Deep Learning on Graphs: Developing and Understanding Neural Architectures for Structured Data

Von der Fakultät für Mathematik, Informatik und Naturwissenschaften der
RWTH Aachen University zur Erlangung des akademischen Grades
eines Doktors der Naturwissenschaften genehmigte Dissertation

vorgelegt von

Jan Martin Tönshoff, Master of Science

aus Mönchengladbach

Berichter: Universitätsprofessor Dr. Martin Grohe
Universitätsprofessor Dr. Stephan Günnemann

Tag der mündlichen Prüfung: 4. November 2024

Diese Dissertation ist auf den Internetseiten der Universitätsbibliothek online verfügbar.

Abstract

Graph Neural Networks (GNNs) have recently emerged as a powerful class of deep learning architectures that can directly learn from graph-structured data. Such data naturally occurs in a wide range of structurally rich learning domains, such as computational chemistry or social network analysis. This thesis aims to study both practical and theoretical aspects of GNNs, with a focus on understanding their capabilities for solving hard learning tasks, exploring new directions for GNN architecture design, and understanding the relative strengths and weaknesses of commonly used models. In this context, we present our main research contributions: Firstly, we propose a GNN-based approach for learning heuristics for Constraint Satisfaction Problems (CSPs) through a general graph representation and neural architecture. Using reinforcement learning, we show that this approach can learn powerful search algorithms, achieving superior performance compared to prior neural methods and being competitive with classical solvers on a range of complex combinatorial problems. Secondly, we examine a novel GNN architecture based on 1D convolutions on sequential features constructed from random walks. We analyze the expressivity of this approach and prove it to be incomparable to the Weisfeiler-Leman hierarchy and many common GNN architectures. The model is further shown to achieve strong empirical performance across various tasks, suggesting a new direction for GNN architecture design beyond traditional message passing. Thirdly, we compare popular GNNs in the context of capturing long-range dependencies in graphs. Through an empirical re-evaluation of commonly used benchmark datasets, we show that standard GNNs based on message passing perform better than previously reported. A theoretical comparison between Graph Transformers and GNNs with Virtual Nodes further shows that neither architecture subsumes the other in terms of uniform expressivity, highlighting the need to consider a wide range of architectures for specific learning tasks.

Zusammenfassung

Graph Neural Networks (GNNs) sind kürzlich als eine leistungsstarke Klasse von Deep-Learning-Architekturen entstanden, die direkt aus graphstrukturierten Daten lernen können. Diese Dissertation zielt darauf ab, sowohl praktische als auch theoretische Aspekte von GNNs zu untersuchen, wobei der Schwerpunkt darauf liegt, ihre Fähigkeiten zur Lösung anspruchsvoller Lernprobleme zu verstehen, neue Richtungen für das Design von GNN-Architekturen zu erkunden und die relativen Stärken und Schwächen von häufig verwendeten GNN-Modellen zu verstehen. In diesem Kontext präsentieren wir unsere primären Forschungsbeiträge: Erstens analysieren wir einen GNN-basierten Ansatz, um Heuristiken für Constraint Satisfaction Probleme (CSPs) durch eine allgemeine Graphkonstruktion und neuronale Architektur zu lernen. Wir zeigen, dass dieser Ansatz leistungsfähige Suchalgorithmen lernen kann, die eine überlegene Leistung im Vergleich zu früheren neuronalen Methoden erzielen und bei einer Reihe von komplexen kombinatorischen Problemen mit klassischen Algorithmen konkurrieren können. Zweitens untersuchen wir eine neuartige GNN-Architektur, die auf 1D-Faltungen und sequenziellen Featuren von Random Walks basiert. Wir analysieren die Ausdrucksstärke dieser Methode und beweisen, dass sie sich fundamental von der Weisfeiler-Leman-Hierarchie und der Ausdruckstärke vieler gängiger GNN-Architekturen unterscheided. Das Modell erreicht starke empirische Ergebnisse für verschiedene Lernaufgaben und schlägt eine neue Richtung für das Design von GNN-Architekturen über das konventionelle Message-Passing hinaus vor. Drittens vergleichen wir die Fähigkeit häufig genutzter GNN-Modelle Graphstrukturen über lange Distanzen zu erfassen. Durch eine empirische Neuauswertung von häufig verwendeten Benchmark-Datensätzen zeigen wir, dass einfache GNNs, die auf Message-Passing basieren, deutlich besser abschneiden als zuvor berichtet. Weiter zeigt ein theoretischer Vergleich zwischen Graph Transformern und GNNs mit virtuellen Knoten, dass keine der beiden Architekturen die andere im Sinne der uniformen Expressivität übertrifft, was die Notwendigkeit hervorhebt, eine breite Palette von Architekturen für spezifische Lernprobleme in Betracht zu ziehen.

Acknowledgements

First and foremost, I thank my supervisor, Martin Grohe, for providing me with support, encouragement, and the opportunity to pursue my research interests. I had the privilege of studying graph learning under his guidance since my bachelor's degree. Since then, I have thoroughly enjoyed working and growing as a researcher in this field and in the research environment fostered by Martin's leadership. I would additionally like to thank Benny Kimelfeld and Christopher Morris for their invaluable mentorship and help throughout my research pursuits.

I also have to thank all of my colleagues at the Chair for Logic and Theory of Discrete Systems for making my time as a Ph.D. student truly delightful. In particular, I thank Eran Rosenbluth, Martin Ritzert, and Hinrikus Wolf for the countless helpful and fun conversations.

Further acknowledgments go to my student assistants, Berke Kisin and Jakob Lindner, for continually supporting my research through their work. I would also like to thank Neta Friedman and Yuval Lubarsky for their collaboration and hospitality.

Special thanks belong to my family and partner for supporting every step of my academic journey and to the friends who made it a shared experience.

Finally, I also have to thank the coffee machine for its tireless work.

Contents

Abstract iii

Zusammenfassung v

Acknowledgements vii

List of Figures xiii

List of Tables xv

1 Introduction 1
- 1.1 Research Contributions . . . 3
- 1.2 List of Publications . . . 4
- 1.3 Preliminaries . . . 6

2 Deep Learning on Graphs 9
- 2.1 Message Passing . . . 9
  - 2.1.1 Message Passing Layers . . . 12
- 2.2 Expressivity of Graph Neural Networks . . . 15
  - 2.2.1 The Weisfeiler-Leman Algorithm . . . 15
  - 2.2.2 Uniform Function Approximation . . . 18
  - 2.2.3 VC Dimension . . . 21
- 2.3 Positional and Structural Encodings . . . 22
  - 2.3.1 Random Walk Structural Encodings . . . 23
  - 2.3.2 Laplacian Positional Encodings . . . 23
- 2.4 Enabling Long-Range Interactions . . . 24
  - 2.4.1 Virtual Nodes . . . 25
  - 2.4.2 Graph Transformers . . . 26
  - 2.4.3 Rewiring . . . 29

3 Learning Global Search Heuristics for CSPs 31
- 3.1 Constraint Satisfaction Problems . . . 32
- 3.2 Reinforcement Learning . . . 34
- 3.3 Related Work . . . 36

3.3.1 S2V-DQN . . . 37
3.3.2 ECO-DQN . . . 38
3.3.3 RLSAT . . . 39
3.3.4 RUNCSP . . . 40
3.4 ANYCSP . . . 42
3.4.1 Graph Construction . . . 43
3.4.2 Architecture . . . 44
3.4.3 Global Search as a Reinforcement Learning Problem . . . 48
3.4.4 Rewarding Iterative Improvements . . . 49
3.4.5 Markov Decision Process . . . 50
3.4.6 Training . . . 50
3.4.7 Expressivity . . . 52
3.4.8 Relabeling Constraint Value Graphs . . . 53
3.5 Experiments . . . 54
3.5.1 Experimental Setup and Hyperparameters . . . 54
3.5.2 Model RB . . . 55
3.5.3 Graph Coloring . . . 57
3.5.4 MaxCut . . . 59
3.5.5 3-SAT . . . 60
3.5.6 Max-k-SAT . . . 61
3.5.7 Cross-Comparison . . . 63
3.5.8 Impact of the Action Space and Reward . . . 64
3.6 Discussion . . . 66

4 Graph Learning with 1D Convolutions and Random Walks 69
4.1 Introduction . . . 69
4.2 Related Work . . . 70
4.3 Method . . . 72
4.3.1 Random Walk Features . . . 73
4.3.2 Architecture . . . 75
4.3.3 Convolution Module . . . 76
4.3.4 Asymptotic Runtime . . . 77
4.4 Expressivity . . . 78
4.4.1 Proof of Theorem 4.1 (1) . . . 83
4.4.2 Proof of Theorem 4.1 (2) . . . 85
4.5 Experiments . . . 86
4.5.1 Benchmark Datasets . . . 87
4.5.2 Synthetic Data . . . 89
4.5.3 Counting Substructures . . . 90
4.5.4 Runtime . . . 91
4.5.5 Ablation Study . . . 92
4.6 Discussion . . . 94

5 Long-Range Interactions in Theory and Practice 97

5.1 Reassessing the Long-Range Graph Benchmark 98
5.1.1 Experimental Setup 99
5.1.2 Results 101
5.1.3 Discussion 104
5.2 Expressivity of Self-Attention and Virtual Nodes 105
5.2.1 Related Work 106
5.2.2 Non-Uniform Function Approximation on Graphs 106
5.2.3 Uniform Function Approximation with Self-Attention and Virtual Nodes 108
5.2.4 Experiments 111
5.2.5 Discussion 114

6 Conclusion **115**
6.1 Directions for Future Work 116

A Appendix Chapter 3 **139**

B Appendix Chapter 4 **143**

C Appendix Chapter 5 **145**

List of Figures

1.1 Example Graphs 2

2.1 Message Passing GNN 10
2.2 1-WL Subtree Illustration 15
2.3 1-WL Counter Example 16
2.4 Experiments on uniform function approximation 20
2.5 VC Dimension of MPGNNs 22
2.6 Illustration of Long-Range Interactions 25
2.7 An overview of compared models 26

3.1 Illustration of S2V-DQN 37
3.2 Constraint Value Graph 43
3.3 Message Passing Scheme of ANYCSP 45
3.4 Illustration of ANYCSP 46
3.5 2-coloring Example 48
3.6 Model RB Results 56
3.7 Max-5-SAT Results 62
3.8 Ablation of Reward and Search Strategy 66

4.1 Illustration of a CRaWl layer 73
4.2 High-level view of a CRaWl model 75
4.3 Graphs from Theorem 4.5 85
4.4 CSL Graphs 90
4.5 Effects of Hyperparameters 93

5.1 Reevaluated results on Peptides datasets 100
5.2 Effect of normalization in vision datasets 102
5.4 Synthetic experiments on uniform approximation 112

List of Tables

3.1 ANYCSP Hyperparameters . . . 54
3.2 Graph Coloring Results . . . 58
3.3 MaxCut Results on GSet . . . 59
3.4 3-Sat Results on SATLIB . . . 60
3.5 Max-k-Sat Comparison . . . 61
3.6 CSP Cross-Comparison . . . 63
3.7 Ablation results for graph coloring and Max-k-SAT . . . 64
3.8 MaxCut Ablation . . . 65

4.1 CRaWl results on benchmark datasets . . . 88
4.2 CRaWl results on LRGB . . . 89
4.3 CRaWl results on CSL graphs . . . 90
4.4 Test MAE for various subgraph counting tasks. . . . 91
4.5 CRaWl runtime comparison . . . 91
4.6 CRaWl ablation study . . . 92

5.1 LRGB statistics . . . 99
5.2 LRGB hyperparameter ranges . . . 101
5.3 Performance of Virtual Nodes on Benchmark Datasets . . . 113

A.1 URLs and versions of used software. . . . 139
A.2 Extended MaxCut results on Gset graphs . . . 141
A.3 Extended results on structured Vertex Coloring instances ($k < 10$) . 142
A.4 Extended results on structured Vertex Coloring instances ($k \geq 10$) . 142

B.1 CRaWl hyperparameters . . . 143
B.2 Statistics for our main datasets. . . . 143

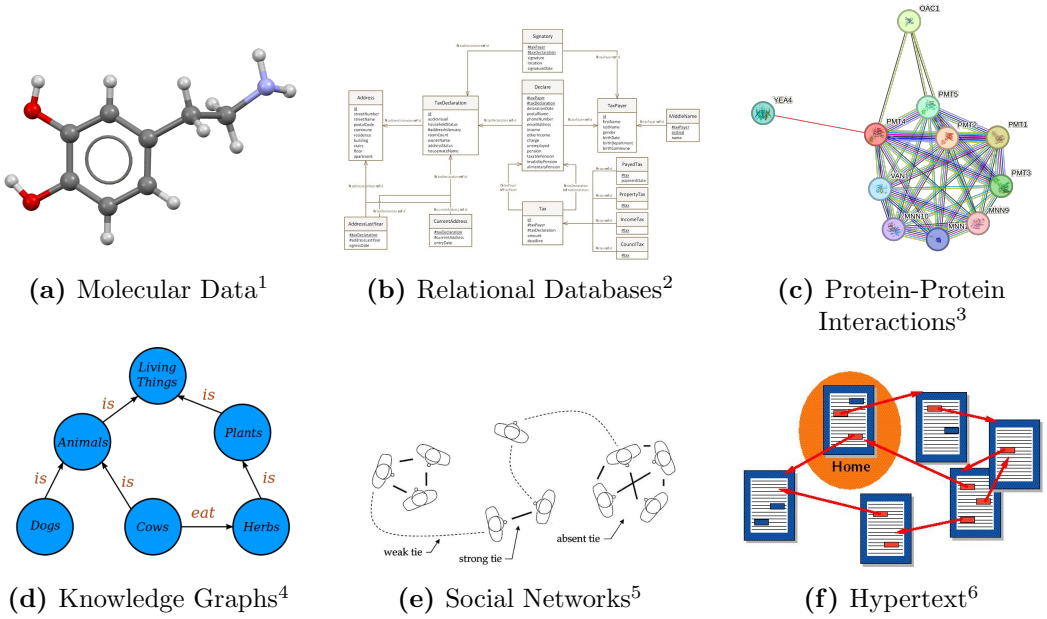
C.1 Hyperparameters on Peptides-Func . . . 145
C.2 Hyperparameters on Peptides-Struct . . . 145
C.3 Hyperparameters on PascalVOC-SP . . . 146
C.4 Hyperparameters on COCO-SP . . . 146
C.5 Hyperparameters on PCQM-Contact . . . 146
C.6 Performance of our models on the Long-Range Graph Benchmark. 147
C.7 Hyperparameters for GatedGCN+VN. . . . 147
C.8 Hyperparameters for GCN+VN. . . . 147

Introduction

The field of deep learning has seen rapid progress in recent years. Improvements in neural architectures, the emergence of large-scale datasets, and powerful hardware accelerators have enabled the training of large neural networks with unprecedented predictive performance on a wide range of tasks. Computer vision and natural language processing, in particular, have been the subject of well-publicized advances such as the development of powerful generative models which quickly achieved widespread adoption. While broadly applicable, both images and texts constitute data with a rigid, regular structure based on grids and sequences, respectively. This facilitates the development of specific neural architectures, such as CNNs, RNNs, and Transformers, but also limits their direct usage to data that shares the same narrow structural constraints.

