. 29, 82

[Duvenaud et al., 2015] Duvenaud, D. K., Maclaurin, D., Iparraguirre, J., Bombarell, R., Hirzel, T., Aspuru-Guzik, A., and Adams, R. P. (2015). Convolutional networks on graphs for learning molecular fingerprints. In *Advances in Neural Information Processing systems (NIPS)*, volume 28, pages 2224–2232. 60

[Dwivedi and Bresson, 2021] Dwivedi, V. P. and Bresson, X. (2021). A generalization of transformer networks to graphs. *AAAI Workshop on Deep Learning on Graphs: Methods and Applications*. 115

[Dwivedi et al., 2020] Dwivedi, V. P., Joshi, C. K., Laurent, T., Bengio, Y., and Bresson, X. (2020). Benchmarking graph neural networks. *CoRR*, abs/2003.00982. 115, 119, 120, 230, 259

[Dwivedi et al., 2022] Dwivedi, V. P., Luu, A. T., Laurent, T., Bengio, Y., and Bresson, X. (2022). Graph neural networks with learnable structural and positional representations. In *10th International Conference on Learning Representations (ICLR)*. OpenReview.net. 115

[Dym and Maron, 2021] Dym, N. and Maron, H. (2021). On the universality of rotation equivariant point cloud networks. In *9th International Conference on Learning Representations (ICLR)*. OpenReview.net. 41

[Dziuk and Elliott, 2013] Dziuk, G. and Elliott, C. M. (2013). Finite element methods for surface pdes. *Acta Numerica*, 22:289–396. 65

[Eppstein, 1995] Eppstein, D. (1995). Subgraph isomorphism in planar graphs and related problems. In *Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, 22-24 January 1995. San Francisco, California, USA*, pages 632–640. ACM/SIAM. 221

[Erdös and Rényi, 1959] Erdös, P. and Rényi, A. (1959). On random graphs i. *Publicationes Mathematicae Debrecen*, 6:290–297. 23

[Erdos and Rényi, 1963] Erdos, P. and Rényi, A. (1963). Asymmetric graphs. *Acta Mathematica Academiae Scientiarum Hungaricae*, 14(295-315):3. 236, 238

[Euler, 1736] Euler, L. (1736). Solutio problematis ad geometriam situs pertinentis. *Commentarii academiae scientiarum Petropolitanae*, pages 128–140. 23

[Fan et al., 2017] Fan, H., Su, H., and Guibas, L. J. (2017). A point set generation network for 3d object reconstruction from a single image. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 605–613. 81

[Fefferman et al., 2016] Fefferman, C., Mitter, S., and Narayanan, H. (2016). Testing the manifold hypothesis. *Journal of the American Mathematical Society*, 29(4):983–1049. 28

[Feldman et al., 2022] Feldman, O., Boyarski, A., Feldman, S., Kogan, D., Mendelson, A., and Baskin, C. (2022). Weisfeiler and leman go infinite: Spectral and combinatorial pre-colorings. *CoRR*, abs/2201.13410. 115

[Feng et al., 2019] Feng, Y., Feng, Y., You, H., Zhao, X., and Gao, Y. (2019). Meshnet: Mesh neural network for 3d shape representation. In *AAAI Conference on Artificial Intelligence*, pages 8279–8286. AAAI Press. 82

[Fereydounian et al., 2022] Fereydounian, M., Hassani, H., Dadashkarimi, J., and Karbasi, A. (2022). The exact class of graph functions generated by graph neural networks. *CoRR*, abs/2202.08833. 116

[Ferreira et al., 2011] Ferreira, R. A., Grossi, R., and Rizzi, R. (2011). Output-sensitive listing of bounded-size trees in undirected graphs. In *Algorithms - ESA 2011 - 19th Annual European Symposium, Saarbrücken, Germany, September 5-9, 2011. Proceedings*, volume 6942 of *Lecture Notes in Computer Science*, pages 275–286. Springer. 221

[Ferreira et al., 2014] Ferreira, R. A., Grossi, R., Rizzi, R., Sacomoto, G., and Sagot, M. (2014). Amortized $\tilde{o}(|v|)$ -delay algorithm for listing chordless cycles in undirected graphs. In *Algorithms - ESA 2014 - 22th Annual European Symposium, Wroclaw, Poland, September 8-10, 2014. Proceedings*, volume 8737 of *Lecture Notes in Computer Science*, pages 418–429. Springer. 220

[Fey and Lenssen, 2019a] Fey, M. and Lenssen, J. E. (2019a). Fast graph representation learning with pytorch geometric. *CoRR*, abs/1903.02428. 226

[Fey and Lenssen, 2019b] Fey, M. and Lenssen, J. E. (2019b). Fast graph representation learning with PyTorch Geometric. In *7th International Conference on Learning Representations (ICLRW) Workshop on Representation Learning on Graphs and Manifolds*. 259

[Fey et al., 2018] Fey, M., Lenssen, J. E., Weichert, F., and Müller, H. (2018). Splinecnn: Fast geometric deep learning with continuous b-spline kernels. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 83

[Fey et al., 2020] Fey, M., Yuen, J. G., and Weichert, F. (2020). Hierarchical inter-message passing for learning on molecular graphs. In *ICML Graph Representation Learning and Beyond (GRL+) Workhop*. 120, 121

[Finn et al., 2017] Finn, C., Abbeel, P., and Levine, S. (2017). Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pages 1126–1135. PMLR. 30

[Foldes, 1994] Foldes, S. (1994). *Fundamental structures of algebra and discrete mathematics*. John Wiley & Sons. 25

[Fout et al., 2017] Fout, A., Byrd, J., Shariat, B., and Ben-Hur, A. (2017). Protein interface prediction using graph convolutional networks. In *Advances in Neural Information Processing Systems (NIPS)*, volume 30, pages 6530–6539. 60

[Franceschi et al., 2018] Franceschi, L., Frasconi, P., Salzo, S., Grazzi, R., and Pontil, M. (2018). Bilevel programming for hyperparameter optimization and meta-learning. In *International Conference on*

Machine Learning, pages 1568–1577. PMLR. 30

[Frankle and Carbin, 2019] Frankle, J. and Carbin, M. (2019). The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *7th International Conference on Learning Representations (ICLR)*. OpenReview.net. 154, 256

[Frasca et al., 2022a] Frasca, F., Bevilacqua, B., Bronstein, M. M., and Maron, H. (2022a). Understanding and extending subgraph gnns by rethinking their symmetries. In *Advances in Neural Information Processing Systems (NeurIPS)*. 116

[Frasca et al., 2022b] Frasca, F., Bevilacqua, B., and Maron, H. (2022b). Exploring the practical and theoretical landscape of expressive graph neural networks. https://www.youtube.com/watch?v=ASQYjbUBYzs. 113

[Fuchs et al., 2020] Fuchs, F., Worrall, D., Fischer, V., and Welling, M. (2020). Se (3)-transformers: 3d roto-translation equivariant attention networks. *Advances in Neural Information Processing Systems (NeurIPS)*, 33:1970–1981. 81

[Fukushima and Miyake, 1982] Fukushima, K. and Miyake, S. (1982). Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In *Competition and cooperation in neural nets*, pages 267–285. Springer. 30

[Fürer, 2010] Fürer, M. (2010). On the power of combinatorial and spectral invariants. *Linear algebra and its applications*, 432(9):2373–2380. 116

[Fürer, 2017] Fürer, M. (2017). On the combinatorial power of the weisfeiler-lehman algorithm. In *International Conference on Algorithms and Complexity (CIAC)*, volume 10236 of *Lecture Notes in Computer Science*, pages 260–271. Springer. 58, 116, 227

[Gage, 1994] Gage, P. (1994). A new algorithm for data compression. *C Users Journal*, 12(2):23–38. 140

[Gama et al., 2020] Gama, F., Ribeiro, A., and Bruna, J. (2020). Stability of graph neural networks to relative perturbations. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 9070–9074. IEEE. 57

[Ganea et al., 2018] Ganea, O., Bécigneul, G., and Hofmann, T. (2018). Hyperbolic neural networks. *Advances in Neural Information Processing Systems (NeurIPS)*, 31:5350–5360. 28

[Gao et al., 2022] Gao, Z., Yan, J., Zhai, G., Zhang, J., and Yang, X. (2022). Robust mesh representation learning via efficient local structure-aware anisotropic convolution. *IEEE Transactions on Neural Networks and Learning Systems*. 161

[Garg et al., 2020] Garg, V., Jegelka, S., and Jaakkola, T. (2020). Generalization and representational limits of graph neural networks. In *International Conference on Machine Learning (ICML)*, volume 82 of *Proceedings of Machine Learning Research*, pages 3419–3430. PMLR. 101, 102, 116

[Garg and Jaakkola, 2019] Garg, V. K. and Jaakkola, T. S. (2019). Solving graph compression via optimal transport. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 32, pages 8012–8023. 153

[Garland and Heckbert, 1997] Garland, M. and Heckbert, P. S. (1997). Surface simplification using quadric error metrics. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 209–216. 85

[Garnelo et al., 2018] Garnelo, M., Rosenbaum, D., Maddison, C. J., Ramalho, T., Saxton, D., Shanahan, M., Teh, Y. W., Rezende, D. J., and Eslami, S. M. A. (2018). Conditional neural processes. *CoRR*, abs/1807.01613. 30

[Gärtner et al., 2003] Gärtner, T., Flach, P. A., and Wrobel, S. (2003). On graph kernels: Hardness results and efficient alternatives. In *Computational Learning Theory and Kernel Machines (COLT)*, volume 2777, pages 129–143. Springer. 119

[Gavish et al., 2010] Gavish, M., Nadler, B., and Coifman, R. R. (2010). Multiscale wavelets on trees, graphs and high dimensional data: Theory and applications to semi supervised learning. In *International Conference on Machine Learning (ICML)*, pages 367–374. Omnipress. 30

[Geerts, 2020] Geerts, F. (2020). The expressive power of kth-order invariant graph networks. *CoRR*, abs/2007.12035. 100, 114

[Geiger and Smidt, 2022] Geiger, M. and Smidt, T. (2022). e3nn: Euclidean neural networks. *CoRR*, abs/2207.09453. 31

[Gilbert, 1959] Gilbert, E. N. (1959). Random graphs. *The Annals of Mathematical Statistics*, 30(4):1141–1144. 23

[Gilmer et al., 2017] Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. (2017). Neural message passing for quantum chemistry. In *International Conference on Machine Learning (ICML)*, volume 70 of *Proceedings of Machine Learning Research*, pages 1263–1272. PMLR. 60, 259

[Girdhar et al., 2016] Girdhar, R., Fouhey, D. F., Rodriguez, M., and Gupta, A. (2016). Learning a predictable and generative vector representation for objects. In *European Conference on Computer Vision (ECCV)*. Springer. 81

[Gómez-Bombarelli et al., 2018] Gómez-Bombarelli, R., Wei, J. N., Duvenaud, D., Hernández-Lobato, J. M., Sánchez-Lengeling, B., Sheberla, D., Aguilera-Iparraguirre, J., Hirzel, T. D., Adams, R. P., and Aspuru-Guzik, A. (2018). Automatic chemical design using a data-driven continuous representation of molecules. *ACS central science*, 4(2):268–276. 119, 259

[Gong et al., 2019] Gong, S., Chen, L., Bronstein, M., and Zafeiriou, S. (2019). Spiralnet++: A fast and highly efficient mesh convolution operator. In *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops (ICCVW)*, pages 0–0. 160

[Gonzalez et al., 2021] Gonzalez, G., Gong, S., Laponogov, I., Bronstein, M., and Veselkov, K. (2021). Predicting anticancer hyperfoods with graph convolutional networks. *Human Genomics*, 15(1):1–12. 65

[Goodall, 1991] Goodall, C. (1991). Procrustes methods in the statistical analysis of shape. *Journal of the Royal Statistical Society: Series B (Methodological)*, 53(2):285–321. 84

[Gori et al., 2005] Gori, M., Monfardini, G., and Scarselli, F. (2005). 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. 60

[Gower, 1975] Gower, J. C. (1975). Generalized procrustes analysis. *Psychometrika*, 40(1):33–51. 84

[Goyal et al., 2019] Goyal, M., Tatwawadi, K., Chandak, S., and Ochoa, I. (2019). Deepzip: Lossless data compression using recurrent neural networks. In *2019 Data Compression Conference (DCC)*, page 575. IEEE. 153

[Granovetter, 1982] Granovetter, M. (1982). The strength of weak ties: A network theory revisited. In *Sociological Theory*, pages 105–130. 112

[Gropp et al., 2020] Gropp, A., Yariv, L., Haim, N., Atzmon, M., and Lipman, Y. (2020). Implicit geometric regularization for learning shapes. In *International Conference on Machine Learning (ICML),*, volume 119 of *Proceedings of Machine Learning Research*, pages 3789–3799. PMLR. 82

[Grover and Leskovec, 2016] Grover, A. and Leskovec, J. (2016). node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864. 28

[Grünwald, 2007] Grünwald, P. D. (2007). *The minimum description length principle*. MIT press. 132, 134

[Gulrajani et al., 2017] Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., and Courville, A. C. (2017). Improved training of wasserstein gans. In *Advances in Neural Information Processing Systems (NIPS)*, volume 30, pages 5767–5777. 86

[Haemers, 2000] Haemers, W. (2000). Matrix techniques for strongly regular graphs and related geometries. *Intensive Course on Finite Geometry and its Applications, University of Ghent*. 76

[Haim et al., 2019] Haim, N., Segol, N., Ben-Hamu, H., Maron, H., and Lipman, Y. (2019). Surface networks via general covers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 632–641. 83

[Hamilton et al., 2017] Hamilton, W., Ying, Z., and Leskovec, J. (2017). Inductive representation learning on large graphs. *Advances in Neural Information Processing Systems (NIPS)*, 30:1024–1034. 28, 120

[Hammond et al., 2011] Hammond, D. K., Vandergheynst, P., and Gribonval, R. (2011). Wavelets on graphs via spectral graph theory. *Applied and Computational Harmonic Analysis*, 30(2):129–150. 75

[Han et al., 2016] Han, S., Mao, H., and Dally, W. J. (2016). Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. In *4th International Conference on Learning Representations (ICLR)*. OpenReview.net. 155

[Han et al., 2015] Han, S., Pool, J., Tran, J., and Dally, W. J. (2015). Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 28, pages 1135–1143. 155

[Han et al., 2013] Han, W., Lee, J., and Lee, J. (2013). Turbo$_{\text{iso}}$: towards ultrafast and robust subgraph isomorphism search in large graph databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2013, New York, NY, USA, June 22-27, 2013*, pages 337–348. ACM. 111

[Hanocka et al., 2019] Hanocka, R., Hertz, A., Fish, N., Giryes, R., Fleishman, S., and Cohen-Or, D. (2019). Meshcnn: a network with an edge. *ACM Transactions on Graphics (TOG)*, 38(4):1–12. 82

[Harary and Palmer, 2014] Harary, F. and Palmer, E. M. (2014). *Graphical enumeration*. Elsevier. 244

[Hassan et al., 2021] Hassan, M., Ghosh, P., Tesch, J., Tzionas, D., and Black, M. J. (2021). Populating 3d scenes by learning human-scene interaction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 14708–14718. 160

[Havasi et al., 2019] Havasi, M., Peharz, R., and Hernández-Lobato, J. M. (2019). Minimal random code learning: Getting bits back from compressed model parameters. In *7th International Conference on Learning Representations (ICLR)*. OpenReview.net. 155, 163

[He et al., 2022] He, X., Hooi, B., Laurent, T., Perold, A., LeCun, Y., and Bresson, X. (2022). A generalization of vit/mlp-mixer to graphs. *CoRR*, abs/2212.13350. 115

[Heimann and Meinzer, 2009] Heimann, T. and Meinzer, H.-P. (2009). Statistical shape models for 3d medical image segmentation: a review. *Medical image analysis*, 13(4):543–563. 84

[Helma et al., 2001] Helma, C., King, R. D., Kramer, S., and Srinivasan, A. (2001). The predictive toxicology challenge 2000–2001. *Bioinformatics*, 17(1):107–108. 154, 258

[Hinton and Van Camp, 1993] Hinton, G. E. and Van Camp, D. (1993). Keeping the neural networks simple by minimizing the description length of the weights. In *Proceedings of the sixth annual conference on Computational learning theory*, pages 5–13. 153

[Hinton et al., 2015] Hinton, G. E., Vinyals, O., and Dean, J. (2015). Distilling the knowledge in a neural network. *CoRR*, abs/1503.02531. 155

[Ho et al., 2022] Ho, J., Chan, W., Saharia, C., Whang, J., Gao, R., Gritsenko, A. A., Kingma, D. P., Poole, B., Norouzi, M., Fleet, D. J., and Salimans, T. (2022). Imagen video: High definition video generation with diffusion models. *CoRR*, abs/2210.02303. 38

[Ho et al., 2019] Ho, J., Lohn, E., and Abbeel, P. (2019). Compression with flows via local bits-back coding. In *Advances in Neural Information Processing Systems(NeurIPS)*, volume 32, pages 3874–3883. 153

[Hoàng et al., 2013] Hoàng, C. T., Kaminski, M., Sawada, J., and Sritharan, R. (2013). Finding and listing induced paths and cycles. *Discret. Appl. Math.*, 161(4-5):633–641. 221

[Hocevar and Demsar, 2014] Hocevar, T. and Demsar, J. (2014). A combinatorial approach to graphlet counting. *Bioinform.*, 30(4):559–565. 111

[Holland et al., 1983] Holland, P. W., Laskey, K. B., and Leinhardt, S. (1983). Stochastic blockmodels: First steps. *Social networks*, 5(2):109–137. 23, 152

[Holland and Leinhardt, 1976] Holland, P. W. and Leinhardt, S. (1976). Local structure in social networks. *Sociological methodology*, 7:1–45. 116

[Hoogeboom et al., 2022] Hoogeboom, E., Gritsenko, A. A., Bastings, J., Poole, B., van den Berg, R., and Salimans, T. (2022). Autoregressive diffusion models. In *10th International Conference on Learning Representations (ICLR)*. OpenReview.net. 153

[Hoogeboom et al., 2019] Hoogeboom, E., Peters, J. W. T., van den Berg, R., and Welling, M. (2019). Integer discrete flows and lossless compression. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 32, pages 12134–12144. 153

[Hornik, 1991] Hornik, K. (1991). Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251–257. 39

[Hornik et al., 1989] Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366. 39

[Horváth et al., 2004] Horváth, T., Gärtner, T., and Wrobel, S. (2004). Cyclic pattern kernels for predictive graph mining. In *ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD)*, pages 158–167. ACM. 117

[Hospedales et al., 2021] Hospedales, T., Antoniou, A., Micaelli, P., and Storkey, A. (2021). Meta-learning in neural networks: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 44(9):5149–5169. 30

[Hotelling, 1933] Hotelling, H. (1933). Analysis of a complex of statistical variables into principal components. *Journal of educational psychology*, 24(6):417. 84

[Houbraken et al., 2014] Houbraken, M., Demeyer, S., Michoel, T., Audenaert, P., Colle, D., and Pickavet, M. (2014). The index-based subgraph matching algorithm with general symmetries (ismags): exploiting symmetry for faster subgraph enumeration. *PloS one*, 9(5):e97896. 111

[Hu et al., 2020] Hu, W., Fey, M., Zitnik, M., Dong, Y., Ren, H., Liu, B., Catasta, M., and Leskovec, J. (2020). Open graph benchmark: Datasets for machine learning on graphs. *Advances in neural information processing systems*, 33:22118–22133. 120, 230, 232

[Hua et al., 2018] Hua, B.-S., Tran, M.-K., and Yeung, S.-K. (2018). Pointwise convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 984–993. 81

[Huffman, 1952] Huffman, D. A. (1952). A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101. 133

[Irwin et al., 2012a] Irwin, J. J., Sterling, T., Mysinger, M. M., Bolstad, E. S., and Coleman, R. G. (2012a). ZINC: A free tool to discover chemistry for biology. *Journal of chemical information and modeling*, 52(7):1757–1768. 119

[Irwin et al., 2012b] Irwin, J. J., Sterling, T., Mysinger, M. M., Bolstad, E. S., and Coleman, R. G. (2012b). Zinc: a free tool to discover chemistry for biology. *Journal of chemical information and modeling*, 52(7):1757–1768. 154, 259

[Isufi et al., 2022] Isufi, E., Gama, F., Shuman, D. I., and Segarra, S. (2022). Graph filters for signal processing and machine learning on graphs. *CoRR*, abs/2211.08854. 57

[Jacot et al., 2018] Jacot, A., Hongler, C., and Gabriel, F. (2018). Neural tangent kernel: Convergence and generalization in neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 8580–8589. 37, 229

[Jain and Seshadhri, 2020] Jain, S. and Seshadhri, C. (2020). The power of pivoting for exact clique counting. In *WSDM '20: The Thirteenth ACM International Conference on Web Search and Data Mining, Houston, TX, USA, February 3-7, 2020*, pages 268–276. ACM. 220

[Jegelka, 2022] Jegelka, S. (2022). Theory of graph neural networks: Representation and learning. *CoRR*, abs/2204.07697. 113

[Jiang et al., 2020] Jiang, B., Zhang, J., Hong, Y., Luo, J., Liu, L., and Bao, H. (2020). Bcnet: Learning body and cloth shape from a single image. In *European Conference on Computer Vision*, pages 18–35. Springer. 161

[Jiang et al., 2019] Jiang, Y., Krishnan, D., Mobahi, H., and Bengio, S. (2019). Predicting the generalization gap in deep networks with margin distributions. In *7th International Conference on Learning Representations (ICLR)*. OpenReview.net. 37

[Jin et al., 2018a] Jin, W., Barzilay, R., and Jaakkola, T. S. (2018a). Junction tree variational autoencoder for molecular graph generation. In *International Conference on Machine Learning, (ICML)*, volume 80 of *Proceedings of Machine Learning Research*, pages 2328–2337. PMLR. 119, 259

[Jin et al., 2018b] Jin, W., Barzilay, R., and Jaakkola, T. S. (2018b). Junction tree variational autoencoder for molecular graph generation. In *International Conference on Machine Learning, (ICML)*, volume 80 of *Proceedings of Machine Learning Research*, pages 2328–2337. PMLR. 153

[Jin et al., 2020] Jin, Y., Loukas, A., and JaJa, J. (2020). Graph coarsening with preserved spectral properties. In Chiappa, S. and Calandra, R., editors, *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, volume 108 of *Proceedings of Machine Learning Research*, pages 4452–4462. PMLR. 153

[Johnson, 1975] Johnson, D. B. (1975). Finding all the elementary circuits of a directed graph. *SIAM J. Comput.*, 4(1):77–84. 220

[Joshi et al., 2022] Joshi, C. K., Bodnar, C., Mathis, S. V., Cohen, T., and Lio, P. (2022). On the expressive power of geometric graph neural networks. In *The First Learning on Graphs Conference*. 161

[Karalias and Loukas, 2020] Karalias, N. and Loukas, A. (2020). Erdos goes neural: an unsupervised learning framework for combinatorial optimization on graphs. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33. 151

[Karger, 1993] Karger, D. R. (1993). Global min-cuts in rnc, and other ramifications of a simple min-cut algorithm. In *SODA*, volume 93, pages 21–30. Citeseer. 249

[Karrer and Newman, 2011] Karrer, B. and Newman, M. E. (2011). Stochastic blockmodels and community structure in networks. *Physical review E*, 83(1):016107. 152

[Karypis and Kumar, 1998] Karypis, G. and Kumar, V. (1998). A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on scientific Computing*, 20(1):359–392. 249

[Kashtan et al., 2004] Kashtan, N., Itzkovitz, S., Milo, R., and Alon, U. (2004). Efficient sampling algorithm for estimating subgraph concentrations and detecting network motifs. *Bioinformatics*, 20(11):1746–1758. 111

[Kelly et al., 1957] Kelly, P. J. et al. (1957). A congruence theorem for trees. *Pacific Journal of Mathematics*, 7(1):961–968. 107

[Keriven and Peyré, 2019] Keriven, N. and Peyré, G. (2019). Universal invariant and equivariant graph neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 32, pages 7090–7099. 41, 60, 100, 114

[Ketkar et al., 2005] Ketkar, N. S., Holder, L. B., and Cook, D. J. (2005). Subdue: Compression-based frequent pattern discovery in graph data. In *Proceedings of the 1st international workshop on open*

source data mining: frequent pattern mining implementations, pages 71–76. 153

[Kidger and Lyons, 2020] Kidger, P. and Lyons, T. J. (2020). Universal approximation with deep narrow networks. In Abernethy, J. D. and Agarwal, S., editors, *Conference on Learning Theory, COLT 2020, 9-12 July 2020, Virtual Event [Graz, Austria]*, volume 125 of *Proceedings of Machine Learning Research*, pages 2306–2327. PMLR. 39

[Kingma et al., 2021] Kingma, D. P., Salimans, T., Poole, B., and Ho, J. (2021). Variational diffusion models. *CoRR*, abs/2107.00630. 153

[Kingma et al., 2019] Kingma, F. H., Abbeel, P., and Ho, J. (2019). Bit-swap: Recursive bits-back coding for lossless compression with hierarchical latent variables. In *International Conference on Machine Learning, (ICML)*, volume 97 of *Proceedings of Machine Learning Research*, pages 3408–3417. PMLR. 153

[Kipf et al., 2018] Kipf, T. N., Fetaya, E., Wang, K., Welling, M., and Zemel, R. S. (2018). Neural relational inference for interacting systems. In *International Conference on Machine Learning, (ICML)*, volume 80 of *Proceedings of Machine Learning Research*, pages 2693–2702. PMLR. 60

[Kipf and Welling, 2017] Kipf, T. N. and Welling, M. (2017). Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations (ICLR)*. OpenReview.net. 28, 60, 74, 120, 121

[Klicpera et al., 2020] Klicpera, J., Groß, J., and Günnemann, S. (2020). Directional message passing for molecular graphs. In *8th International Conference on Learning Representations (ICLR)*. OpenReview.net. 161

[Kolmogorov, 1998] Kolmogorov, A. N. (1998). On tables of random numbers. *Theoretical Computer Science*, 207(2):387–395. 130

[Kolouri et al., 2021] Kolouri, S., Naderializadeh, N., Rohde, G. K., and Hoffmann, H. (2021). Wasserstein embedding for graph learning. In *9th International Conference on Learning Representations (ICLR)*. 119

[Kondor, 2018] Kondor, R. (2018). N-body networks: a covariant hierarchical neural network architecture for learning atomic potentials. *CoRR*, abs/1803.01588. 31

[Kondor et al., 2018] Kondor, R., Son, H. T., Pan, H., Anderson, B. M., and Trivedi, S. (2018). Covariant compositional networks for learning graphs. In *6th International Conference on Learning Representations, Workshop Track Proceedings (ICLRW)*. OpenReview.net. 60, 114

[Kondor and Trivedi, 2018] Kondor, R. and Trivedi, S. (2018). On the generalization of equivariance and convolution in neural networks to the action of compact groups. In *International Conference on Machine Learning (ICML)*, volume 80 of *Proceedings of Machine Learning Research*, pages 2752–2760. PMLR. 60

[Körner, 1973] Körner, J. (1973). Coding of an information source having ambiguous alphabet and the entropy of graphs. In *6th Prague conference on information theory*, pages 411–425. 152

[Kostrikov et al., 2017] Kostrikov, I., Jiang, Z., Panozzo, D., Zorin, D., and Bruna, J. (2017). Surface networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 82

[Koutra et al., 2014] Koutra, D., Kang, U., Vreeken, J., and Faloutsos, C. (2014). Vog: Summarizing and understanding large graphs. In *Proceedings of the 2014 SIAM international conference on data mining*, pages 91–99. SIAM. 153

[Kovachki et al., 2021a] Kovachki, N., Lanthaler, S., and Mishra, S. (2021a). On universal approximation and error bounds for fourier neural operators. *Journal of Machine Learning Research*, 22:Art–No. 40

[Kovachki et al., 2021b] Kovachki, N. B., Li, Z., Liu, B., Azizzadenesheli, K., Bhattacharya, K., Stuart, A. M., and Anandkumar, A. (2021b). Neural operator: Learning maps between function spaces. *CoRR*, abs/2108.08481. 29, 40, 161

[Kratsios and Bilokopytov, 2020] Kratsios, A. and Bilokopytov, I. (2020). Non-euclidean universal approximation. *Advances in Neural Information Processing Systems (NeurIPS)*, 33:10635–10646. 39

[Kratsios and Papon, 2022] Kratsios, A. and Papon, L. (2022). Universal approximation theorems for differentiable geometric deep learning. *Journal of Machine Learning Research*, 23(196):1–73. 39

[Kreher and Stinson, 2020] Kreher, D. L. and Stinson, D. R. (2020). *Combinatorial algorithms: generation, enumeration, and search*. CRC press. 262

[Kreuzer et al., 2021] Kreuzer, D., Beaini, D., Hamilton, W., Létourneau, V., and Tossou, P. (2021). Rethinking graph transformers with spectral attention. *Advances in Neural Information Processing Systems (NeurIPS)*, 34:21618–21629. 115

[Kriege et al., 2020] Kriege, N. M., Johansson, F. D., and Morris, C. (2020). A survey on graph kernels. *Applied Network Science*, 5(1):6. 24

[Kriege and Mutzel, 2012] Kriege, N. M. and Mutzel, P. (2012). Subgraph matching kernels for attributed graphs. In *International Conference on Machine Learning, (ICML)*. icml.cc / Omnipress. 117, 258

[Krizhevsky et al., 2017] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2017). Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90. 30

[Kulon et al., 2020] Kulon, D., Guler, R. A., Kokkinos, I., Bronstein, M. M., and Zafeiriou, S. (2020). Weakly-supervised mesh-convolutional hand reconstruction in the wild. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4990–5000. 160

[Kusner et al., 2017] Kusner, M. J., Paige, B., and Hernández-Lobato, J. M. (2017). Grammar variational autoencoder. In *International Conference on Machine Learning, (ICML)*, volume 70 of *Proceedings of Machine Learning Research*, pages 1945–1954. PMLR. 119, 259

[Ladner, 1975] Ladner, R. E. (1975). On the structure of polynomial time reducibility. *Journal of the ACM (JACM)*, 22(1):155–171. 55

[Lanthaler et al., 2022] Lanthaler, S., Mishra, S., and Karniadakis, G. E. (2022). Error estimates for deeponets: A deep learning framework in infinite dimensions. *Transactions of Mathematics and Its Applications*, 6(1):tnac001. 40

[Latapy, 2008] Latapy, M. (2008). Main-memory triangle computations for very large (sparse (power-law)) graphs. *Theor. Comput. Sci.*, 407(1-3):458–473. 220

[LeCun et al., 1989] LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551. 30

[LeCun et al., 1998] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324. 30

[Lee et al., 2019] Lee, J. B., Rossi, R., Kong, X., Kim, S., Koh, E., and Rao, A. (2019). Graph convolutional networks with motif-based attention. In *28th ACM International Conference on Information and Knowledge Management (CIKM)*, pages 499–508. ACM. 117

[LeFevre and Terzi, 2010] LeFevre, K. and Terzi, E. (2010). Grass: Graph structure summarization. In *Proceedings of the 2010 SIAM International Conference on Data Mining*, pages 454–465. SIAM. 153

[Lewis, 1983] Lewis, H. R. (1983). Michael r. πgarey and david s. johnson. computers and intractability. a guide to the theory of np-completeness. wh freeman and company, san francisco1979, x+ 338 pp. *The Journal of Symbolic Logic*, 48(2):498–500. 55

[Li et al., 2020] Li, P., Wang, Y., Wang, H., and Leskovec, J. (2020). Distance encoding: Design provably more powerful neural networks for graph representation learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pages 4465–4478. 115

[Li et al., 2017a] Li, T., Bolkart, T., Black, M. J., Li, H., and Romero, J. (2017a). Learning a model of facial shape and expression from 4D scans. *ACM Transactions on Graphics, (Proc. SIGGRAPH Asia)*. 84

[Li et al., 2018] Li, Y., Vinyals, O., Dyer, C., Pascanu, R., and Battaglia, P. W. (2018). Learning deep generative models of graphs. *CoRR*, abs/1803.03324. 153

[Li et al., 2017b] Li, Z., Zhou, F., Chen, F., and Li, H. (2017b). Meta-sgd: Learning to learn quickly for few shot learning. *CoRR*, abs/1707.09835. 30

[Liao et al., 2019] Liao, R., Li, Y., Song, Y., Wang, S., Hamilton, W. L., Duvenaud, D., Urtasun, R., and Zemel, R. S. (2019). Efficient graph generation with graph recurrent attention networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 32, pages 4257–4267. 153, 154

[Liao et al., 2021] Liao, R., Urtasun, R., and Zemel, R. S. (2021). A pac-bayesian approach to generalization bounds for graph neural networks. In *9th International Conference on Learning Representations (ICLR)*. OpenReview.net. 102

[Lim et al., 2023] Lim, D., Robinson, J., Zhao, L., Smidt, T. E., Sra, S., Maron, H., and Jegelka, S. (2023). Sign and basis invariant networks for spectral graph representation learning. In *11th International Conference on Learning Representations (ICLR)*. OpenReview.net. 115

[Lim et al., 2018] Lim, I., Dielen, A., Campen, M., and Kobbelt, L. (2018). A simple approach to intrinsic correspondence learning on unstructured 3d meshes. *Proceedings of the European Conference on Computer Vision Workshops (ECCVW)*. 71, 73, 86, 89, 90

[Lim et al., 2014] Lim, Y., Kang, U., and Faloutsos, C. (2014). Slashburn: Graph compression and mining beyond caveman communities. *IEEE Transactions on Knowledge and Data Engineering*, 26(12):3077–3089. 132, 152

[Lin et al., 2017] Lin, H. W., Tegmark, M., and Rolnick, D. (2017). Why does deep and cheap learning work so well? *Journal of Statistical Physics*, 168:1223–1247. 37

[Linial, 1992] Linial, N. (1992). Locality in distributed graph algorithms. *SIAM Journal on Computing*, 21(1):193–201. 114

[Litany et al., 2017] Litany, O., Remez, T., Rodola, E., Bronstein, A., and Bronstein, M. (2017). Deep functional maps: Structured prediction for dense shape correspondence. In *Proceedings of the IEEE international conference on computer vision*, pages 5659–5667. 82

[Liu et al., 2021] Liu, M., Yan, K., Oztekin, B., and Ji, S. (2021). Graphebm: Molecular graph generation with energy-based models. *CoRR*, abs/2102.00546. 153

[Liu et al., 2018] Liu, Q., Allamanis, M., Brockschmidt, M., and Gaunt, A. L. (2018). Constrained graph variational autoencoders for molecule design. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 31, pages 7806–7815. 153

[Liu et al., 2019] Liu, Q., Nickel, M., and Kiela, D. (2019). Hyperbolic graph neural networks. *Advances in Neural Information Processing Systems (NeurIPS)*, 32:4869–4880. 28

[Liu et al., 2017a] Liu, W., Wen, Y., Yu, Z., Li, M., Raj, B., and Song, L. (2017a). Sphereface: Deep hypersphere embedding for face recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 212–220. 28

[Liu et al., 2017b] Liu, Z., Nalluri, S. K. M., and Stoddart, J. F. (2017b). Surveying macrocyclic chemistry: from flexible crown ethers to rigid cyclophanes. *Chemical Society Reviews*, 46(9):2459–2478. 228

[Locatello et al., 2020] Locatello, F., Weissenborn, D., Unterthiner, T., Mahendran, A., Heigold, G., Uszkoreit, J., Dosovitskiy, A., and Kipf, T. (2020). Object-centric learning with slot attention. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33. 249

[Loper et al., 2015] Loper, M., Mahmood, N., Romero, J., Pons-Moll, G., and Black, M. J. (2015). SMPL: A skinned multi-person linear model. *ACM Trans. Graphics (Proc. SIGGRAPH Asia)*, 34(6):248:1–248:16. 83, 84

[Louizos et al., 2017] Louizos, C., Ullrich, K., and Welling, M. (2017). Bayesian compression for deep learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 30, pages 3288–3298. 155

[Loukas, 2019] Loukas, A. (2019). Graph reduction with spectral and cut guarantees. *J. Mach. Learn. Res.*, 20(116):1–42. 153

[Loukas, 2020] Loukas, A. (2020). What graph neural networks cannot learn: depth vs width. In *8th International Conference on Learning Representations (ICLR)*. 39, 41, 60, 66, 80, 96, 100, 101, 105, 112, 113, 116, 161, 259

[Loukas and Vandergheynst, 2018] Loukas, A. and Vandergheynst, P. (2018). Spectrally approximating large graphs with smaller graphs. In *International Conference on Machine Learning, (ICML)*, volume 80 of *Proceedings of Machine Learning Research*, pages 3243–3252. PMLR. 153

[Lu et al., 2021] Lu, L., Jin, P., Pang, G., Zhang, Z., and Karniadakis, G. E. (2021). Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3(3):218–229. 29, 40

[Lu et al., 2017] Lu, Z., Pu, H., Wang, F., Hu, Z., and Wang, L. (2017). The expressive power of neural networks: A view from the width. In *Advances in Neural Information Processing Systems (NIPS)*, volume 30, pages 6231–6239. 39

[Luo et al., 2021] Luo, Y., Yan, K., and Ji, S. (2021). Graphdf: A discrete flow model for molecular graph generation. In *International Conference on Machine Learning, (ICML)*, volume 139 of *Proceedings of Machine Learning Research*, pages 7192–7203. PMLR. 153

[Lyle et al., 2020] Lyle, C., van der Wilk, M., Kwiatkowska, M., Gal, Y., and Bloem-Reddy, B. (2020). On the benefits of invariance in neural networks. *CoRR*, abs/2005.00178. 34

[Ma et al., 2022] Ma, Y., Liu, X., Shah, N., and Tang, J. (2022). Is homophily a necessity for graph neural networks? In *International Conference on Learning Representations*. 37

[MacKay, 2003] MacKay, D. J. (2003). *Information theory, inference and learning algorithms*. Cambridge university press. 129, 132, 133

[Mahdi et al., 2022a] Mahdi, S. S., Matthews, H., Nauwelaers, N., Vanneste, M., Gong, S., Bouritsas, G., Baynam, G. S., Hammond, P., Spritz, R., Klein, O. D., Hallgrímsson, B., Peeters, H., Bronstein, M., and Claes, P. (2022a). Multi-scale part-based syndrome classification of 3d facial images. *IEEE Access*, 10:23450–23462. 48, 161

[Mahdi et al., 2022b] Mahdi, S. S., Matthews, H., Vanneste, M., Nauwelaers, N., Gong, S., Bouritsas, G., Baynam, G. S., Hammond, P., Spritz, R., Klein, O. D., et al. (2022b). A 3d clinical face phenotype space of genetic syndromes using a triplet-based singular geometric autoencoder. *bioRxiv*, pages 2022–12. 48

[Mahdi et al., 2021a] Mahdi, S. S., Nauwelaers, N., Joris, P., Bouritsas, G., Gong, S., Bokhnyak, S., Walsh, S., Shriver, M. D., Bronstein, M., and Claes, P. (2021a). 3d facial matching by spiral convolutional metric learning and a biometric fusion-net of demographic properties. In *2020 25th International Conference on Pattern Recognition (ICPR)*, pages 1757–1764. IEEE. 48, 161

[Mahdi et al., 2021b] Mahdi, S. S., Nauwelaers, N., Joris, P., Bouritsas, G., Gong, S., Walsh, S., Shriver, M. D., Bronstein, M., and Claes, P. (2021b). Matching 3d facial shape to demographic properties by geometric metric learning: A part-based approach. *IEEE Transactions on Biometrics, Behavior, and Identity Science*. 48, 161

[Mahoney, 2000] Mahoney, M. V. (2000). Fast text compression with neural networks. In *FLAIRS conference*, pages 230–234. 130, 153

[Maiorov and Pinkus, 1999] Maiorov, V. and Pinkus, A. (1999). Lower bounds for approximation by MLP neural networks. *Neurocomputing*, 25(1-3):81–91. 39

[Makino and Uno, 2004] Makino, K. and Uno, T. (2004). New algorithms for enumerating all maximal cliques. In *Algorithm Theory - SWAT 2004, 9th Scandinavian Workshop on Algorithm Theory, Humlebaek, Denmark, July 8-10, 2004, Proceedings*, volume 3111 of *Lecture Notes in Computer Science*, pages 260–272. Springer. 220

[Maneth and Peternek, 2016] Maneth, S. and Peternek, F. (2016). Compressing graphs by grammars. In *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*, pages 109–120. IEEE. 152

[Maron et al., 2019a] Maron, H., Ben-Hamu, H., Serviansky, H., and Lipman, Y. (2019a). Provably powerful graph networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 32, pages 2153–2164. 58, 60, 101, 109, 114, 118, 119, 229

[Maron et al., 2019b] Maron, H., Ben-Hamu, H., Shamir, N., and Lipman, Y. (2019b). Invariant and equivariant graph networks. In *7th International Conference on Learning Representations, (ICLR)*.

OpenReview.net. 60, 100, 110, 114, 119, 229

[Maron et al., 2019c] Maron, H., Fetaya, E., Segol, N., and Lipman, Y. (2019c). On the universality of invariant networks. In *International Conference on Machine Learning (ICML)*, volume 97 of *Proceedings of Machine Learning Research*, pages 4363–4371. PMLR. 41, 60, 100, 101, 114

[Maron et al., 2017] Maron, H., Galun, M., Aigerman, N., Trope, M., Dym, N., Yumer, E., Kim, V. G., and Lipman, Y. (2017). Convolutional neural networks on surfaces via seamless toric covers. *ACM Trans. Graph.*, 36(4):71–1. 83

[Masci et al., 2015] Masci, J., Boscaini, D., Bronstein, M., and Vandergheynst, P. (2015). Geodesic convolutional neural networks on riemannian manifolds. In *Proceedings of the IEEE International Conference on Computer Vision Workshops (ICCVW)*, pages 37–45. 60, 78, 83

[Mateos et al., 2019] Mateos, G., Segarra, S., Marques, A. G., and Ribeiro, A. (2019). Connecting the dots: Identifying network structure via graph signal processing. *IEEE Signal Processing Magazine*, 36(3):16–43. 57

[Mateti and Deo, 1976] Mateti, P. and Deo, N. (1976). On algorithms for enumerating all circuits of a graph. *SIAM J. Comput.*, 5(1):90–99. 220

[Maturana and Scherer, 2015] Maturana, D. and Scherer, S. (2015). Voxnet: A 3d convolutional neural network for real-time object recognition. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 81

[McCreesh and Prosser, 2015] McCreesh, C. and Prosser, P. (2015). A parallel, backjumping subgraph isomorphism algorithm using supplemental graphs. In Pesant, G., editor, *Principles and Practice of Constraint Programming - 21st International Conference, CP 2015, Cork, Ireland, August 31 - September 4, 2015, Proceedings*, volume 9255 of *Lecture Notes in Computer Science*, pages 295–312. Springer. 111

[McKay, 1997] McKay, B. D. (1997). Small graphs are reconstructible. *Australasian J. Combinatorics*, 15:123–126. 107

[McKay et al., 1981] McKay, B. D. et al. (1981). Practical graph isomorphism. 137

[McKay and Piperno, 2014] McKay, B. D. and Piperno, A. (2014). Practical graph isomorphism, II. *Journal of Symbolic Computation*, 60:94–112. 137

[McPherson et al., 2001] McPherson, M., Smith-Lovin, L., and Cook, J. M. (2001). Birds of a feather: Homophily in social networks. *Annual review of sociology*, pages 415–444. 37

[Mentzer et al., 2019] Mentzer, F., Agustsson, E., Tschannen, M., Timofte, R., and Gool, L. V. (2019). Practical full resolution learned lossless image compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10629–10638. 130, 153

[Mescheder et al., 2019] Mescheder, L., Oechsle, M., Niemeyer, M., Nowozin, S., and Geiger, A. (2019). Occupancy networks: Learning 3d reconstruction in function space. *CVPR*. 82

[Mhaskar and Hahm, 1997] Mhaskar, H. N. and Hahm, N. (1997). Neural networks for functional approximation and system identification. *Neural Computation*, 9(1):143–159. 40

[Mialon et al., 2021] Mialon, G., Chen, D., Selosse, M., and Mairal, J. (2021). Graphit: Encoding graph structure in transformers. *CoRR*, abs/2106.05667. 115

[Micikevicius et al., 2018] Micikevicius, P., Narang, S., Alben, J., Diamos, G. F., Elsen, E., García, D., Ginsburg, B., Houston, M., Kuchaiev, O., Venkatesh, G., and Wu, H. (2018). Mixed precision training. In *6th International Conference on Learning Representations (ICLR)*. OpenReview.net. 256

[Mikolov et al., 2013] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems (NIPS)*, volume 26, pages 3111–3119. 92

[Milano et al., 2020] Milano, F., Loquercio, A., Rosinol, A., Scaramuzza, D., and Carlone, L. (2020). Primal-dual mesh convolutional neural networks. *Advances in Neural Information Processing Systems (NeurIPS)*, 33:952–963. 82

[Mildenhall et al., 2021] Mildenhall, B., Srinivasan, P. P., Tancik, M., Barron, J. T., Ramamoorthi, R., and Ng, R. (2021). Nerf: Representing scenes as neural radiance fields for view synthesis. *Communications of the ACM*, 65(1):99–106. 82

[Milenković and Pržulj, 2008] Milenković, T. and Pržulj, N. (2008). Uncovering biological network function via graphlet degree signatures. *Cancer informatics*, 6:257–273. 116

[Milo et al., 2002] Milo, R., Shen-Orr, S., Itzkovitz, S., Kashtan, N., Chklovskii, D., and Alon, U. (2002). Network motifs: simple building blocks of complex networks. *Science*, 298(5594):824–827. 55, 112, 116

[Mishra et al., 2018] Mishra, N., Rohaninejad, M., Chen, X., and Abbeel, P. (2018). A simple neural attentive meta-learner. In *6th International Conference on Learning Representations (ICLR)*. OpenReview.net. 30

[Monti et al., 2017] Monti, F., Boscaini, D., Masci, J., Rodola, E., Svoboda, J., and Bronstein, M. M. (2017). Geometric deep learning on graphs and manifolds using mixture model cnns. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5425–5434. 78, 83, 89, 120

[Monti et al., 2018] Monti, F., Otness, K., and Bronstein, M. M. (2018). MOTIFNET: A motif-based graph convolutional network for directed graphs. In *2018 IEEE Data Science Workshop (DSW)*, pages 225–228. IEEE. 117

[Morris et al., 2020a] Morris, C., Kriege, N. M., Bause, F., Kersting, K., Mutzel, P., and Neumann, M. (2020a). Tudataset: A collection of benchmark datasets for learning with graphs. In *ICML 2020 Workshop on Graph Representation Learning and Beyond (GRL+ 2020)*. 154, 258

[Morris et al., 2021] Morris, C., Lipman, Y., Maron, H., Rieck, B., Kriege, N. M., Grohe, M., Fey, M., and Borgwardt, K. M. (2021). Weisfeiler and leman go machine learning: The story so far. *CoRR*, abs/2112.09992. 113

[Morris et al., 2022] Morris, C., Rattan, G., Kiefer, S., and Ravanbakhsh, S. (2022). Speqnets: Sparsity-aware permutation-equivariant graph networks. In *International Conference on Machine Learning, (ICML)*, volume 162 of *Proceedings of Machine Learning Research*, pages 16017–16042. PMLR. 114

[Morris et al., 2020b] Morris, C., Rattan, G., and Mutzel, P. (2020b). Weisfeiler and leman go sparse: Towards scalable higher-order graph embeddings. *Advances in Neural Information Processing Systems (NeurIPS)*, 33:21824–21840. 114

[Morris et al., 2019] Morris, C., Ritzert, M., Fey, M., Hamilton, W. L., Lenssen, J. E., Rattan, G., and Grohe, M. (2019). Weisfeiler and leman go neural: Higher-order graph neural networks. In *AAAI Conference on Artificial Intelligence*, volume 33, pages 4602–4609. AAAI Press. 58, 96, 100, 109, 110, 114, 248

[Mowshowitz and Dehmer, 2012] Mowshowitz, A. and Dehmer, M. (2012). Entropy and the complexity of graphs revisited. *Entropy*, 14(3):559–570. 152

[Murphy et al., 2019] Murphy, R., Srinivasan, B., Rao, V., and Ribeiro, B. (2019). Relational pooling for graph representations. In *International Conference on Machine Learning (ICML)*, volume 97 of *Proceedings of Machine Learning Research*, pages 4663–4673. PMLR. 41, 100, 109, 110, 114

[Muzio et al., 2021] Muzio, G., O'Bray, L., and Borgwardt, K. (2021). Biological network analysis with deep learning. *Briefings in bioinformatics*, 22(2):1515–1530. 95

[Nakkiran et al., 2020] Nakkiran, P., Kaplun, G., Bansal, Y., Yang, T., Barak, B., and Sutskever, I. (2020). Deep double descent: Where bigger models and more data hurt. In *8th International Conference on Learning Representations (ICLR)*. OpenReview.net. 42, 138

[Naor, 1990] Naor, M. (1990). Succinct representation of general unlabeled graphs. *Discrete Applied Mathematics*, 28(3):303–307. 152

[Naor and Stockmeyer, 1993] Naor, M. and Stockmeyer, L. J. (1993). What can be computed locally? In *ACM Symposium on Theory of Computing (STOC)*, pages 184–193. ACM. 114

[Nazi et al., 2019] Nazi, A., Hang, W., Goldie, A., Ravi, S., and Mirhoseini, A. (2019). GAP: generalizable approximate graph partitioning framework. *CoRR*, abs/1903.00614. 151

[Neumann et al., 2016] Neumann, M., Garnett, R., Bauckhage, C., and Kersting, K. (2016). Propagation kernels: efficient graph kernels from propagated information. *Machine Learning*, 102(2):209–245.

119

[Neyshabur et al., 2018] Neyshabur, B., Li, Z., Bhojanapalli, S., LeCun, Y., and Srebro, N. (2018). Towards understanding the role of over-parametrization in generalization of neural networks. *CoRR*, abs/1805.12076. 138

[Ng et al., 2001] Ng, A. Y., Jordan, M. I., and Weiss, Y. (2001). On spectral clustering: Analysis and an algorithm. In *Advances in Neural Information Processing Systems (NIPS)*, volume 14, pages 849–856. MIT Press. 249

[Nguyen and Maehara, 2020] Nguyen, H. and Maehara, T. (2020). Graph homomorphism convolution. In *International Conference on Machine Learning (ICML)*, volume 119 of *Proceedings of Machine Learning Research*, pages 7306–7316. PMLR. 117

[Nickel and Kiela, 2017] Nickel, M. and Kiela, D. (2017). Poincaré embeddings for learning hierarchical representations. *Advances in Neural Information Processing Systems (NIPS)*, 30:6338–6347. 28

[Nickel and Kiela, 2018] Nickel, M. and Kiela, D. (2018). Learning continuous hierarchies in the lorentz model of hyperbolic geometry. In *International Conference on Machine Learning (ICML)*, volume 80 of *Proceedings of Machine Learning Research*, pages 3776–3785. PMLR. 28

[Niemeyer et al., 2020] Niemeyer, M., Mescheder, L., Oechsle, M., and Geiger, A. (2020). Differentiable volumetric rendering: Learning implicit 3d representations without 3d supervision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3504–3515. 82

[Nikolentzos et al., 2021] Nikolentzos, G., Siglidis, G., and Vazirgiannis, M. (2021). Graph kernels: A survey. *Journal of Artificial Intelligence Research*, 72:943–1027. 24

[Niu et al., 2020] Niu, C., Song, Y., Song, J., Zhao, S., Grover, A., and Ermon, S. (2020). Permutation invariant graph generation via score-based generative modeling. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 4474–4484. PMLR. 153

[Noé et al., 2020] Noé, F., Tkatchenko, A., Müller, K.-R., and Clementi, C. (2020). Machine learning for molecular simulation. *Annual review of physical chemistry*, 71:361–390. 95

[Nourbakhsh et al., 2015] Nourbakhsh, F., Bulo, S. R., and Pelillo, M. (2015). A matrix factorization approach to graph compression with partial information. *International Journal of Machine Learning and Cybernetics*, 6(4):523–536. 153

[Ortega et al., 2018] Ortega, A., Frossard, P., Kovačević, J., Moura, J. M., and Vandergheynst, P. (2018). Graph signal processing: Overview, challenges, and applications. *Proceedings of the IEEE*, 106(5):808–828. 23

[O'Sullivan et al., 2022] O'Sullivan, E., van de Lande, L. S., Papaioannou, A., Breakey, R. W., Jeelani, N. O., Ponniah, A., Duncan, C., Schievano, S., Khonsari, R. H., Zafeiriou, S., et al. (2022).

Convolutional mesh autoencoders for the 3-dimensional identification of fgfr-related craniosynostosis. *Scientific reports*, 12(1):2230. 161

[Papp et al., 2021] Papp, P. A., Martinkus, K., Faber, L., and Wattenhofer, R. (2021). Dropgnn: Random dropouts increase the expressiveness of graph neural networks. *Advances in Neural Information Processing Systems (NeurIPS)*, 34:21997–22009. 115

[Papp and Wattenhofer, 2022] Papp, P. A. and Wattenhofer, R. (2022). A theoretical comparison of graph neural network extensions. In *International Conference on Machine Learning*, pages 17323–17345. PMLR. 116, 162

[Paranjape et al., 2017] Paranjape, A., Benson, A. R., and Leskovec, J. (2017). Motifs in temporal networks. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining, WSDM*, pages 601–610. ACM. 112

[Park et al., 2019] Park, J. J., Florence, P., Straub, J., Newcombe, R., and Lovegrove, S. (2019). Deepsdf: Learning continuous signed distance functions for shape representation. *CVPR*. 82

[Park et al., 2021] Park, S., Yun, C., Lee, J., and Shin, J. (2021). Minimum width for universal approximation. In *9th International Conference on Learning Representations (ICLR)*. OpenReview.net. 39

[Pasco, 1976] Pasco, R. C. (1976). *Source coding algorithms for fast data compression*. PhD thesis, Stanford University CA. 133

[Paszke et al., 2019] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. (2019). Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 8024–8035. 226

[Peixoto, 2012] Peixoto, T. P. (2012). Entropy of stochastic blockmodel ensembles. *Physical Review E*, 85(5):056122. 152, 154

[Peixoto, 2013] Peixoto, T. P. (2013). Parsimonious module inference in large networks. *Physical review letters*, 110(14):148701. 145, 152, 154, 249

[Peixoto, 2014] Peixoto, T. P. (2014). The graph-tool python library. *figshare*. 246, 259

[Peixoto, 2017] Peixoto, T. P. (2017). Nonparametric bayesian inference of the microcanonical stochastic block model. *Physical Review E*, 95(1):012317. 152, 154

[Peixoto, 2019] Peixoto, T. P. (2019). Bayesian stochastic blockmodeling. *Advances in network clustering and blockmodeling*, pages 289–332. 143, 146, 152, 154, 245, 246

[Pérez et al., 2019a] Pérez, G. V., Camargo, C. Q., and Louis, A. A. (2019a). Deep learning generalizes because the parameter-function map is biased towards simple functions. In *7th International Conference on Learning Representation (ICLR)*. OpenReview.net. 37

[Pérez et al., 2019b] Pérez, J., Marinkovic, J., and Barceló, P. (2019b). On the turing completeness of modern neural network architectures. In *7th International Conference on Learning Representations (ICLR)*. OpenReview.net. 39

[Perozzi et al., 2014] Perozzi, B., Al-Rfou, R., and Skiena, S. (2014). Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710. 28

[Peshkin, 2007] Peshkin, L. (2007). Structure induction by lossless graph compression. In *2007 Data Compression Conference (DCC)*, pages 53–62. IEEE. 152

[Pinkus, 1999] Pinkus, A. (1999). Approximation theory of the mlp model in neural networks. *Acta numerica*, 8:143–195. 39

[Ploumpis et al., 2019] Ploumpis, S., Wang, H., Pears, N., Smith, W. A., and Zafeiriou, S. (2019). Combining 3d morphable models: A large scale face-and-head model. *CVPR*. 84

[Poole et al., 2022] Poole, B., Jain, A., Barron, J. T., and Mildenhall, B. (2022). Dreamfusion: Text-to-3d using 2d diffusion. *CoRR*, abs/2209.14988. 38

[Potamias et al., 2022] Potamias, R. A., Bouritsas, G., and Zafeiriou, S. (2022). Revisiting point cloud simplification: A learnable feature preserving approach. In *European Conference on Computer Vision*, pages 586–603. Springer. 48

[Potamias et al., 2020] Potamias, R. A., Zheng, J., Ploumpis, S., Bouritsas, G., Ververas, E., and Zafeiriou, S. (2020). Learning to generate customized dynamic 3d facial expressions. In *European Conference on Computer Vision*, pages 278–294. Springer. 48, 161

[Poulenard and Ovsjanikov, 2018] Poulenard, A. and Ovsjanikov, M. (2018). Multi-directional geodesic neural networks via equivariant convolution. *ACM Transactions on Graphics (TOG)*, 37(6):1–14. 83

[Pržulj, 2007] Pržulj, N. (2007). Biological network comparison using graphlet degree distribution. *Bioinformatics*, 23(2):177–183. 55, 116, 117

[Pržulj et al., 2004] Pržulj, N., Corneil, D. G., and Jurisica, I. (2004). Modeling interactome: scale-free or geometric? *Bioinformatics*, 20(18):3508–3515. 55, 112, 116

[Puny et al., 2022] Puny, O., Atzmon, M., Smith, E. J., Misra, I., Grover, A., Ben-Hamu, H., and Lipman, Y. (2022). Frame averaging for invariant and equivariant network design. In *The Tenth International Conference on Learning Representations (ICLR)*. OpenReview.net. 114

[Puny et al., 2023] Puny, O., Lim, D., Kiani, B. T., Maron, H., and Lipman, Y. (2023). Equivariant polynomials for graph neural networks. *CoRR*. 101

[Qi et al., 2017a] Qi, C. R., Su, H., Mo, K., and Guibas, L. J. (2017a). Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 652–660. 31, 81

[Qi et al., 2016] Qi, C. R., Su, H., Nießner, M., Dai, A., Yan, M., and Guibas, L. J. (2016). Volumetric and multi-view cnns for object classification on 3d data. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5648–5656. 81

[Qi et al., 2017b] Qi, C. R., Yi, L., Su, H., and Guibas, L. J. (2017b). Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *Advances in Neural Information Processing Systems (NIPS)*, 30:5099–5108. 31

[Qian et al., 2022] Qian, C., Rattan, G., Geerts, F., Niepert, M., and Morris, C. (2022). Ordered subgraph aggregation networks. In Oh, A. H., Agarwal, A., Belgrave, D., and Cho, K., editors, *Advances in Neural Information Processing Systems*. 116

[Raghavan et al., 2007] Raghavan, U. N., Albert, R., and Kumara, S. (2007). Near linear time algorithm to detect community structures in large-scale networks. *Physical review E*, 76(3):036106. 154

[Rahaman et al., 2019] Rahaman, N., Baratin, A., Arpit, D., Draxler, F., Lin, M., Hamprecht, F., Bengio, Y., and Courville, A. (2019). On the spectral bias of neural networks. In *International Conference on Machine Learning (ICML)*, volume 97 of *Proceedings of Machine Learning Research*, pages 5301–5310. PMLR. 37

[Ramesh et al., 2022] Ramesh, A., Dhariwal, P., Nichol, A., Chu, C., and Chen, M. (2022). Hierarchical text-conditional image generation with CLIP latents. *CoRR*, abs/2204.06125. 38

[Ranjan et al., 2018] Ranjan, A., Bolkart, T., Sanyal, S., and Black, M. J. (2018). Generating 3d faces using convolutional mesh autoencoders. *Proceedings of the European Conference on Computer Vision (ECCV)*. 74, 76, 85, 86, 87, 88, 91

[Rasmussen and Ghahramani, 2000] Rasmussen, C. and Ghahramani, Z. (2000). Occam's razor. *Advances in Neural Information Processing Systems (NIPS)*, 13. 42

[Ravanbakhsh, 2020] Ravanbakhsh, S. (2020). Universal equivariant multilayer perceptrons. In *International Conference on Machine Learning (ICML)*, volume 119 of *Proceedings of Machine Learning Research*, pages 7996–8006. PMLR. 114

[Ravi and Larochelle, 2017] Ravi, S. and Larochelle, H. (2017). Optimization as a model for few-shot learning. In *International conference on learning representations*. 30

[Reiser et al., 2022] Reiser, P., Neubert, M., Eberhard, A., Torresi, L., Zhou, C., Shao, C., Metni, H., van Hoesel, C., Schopmans, H., Sommer, T., et al. (2022). Graph neural networks for materials science and chemistry. *Communications Materials*, 3(1):93. 95

[Riegler et al., 2017] Riegler, G., Osman Ulusoy, A., and Geiger, A. (2017). Octnet: Learning deep 3d representations at high resolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3577–3586. 81

[Rissanen and Langdon, 1979] Rissanen, J. and Langdon, G. G. (1979). Arithmetic coding. *IBM Journal of research and development*, 23(2):149–162. 133

[Rissanen, 1976] Rissanen, J. J. (1976). Generalized kraft inequality and arithmetic coding. *IBM Journal of research and development*, 20(3):198–203. 133

[Romero et al., 2017] Romero, J., Tzionas, D., and Black, M. J. (2017). Embodied hands: Modeling and capturing hands and bodies together. *ACM Transactions on Graphics, (Proc. SIGGRAPH Asia)*. 83, 84

[Rossi and Conan-Guez, 2005] Rossi, F. and Conan-Guez, B. (2005). Functional multi-layer perceptron: a non-linear tool for functional data analysis. *Neural networks*, 18(1):45–60. 40

[Rossi et al., 2018] Rossi, R. A., Ahmed, N. K., and Koh, E. (2018). Higher-order network representation learning. In *Companion of the The Web Conference 2018 on The Web Conference 2018, WWW 2018, Lyon , France, April 23-27, 2018*, pages 3–4. ACM. 117