In many domains, we instead work with structurally rich data that can naturally be modeled within the formalism of *graphs* as sets of *vertices* connected by *edges*. Such graph-structured data is ubiquitous and occurs in a wide range of contexts. Figure 1.1 illustrates common examples: Molecules can be viewed as graphs where atoms form vertices and chemical bonds define edges. Mutual interactions between proteins in biological processes are naturally modeled as graphs. The content of relational databases is inherently graph-structured as foreign key constraints define edges between rows. Knowledge graphs are frequently used to store structured knowledge about entities and their mutual relationships. Social networks model members and their interactions as vertices and edges. Hypertext defines a graph structure where hyperlinks connect documents.

The field of *graph learning* studies machine learning approaches for graph-structured data. In the context of deep learning, the main focus are *Graph Neural Networks* (GNNs), which are neural architectures that can process graphs as inputs and incorporate structural features in their predictions. The large spectrum of applications includes molecular property prediction (Gilmer et al., 2017), drug discovery (Gaudelet et al., 2021), physics simulation (Lam et al., 2023), combinatorial optimization (Cappart et al., 2021), planning (Ståhlberg et al., 2022), recommender systems (Wu et al., 2022), fraud detection (Liu et al., 2021), knowledge graph reasoning (Yasunaga et al., 2021) and database completion (Cvitkovic, 2019), among others.



(a) Molecular Data[1] **(b)** Relational Databases[2] **(c)** Protein-Protein Interactions[3]

(d) Knowledge Graphs[4] **(e)** Social Networks[5] **(f)** Hypertext[6]

Figure 1.1: Six common examples of graph-structured data.

Given the extensive set of tasks, the development and analysis of GNNs have recently received great interest. To understand the strengths and weaknesses of the proposed architectures it is critical to understand the main challenges that arise when learning on graph-structured data:

(1) *Generality*: A GNN should impose minimal assumptions on the structure of the inputs and should in principle be able to process graphs of arbitrary size and structure.

(2) *Expressivity*: An architecture has to be able to express sufficiently powerful functions that can capture important structural features while generalizing to unseen data.

(3) *Equivariance*: A GNN model should respect the underlying symmetries of graphs and, for example, yield equivalent outputs for two isomorphic graphs.

(4) *Efficiency*: GNNs should be able to scale to large graphs and datasets while effectively utilizing modern hardware accelerators, such as GPUs.

Utilizing GNNs in practice requires balancing these considerations based on the particular demands of a given learning task. This necessitates a principled understanding of the capabilities and limitations of different GNN architectures to aid the development of suitable approaches for specific problems. Contributing towards this understanding from both theoretical and empirical viewpoints is the general aim this thesis.

[1]Dopamine Molecule; [2]NETF-logical database © Ridene Youssef, CC BY-SA 4.0
[3]PMT4 Interaction Graph © STRING Consortium 2023, CC BY 4.0
[4]Knowledge Graph Example © Jayarathina, CC BY 4.0; [5]Social Tie Network
[6]Documents connected by Hyperlinks © Robert Cailliau, CC BY-SA 3.0

1.1 Research Contributions

Let us summarize the main research objectives and associated contributions that are presented throughout this thesis.

1. Develop novel high-performance approaches for challenging applications of graph learning. To this end, we introduce and evaluate ANYCSP, a GNN-based approach for learning heuristics for constraint satisfaction problems (CSPs), which are a large class of hard combinatorial problems. Our approach combines a general graph representation and GNN architecture with reinforcement learning to obtain data-driven search heuristics for any combinatorial problem within the CSP formalism. ANYCSP does not just outperform prior neural approaches by a significant margin but manages to compete with classical solvers on a range of important problems. We thereby probe the capabilities of GNNs in the context of combinatorial problems and demonstrate the significant potential of neural methods for learning complex algorithms.

2. Design new architectures with an expressivity that is provably distinct from prior approaches. Novel methods that are fundamentally different from existing architectures are of significant interest, as they may be able to fill niches in which prior work struggles. To contribute towards this goal we introduce CRaWl as a neural architecture that deviates from prior work based on message passing and is instead based on an architecture that applies 1D convolutions to sequential features obtained through random walks. We show that this approach yields an expressivity that is provably incomparable to standard Message Passing GNNs (MPGNNs) and their higher-order variants. We further show that CRaWl yields strong empirical performance on a range of learning tasks.

3. Conduct novel empirical and theoretical comparisons of existing neural architectures for graph learning. With the large number of recently proposed GNN architectures, it is crucial to provide principled and reliable comparisons of existing approaches in both theory and practice. Both kinds of comparison are vital for helping practitioners make informed decisions when selecting architectures for a given application. Our main contribution towards this objective is two-fold: First, we empirically reevaluate commonly cited results for benchmark datasets that aim to test how well GNNs capture long-range features in graphs. There, we show that simple MPGNNs perform significantly better than previously thought. Second, we contribute new theoretical insights on the uniform expressivity of two popular classes of GNNs, namely Graph Transformers and MPGNNs with Virtual Nodes. We show that the two approaches have separate strengths and weaknesses as neither is strictly stronger than the other with respect to approximating functions uniformly.

Before presenting these main contributions in Chapters 3 to 5, we introduce relevant background on deep learning for graph-structured data in Chapter 2. There, we provide an overview of key architectures, expressivity results, and commonly used techniques to enhance GNN performance in practice.

1.2 List of Publications

The content of this thesis is based on peer-reviewed research published by the author. The main research contributions presented in Chapters 3 to 5 are based on the following publications:

- Jan Tönshoff, Berke Kisin, Jakob Lindner, and Martin Grohe. One Model, Any CSP: Graph Neural Networks as Fast Global Search Heuristics for Constraint Satisfaction. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence* (IJCAI), 8 2023.
  Contribution: The author contributed the majority of the concept, methodological development, evaluation, and writing.

- Jan Tönshoff, Martin Ritzert, Hinrikus Wolf, and Martin Grohe. Walking out of the Weisfeiler Leman Hierarchy: Graph Learning Beyond Message Passing. *Transactions on Machine Learning Research* (TMLR), 9 2023.
  Contribution: The author contributed a majority of the concept, methodological development, and evaluation, as well as significant parts of the writing.

- Jan Tönshoff, Martin Ritzert, Eran Rosenbluth, and Martin Grohe. Where did the Gap Go? Reassessing the Long-Range Graph Benchmark. *Transactions on Machine Learning Research* (TMLR), 5 2024.
  Contribution: The author contributed the majority of the concept and writing and significant parts of the experimental evaluation.

- Eran Rosenbluth*, Jan Tönshoff*, Martin Ritzert*, Berke Kisin, and Martin Grohe. Distinguished in Uniform: Self-Attention vs. Virtual Nodes. *In The Twelfth International Conference on Learning Representations* (ICLR), 4 2024. (* Equal Contribution)
  Contribution: The author contributed the majority of the concept and experimental evaluation, significant parts of the writing, and minor parts of the theoretical results.

The author has further contributed as a co-author to relevant related work presented as background in this thesis:

- Eran Rosenbluth, Jan Tönshoff, and Martin Grohe. Some Might Say All You Need is Sum. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence*, (IJCAI), 8 2023.
  Contribution: The author contributed the majority of the experimental evaluation and associated writing.

- Christopher Morris, Floris Geerts, Jan Tönshoff, and Martin Grohe. WL meet VC. In *Proceedings of the 40th International Conference on Machine Learning* (ICLR) 7 2023.
  Contribution: The author contributed parts of the experimental evaluation and associated writing.

- Jan Tönshoff, Martin Ritzert, Hinrikus Wolf, and Martin Grohe. Graph Neural Networks for Maximum Constraint Satisfaction. *Frontiers in Artificial Intelligence*, 3 2021.
  Contribution: The author contributed the majority of the methodological development, evaluation, and writing. The work was conducted for the author's master thesis "Unsupervised Machine Learning for Constraint Satisfaction Problems".

Additional published work of the author that is beyond the scope of this thesis is listed in the following:

- Jan Tönshoff, Neta Friedman, Martin Grohe, and Benny Kimelfeld. Stable tuple embeddings for dynamic databases. In *IEEE 39th International Conference on Data Engineering* (ICDE), 4 2023.
  Contribution: The author contributed a significant part of the methodological development, evaluation, and writing.

- Yuval Lubarsky, Jan Tönshoff, Martin Grohe, and Benny Kimelfeld. Selecting walk schemes for database embedding. *Conference on Information and Knowledge Management* (CIKM), 10 2023.
  Contribution: The author contributed a significant part of the methodological development, evaluation, and writing.

- Thorben Finke, Michael Krämer, Alexander Mück, and Jan Tönshoff. Learning the Language of QCD Jets with Transformers. *Journal of High Energy Physics* (JHEP), 6 2023.
  Contribution: The author contributed significantly to the concept and implemented the initial code base.

1.3 Preliminaries

We use $\{\cdot\}$ to denote sets and $\{\!\{\cdot\}\!\}$ to denote multisets. With $\mathbb{N}, \mathbb{N}_{>0}, \mathbb{R}$ we denote the sets of nonnegative integers, positive integers, and real numbers, respectively. For $n \in \mathbb{N}_{>0}$, we use the notation $[n] \coloneqq \{i \in \mathbb{N} \mid 1 \leq i \leq n\}$.

Graph Theory

A graph $G = (V, E)$ is defined by a finite set of vertices $V = [n]$ and edges $E \subseteq V \times V$. The *order* of G is defined as its number of vertices n. A graph is *undirected* if for all $(u, v) \in E$ it also holds that $(v, u) \in E$. Otherwise, we refer to G as *directed*. Unless stated otherwise, we usually assume graphs to be undirected. The *neighborhood* $\mathcal{N}(v) = \{u \mid (u, v) \in E\}$ of a vertex $v \in V$ is the set of vertices that are adjacent to v and we refer to the size of this neighborhood $|\mathcal{N}(v)|$ as the *degree* of v. For directed graphs, we further distinguish between the in-neighborhood $\mathcal{N}^-(v) = \{u \mid (u, v) \in E\}$ and out-neighborhood $\mathcal{N}^+(v) = \{u \mid (v, u) \in E\}$. The *adjacency matrix* $A \in \{0, 1\}^{n \times n}$ of a graph G is defined such that for all $i, j \in [n]$ $A_{i,j} = 1 \Leftrightarrow (i, j) \in E$. A featured graph $G = (V, E, x)$ is a graph with an additional vertex feature map $x : V \to D$ that maps each vertex in G to a value in some feature domain D. We denote with $\mathcal{G}_D$ the set of all featured graphs $G = (V, E, x)$ with feature domain D. An edge-featured graph $G = (V, E, x, e)$ additionally provides an edge feature map $e : E \to D_E$.

Two graphs $G = (V, E)$ and $H = (V', E')$ are *isomorphic* if there exists a bijective mapping $\pi : V \to V'$ such that $(u, v) \in E \Leftrightarrow (\pi(u), \pi(v)) \in E'$. We denote this with $G \cong H$. For two featured graphs $G = (V, E, x)$ and $H = (V', E', x')$ with shared feature domain $\mathcal{X}$ we additionally require $x(v) = x'(\pi(v))$ for all $v \in V$.

For two feature maps $x_1 : V \to D_1$ and $x_2 : V \to D_2$ of the same graph $G = (V, E)$ we say that x_2 *refines* x_1 if for all $u, v \in V$ it holds that $x_2(u) = x_2(v) \Rightarrow x_1(u) = x_1(v)$ and denote this with $x_1 \sqsubseteq x_2$. We say that x_1 and x_2 are *equivalent*, denoted as $x_1 \equiv x_2$, if $x_1 \sqsubseteq x_2$ and $x_2 \sqsubseteq x_1$

A feature transformation $f : \mathcal{G}_D \to \mathcal{G}_{D'}$ is a function that maps every graph $G = (V, E, x) \in \mathcal{G}_D$ to a graph $f(G) = (V, E, y(G)) \in \mathcal{G}_{D'}$ that is structurally identical but has a different feature map $y(G) : V \to D'$ over the target domain D'. For all $v \in V$ we will write $f(G, v) = y(G)(v)$ to denote the new feature value assigned to v by f. With $\mathcal{F}_{D \to D'}$ we denote the set of all feature transformations f with input feature domain D and target domain D'. Often, we consider featured graphs with feature domains $D = \mathbb{R}^d$ for some dimension d. In this case, we may also refer to a feature map as a *vertex embedding*. We can view such an embedding as a matrix $X \in \mathbb{R}^{n \times k}$ that is the row-wise stack of embedding vectors $X_v = x(v)$ for $v \in V$. We will alternate between both views and use capital letters to denote the matrix representation. A function $f : \mathcal{G}_D \to \mathbb{R}^d$ that maps (featured) graphs to real-valued vectors is analogously referred to a *graph embedding*.