[Rossi and Zhou, 2018] Rossi, R. A. and Zhou, R. (2018). Graphzip: a clique-based sparse graph compression method. *Journal of Big Data*, 5(1):1–14. 152

[Rosvall and Bergstrom, 2008] Rosvall, M. and Bergstrom, C. T. (2008). Maps of random walks on complex networks reveal community structure. *Proceedings of the National Academy of Sciences*, 105(4):1118–1123. 152

[Roweis and Saul, 2000] Roweis, S. T. and Saul, L. K. (2000). Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326. 28

[Ruan et al., 2021] Ruan, Y., Ullrich, K., Severo, D., Townsend, J., Khisti, A., Doucet, A., Makhzani, A., and Maddison, C. J. (2021). Improving lossless compression rates via monte carlo bits-back coding. In *International Conference on Machine Learning, (ICML)*, volume 139 of *Proceedings of Machine Learning Research*, pages 9136–9147. PMLR. 153

[Rustamov and Guibas, 2013] Rustamov, R. and Guibas, L. J. (2013). Wavelets on graphs via deep learning. *Advances in neural information processing systems (NIPS)*, 26:998–1006. 30

[Ryabko, 1980] Ryabko, B. Y. (1980). Data compression by means of a "book stack". *Problemy Peredachi Informatsii*, 16(4):16–21. 141

[Saade et al., 2014] Saade, A., Krzakala, F., and Zdeborová, L. (2014). Spectral clustering of graphs with the bethe hessian. In *Advances in Neural Information Processing Systems (NIPS)*, volume 27, pages 406–414. 57

[Sala et al., 2018] Sala, F., De Sa, C., Gu, A., and Ré, C. (2018). Representation tradeoffs for hyperbolic embeddings. In *International Conference on Machine Learning (ICML)*, volume 80 of *Proceedings of Machine Learning Research*, pages 4457–4466. PMLR. 28

[Sandfelder et al., 2021] Sandfelder, D., Vijayan, P., and Hamilton, W. L. (2021). Ego-gnns: Exploiting ego structures in graph neural networks. In *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8523–8527. IEEE. 115

[Sandryhaila and Moura, 2013] Sandryhaila, A. and Moura, J. M. (2013). Discrete signal processing on graphs. *IEEE transactions on signal processing*, 61(7):1644–1656. 57

[Sankar et al., 2019] Sankar, A., Zhang, X., and Chang, K. C.-C. (2019). Meta-gnn: metagraph neural network for semi-supervised learning in attributed heterogeneous information networks. In *Proceedings of the 2019 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, pages 137–144. ACM. 117

[Sannai et al., 2021] Sannai, A., Imaizumi, M., and Kawano, M. (2021). Improved generalization bounds of group invariant/equivariant deep networks via quotient feature spaces. In *Uncertainty in Artificial Intelligence*, pages 771–780. PMLR. 34

[Santoro et al., 2016] Santoro, A., Bartunov, S., Botvinick, M., Wierstra, D., and Lillicrap, T. (2016). Meta-learning with memory-augmented neural networks. In *International conference on machine learning*, pages 1842–1850. PMLR. 30

[Sarajlić et al., 2016] Sarajlić, A., Malod-Dognin, N., Yaveroğlu, Ö. N., and Pržulj, N. (2016). Graphlet-based characterization of directed networks. *Scientific reports*, 6:35098. 116

[Sato, 2020] Sato, R. (2020). A survey on the expressive power of graph neural networks. *CoRR*, abs/2003.04078. 20, 109, 113

[Sato et al., 2019] Sato, R., Yamada, M., and Kashima, H. (2019). Approximation ratios of graph neural networks for combinatorial problems. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 32, pages 4083–4092. 60, 66, 79, 80, 96, 105, 113, 116

[Sato et al., 2021] Sato, R., Yamada, M., and Kashima, H. (2021). Random features strengthen graph neural networks. In *Proceedings of the 2021 SIAM International Conference on Data Mining, SDM 2021, Virtual Event, April 29 - May 1, 2021*, pages 333–341. SIAM. 80, 100, 114, 123

[Satorras et al., 2021] Satorras, V. G., Hoogeboom, E., and Welling, M. (2021). E (n) equivariant graph neural networks. In *International conference on machine learning*, pages 9323–9332. PMLR. 31, 161

[Scarselli et al., 2008] Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., and Monfardini, G. (2008). The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80. 60

[Scarselli et al., 2018] Scarselli, F., Tsoi, A. C., and Hagenbuchner, M. (2018). The vapnik–chervonenkis dimension of graph and recursive neural networks. *Neural Networks*, 108:248–259. 102

[Schäfer and Zimmermann, 2006] Schäfer, A. M. and Zimmermann, H. (2006). Recurrent neural networks are universal approximators. In *Artificial Neural Networks - ICANN 2006, 16th International*

Conference, Athens, Greece, September 10-14, 2006. Proceedings, Part I, volume 4131 of *Lecture Notes in Computer Science*, pages 632–640. Springer. 39

[Schmidhuber, 1997] Schmidhuber, J. (1997). Discovering neural nets with low kolmogorov complexity and high generalization capability. *Neural Networks*, 10(5):857–873. 37

[Schmidhuber and Heil, 1996] Schmidhuber, J. and Heil, S. (1996). Sequential neural text compression. *IEEE Transactions on Neural Networks*, 7(1):142–146. 130, 153

[Schölkopf et al., 1998] Schölkopf, B., Smola, A., and Müller, K.-R. (1998). Nonlinear component analysis as a kernel eigenvalue problem. *Neural computation*, 10(5):1299–1319. 28

[Schütt et al., 2021] Schütt, K., Unke, O., and Gastegger, M. (2021). Equivariant message passing for the prediction of tensorial properties and molecular spectra. In *International Conference on Machine Learning*, pages 9377–9388. PMLR. 161

[Segol and Lipman, 2020] Segol, N. and Lipman, Y. (2020). On universal equivariant set networks. In *8th International Conference on Learning Representations (ICLR)*. OpenReview.net. 41

[Severo et al., 2022] Severo, D., Townsend, J., Khisti, A., Makhzani, A., and Ullrich, K. (2022). Compressing multisets with large alphabets. In *2022 Data Compression Conference (DCC)*, pages 322–331. IEEE. 153

[Shannon, 1948] Shannon, C. E. (1948). A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423. 128, 130

[Sharma et al., 2016] Sharma, A., Grau, O., and Fritz, M. (2016). Vconv-dae: Deep volumetric shape learning without object labels. In *European Conference on Computer Vision*, pages 236–250. Springer. 81

[Sharp et al., 2022] Sharp, N., Attaiki, S., Crane, K., and Ovsjanikov, M. (2022). Diffusionnet: Discretization agnostic learning on surfaces. *ACM Transactions on Graphics (TOG)*, 41(3):1–16. 83

[Shaw et al., 2018] Shaw, P., Uszkoreit, J., and Vaswani, A. (2018). Self-attention with relative position representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT, Volume 2 (Short Papers)*, pages 464–468. Association for Computational Linguistics. 105

[Shawe-Taylor, 1991] Shawe-Taylor, J. (1991). Threshold network learning in the presence of equivalences. *Advances in Neural Information Processing Systems (NIPS)*, 4. 34

[Shawetaylor, 1995] Shawetaylor, J. (1995). Sample sizes for threshold networks with equivalences. *Information and Computation*, 118(1):65–72. 34

[Shen et al., 2020] Shen, W., Zhang, B., Huang, S., Wei, Z., and Zhang, Q. (2020). 3d-rotation-equivariant quaternion neural networks. In *16th European Conference on Computer Vision - (ECCV)*, volume 12365 of *Lecture Notes in Computer Science*, pages 531–547. Springer. 31, 81

[Shervashidze et al., 2011] Shervashidze, N., Schweitzer, P., Van Leeuwen, E. J., Mehlhorn, K., and Borgwardt, K. M. (2011). Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research (JMLR)*, 12:2539–2561. 119

[Shervashidze et al., 2009] Shervashidze, N., Vishwanathan, S., Petri, T., Mehlhorn, K., and Borgwardt, K. (2009). Efficient graphlet kernels for large graph comparison. In *Artificial Intelligence and Statistics (AISTATS)*, volume 5, pages 488–495. PMLR. 117, 119

[Shi et al., 2020] Shi, C., Xu, M., Zhu, Z., Zhang, W., Zhang, M., and Tang, J. (2020). Graphaf: a flow-based autoregressive model for molecular graph generation. In *8th International Conference on Learning Representations (ICLR)*. OpenReview.net. 153

[Shlomi et al., 2020] Shlomi, J., Battaglia, P., and Vlimant, J.-R. (2020). Graph neural networks in particle physics. *Machine Learning: Science and Technology*, 2(2):021001. 95

[Shuman et al., 2013] Shuman, D. I., Narang, S. K., Frossard, P., Ortega, A., and Vandergheynst, P. (2013). The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *IEEE signal processing magazine*, 30(3):83–98. 23

[Siegelmann and Sontag, 1992] Siegelmann, H. T. and Sontag, E. D. (1992). On the computational power of neural nets. In *Proceedings of the Fifth Annual ACM Conference on Computational Learning Theory, COLT 1992, Pittsburgh, PA, USA, July 27-29, 1992*, pages 440–449. ACM. 39

[Sinha et al., 2016] Sinha, A., Bai, J., and Ramani, K. (2016). Deep learning 3d shape surfaces using geometry images. In *European conference on computer vision*, pages 223–240. Springer. 83

[Sitzmann et al., 2020a] Sitzmann, V., Chan, E., Tucker, R., Snavely, N., and Wetzstein, G. (2020a). Metasdf: Meta-learning signed distance functions. *Advances in Neural Information Processing Systems (NeurIPS)*, 33:10136–10147. 82

[Sitzmann et al., 2020b] Sitzmann, V., Martel, J., Bergman, A., Lindell, D., and Wetzstein, G. (2020b). Implicit neural representations with periodic activation functions. *Advances in Neural Information Processing Systems (NeurIPS)*, 33:7462–7473. 38, 82

[Sitzmann et al., 2019] Sitzmann, V., Zollhöfer, M., and Wetzstein, G. (2019). Scene representation networks: Continuous 3d-structure-aware neural scene representations. *Advances in Neural Information Processing Systems (NeurIPS)*, 32:1119–1130. 82

[Smirnov and Solomon, 2021] Smirnov, D. and Solomon, J. (2021). Hodgenet: Learning spectral geometry on triangle meshes. *ACM Transactions on Graphics (TOG)*, 40(4):1–11. 82

[Sokolic et al., 2017] Sokolic, J., Giryes, R., Sapiro, G., and Rodrigues, M. (2017). Generalization error of invariant classifiers. In *Artificial Intelligence and Statistics*, pages 1094–1103. PMLR. 34, 98

[Solomonoff, 1960] Solomonoff, R. J. (1960). A preliminary report on a general theory of inductive inference. 130

[Solomonoff, 1964] Solomonoff, R. J. (1964). A formal theory of inductive inference. part i. *Information and control*, 7(1):1–22. 130

[Sonoda and Murata, 2017] Sonoda, S. and Murata, N. (2017). Neural network with unbounded activation functions is universal approximator. *Applied and Computational Harmonic Analysis*, 43(2):233–268. 39

[Sperduti and Starita, 1997] Sperduti, A. and Starita, A. (1997). Supervised neural networks for the classification of structures. *IEEE Transactions on Neural Networks*, 8(3):714–735. 60

[Steinruecken, 2015] Steinruecken, C. (2015). *Lossless data compression*. PhD thesis, University of Cambridge, UK. 129, 137, 262

[Steinruecken, 2016] Steinruecken, C. (2016). Compressing combinatorial objects. In *2016 Data Compression Conference (DCC)*, pages 389–396. IEEE. 262

[Stone, 1937] Stone, M. H. (1937). Applications of the theory of boolean rings to general topology. *Transactions of the American Mathematical Society*, 41(3):375–481. 39

[Stone, 1948] Stone, M. H. (1948). The generalized weierstrass approximation theorem. *Mathematics Magazine*, 21(5):237–254. 39

[Sullivan et al., 2020] Sullivan, E. O., van de Lande, L., Osolos, A., Schievano, S., Dunaway, D. J., Bulstrode, N., and Zafeiriou, S. (2020). Ear cartilage inference for reconstructive surgery with convolutional mesh autoencoders. In *Medical Image Computing and Computer Assisted Intervention–MICCAI 2020: 23rd International Conference, Lima, Peru, October 4–8, 2020, Proceedings, Part III 23*, pages 76–85. Springer. 161

[Sun et al., 2020] Sun, Z., Rooke, E., Charton, J., He, Y., Lu, J., and Baek, S. (2020). Zernet: Convolutional neural networks on arbitrary surfaces via zernike local tangent space estimation. In *Computer Graphics Forum*, volume 39, pages 204–216. Wiley Online Library. 78, 83

[Sutton, 2019] Sutton, R. (2019). The bitter lesson. *Incomplete Ideas (blog)*, 13(1). 38

[Takeaki, 2012] Takeaki, U. (2012). Implementation issues of clique enumeration algorithm. *Special issue: Theoretical computer science and discrete mathematics, Progress in Informatics*, 9:25–30. 220

[Tam et al., 2013] Tam, G. K. L., Cheng, Z., Lai, Y., Langbein, F. C., Liu, Y., Marshall, A. D., Martin, R. R., Sun, X., and Rosin, P. L. (2013). Registration of 3d point clouds and meshes: A survey from rigid to nonrigid. *IEEE Trans. Vis. Comput. Graph.*, 19(7):1199–1217. 83

[Tan et al., 2019] Tan, Q., Liu, N., and Hu, X. (2019). Deep representation learning for social network analysis. *Frontiers in big Data*, 2:2. 95

[Tancik et al., 2021] Tancik, M., Mildenhall, B., Wang, T., Schmidt, D., Srinivasan, P. P., Barron, J. T., and Ng, R. (2021). Learned initializations for optimizing coordinate-based neural representations. In

Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pages 2846–2855. 82

[Tancik et al., 2020] Tancik, M., Srinivasan, P., Mildenhall, B., Fridovich-Keil, S., Raghavan, N., Singhal, U., Ramamoorthi, R., Barron, J., and Ng, R. (2020). Fourier features let networks learn high frequency functions in low dimensional domains. *Advances in Neural Information Processing Systems (NeurIPS)*, 33:7537–7547. 38, 82

[Tenenbaum et al., 2000] Tenenbaum, J. B., Silva, V. d., and Langford, J. C. (2000). A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323. 28

[Thiede et al., 2021] Thiede, E., Zhou, W., and Kondor, R. (2021). Autobahn: Automorphism-based graph neural nets. *Advances in Neural Information Processing Systems (NeurIPS)*, 34:29922–29934. 115

[Thomas et al., 2019] Thomas, H., Qi, C. R., Deschaud, J.-E., Marcotegui, B., Goulette, F., and Guibas, L. J. (2019). Kpconv: Flexible and deformable convolution for point clouds. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 6411–6420. 31, 81

[Thomas et al., 2018] Thomas, N., Smidt, T. E., Kearnes, S., Yang, L., Li, L., Kohlhoff, K., and Riley, P. (2018). Tensor field networks: Rotation- and translation-equivariant neural networks for 3d point clouds. *CoRR*, abs/1802.08219. 31, 81

[Townsend et al., 2019] Townsend, J., Bird, T., and Barber, D. (2019). Practical lossless compression with latent variables using bits back coding. In *7th International Conference on Learning Representations (ICLR)*. 153

[Townsend et al., 2020] Townsend, J., Bird, T., Kunze, J., and Barber, D. (2020). Hilloc: lossless image compression with hierarchical latent variable models. In *8th International Conference on Learning Representations (ICLR)*. OpenReview.net. 153

[Tran et al., 2019] Tran, D., Vafa, K., Agrawal, K. K., Dinh, L., and Poole, B. (2019). Discrete flows: Invertible generative models of discrete data. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 14692–14701. 153

[Trucco, 1956a] Trucco, E. (1956a). A note on the information content of graphs. *The bulletin of mathematical biophysics*, 18(2):129–135. 152

[Trucco, 1956b] Trucco, E. (1956b). On the information content of graphs: compound symbols; different states for each point. *The bulletin of mathematical biophysics*, 18(3):237–253. 152

[Turán, 1984] Turán, G. (1984). On the succinct representation of graphs. *Discrete Applied Mathematics*, 8(3):289–294. 152

[Ulam, 1960] Ulam, S. M. (1960). *A collection of mathematical problems*, volume 8. Interscience Publishers. 107

[Ullmann, 1976] Ullmann, J. R. (1976). An algorithm for subgraph isomorphism. *J. ACM*, 23(1):31–42. 111

[van den Berg et al., 2021] van den Berg, R., Gritsenko, A. A., Dehghani, M., Sønderby, C. K., and Salimans, T. (2021). IDF++: analyzing and improving integer discrete flows for lossless compression. In *9th International Conference on Learning Representations (ICLR)*. OpenReview.net. 153

[van den Oord et al., 2016] van den Oord, A., Kalchbrenner, N., and Kavukcuoglu, K. (2016). Pixel recurrent neural networks. In *International Conference on Machine Learning, (ICML)*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 1747–1756. JMLR.org. 130, 153

[Velickovic et al., 2018] Velickovic, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., and Bengio, Y. (2018). Graph attention networks. In *6th International Conference on Learning Representations (ICLR)*. OpenReview.net. 28, 79, 89, 120

[Verma et al., 2018] Verma, N., Boyer, E., and Verbeek, J. (2018). Feastnet: Feature-steered graph convolutions for 3d shape analysis. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition(CVPR)*, pages 2598–2606. 79, 83, 89

[Vigna and Boldi, 2004] Vigna, P. B. S. and Boldi, P. (2004). The webgraph framework i: compression techniques. In *Proceedings of the 13th International Conference on World Wide Web*, pages 595–602. 132, 152

[Vignac et al., 2022] Vignac, C., Krawczuk, I., Siraudin, A., Wang, B., Cevher, V., and Frossard, P. (2022). Digress: Discrete denoising diffusion for graph generation. *CoRR*, abs/2209.14734. 162

[Vignac et al., 2020] Vignac, C., Loukas, A., and Frossard, P. (2020). Building powerful and equivariant graph neural networks with structural message-passing. *Advances in Neural Information Processing Systems (NeurIPS)*, 33:14143–14155. 109, 114, 120, 249

[Villar et al., 2021] Villar, S., Hogg, D. W., Storey-Fisher, K., Yao, W., and Blum-Smith, B. (2021). Scalars are universal: Equivariant machine learning, structured like classical physics. *Advances in Neural Information Processing Systems (NeurIPS)*, 34:28848–28863. 41

[Vincent-Cuaz et al., 2021] Vincent-Cuaz, C., Vayer, T., Flamary, R., Corneli, M., and Courty, N. (2021). Online graph dictionary learning. In *International Conference on Machine Learning, (ICML)*, volume 139 of *Proceedings of Machine Learning Research*, pages 10564–10574. PMLR. 153

[Wallace, 1990] Wallace, C. S. (1990). Classification by minimum-message-length inference. In *International Conference on Computing and Information*, pages 72–81. Springer. 153

[Wang et al., 2022] Wang, H., Yin, H., Zhang, M., and Li, P. (2022). Equivariant and stable positional encoding for more powerful graph neural networks. In *10th International Conference on Learning*

Representations (ICLR). OpenReview.net. 115

[Wang et al., 2019] Wang, Y., Sun, Y., Liu, Z., Sarma, S. E., Bronstein, M. M., and Solomon, J. M. (2019). Dynamic graph cnn for learning on point clouds. *Acm Transactions On Graphics (tog)*, 38(5):1–12. 81

[Wasa et al., 2014] Wasa, K., Arimura, H., and Uno, T. (2014). Efficient enumeration of induced subtrees in a k-degenerate graph. In *Algorithms and Computation - 25th International Symposium, ISAAC 2014, Jeonju, Korea, December 15-17, 2014, Proceedings*, volume 8889 of *Lecture Notes in Computer Science*, pages 94–102. Springer. 221

[Weierstrass, 1885] Weierstrass, K. (1885). Über die analytische darstellbarkeit sogenannter willkürlicher functionen einer reellen veränderlichen. *Sitzungsberichte der Königlich Preußischen Akademie der Wissenschaften zu Berlin*, 2:633–639. 39

[Weiler et al., 2021] Weiler, M., Forré, P., Verlinde, E., and Welling, M. (2021). Coordinate independent convolutional networks - isometry and gauge equivariant convolutions on riemannian manifolds. *CoRR*, abs/2106.06020. 78, 83

[Weininger et al., 1989] Weininger, D., Weininger, A., and Weininger, J. L. (1989). Smiles. 2. algorithm for generation of unique smiles notation. *Journal of chemical information and computer sciences*, 29(2):97–101. 159

[Weisfeiler and Leman, 1968] Weisfeiler, B. and Leman, A. (1968). The reduction of a graph to canonical form and the algebra which appears therein. *NTI, Series 2*, 9:12–16. English translation is available at https://www.iti.zcu.cz/wl2018/pdf/wl_paper_translation.pdf. 58, 136, 261

[Welch, 1984] Welch, T. A. (1984). A technique for high-performance data compression. *Computer*, 17(06):8–19. 129, 140

[Wernicke, 2005] Wernicke, S. (2005). A faster algorithm for detecting network motifs. In *Algorithms in Bioinformatics, 5th International Workshop, WABI*, volume 3692 of *Lecture Notes in Computer Science*, pages 165–177. Springer. 111

[Wernicke, 2006] Wernicke, S. (2006). Efficient detection of network motifs. *IEEE ACM Transactions on Computational Biology and Bioinformatics*, 3(4):347–359. 111

[Wernicke and Rasche, 2006] Wernicke, S. and Rasche, F. (2006). FANMOD: a tool for fast network motif detection. *Bioinformatics*, 22(9):1152–1153. 111

[Whitney, 1932] Whitney, H. (1932). Congruent graphs and the connectivity of graphs. *American Journal of Mathematics*, 54(1):150–168. 55

[Wieder et al., 2020] Wieder, O., Kohlbacher, S., Kuenemann, M., Garon, A., Ducrot, P., Seidel, T., and Langer, T. (2020). A compact review of molecular property prediction with graph neural networks. *Drug Discovery Today: Technologies*, 37:1–12. 95

[Wiersma et al., 2020] Wiersma, R., Eisemann, E., and Hildebrandt, K. (2020). Cnns on surfaces using rotation-equivariant features. *ACM Transactions on Graphics (ToG)*, 39(4):92–1. 83

[Wilder et al., 2019] Wilder, B., Ewing, E., Dilkina, B., and Tambe, M. (2019). End to end learning and optimization on graphs. In *Advances in Neural Information Processing Systems(NeurIPS)*, volume 32, pages 4674–4685. 151, 249

[Williams, 1992] Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256. 151

[Wilson et al., 2014] Wilson, R. C., Hancock, E. R., Pekalska, E., and Duin, R. P. (2014). Spherical and hyperbolic embeddings of data. *IEEE transactions on pattern analysis and machine intelligence*, 36(11):2255–2269. 28

[Witten et al., 1987] Witten, I. H., Neal, R. M., and Cleary, J. G. (1987). Arithmetic coding for data compression. *Communications of the ACM*, 30(6):520–540. 133, 261

[Wu et al., 2016] Wu, J., Zhang, C., Xue, T., Freeman, B., and Tenenbaum, J. (2016). Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. *Advances in Neural Information Processing Systems (NIPS)*, 29:82–90. 81

[Wu et al., 2019] Wu, W., Qi, Z., and Fuxin, L. (2019). Pointconv: Deep convolutional networks on 3d point clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9621–9630. 81

[Wu et al., 2015] Wu, Z., Song, S., Khosla, A., Yu, F., Zhang, L., Tang, X., and Xiao, J. (2015). 3d shapenets: A deep representation for volumetric shapes. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1912–1920. 81

[Xu et al., 2019] Xu, K., Hu, W., Leskovec, J., and Jegelka, S. (2019). How powerful are graph neural networks? In *7th International Conference on Learning Representations (ICLR)*. OpenReview.net. 58, 60, 76, 96, 100, 106, 110, 114, 118, 119, 120, 121, 216, 217, 227, 228, 229, 232, 248

[Xu et al., 2018a] Xu, K., Li, C., Tian, Y., Sonobe, T., Kawarabayashi, K., and Jegelka, S. (2018a). Representation learning on graphs with jumping knowledge networks. In *International Conference on Machine Learning, (ICML)*, volume 80 of *Proceedings of Machine Learning Research*, pages 5449–5458. PMLR. 228

[Xu et al., 2020] Xu, K., Li, J., Zhang, M., Du, S. S., Kawarabayashi, K., and Jegelka, S. (2020). What can neural networks reason about? In *8th International Conference on Learning Representations (ICLR)*. OpenReview.net. 102

[Xu et al., 2018b] Xu, Y., Fan, T., Xu, M., Zeng, L., and Qiao, Y. (2018b). Spidercnn: Deep learning on point sets with parameterized convolutional filters. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 87–102. 81

[Yanardag and Vishwanathan, 2015] Yanardag, P. and Vishwanathan, S. (2015). Deep graph kernels. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1365–1374. 154, 259

[Yang et al., 2019] Yang, C., Zhuang, P., Shi, W., Luu, A., and Li, P. (2019). Conditional structure generation through graph variational generative adversarial nets. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 32, pages 1338–1349. 153

[Yang et al., 2018] Yang, Y., Feng, C., Shen, Y., and Tian, D. (2018). Foldingnet: Point cloud auto-encoder via deep grid deformation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 206–215. 81

[Yarotsky, 2017] Yarotsky, D. (2017). Error bounds for approximations with deep relu networks. *Neural Networks*, 94:103–114. 39

[Yarotsky, 2022] Yarotsky, D. (2022). Universal approximations of invariant maps by neural networks. *Constructive Approximation*, 55(1):407–474. 40, 41

[Yi et al., 2017] Yi, L., Su, H., Guo, X., and Guibas, L. J. (2017). Syncspeccnn: Synchronized spectral cnn for 3d shape segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 82

[Ying et al., 2021] Ying, C., Cai, T., Luo, S., Zheng, S., Ke, G., He, D., Shen, Y., and Liu, T.-Y. (2021). Do transformers really perform badly for graph representation? *Advances in Neural Information Processing Systems (NeurIPS)*, 34:28877–28888. 115

[Ying et al., 2018] Ying, R., He, R., Chen, K., Eksombatchai, P., Hamilton, W. L., and Leskovec, J. (2018). Graph convolutional neural networks for web-scale recommender systems. In *ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD)*, pages 974–983. ACM. 60

[Ying et al., 2020a] Ying, R., Lou, Z., You, J., Wen, C., Canedo, A., and Leskovec, J. (2020a). Neural subgraph matching. *CoRR*, abs/2007.03092. 111

[Ying et al., 2020b] Ying, R., Wang, A. Z., You, J., and Leskovec, J. (2020b). Frequent subgraph mining by walking in order embedding space. In *International Conference on Machine Learning Workshops (ICMLW)*. 111

[You et al., 2021] You, J., Gomes-Selman, J. M., Ying, R., and Leskovec, J. (2021). Identity-aware graph neural networks. In *AAAI conference on artificial intelligence*, volume 35, pages 10737–10745. AAAI Press. 115

[You et al., 2018] You, J., Ying, R., Ren, X., Hamilton, W. L., and Leskovec, J. (2018). Graphrnn: Generating realistic graphs with deep auto-regressive models. In *International Conference on Machine Learning, (ICML)*, volume 80 of *Proceedings of Machine Learning Research*, pages 5694–5703. PMLR. 153, 154

[Yu et al., 2021] Yu, A., Ye, V., Tancik, M., and Kanazawa, A. (2021). pixelnerf: Neural radiance fields from one or few images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4578–4587. 82

[Yuille et al., 1992] Yuille, A. L., Hallinan, P. W., and Cohen, D. S. (1992). Feature extraction from faces using deformable templates. *International journal of computer vision*, 8(2):99–111. 84

[Yun et al., 2020] Yun, C., Bhojanapalli, S., Rawat, A. S., Reddi, S. J., and Kumar, S. (2020). Are transformers universal approximators of sequence-to-sequence functions? In *8th International Conference on Learning Representations (ICLR)*. OpenReview.net. 39

[Zaheer et al., 2017] Zaheer, M., Kottur, S., Ravanbakhsh, S., Poczos, B., Salakhutdinov, R. R., and Smola, A. J. (2017). Deep sets. *Advances in Neural Information Processing Systems (NIPS)*, 30:3391–3401. 31, 61, 124, 232, 247

[Zhang et al., 2023] Zhang, B., Luo, S., Wang, L., and He, D. (2023). Rethinking the expressive power of GNNs via graph biconnectivity. In *11th International Conference on Learning Representations (ICLR)*. OpenReview.net. 101, 116

[Zhang et al., 2018] Zhang, M., Cui, Z., Neumann, M., and Chen, Y. (2018). An end-to-end deep learning architecture for graph classification. In *AAAI Conference on Artificial Intelligence*, volume 32, pages 4438–4445. AAAI Press. 119

[Zhang and Li, 2021] Zhang, M. and Li, P. (2021). Nested graph neural networks. *Advances in Neural Information Processing Systems (NeurIPS)*, 34:15734–15747. 115

[Zhang et al., 2012] Zhang, X., Dong, X., and Frossard, P. (2012). Learning of structured graph dictionaries. In *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3373–3376. IEEE. 153

[Zhao et al., 2022] Zhao, L., Jin, W., Akoglu, L., and Shah, N. (2022). From stars to subgraphs: Uplifting any GNN with local structure awareness. In *10th International Conference on Learning Representation (ICLR)*. OpenReview.net. 115

[Zheng et al., 2022] Zheng, X., Liu, Y., Pan, S., Zhang, M., Jin, D., and Yu, P. S. (2022). Graph neural networks for graphs with heterophily: A survey. *CoRR*, abs/2202.07082. 38

[Zhou, 2020] Zhou, D.-X. (2020). Universality of deep convolutional neural networks. *Applied and computational harmonic analysis*, 48(2):787–794. 40

[Zhou et al., 2020] Zhou, Y., Wu, C., Li, Z., Cao, C., Ye, Y., Saragih, J., Li, H., and Sheikh, Y. (2020). Fully convolutional mesh autoencoder using efficient spatially varying kernels. *Advances in Neural Information Processing Systems (NeurIPS)*, 33:9251–9262. 78, 83

[Zhu et al., 2020] Zhu, J., Yan, Y., Zhao, L., Heimann, M., Akoglu, L., and Koutra, D. (2020). Beyond homophily in graph neural networks: Current limitations and effective designs. *Advances in Neural*

Information Processing Systems (NeurIPS), 33:7793–7804. 37

[Zhu et al., 2021] Zhu, S., An, B., and Huang, F. (2021). Understanding the generalization benefit of model invariance from a data perspective. *Advances in Neural Information Processing Systems (NeurIPS)*, 34:4328–4341. 34

[Ziv and Lempel, 1977] Ziv, J. and Lempel, A. (1977). A universal algorithm for sequential data compression. *IEEE Transactions on information theory*, 23(3):337–343. 129, 140

[Ziv and Lempel, 1978] Ziv, J. and Lempel, A. (1978). Compression of individual sequences via variable-rate coding. *IEEE transactions on Information Theory*, 24(5):530–536. 129, 140

[Zopf, 2022] Zopf, M. (2022). 1-wl expressiveness is (almost) all you need. In *International Joint Conference on Neural Networks, IJCNN 2022, Padua, Italy, July 18-23, 2022*, pages 1–8. IEEE. 102

[Zuffi et al., 2017] Zuffi, S., Kanazawa, A., Jacobs, D. W., and Black, M. J. (2017). 3d menagerie: Modeling the 3d shape and pose of animals. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6365–6373. 83

A

Appendix to chapter 3

A.1 Implementation Details

Open surfaces. Looking more closely into the algorithmic procedure of section 3.3.2, one can observe that the inductive condition in Step 3 might not be satisfied for any of the remaining vertices of the current ring. This happens for *open surfaces/meshes with boundaries* (i.e. when the l.h.s. of Eq. (3.4) is satisfied with equality). To address this, whenever the inductive condition is not satisfied in the ring $\mathcal{R}_{\ell+1}$, i.e. when the spiral intersects the boundary, we continue the traversal from the remaining neighbour of the initial ring vertex i_{r_i+1} (e.g. the vertex i_1 in the first ring) that we have not visited so far. Then we proceed again by using the inductive rule until the entire ring is visited. To ensure that the spiral orientation will be consistent, the vertices added after the boundary intersection are placed into τ_i in reverse order (and optionally a dummy vertex that indicates zero-padding is inserted right after the vertex on which the intersection happened). For the same reason, in Step 4, it is possible that the first and last vertex of a ring do not have an additional common neighbour. In this case, we look for a common neighbour between the first and the second vertex or the second and the third and so on and so forth.

Network architectures and hyperparameters. We denote with $\mathrm{SC}_{d,\rho,M}$ a spiral convolution of width d using ρ-hop spiral shift operators computed on the mesh M, followed by an element-wise non-linearity $\sigma(\cdot)$. Additionally, we denote with $\mathrm{DS}_{r,M}$ and $\mathrm{US}_{r,M}$ a downsampling (pooling) and an upsampling (unpooling) layer, where the coarsening is applied on mesh M, which reduces or increases, respectively, the mesh size by a factor r. Finally, we denote with FC_d a

fully connected linear layer of width d, and with $|\mathcal{V}_{\text{last}}|$ the number of vertices after the last downsampling layer. The simple variant for the COMA and DFAUST datasets is the following:

$$h_{\text{ENC}}(\mathbf{u}) = \text{FC}_{d_{\text{out}}} \circ \text{DS}_{4,M_4} \circ \text{SC}_{32,1,M_4} \circ \text{DS}_{4,M_3} \circ \text{SC}_{16,1,M_3} \circ \text{DS}_{4,M_2} \circ \text{SC}_{16,1,M_2} \circ \text{DS}_{4,M_1} \circ \text{SC}_{16,1,M_1}(\mathbf{u})$$

$$h_{\text{DEC}}(\mathbf{z}) = \text{SC}_{d_{\text{in}},1,M_1} \circ \text{US}_{4,M_1} \circ \text{SC}_{16,1,M_2} \circ \text{US}_{4,M_2} \circ \text{SC}_{16,1,M_3} \circ \text{US}_{4,M_3} \circ \text{SC}_{32,1,M_4} \circ \text{US}_{4,M_4} \circ \text{FC}_{|\mathcal{V}_{\text{last}}| \times 32}(\mathbf{z}),$$

where $M_1, \ldots, M_4$ are the original and coarsened meshes, d_{in} is the dimension of the input signal and d_{out} is the dimension of the latent vector. For Mein3D, due to the high vertex count, we modified the COMA architecture in order to reduce the parameter count of the simple Neural3DMM by adding an extra convolution and an extra downsampling/upsampling layer in the encoder and the decoder respectively (encoder widths from 1st to last layer: $(8, 16, 16, 32, 32)$, decoder: mirror of the encoder). The larger Neural3DMM follows the above architecture, but with increased widths (encoder: COMA: $(64, 64, 64, 128)$, DFAUST: $(16, 32, 64, 128)$, Mein3D: $(8, 16, 32, 64, 128)$, decoder: mirror of the encoder).[1] All of our activation functions were ELUs [Clevert et al., 2016]. Our learning rate was 10^{-3} with a decay of 0.99 after each epoch, and our weight decay was 5×10^{-5}. All models were trained for 300 epochs.

GAN architecture. For the GAN and the PCA-based morphable model that we used for comparison, we used a latent space dimension $d_{\text{out}} = 256$ (for PCA this corresponds to the principal components explaining 99.4% of the variance of the data). The generator and discriminator networks had the following architectures:

$$h_{\text{DISC}}(\mathbf{u}) = \text{FC}_1 \circ \text{DS}_{4,M_5} \circ \text{SC}_{256,1,M_4} \circ \text{DS}_{4,M_4} \circ \text{SC}_{128,1,M_4} \circ \text{DS}_{4,M_3} \circ \text{SC}_{128,1,M_3} \circ \text{DS}_{4,M_2} \circ \text{SC}_{128,1,M_2} \circ \text{DS}_{4,M_1} \circ \text{SC}_{64,1,M_1}(\mathbf{u})$$

$$h_{\text{GEN}}(\mathbf{z}) = \text{SC}_{d_{\text{in}},1,M_1} \circ \text{US}_{4,M_1} \circ \text{SC}_{64,1,M_2} \circ \text{US}_{4,M_2} \circ \text{SC}_{128,1,M_2} \circ \text{US}_{4,M_3} \circ \text{SC}_{128,1,M_3} \circ \text{US}_{4,M_4} \circ \text{SC}_{128,1,M_4} \circ \text{US}_{4,M_5} \circ \text{FC}_{|\mathcal{V}_{\text{last}}| \times 256}(\mathbf{z}),$$

[1] In DFAUST, we also experimented with dilated convolutions using $\rho = 2$, and dilation ratio $r = 2$ that slightly improved the results

A.2 Additional quantitative and qualitative results

Generalisation error. In Tables A.1, A.3, and A.2, we report the exact results and parameter counts of the methods of Fig. 3.5.

Shape analogies. In Figures A.1 and A.2a, we show additional shape analogies (body pose and facial expression transfer in the DFAUST and COMA respectively) similar to Fig. 3.7b.

Synthetic faces generated by PCA. Finally, in Figure A.2b we included sampled synthetic faces generated with PCA in order to visually compare with those generated by the GAN. As already mentioned in the main text, although PCA tends to produce smooth and noise-free surfaces, the synthetic faces look artificial, due to the absence of high-frequency detail.

Table A.1: COMA dataset comparison

Latent Size	Explained Variance	Model	# of Params	Generalization (mm)
8	83.1 %	PCA	120k	1.636
	n/a	COMA	28k	0.885
	n/a	Neural3DMM (small)	38k	0.801
	n/a	Neural3DMM (ours)	381k	**0.472**
16	94.6 %	PCA	241k	0.825
	n/a	COMA	39k	0.751
	n/a	Neural3DMM (small)	48k	0.635
	n/a	Neural3DMM (ours)	425k	**0.377**
64	99.1 %	PCA	965k	0.284
	n/a	COMA	100k	0.611
	n/a	Neural3DMM (small)	113k	0.449
	n/a	Neural3DMM (ours)	682k	**0.260**

Table A.2: DFAUST dataset comparison

Latent Size	Explained Variance	Model	# of Params	Generalization (mm)
8	84.8 %	PCA	165k	59.30
	n/a	COMA	32k	28.09
	n/a	Neural3DMM (small)	41k	28.69
	n/a	Neural3DMM (ours)	274k	**19,77**
16	96.1 %	PCA	330k	32.16
	n/a	COMA	46k	17.03
	n/a	Neural3DMM (small)	56k	15.30
	n/a	Neural3DMM (ours)	332k	**11.20**
64	99.8 %	PCA	1.32M	5.28
	n/a	COMA	129k	8.98
	n/a	Neural3DMM (small)	142k	5.51
	n/a	Neural3DMM (ours)	676k	**4.29**

Table A.3: Mein3D dataset comparison

Latent Size	Explained Variance	Model	# of Params	Generalization (mm)
16	86.0 %	PCA	1.36M	0.739
	n/a	COMA	53k	0.812
	n/a	Neural3DMM (small)	66k	0.718
	n/a	Neural3DMM (ours)	320k	**0.711**
32	93.0 %	PCA	2.79M	0.525
	n/a	COMA	82k	0.616
	n/a	Neural3DMM (small)	95k	0.518
	n/a	Neural3DMM (ours)	438k	**0.502**
128	98.5 %	PCA	10.91M	0.235
	n/a	COMA	254k	0.400
	n/a	Neural3DMM (small)	274k	0.269
	n/a	Neural3DMM (ours)	1.15M	**0.229**

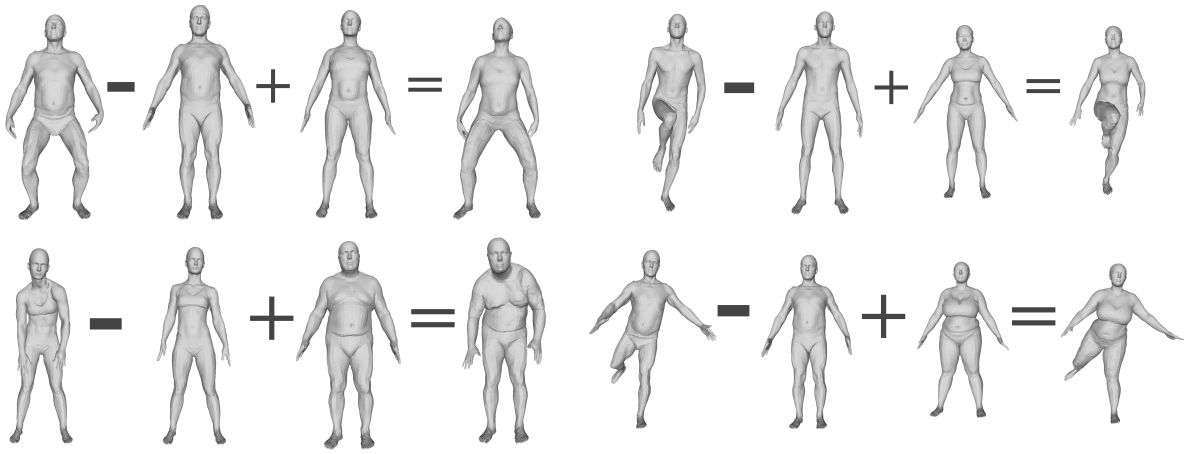
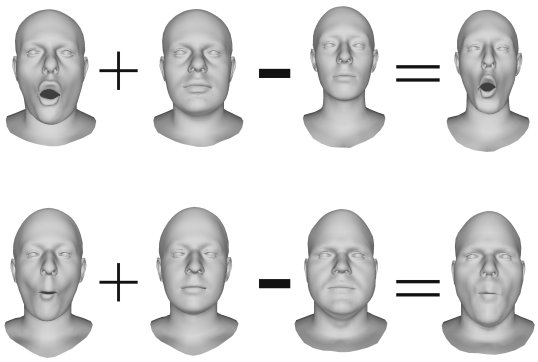
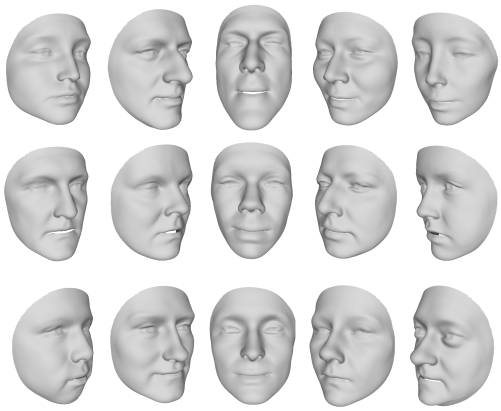


Figure A.1: Pose transfer examples through latent space analogies in the DFAUST dataset



(a) Expression transfer examples via latent space analogies in the COMA dataset



(b) Faces sampled from the PCA-based 3DMM

Figure A.2: Spiral ordering on a mesh and an image patch

B

Appendix to chapter 4

B.1 Omitted proofs

B.1.1 GSN is permutation equivariant

Proposition B.1. *GSN layers are GI-equivariant.*

Proof. Let $G = (\mathcal{V}, \mathcal{E}, \mathbf{u}_\mathcal{V}, \mathbf{u}_\mathcal{E})$ be a graph, where $\mathbf{u}_\mathcal{V} \in \mathbb{R}^{|\mathcal{V}|\times d_v}, \mathbf{u}_\mathcal{E} \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|\times d_e}$ are the optional vertex and edge attributes/signals. The proof trivially follows from the permutation equivariance of MPNN layers. In particular, it is easy to see that a GSN layer can be written as an MPNN by simply modifying its input. In particular, let h_{MPNN} be a single layer ($t = 1$) as defined in Eq. (2.11) without a READ function and without using any weighting functions $\mathbf{w}_\mathcal{V}, \mathbf{w}_\mathcal{E}$. Then, by observing Eq. (2.11) one can see that $h_{\text{GSN}}(G) = h_{\text{MPNN}}\big(h_{\text{count}}(G)\big)$, where $h_{\text{count}}(G) = (\mathcal{V}, \mathcal{E}, [\mathbf{u}_\mathcal{V}; \mathbf{w}_\mathcal{V}], [\mathbf{u}_\mathcal{E}; \mathbf{w}_\mathcal{E}])$,and $[\mathbf{u}_\mathcal{V}; \mathbf{w}_\mathcal{V}] \in \mathbb{R}^{d_v+d_v^w}$, $[\mathbf{u}_\mathcal{E}; \mathbf{w}_\mathcal{E}] \in \mathbb{R}^{d_e+d_e^w}$

Define $\text{Iso}(G_1, G_2)$ as the set of adjacency preserving bijections t*hat also preserve the vertex and edge attributes* (i.e. now G_1, G_2 are isomorphic iff there exists a permutation of the vertices of G_1 that when simultaneously applied to its adjacency matrix, to its vertex attribute matrix and to its edge attribute matrix, yields the corresponding ones of G_2). We know that MPNN layers are permutation equivariant: if $f \in \text{Iso}(G_1, G_2)$, then $h_{\text{MPNN}}\big(G_1\big)\big(f(i)\big) = h_{\text{MPNN}}\big(G_2\big)\big(i\big)$.

Now, it is straightforward to see that the functions in Eq. (4.3) and (4.4) that count orbit appearances are equivariant to isomorphism, i.e. if $f \in \text{Iso}(G_1, G_2)$ then $\texttt{vcount}(G_1, \alpha)(f(i), v) = \texttt{vcount}(G_2, \alpha)(i, v)$ and similarly for $\texttt{ecount}(\cdot, \cdot)$. Therefore, if $f \in \text{Iso}(G_1, G_2)$, then $f \in$

$\mathrm{Iso}(h_{\mathrm{count}}(G_1), h_{\mathrm{count}}(G_2))$. Thus, if $f \in \mathrm{Iso}(G_1, G_2)$, then

$$h_{\mathrm{GSN}}\big(G_1\big)\big(f(i)\big) = h_{\mathrm{MPNN}}\big(h_{\mathrm{count}}(G_1)\big)\big(f(i)\big) = h_{\mathrm{MPNN}}\big(h_{\mathrm{count}}(G_2)\big)\big(i\big) = h_{\mathrm{GSN}}\big(G_2\big)\big(i\big) \quad \text{(B.1)}$$

Overall, a GSN network is GI-equivariant as a composition of GI-equivariant functions, or GI-invariant when composed with a permutation invariant layer (READ) at the end.

□

B.1.2 Proof of Theorem 4.2: GSN is at least as powerful as the 1-WL test

Proof. The proof that GSN is at least as expressive as the 1-WL test follows directly from the proof of Theorem 3 in [Xu et al., 2019]. We repurpose it for our needs and re-write it here for completeness. We will consider vertex-labelled graphs, since traditionally the 1-WL test does not take into account edge labels.[1] It suffices to show that there exists a function in the GSN hypothesis class which is at least as expressive as WL. Consider the general formulation of Eq. (2.11) and assume a hypothesis h_{GSN} where the update functions UP_t and the aggregation functions AGGR_t of each layer t are *injective* w.r.t. all their arguments We will show that this hypothesis is at least as powerful as the 1-WL.

We can rephrase the statement as follows: *If GSN deems two graphs G_1, G_2 as isomorphic, then also 1-WL deems them isomorphic.* Given that the graph-level representation is extracted by a readout function READ that receives the multiset of the vertex-wise colours/representations in its input, then it suffices to show that if for the two graphs, the multiset of the vertex colours that GSN infers is the same, then also 1-WL will infer the same multiset for the two graphs. Since equal multisets contain the same elements with the same multiplicities, it further suffices to show that if two vertices i, j are assigned the same GSN hidden representations $\mathbf{x}_t(i) = \mathbf{x}_t(j)$ at any iteration t, then they will be also assigned the same colours $c_t(i) = c_t(j)$ by 1-WL. We prove the above by induction (similarly to [Xu et al., 2019]) for the selected function h_{GSN}.

- For $t = 0$ the statement holds since the initial vertex features are the same for both GSN and 1-WL (either $\mathbf{u}_{\mathcal{V}}$ or a constant value for graphs without attributes), i.e. $\mathbf{x}_0(i) = c_0(i)$.

[1] If one considers a simple 1-WL extension that concatenates edge labels to neighbour colours, then the same proof applies.

- Suppose the statement holds for $t-1$, i.e. $\mathbf{x}_{t-1}(i) = \mathbf{x}_{t-1}(j) \Rightarrow c_{t-1}(i) = c_{t-1}(j)$. Then we show that it also holds for t. Recall that $\mathbf{x}_t(i) = \mathrm{UP}_t\big(\mathbf{x}_{t-1}(i), \mathbf{w}_{\mathcal{V}}(i), \mathbf{m}_t(i)\big)$. If $\mathbf{x}_t(i) = \mathbf{x}_t(j)$, then, using that UP_t is injective, we have the following:

  (a) $\mathbf{x}_{t-1}(i) = \mathbf{x}_{t-1}(j)$, which from the induction hypothesis implies that $c_{t-1}(i) = c_{t-1}(j)$.

  (b) $\mathbf{m}_t(i) = \mathbf{m}_t(j)$. Since AGGR_t is injective w.r.t. its input multiset we get: [2]

$$\mathbf{m}_t(i) = \mathbf{m}_t(j) \Rightarrow \wr \mathbf{x}_{t-1}(w) \int_{w \in \mathcal{N}(i)} = \wr \mathbf{x}_{t-1}(z) \int_{z \in \mathcal{N}(j)}$$

  Invoking again the induction hypothesis we obtain that $\wr c_{t-1}(w) \int_{w \in \mathcal{N}(i)} = \wr c_{t-1}(z) \int_{z \in \mathcal{N}(j)}$.

Now, simply recall the update rule of 1-WL: $c_t(i) = \mathrm{HASH}\Big(c_{t-1}(i),\ \wr c_{t-1}(w) \int_{w \in \mathcal{N}(i)}\Big)$, and therefore since both arguments of the HASH function are equal for the vertices i and j, it must hold that $c_t(i) = c_t(j)$. And this concludes the proof.

□

B.1.3 Proof of Corollary 4.3

Proof. In order to prove the universality of GSN, we will show that when the substructure collection contains all graphs of size $n-1$, then there exists a parametrisation of GSN that can infer the isomorphism classes of all vertex-deleted subgraphs of the graph G (known as the *deck* of G). The reconstruction conjecture states that two graphs with at least three vertices are isomorphic if and only if they have the same deck. Thus, the deck is sufficient to distinguish all non-isomorphic graphs. Then universality of GSN follows from the arguments in [Dasoulas et al., 2020, Chen et al., 2019]. The deck can be defined as follows:

Let $\mathcal{D} = \{\alpha_1, \alpha_2, \ldots, \alpha_{|\mathcal{D}|}\} = \mathfrak{G}_{n-1}$ be the set of all possible graphs of size $n-1$. The vertex-deleted subgraphs of G are by definition all the induced subgraphs of G with size $n-1$, which we denote as:

$$\mathcal{H}_{n-1} = \{H = (\mathcal{V}_H, \mathcal{E}_H) \in \mathcal{S}(G) \text{ with } |\mathcal{V}_H| = n-1\}.$$

[2] Lemma 5 from [Xu et al., 2019] states that such a function always exists assuming that the elements of the multiset originate from a countable domain

Then, the deck $\mathrm{Deck}(G)$ can be defined as a vector of size $|\mathcal{D}|$, the j-th element of which reads:

$$\mathrm{Deck}(G)(j) = \left|\left\{H \in \mathcal{H}_{n-1} \mid H \simeq \alpha_j\right\}\right| = \sum_{H \in \mathcal{H}_{n-1}} \mathbb{1}[H \simeq \alpha_j],$$

where $\mathbb{1}[\cdot]$ the indicator function. Now the proof closely follows the proof of 4.2 of the main part of the thesis. In particular, the structural feature $\texttt{vcount}(G, \alpha_j)(i, v)$ for each atom α_j and orbit v are computed as follows:

$$\begin{aligned}\texttt{vcount}(G, \alpha_j)(i, v) &= \left|\left\{H \in \mathcal{S}(G) \mid \exists f \in \mathrm{Iso}(H, \alpha_j) \text{ s.t. } i \in \mathcal{V}_H,\ f(i) \in \mathcal{O}_v^{\mathcal{V}_{\alpha_j}}\right\}\right| \\ &= \sum_{H \in \mathcal{H}_{n-1}} \mathbb{1}[i \in \mathcal{V}_H]\mathbb{1}[\exists f \in \mathrm{Iso}(H, \alpha_j) \text{ s.t. } f(i) \in \mathcal{O}_v^{\mathcal{V}_{\alpha_j}}]\end{aligned}$$

where $\mathrm{Iso}(H, \alpha_j) = \emptyset$ if $H \not\simeq \alpha_j$, otherwise it contains all the adjacency-preserving bijective mappings. The deck can be inferred as follows:

$$\begin{aligned}\sum_{i \in \mathcal{V}} \sum_{v \in \mathcal{V}_{\alpha_j}/\mathrm{Aut}(\alpha_j)} \mathbf{w}_{\mathcal{V}}(i, v) &= \sum_{i \in \mathcal{V}} \sum_{v \in \mathcal{V}_{\alpha_j}/\mathrm{Aut}(\alpha_j)} \sum_{H \in \mathcal{H}_{n-1}} \mathbb{1}[i \in \mathcal{V}_H]\mathbb{1}[\exists f \in \mathrm{Iso}(H, \alpha_j) \text{ s.t. } f(i) \in \mathcal{O}_v^{\mathcal{V}_{\alpha_j}}] \\ &= \sum_{i \in \mathcal{V}} \sum_{H \in \mathcal{H}_{n-1}} \mathbb{1}[i \in \mathcal{V}_H] \sum_{v \in \mathcal{V}_{\alpha_j}/\mathrm{Aut}(\alpha_j)} \mathbb{1}[\exists f \in \mathrm{Iso}(H, \alpha_j) \text{ s.t. } f(i) \in \mathcal{O}_v^{\mathcal{V}_{\alpha_j}}] \\ &= \sum_{H \in \mathcal{H}_{n-1}} \mathbb{1}[H \simeq \alpha_j] \sum_{i \in \mathcal{V}} \mathbb{1}[i \in \mathcal{V}_H] \\ &= \sum_{H \in \mathcal{H}_{n-1}} (n-1)\mathbb{1}[H \simeq \alpha_j] \\ &= (n-1)\mathrm{Deck}_j(G),\end{aligned}$$

where we used that $\sum_{\mathcal{V}_{\alpha_j}/\mathrm{Aut}(\alpha_j)} \mathbb{1}[\exists f \in \mathrm{Iso}(H, \alpha_j) \text{ s.t. } f(i) \in \mathcal{O}_v^{\mathcal{V}_{\alpha_j}}] = \mathbb{1}[H \simeq \alpha_j]$, since each vertex can be mapped to at most one orbit. Thus, the deck can be inferred by a simple GSN-v parametrisation (a linear layer with depth equal to $|\mathcal{D}|$ that performs orbit-wise summation and division by the constant $n-1$ for each vertex separately, followed by a sum readout). Since GSN-v can be simulated by a GSN-e (Theorem 4.4), then GSN-e is also universal. □

B.1.4 Proof of Theorem 4.4: GSN-e is at least as powerful as GSN-v

Proof. Without loss of generality, we will show Theorem 4.4 for the case of a single atom α. In order to show that GSN-e can express GSN-v, we will first prove the following: *The vertex*

structural feature $\mathbf{w}_{\mathcal{V}}(i,:)$ *of a vertex* i *can be inferred by the edge structural features* $\mathbf{w}_{\mathcal{E}}(i,j,:)$ *of its incident edges.*

To simplify notation we define the following. For an orbit $\mathrm{Orb}_{\mathcal{V}}(i)$, denote the *orbit neighbourhood* as the multiset of the orbits of the neighbours of i and the *orbit degree* as the degree of any vertex in $\mathrm{Orb}_{\mathcal{V}}(i)$:

$$\mathcal{N}\big(\mathrm{Orb}_{\mathcal{V}}(i)\big) = \wr \mathrm{Orb}_{\mathcal{V}}(j) \mid j \in \mathcal{N}(i) \wr \text{ and } \deg(i) = \deg(\mathrm{Orb}_{\mathcal{V}}(i)) = |\mathcal{N}\big(\mathrm{Orb}_{\mathcal{V}}(i)\big)|. \tag{B.2}$$

To start, let us assume that there exists only one subgraph $H \in \mathcal{S}(G)$ such that $H \simeq \alpha$ and the bijection between $\mathcal{V}_H$ and $\mathcal{V}_\alpha$ is f. Then, for an arbitrary vertex i and a vertex orbit $\mathcal{O}_v^{\mathcal{V}_\alpha}$, one of the following holds:

- $v \notin \mathcal{V}_H$. Then $\mathbf{w}_{\mathcal{V}}(i,v) = 0$ and $\mathbf{w}_{\mathcal{E}}\big(i,j,(v,u)\big) = 0,\ \forall j \in \mathcal{N}(i), \forall u \in \mathcal{N}\big(\mathcal{O}_v^{\mathcal{V}_\alpha}\big)$,
- $v \in\in \mathcal{V}_H$ and $\mathrm{Orb}_{\mathcal{V}_\alpha}\big(f(i)\big) \neq \mathcal{O}_v^{\mathcal{V}_\alpha}$. Then, $\mathbf{w}_{\mathcal{V}}(i,v) = 0$ and $\mathbf{w}_{\mathcal{E}}\big(i,j,(v,u)\big) = 0,\ \forall j \in \mathcal{N}(i), \forall u \in \mathcal{N}\big(\mathcal{O}_v^{\mathcal{V}_\alpha}\big)$. Note that here the directionality of the edge is important, otherwise, we would not be able to determine the value of $\mathbf{w}_{\mathcal{E}}\big(i,j,(v,u)\big)$ unless we also know the orbit of $f(j)$.
- $v \in \mathcal{V}_H$ and $\mathrm{Orb}_{\mathcal{V}_\alpha}\big(f(i)\big) = \mathcal{O}_v^{\mathcal{V}_\alpha}$. Then, $\mathbf{w}_{\mathcal{V}}(i,v) = 1$ and since $f(i)$ has exactly $\deg(\mathcal{O}_v^{\mathcal{V}_\alpha})$ neighbours in α, then v has exactly $\deg(\mathcal{O}_v^{\mathcal{V}_\alpha})$ neighbours in H with vertex orbits in $\mathcal{N}\big(\mathcal{O}_v^{\mathcal{V}_\alpha}\big)$. This implies that:

$$\sum_{j \in \mathcal{N}(i)} \sum_{u\,:\,\mathcal{O}_u^{\mathcal{V}_\alpha} \in \mathcal{N}(\mathcal{O}_v^{\mathcal{V}_\alpha})} \mathbf{w}_{\mathcal{E}}\big(i,j,(v,u)\big) = \deg(\mathcal{O}_v^{\mathcal{V}_\alpha})$$

Thus, by induction, for m matched subgraphs $H \simeq \alpha$ with $i \in \mathcal{V}_H$ and $\mathrm{Orb}_{\mathcal{V}_\alpha}\big(f(v)\big) = \mathcal{O}_v^{\mathcal{V}_\alpha}$, it holds that $\mathbf{w}_{\mathcal{V}}(i,v) = m$ and $\sum_{j \in \mathcal{N}(i)} \sum_{u\,:\,\mathcal{O}_u^{\mathcal{V}_\alpha} \in \mathcal{N}(\mathcal{O}_v^{\mathcal{V}_\alpha})} \mathbf{w}_{\mathcal{E}}\big(i,j,(v,u)\big) = m \cdot \deg(\mathcal{O}_v^{\mathcal{V}_\alpha})$. Then it follows that:

$$\mathbf{w}_{\mathcal{V}}(i,v) = \frac{1}{\deg\big(\mathcal{O}_v^{\mathcal{V}_\alpha}\big)} \sum_{j \in \mathcal{N}(i)} \sum_{u\,:\,\mathcal{O}_u^{\mathcal{V}_\alpha} \in \mathcal{N}(\mathcal{O}_v^{\mathcal{V}_\alpha})} \mathbf{w}_{\mathcal{E}}\big(i,j,(v,u)\big) \tag{B.3}$$

The rest of the proof is straightforward: we will assume a GSN-v using substructure counts of the graph α, with T layers and width d. Then, there exists a GSN-e with $T+1$ layers, where the first

layer has width $d_v + d_v^w$ and implements the following function: $\mathrm{UP}_1\big(\mathbf{x}_0(i), \mathbf{m}_1(i)\big) = [\mathbf{x}_0(i); \mathbf{m}_1(i)]$, where:

$$\begin{aligned}\mathbf{m}_1(i) &= \mathrm{AGGR}_1\big(\wr(\mathbf{x}_0(i), \mathbf{x}_0(j), \mathbf{w}_{\mathcal{E}}(i,j), \mathbf{u}_{\mathcal{E}}(i,j)\wr_{j\in\mathcal{N}(i)}\big) \\ &= [\frac{1}{\deg\big(\mathcal{O}_1^{\mathcal{V}_\alpha}\big)} \sum_{j\in\mathcal{N}(i)} \sum_{u\,:\,\mathcal{O}_u^{\mathcal{V}_\alpha}\in\mathcal{N}(\mathcal{O}_1^{\mathcal{V}_\alpha})} \mathbf{w}_{\mathcal{E}}\big(i,j,(1,u)\big);\ \ldots\ ; \\ &\quad \frac{1}{\deg\big(\mathcal{O}_{d_v^w}^{\mathcal{V}_\alpha}\big)} \sum_{j\in\mathcal{N}(i)} \sum_{u\,:\,\mathcal{O}_u^{\mathcal{V}_\alpha}\in\mathcal{N}(\mathcal{O}_{d_v^w}^{\mathcal{V}_\alpha})} \mathbf{w}_{\mathcal{E}}\big(i,j,(d_v^w,u)\big)] = \mathbf{w}_{\mathcal{V}}(i)\end{aligned}$$

Note that since AGGR_1 is a universal multiset function approximator, then there exists a parameterisation with which the above function can be computed. The next T layers of GSN-e can implement a traditional MPNN where now the input vertex features are $[\mathbf{x}_1(i); \mathbf{w}_{\mathcal{V}}(i)]$ (which is exactly the formulation of GSN-v) and this concludes the proof.