Deep Learning

We refer to a function $f_\theta : \mathbb{R}^m \to \mathbb{R}^n$ as *trainable* if it is parameterized by some vector of real parameters $\theta \in \mathbb{R}^d$ and differentiable via backpropagation. For simplicity, we usually omit the subscript θ and only write f. We will use the term "architecture" to describe the function space $\{f_\theta \mid \theta \in \mathbb{R}^d\}$ induced by all parameterizations. We will further use the terms "model" or "network" when referring to the function f_θ computed by some fixed parameter vector $\theta \in \mathbb{R}^d$.

A single-layer perceptron is a trainable function $f : \mathbb{R}^m \to \mathbb{R}^n, x \mapsto \sigma(Wx + b)$ where $W \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^n$ are trainable parameters and σ is an element-wise *activation function*, such as ReLU. A *multi-layer perceptron* (MLP) is a trainable function $f : \mathbb{R}^m \to \mathbb{R}^n, x \mapsto f^{(\ell)} \circ f^{(\ell-1)} \circ \cdots \circ f^{(1)}(x)$ that stacks a sequence of single-layer perceptrons $f^{(1)}, \ldots, f^{(\ell)}$ where ℓ is the number of layers.

Deep Learning on Graphs

In this chapter, we introduce relevant background about deep learning on graph-structured data. We use the term "Graph Neural Network" (GNN) to describe any neural network that is able to process graph-structured data. To avoid confusion we usually use more exact terms such as Message Passing Graph Neural Network (MPGNN) or Graph Transformer (GT) to refer to specific classes of GNN architectures. We first introduce the general idea of message passing as a central paradigm of graph learning in Section 2.1. Then, Section 2.2 provides an overview of relevant results on the expressivity of GNNs. In Section 2.3 we discuss precomputed features that can enhance expressivity and empirical performance. Finally, Section 2.4 introduces approaches for increasing the ability of GNNs to capture long-range dependencies between vertices, such as Virtual Nodes and Graph Transformers.

2.1 Message Passing

The concept of *message passing* (Scarselli et al., 2009, Gilmer et al., 2017) is the central paradigm on which most deep learning architectures for graphs are built. The underlying intuition is to learn a set of functions through which the vertices in a given input graph can iteratively exchange information with their neighbors. Through this exchange, each vertex extracts local structural information. By using the same functions for each vertex, the resulting model can process any graph size and structure and is inherently equivariant to vertex permutations.

Formally, a message passing layer is a tuple $\mathrm{MP} = (\mathrm{UPD}, \mathrm{AGG})$. Here, the *update* function UPD is a trainable neural network, and the *aggregation* function AGG is a differentiable order-invariant function that maps a multiset of vectors to one vector. Common choices for AGG are element-wise sum-, mean-, or max-aggregation, although some architectures use more elaborate aggregations that can also have trainable parameters. For a given graph $G = (V, E, x)$ with node feature map $x : V \to \mathbb{R}^k$ a message passing layer computes a vertex embedding $h : V \to \mathbb{R}^d$ as follows:

$$h(v) \coloneqq \mathrm{UPD}\Big(x(v),\ \mathrm{AGG}\big(\{\!\{x(u) \mid u \in \mathcal{N}_G(v)\}\!\}\big)\Big). \tag{2.1}$$

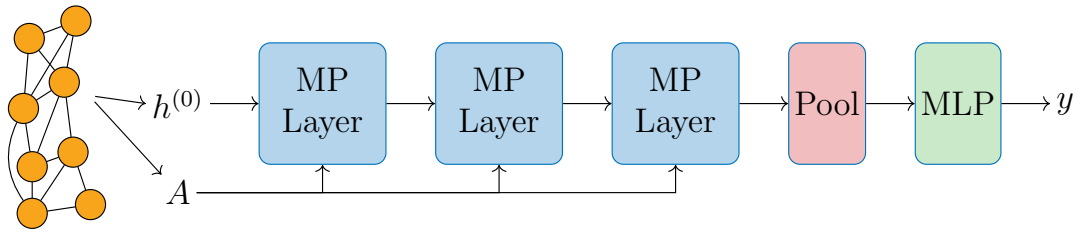


Figure 2.1: Illustration of a graph-level MPGNN. Each layer refines the latent embedding by performing a message pass. The final embedding is pooled and passed through a final MLP to obtain a graph-level prediction.

A Message Passing Graph Neural Network (MPGNN) sequentially combines $L \in \mathbb{N}$ message passing layers $\mathrm{MP}^{(1)}, \ldots, \mathrm{MP}^{(L)}$ to iteratively update latent vertex embeddings $h^{(0)}, h^{(1)}, \ldots h^{(L)}$. For $\ell \geq 1$, the embedding $h^{(\ell)} : V \to \mathbb{R}^d$ is the output of the ℓ-th message passing layer and is passed as a node feature map to layer $\ell + 1$. The node feature map of the input graph $x : V \to \mathbb{R}^k$ is used as the initial embedding $h^{(0)} = x$ that is passed as input to $\mathrm{MP}^{(1)}$. The final vertex embedding $h^{(L)}$ is then mapped to a prediction by a *readout* function R, which is also a trainable neural network. For a node-level task, the readout function $\mathrm{R} = (\mathrm{M})$ simply applies an MLP $\mathrm{M} : \mathbb{R}^d \to \mathbb{R}^n$ which yields a node-level prediction $y(v) = \mathrm{M}(h^{(L)}(v)) \in \mathbb{R}^n$. Here, $n \in \mathbb{N}$ is the output dimension of the MPGNN. To obtain a graph-level output, the readout function $\mathrm{R} = (\mathbf{C}, \mathrm{Pool})$ first applies a *pooling* function Pool to reduce the vertex embedding $h^{(L)}(v)$ to a graph embedding before applying M:

$$y(G) = \mathrm{M}\left(\mathrm{Pool}\left(\{\!\{h^{(L)}(v) \mid v \in V\}\!\}\right)\right). \tag{2.2}$$

The pooling function can be any differentiable order-invariant function that maps a multiset of vectors to one vector. Common choices are element-wise sum-, mean- or max-functions, similar to the aggregation function in the message passing layers. Figure 2.1 provides an illustration of a graph-level MPGNN with 3 message passing layers.

Formally, an MPGNN model $N = (\mathrm{MP}^{(1)}, \ldots, \mathrm{MP}^{(L)}, \mathrm{R})$ is a tuple that combines the message passing layers and the readout function. A node-level MPGNN N computes a feature transformation $f_N \in \mathcal{F}_{\mathbb{R}^k \to \mathbb{R}^n}$. A graph-level MPGNN computes a graph embedding function $f_N : \mathcal{G}_{\mathbb{R}^k} \to \mathbb{R}^n$. In most contexts, we will refer to the model N and its computed function f_N interchangeably. The parameters of an MPGNN model can be learned end-to-end with gradient descent on available training data. MPGNNs are inherently equipped with some highly useful properties for learning on graph-structured data, namely *equivariance*, *efficiency*, and *locality*, which we discuss in the following.

Equivariance For most graph learning tasks the predictions should depend on the graph's structure and features but not the arbitrary order of nodes in the graph representation. A crucial property of MPGNNs is their inherent *equivariance* to vertex permutations. Formally, let $G = (V, E, x)$ and $H = (V', E', x')$ be two

isomorphic graphs with isomorphism $\pi : V \to V'$. Then for every layer $\ell \geq 0$ of an MPGNN the following holds for all vertices $v \in V$ in G:

$$h_G^{(\ell)}(v) = h_H^{(\ell)}(\pi(v)). \tag{2.3}$$

Here, $h_G^{(\ell)}$ and $h_H^{(\ell)}$ are the latent vertex embeddings computed by the ℓ-th layer for the input graphs G and H, respectively. It is easy to see that this holds by construction, as long as the used aggregation function is order invariant. The embedding of each vertex is therefore not affected by permutations and only depends on the structural features of the graph. For a graph-level MPGNN, this further implies that the final graph-level prediction obtained after pooling the vertex embeddings is *permutation invariant* and will always be identical for two isomorphic input graphs. The equivariance of MPGNNs is analogous to the shift-invariance of CNNs on image data (LeCun et al., 1989) and the order-invariance of DeepSets (Zaheer et al., 2017) and Transformers (Vaswani et al., 2017) on sets and multisets. By designing neural networks that are inherently invariant to the symmetries of their input data, we enable them to focus on learning the underlying patterns in the data, rather than expending resources on learning to recognize numerous permutations or shifts of the same pattern.

Efficiency The computation of a message passing layer is generally linear in the size of the input graph as the runtime and memory scale in $\mathcal{O}(|V|+|E|)$. MPGNNs are particularly efficient on sparse inputs since messages are only sent along existing edges. This is a very useful property, as many applications work with highly sparse graphs, such as molecules or social networks. MPGNNs are typically implemented using GPU-accelerated sparse matrix operations and are therefore highly scalable and inherently parallel. In some applications, MPGNNs can process graphs with well over 100K vertices on a single GPU (Wang et al., 2019).

Locality The computation of an MPGNN is inherently local. More specifically, the embedding vector $h^{(\ell)}(v)$ of any vertex $v \in V$ after ℓ message passes depends only on the local ℓ-hop neighborhood of v in the input graph G. This is a key advantage when solving node- or link-level tasks on large graphs. In some domains, such as social networks, graphs can grow to billions of vertices. Passing a graph of this size through an MPGNN at once would be prohibitively expensive. However, when computing a prediction for a target vertex $v \in V$ in a large graph G with a depth-L MPGNN it is sufficient to apply the model to the local L-hop neighborhood of v. In practice, it is common to improve efficiency further by sampling a bounded-size subgraph from the L-hop neighborhood with a suitable sampling strategy (Hamilton et al., 2017, Zeng et al., 2019, Hu et al., 2020a, Zeng et al., 2021). This approach enables MPGNNs to be trained on web-scale graphs, even with moderate hardware. Depending on the context, the locality of message passing can also be viewed as a limitation as it bounds the distance that information can propagate within a fixed number of layers. This concern has received much interest in recent years and many approaches have been proposed to improve long-range interactions (LRI) in GNNs, which we discuss in greater detail in Section 2.4.

2.1.1 Message Passing Layers

Concrete MPGNN architectures make specific choices for the update and aggregation function in the general formulation of Equation 2.1. A wide range of MPGNNs has been suggested with varying design objectives such as efficiency, expressivity, support for edge features, stable gradients, prevention of over-smoothing or over-squashing (see Section 2.4), and support of special graph types, such as temporal or heterogeneous graphs. In the following, we will discuss several popular MPGNN architectures and variations of the message passing formalism. We refer to (Zhou et al., 2020) for a comprehensive taxonomy of MPGNNs.

Graph Convolutional Network Kipf and Welling (2017) proposed the Graph Convolutional Network (GCN) which is motivated as an approximation of spectral graph convolutions (Hammond et al., 2011). It is among the most common architectures throughout the graph learning literature and has been studied extensively in both theory and practice. Formally, let $X \in \mathbb{R}^{|V| \times k}$ be the feature matrix of an input graph $G = (V, E, x)$. Then a GCN layer computes a vertex embedding $H \in \mathbb{R}^{|V| \times d}$ with the following rule:

$$H = \sigma(\tilde{A} \cdot X \cdot W) \tag{2.4}$$

Here, $W \in \mathbb{R}^{k \times d}$ is a learnable weight matrix, σ denotes a non-linear activation function, and $\tilde{A}$ represents the normalized adjacency matrix of the input graph G with added self-loops:

$$\tilde{A} = \hat{D}^{\frac{1}{2}} \hat{A} \hat{D}^{\frac{1}{2}}, \tag{2.5}$$

with $\hat{A} = A + I$ and diagonal matrix $D \in \mathbb{R}^{|V| \times |V|}$ with $D_{i,i} = \deg(v) + 1$. GCN is notable for its simplicity and efficiency as each layer only requires two simple matrix multiplications to update all vertex embeddings in parallel. Despite its popularity, GCN also has known drawbacks as it is strongly affected by problems such as over-smoothing (Li et al., 2018) and over-squashing (Alon and Yahav, 2021) which limits its performance in some use cases (see Section 2.4). Several modifications of GCN have been proposed to overcome these limitations (Li et al., 2019, Chen et al., 2020a).

Graph Isomorphism Network The Graph Isomorphism Network (GIN) was proposed by Xu et al. (2019). The aim of this architecture is to construct a maximally expressive MPGNN architecture in the sense that its ability to distinguish graphs is provably equivalent to the 1-dimensional Weisfeiler-Leman (WL) graph isomorphism test (see Section 2.2.1). GIN combines sum-aggregation with a deep update function. For a given feature map $x : V \to \mathbb{R}^d$ a GIN layer computes the following:

$$h(v) = \mathrm{MLP}\left((1 + \epsilon) \cdot x(v) + \sum_{w \in \mathcal{N}(v)} x(w)\right). \tag{2.6}$$

Here, MLP is a deep multi-layer perceptron and $\epsilon \geq 0$ is either a trainable parameter or a fixed hyperparameter. GINE (Hu et al., 2020b) is a common extension of GIN that is able to incorporate additional edge features. Let $G = (V, E, x, e)$ be a graph with node feature map $x : V \to \mathbb{R}^k$ and edge feature map $e : E \to \mathbb{R}^k$. Then a GINE layer computes the following:

$$h(v) = \text{MLP}\left((1+\epsilon)\cdot x(v) + \sum_{w \in \mathcal{N}(v)} \text{ReLU}\left(x\left(w\right) + e\left(v, w\right)\right)\right) \tag{2.7}$$

Handling edge features is crucial in some domains. A common example is molecular data where different types of chemical bonds are naturally modeled as edge features in a molecular graph. It is straightforward to extend the message passing formalism towards edge features by fusing them with the node features before the aggregation function is applied. GINE is a simple example of such a modification.