□

B.2 Scalability analysis

B.2.1 Improved subgraph enumeration algorithms

Graphs with community structure (social and protein networks). The typical substructures of interest for these distributions are triangles and cliques. Their enumeration is well-studied [Chiba and Nishizeki, 1985, Danisch et al., 2018, Bron and Kerbosch, 1973, Makino and Uno, 2004, Takeaki, 2012, Jain and Seshadhri, 2020, Alon et al., 1997, Latapy, 2008, Björklund et al., 2014] and has been shown to admit an $O\big(a(G)^{k-2}m\big)$ complexity, where $a(G) \leq m^{1/2}$ the arboricity of the graph - an indicator of its sparsity - m the number of edges and k the size of the clique. This is an important improvement for sparse graphs, as it yields linear time when the arboricity is a constant and $O(n^{k/2})$ for moderate sparsity, i.e. when m grows at the same rate with n. The naive enumeration bound $O(n^k)$ is only recovered for very dense graphs, i.e. when m grows at the same rate with n^2. Moreover, parallel implementations have been shown to empirically scale well to million-edge graphs [Danisch et al., 2018].

Molecular graphs. Several algorithms have been also proposed for k-cycle enumeration [Johnson, 1975, Mateti and Deo, 1976, Birmelé et al., 2013, Ferreira et al., 2014] attaining an

optimal scaling of $O\big(m + kc(G, \alpha)\big)$ (this cannot be improved since one needs to traverse the m edges of the graph at least once, and then list all k vertices of each of the $c(G, \alpha)$ subgraphs).

Others. Improved bounds have been also shown for trees [Wasa et al., 2014, Ferreira et al., 2011] and paths [Hoàng et al., 2013] among others, while planar graphs enjoy linear-time complexity, $O\big(n + c(G, \alpha)\big)$, for any $O(1)$-sized subgraph [Eppstein, 1995, Dorn, 2010]. This is of great interest for chemoinformatics/bioinformatics applications, since almost all molecules are planar.

B.2.2 Quantitative analysis of the runtime: preprocessing

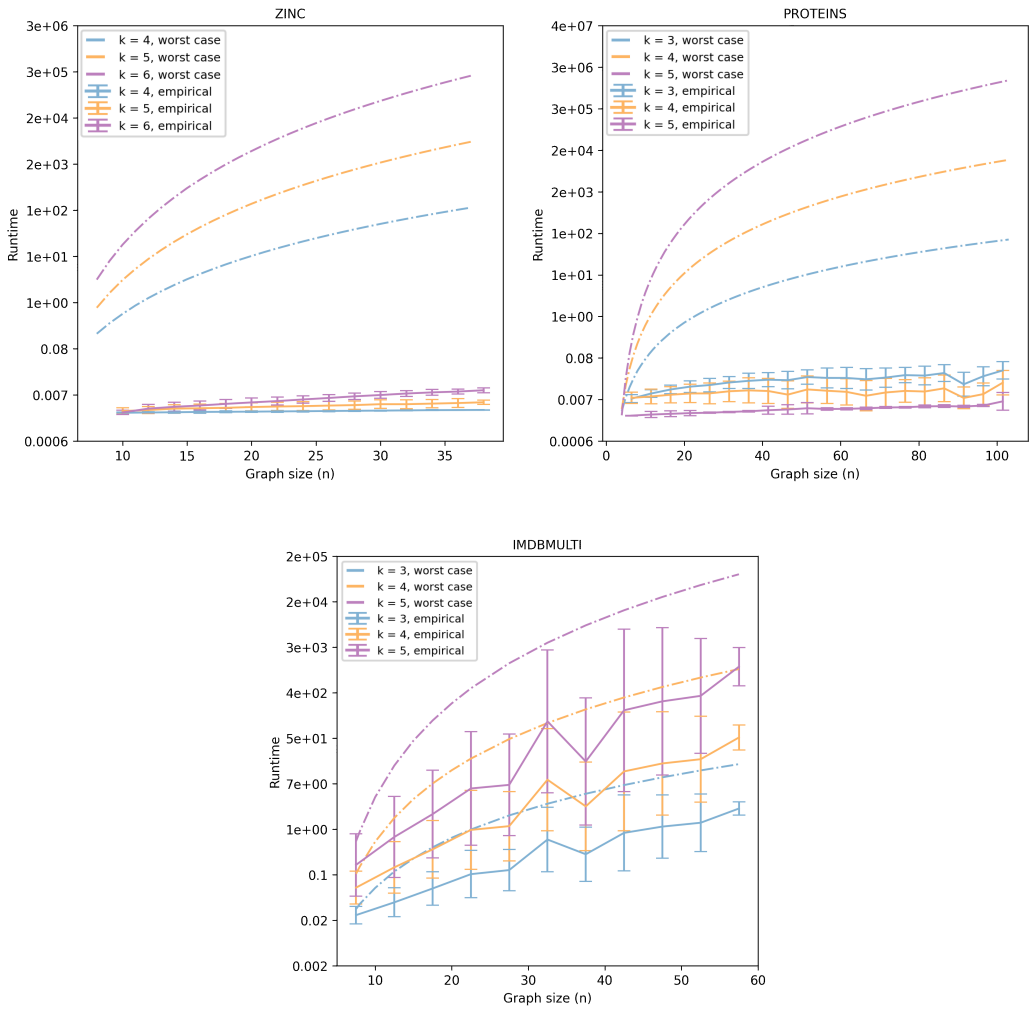


Figure B.1: Empirical (solid) vs worst case (dashed) runtime (in seconds) for different graph distributions (in seconds, log scale). For each distribution, we count the best performing (and frequent) substructures of increasing sizes k. The computational complexity for real-life graphs is significantly better than the worst case.

In Figure B.1, we study the empirical runtime of VF2 as a function of the graph size (n) and

that of the substructure (k). We compare the performance of VF2 against the worst case, for three different graph distributions: molecules, protein contact maps, and social networks. In accordance with the bounds of Table 4.1, we observe that in the first two cases, where the graphs are sparse and the number of pattern occurrences is small, the runtime is significantly smaller than the worst-case upper bound, while it scales better with the size of the graph n. Even, in the case of social networks, where several examples are near-complete graphs, the scaling is also better w.r.t both n and k.

Table B.1: Dataset statistics and preprocessing runtimes (avg and total in seconds). *Molecules.*

	Dataset name	**MUTAG**		**PTC**		**NCI1**		**ZINC**		**MOLHIV**		**MOLPCBA**	
Statistics	# graphs	188		344		4110		12,000		41,127		437,929	
	avg. # vertices	17.93		25.56		29.87		23.16		25.51		25.97	
	avg. # edges	19.79		25.96		32.3		24.92		27.46		28.11	
	avg. degree	2.19		1.99		2.16		2.15		2.14		2.16	
	avg. clust. coeff.	0.0		0.003		0.003		0.006		0.002		0.002	
	k	avg	total	avg	total	avg	total	avg	total	avg	total	avg	total
VF2	4	3.7e-5	0.007	3.9e-5	0.01	5.3e-5	0.22	4.1e-5	0.49	4.7e-5	1.90	4.5e-5	20
	6	3.2e-4	0.06	1.9e-04	0.07	3.9e-4	1.60	2.6e-4	3.10	3.2e-4	13	3.2e-4	140
	8	1.6e-4	0.03	1.3e-4	0.04	2.5e-4	1.10	1.4e-4	1.70	1.9e-4	7.80	1.7e-4	76
	10	2.2e-4	0.04	1.6e-4	0.06	3.8e-4	1.62	1.9e-4	2.20	2.6e-4	11	2.4e-4	100
VF3	4	0	0	7.2e-7	2.5e-4	2.4e-6	0.009	8.6e-7	0.01	1.4e-6	0.06	5.6e-7	0.24
	6	1.1e-4	0.02	6.9e-5	0.02	1.4e-4	0.58	1.0e-4	1.20	1.2e-4	4.80	1.2e-4	53
	8	0	0	8.9e-6	0.003	2.4e-4	1.00	5.4e-6	0.07	1.8e-5	0.75	9.2e-6	4
	10	1.7e-4	0.03	5.3e-5	0.02	2.0e-4	0.80	3.4e-5	0.40	1.0e-4	4.30	6.1e-5	27

Table B.2: Dataset statistics and preprocessing runtimes (avg and total in seconds). *Proteins & social networks.*

	Dataset name	**Proteins**		**DD**		**ogbg-ppa**		**Collab**		**IMDB-B**		**IMDB-M**	
Statistics	# graphs	1113		1178		158,100		5000		1000		1500	
	avg # verts	39.06		284.32		243.42		74.50		19.77		13.00	
	avg # edges	72.82		715.66		2,266.10		2,457.78		96.53		65.94	
	avg degree	3.74		4.98		18.33		37.37		8.89		8.10	
	avg clust coeff	0.51		0.48		0.51		0.88		0.95		0.97	
	k	avg	total	avg	total	avg	total	avg	total	avg	total	avg	total
VF2	3	0.002	1.90	0.04	47	1.80	2.9e+5	1.30	6.3e+3	0.03	27	0.02	35
	4	8.3e-4	0.93	0.04	52	21	3.3e+6	OOM	OOM	0.50	500	0.46	700
	5	7.5e-5	0.08	0.02	25	OOM	OOM	OOM	OOM	11	1.1e+4	13	1.9e+4
VF3	3	1.9e-4	0.21	0.002	2.40	0.03	5.4e+3	0.41	2.1e+3	0.001	1.10	8.0e-4	1.20
	4	2.2e-4	0.24	0.004	5.10	1.20	2.0e+5	OOM	OOM	0.02	20	0.02	25
	5	2.6e-6	0.003	0.005	5.60	OOM	OOM	OOM	OOM	0.26	262	0.27	410

In Tables B.1 and B.2 we report a comprehensive analysis of the empirical runtime performance of two subgraph isomorphism algorithms: VF2 and VF3. The study encompasses graph domains of different topological characteristics. *First-off, we should note that VF3 provided a consistent speed-up in all datasets considered.* This highlights another advantage of our strategy to increase the computational complexity of the preprocessing instead of that of the neural network:

the practitioner can benefit from improved specialised algorithms and is not constrained to tensor-based deep learning frameworks.

Table B.1 reports the results for molecular structures; they are sparse graphs characterised by a low clustering coefficient. We match cycle graphs of typical sizes: $4, 6, 8, 10$. On all the benchmarks, both algorithms perform isomorphism counting in a fraction of a second, allowing to complete the task in the order of a couple of minutes on the large-scale `ogbg-molpcba` dataset ($\sim$ 438k graphs). These results remark on the suitability of GSN on molecular modelling tasks.

Table B.2 reports results for denser graph families, where we match cliques of sizes 3, 4, and 5. Graphs in Proteins and DD represent protein structure contact maps, while those in `ogbg-ppa` model protein-protein associations. These graphs feature a medium clustering coefficient and span through medium to high density. The remaining three datasets are of a social nature, with an extremely high clustering coefficient and medium (IMDB-B, IMDB-M) to high density (Collab). Both two algorithms exhibit negligible run times on Proteins and DD. In particular, VF3 requires only a few seconds to complete the enumeration on the entire DD dataset. On the IMDB datasets run times are still tractable; on these benchmarks, it is possible to appreciate the significant speed-up achieved by VF3 w.r.t. VF2. (two order of magnitudes for $k = 4, 5$). We observed the two algorithms to run Out Of Memory (OOM) on Collab and `ogbg-ppa` for, respectively, $k \geq 4$ and $k = 5$. We attribute this behaviour to the extreme density of these graphs; here the number of cliques grows exponentially, making it unfeasible even to list them. Either way, we remark that triangle listing ($k = 3$) is tractable in both benchmarks and remind that this was sufficient to allow GSN to outperform the GIN base architecture. It is important to notice that these runtimes were obtained with *generic* subgraph isomorphism algorithms, and that specialised routines would allow for even more efficient computations (see section B.2.1).

B.2.3 Quantitative analysis of the runtime: total

We conclude our computational complexity discussion with an empirical analysis of the overall GSN runtime, both during training and during inference, including both steps of the algorithm, i.e. preprocessing and NN training (across the entire dataset)/inference (per graph). Tables B.3 and B.4 compare the total runtime of the best GSN models (the performances of which are reported in the main part of the thesis) against that of the corresponding backbone MPNN (i.e. the same architecture without structural features). First off, *for all datasets considered,*

Table B.3: Number of parameters, preprocessing and NN training & inference runtimes (in seconds) for *molecular datasets*. Best GSN model vs backbone MPNN. *Percentage* indicates the ratio of the preprocessing to the total runtime.

	Dataset name	MUTAG		PTC		NCI1		ZINC		MOLHIV		MOLPCBA	
	Model	backbone MPNN	GSN-v (k=12)	backbone MPNN	GSN-e (k=6)	backbone MPNN	GSN-v (k=3)	backbone MPNN	GSN-v (k=8)	backbone MPNN	GSN-e (k=6)	backbone MPNN	GSN-e (k=6)
Space	**#params**	2.4K	2.8K	2.7K	3K	9.4K	9.5K	378.9K	384.5K	333.3K	333.9K	1.923M	1.928M
Training time (dataset)	Preprocessing	-	0.05	-	0.03	-	0.01	-	2.15	-	7.49	-	75.99
	Training	328.42	334.08	332.63	351.27	321.27	329.13	429.35	310.42	4068.53	4862.51	5675.67	6250.23
	Total	310.55	334.13	332.63	351.30	321.27	329.14	429.35	312.57	4068.53	4870.00	5675.7	6326.22
	Percentage	-	0.015%	-	0.009%	-	0.003%	-	0.688%	-	0.154%	-	1.201%
Inference time (per graph)	Preprocessing	-	0.27e-3	-	0.09e-3	-	0.002e-3	-	0.22e-3	-	0.22e-3	-	0.22e-3
	Inference	3.77e-3	5.60e-3	3.93e-3	4.93e-3	3.90e-3	4.36e-3	5.20e-3	6.54e-3	8.97e-3	10.72e-3	7.90e-3	8.54e-3
	Total	3.77e-3	6.27e-3	3.93e-3	5.02e-3	3.90e-3	4.362e-3	5.20e-3	6.76e-3	8.97e-3	10.94e-3	7.90e-3	8.76e-3
	Percentage	-	4.76%	-	1.79%	-	0.045%	-	3.25%		2.01%	-	2.51%

Table B.4: Number of parameters, preprocessing and NN training & inference runtimes (in seconds) for *protein and social network datasets*. Best GSN model vs backbone MPNN. *Percentage* indicates the ratio of the preprocessing to the total runtime.

	Dataset name	Proteins		ogbg-ppa		Collab		IMDB-B		IMDB-M	
	Model	backbone MPNN	GSN-e (k=4)	backbone MPNN	GSN-e (k=3)	backbone MPNN	GSN-e (k=3)	backbone MPNN	GSN-e (k=5)	backbone MPNN	GSN-e (k=5)
Space	**# params**	8.3K	8.8K	3.286M	3.375M	31.1K	52.2K	30.8K	64.9K	34.8K	66K
Training time (dataset)	Preprocessing	-	0.41	-	2670.97	-	1845	-	254.79	-	392.58
	Training	327.43	329.50	19.42K	21.48K	373.13	456.55	327.37	333.14	327.65	329,89
	Total	327.43	329.931	19.42K	24.18K	373.13	2301.55	327.37	587.93	327.65	722,47
	Percentage	-	0.12%	-	12.43%	-	80.16%	-	43,34%	-	54.34%
Inference time (per graph)	Preprocessing	-	0.41e-3	-	0.034	-	0.410	-	0.283	-	0.291
	Inference	3.77e-3	4.44e-3	8.21e-3	8.54e-3	4.40e-3	5.17e-3	4.34e-3	5.34e-3	4.43e-3	5.34e-3%
	Total	3.77e-3	4.85e-3	8.21e-3	0.042	4.40e-3	0.415	4.34e-3	0.288	4.43e-3	0.296
	Percentage	-	8.45%	-	80.95%	-	98.80%	-	98.26%	-	98.31%

we observe only a small increase in the NN training time, compared to that of the backbone MPNN (approx 0.4%-5.6% increase for most TU datasets, 10.1%-19.5% for ogb and 22.4% for Collab), while in one case (ZINC), the training time is reduced, since the structural features allow for faster convergence (approx 27% reduction). *The same holds for the NN inference time* (8%- 25% increase), perhaps with the exception of the MUTAG dataset, where a slightly larger increase (48.5%) can be attributed to the fact that the dictionary used is larger compared to the rest of the datasets (12 atoms). The above results are in accordance with section 4.5, where it was discussed that the NN complexities of both GSN and the backbone MPNN are linear in the number of vertices and edges. Note that the training runtime also depends heavily on the size of the dataset and the number of epochs, which justifies the variability across datasets (e.g. for all the datasets in the TUD benchmarks we used the same number of total iterations, regardless of the dataset size, in the ZINC dataset we perform early stopping, while in the ogb benchmarks we train the network for 100 epochs, according to the required training protocols).

In terms of the total runtime, we observe that *for all molecular datasets, the preprocessing is negligible compared to the NN runtime, both during training and during inference.* This showcases

that molecular distributions, as well as distributions of graphs with small densities and clustering coefficients, are an excellent regime for GSN in terms of its computational complexity (as well as its performance as we saw in the main part of the thesis).

However, *this fraction increases for the datasets that exhibit medium to high clustering coefficient and density.* It remains negligible for the Proteins dataset (medium clustering coefficient and density), small for the ogbg-ppa dataset (medium clustering coefficient, high density), but becomes comparable to the training time for the IMDB-B and IMDB-M datasets (very high clustering coefficient and medium density) and eventually exceeds it for the Collab dataset (very high clustering coefficient and high density). This can be mitigated by preprocessing the dataset in parallel (e.g. with 16 processes we observed approximately a 6-fold speedup), while as mentioned in the main part of the thesis, it might be of lesser importance, since in ML applications, typically the same NN is trained multiple times, while the preprocessing is performed only once. However, it becomes problematic when it comes to NN inference, where for high density and high clustering coefficient graphs, the preprocessing might be orders of magnitude larger than the NN runtime. Although, arguably, this ratio can be reduced by further optimisations and specialised algorithms, the aforementioned result illustrates a limitation of our method.

Finally, to complement our results, we also report the space complexity in terms of the number of parameters, where the extra parameters of GSN, compared to the backbone MPNN, can be calculated to be linear in the dimensions of the structural features. In all the experiments the structural features are one-hot-encoded for each atom, therefore the increase will be linear in the sum of the number of unique values across all orbits of all atoms in the dictionary (or more loosely, the product of the maximum number of unique values, the size of the dictionary and the maximum k, for GSN-v, or k^2, for GSN-e). In most cases, we observe a small to medium increase (0.2% - 16.7%), with the exceptions of Collab, IMDB-B and IMDB-M, where the increase is larger (67.9%-110.7%), due to the large number of unique values for the clique counts in the dataset. However, note that this percentage can be also largely attributed to the relatively small size of the NN used, while for large overparameterised NNs, the increase percentages are neglible (0.2% - 2.7% for the ogb datasets).

B.3 Experimental Settings - Additional Details

In this section, we provide additional implementation details of our experiments. All experiments were performed on a server equipped with 8 Tesla V100 16 GB GPUs, except for the Collab dataset where a Tesla V100 GPU with 32 GB RAM was used due to larger memory requirements. Experimental tracking and hyperparameter optimisation were done via the Weights & Biases platform (wandb) [Biewald, 2020]. Our implementation is based on native PyTorch sparse operations [Paszke et al., 2019] in order to ensure complete reproducibility of the results. PyTorch Geometric [Fey and Lenssen, 2019a] was used for additional operations (such as preprocessing and data loading).

In each one of the different experiments, we aim to show that **structural identifiers can be used off-the-shelf and independently of the architecture.** At the same time we aim to suppress the effect of other confounding factors in the model performance, thus wherever possible we build our model on top of a baseline architecture. For more details, please see the relevant subsections. Interestingly, we observed that in most of the cases, it was sufficient to replace only the first layer of the baseline architecture with a GSN layer, in order to obtain a boost in performance.

Throughout the experimental evaluation, the structural identifiers $\mathbf{w}_{\mathcal{V}}$ and $\mathbf{w}_{\mathcal{E}}$ are one-hot encoded, by taking into account the unique count values present in the dataset. Other more sophisticated methods can be used, e.g. transformation to continuous features via a normalisation scheme or binning. However, we found that the number of unique values in our datasets was usually relatively small (which is a good indication of recurrent structural roles) and thus such methods were not necessary.

B.3.1 Automorphism computation and orbit matching

In order to compute the automorphism of a graph in the dictionary we first run a graph isomorphism algorithm between the graph and itself and obtain a set of vertex bijections. Subsequently, to compute the orbits, we initialise each orbit to contain a single vertex (using an arbitrary vertex ordering) and then iterate over the bijections found in the previous step. Whenever two vertices of different orbits are matched by a bijection, the two orbits are merged. Both steps need $O(k!k)$ time in the worst case, but since k is a small constant (independent of

the size of the target graphs) and this operation is performed only once, its run time is negligible (in practice it takes a fraction of a millisecond).

Assigning orbit identifiers to each vertex/edge (for GSN-v, GSN-e respectively) is a straightforward procedure. In particular, the subgraph isomorphism algorithm returns mappings (bijections) between the vertices of the substructure in the dictionary and the vertices of the target graph, which implies that we can infer the corresponding orbit for each vertex in $O(1)$ by simply using the look-up table that we computed in the previous step (we need $O(c(G, \alpha)k)$ time to traverse all the detected subgraphs, where $c(G, \alpha)$ their number, but this process can be merged with the subgraph enumeration).

B.3.2 Experimental details

Graph Isomorphism testing. For the Strongly Regular graphs dataset (available from http://users.cecs.anu.edu.au/~bdm/data/graphs.html) we use all the available families of graphs with a size of at most 35 vertices: The total number of non-isomorphic pairs of the same size is $\approx 7 * 10^7$. We used a simple 2-layer architecture with a width equal to 64. The message aggregation was performed as in the general formulation of (2.11), where the update and the message functions are MLPs. The prediction is inferred by applying a sum readout function in the last layer and then passing the output through an MLP. Regarding the substructures, we use *graphlet* counting, as certain *motifs* (e.g. cycles of length up to 7) are known to be unable to distinguish strongly regular graphs (since they can be counted by the 2-FWL [Fürer, 2017, Arvind et al., 2019]).

Given the adversities that strongly regular graphs pose in graph isomorphism testing, it would be interesting to see how this method can perform in other categories of hard instances, such as the classical *CFI* counter-examples for k-WL proposed in [Cai et al., 1992], and explore further its expressive power and combinatorial properties. We leave this direction to future work.

TUD benchmarks. For this family of experiments, due to the usually small size of the datasets, we choose a parameter-efficient architecture, in order to reduce the risk of overfitting. In particular, we follow the simple GIN architecture [Xu et al., 2019] and we concatenate structural identifiers to vertex or edge features depending on the variant. Then for GSN-v, the

hidden representation is updated as follows:

$$\mathbf{x}_{t+1}(i) = \mathrm{UP}_{t+1}\Big([\mathbf{x}_t(i); \mathbf{w}_{\mathcal{V}}(i)] + \sum_{j \in \mathcal{N}(i)} [\mathbf{x}_t(j); \mathbf{w}_{\mathcal{V}}(j)]\Big), \tag{B.4}$$

and for GSN-e:

$$\mathbf{x}_{t+1}(i) = \mathrm{UP}_{t+1}\Big([\mathbf{x}_t(i); \mathbf{w}_{\mathcal{E}}(i,i)] + \sum_{j \in \mathcal{N}(i)} [\mathbf{x}_t(j); \mathbf{w}_{\mathcal{E}}(i,j))]\Big), \tag{B.5}$$

where $\mathbf{w}_{\mathcal{E}}(i,i)$ is a dummy variable (also one-hot encoded) used to distinguish self-loops from edges. Empirically, we did not find training the ϵ parameter used in GIN to make a difference.

We implement an architecture similar to GIN [Xu et al., 2019], i.e. *message passing layers*: 4, *jumping knowledge* from all the layers [Xu et al., 2018a] (including the input), *transformation of each intermediate graph-level representation*: linear layer, *readout*: sum for biological and mean for social networks. Vertex features are one-hot encodings of categorical vertex labels. Similarly to the baseline, the hyperparameters search space is the following: *batch size* in {32, 128} (except for Collab where only 32 was searched due to GPU memory limits), *dropout* in {0,0.5}, *network width* in {16,32} for biological networks, 64 for social networks, *learning rate* in {0.01, 0.001}, *decay rate* in {0.5,0.9} and *decay steps* in {10,50} (number of epochs after which the learning rate is reduced by multiplying with the decay rate). For social networks, since they are not attributed graphs, we also experimented with using the degree as a vertex feature, but in most cases the structural identifiers were sufficient.

Model selection is done in two stages. First, we choose a substructure that we perceive as promising based on indications from the specific domain: *triangles* for social networks and Proteins, and *6-cycles (motifs)* for molecules. Under this setting, we tune model hyperparameters for a GSN-e model. Then, we extend our search to the parameters related to the substructure collection: i.e. *the maximum size* k and *motifs vs graphlets*. In all the molecular datasets we search cycles with $k = 3, \ldots, 12$, except for NCI1, where we also consider larger sizes due to the presence of large rings in the dataset (*macrocycles* [Liu et al., 2017b]). For social networks, we searched cliques with $k = 3, 4, 5$. In Table B.5 we report the hyperparameters chosen by our model selection procedure, including the best-performing substructures.

The seven datasets[3] we chose are the intersection of the datasets used by the authors of our

[3]more details on the description of the datasets and the corresponding tasks can be found at [Xu et al., 2019].

Table B.5: Chosen hyperparameters for GSN-e and GSN-v on the TUD datasets.

	Dataset	MUTAG	PTC	Proteins	NCI1	Collab	IMDB-B	IMDB-M
GSN-e	batch size	32	128	32	32	32	32	32
	width	32	16	32	32	64	64	64
	decay rate	0.9	0.5	0.5	0.9	0.5	0.5	0.5
	decay steps	50	50	10	10	50	10	10
	dropout	0.5	0	0.5	0	0	0	0
	lr	10^{-3}	10^{-3}	10^{-2}	10^{-3}	10^{-2}	10^{-3}	10^{-3}
	degree	No	No	No	No	No	No	Yes
	subgraph type	graphlets	motifs	same	graphlets	same	same	same
	atom family	cycles	cycles	cliques	cycles	clique	clique	cliques
	k	6	6	4	15	3	5	5
GSN-v	batch size	32	128	32	32	32	32	32
	width	32	16	32	32	64	64	64
	decay rate	0.9	0.5	0.5	0.9	0.5	0.5	0.5
	decay steps	50	50	10	10	50	10	10
	dropout	0.5	0	0.5	0	0	0	0
	lr	10^{-3}	10^{-3}	10^{-2}	10^{-3}	10^{-2}	10^{-3}	10^{-3}
	degree	No	No	No	No	No	Yes	Yes
	subgraph type	graphlets	graphlets	same	same	same	same	same
	atom family	cycles	cycles	cliques	cycles	cliques	clique	cliques
	k	12	10	4	3	3	4	3

main baselines: the Graph Isomorphism Network (GIN) [Xu et al., 2019], a simple, yet powerful GNN with expressive power equal to the 1-WL test, and the Provably Powerful Graph Network (PPGN) [Maron et al., 2019a], a polynomial alternative to the Invariant Graph Network [Maron et al., 2019b], that increases its expressive power to match the 2-FWL. We also compare our results to other GNNs as well as Graph Kernel approaches. Our main baseline from the GK family is the Graph Neural Tangent Kernel (GNTK) [Du et al., 2019a], which is a kernel obtained from a GNN of infinite width that operates in the Neural Tangent Kernel regime [Jacot et al., 2018, Allen-Zhu et al., 2019, Du et al., 2019c].

ZINC benchmark - Experimental Details. The ZINC dataset includes 12k molecular graphs of which 10k form the training set and the remaining 2k are equally split between validation and test (splits obtained from `https://github.com/graphdeeplearning/benchmarking-gnns`). Molecule sizes range from 9 to 37 vertices/atoms. Vertex features encode the type of atoms and edge features the chemical bonds between them. Again, here vertex and edge features are one-hot encoded.

Our MPNN baseline model updates vertex representations as follows: $\mathbf{x}_{t+1}(i) = \mathrm{MLP}_{t+1}\big(\mathbf{x}_t(i), \mathbf{m}_{t+1}(i)\big)$, $\mathbf{m}_{t+1}(i) = \sum_{j\in\mathcal{N}(i)} \mathrm{MLP}_t\big(\mathbf{x}_t(i), \mathbf{x}_t(j), \mathbf{u}_{\mathcal{E}}(i,j)\big)$. Our instantiation of GSN is a simple extension

where structural identifiers are also given as input to the message MLP. Following the same rationale as before, the network configuration is minimally modified w.r.t. the baselines provided in [Dwivedi et al., 2020], while here no hyperparameter tuning is done and we use the default ones provided by the authors. In particular, the parameters are the following: *message passing layers*: 4, *transformation of the output of the last layer*: MLP, *readout*: sum, *batch size*: 128, *dropout*: 0.0, *network width*: 128, *learning rate*: 0.001. The learning rate is reduced by 0.5 (decay rate) after 5 epochs (*decay rate patience*) without improvement in the validation loss. Training is stopped when the learning rate reaches the *minimum learning rate* value of 10^{-5}. Validation and test metrics are inferred using the model at the last training epoch.

We select our best-performing substructure-related parameters based on the performance in the validation set in the last epoch. We search cycles with $k = 3, \ldots, 10$, *graphlets vs motifs*, and *GSN-v vs GSN-e*. The chosen hyperparameters for GSN are: *GSN-e, cycle graphlets of 10 vertices* and for GSN-*EF*: *GSN-v, cycle motifs of 8 vertices*. Once the model is chosen, we repeat the experiment 10 times with different seeds and report the mean and standard deviation of the test MAE in the last epoch.

Disambiguation Scores: In Table B.6, we provide the disambiguation scores $1 - \delta_{|\mathfrak{D}|}$ as defined in section 4.8.5 for different types of substructures. These are computed based on vertex structural identifiers (GSN-v).