Attentive Architectures Attention has emerged as a central paradigm in deep learning (Bahdanau et al., 2015, Vaswani et al., 2017). The underlying idea is to aggregate embeddings with weighted averages for which the weights are inferred by the model. Attentive architectures can therefore dynamically assign larger weights to high-importance features. This concept has naturally been incorporated into a range of MPGNN architectures that use variations of attentive aggregation functions. A well-known example is the Graph Attention Network (GAT) (Veličković et al., 2017), which uses a form of additive attention (Bahdanau et al., 2015). Formally, given a vertex embedding $x : V \to \mathbb{R}^k$ of a given input graph $G = (V, E)$, a GAT layer computes a scalar attention score $\alpha_{vw} \in \mathbb{R}$ for every directed edge $(v, w) \in E$:

$$\alpha_{vw} = \frac{\exp\left(\text{LeakyReLU}\left(a^T[Wx(v) \| Wx(w)]\right)\right)}{\sum_{w' \in \mathcal{N}(v)\cup\{v\}} \exp\left(\text{LeakyReLU}\left(a^T[Wx(v) \| Wx(w')]\right)\right)}. \tag{2.8}$$

Here, $W \in \mathbb{R}^{d\times k}$ and $a \in \mathbb{R}^{2d}$ are trainable parameters and $[\cdot\|\cdot]$ denotes concatenation. An updated latent vertex embedding h is then obtained as follows:

$$h(v) = \sigma\left(\sum_{w \in \mathcal{N}(v)\cup\{v\}} \alpha_{vw} W x(w)\right) \tag{2.9}$$

This attention mechanism aims to enable the model to dynamically adjust the influence of each node's neighbors based on their features. Later work by Brody et al. (2022) identifies weaknesses in the specific attention mechanism used by GAT and suggest an improved version of the layer. Other works, such as Shi et al. (2021) and Dwivedi and Bresson (2021), propose MPGNN layers that utilize dot-product-based attention that is more similar to the attention mechanism used by Transformers (Vaswani et al., 2017).

Recurrent Architectures One variation of MPGNNs are *recurrent* architectures which train a single message passing layer that is reused in each message passing step in a recurrent manner. In this setting, the update function can be a recurrent cell such as an LSTM- (Hochreiter and Schmidhuber, 1997) or GRU-Cell (Cho et al., 2014). With this modification, recurrent MPGNNs are not bound to a fixed number of message passes. This can be useful in applications where the desired number of message passes is either extremely large or varies with the input instance. A common example is learning heuristics for combinatorial optimization, which may require a large number of search steps, each corresponding to one or more message passes. With a recurrent architecture, an MPGNN can continually propagate information on a given input graph until some external stopping criterion is reached, such as a timeout. Some of the earliest works on GNNs studied recurrent architectures (Sperduti and Starita, 1997, Gori et al., 2005, Scarselli et al., 2009, Li et al., 2015), but more recent research has primarily focused on non-recurrent feedforward architectures with fixed depth. We will make extensive use of recurrent MPGNNs in Chapter 3 to learn heuristics for constraint satisfaction problems. There, we demonstrate that a recurrent MPGNN can solve hard combinatorial problems after performing over 100K message passes on some problem instances. This could not be done with non-recurrent feedforward architectures as the number of required layers would be intractable.

Heterogeneous Architectures Another popular adaptation of the message passing framework are *heterogenous* MPGNNs (HetGNNs). These aim to solve learning tasks for heterogeneous graphs $G = (V, E, \tau_V, \tau_E)$, where the vertices and edges are assigned to types by the mappings $\tau_V : V \to \mathcal{T}_V$ and $\tau_E : E \to \mathcal{T}_E$, respectively. Such typed graphs naturally occur in a range of domains. Citation graphs are a common example where nodes may have types such as "Paper" or "Author" and edges represent different types of relationships such as "authored by" or "reviewed by". Generally, HetGNNs propose to incorporate type information by learning separate message passing functions for each type. An early architecture based on this idea is RGCN (Schlichtkrull et al., 2018), which applies a type-dependent linear transformation to the messages sent along each edge. Another line of work, such as HAN (Xiao et al., 2019) and HGT (Hu et al., 2020a), proposes heterogeneous attention mechanisms. It should be noted that types can also be viewed as categorical features and can therefore also be passed to a regular (homogeneous) MPGNN architecture that supports edge features. While receiving much research interest recently (Shi et al., 2022), the theoretical foundation of HetGNNs is still comparatively underdeveloped and reproducibility studies have also questioned the empirical advantage (Lv et al., 2021) over their simpler homogeneous counterparts. Identifying expressive and effective means for learning on heterogeneous graphs therefore remains an active field of study, with open questions to be answered.

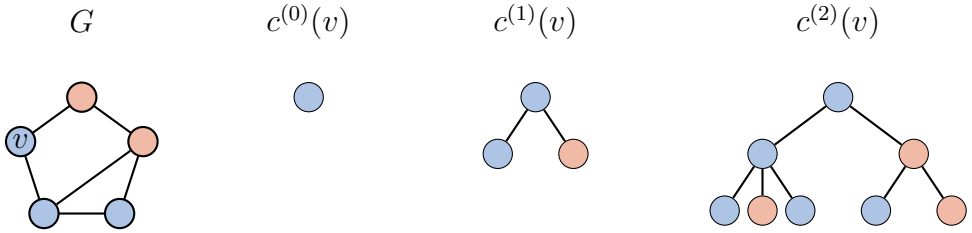


Figure 2.2: For a graph $G = (V, E, x)$ and vertex $v \in V$ the 1-WL color $c^{(\ell)}(v)$ after ℓ rounds of refinement uniquely identifies the subtree of depth ℓ rooted in v. Here, the red and blue colors illustrate binary node features $x : V \to \{\blacksquare, \blacksquare\}$.

2.2 Expressivity of Graph Neural Networks

Expressivity is a key subject of GNN research that aims to formally characterize the space of functions that different architectures can express. With their inherent connection to graph theory, the expressivity of GNNs has been studied in a plethora of theoretical contexts, such as the Weisfeiler-Leman (WL) algorithm (Morris et al., 2019, Xu et al., 2019), finite variable counting logic (Barceló et al., 2020, Grohe, 2021a), Boolean circuit complexity (Grohe, 2023), uniform function approximation (Rosenbluth et al., 2023), and Vapnik–Chervonenkis (VC) dimension (Morris et al., 2023). Here, we first discuss the connection between MPGNNs and the WL algorithm, which provides insights into the ability of MPGNN to capture graph structure. In Section 2.2.2 and Section 2.2.3 we then examine two works that focus on the ability to generalize to new data after training by studying expressivity in terms of uniform function approximation and the VC dimension, respectively.

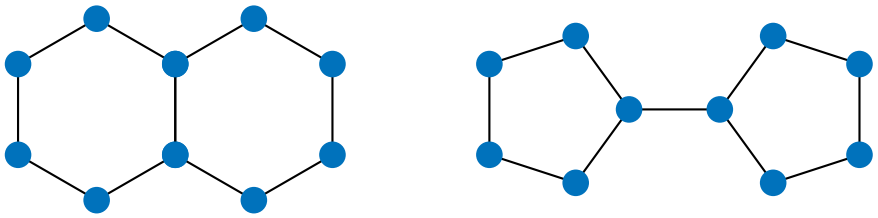
2.2.1 The Weisfeiler-Leman Algorithm

Let $G = (V, E, x)$ be a featured graph. The one-dimensional Weisfeiler-Leman algorithm (1-WL) iteratively updates a vertex coloring $c^{(t)} : V \to \Sigma$, where Σ is a suitable codomain of colors. The initial coloring is set to the vertex features $c^{(0)}(v) = x(v)$ for all $v \in V$. In round $t \in \mathbb{N}$ the coloring is then refined using the following update rule:

$$c^{(t+1)}(v) = \mathrm{HASH}\Big(\big(c^{(t)}(v), \{\!\{c^{(t)}(u) \mid u \in \mathcal{N}(v)\}\!\}\big)\Big). \tag{2.10}$$

Here, HASH is an *injective* hash function that projects the given tuple to a unique new color. The algorithm terminates once a stable coloring $c^{(t)}$ with $c^{(t)} \equiv c^{(t+1)}$ is reached, which takes at most $|V|$ steps of refinement. We say that 1-WL *distinguishes* two graphs $G = (V, E, x)$ and $H = (V', E', x')$ after t iterations if the multiset of colors over all vertices differs after t steps:

$$\{\!\{c^{(t)}(v) \mid v \in V\}\!\} \neq \{\!\{c^{(t)}(v) \mid v \in V'\}\!\}. \tag{2.11}$$



a) Decalin b) Bicyclopentyl

Figure 2.3: Two non-isomorphic real-world graphs that are not distinguished by 1-WL. The graphs model the molecular structure of Decalin and Bicyclopentyl, respectively. The vertices represent carbon atoms.

The 1-WL algorithm is a heuristic for graph isomorphism that correctly distinguishes a large class of graphs (Babai and Kucera, 1979). However, it is an incomplete heuristic that fails to distinguish certain non-isomorphic graphs (Cai et al., 1992). Figure 2.3 provides an example of two non-isomorphic graphs which are not distinguished by 1-WL.

It is straightforward to see that the message passing procedure of MPGNNs is conceptually similar to the 1-WL algorithm. This connection was already proposed by Kipf and Welling (2017) as an informal intuition that views MPGNNs as a differentiable and trainable variation of the 1-WL algorithm where the vertex embeddings serve as "colors". Subsequent work formalized this relationship by studying the power of MPGNNs to distinguish graphs. Here, an MPGNN distinguishes two graphs if the multisets of vertex embeddings after message passing differ. A central theoretical result of graph learning is that the ability of standard MPGNNs to distinguish graphs is exactly that of 1-WL.

Theorem 2.1 (Morris et al. (2019), Xu et al. (2019))**.** *For every pair of (featured) graphs $G = (V, E, x)$ and $H = (V', E', x')$ the following two statements are equivalent:*

(1) 1-WL distinguishes G and H.

(2) There exists an MPGNN that distinguishes G and H.

Morris et al. (2019) and Xu et al. (2019) provide constructive proofs of Theorem 2.1 by providing a specific MPGNN that distinguishes G and H. Note that this model depends on the given graphs G and H. The theorem is therefore set in a *non-uniform* setting, as opposed to the *uniform* case where a single MPGNN would have to "work" for all graphs of all sizes (see Section 2.2.2).

Theorem 2.1 has broad implications for MPGNNs in practice. In particular, a standard MPGNN can only distinguish two graphs if these are also distinguished by 1-WL, which can be a major limitation. Consider the two molecular graphs in Figure 2.3 that 1-WL can not distinguish. A graph-level MPGNN can never map these two graphs to a different prediction, even if the true target values differ in the underlying learning task. Without additional features or architectural

modifications standard MPGNNs may therefore struggle at learning tasks that depend strongly on structural details, which is often the case in applications such as computational chemistry.

Notably, the MPGNNs constructed by Morris et al. (2019) and Xu et al. (2019) to prove Theorem 2.1 rely on sum-aggregation in each layer. Xu et al. (2019) further investigate the role of the aggregation function in this context by comparing the expressive power of sum-, mean-, and max-aggregation. They show that there are graphs that can be distinguished by an MPGNNs when using sum-aggregation but not with mean- or max-aggregation. Sum-aggregation is therefore strictly more powerful in terms of distinguishing graphs in a non-uniform setting.

2.2.1.1 k-WL

The 1-WL algorithm can be generalized to the k-dimensional Weisfeiler-Leman algorithm (k-WL) for a given integer $k \geq 2$. The k-WL algorithm assigns a color $c_k^{(t)}(\mathbf{v})$ to every k-tuple of vertices $\mathbf{v} \in V^k$. For $j \in [k]$ the j-th neighborhood of $\mathbf{v} = (v_1, \ldots, v_k) \in V^k$ is defined as $\mathcal{N}_{k,j} = \{(v_1, \ldots, v_{j-1}, u, v_{j+1}, \ldots, v_k) \mid u \in V\}$. The initial coloring $c_k^{(0)}$ is chosen such that $c_k^{(0)}(\mathbf{v}) = c_k^{(0)}(\mathbf{u})$ for two tuples $\mathbf{v} = (v_1, \ldots, v_k), \mathbf{u} = (u_1, \ldots, u_k) \in V^k$ if the mapping $v_i \mapsto u_i$ induces a labeled isomorphism between the induced subgraphs of $\mathbf{v}$ and $\mathbf{u}$. These colors are then updated iteratively as follows:

$$C_k^{(t+1)}(\mathbf{v}) = \text{HASH}\Big(\{\!\{c_k^{(t)}(\mathbf{u}) \mid \mathbf{u} \in \mathcal{N}_{k,j}(\mathbf{v})\}\!\}\Big) \tag{2.12}$$

$$c_k^{(t+1)}(\mathbf{v}) = \text{HASH}\Big(\big(c_k^{(t)}(\mathbf{v}), (C_{k,1}^{(t+1)}(\mathbf{v}), \ldots, C_{k,k}^{(t+1)}(\mathbf{v}))\big)\Big) \tag{2.13}$$

Here, HASH is once again an injective function. It has been shown that $(k+1)$-WL is more powerful than k-WL in terms of distinguishing non-isomorphic graphs for all $k \geq 2$ (Cai et al., 1992) and that 3-WL is strictly more powerful than 1-WL. We note that Cai et al. (1992) study a variation of the algorithm known as *folklore* k-WL (k-FWL) that uses a slightly different update rule. For $k \geq 2$ it holds that $(k+1)$-WL as defined in Equation 2.12 is as powerful as k-FWL (Grohe and Otto, 2015, Huang and Villar, 2021).

To address the expressive limitations of standard MPGNNs, Morris et al. (2019) propose higher-dimensional MPGNNs (k-GNNs) as an analogous generalization of standard message passing. A k-GNN performs message passing between k-tuples of vertices with trainable functions. Morris et al. (2019) prove that for every $k \in \mathbb{N}$ their k-GNNs are equivalent to (set-based) k-WL in terms of their ability to distinguish graphs and are therefore more powerful than standard MPGNNs. Several variations and improvements of higher-order GNNs have since been suggested (Maron et al., 2019, Geerts, 2020a, Azizian and Lelarge, 2020, Morris et al., 2022). As a downside, the space and runtime requirements of higher-order GNNs tend to scale poorly for increasing k, often exponentially.

2.2.1.2 The Weisfeiler-Leman Graph Kernel

The Weisfeiler-Leman Graph Kernel (Shervashidze et al., 2011) is a popular non-neural learning method for graphs. This method projects a given labeled graph to a feature vector that is the histogram of vertex colors that occur during a fixed number of 1-WL iterations (or k-WL for higher dimensional versions). The kernel induced by the inner product of these histograms can then be combined with Support Vector Machines (SVMs) to solve learning tasks over graphs.

Theorem 2.1 implies that the latent vertex embeddings extracted by a standard MPGNN do not capture more structural information about the input graph than the feature vectors of the WL graph kernel. One could therefore wonder what advantage standard MPGNNs have over the WL graph kernel. To answer this question it is important to note that distinguishing graph structure is only one aspect of learning functions over graphs, which is ultimately about approximating a target function in a way that generalizes well to unseen data. At test time, the WL graph kernel ignores any colors that did not occur in the training data as it has no notion of similarity between the observed vertex colors. In contrast, MPGNNs can learn to discard negligible differences between similar substructures and thereby express a more robust function that generalizes better to new data. Additional advantages of MPGNNs over the WL graph kernel are their ability to seamlessly handle non-discrete features, scale to large datasets, and be integrated end-to-end with other deep learning systems. Nonetheless, the WL graph kernel can still be competitive with MPGNNs on some tasks, especially on smaller datasets as observed by both Morris et al. (2019) and Xu et al. (2019).

2.2.2 Uniform Function Approximation

Many theoretical results on MPGNNs, such as Theorem 2.1, are in a non-uniform setting, where a concrete GNN model is only shown to express a certain function for graphs up to some bounded size. Uniform function approximation represents an alternative notion of expressivity. Intuitively, it asks if a class of GNNs contains one model that can approximate a given target function for all possible input graphs of all sizes. Uniform function approximation is therefore closely linked to the ability of an architecture to compute functions that generalize to out-of-distribution instances after training.

Formally, fix some feature domain $D \subseteq \mathbb{R}^d$ and some class C of node-level GNNs. Let $f^* \in \mathcal{F}_{D\to\mathbb{R}}$ be some target function for node-level regression. We say that C *uniformly additively approximates* f^*, denoted by $C \approx f^*$, if and only if for all $\epsilon > 0$ there exists $f \in C$:

$$\forall G = (V, E, x) \in \mathcal{G}_D \; \forall v \in V : \; |f(G, v) - f^*(G, v)| \leq \epsilon \tag{2.14}$$

We may equivalently say that C expresses f^* uniformly. Note that this definition considers node-level regression tasks. Uniform approximation for graph-level tasks

can be defined analogously. For two different classes of GNNs C and C' we say that C *subsumes* C' if $C \approx f'$ holds for all $f' \in C'$. We denote this with $C' \leq^D C$. Whether or not one class of GNNs is subsumed by another can depend on the considered feature domain D, which is why we add it as a superscript in this notation.

In Rosenbluth et al. (2023), we study the uniform expressivity of sum-, mean- and max-aggregation. Sum-GNNs are defined as the class of node-level MPGNNs that use element-wise sum as an aggregation function in all layers. The update function of each layer and the final readout function are assumed to be MLPs with piecewise linear activation functions, such as ReLU. Mean-GNNs and Max-GNNs are defined analogously. We show that the relationship of these GNN classes in terms of uniform expressivity is intricate and depends on the specific feature domain D. We summarize these results in the following.

Bounded Feature Domains If the features are restricted to some bounded domain $D \subseteq [a, b]^d$ with $a, b \in \mathbb{R}$ it is shown that Sum-GNNs strictly subsume Mean-GNNs and Max-GNNs. Furthermore, Mean-GNNs do not subsume Max-GNNs and vice versa.

Theorem 2.2 (Rosenbluth et al. (2023))**.** *Let $D \subseteq \mathbb{R}^d$ be some bounded feature domain. Then the following statements hold:*

(1) Mean-GNNs $\lneq^D$ Sum-GNNs

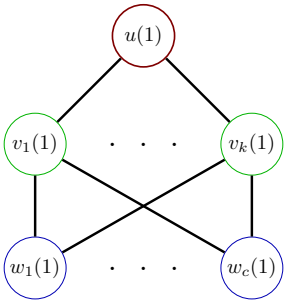
(2) Max-GNNs $\lneq^D$ Sum-GNNs

(3) Mean-GNNs $\not\leq^D$ Max-GNNs

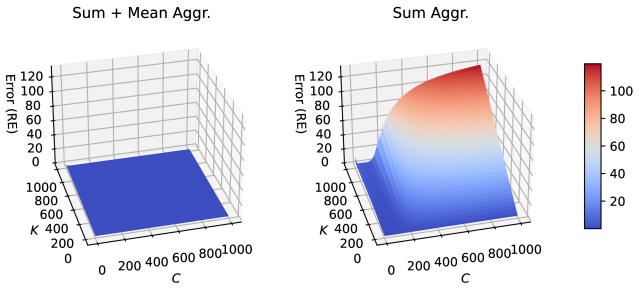
(4) Max-GNNs $\not\leq^D$ Mean-GNNs

In this setting, sum-aggregation is therefore strictly more powerful than mean and max in terms of uniform expressivity. To prove this theorem, we construct a uniformly approximating Sum-GNN for every function f computed by a Mean- or Max-GNN. Note that the constructed Sum-GNN is larger than the underlying Mean/Max-GNN by a factor of $\Theta(\frac{dL}{\epsilon})$, where d and L are the hidden dimension and depth of the Mean/Max-GNN, respectively. While Theorem 2.2 states that sum-aggregation subsumes mean- and max-aggregation, it does not guarantee that Sum-GNNs are the most efficient architecture to approximate a certain target function or that the specific construction used in the proof can be learned with stochastic gradient descent.

Unbounded Feature Domains Theorem 2.2 does not apply if the feature domain is unbounded. In this case, it is shown that neither mean- nor max-aggregation is subsumed by sum in terms of uniform expressivity, as stated formally in the following.



(a) A graph $G_{k,c}$ with parameters $k, c \in \mathbb{N}$ as used in the proof of Theorem 2.4



(b) Relative error of MPGNN models with different aggregation functions as a function of the parameters k and c.

Figure 2.4: Experiments on learning to approximate the target function that separates Sum+Mean-GNNs and Sum-GNNs in the proof of Theorem 2.4. We provide the measured relative test error (RE) for different choices for the parameters c and k.

Theorem 2.3 (Rosenbluth et al. (2023)). *Let $D \subseteq \mathbb{R}^d$ be some unbounded, countable feature domain. Then the following statements hold:*

(1) Mean-GNNs $\not\leq^D$ Sum-GNNs

(2) Max-GNNs $\not\leq^D$ Sum-GNNs

The proof of Theorem 2.3 is based on a simple function f where a vertex v has to output the common node feature $c \in D$ of its neighbors in cases where all neighbors have the same feature. While mean- and max-aggregation can trivially compute such a function, it can be shown that Sum-GNNs can not approximate f uniformly across all graph sizes. It should be noted that in practice the input features of neural networks are often normalized to a bounded range, which limits the applicability of Theorem 2.3.

Multi-Aggregation We additionally consider the case of multi-aggregation where several aggregation functions are used in parallel. For example, Sum+Mean-GNNs perform both sum- and mean-aggregation in every message passing layer and concatenate the results before passing it to the update function. It can be shown that this extension is strictly more powerful than only using sum, even when the feature domain is finite (and therefore bounded).

Theorem 2.4 (Rosenbluth et al. (2023)). *Let $D \subseteq \mathbb{R}^d$ be some finite feature domain. Then the following two statements hold:*

(1) Sum-GNNs $\lneq^D$ Sum+Mean-GNNs

(2) Sum-GNNs $\lneq^D$ Sum+Max-GNNs

The proof of this theorem relies on a family of graphs $\{G_{k,c}\}_{k,c\in\mathbb{N}}$ dependent on two parameters $k, c \in \mathbb{N}$ as illustrated in Figure 2.4a. The vertices of these graphs all have a constant node feature of 1 and the target function f^* is defined such that the vertex u (shown in red in Figure 2.4a) has to output the value c. It is shown that no Sum-GNN can approximate this function with bounded error across all choices for k and c, while a Sum+Mean-GNN can compute the target function exactly. In Rosenbluth et al. (2023) we further provide supplementary experiments to evaluate this result empirically. They train a Sum-GNN and a Sum+Mean-GNN for the target function f^* on training graphs where $k, c \leq 100$ and then test the generalization of these models to larger graphs with $k, c > 100$. The reported results are provided in Figure 2.4b and show that the Sum+Mean-GNN has learned a function that generalizes out-of-distribution while the Sum-GNN has not. The theoretical results of Theorem 2.4 therefore translate to practice in the sense that the uniformly expressing class of GNNs can in fact learn a generalizing function from data using standard SGD-based optimization.

Combining sum-aggregation with other functions can therefore strictly strengthen the uniform expressivity when compared to only using sum. We note that some popular MPGNN architectures, such as PNA (Corso et al., 2020), are based on multi-aggregation. These tend to perform better than single-aggregation architectures on some real-world learning tasks, an observation that generally aligns with the provably stronger uniform expressivity in Theorem 2.4.

2.2.3 VC Dimension

The Vapnik-Chervonenkis (VC) dimension is a concept from statistical learning theory that measures the capacity of a model or hypothesis space to effectively fit arbitrary patterns of input data (Vapnik, 1995, Mohri et al., 2018). The VC dimension has been shown to have strong connections to a model's generalization ability. A model with a higher VC dimension can fit more complex data patterns but might also be more prone to overfitting and therefore worse at generalizing to unseen data.

In Morris et al. (2023), we study the VC dimension of MPGNNs. We prove multiple bounds that characterize how the VC dimension depends on different properties of the MPGNNs and input graphs, such as model size and the complexity of the graph structure. These results therefore provide insights into how these factors impact the ability of MPGNNs to generalize to new data after training.

Formally, let C be some class of graph-level GNNs and $\mathcal{G}$ be a class of input graphs. We say that a set of graphs $G_1, \ldots, G_m \in \mathcal{G}$ is *shattered* by C if for any $\tau \in \{0,1\}^m$ there exists a GNN $f \in C$ such that for all $i \in [m]$:

$$f(G_i) = \begin{cases} \geq 2/3 & \text{if } \tau_i = 1, \text{ and} \\ \leq 1/3 & \text{if } \tau_i = 0. \end{cases} \tag{2.15}$$

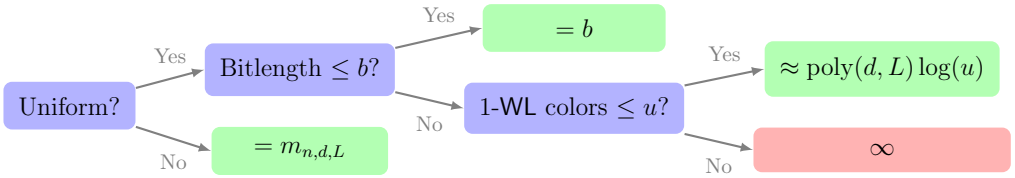


Figure 2.5: An illustration of the different bounds for the VC dimension of MPGNNs. (Image Credit: Morris et al. (2023))

Intuitively, a set of inputs is shattered if the given class of functions can express all possible binary target functions. The VC dimension $\mathsf{VC\text{-}dim}_{\mathcal{G}}(C)$ is then defined as the maximal number m of graphs $G_1, \ldots, G_m$ in $\mathcal{G}$ that can be shattered by C.

We study $\mathsf{VC\text{-}dim}_{\mathcal{G}}(C)$ for a class C of standard sum-aggregation MPGNNs with $L \in \mathbb{N}$ layers. The input graphs are assumed to have d-dimensional binary features. In this setting, we derive multiple bounds for the VC dimension:

(1) In a non-uniform setting, $\mathcal{G}$ only contains graphs $G = (V, E) \in C$ of bounded order $|V| \leq n \in \mathbb{N}$. In this case, $\mathsf{VC\text{-}dim}_{\mathcal{G}}(C)$ is shown to equal $m_{n,d,L}$, which is defined as the number of graphs with order up to n and binary feature dimension d that are distinguished by L rounds of the 1-WL algorithm.

(2) In the uniform setting with unbounded graph size $\mathsf{VC\text{-}dim}_{\mathcal{G}}(C)$ is shown to be lower and upper bounded by the *bitlengths* of the MPGNNs model, which is proportional to the number of model parameters in practice.

(3) The VC dimension is further shown to depend logarithmically on the *color complexity* u, that is the number of distinct 1-WL colors for a graph.

Figure 2.5 summarizes the main observations in a flow chart. Overall, the VC dimension of MPGNNs is shown to increase with both the model size and the structural complexity of the input graphs. These factors therefore also contribute to the risk of overfitting and increased generalization error.

In Morris et al. (2023), we further provide an experimental evaluation of MPGNNs of different sizes (bitlengths). We measure the generalization error on real-world data and quantify the ability of the models to shatter synthetic data. The observed results are consistent with the theoretical bounds on the VC dimension.