Table B.6: Disambiguation scores $1 - \delta_{|\mathfrak{D}|}$ on *ZINC* for different substructure families and maximum size k. Size $k = 0$ refers to using only the original vertex features.

k	Cycles	Paths	Trees
0	0.196	0.196	0.196
3	0.199	0.540	0.540
4	0.200	0.746	0.762
5	0.256	0.866	0.875
6	0.327	0.895	0.897
7	0.330	0.900	0.900
8	0.330	0.901	0.901
9	0.330	0.901	0.901
10	0.330	0.901	0.901

OGB benchmarks We extend the following base architectures:

GIN. We follow the design choices of the authors of [Hu et al., 2020] and extend their architectures to include structural identifiers. For `ogbg-molhiv` and `ogbg-molpcba`, initial vertex

and edge features are multi-hot encodings, while for `ogbg-ppa` in absence of vertex features, we use a constant, and the edge features are continuous values in $[0, 1]$ indicating the confidence of each edge (from 7 different sources of information). These are passed through linear layers that project them in the same embedding space, i.e. $\mathbf{x}_0(i) = \mathbf{W}_{0,x} \cdot \mathbf{x}_{\text{in}}(i)$, $\mathbf{u}^t_{\mathcal{E}}(i,j) = \mathbf{W}_{t,e} \cdot \mathbf{u}^{\text{in}}_{\mathcal{E}}(i,j)$. The baseline model is a modification of GIN that allows for edge features: for each neighbour, the hidden representation is added to an embedding of its associated edge feature. Then the result is passed through a ReLU non-linearity which produces the neighbour's message. Formally, the aggregation is as follows:

$$\mathbf{x}_{t+1}(i) = \text{UP}_{t+1}\left(\mathbf{x}_t(i) + \sum_{j \in \mathcal{N}(i)} \sigma\left(\mathbf{x}_t(j) + \mathbf{u}^t_{\mathcal{E}}(i,j)\right)\right) \tag{B.6}$$

GIN + VN. In order to allow global information to be broadcasted to the vertices, a *virtual node* takes part in the message passing. The virtual node representation, denoted as $\mathbf{G}_t$, is initialised as a zero vector $\mathbf{G}_0$ and then Message Passing becomes:

$$\begin{aligned} &\tilde{\mathbf{x}}_t(i) = \mathbf{x}_t(i) + \mathbf{G}_t,\ \mathbf{x}_{t+1}(i) = \text{UP}_{t+1}\left(\tilde{\mathbf{x}}_t(i) + \sum_{j \in \mathcal{N}(i)} \sigma\left(\tilde{\mathbf{x}}_t(j) + \mathbf{u}^t_{\mathcal{E}}(i,j)\right)\right), \\ &\mathbf{G}_{t+1} = \text{MLP}_{t+1}\left(\mathbf{G}_t + \sum_{j \in \mathcal{V}} \tilde{\mathbf{x}}_t(j)\right) \end{aligned} \tag{B.7}$$

We modify these models, as follows: first, the substructure counts are embedded into the same embedding space as the rest of the features. Then, for GSN-v, they are added to the corresponding vertex embeddings: $\acute{\mathbf{x}}_t(i) = \mathbf{x}_t(i) + \mathbf{W}^t_V \cdot \mathbf{w}_{\mathcal{V}}(i)$, or for GSN-e, they are added to the edge embeddings $\acute{\mathbf{u}}_{\mathcal{E}}{}^t(i,j) = \mathbf{u}^t_{\mathcal{E}}(i,j) + \mathbf{W}^t_E \cdot \mathbf{w}_{\mathcal{E}}(i,j)$.

DGN + substructures. We use the directional average operator as defined in [Beaini et al., 2021]:

$$\mathbf{m}_{t+1}(i) = [\sum_{j \in \mathcal{N}(i)} a(i,j,1)\mathbf{x}_t(j); \ldots; \sum_{j \in \mathcal{N}(i)} a(i,j,L)\mathbf{x}_t(j)], \tag{B.8}$$

where $a(i,j,\ell)$ are weighting average coefficients. In our case, each orbit v induces a separate set of averaging coefficients. For example, for GSN-e $a(i,j,\cdot) = \frac{\left|\mathbf{w}_{\mathcal{E}}\left(i,j,\cdot\right)\right|}{\epsilon + \sum_{j' \in \mathcal{N}(i)} \left|\mathbf{w}_{\mathcal{E}}\left(i,j',\cdot\right)\right|}$, where $\mathbf{w}_{\mathcal{E}}\left(i,j,\cdot\right)$ denotes edge-wise substructure counts (the index of the orbit (v,w) was dropped to simplify notation). Similarly, for GSN-v, $\alpha(i,j,\cdot) = \frac{|\mathbf{w}_{\mathcal{V}}(i,\cdot) - \mathbf{w}_{\mathcal{V}}(j,\cdot)|}{\epsilon + \sum_{j' \in \mathcal{N}(i)} |\mathbf{w}_{\mathcal{V}}(i,\cdot) - \mathbf{w}_{\mathcal{V}}(j',\cdot)|}$. Subsequently,

the vertex representation is updated as follows: $\mathbf{x}_{t+1}(i) = \mathrm{MLP}_{t+1}(\mathbf{m}_{t+1}(i))$. Observe that this model is simpler than the aforementioned, in terms of both its parameter count and its expressive power. Since the MOLHIV dataset poses a significant challenge w.r.t. generalisation (the data splits reflect different molecular distributions), architectures biased towards simpler solutions usually perform better, since they mitigate the risk of overfitting.

In both cases, we use the same hyperparameters as the ones provided by the authors, and only select the substructure-related parameters based on the highest validation metric (choosing the best scoring epoch as in [Hu et al., 2020]). We compare *GSN-v vs GSN-e* and search cycles with $k = 3, \ldots, 12$ for the molecular datasets, while for `ogbg-ppa` we only tested triangles. The chosen hyperparameters are *GSN-e* (all 3 datasets), *cycle graphlets of 6 vertices* (for molecules). We repeat the experiment 10 times with different seeds and report the mean and standard deviation of the train, validation and test ROC-AUC, again by choosing the best scoring epoch w.r.t the validation set.

DeepSets on structural features. The baseline architecture treats the input vertex and edge features, along with the structural identifiers, as a *set*. In particular, we consider each graph as a set of independent edges (i, j) endowed with the features of the endpoint vertices $\mathbf{u}_{\mathcal{V}}(i), \mathbf{u}_{\mathcal{V}}(j)$, the structural identifiers $\mathbf{w}_{\mathcal{V}}(i), \mathbf{w}_{\mathcal{V}}(j)$ and the edge features $\mathbf{u}_{\mathcal{E}}(i, j)$, and we implement a DeepSets universal set function approximator [Zaheer et al., 2017] to learn a prediction function:

$$h\Big(\Big\{\big(\mathbf{u}_{\mathcal{V}}(i), \mathbf{u}_{\mathcal{V}}(j), \mathbf{w}_{\mathcal{V}}(i), \mathbf{w}_{\mathcal{V}}(j), \mathbf{u}_{\mathcal{E}}(i, j)\big)\Big\}_{(i,j)\in\mathcal{E}}\Big) = \psi\Big(\sum_{(i,j)\in\mathcal{E}} \phi\big(\mathbf{u}_{\mathcal{V}}(i), \mathbf{u}_{\mathcal{V}}(j), \mathbf{w}_{\mathcal{V}}(i), \mathbf{w}_{\mathcal{V}}(j), \mathbf{u}_{\mathcal{E}}(i, j)\big)\Big), \tag{B.9}$$

with $\mathcal{E}$ the edge set of the graph and ψ, ϕ MLPs. This baseline is naturally extended to the case where we consider edge structural identifiers by replacing $(\mathbf{w}_{\mathcal{V}}(i), \mathbf{w}_{\mathcal{V}}(j))$ with $\mathbf{w}_{\mathcal{E}}(i, j)$. For fairness of evaluation, we follow the exact same parameter tuning procedure as the one we followed for our GSN models for each benchmark, i.e. for the TUD datasets we first tune network and optimisation hyperaparameters (network width was set to be either equal to the ones we tuned for GSN, or such that the absolute number of learnable parameters was equal to those used by GSN; depth of the MLPs was set to 2) and subsequently we choose the substructure related parameters based on the evaluation protocol of [Xu et al., 2019]. For ZINC and ogbg-molhiv we perform only substructure selection, based on the performance on the validation set. Using the same widths as in GSN leads to smaller baseline models w.r.t the absolute number of parameters,

and we interestingly observed this to lead to particularly strong performance in some cases, especially Proteins and MUTAG, where our DeepSets implementation attains state-of-art results. This finding motivated us to explore 'smaller' GSNs (with either reduced layer width or a single message passing layer). These GSN variants exhibited a similar trend, i.e. to perform better than their 'larger' counterparts over these two datasets. We hypothesise this phenomenon to be mostly due to the small size of these datasets, which encourages overfitting when using architectures with larger capacities. In Table 4.6 in the main part of the thesis, we report the result for the best-performing architectures, along with the number of learnable parameters.

C

Appendix to chapter 5

C.1 Omitted proofs

C.1.1 The benefits of unlabelled graph compression - proof of Theorem 5.1.

Proof. First, let us introduce some additional notation that will be useful for the proof. Denote the *preimage* of a set $\mathcal{Y}' \subseteq \mathcal{Y}$ under a function $f : \mathcal{X} \to \mathcal{Y}$ with $f^{-1}[\mathcal{Y}'] = \{x \in \mathcal{X} \mid f(x) \in \mathcal{Y}'\}$. For example, with this notation we can define the *level sets* $f^{-1}[y] = \{x \in \mathcal{X} \mid f(x) = y\}$. Of interest is the level set $h^{-1}_{\text{g-enc}}[z] = \{G \in \mathfrak{G} \mid h_{\text{g-enc}}(G) = z\}$ and more specifically, for a given G the following level set:

$$h^{-1}_{\text{g-enc}}[h_{\text{g-enc}}(G)] = \{G' \in \mathfrak{G} \mid h_{\text{g-enc}}(G') = h_{\text{g-enc}}(G)\}. \tag{C.1}$$

Observe that since $h_{\text{g-enc}}$ is GI-injective we have that if $G' \in h^{-1}_{\text{g-enc}}[h_{\text{g-enc}}(G)]$ then $G' \simeq G$. Therefore, $h^{-1}_{\text{g-enc}}[h_{\text{g-enc}}(G)] \subseteq \text{Orb}(G)$. Moreover, for any distribution q_G on $\mathfrak{G}$ and its push-forward q_z on $\mathcal{Z}$ we have that $q_z(z) = \sum_{G \in \mathfrak{G}} q_{z|G}(z|G) q_G(G) = \sum_{G \in h^{-1}_{\text{g-enc}}[z]} q_G(G)$. Now, for the true distribution p_G on $\mathfrak{G}$ we have that $p_G(G) = \frac{p_S(\text{Orb}(G)}{|\text{Orb}(G)|}$, $\forall G \in \text{Orb}(G)$, and therefore we derive the following identity which will be useful for our derivation:

$$p_z\big(h_{\text{g-enc}}(G)\big) = \sum_{G' \in h^{-1}_{\text{g-enc}}[h_{\text{g-enc}}(G)]} p_G(G') = p_S\big(\text{Orb}(G)\big) \frac{\big|h^{-1}_{\text{g-enc}}[h_{\text{g-enc}}(G)]\big|}{\big|\text{Orb}(G)\big|}. \tag{C.2}$$

Now, we can proceed as follows:

$$
\begin{aligned}
\min_{q_z} \mathbb{E}_{z\sim p_z}\big[-\log q_z(z)\big] &= \mathbb{E}_{z\sim p_z}\big[-\log p_z(z)\big] \\
&= \mathbb{E}_{G\sim p_G}\big[-\log p_z\big(h_{\text{g-enc}}(G)\big)\big] \\
&= \mathbb{E}_{G\sim p_G}\Big[-\log\Big(p_S\big(\mathrm{Orb}(G)\big)\frac{\big|h^{-1}_{\text{g-enc}}[h_{\text{g-enc}}(G)]\big|}{\big|\mathrm{Orb}(G)\big|}\Big)\Big] \\
&= \mathbb{E}_{G\sim p_G}[-\log p_S\big(\mathrm{Orb}(G)\big)] + \mathbb{E}_{G\sim p_G}\Big[-\log\Big(\frac{\big|h^{-1}_{\text{g-enc}}[h_{\text{g-enc}}(G)]\big|}{\big|\mathrm{Orb}(G)\big|}\Big)\Big] \\
&= -\sum_{G\in\mathfrak{G}} p_G(G)\log p_S\big(\mathrm{Orb}(G)\big) + \mathbb{E}_{G\sim p_G}\Big[-\log\Big(\frac{\big|h^{-1}_{\text{g-enc}}[h_{\text{g-enc}}(G)]\big|}{\big|\mathrm{Orb}(G)\big|}\Big)\Big] \\
&= -\sum_{S\in\mathfrak{G}/\simeq}\sum_{G\in S}\frac{p_S(S)}{|S|}\log p_S(S) + \mathbb{E}_{G\sim p_G}\Big[-\log\Big(\frac{\big|h^{-1}_{\text{g-enc}}[h_{\text{g-enc}}(G)]\big|}{\big|\mathrm{Orb}(G)\big|}\Big)\Big] \\
&= -\sum_{S\in\mathfrak{G}/\simeq} p_S(S)\log p_S(S) + \mathbb{E}_{G\sim p_G}\Big[-\log\Big(\frac{\big|h^{-1}_{\text{g-enc}}[h_{\text{g-enc}}(G)]\big|}{\big|\mathrm{Orb}(G)\big|}\Big)\Big] \\
&= \mathbb{H}_{S\sim p_S}[S] + \mathbb{E}_{G\sim p_G}\big[\log\big(|\mathrm{Orb}(G)|\big)\big] - \mathbb{E}_{G\sim p_G}\big[\log\big(\big|h^{-1}_{\text{g-enc}}[h_{\text{g-enc}}(G)]\big|\big)\big].
\end{aligned}
$$

The minimum and maximum values of the above quantity are obtained by inspecting the following cases:

- When $h_{\text{g-enc}}$ is isomorphism-invariant we have that $h^{-1}_{\text{g-enc}}[h_{\text{g-enc}}(G)] = \mathrm{Orb}(G), \forall G \in \mathfrak{G}$ and therefore

$$\min_{q_z} \mathbb{E}_{z\sim p_z}\big[-\log q_z(z)\big] = \mathbb{H}_{S\sim p_S}[S], \tag{C.3}$$

  which is the optimal bit rate one can obtain.

- When $h_{\text{g-enc}}$ is injective, e.g. the identity function when doing labelled graph compression, we have that $h^{-1}_{\text{g-enc}}[h_{\text{g-enc}}(G)] = \{G\}$ and therefore

$$
\begin{aligned}
\min_{q_z} \mathbb{E}_{z\sim p_z}\big[-\log q_z(z)\big] &= \mathbb{H}_{S\sim p_S}[S] + \mathbb{E}_{G\sim p_G}\big[\log\big(|\mathrm{Orb}(G)|\big)\big] \\
&= \mathbb{H}_{S\sim p_S}[S] + \mathbb{E}_{G\sim p_G}\Big[\log\frac{|\mathcal{V}_G!|}{|\mathrm{Aut}(G)|}\Big],
\end{aligned} \tag{C.4}
$$

  which recovers the result of [Choi and Szpankowski, 2012]. Assuming that all graphs in the distribution have n vertices and w.h.p. the graphs in the distribution have few automorphisms (this is a mild assumption and it holds for most medium and large size graphs [Erdos and Rényi, 1963]), the above simplifies to $\mathbb{H}_{S\sim p_S}[S] + \log n! - \mathbb{E}_{G\sim p_G}\big[\log|\mathrm{Aut}(G)|\big] =$

$\mathbb{H}_{S\sim p_S}[S] + O\big(n \log n\big)$, where we used Stirling's approximation $\log n! \approx n \log n$ and $\mathbb{E}_{G\in S\sim p_S}\big[\log|\mathrm{Aut}(G)|\big] = O(1)$.

□

C.1.2 Preliminaries for the proofs of section 5.5

Recall the definition of the binary entropy $\mathrm{H}(p) = -p\log p - (1-p)\log(1-p)$. A useful approximation that will be used in the analysis is the following:

$$\log\binom{n}{m} \approx n\,\mathrm{H}\left(\frac{m}{n}\right), \tag{C.5}$$

can be derived using Stirling's approximation. For the following comparisons, we will be considering a graph distribution on graphs with a fixed number of n vertices and therefore $\mathrm{L}(G) = \mathrm{L}(G \mid n)$. For better exposition, the analysis will be performed for directed graphs, where self-loops are allowed. The same trends in the bounds hold also for undirected graphs and graphs where self-loops are prevented.

Conventional graph encodings. We consider two types of baseline graph encodings:

- *uniform*: $\mathrm{L}^{\text{unif-G}}(G) = n^2$. Here no assumptions are made about the graph; all graphs are considered to be equally probable.
- *Erdős-Renyi*: $\mathrm{L}^{\text{ER-G}}(G) = \log\binom{n^2}{m} + \log(n^2+1) \approx n^2\mathrm{H}_m + \log(n^2+1)$, where m the number of edges and $\mathrm{H}_m := \mathrm{H}\big(\frac{m}{n^2}\big)$. This baseline is efficient at encoding graphs that are either very sparse or very dense.

Though the above encodings assign the same probability to every pair of isomorphic graphs, they always map them to different codewords. Hence, they are heavily redundant when dealing with unlabelled graphs. The following variants are more efficient by taking into account isomorphism:

- *uniform - isomorphism classes*: $\mathrm{L}^{\text{unif-S}}(G) = \log|\mathfrak{G}_n/\simeq| \approx n^2 - n\log n$.
- *Erdős-Renyi - isomorphism classes*: $\mathrm{L}^{\text{ER-S}}(G) = \log|\mathfrak{G}_{n,m}/\simeq| + \log(n^2+1) \approx n^2\mathrm{H}_m + \log(n^2+1) - n\log n$,[1]

[1]This approximation is less tight for very sparse or very dense graphs for which the size of the automorphism group is large.

where $\mathfrak{G}_n/\simeq$ and $\mathfrak{G}_{n,m}/\simeq$ are the set of all unlabelled graphs with n vertices and the set of all unlabelled graphs with n vertices and m edges, respectively. In both cases, we used the fact that asymptotically almost all graphs are rigid i.e., that they have only the trivial automorphism [Erdos and Rényi, 1963]. Observing that all four encodings asymptotically grow quadratically with the number of nodes we can derive the following lemma:

Lemma C.0.1. *Consider a graph distribution p over graphs with n vertices and denote by $\bar{\mathrm{H}}_m := \mathbb{E}_{G\sim p}[\mathrm{H}(\frac{m}{n^2})]$ the expected value of the binary entropy of the number of edges m. If $\bar{\mathrm{H}}_m < 1$, then the expected description lengths of the baseline models are asymptotically ordered as follows:*

$$\mathbb{E}_{G\sim p}[\mathrm{L}^{\text{ER-S}}(G)] \lesssim \mathbb{E}_{G\sim p}[\mathrm{L}^{\text{ER-G}}(G)] \lesssim \mathbb{E}_{G\sim p}[\mathrm{L}^{\text{unif-S}}(G)] \lesssim \mathbb{E}_{G\sim p}[\mathrm{L}^{\text{unif-G}}(G)] \tag{C.6}$$

The compression gain when encoding isomorphism classes instead of labelled graphs is $\Theta(n\log n)$, while that of the Erdős-Renyi encoding compared to the uniform one is $\Theta\big(n^2(1-\bar{\mathrm{H}}_m)\big)$.

The proof follows directly from the equations above using only the quadratic term that dominates the growth with n. More precisely the following hold:

- $\mathbb{E}_{G\sim p}[\mathrm{L}^{\text{unif-S}}(G)] \leq \mathbb{E}_{G\sim p}[\mathrm{L}^{\text{unif-G}}(G)]$ and $\mathbb{E}_{G\sim p}[\mathrm{L}^{\text{ER-S}}(G)] \leq \mathbb{E}_{G\sim p}[\mathrm{L}^{\text{ER-G}}(G)]$, and the inequalities are strict whenever the expected size of the automorphism group of a group is $\leq n!$. With the assumption that the graphs in our distribution are with high probability rigid/asymmetric, we get that the compression gains are in both cases $O(\log n!) \approx O(n\log n)$.
- $\bar{\mathrm{H}}_m < 1 - \frac{\log(n^2+1)}{n^2} \implies \mathbb{E}_{G\sim p}[\mathrm{L}^{\text{ER-G}}(G)] \leq \mathbb{E}_{G\sim p}[\mathrm{L}^{\text{unif-G}}(G)]$. The inequality is easily satisfied even for small n when $\bar{\mathrm{H}}_m < 1$. Since most real-world graphs are sparse, then the condition $\bar{\mathrm{H}}_m < 1$ is almost always true and therefore the uniform encoding is rarely beneficial. A similar conclusion can be drawn for the comparison between $\mathbb{E}_{G\sim p}[\mathrm{L}^{\text{ER-S}}(G)]$ and $\mathbb{E}_{G\sim p}[\mathrm{L}^{\text{unif-S}}(G)]$.
- $\bar{\mathrm{H}}_m < 1 - \frac{\log(n^2+1)}{n^2} - \frac{\log(n!)}{n^2} \implies \mathbb{E}_{G\sim p}[\mathrm{L}^{\text{ER-G}}(G)] \leq \mathbb{E}_{G\sim p}[\mathrm{L}^{\text{unif-S}}(G)]$. In this case, the last term diminishes slower - with $O(\frac{\log n}{n})$. Therefore, the range of values of $\bar{\mathrm{H}}_m$ for which the inequality holds is smaller compared to the previous case (for the same value of n), however, we argue that the inequality holds for most real-world graph distributions, due to their sparsity, once again.

Partitioning (only). The partitioning models considered in this analysis assume that each graph is clustered into b subgraphs of k vertices each (i.e. $b = \frac{n}{k}$) and that the intra- and inter-subgraph edges are encoded independently.[2] Note that the preamble terms that encode the number of blocks in the partition and the number of vertices per block are unnecessary since k is fixed. Overall, the code length is given by $\mathrm{L}^{\mathrm{Part}}(G) = \mathrm{L}(\mathcal{H}) + \mathrm{L}(C|\mathcal{H})$, with the two terms defined respectively as follows:

$$\mathrm{L}(\mathcal{H}) = \sum_{i=1}^{b} \left(\log(k^2+1) + \log \binom{k^2}{m_i} \right) \quad \text{and} \quad \mathrm{L}(C|\mathcal{H}) = \sum_{i \neq j}^{b} \left(\log(k^2+1) + \log \binom{k^2}{m_{ij}} \right). \tag{C.7}$$

Above, m_i is the number of edges in subgraph H_i and m_{ij} is the size of the cut between subgraphs H_i and H_j. In both cases, the first term encodes the number of edges, and the second their arrangement across the vertices.

Partition and Code (PnC). We make the same assumptions for PnC as in $\mathrm{L}^{\mathrm{Part}}(G)$: the graph is partitioned into b subgraphs of k vertices and the same encoding for non-dictionary subgraphs $\mathcal{H}_{\mathrm{null}}$ and cuts C is used. We use an arbitrary distribution (categorical) to encode the number of dictionary subgraphs b_{dict}. The indices $\mathcal{H}_{\mathrm{dict}}$ mapping the dictionary subgraphs to dictionary atoms are encoded with a Multinomial as in Eq. (5.15). Again, the number of blocks in the partition does not need to be included in the encoding since n and k are fixed. The overall encoding length is

$$\mathrm{L}^{\mathrm{PnC}}(G) = \mathrm{L}(b_{\mathrm{dict}}; \boldsymbol{\varphi}) + \mathrm{L}(\mathcal{H}_{\mathrm{dict}} \mid b_{\mathrm{dict}}; \boldsymbol{\varphi}) + \mathrm{L}(\mathcal{H}_{\mathrm{null}} \mid b_{\mathrm{null}}; \boldsymbol{\varphi}) + \mathrm{L}(C \mid \mathcal{H}; \boldsymbol{\varphi}), \tag{C.8}$$

The dictionary atoms are encoded as if they were sampled independently from any of the null models mentioned in the "conventional graph encodings" paragraph, hence $\mathrm{L}(\mathcal{D}) = O(|\mathcal{D}|k^2)$.

[2] To make the analysis more tangible, we examine here a slightly simpler encoding than the one used in the experiments - see Eq. (C.12).

C.1.3 Proof of Theorem 5.2.a: Why partitioning? Partitioning vs Null Models

As a warm-up, we will discuss the case of encodings based only on graph partitioning, such as the Stochastic Block Model ("Partitioning non-parametric" in the Tables 5.1 and 5.2). We remind the reader that these encodings do *not* take into account the isomorphism class of the identified subgraphs and cannot be adapted to different subgraph distributions. In the following, we derive a sufficient condition for the sparsity of the connections between subgraphs, under which partitioning-based encodings will yield a smaller expected description length than the baseline null models. Formally:

Theorem 1a. *Let every $G \sim p$ be partitioned into b blocks of $k = O(1)$ vertices and suppose that the partitioning-based encoding of Eq. (C.7) is utilised. The following holds:*

$$\mathbb{E}_{G\sim p}[\mathrm{L}^{\mathrm{Part}}(G)] \lesssim \mathbb{E}_{G\sim p}[\mathrm{L}^{\text{ER-S}}(G)] - n^2\Big(\bar{\mathrm{H}}_m - \frac{\log(k^2+1)}{k^2} - \bar{\mathrm{H}}_{m_{ij}}\Big), \tag{C.9}$$

where $\bar{\mathrm{H}}_{m_{ij}} := \frac{1}{b^2-b}\sum_{i\neq j}^{b}\mathbb{E}_{G\sim p}[\mathrm{H}(\frac{m_{ij}}{k^2})]$ and $\bar{\mathrm{H}}_m := \mathbb{E}_{G\sim p}[\mathrm{H}(\frac{m}{n^2})]$ are the expected binary entropy of the cuts (averaged over all subgraph pairs) and of the total number of edges, respectively.

Proof.

$$\begin{aligned}
\mathbb{E}_{G\sim p}[\mathrm{L}^{\mathrm{part}}(G)] &\approx \mathbb{E}_{G\sim p}\Big[\sum_{i=1}^{b}\Big(\log(k^2+1) + k^2\mathrm{H}(\frac{m_i}{k^2})\Big) + \sum_{i\neq j}^{b}\Big(\log(k^2+1) + k^2\mathrm{H}(\frac{m_{ij}}{k^2})\Big)\Big] \\
&= \frac{n}{k}\Big(\log(k^2+1) + k^2\bar{\mathrm{H}}_{m_i}\Big) + (\frac{n^2}{k^2} - \frac{n}{k})\Big(\log(k^2+1) + k^2\bar{\mathrm{H}}_{m_{ij}}\Big) \\
&= n^2\Big(\frac{\log(k^2+1)}{k^2} + \bar{\mathrm{H}}_{m_{ij}}\Big) + nk\Big(\bar{\mathrm{H}}_{m_i} - \bar{\mathrm{H}}_{m_{ij}}\Big) \\
&= \mathbb{E}_{G\sim p}[\mathrm{L}^{\text{ER-S}}(G)] - n^2\Big(\bar{\mathrm{H}}_m - \frac{\log(k^2+1)}{k^2} - \bar{\mathrm{H}}_{m_{ij}}\Big) \\
&+ nk\Big(\bar{\mathrm{H}}_{m_i} - \bar{\mathrm{H}}_{m_{ij}} + \log n\Big) - \log(n^2+1) \\
&\lesssim \mathbb{E}_{G\sim p}[\mathrm{L}^{\text{ER-S}}(G)] - n^2\Big(\bar{\mathrm{H}}_m - \frac{\log(k^2+1)}{k^2} - \bar{\mathrm{H}}_{m_{ij}}\Big),
\end{aligned}$$

where $\bar{\mathrm{H}}_{m_i} = \frac{1}{b}\sum_{i=1}^{b}\mathbb{E}_{G\sim p}[\mathrm{H}(\frac{m_i}{k^2})]$ and in the last step we derive an asymptotic inequality using the dominating quadratic term. In other words, partitioning-based encoding is quadratically

superior to the best null model whenever there exists a k such that

$$\frac{\log(k^2+1)}{k^2} < \bar{\mathrm{H}}_m - \bar{\mathrm{H}}_{m_{ij}}.$$

The above concludes the proof. □

C.1.4 Proof of Theorem 5.2.b: The importance of the dictionary: PnC vs Partitioning

We proceed to mathematically justify why encoding subgraphs with a dictionary can yield extra compression gains compared to pure partitioning-based encodings. As our main theorem shows, utilising a dictionary allows us to reduce the linear $O(n)$ terms of the partitioning-based description length:

Theorem 1b. *Let every $G \sim p$ be partitioned into b blocks of $k = O(1)$ vertices and suppose that the PnC encoding of Eq. (C.8) is used. Assume that the dictionary-subgraph encoding used by PnC is optimal for the distribution p. Let $\bar{\mathrm{H}}_{m_i} = \frac{1}{b}\sum_{i=1}^{b} \mathbb{E}_{G\sim p}[\mathrm{H}(\frac{m_i}{k^2})]$, $\bar{\mathrm{H}}^{\text{null}}_{m_i} = \frac{1}{b\cdot\delta_{\text{true}}}\mathbb{E}_{G\sim p}[\sum_{H_i\in\mathcal{H}_{\text{null}}} \mathrm{H}(\frac{m_i}{k^2})]$ be the expected binary entropy of the subgraph edges averaged over all subgraphs and over non-dictionary subgraphs respectively, and $\delta_{\text{true}} = \frac{\mathbb{E}_{G\sim p}[b_{\text{null}}]}{b}$ the expected percentage of non-dictionary subgraphs. For every dictionary for which the following holds:*

$$\log|\mathcal{D}| < \log(k^2+1) + k^2\frac{\bar{\mathrm{H}}_{m_i} - \delta_{\text{true}}\bar{\mathrm{H}}^{\text{null}}_{m_i}}{1-\delta_{\text{true}}} \tag{C.10}$$

then an optimal PnC compressor will achieve linear gains compared to its partitioning-based counterpart :

$$\mathbb{E}_{G\sim p}[\mathrm{L}^{\text{PnC}}(G)] \lesssim \mathbb{E}_{G\sim p}[\mathrm{L}^{\text{part}}(G)] - nk(1-\delta_{\text{true}})\left(\frac{\bar{\mathrm{H}}_{m_i} - \delta_{\text{true}}\bar{\mathrm{H}}^{\text{null}}_{m_i}}{1-\delta_{\text{true}}} - \frac{\mathbb{H}(\mathcal{D}) - \log(k^2+1)}{k^2}\right), \tag{C.11}$$

where $\mathbb{H}(\mathcal{D})$ is the entropy of the distribution q over the dictionary atoms.

Proof. We will analyse the description length of each of the components of Eq. (C.8).