2.3 Positional and Structural Encodings

An intuitive way of strengthening the expressive power of GNNs is to pre-compute graph features that are inaccessible due to limited expressivity and pass them to the model as additional vertex or edge features. Such encodings have been shown to significantly improve performance on a range of tasks (Rampášek et al., 2022). As a downside, the added features introduce computational overhead and

additional hyperparameters. One line of work focuses on *structural* encodings (SE) that measure the counts or frequencies of specific structural features, such as closed random walks (Dwivedi et al., 2021) or predefined graph motifs (Bouritsas et al., 2020, Sankar et al., 2017, Lee et al., 2019, Peng et al., 2020). Another line of work focuses on *positional* encodings (PE) that capture the global structural relationships between vertices. Many of these approaches are based on encoding spectral features (Dwivedi et al., 2020, Kreuzer et al., 2021, Wang et al., 2022, Lim et al., 2023, Huang et al., 2024). PEs for graphs can be viewed as generalizations of sequential positional encodings used by Transformers (Vaswani et al., 2017) for language modeling. Consequently, a major use case of PEs are Graph Transformers (see Section 2.4), which often rely on these encodings to inject structural information into the model. In the following, we will discuss two exemplary encodings that are frequently used throughout the literature, namely RWSE and LapPE.

2.3.1 Random Walk Structural Encodings

Random Walk Structural Encodings (RWSE) (Dwivedi et al., 2021) are based on a random walk diffusion process. Formally, let $W = AD^{-1} \in \mathbb{R}^{|V|\times|V|}$ be the normalized adjacency matrix of the input graph $G = (V, E)$ where for each vertex the weights of outgoing edges sum up to one. RWSE defines a vertex feature map $x_{\mathrm{SE}}(G) : V \to \mathbb{R}^k$ for G as follows:

$$x_{\mathrm{SE}}(G, v) = (W_{v,v}, W^2_{v,v}, \ldots, W^k_{v,v}) \in \mathbb{R}^k \tag{2.16}$$

Here, the dimension $k \in \mathbb{N}$ is a hyperparameter. By construction, the i-th dimension of the RWSE features contains the probability that a random walk of length i that starts at vertex $v \in V$ is a *closed* walk (i.e. a walk that also ends on v). These features therefore encode information on the occurrence of cycles in the vicinity of each vertex. RWSE has been observed to be highly effective for many molecular learning tasks (Dwivedi et al., 2021, Rampášek et al., 2022), where the distribution of cycles is a key feature that is not directly accessible to standard MPGNNs.

2.3.2 Laplacian Positional Encodings

Laplacian Positional Encodings (LapPE) (Dwivedi et al., 2020, Kreuzer et al., 2021) are based on the eigendecomposition of the graph Laplacian. Formally, the Laplacian matrix of a given input graph $G = (V, E)$ is defined as $L = D - A$ where $A \in \mathbb{R}^{|V|\times|V|}$ and $D \in \mathbb{N}^{|V|\times|V|}$ are the adjacency matrix and the diagonal degree matrix, respectively. Let $\Sigma \in \mathbb{R}^{|V|\times|V|}$ be the matrix of eigenvectors of L and let $\lambda \in \mathbb{R}^{|V|}$ be the corresponding vector of eigenvalues. LapPE then defines a vertex feature map $x_{\mathrm{PE}}(G) : V \to \mathbb{R}^k$ as follows:

$$x_{\mathrm{PE}}(G)(G, v) = f(\Sigma, \lambda, v), \tag{2.17}$$

where f is some trainable function such as a Transformer (in Kreuzer et al., 2021) or DeepSets (Zaheer et al., 2017) as used by Rampášek et al. (2022). In some formulations, such as Dwivedi et al. (2020), the encodings are only based on the first m eigenvalues and eigenvectors, where m is a hyperparameter. Note that the eigendecomposition of L and the resulting LapPE features are *not* permutation equivariant. The output of a model that receives these features as input is therefore also generally not equivariant.

2.4 Enabling Long-Range Interactions

Standard MPGNNs are local in the sense that the embedding computed for a vertex $v \in V$ by an L-layer MPGNN only depends on the local L-hop subgraph around v. In particular, two vertices $v, u \in V$ with a shortest-path distance greater than L do not influence each other's embedding. This lack of long-range interactions (LRI) has been a key concern of graph learning research in recent years, as MPGNNs may fail to capture the global structure of a given input graph. Concretely, *Under-reaching* (Barceló et al., 2020, Gabrielsson et al., 2023, Sun et al., 2022) refers to the problem of an MPGNN simply being too shallow to propagate information sufficiently far to approximate an LRI-dependent target function. A related issue is *over-squashing* (Alon and Yahav, 2021, Di Giovanni et al., 2023) where a vertex fails to receive information from distant vertices due to structural bottlenecks in the input graph. Both phenomena have been identified as performance bottlenecks on some learning tasks when using standard MPGNN architectures like GCN.

Another important issue in the context of LRI is *over-smoothing* (Li et al., 2018, Oono and Suzuki, 2020, Cai and Wang, 2020), where the latent vertex embeddings computed by an MPGNN converge to the same vector with increasing model depth. Strong over-smoothing can decrease the predictive performance as the smoothed vertex embeddings can become uninformative. This well-studied problem can occur in some popular MPGNNs, such as GCN, and is explained formally by a close connection between convolutional message passing architectures and Laplacian smoothing (Li et al., 2018) which induces a bias towards over-smoothing in the networks. Architectures that suffer from over-smoothing are usually configured to be comparatively shallow to mitigate the problem. This however limits the receptive field of the model, thereby decreasing its ability to capture long-range dependencies in an input graph. Note that not all message passing architectures suffer from over-smoothing to a problematic degree and straightforward solutions, such as residual connections, can mitigate its worst effects (Li et al., 2019, Chen et al., 2020a). We refer to Chapter 3 for an example of a recurrent MPGNN that remains effective for well over 100K message passes. Nonetheless, some of the affected architectures have become popular defaults in graph learning, and understanding their limitations is crucial.

We note that there is no formal consensus on an exact shortest-path distance above which the interaction between two vertices is considered to be "long-range", as this is usually context-dependent. Here, we will refer to a neural architecture as

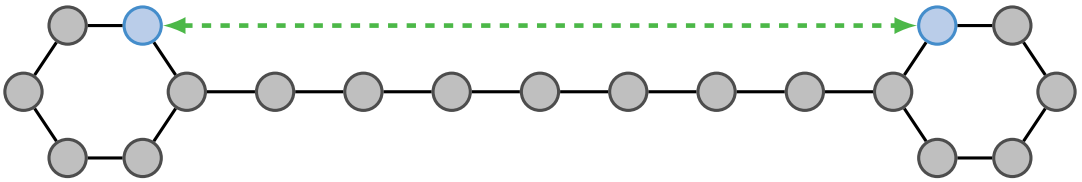


Figure 2.6: Long-range interactions in graphs: With a standard MPGNN two vertices only affect each other's embeddings if their shortest-path distance does not exceed the number of message passes and the model is able to propagate information effectively without running into over-smoothing or over-squashing.

LRI-enabling if it can express a function with an unlimited receptive field, that is, the embedding computed for each vertex can depend on the whole graph, regardless of model depth or graph size. In the following, we discuss approaches for enabling LRI in GNNs, such as Virtual Nodes and Graph Transformers, and Rewiring. In Chapter 5, these methods are compared experimentally and theoretically in more detail.

2.4.1 Virtual Nodes

An early approach to facilitate LRI in MPGNNs are *Virtual Nodes* (VN) as suggested by Gilmer et al. (2017). Intuitively, a Virtual Node is a special vertex that is added to the input graph and connected to all other vertices. This additional vertex facilitates global information flow as it reduces the shortest path distance between any two vertices to at most two. In most implementations, the Virtual Node uses a separate set of message passing functions to distinguish it from the true vertices of the input graph. This simple extension of MPGNNs has been observed to improve the performance across a range of real-world graph learning tasks (see Gilmer et al. (2017), Hu et al. (2020c)). A key feature of VNs is their efficiency, as the additional runtime is in $\mathcal{O}(|V|)$ and therefore linear in the graph size. This is a key strength when compared to approaches based on dense self-attention, namely Graph Transformers, which usually incur a quadratic cost.

Formally, a Virtual Node applies the following transformation to the output embedding $h^{(\ell)}$ of the ℓ-th layer of an MPGNN:

$$\tilde{h}^{(\ell)}(v) = h^{(\ell)}(v) + \mathrm{VN}^{(\ell)}\left(\mathrm{Pool}\left(\{\!\{h^{(\ell)} \mid v \in V\}\!\}\right)\right). \tag{2.18}$$

Here, VN is a trainable MLP and Pool is an aggregation function that is either an element-wise sum-, mean- or max-aggregation. $\tilde{h}^{(\ell)}$ is an updated node embedding that is used as the input to the next layer $\ell + 1$. We further define a MPGNN with Virtual Node (MPGNN+VN) as an MPGNN where every message passing layer is extended by a Virtual Node as defined in Equation 2.18. We illustrate MPGNNs+VN in Figure 2.7b. The virtual node effectively performs an intermediate readout after each round of message passing and passes the aggregated global information back to each vertex through a simple addition. VNs can therefore also

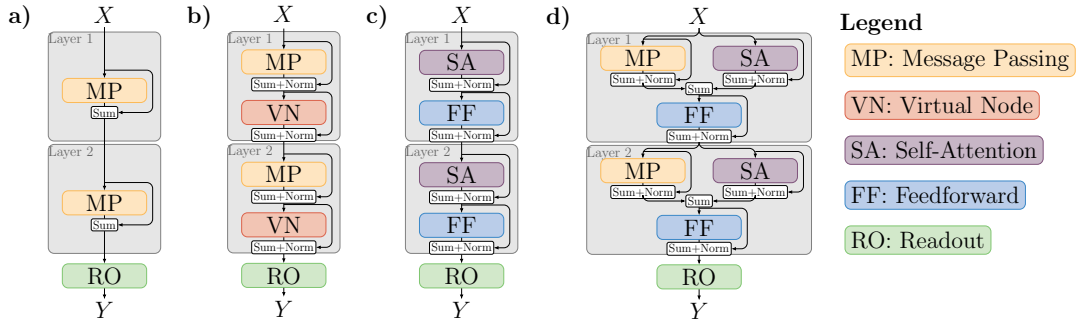


Figure 2.7: **a)** A stack of MPGNN layers with skip connections **b)** A GNN with a Virtual Node **c)** The Transformer architecture **d)** The GPS architecture as proposed by Rampášek et al. (2022)

be viewed as a form of the "intermediate global readout" suggested by Barceló et al. (2020).

We note that Gilmer et al. (2017) also study models with *Virtual Edges* for improving long-range interactions. These models perform message passing on a fully connected graph that models the true and false edges of the input graph through separate edge types. The used aggregation function is a simple sum rather than dot-product attention and the model is therefore not a Transformer. However, the approach is still closely related to Graph Transformers which also conduct fully connected message passing to enable long-range information flow. In their experiments, Gilmer et al. (2017) do not observe Virtual Edges to be superior to the more efficient virtual nodes.

2.4.2 Graph Transformers

The Transformer architecture (Vaswani et al., 2017) has recently received much attention as it enabled major advances in deep learning. While initially proposed for processing sequences in the context of natural language processing, it has since been successfully used across a range of domains such as images (Dosovitskiy et al., 2021), videos (Arnab et al., 2021) and audio data (Baevski et al., 2020). The Transformer architecture is fundamentally linked to graph learning and MPGNNs as it effectively conducts attentive message passing on a fully connected graph to iteratively refine a set of input embeddings. When applied to sequential data this simple process offers key advantages over previous architectures, namely recurrent neural networks, as it is less prone to information bottlenecks and more parallelizable. Its main downside is the quadratic computational cost.

Graph Transformers (GTs) aim to utilize Transformers for graph-structured inputs and have quickly emerged as an architectural paradigm in recent years. Here, we will provide a summary of representative GT architectures. We refer to Müller et al. (2024) for a comprehensive overview of this subject. Note that GTs should not be confused with attentive MPGNN architectures such as GAT (Veličković et al., 2017), which use attention-based aggregations to pass messages along the true

edges of a given (sparse) graph. In contrast, GTs incorporate dense self-attention as an architectural component to enable a fully-connected information exchange between all vertices, thereby enabling long-range interactions.

Let us formally define Transformers before discussing architectural adaptations for graph-structured data. An *attention head* is a trainable function parameterized by three matrices $W_Q, W_K, W_V \in \mathbb{R}^{d \times d_h}$, where d is the input dimension and d_h is the head dimension. For an input matrix $X \in \mathbb{R}^{n \times d}$ (e.g. the matrix of token or node embeddings) the function computes a matrix $H \in \mathbb{R}^{n \times d_h}$ as follows:

$$\mathrm{H}(X) \coloneqq \mathrm{softmax}\left(\frac{(XW_Q)(XW_K)^T}{\sqrt{d_h}}\right) XW_V. \tag{2.19}$$

Here, the softmax function is applied row-wise. A *multi-head self-attention module* SA consists of k attention heads and a weight matrix $W_O \in \mathbb{R}^{kd_h \times d}$ and computes the function

$$\mathrm{SA}(X) \coloneqq [H_1(X), \ldots, H_k(X)]W_O, \tag{2.20}$$

where $[\cdot]$ denotes row-wise concatenation and $H_i(X)$ is the output of the i-th attention head for the input X. A *Transformer Encoder* layer consists of a self-attention module SA and an multi-layer perceptron FF and applies the following transformation to in input matrix $X \in \mathbb{R}^{n \times d}$:

$$Y \coloneqq \mathrm{norm}_1\big(X + \mathrm{SA}(X)\big), \tag{2.21}$$

$$Z \coloneqq \mathrm{norm}_2\big(Y + \mathrm{FF}(Y)\big). \tag{2.22}$$

Here, $Z \in \mathbb{R}^{n \times d}$ is the output matrix of the layer (of dimension d) and $\mathrm{norm}_1, \mathrm{norm}_2$ are optional normalization layers such as LayerNorm (Ba et al., 2016) or BatchNorm (Ioffe and Szegedy, 2015) that are applied after skip connections. A Transformer model stacks multiple such layers sequentially to iteratively refine a set of input embeddings. Note that we focus on fully-connected self-attention without a causal mask, which is common in the context of Graph Transformers.

Dwivedi and Bresson (2021) conducted an early study on using Transformers for graphs. They evaluate an approach that combines the node features of the input graph with laplacian positional encodings (LapPE). The resulting set of node embeddings is then processed with a Transformer encoder model followed by a readout function. This model relies entirely on positional encodings to inject information about the graph structure. They show that the predictive performance of this model is worse than that of sparse MPGNNs on a variety of benchmark tasks. Nonetheless, the work demonstrates that pure Transformers can learn functions on graph-structured data.

Kreuzer et al. (2021) propose SAN (Spectral Attention Networks) that implements a similar idea built on an extension of the LapPE features. Their approach first processes the spectral decomposition of the graph Laplacian with a separate Transformer model to extract positional encodings in a trainable manner. They show that this variation of LapPE significantly improves the predictive results, which are on par with MPGNN baselines on a range of benchmark datasets.

Rampášek et al. (2022) propose GPS (General, Powerful, Scalable Graph Transformer), which extends the architecture of a Transformer encoder layer with a message passing module that is applied in parallel to the self-attention mechanism. The specific MPGNN architecture used for this extension is left as a hyperparameter. GPS therefore constitutes a general framework for hybrid models that combine self-attention and MPGNNs. Let us formally introduce the architecture of GPS, as we will examine its theoretical properties in Section 5.2.3. Formally, a GPS layer $\mathrm{GPS} = (\mathrm{SA}, \mathrm{MP}, \mathrm{FF})$ consists of a self-attention module SA, a message passing layer MP and an MLP FF, all with a shared input and output dimension $d \in \mathbb{N}$. The layer then computes the following function for a given matrix $X \in \mathbb{R}^{|V| \times d}$ of node embeddings:

$$Y \coloneqq \mathrm{norm}_1\big(X + \mathrm{SA}(X)\big) + \mathrm{norm}_2\big(X + \mathrm{MP}(X)\big) \tag{2.23}$$

$$Z \coloneqq \mathrm{norm}_3\big(Y + \mathrm{FF}(Y)\big) \tag{2.24}$$

A GPS model $N = (\mathrm{GPS}^{(1)}, \ldots, \mathrm{GPS}^{(L)}, R)$ combines a sequence of GPS layers $\mathrm{GPS}^{(\ell)}$ with a readout function R. We illustrate a GPS network in Figure 2.7d. The GPS architecture can optionally be configured without a message passing module. In this case, it is identical to the pure Transformer architecture studied by Dwivedi and Bresson (2021). Shirzad et al. (2023) further propose Exphormer as a variation of GPS that uses a specialized sparse attention mechanism. An alternative hybrid Graph Transformer model called LGI-GT has been suggested by Yin and Zhong (2023), who stack message passing and self-attention modules sequentially as opposed to the concurrent layout used by a GPS layer.

Other variations of Graph Transformers inject structural information by modifying the self-attention mechanism. Graphormer (Ying et al., 2021) proposes a variation of Transformers that encodes structural relationships between vertices through an additive attention bias in the self-attention module. This additive bias is computed for each vertex pair (v, u) from the features along the shortest walk that connects v and u and therefore encodes distance information. The Graphormer architecture yields strong results on some molecular datasets and is the winner of the graph-level task of the first large-scale challenge for machine learning on graphs (OGB-LSC) (Hu et al., 2021). Graph Inductive Biases Transformer (GRIT) (Ma et al., 2023) uses an adapted self-attention mechanism that jointly updates embeddings of vertices and vertex pairs. This architecture relies on a variation of RWSE features (Dwivedi et al., 2021, see Section 2.3.1) to inject pair-wise structural information about vertices.

Another class of architectures samples subgraphs from a given input graph, embeds each subgraph separately with an MPGNN, and then processes the resulting set of subgraph embeddings with a Transformer. Examples of this general idea are Structure-Aware Transformers (SAT) (Chen et al., 2022) and Graph ViT (He et al., 2023). Conceptually, these approaches are related to Vision Transformers (ViT) (Dosovitskiy et al., 2021), which first divide an image into patches and then apply a Transformer to extracted patch embeddings. In their work He et al. (2023) also propose "Graph MLP-Mixer", which analogously adapts MLP-Mixers (Tolstikhin et al., 2021) from image data to graphs.

Kim et al. (2022) propose TokenGT. This approach "tokenizes" an input graph by viewing each vertex and edge as a token. These tokens are associated with embeddings that combine positional encodings with type and feature information. These embeddings are then processed by a standard Transformer model.

2.4.3 Rewiring

Rewiring techniques aim to improve information flow between vertices by altering the edge relation of the input graph. Essentially, these approaches can be viewed as an intermediate step between vanilla MPGNNs and Graph Transformers: While Graph Transformers are typically based on a fully connected information exchange between vertices, rewiring methods add *some* edges to an input graph while maintaining sparsity and efficiency.

Spatial rewiring techniques generally connect vertices to their k-hop neighborhood (Gabrielsson et al., 2023, Abboud et al., 2022). *Spectral* approaches aim to modify the graph structure such that some theoretical graph property is optimized, such as the spectral gap (Arnaiz-Rodríguez et al., 2022, Karhadkar et al., 2023). DReW (Gutteridge et al., 2023) is a variation of rewiring where the edge relation changes towards denser connectivity with each message passing layer.

Learning Global Search Heuristics for Constraint Satisfaction Problems

In this chapter, we present work on learning heuristics for constraint satisfaction problems (CSPs) with GNNs. The content is based on Tönshoff et al. (2023a) and falls within the area of *neural combinatorial optimization*, which aims to utilize deep learning in the context of combinatorial problems. This has been an active field of research in recent years and approaches have been proposed for a wide range of well-known combinatorial problems, such as the travelling salesman problem (TSP) (Vinyals et al., 2015), maximum cut (MaxCut) (Khalil et al., 2017), Boolean satisfiability (Sat) (Selsam et al., 2018), mixed-integer programming (MIP) (Gasse et al., 2019), and constraint satisfaction problems (CSPs) (Tönshoff et al., 2021). We refer to Cappart et al. (2021) for a comprehensive overview. Most of these approaches are based on GNNs, since combinatorial problems typically have natural graph representations that capture the discrete problem structure. Some works construct hybrid approaches that enhance classical algorithms with learnable components. Here, we focus on the alternative end-to-end setting where the aim is to learn stand-alone GNNs that directly map input instances to solutions. Learning full algorithms in such a manner is a highly challenging problem but also offers some unique advantages: Firstly, it enables us to learn novel algorithms from scratch which can differ fundamentally from classical approaches, thereby expanding the repertoire of available solvers. Secondly, the hard nature of this learning task enables us to effectively probe what GNNs are capable of, since solving combinatorial problems generally requires the extraction of fine-grained structural information and many iterations of search. This raises questions on how to design GNNs with sufficient expressivity and scalability to learn effective heuristics. Exploring solutions to these challenges is of general interest to the wider field of graph learning and aligns with the main research question of this thesis.

We also note that the learned algorithms are inherently GPU-compatible since they are fully implemented as GNNs. We can therefore utilize modern hardware accelerators for solving combinatorial problems. However, this should not be over-stressed as an advantage. GNNs are very costly algorithms when compared to classical search heuristics, which typically are orders of magnitude faster in terms of "search steps per time" while running on a CPU. Obtaining competitive heuristics despite this deficit is a key challenge when designing end-to-end approaches based on GNNs.

This chapter is organized as follows: We first provide relevant background on CSPs and reinforcement learning in Section 3.1 and Section 3.2, respectively. After discussing related approaches for neural combinatorial optimization in Section 3.3 we introduce ANYCSP as a high-performance approach to learning CSP heuristics with GNNs in Section 3.4. Section 3.5 provides an extensive experimental evaluation of the proposed method.

3.1 Constraint Satisfaction Problems

Constraint satisfaction problems (CSPs) are a ubiquitous framework for specifying combinatorial search and optimization problems. They include many well-known NP-hard problems such as Boolean satisfiability (Sat), graph coloring (Col), and maximum cut (MaxCut) and can flexibly be adapted to specific application-dependent problems. CSP solution strategies range from general solvers based on methods such as constraint propagation or local search (see Russell et al. (2020), Chapter 6) to specialized solvers for individual problems like Sat (see Biere et al. (2021)).

Formally, a *CSP instance* is a triple $\mathcal{I} = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ consisting of the following components:

(1) $\mathcal{X}$ is a finite set of variables.

(2) $\mathcal{D}$ assigns a *domain* $\mathcal{D}(X)$ to each variable $X \in \mathcal{X}$, which is the finite set of values that X can be assigned to.

(3) $\mathcal{C}$ is a set of constraints $C = \big(s^C, R^C\big)$, where for some $k \geq 1$, the *scope* $s^C = (X_1, \ldots, X_k) \in \mathcal{X}^k$ is a tuple of variables and $R^C \subseteq \mathcal{D}(X_1) \times \ldots \times \mathcal{D}(X_k)$ is a k-ary relation over the corresponding domains.

We always assume that the variables in the scope s^C of a constraint C are mutually distinct; we can easily transform an instance not satisfying this condition into one that does by adapting the relation R^C accordingly.

An *assignment* for a CSP instance $\mathcal{I} = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ is a mapping α that assigns a domain element $\alpha(X) \in \mathcal{D}(X)$ to each variable X. An assignment α *satisfies* a constraint $C = ((X_1, \ldots, X_k), R)$ (we write $\alpha \models C$) if $(\alpha(X_1), \ldots, \alpha(X_k)) \in R$. Furthermore, α *satisfies* $\mathcal{I}$, or is a *solution* to $\mathcal{I}$, if it satisfies all constraints in $\mathcal{C}$. The objective of a CSP is to decide if a given instance has a satisfying assignment and to find one if it does. To distinguish this problem from the maximization version introduced below, we sometimes speak of the *decision version*.

We define the *quality* $Q_{\mathcal{I}}(\alpha)$ of an assignment α to be the fraction of constraints in $\mathcal{C}$ satisfied by α: $Q_{\mathcal{I}}(\alpha) = |\{C | C \in \mathcal{C}, \alpha \models C\}|/|\mathcal{C}|$. An assignment α is *optimal* if it maximizes $Q_{\mathcal{I}}(\alpha)$ for the instance $\mathcal{I}$. The goal of the maximization problem MaxCSP is to find an optimal assignment for a given instance.

Many well-known combinatorial problems can naturally be modeled within the general CSP-formalism. A straightforward example are Boolean formulas in conjunctive normal form where each clause can be viewed as a constraint over binary variables.

Example 3.1. *Consider the Boolean CNF-formula* $f = (X_1 \vee \neg X_2) \wedge X_3$*. Then an equivalent CSP instance* $\mathcal{I} = (\mathcal{X}, \mathcal{D}, \{C_1, C_2\})$ *can be constructed as follows:*

$$\begin{aligned}
&\mathcal{X} = \{X_1, X_2, X_3\},\\
&\mathcal{D}(X_i) = \{0, 1\},\\
&C_1 = \big((X_1, X_2), \{(0,0),(1,0),(1,1)\}\big),\\
&C_2 = \big((X_3), \{(1)\}\big)
\end{aligned}$$

Graph coloring is another well-studied problem with a simple translation to CSPs. In this case, the vertices of a graph can be viewed as variables and every edge induces a "not equal"-constraint.

Example 3.2. *Consider the problem of finding a vertex coloring with* $k \in \mathbb{N}$ *colors for a given graph* $G = (V, E)$*. Then an equivalent CSP instance* $\mathcal{I} = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ *can be constructed as follows:*

$$\begin{aligned}
&\mathcal{X} = V,\\
&\mathcal{D}(v) = \{1, \dots, k\},\\
&\mathcal{C} = \{C_{uv} | uv \in E\},\\
&C_{uv} = \big((u, v), \{(i, j) \mid i, j \in [k], i \neq j\}\big)
\end{aligned}$$

Let us introduce some further notation that will be convenient in Section 3.4. Slightly abusing terminology, we call a pair (X, d) where $X \in \mathcal{X}$ and $d \in \mathcal{D}(X)$ a *value* for variable X. For all $X \in \mathcal{X}$ we let $\mathcal{V}_X = \{X\} \times \mathcal{D}(X)$ be the set of all values for X, and we let $\mathcal{V} = \bigcup_{X \in \mathcal{X}} \mathcal{V}_X$ be the set of all values. We usually denote values by v. Working with these values instead of domain elements is convenient because the sets $\mathcal{V}_X$, for $X \in \mathcal{X}$, are mutually disjoint, whereas the domains $\mathcal{D}(X)$ are not necessarily. We may then view an assignment α as a subset $\alpha \subseteq \mathcal{V}$ that contains exactly one value from each $\mathcal{V}_X$. Depending on the context, we use either view, and we synonymously write $\alpha(X) = d$ or $(X, d) \in \alpha$.

Finally, we define a *soft assignment* for a CSP instance $\mathcal{I}$ is a mapping $\varphi : \mathcal{V} \to [0, 1]$ such that $\sum_{v \in \mathcal{V}_X} \varphi(v) = 1$ for all $X \in \mathcal{X}$. We interpret the numbers $\varphi(v)$ as probabilities and say that an assignment α is *sampled* from a soft assignment φ (we write $\alpha \sim \varphi$) if for each variable $X \in \mathcal{X}$ we independently draw a value $v \in \mathcal{V}_X$ with probability $\varphi(v)$.

3.2 Reinforcement Learning

Reinforcement Learning (RL) is a branch of machine learning concerned with making sequences of decisions in order to maximize a cumulative reward. It is particularly suitable for problems where explicit guidance is not available and where an agent must learn to interact with an environment through trial and error. Well-known applications include robot control (Peters et al., 2003), playing Go (Silver et al., 2016) and tuning language models through human feedback (Ouyang et al., 2022). Here, we will briefly summarize the background on RL that is most relevant to neural combinatorial optimization, namely Q-learning and policy gradient methods. We refer to Sutton and Barto (2018) for a comprehensive overview on the vast subject of RL.