Number of dictionary subgraphs. Assuming that our PnC compressor operates in an optimal regime, the expected description length of the number of subgraphs is approximately

equal to the true entropy of the distribution of the number of dictionary subgraphs:

$$\mathbb{E}_{G\sim p}[\mathrm{L}(b_{\text{dict}};\boldsymbol{\varphi})] = \mathbb{E}_{G\sim p}[-\log q(b_{\text{dict}};\boldsymbol{\varphi})] = \mathbb{E}_{G\sim p}[-\log p(b_{\text{dict}};\boldsymbol{\varphi})] \leq \log(b+1) = \log\left(\frac{n}{k}+1\right)$$

Dictionary subgraphs. Here we will need a stronger assumption to proceed. In particular, we will need to assume that the dictionary subgraph encoding is optimal, i.e. that the distribution of the dictionary subgraphs is a Multinomial (which is the same regardless of the value of b_{dict}). Then, if PnC has converged to its optimal form we can get the following, for the expected number of dictionary subgraphs mapped to atom α:

$$\begin{aligned}\mathbb{E}_{G\sim p}[b_\alpha] &= \sum_{b_{\text{dict}}=0}^{b} \mathbb{E}_{b_\alpha\sim p(b_a|b_{\text{dict}})}[b_\alpha \mid b_{\text{dict}}]p(b_{\text{dict}}) = \sum_{b_{\text{dict}}=0}^{b} p(\alpha)b_{\text{dict}}p(b_{\text{dict}}) \\ &= p(\alpha)\mathbb{E}_{G\sim p}[b_{\text{dict}}] = q(\alpha)b(1-\delta_{\text{true}})\end{aligned}$$

Now we can upper bound the expected description length of the dictionary subgraphs as follows:

$$\begin{aligned}\mathbb{E}_{G\sim p}[\mathrm{L}(\mathcal{H}_{\text{dict}} \mid b_{\text{dict}};\boldsymbol{\varphi})] &= \mathbb{E}_{G\sim p}\left[-\log\frac{b_{\text{dict}}!}{\prod_{\alpha\in\mathcal{D}} b_\alpha!} - \sum_{\alpha\in\mathcal{D}} b_\alpha \log q(\alpha)\right] \\ &\leq \mathbb{E}_{G\sim p}[-\sum_{\alpha\in\mathcal{D}} b_\alpha \log q(\alpha)] \\ &= -\sum_{\alpha\in\mathcal{D}} \mathbb{E}_{G\sim p}[b_\alpha]\log q(\alpha) \\ &= -b(1-\delta_{\text{true}})\sum_{\alpha\in\mathcal{D}} q(\alpha)\log q(\alpha) = \frac{n}{k}(1-\delta_{\text{true}})\mathbb{H}(\mathcal{D})\end{aligned}$$

where we used the fact that $b_{\text{dict}}! \geq \prod_{\alpha\in\mathcal{D}} b_\alpha!$ and $\mathbb{H}(\mathcal{D}) = \mathbb{H}_{\alpha\sim q(\alpha)}[\alpha] = -\sum_{\alpha\in\mathcal{D}} q(\alpha)\log q(\alpha)$

Non-dictionary subgraphs.

$$\begin{aligned}
\mathbb{E}_{G\sim p}[\mathrm{L}(\mathcal{H}_{\text{null}} \mid b_{\text{null}}; \boldsymbol{\varphi})] &= \mathbb{E}_{G\sim p}\left[\sum_{H_i\in\mathcal{H}_{\text{null}}}\left(\log(k^2+1) + k^2\mathrm{H}\Big(\frac{m_i}{k^2}\Big)\right)\right] \\
&= \log(k^2+1)\,\mathbb{E}_{G\sim p}\left[\sum_{H_i\in\mathcal{H}_{\text{null}}} 1\right] + k^2\,\mathbb{E}_{G\sim p}\left[\sum_{H_i\in\mathcal{H}_{\text{null}}} \mathrm{H}\Big(\frac{m_i}{k^2}\Big)\right] \\
&= \log(k^2+1)\,\mathbb{E}_{G\sim p}\left[b_{\text{null}}\right] + k^2\,\mathbb{E}_{G\sim p}\left[\sum_{H_i\in\mathcal{H}_{\text{null}}} \mathrm{H}\Big(\frac{m_i}{k^2}\Big)\right] \\
&= \log(k^2+1) b\delta_{\text{true}} + k^2\bar{\mathrm{H}}^{\text{null}}_{m_i} b\delta_{\text{true}} \\
&= \frac{n}{k}\delta_{\text{true}}\Big(\log(k^2+1) + k^2\bar{\mathrm{H}}^{\text{null}}_{m_i}\Big),
\end{aligned}$$

where $\bar{\mathrm{H}}^{\text{null}}_{m_i} = \frac{1}{b\cdot\delta_{\text{true}}}\mathbb{E}_{G\sim p}[\sum_{H_i\in\mathcal{H}_{\text{null}}} \mathrm{H}_{m_i}]$.

Cuts. Re-using the derivation of Theorem 1a we have:

$$\mathbb{E}_{G\sim p}[\mathrm{L}(C \mid \mathcal{H}; \boldsymbol{\varphi})] = n^2\left(\frac{\log(k^2+1)}{k^2} + \bar{\mathrm{H}}_{m_{ij}}\right) - n\left(\frac{\log(k^2+1)}{k} + k\bar{\mathrm{H}}_{m_{ij}}\right)$$

Overall. Putting everything together we get:

$$\begin{aligned}
\mathbb{E}_{G\sim p}[\mathrm{L}^{\text{PnC}}(G)] &\lessapprox \log\left(\frac{n}{k}+1\right) + n^2\left(\frac{\log(k^2+1)}{k^2} + \bar{\mathrm{H}}_{m_{ij}}\right) \\
&\quad + n\left(k\delta_{\text{true}}\bar{\mathrm{H}}^{\text{null}}_{m_i} + (1-\delta_{\text{true}})\left(\frac{\mathbb{H}(\mathcal{D}) - \log(k^2+1)}{k}\right) - k\bar{\mathrm{H}}_{m_{ij}}\right) \\
&= \mathbb{E}_{G\sim p}[\mathrm{L}^{\text{part}}(G)] + nk\left(\delta_{\text{true}}\bar{\mathrm{H}}^{\text{null}}_{m_i} - \bar{\mathrm{H}}_{m_i} + (1-\delta_{\text{true}})\Big(\frac{\mathbb{H}(\mathcal{D}) - \log(k^2+1)}{k^2}\Big)\right) \\
&\quad + \log\left(\frac{n}{k}+1\right)
\end{aligned}$$

Including the description length of the dictionary and amortising it over each graph in a dataset of $\mathfrak{D}$ graphs, we conclude

$$\begin{aligned}
\mathbb{E}_{G\sim p}[\mathrm{L}^{\text{PnC}}(G)] &= \mathbb{E}_{G\sim p}[\mathrm{L}^{\text{part}}(G)] - nk(1-\delta_{\text{true}})\left(\frac{\bar{\mathrm{H}}_{m_i} - \delta_{\text{true}}\bar{\mathrm{H}}^{\text{null}}_{m_i}}{1-\delta_{\text{true}}} - \frac{\mathbb{H}(\mathcal{D}) - \log(k^2+1)}{k^2}\right) \\
&\quad + O\Big(\log\frac{n}{k} + \frac{|\mathcal{D}|}{|\mathfrak{D}|}k^2\Big).
\end{aligned}$$

Hence, if $k = O(1)$ and $|\mathcal{D}| \ll |\mathfrak{D}|$ (more precisely, the ratio $\frac{|\mathcal{D}|}{|\mathfrak{D}|}$ shouldn't grow with n), then a

linear compression gain is obtained if $1 - \delta_{\text{true}} > 0$ and:

$$\frac{\bar{\mathrm{H}}_{m_i} - \delta_{\text{true}}\bar{\mathrm{H}}^{\text{null}}_{m_i}}{1 - \delta_{\text{true}}} - \frac{\mathbb{H}(\mathcal{D}) - \log(k^2+1)}{k^2} > 0 \iff \mathbb{H}(\mathcal{D}) < \log(k^2+1) + k^2\frac{\bar{\mathrm{H}}_{m_i} - \delta_{\text{true}}\bar{\mathrm{H}}^{\text{null}}_{m_i}}{1 - \delta_{\text{true}}}$$

The proof concludes by recalling that $\mathbb{H}(\mathcal{D}) < \log|\mathcal{D}|$. □

C.1.5 Proof of Theorem 5.3: The importance of subgraph isomorphism

Proof. In the context of this comparison, we are only interested in the description length of the dictionary subgraphs. For simplicity, we will assume that these are encoded with a *categorical* distribution instead of a multinomial and we will make the same assumption as in Theorem 1b (i.e. that the dictionary-subgraph encoding is optimal).

$$\mathbb{E}_{G\sim p}[\mathrm{L}(\mathcal{H}_{\text{dict}}|b_{\text{dict}})] = \mathbb{E}_{G\sim p}\left[-\sum_{\alpha\in\mathcal{D}} b_\alpha \log q(\alpha)\right] = \frac{n}{k}(1-\delta_{\text{true}})\mathbb{H}(\mathcal{D})$$

Hence, in order to compare the two variants, we are interested in the entropy $\mathbb{H}(\mathcal{D})$, which requires enumerating the possible outcomes of the categorical distribution, i.e., the dictionary atoms.

Denote with $q_G(\alpha), q_S(\alpha)$ the categorical distributions used by PnC-G and PnC-S respectively, and (with a slight abuse of notation) with $\mathbb{H}_G(\mathcal{D}) = \mathbb{H}_{\alpha\sim q_G(\alpha)}[\alpha]$, $\mathbb{H}_S(\mathcal{D}) = \mathbb{H}_{\alpha\sim q_S(\alpha)}[\alpha]$ their corresponding entropies. Let $\mathcal{D}_G$ be the optimal dictionary of PnC-G and $\mathcal{D}_S$ the optimal dictionary of PnC-S. For simplicity, we will assume that $\mathcal{D}_G = \mathfrak{G}_k$, i.e. it contains the adjacency matrices of all labelled graphs with k vertices, while $\mathcal{D}_S = \mathfrak{G}/\simeq$, or more precisely it contains only the adjacency matrix of one labelled representative from each equivalence class.

Regarding PnC-G, since $p(G)$ is isomorphism-invariant, then the same will hold for the subgraphs $H \subseteq G$, i.e. $p(H') = p(H)$ if $H' \simeq H$. Hence, for the optimal $q_G(\alpha)$ it should hold that for each atom α in $\mathcal{D}$, then all $\alpha' \cong \alpha$ will be also contained in the dictionary and assigned the same probability, i.e., $q_G(\alpha) = q_G(\alpha') = p_G(\alpha) = p_G(\alpha')$.

Now regarding PnC-S, the probability of the equivalence class of an atom $\mathrm{Orb}(\alpha)$ will be $p_S\big(\mathrm{Orb}(\alpha)\big) = \sum_{\alpha\in\mathrm{Orb}(\alpha)} p_G(\alpha) = |\mathrm{Orb}(\alpha)|p_G(\alpha)$, and therefore the same will also hold for the optimal model $q_S\big(\mathrm{Orb}(\alpha)\big) = |\mathrm{Orb}(\alpha)|p_G(\alpha)$. Recall that $|\mathrm{Orb}(\alpha)| = \frac{k!}{|\mathrm{Aut}(\alpha)|}$ [Harary and Palmer, 2014].

Then, using a similar argument to [Choi and Szpankowski, 2012], we can derive the following for the entropy $\mathbb{H}_G(\mathcal{D})$:

$$\begin{aligned}
\mathbb{H}_G(\mathcal{D}) &= -\sum_{\alpha\in\mathcal{D}_G} q_G(\alpha)\log q_G(\alpha) \\
&= -\sum_{\alpha\in\mathcal{D}_G} \frac{q_S\big(\mathrm{Orb}(\alpha)\big)}{|\mathrm{Orb}(\alpha)|}\log\frac{q_S\big(\mathrm{Orb}(\alpha)\big)}{|\mathrm{Orb}(\alpha)|} \\
&= -\sum_{\mathrm{Orb}(\alpha)\in\mathcal{D}_S}\sum_{\alpha\in\mathrm{Orb}(\alpha)} \frac{q_S\big(\mathrm{Orb}(\alpha)\big)}{|\mathrm{Orb}(\alpha)|}\log\frac{q_S\big(\mathrm{Orb}(\alpha)\big)}{|\mathrm{Orb}(\alpha|} \\
&= -\sum_{\mathrm{Orb}(\alpha)\in\mathcal{D}_S} q_S\big(\mathrm{Orb}(\alpha)\big)\log\frac{q_S\big(\mathrm{Orb}(\alpha)\big)}{|\mathrm{Orb}(\alpha)|} \\
&= \mathbb{H}_S(\mathcal{D}) + \sum_{\mathrm{Orb}(\alpha)\in\mathcal{D}_S} q_S\big(\mathrm{Orb}(\alpha)\big)\log|\mathrm{Orb}(\alpha)| \\
&= \mathbb{H}_S(\mathcal{D}) + \log k! - \sum_{\mathrm{Orb}(\alpha)\in\mathcal{D}_S} q_S\big(\mathrm{Orb}(\alpha)\big)\log|\mathrm{Aut}(\alpha)|
\end{aligned}$$

At this point we will assume that almost all graphs in the dictionary are rigid, or more precisely we require that $\sum_{\mathrm{Orb}(\alpha)\in\mathcal{D}_S} q_S\big(\mathrm{Orb}(\alpha)\big)\log|\mathrm{Aut}(\alpha)| \approx 0$, which can be also satisfied when non-rigid dictionary atoms have a small probability. In practice, although for very small graphs of up to 4 or 5 vertices, many graphs have non-trivial automorphisms, this condition is easily satisfied for larger k (but still of constant size w.r.t. n), that were also considered in practice. Then, the result immediately follows:

$$\begin{aligned}
\mathbb{E}_{G\sim p}[\mathrm{L}^{\text{PnC-G}}(\mathcal{H}_{\text{dict}} \mid b_{\text{dict}})] \approx \mathbb{E}_{G\sim p}[\mathrm{L}^{\text{PnC-S}}(\mathcal{H}_{\text{dict}} \mid b_{\text{dict}})] + \frac{n}{k}(1-\delta_{\text{true}})\log k! \Longrightarrow \\
\mathbb{E}_{G\sim p}[\mathrm{L}^{\text{PnC-S}}(G)] \approx \mathbb{E}_{G\sim p}[\mathrm{L}^{\text{PnC-G}}(G)] - n(1-\delta_{\text{true}})\log k,
\end{aligned}$$

where we used Stirling's approximation $\log k! \approx k\log k$. □

C.2 Algorithmic Details

C.2.1 Baseline Encodings

Clustering. The encoding we used for the clustering baselines is optimal for labelled graphs under SBM assumptions and is obtained from [Peixoto, 2019] with small modifications. It

consists of the following uniform encodings: number of graph vertices, number of graph edges, number of blocks, number of vertices in each block, number of edges inside each block and between each pair of blocks, and finally the arrangements of intra- and inter-block edges (a detailed explanation for each term can be found in [Peixoto, 2019] and [Peixoto, 2014]):

$$\begin{aligned}\mathrm{L}(G) &= \log(n_{\max}+1) + \log\left(n(n-1)/2+1\right) + \log(n) + \log\binom{n-1}{b-1} \\ &+ \log\binom{b(b+1)/2+m-1}{m} + \sum_{i=1}^{b}\log\binom{\binom{k_i}{2}}{m_i} + \sum_{i<j}^{b}\log\binom{k_ik_j}{m_{ij}}\end{aligned} \tag{C.12}$$

C.2.2 Dictionary Learning - Continuous Relaxation

In the following section, we will relax the Minimum Description Length objective of Eq. (5.19) by introducing the fractional membership variables $\hat{\boldsymbol{x}}$. The dictionary description length, Eq. (5.18), can be trivially rewritten as follows:

$$\mathrm{L}(\mathcal{D};\hat{\boldsymbol{x}}) = \sum_{\alpha\in\mathfrak{U}} \hat{x}_\alpha\, \mathrm{L}^{\mathrm{null}}(\alpha). \tag{C.13}$$

Regarding the description length of the graphs, the membership variables are the ones that select when a subgraph is encoded as a dictionary atom or when with the help of the null model. The relaxation of the graph description length was done with the following modifications $b_\alpha(\hat{\boldsymbol{x}}) = \hat{x}_\alpha b_\alpha$, $b_{\mathrm{dict}}(\hat{\boldsymbol{x}}) = \sum_{\alpha\in\mathfrak{U}} b_\alpha(\hat{\boldsymbol{x}})$, and $q(\alpha;\hat{\boldsymbol{x}}) = \frac{\hat{x}_\alpha e^{\zeta_\alpha}}{\sum_{\alpha'\in\mathfrak{U}}\hat{x}_{\alpha'}e^{\zeta_{\alpha'}}}$, where $\zeta_a \in \mathbb{R}$ are learnable parameters. The rest of the components of the graph description length are unaffected by the choice of the dictionary. Now Eq. (5.14)-(5.16) can be rewritten as:

$$\begin{aligned}
&\mathrm{L}(b_{\mathrm{dict}}, b; \boldsymbol{\phi}, \hat{\boldsymbol{x}}) = -\log\binom{b}{b_{\mathrm{dict}}(\hat{\boldsymbol{x}})} - b_{\mathrm{dict}}(\hat{\boldsymbol{x}})\log(1-\delta) - \left(b - b_{\mathrm{dict}}(\hat{\boldsymbol{x}})\right)\log(\delta) - \log q(b) \\
&\mathrm{L}(\mathcal{H}_{\mathrm{dict}} \mid b_{\mathrm{dict}}; \boldsymbol{\varphi}, \hat{\boldsymbol{x}}) = -\log\left(b_{\mathrm{dict}}(\hat{\boldsymbol{x}})!\right) + \sum_{\alpha\in\mathfrak{U}}\log\left(b_\alpha(\hat{\boldsymbol{x}})!\right) - \sum_{\alpha\in\mathfrak{U}} b_\alpha(\hat{\boldsymbol{x}})\log q(\alpha;\hat{\boldsymbol{x}}) \\
&\mathrm{L}_(\mathcal{H}_{\mathrm{null}} \mid b_{\mathrm{null}}; \hat{\boldsymbol{x}}) = -\sum_{H\in\mathcal{H}}\log q_{\mathrm{null}}(H)(1-\hat{x}_H), \text{ where } \hat{x}_H = \begin{cases} \hat{x}_i & \exists\, a_i \in \mathfrak{U} \text{ s.t. } H \cong a_i \\ 0 & \text{otherwise.}\end{cases}
\end{aligned} \tag{C.14}$$

To obtain a continuous version of the terms where factorials are involved we used the Γ function, where $\Gamma(n+1) = n!$, for positive integers n. The rest of the terms are differentiable w.r.t $\hat{\boldsymbol{x}}$.

C.2.3 Learning to Partition

We remind that our algorithm is based on a double iterative procedure: the external iteration refers to subgraph selection and the internal to vertex selection. In order for the algorithm to be able to make decisions, we maintain a representation of two states: the *subgraph state* $S_t^H = (H_1, H_2, \dots H_t)$ that summarises the decisions made at the subgraph level (external iteration) up to step t, and the *vertex state* $S_{t,i}^V = (v_{t,1}, v_{t,2}, \dots v_{t,i})$ that summarises the decisions made at the vertex level (internal iteration) up to the i-th vertex selection during step t. Overall, we need to calculate the probability of S_T^H, where T is the number of iterations:

$$\begin{aligned} p(S_T^H \mid G; \boldsymbol{\theta}) &= p(H_T \mid S_{T-1}^H, G; \boldsymbol{\theta}) p(S_{T-1}^H \mid G; \boldsymbol{\theta}) = \prod_{t=1}^{T} p(H_t | S_{t-1}^H, G; \boldsymbol{\theta}) \\ &= \prod_{t=1}^{T} \Bigg(\prod_{i=1}^{k_t} p(v \mid S_{t,i-1}^V, k_t, S_{t-1}^H, G; \boldsymbol{\theta}) \Bigg) p(k_t \mid S_{t-1}^H, G; \boldsymbol{\theta}) \end{aligned} \tag{C.15}$$

Hence, the parametrisation of the algorithm boils down to defining the vertex count probability $p(k_t \mid S_{t-1}^H, G; ; \boldsymbol{\theta})$ and the vertex selection probability $p(v \mid S_{t,i-1}^V, k_t, S_{t-1}^H, G; \boldsymbol{\theta})$, where $v \in \mathcal{V}_t$ and $\mathcal{V}_t$ the set of the remaining vertices at step t.

Now we explain in detail how we parametrise each term. First, we use a GNN to embed each vertex into a vector representation $\mathbf{h}(v) = \text{GNN}_v(G)$, while the graph itself is embedded in a similar way $\mathbf{h}(G) = \text{GNN}_G(G)$. Each subgraph is represented by a permutation invariant function on the embeddings of its vertices, i.e., $\mathbf{h}(H_t) = \text{DeepSets}\Big(\{\mathbf{h}(v) \mid v \in H_t\}\Big)$, where we used DeepSets [Zaheer et al., 2017] as a set function approximator. Similarly, the subgraph state summarises the subgraph representations in a permutation invariant manner to ensure that future decisions of the algorithm do not depend on the order of the past ones: $\mathbf{h}(S_t^H) = \text{DeepSets}\Big(\{\mathbf{h}(H_t) \mid H_t \in S_t^H\}\Big)$.

Given the above, the probability of the vertex count at step t is calculated as follows:

$$p(k_t \mid S_{t-1}^H, G; \boldsymbol{\theta}) = \text{softmax}_{k_t=1}^{|\mathcal{V}_t|} \text{MLP}\Big(\mathbf{h}(S_{t-1}^H), \mathbf{h}(G)\Big), \tag{C.16}$$

where MLP is a multi-layer perceptron. As already mentioned, the probability of the selection

of each vertex is computed in a way that guarantees connectivity:

$$p(v \mid S^V_{t,i-1}, k_t, S^H_{t-1}, G; \boldsymbol{\theta}) = \begin{cases} \operatorname*{softmax}_{v \in \mathcal{V}_t} \mathrm{MLP}\Big(\mathbf{h}(v), \mathbf{h}(S^H_{t-1})\Big) & i = 0, v \in \mathcal{V}_t \\ \operatorname*{softmax}_{v \in \mathcal{V}_t \cap \mathcal{N}(S^V_{t,i-1})} \mathrm{MLP}\Big(\mathbf{h}(v), \mathbf{h}(S^H_{t-1})\Big) & 0 < i < k_t, v \in \mathcal{V}_t \cap \mathcal{N}(S^V_{t,i-1}) \\ 0 & \text{otherwise,} \end{cases} \tag{C.17}$$

where $\mathcal{N}(S^V_{t,i-1})$ denotes the union of the neighbourhoods of the already selected vertices, excluding themselves: $\mathcal{N}(S^V_{t,i-1}) = \bigcup_{i'=1}^{i-1} \mathcal{N}(v_{t,i'}) - \{v_{t,1}, v_{t,2}, \dots v_{t,i-1}\}$, and $\mathcal{N}(S^V_{t,0}) = \mathcal{V}_t$. Overall, the parameter set $\boldsymbol{\theta}$ is the set of the parameters of the neural networks involved, i.e., GNNs, DeepSets and MLPs. In the algorithm 1 we schematically illustrate the different steps described above.

Algorithm 1: Partitioning algorithm

```
Input: graph G = (V, E)
Output: partition H
Initialisations: h(v) = GNN_v(G), h(G) = GNN_G(G), V_1 = V, S_0^H = ∅
t ← 1
while V_t ≠ ∅ do
    k_t ~ p(k_t | S_{t-1}^H, G; θ) // sample maximum vertex count
    Initialise S_{t,0}^V = ∅
    while i = 1 ≤ k_t and N(S_{t,i-1}^V) ≠ ∅ do
        v_{t,i} ~ p(v | S_{t,i-1}^V, k_t, S_{t-1}^H, G; θ) // sample new vertex
        S_{t,i}^V = S_{t,i-1}^V ∪ {v_{t,i}}
    end
    H_t = S_{t,i}^V
    S_t^H = S_{t-1}^H ∪ {H_t}
    t ← t + 1
end
H = S_t^H
```

Limitations. Below we list two limitations of the learnable partitioning algorithm that we would like to address in future work. First, it is well known that GNNs have limited expressivity which is bounded by the Weisfeiler Leman test [Xu et al., 2019, Morris et al., 2019]. The most important implication of this is that they have difficulties in detecting and counting substructures [Chen et al., 2020]. Since in our case, subgraph detection is crucial in order to be able to partition the graph into repetitive substructures, the expressivity of the GNN might be an issue. Although iterative sampling may mitigate this behaviour up to a certain extent, the GNN will not be able to express arbitrary randomised algorithms. Modern architectures such

as [Vignac et al., 2020, Bouritsas et al., 2022] might be more suitable for this task, which makes them good candidates for future exploration on the problem.

Second, more sophisticated inference schemes should be explored, since currently a partition is decoded from the randomised algorithm by taking a single sample from the learned distribution. In particular, currently at each step t the algorithm can only sample k_t vertices as dictated by the initial sampling on the vertex count. However, there might be benefits from expanding the subgraph more or stopping earlier than k_t when no other vertex addition can contribute towards a smaller description length. However, there is no control on the stopping criterion apart from the initial vertex count prediction. To this end, it is of interest to explore alternatives that will allow the algorithm to choose from a pool of candidate decisions based on the resulting description lengths (i.e. in hindsight). Further inspiration can be taken from a variety of clustering and graph partitioning algorithms, e.g. k-means or soft clustering in a latent space [Wilder et al., 2019, Locatello et al., 2020], agglomerative [Karger, 1993, Blondel et al., 2008] and Markov Chain Monte Carlo as in [Peixoto, 2013] where a modified Metropolis-Hastings algorithm is proposed.

Special cases of note. A pertinent question is whether one can determine the optimal way to partition a graph when minimising (5.19). Though a rigorous statement is beyond our current understanding, in the following we discuss two special cases:

(a) Small predefined universe. When the subgraphs are chosen from a small and predefined $\mathfrak{U}$, one may attempt to identify all the possible atom appearances in G by repeatedly calling a subgraph isomorphism subroutine. The minimisation of (5.19) then simplifies to that of selecting a subset of subgraphs that have no common edges (as per the definition in section 5.4.1). The latter problem can be cast as a discrete optimisation problem under an independent set constraint (by building an auxiliary graph the vertices of which are candidate subgraphs and two vertices are connected by an edge when two subgraphs overlap and looking for an independent set that minimises the description length).

(b) Unconstrained universe. When $\mathfrak{U}$ contains all possible graphs, the problem can be seen as a special graph partitioning problem. However, contrary to traditional clustering algorithms [Ng et al., 2001, Karypis and Kumar, 1998, Blondel et al., 2008, Karger, 1993], our objective is not necessarily optimised by finding small cuts between clusters (see section 5.5).

Since most independent set and partitioning problems are NP-hard, we suspect that similar arguments can be put forward for (5.19). This highlights the need to design parametric (learnable) alternatives that can provide fast solutions without the need to be adapted to unseen data.

C.2.4 Construction of the graph encoder-decoder functions

In the following section we will showcase the exact construction of the graph encoder and decoder functions $h_{\text{g-enc}}, h_{\text{g-dec}}$. To show that $h_{\text{g-enc}}$ is GI-injective it suffices to show that there exists a decoder function $h_{\text{g-dec}}$ such that $h_{\text{g-dec}}\big(h_{\text{g-enc}}(G)\big) \simeq G$, $\forall G \in \mathfrak{G}$. For illustration purposes, it will be more convenient to present our results using the adjacency matrix representation. First, we show why partitioning allows us to define a GI-injective function.

Remark C.1. *Let $G = (\mathcal{V}, \mathcal{E}) \in \mathfrak{G}$ be a graph with adjacency matrix $\mathbf{A}(G) \in \{0,1\}^{|\mathcal{V}|\times|\mathcal{V}|}$ and a partitioning algorithm that partitions the graph as follows: $\mathsf{PART}(G) = \Big(\big(H_1, H_2, \cdots, H_b\big), C\Big)$. Denote the (extended) adjacency matrix of a subgraph H_j with $\mathbf{A}(H_j) \in \{0,1\}^{|\mathcal{V}|\times|\mathcal{V}|}$,[3] i.e. $\mathbf{A}(H_j)(u,v) = 1$ iff $(u,v) \in \mathcal{E}_{H_i}$, $\forall u, v \in \mathcal{V}$. Similarly, denote with $\mathbf{A}(C) \in \{0,1\}^{|\mathcal{V}|\times|\mathcal{V}|}$ the extended adjacency of the b-partite cut graph, i.e. $\mathbf{A}(C)(u,v) = 1$ iff $(u,v) \in \mathcal{E}_C$, $\forall u, v \in \mathcal{V}$. Then, there exists a permutation matrix $\mathbf{P}_{part} \in S(|\mathcal{V}|)$, such that:*

$$\mathbf{A}' = \mathbf{P}_{part}\mathbf{A}(G)\mathbf{P}_{part}^\top = \mathbf{H}+\mathbf{C} = \sum_{j=1}^{b}\mathbf{H}_j+\mathbf{C} = \begin{bmatrix} \mathbf{A}_{1,1} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{A}_{2,2} & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{A}_{b,b} \end{bmatrix} + \begin{bmatrix} \mathbf{0} & \mathbf{A}_{1,2} & \cdots & \mathbf{A}_{1,b} \\ \mathbf{A}_{2,1} & \mathbf{0} & \cdots & \mathbf{A}_{2,b} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{A}_{b,1} & \mathbf{A}_{b,2} & \cdots & \mathbf{0} \end{bmatrix}, \tag{C.18}$$

with $\mathbf{P}_{part}\mathbf{A}(H_j)\mathbf{P}_{part}^\top = \mathbf{H}_j$ and $\mathbf{P}_{part}\mathbf{A}(C)\mathbf{P}_{part}^\top = \mathbf{C}$, where $\mathbf{H}_j$ is a block-diagonal matrix with only one non-zero block $\mathbf{A}_{j,j}$ in the position j and $\mathbf{C}$ is a block matrix with zero blocks in the main diagonal. Then, a graph encoder function

$$h_{g\text{-}enc}(G) = \Big(\mathbf{A}_{1,1}, \ldots, \mathbf{A}_{b,b}, \mathbf{A}_{1,2}, \ldots, \mathbf{A}_{b-1,b}\Big)$$

is a GI-injective function.

To see the above, recall that the vertex sets are disjoint. Therefore, we can constructively define a permutation matrix as follows: first, we map the vertices of $\mathcal{V}_{H_1}$ to $\{1, \ldots, |\mathcal{V}_{H_1}|\}$ with an arbitrary bijection, then we map the vertices of $\mathcal{V}_{H_2}$ to $\{|\mathcal{V}_{H_1}|+1, \ldots, |\mathcal{V}_{H_1}|+|\mathcal{V}_{H_2}|\}$ and so

[3] we slightly abused notation, w.r.t. the dimensions.

on and so forth. Applying the resulting permutation matrix to the original adjacency matrix will result in the form of in Eq. (C.18). It is obvious that this is a GI-injective function since one can define the graph decoder to simply be a function reconstructing the matrix $\mathbf{A}'$, i.e. $h_{\text{g-dec}}(h_{\text{g-enc}}(G)) = \mathbf{A}'$ which yields a graph which is by definition isomorphic to G. In other words, we can think of a partitioning algorithm as a procedure that first permutes and then decomposes the adjacency matrix as in Eq. (C.18).

Next, we show how one can construct a graph encoder with the help of a graph dictionary. For now, we will assume that *all* subgraphs can be mapped to exactly one dictionary atom.

Remark C.2. *Let $G = (\mathcal{V}, \mathcal{E}) \in \mathfrak{G}$ be a graph with adjacency matrix $\mathbf{A}(G) \in \{0,1\}^{|\mathcal{V}|\times|\mathcal{V}|}$ and a partitioning algorithm that partitions the graph as in Eq. (C.18). Let $\mathcal{D} = \{\alpha_1, \dots, \alpha_{|\mathcal{D}|}\}$ be a graph dictionary, where $\alpha_k \not\simeq \alpha_\ell$, $\forall k, \ell \in [|\mathcal{D}|], k \neq \ell$. Define the function $\psi_{\mathcal{D}}(\cdot)$ that maps each subgraph of the partition to a dictionary atom as follows: $\psi_{\mathcal{D}}(H) = i$, if $\exists\ i \in [|\mathcal{D}|]$, s.t. $\alpha_i \in\simeq H$, and define $i_j = \psi_{\mathcal{D}}(H_j)$. Then, there exist permutation matrices $\mathbf{P}_j$, such that $\mathbf{A}(\alpha_{i_j}) = \mathbf{P}_j \mathbf{A}_{j,j} \mathbf{P}_j^\top$. A GI-injective function can be defined as follows:*

$$h_{g\text{-}enc}(G; \mathcal{D}) = \Big(i_1, \dots, i_b, \mathbf{P}_1 \mathbf{A}_{1,2} \mathbf{P}_2^\top \dots \mathbf{P}_{b-1} \mathbf{A}_{b-1,b} \mathbf{P}_b^\top\Big) \tag{C.19}$$

Moreover, assume a permutation π of the indices $i_1, \dots, i_b$ such that $i_{\pi(1)} \leq i_{\pi(2)}, \cdots \leq i_{\pi(b)}$. Then, we can define an improved graph encoder function which is also GI-injective:

$$h_{g\text{-}enc}(G; \mathcal{D}) = \Big(\wr i_1, \dots, i_b \int, \mathbf{P}_{\pi(1)} \mathbf{A}_{\pi(1),\pi(2)} \mathbf{P}_{\pi(2)}^\top \dots, \mathbf{P}_{\pi(b-1)} \mathbf{A}_{\pi(b-1),\pi(b)} \mathbf{P}_{\pi(b)}^\top\Big) \tag{C.20}$$

The above is easy to see by defining a graph decoder that computes a graph isomorphic to the input. In the first case:

$$\begin{aligned}
h_{\text{g-dec}}(h_{\text{g-enc}}(G; \mathcal{D})) &= \begin{bmatrix} \mathbf{A}(\alpha_{i_1}) & \mathbf{P}_1 \mathbf{A}_{1,2} \mathbf{P}_2^\top & \cdots & \mathbf{P}_1 \mathbf{A}_{1,b} \mathbf{P}_b^\top \\ \mathbf{P}_2 \mathbf{A}_{2,1} \mathbf{P}_1^\top & \mathbf{A}(\alpha_{i_2}) & \cdots & \mathbf{P}_2 \mathbf{A}_{2,b} \mathbf{P}_b^\top \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{P}_b \mathbf{A}_{b,1} \mathbf{P}_1^\top & \mathbf{P}_b \mathbf{A}_{b,2} \mathbf{P}_2^\top & \cdots & \mathbf{A}(\alpha_{i_b}) \end{bmatrix} = \begin{bmatrix} \mathbf{P}_1 \mathbf{A}_{1,1} \mathbf{P}_1^\top & \mathbf{P}_1 \mathbf{A}_{1,2} \mathbf{P}_2^\top & \cdots & \mathbf{P}_1 \mathbf{A}_{1,b} \mathbf{P}_b^\top \\ \mathbf{P}_2 \mathbf{A}_{2,1} \mathbf{P}_1^\top & \mathbf{P}_2 \mathbf{A}_{2,2} \mathbf{P}_2^\top & \cdots & \mathbf{P}_2 \mathbf{A}_{2,b} \mathbf{P}_b^\top \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{P}_b \mathbf{A}_{b,1} \mathbf{P}_1^\top & \mathbf{P}_b \mathbf{A}_{b,2} \mathbf{P}_2^\top & \cdots & \mathbf{P}_b \mathbf{A}_{b,b} \mathbf{P}_b^\top \end{bmatrix} \\
&= \begin{bmatrix} \mathbf{P}_1 & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{P}_2 & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{P}_b \end{bmatrix} \begin{bmatrix} \mathbf{A}_{1,1} & \mathbf{A}_{1,2} & \cdots & \mathbf{A}_{1,b} \\ \mathbf{A}_{2,1} & \mathbf{A}_{2,2} & \cdots & \mathbf{A}_{2,b} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{A}_{b,1} & \mathbf{A}_{b,2} & \cdots & \mathbf{A}_{b,b} \end{bmatrix} \begin{bmatrix} \mathbf{P}_1^\top & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{P}_2^\top & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{P}_b^\top \end{bmatrix} \\
&= \mathbf{P}' \mathbf{A}' \mathbf{P}'^\top = \big(\mathbf{P}' \mathbf{P}_{\text{part}}\big) \mathbf{A} \big(\mathbf{P}_{\text{part}}^\top \mathbf{P}'^\top\big).
\end{aligned}$$

which yields an isomorphic graph to G. In the second case, the graph decoder first computes a

permutation π' of the indices $i_1, \ldots, i_b$ such that $i_{\pi'(1)} \leq i_{\pi'(2)}, \cdots \leq i_{\pi'(b)}$. It is not hard to see that even if $\pi' \neq \pi$ we will have that $i_{\pi'(j)} = i_{\pi(j)}$ and therefore we can define a graph decoder as:

$$
\begin{aligned}
h_{\text{g-dec}}(h_{\text{g-enc}}(G;\mathcal{D})) &= \begin{bmatrix} \mathbf{A}(\alpha_{i_{\pi(1)}}) & \cdots & \mathbf{P}_{\pi(1)}\mathbf{A}_{\pi(1),\pi(b)}\mathbf{P}_{\pi(b)}^\top \\ \vdots & \ddots & \vdots \\ \mathbf{P}_{\pi(b)}\mathbf{A}_{\pi(b),\pi(1)}\mathbf{P}_{\pi(1)}^\top & \cdots & \mathbf{A}(\alpha_{i_{\pi(b)}}) \end{bmatrix} = \begin{bmatrix} \mathbf{P}_{\pi(1)}\mathbf{A}_{\pi(1),\pi(1)}\mathbf{P}_{\pi(1)}^\top & \cdots & \mathbf{P}_{\pi(1)}\mathbf{A}_{\pi(1),\pi(b)}\mathbf{P}_{\pi(b)}^\top \\ \vdots & \ddots & \vdots \\ \mathbf{P}_{\pi(b)}\mathbf{A}_{\pi(b),\pi(1)}\mathbf{P}_{\pi(1)}^\top & \cdots & \mathbf{P}_{\pi(b)}\mathbf{A}_{\pi(b)),\pi(b)}\mathbf{P}_{\pi(b)}^\top \end{bmatrix} \\
&= \begin{bmatrix} \mathbf{P}_{\pi(1)} & \cdots & \mathbf{0} \\ \vdots & \ddots & \vdots \\ \mathbf{0} & \cdots & \mathbf{P}_{\pi(b)} \end{bmatrix} \begin{bmatrix} \mathbf{A}_{\pi(1),\pi(1)} & \cdots & \mathbf{A}_{\pi(1),\pi(b)} \\ \vdots & \ddots & \vdots \\ \mathbf{A}_{\pi(b),\pi(1)} & \cdots & \mathbf{A}_{\pi(b),\pi(b)} \end{bmatrix} \begin{bmatrix} \mathbf{P}_{\pi(1)}^\top & \cdots & \mathbf{0} \\ \vdots & \ddots & \vdots \\ \mathbf{0} & \cdots & \mathbf{P}_{\pi(b)}^\top \end{bmatrix} \\
&= \mathbf{P}'''\big(\mathbf{P}''\mathbf{A}'\mathbf{P}''^\top\big)\mathbf{P}'''^\top = \big(\mathbf{P}'''\mathbf{P}''\mathbf{P}_{\text{part}}\big)\mathbf{A}\big(\mathbf{P}_{\text{part}}^\top\mathbf{P}''^\top\mathbf{P}'''^\top\big).
\end{aligned}
$$

Finally, when some subgraphs cannot be mapped to any of the dictionary atoms, their original adjacency matrices (without being permuted) are placed right after the multiset of atom indices and the same result as above can be shown using a subgraph-level permutation π once again.

C.3 Additional Experiments

C.3.1 Ablation studies

Compression of unseen data. In Tables C.1 and C.2 we report the compression rates (in bpe) of the training and the test data separately for all the PnC variants. As can be seen, in most of the cases the generalisation gap is small, which implies that there was no evidence of overfitting and the compressor can be used to unseen data with a small degradation in the compression quality.

Table C.1: Train vs test data (avg negative log-likelihood in bpe). *Molecules.*

Dataset name	MUTAG		PTC		ZINC	
Set	train	test	train	test	train	test
PnC + SBM	3.81	3.85	4.40	4.25	3.33	3.41
PnC + Louvain	2.18	2.39	2.67	2.74	1.96	1.97
PnC + PropClust	2.37	2.89	3.33	3.83	2.19	2.27
PnC + Neural Part	2.16±0.02	2.28±0.03	2.64±0.24	2.59±0.21	2.01±0.02	2.03±0.01

Table C.2: Train vs test data (avg negative log-likelihood in bpe). *Proteins & social networks.*

Dataset name	PROTEINS		IMDB-B		IMDB-M	
Set	train	test	train	test	train	test
PnC + SBM	3.24	3.46	0.48	0.50	0.35	0.31
PnC + Louvain	3.33	3.47	0.94	0.95	0.66	0.59
PnC + PropClust	3.41	3.53	1.43	1.61	0.95	0.75
PnC + Neural Part	3.33±0.24	3.36±0.29	1.01±0.05	0.99±0.04	0.70±0.03	0.63±0.02

Out-of-distribution compression. In the following experiment we tested the ability of PnC to compress data sampled from different distributions. In particular, we trained the Neural Partitioning variant on one of the MUTAG and IMDB-B datasets and then used the pre-trained compressor on the remaining ones. In Table C.3 (left) we report the data as well as the total (data + model) description length, in accordance with the experiments of section 5.8. We make the following two observations: (1) As expected, PnC can generalise to similar distributions relatively well (in the table we highlight the MUTAG → ZINC and the IMDB-B → IMDB-M transfer), but fails to do so when there is significant distribution shift. (2) Although MUTAG contains only approximately 100 graphs, it is sufficient to train a compressor that can generalise to a significantly larger dataset (ZINC contains approximately 10K graphs), which is an indication that PnC is sample efficient.

Table C.3: Out of distribution compression (left) and probability of a subgraph to belong in the dictionary (right).

	Training dataset					
	MUTAG		IMDB-B		same dataset	
Test dataset	data	total	data	total	data	total
MUTAG	-	-	6.68	7.61	2.17±0.02	2.45±0.02
PTC	4.14	4.48	8.16	8.55	2.63±0.26	2.97±0.14
ZINC	**2.62**	**2.63**	6.92	6.94	2.01±0.02	2.07±0.03
PROTEINS	4.74	4.87	4.31	4.44	3.34±0.25	3.51±0.23
IMDB-B	1.83	1.86	-	-	1.00±0.04	1.05±0.04
IMDB-M	1.37	1.39	**0.74**	**0.77**	0.66±0.05	0.72±0.05

dataset	1 - δ
MUTAG	0.998
PTC	0.995
ZINC	0.999
PROTEINS	0.999
IMDB-B	0.995
IMDB-M	0.997

How frequently do we encounter dictionary subgraphs? In Table C.3 (right), we report the probability $1-\delta$ for the Neural Partitioning variant of PnC, i.e., the estimated probability of an arbitrary subgraph to belong in the dictionary. Interestingly, since the values are very close to 1, it becomes evident that the partitioning algorithm learns to detect frequent subgraphs in the distribution, which (following Theorem 5.2.b) can in part justify the high compression gains

of PnC in all the datasets.

C.3.2 Reducing the model size of deep generative models

Smaller architectures

It is made clear by the experimental results of section 5.8 that deep neural compression is particularly costly due to heavy overparameterisation. Yet, we also observe that these models achieve strong results in terms of the likelihood of the data. *Is it possible to strike a better balance between the number of parameters and the compression cost for a deep generative model?* To investigate this, we have conducted the following experiment. We trained 5 GRAN models that differ in parameter count, on 3 different datasets, and monitored the total BPE.

In order to consistently scale the number of parameters across these 5 different models, we have fixed the GNN depth for all models to one, and set for each model the size of the embedding, attention, and hidden dimension, to a constant c. Using a different c for each model allows us to explore different scales for the parameter count of the GRAN model. Furthermore, to facilitate comparison with PnC, one of the 5 models is trained without attention and features a reduced amount of mixture components. This is the minimum, in terms of parameter count, working instantiation of GRAN. Finally, we also considered the BPE for the null Erdős-Renyi (ER) model. Figure C.1 plots the total BPE of the different models against the number of parameters.

Results. In the low parameter regime, the GRAN models are not capable of outperforming the null models and fall significantly behind PnC. At scales that range from 10^3 to 10^4 parameters, we observe slight improvements in the total BPE of GRAN on the Proteins and the IMDB-Binary datasets. However, the improved likelihood is not able to compensate sufficiently for the increase in the number of parameters. This becomes more pronounced on larger scales, where GRAN experiences diminishing returns as the cost of parameters outpaces the likelihood gains. On the other hand, the results consistently worsen as the number of parameters grows on MUTAG. In this case, the size of the dataset is an additional detrimental factor that weighs against overparameterised models. Overall, the experiment suggests that, as the number of parameters grows, the off-the-shelf GRAN model becomes increasingly inefficient and is thus not well suited for compression.

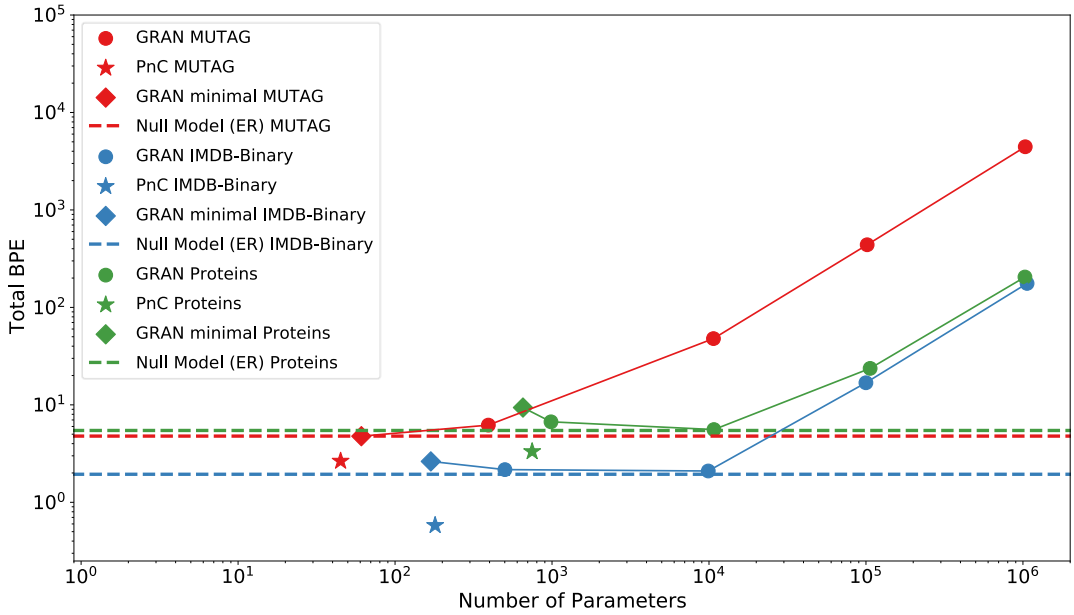


Figure C.1: Total BPE of likelihood-based models as a function of the parameter count. *GRAN minimal* refers to the minimum working GRAN model that does not feature attention and multiple mixture components. The non-parametric ER model is represented with dashed lines.

Pruning

Based on the results of the previous section, parameter search alone cannot mitigate the cost of overparameterisation. A more efficient approach to managing the tradeoff between model size and data likelihood is required. However, as shown in table C.4, an (at least) two orders of magnitude reduction in the model size without a decrease in the data likelihood is required for a more competitive neural compressor with deep generative models. In the following section, we experimented with a combination of modern model compression techniques as a heuristic to reduce the model size.

Model compression techniques aim to reduce the size of a given model while maintaining its performance. Research in model compression has empirically demonstrated that large models can often be considerably shrunk without suffering from major performance losses. Combined with parameter search and mixed precision training, model compression may result in more cost-effective neural compressors. We investigate the feasibility of such an approach in the following experiment.

First, we decrease the model size by hyper-parameter search. We fix the depth of both GraphRNN

and GRAN to the one provided in the original implementations, and gradually reduce their width to identify a compact version of the network that maintains high performance. This leads to fixing the width of both GraphRNN and GRAN to 16. *Then, we train both models using an iterative weight pruning technique.* We opt for global unstructured L1 weight pruning, using the lottery ticket procedure [Frankle and Carbin, 2019] that has been shown to be effective in the literature. The method we utilised proceeds in the following way: A model is trained for T iterations (T is a hyperparameter chosen based on the convergence and running time of the models on each dataset), then a percentage of the weights are pruned (25% in our case). After pruning, the unpruned weights are reset to their initial state and the process is repeated from the beginning using the new pruned network. This yields up to a 10x reduction in the size of the model in most datasets (we report the best total description length between all pruning phases). *Finally, we attempted to reduce the storage size of the model weights using half precision.* Traditionally, NNs are trained with 32-bit floating point numbers. Recently, progress has been made in mixed precision training which can enable the use of 16-bit tensors [Micikevicius et al., 2018]. We follow the same procedure and at the end of training, we store the model weights using 16 bits.

Tables C.5 and C.6 contain the results of both single and half-precision pruned models on all datasets. As can be observed in the results, both models benefit significantly from this hybrid approach, albeit at the cost of reduced data likelihood. *However, PnC is still able to outperform the pruned versions.* Although our approach to model compression is by no means exhaustive, it becomes evident that the procedure is quite tedious and choosing the right trade-off between the data likelihood and the model size is based on heuristics, hence the minimisation of the total description length cannot be guaranteed. Nevertheless, we believe that this is an important research direction that should be further explored in a more principled manner.

Table C.4: Minimum model compression ratios required for overparameterised neural compressors to outperform PnC. We assume zero degradation of the data likelihood.

dataset	**GraphRNN**	**GRAN**
MUTAG	x2980	x7657
PTC	x995	x6668
ZINC	x112	x227
PROTEINS	x105	x303
IMDBB	infeasible	x1745
IMDBM	infeasible	x2262

Table C.5: Pruning deep graph generators (single, half-precision). *Molecules.*

dataset name	MUTAG			PTC			ZINC		
	data	total	params	data	total	params	data	total	params
GraphRNN (half)	4.70	10.77	1.08K	9.53	12.10	1.10K	3.89	4.10	2.64K
GRAN (half)	2.41	14.84	2.21K	4.35	9.86	2.36K	3.26	3.38	1.67K
GraphRNN (single)	1.95	12.39	1.08K	2.16	6.71	1.10K	1.79	2.02	1.90K
GRAN (single)	2.59	24.56	2.23K	4.31	14.00	2.36K	3.26	3.47	1.69K

Table C.6: Pruning deep graph generators (single, half-precision). *Proteins & social networks.*

dataset name	PROTEINS			IMDB-B			IMDB-M		
	data	total	params	data	total	params	data	total	params
GraphRNN (half)	27.10	27.47	1.43K	4.21	4.49	1.28K	2.91	3.16	1.20K
GRAN (half)	3.89	4.70	3.16K	0.89	1.41	2.39K	0.61	1.10	2.31K
GraphRNN (single)	2.63	3.76	2.56K	1.43	1.92	1.28K	0.91	1.39	1.28k
GRAN (single)	4.28	5.11	1.78K	0.84	1.75	2.38K	0.55	1.41	2.31K

C.3.3 Vertex and Edge attributes

Our method can be easily extended to account for the presence of discrete vertex and edge attributes, the distribution of which can also be learned from the data. Assuming a discrete vertex attribute domain $\mathbb{N}^{d_v}$ and an edge attribute domain $\mathbb{N}^{d_e}$, we can use the following simple encodings for a graph with n vertices and m edges:

$$\mathrm{L}(\mathbf{u}_{\mathcal{V}}) = n \log |d_v| \text{ and } \mathrm{L}(\mathbf{u}_{\mathcal{E}}) = m \log |d_e|, \tag{C.21}$$

where $\mathbf{u}_{\mathcal{V}} \in \mathbb{N}^{\mathcal{V} \times d_V}$ the vertex attributes and $\mathbf{u}_{\mathcal{E}} \in \mathbb{N}^{\mathcal{E} \times |d_e|}$ the edge attributes. One could also choose a more sophisticated encoding by explicitly learning the probability of each attribute. In this case, the dictionary becomes even more relevant, since when simply partitioning the graph, the attributes will still have to be stored in the same manner for each subgraph and each edge in the cut. Hence, in the absence of the dictionary, the attributes will have to be encoded in a raw format.

In Table C.7, we showcase a proof of concept in the attributed MUTAG and PTC MR datasets, which are variations of those used for structure-only compression. Vertex attributes represent atom types and edge attributes represent the type of bond between two atoms. As mentioned in the previous paragraph, it is clear that non-dictionary methods are hardly improving w.r.t the

null model, which is mainly due to the fact that the attributes constitute the largest portion of the total description length. Another interesting observation is that since the clustering algorithms we used are oblivious to the existence of attributes, they are less likely to partition the graph in such a way that the attributed subgraphs will be repetitive unless the structure is strongly correlated with the attributes. This becomes clear in the PTC MR dataset, where between the different PnC variants, the neural partitioning performs considerably better since the partitioning is optimised in coordination with the dictionary. In Figure 5.2 of the main part of the thesis, we show the most probable substructures that the Neural Partitioning yields for the MUTAG dataset. It is interesting to observe that typical molecular substructures are extracted. This highlights an interesting application of molecular graph compression, i.e., discovering representative patterns of the molecular distribution in question.

Table C.7: Average bpe for *attributed molecular* datasets. First, Second, Third

Method Family	Dataset name	Atrributed MUTAG			Atrributed PTC MR		
		data	total	params	data	total	params
Null	Uniform (raw adjacency)	-	13.33	-	-	16.32	-
	Edge list	-	12.62	-	-	14.06	-
	Erdős-Renyi	-	9.38	-	-	10.87	-
Clustering	SBM-Bayes	-	9.17	-	-	10.61	-
	Louvain	-	9.37	-	-	10.76	-
	PropClust	-	9.52		-	10.80	
PnC	PnC + SBM	6.56	7.49	78	8.05	**9.49**	158
	PnC + Louvain	3.52	**4.45**	78	5.56	**7.65**	200
	PnC + PropClust	5.21	**6.30**	54	8.51	9.58	118
	PnC + Neural Part.	3.83±0.06	**4.78±0.12**	74±6	5.19±0.39	**6.49±0.54**	170±30

C.4 Implementation Details

Datasets: We evaluated our method on a variety of datasets that are well-established in the GNN literature. In specific, we chose the following from the TUDataset collection [Morris et al., 2020a]: the molecular datasets MUTAG [Debnath et al., 1991, Kriege and Mutzel, 2012] (mutagenicity prediction) and PTC-MR [Helma et al., 2001, Kriege and Mutzel, 2012] (carcinogenicity prediction), the protein dataset PROTEINS [Borgwardt et al., 2005, Dobson and Doig, 2003] (protein function prediction - vertices represent secondary structure elements and edges either neighbourhoods in the aminoacid sequence or proximity in the 3D space) and the social network datasets IMDBBINARY and IMDBMULTI [Yanardag and Vishwanathan,

2015] (movie collaboration datasets where each graph is an ego-net for an actor/actress). We also experimented with the ZINC dataset [Irwin et al., 2012b, Kusner et al., 2017, Gómez-Bombarelli et al., 2018, Jin et al., 2018a] (molecular property prediction), which is a larger molecular dataset from the dataset collection introduced in [Dwivedi et al., 2020]. A random split is chosen for the TUDatasets (90% train, 10% test), since we are not interested in the class labels, while for the ZINC dataset, we use the split given by the authors of [Dwivedi et al., 2020] (we unify the test and the validation split since we do not use the validation set for hyperparameter tuning/model selection).

PnC model architecture and hyperparameter tuning: The GNN used for the Neural partitioning variant of PnC is a traditional Message Passing Neural Network [Gilmer et al., 2017], where a general formulation is employed for the message and the update functions (i.e., we use Multi-layer Perceptrons similar to [Loukas, 2020]). We optimise the following hyperparameters: batch size in {16, 64, 128}, network width in {16, 64} number of layers in {2, 4}. The learning rate for the updates of the dictionary and probabilistic model parameters was 1 and 0.1 for the fixed partitioning and the neural partitioning variants respectively, while the learning rate of the GNN (neural partitioning only) was set to 0.001. For all the variants we further tune the maximum number of vertices for the dictionary atoms k in {6, 8, 10, 12}. Note that the last hyperparameter mainly affects the optimisation of the Neural Partitioning variant: small values of k will constrain the possible subgraph choices, but will facilitate the network to find good partitions by exploitation. On the other hand, larger values of k will encourage exploration, but the optimisation landscape becomes significantly more complex, thus in some cases (mainly for social networks, where there is a larger variety of non-isomorphic subgraphs) we observed that the optimisation algorithm could not converge to good solutions.

We optimise each PnC variant for 100 epochs and report the result on the epoch where the description length of the training set is minimum. The best hyperparameter set is also chosen w.r.t the lowest training set description length, and after its selection, we repeat the experiment for 3 different seeds (in total). Table C.8 shows the chosen hyperparameters.

We implement our framework using PyTorch Geometric [Fey and Lenssen, 2019b], while the predefined partitioning algorithms were implemented using graph-tool [Peixoto, 2014] for the SBM fitting and scikit-network [Bonald et al., 2020] for the Louvain and the Propagation Clustering algorithms. To track our experiments we used the wandb platform [Biewald, 2020].

Table C.8: Chosen hyperparameters for each dataset (PnC + NeuralPart)

dataset	batch size	width	number of layers	k
MUTAG	16	16	2	10
PTC	16	16	2	10
ZINC	128	16	2	10
PROTEINS	16	16	4	8
IMDB-B	16	16	2	8
IMDB-M	64	64	2	8

Deep generative models and pruning. For the generative model baselines, we have used the official implementations provided in the corresponding repositories[4]. For GraphRNN, we trained with the default parameters provided with the official implementation and only tuned the number of training epochs according to the time required for convergence. For GRAN, we adopt one of the configurations provided in the official repository with minor modifications. Namely, we used a DFS ordering, stride and block size 1, 20 Bernoulli mixture components for the parametrisation of the likelihood, and switched of the subgraph sampling feature.

For our iterative pruning protocol, we fix the same number of pruning iterations for both models on each dataset. Specifically, we use $\{450, 270, 10, 90, 90, 90\}$ total epochs and a pruning interval T of $\{50, 30, 1, 10, 10, 10\}$ for MUTAG, PTC, ZINC, PROTEINS, IMDB-B, and IMDB-M respectively. We used a 25% pruning percentage, which lead to a 10-fold reduction in model size in most cases. Further pruning was not found to be consistently beneficial in the parameter ranges that we experimented on.

Model parameter cost. For the PnC variants, we could seamlessly use half-precision (16 bits) to store the model parameters (section 5.6) without sacrificing compression quality. However, as discussed in Appendix C.3.2 we were not able to retain similar likelihood estimates when storing with half-precision the weights of deep generative models, hence in the results reported in the main tables, we used 32 bits to store the model weights.[5] Additionally, regarding the pruned versions of deep generative models, we need to transmit the locations of the non-zero weights for each parameter matrix in $\mathbb{R}^{d_1 \times d_2}$, which are encoded as follows: $\log(d_1 d_2 + 1) + \log \binom{d_1 d_2}{e}$, where e is the number of the non-zero elements. For all methods compared, the decompression

[4] https://github.com/JiaxuanYou/graph-generation & https://github.com/lrjconan/GRAN

[5] In a preliminary version of the paper we assumed a 16-bit weight encoding without likelihood losses. However, our subsequent implementation with half-precision deep graph generators demonstrated that this might not be possible in practice.

algorithm and the neural network architectures are assumed to be public, hence they do not need to be transmitted.

Isomorphism. In order to speed up isomorphism testing between dictionary atoms and the subgraphs that the partitioning algorithm yields, we make the following design choices: (a) Dictionary atoms are sorted by their frequencies of appearance (these are computed by an exponential moving average that gets updated during training). In this way, the expected number of comparisons drops to $O(1)$ from $O(|D|)$. (b) We choose the parameter k to be a small constant value (i.e., does not scale with the number of vertices of the graph), as previously mentioned. It becomes clear, that except for the importance of k in the optimisation procedure, it also plays a crucial role in scalability, since as mentioned in the introduction of the main part of the thesis, solving the isomorphism problem quickly becomes inefficient when the number of vertices increases. (c) Additionally, one can choose to approximate isomorphism with faster algorithms, such as the Weisfeiler-Leman test [Weisfeiler and Leman, 1968], or with more expressive Graph Neural Networks, such as the ones we saw in section 4.7.1. These algorithms will always provide a correct negative answer whenever two graphs are non-isomorphic, but a positive answer does not always guarantee isomorphism. In that case, exact isomorphism can be employed only when the faster alternatives give a positive answer.

Translating probabilities into codes. In the following section, we explain how a partitioned graph can be represented into a bitstream using our probabilistic model. The general principle for modern entropy encoders (Arithmetic Coding [Witten et al., 1987] and Asymmetric Numeral Systems [Duda, 2013]) is that both the encoder and the decoder need to possess the cumulative distribution function (c.d.f.) of each component they are required to encode/decode. Hence, the encoder initially sends to the decoder the parameters of the model $\boldsymbol{\varphi}$ using a fixed precision encoding (e.g., we used 16 bits for our comparisons). The rest of the bitstream is described below:

- **Dictionary.** The dictionary is sent as part of the preamble of the message. It consists of the following:

  (a) the size of the dictionary (we assume fixed precision for this value),

  (b) a sequence of dictionary atoms encoded with the null model, where each message

includes the number of vertices, the number of edges and finally the adjacency matrix: $k_i, m_i, \mathbf{A}(\alpha_i)$ (see Eq. (5.18)).

- **Graphs:** Subsequently, graphs are sequentially transmitted. The message contains the following:

  (a) the total number of subgraphs b using a categorical distribution parametrised with $q(b)$, and the number of dictionary subgraphs b_{dict} that are encoded using δ and b to parametrise the binomial distribution $\text{Binomial}(b_{\text{dict}} \mid b; \boldsymbol{\varphi})$. The c.d.f. of the binomial distribution can be computed using a factorisation described in [Steinruecken, 2015],

  (b) the subgraphs that belong in the dictionary, which are encoded using the multinomial distribution $q(\mathcal{H}_{\text{dict}} \mid b_{\text{dict}}; \boldsymbol{\varphi})$. As above, a factorisation described in [Steinruecken, 2015] can be used to compute the c.d.f.,

  (c) the non-dictionary subgraphs. These are encoded with the null model (same as the encoding of dictionary atoms as mentioned above),

  (d) the cuts, which are encoded using Eq. (5.17).

Several of our encodings involve uniform distributions over combinations of elements (e.g., for the adjacency matrix in the null model). To compute them, we can either factorise the distribution as in [Steinruecken, 2016] in order to efficiently compute the c.d.f or use a ranking function (and its inverse for the decoder) that maps a combination to its index in lexicographic order (e.g., see [Kreher and Stinson, 2020]).