Markov Decision Process (MDP) The environment that a reinforcement learning agent interacts with is usually formalized as a *Markov Decision Process* (MDP). Concretely, an MDP $\mathcal{M} = (S, A, P, R)$ is a 4-tuple consisting of the following components:

(1) **State Space** S: A finite set of states representing the possible configurations a system can be in.

(2) **Action Space** A: A finite set of actions A available to the agent in each step.

(3) **Transition Probability Function** P: A function that defines the probability $P(s'|s, a)$ of transitioning from state $s \in S$ to another state $s' \in S$ given an action $a \in A$.

(4) **Reward Function** R: A function that defines the immediate reward $R(s, a, s') \in \mathbb{R}$ received by the agent for transitioning from one state $s \in S$ to $s' \in S$ given an action $a \in A$.

Policies A *policy* $\pi : S \times A \to [0, 1]$ is a mapping that assigns to each state $s \in S$ a probability distribution over actions. It defines the strategy that an agent uses to stochastically select actions in each state. Policies may be deterministic and in this case, we can alternatively view them as functions $\pi : S \to A$ that directly map states to actions. A *trajectory* $\tau = s_0, a_0, s_1, a_1, s_2 \dots$ is a sequence of states and actions. Given a policy π and an initial state $s_0 \in S$ we can obtain a trajectory by iteratively sampling the next action $a_t \sim \pi(s_t)$ from the policy and then draw the next state $s_{t+1} \sim P(\cdot \mid s_t, a_t)$ from the underlying MDP.

The primary objective of reinforcement learning is to find a policy π^* that maximizes the expected cumulative reward over sampled trajectories:

$$\pi^* = \arg\max_{\pi} \mathbf{E}_{\pi}\Big[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t, s_{t+1})\Big] \tag{3.1}$$

Here, the *discount factor* $\gamma \in [0, 1]$ is a hyperparameter representing the degree of importance of future rewards compared to immediate rewards. Note that we also assume some distribution of initial states from which s_0 is sampled. The arg max in Equation 3.1 is usually defined over some chosen set of candidate policies, such as all policies parameterized by a given neural network architecture.

Q-Learning Q-Learning (Watkins, 1989) is a popular class of RL algorithms used to approximate the optimal action-selection policy for a given finite MDP. The core idea is to learn an action-value function $\hat{Q}_\theta : S \times A \to \mathbb{R}$, which estimates the expected cumulative reward of taking action a in state s:

$$\hat{Q}_\theta(s_t, a) \stackrel{!}{=} R(s_t, a, s_{t+1}) + \gamma \max_{a' \in A} \hat{Q}_\theta(s_{t+1}, a'). \tag{3.2}$$

The function $\hat{Q}_\theta$ with parameter vector θ can be any trainable function such as a simple linear projection or, in the case of deep Q-learning (DQN), a deep neural network. A key advantage of Q-learning is its simplicity as learning $\hat{Q}_\theta$ based on observed trajectories is a standard regression problem. A common choice for training $\hat{Q}_\theta(s, a)$ is to minimize the mean squared error for the objective in Equation 3.2. This learned Q-function defines a deterministic policy that chooses the next action in a greedy manner based on the predictions:

$$\pi(s_t) = \arg\max_{a \in A} \hat{Q}_\theta(s_t, a). \tag{3.3}$$

As a drawback, Q-learning tends to scale poorly in the size of the action space, since computing the policy in Equation 3.3 requires evaluating $\hat{Q}_\theta(s_t, a)$ over all possible actions a. Another key problem is a lack of exploration when following the simple deterministic policy in Equation 3.3.

Throughout training, the distribution of sampled trajectories changes with the underlying policy. This can lead to the model forgetting information it learned at the beginning of training. A common solution to this issue is *experience replay* (Lin, 1992), which retains old trajectories in a replay buffer. In each training step, the model is trained on a batch that mixes new trajectories with older data retained in the buffer. This aids in retaining knowledge throughout training and has been shown to significantly improve sample efficiency and speed of convergence when training, especially when using deep neural networks to learn $\hat{Q}_\theta$.

Policy Gradient Methods Policy gradient methods aim to directly learn the policy without the need for an action-value function. The policy π_θ is a trainable function that is parameterized by a vector $\theta \in \mathbb{R}^d$ of learnable weights. These parameters are then optimized with gradient descent to maximize the expected cumulative reward. Concretely, the primary optimization objective is defined as

the true value function of the policy when parameterized by θ:

$$J(\theta) = \mathbf{E}_{\pi_\theta}\Big[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t, s_{t+1})\Big] \tag{3.4}$$

One of the simplest policy gradient methods is REINFORCE (Williams, 1992). Given a finite trajectory $s^0 a^0 s^1 \dots s^T$ obtained with the policy π_θ, Williams (1992) show that the following equation yields an unbiased estimate of the policy gradient $\nabla_\theta J(\theta)$:

$$\nabla_\theta J(\theta) \mathrel{\hat{=}} \nabla_\theta \sum_{t=0}^{T} G_t \log \mathbf{P}(a^t | \pi_\theta(s^t)) \tag{3.5}$$

Here, the return G_t is defined as the cumulative reward observed in the trajectory after step t:

$$G_t = \sum_{t'=t}^{T} \gamma^{t-t'} R(s_t, a_t, s_{t+1}) \tag{3.6}$$

Note that Equation 3.5 is not a strict equality as the right-hand side is an empirical estimate of $\nabla_\theta J(\theta)$. This gradient estimate can be used to train π_θ with gradient ascent to maximize $J(\theta)$. After each training step, a new batch of trajectories is sampled with the updated policy parameters before this process is repeated. A major advantage of this method is its ability to handle large action spaces since the policy is a neural network that, given the current state s_t, directly infers a distribution from which the next action a_t can be sampled. Crucially, the gradient estimate in Equation 3.5 only depends on the T actions in the given trajectory. An underlying assumption is that the gradient $\nabla_\theta \mathbf{P}(a^{(t)} | \pi_\theta(s^{(t)}))$ of the policy parameters with respect to the action probabilities can be computed efficiently, which is usually the case for neural networks.

As a downside, the gradient estimate of Equation 3.5 is prone to high variance which can lead to slow convergence of the training. More advanced policy-gradient approaches, such as actor-critic methods (Degris et al., 2012), adapt the gradient estimate to reduce its variance at the expense of a more complicated architecture and training setup.

3.3 Related Work

In the following, we will discuss related work on neural combinatorial optimization. We focus on end-to-end approaches for problems that can be modeled as CSPs, since these are most relevant for the motivation and evaluation of ANYCSP.

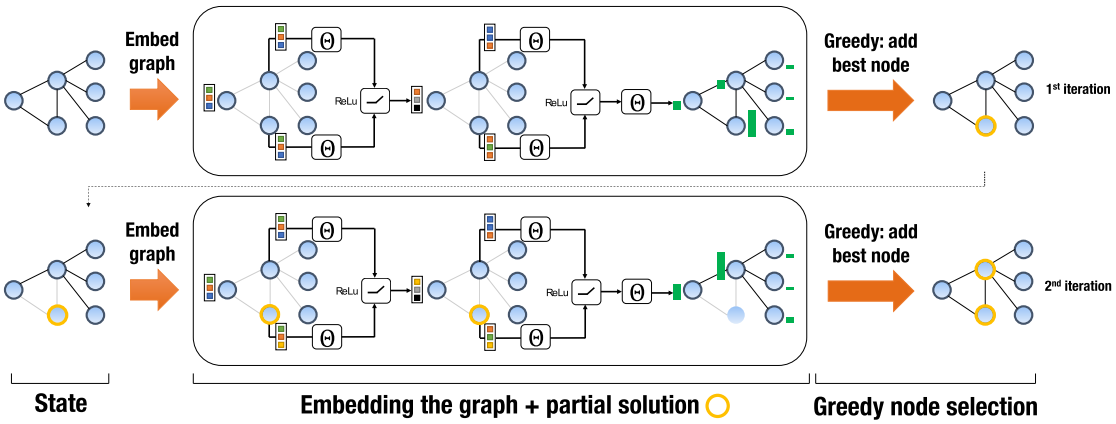


Figure 3.1: An illustration of S2V-DQN. A GNN is applied to the input graph $G = (V, E)$ to rate all possible expansions of the current partial solution $S \subseteq V$. The estimated best vertex is greedily added to the partial solution and the process is repeated. (Image Credit: Khalil et al. (2017))

3.3.1 S2V-DQN

Khalil et al. (2017) propose S2V-DQN, an approach that uses GNNs to learn greedy heuristics for optimization problems over graphs. More specifically, the method is aimed at problems that can be modeled as a task of selecting a subset (or sequence) of vertices $S \subseteq V$ for a given graph $G = (V, E)$, such as minimum vertex cover TSP, MaxCut and minimum vertex cover (MVC). In S2V-DQN, a solution set S is iteratively constructed by greedily adding vertices according to a learned selection strategy based on a GNN. This setup is well suited for Q-learning, which is used to learn problem-specific heuristics from data without knowledge of optimal solutions.

Concretely, S2V-DQN trains an MPGNN $\hat{Q}_\theta$ which takes as input the given problem instance $G = (V, E)$, a partial solution $S \subseteq V$ and a vertex $v \in V$ to compute the estimated return $\hat{Q}_\theta(G, S, v) \in \mathbb{R}$ for adding v to S. The GNN model is a feedforward MPGNN based on the structure2vec architecture (Dai et al., 2016) and the partial solution $S \subseteq V$ is encoded in the input as a vertex labeling for G. Starting from an empty set $S^{(0)} = \varnothing$ a solution can be constructed in a greedy manner:

$$v^{(t+1)} = \underset{v \in V \setminus S^{(t)}}{\arg\max}\ \hat{Q}_\theta(G, S^{(t)}, v) \tag{3.7}$$

$$S^{(t+1)} = S^{(t)} \cup \{v^{(t+1)}\} \tag{3.8}$$

A problem-specific stopping criterion and reward determines when the process terminates and what the optimization objective is. For example, when training a heuristic for MVC, the construction stops once all edges in G are covered by a vertex in S and the reward function is chosen as $R(S) = -|S|$ to incentivize smaller solutions. The training of S2V-DQN is based on n-step Q-learning with an experience replay buffer (Lin, 1992, Riedmiller, 2005).

Khalil et al. (2017) experimentally evaluate S2V-DQN by learning heuristics for three combinatorial problems, namely MVC, MaxCut and TSP. The approach scales well from training instances with 50-100 vertices to larger instances with over 1000 vertices. Note that the greedy construction used by S2V-DQN will always terminate in at most $|V|$ steps, ensuring fast termination of the algorithm. However, the monotonic nature of this process also prohibits the heuristics from undoing suboptimal decisions once they are made which limits the solution quality.

3.3.2 ECO-DQN

Barrett et al. (2020) propose ECO-DQN (**E**xploratory **C**ombinatorial **O**ptimization **DQN**) which learns local search heuristics for combinatorial optimization problems on graphs. Rather than greedily constructing a single solution for a given instance, like S2V-DQN, this approach iteratively refines a candidate solution for any number of search steps. In principle, a trained ECO-DQN model should therefore be able to explore a wider range of solutions and yield better results.

In each search step a single vertex is chosen and either added or removed from the candidate solution, which is initially chosen uniformly at random. Formally, this procedure corresponds to a local search heuristic where a single variable (or vertex) is allowed to change its state in each iteration. The action space of the underlying MDP is simply the set of vertices, one of which has to be selected in each step. ECO-DQN trains an MPGNN as an action-value function for this MDP using standard Q-learning techniques. Barrett et al. (2020) design a reward function that aims to encourage exploration. In particular, whenever the new solution improves over the previous best it is rewarded with the margin of improvement.

Barrett et al. (2020) provide an experimental evaluation by learning heuristics for the MaxCut problem. They demonstrate that ECO-DQN outperforms S2V-DQN by a significant margin, showing that an improved ability to explore the solution space does yield stronger results in practice.

ECORD (**E**xploratory **C**ombinatorial **O**ptimization with **R**ecurrent **D**ecoding) is further suggested by Barrett et al. (2022) as an improved approach that aims to lower the computational cost of evaluating the action-value function in each step, which is a key performance bottleneck of ECO-DQN. ECORD deploys a special neural architecture that initially applies an MPGNN once to extract structural embeddings for each vertex. During the search itself, these embeddings are then processed by a comparatively lightweight RNN-based model to estimate the action-value function. This change is shown experimentally shown to yield better solutions for the MaxCut problem when compared to ECO-DQN.

Overall, the ability to undo suboptimal decisions and continually search through the solution space is a key improvement. However, the local search paradigm used by both ECO-DQN and ECORD remains limiting as it only allows a single vertex to change its state in each step. With this restriction, it is difficult for neural

approaches to compete with classical local search heuristics at scale, as these tend to require significantly less compute per search step.

3.3.3 RLSAT

Yolcu and Póczos (2019) propose RLSAT, an approach for learning local search heuristics for Boolean satisfiability with GNNs. RLSAT is based on a standard local search heuristic for SAT, which searches for a satisfying assignment of a given CNF formula f by starting with a random assignment and then iterative flipping a variable according to some variable selection strategy. In the case of RLSAT, the variable selection strategy is represented by trainable MPGNN. The heuristic can then be tuned towards specific distributions of CNF formulas using reinforcement learning.

The graph representation is the bipartite factor graph of the given CNF formula f that models variables and clauses as vertices and encodes the sign of literals through edge types. RLSAT uses a heterogenous MPGNN architecture to learn a policy network π_θ with parameters θ. In each search iteration, the policy MPGNN π_θ takes the graphs $G_{f,\alpha^{(t-1)}}$ as input, which is the factor graph of f where the variables are labeled with the previous variable assignments $\alpha^{(t-1)}$. The output is a probability distribution $\rho^{(t)} : X_f \to [0, 1]$ that maps every variable $x \in X_f$ in x to a selection probability:

$$\rho^{(t)} = \pi_\theta(G_{f,\alpha^{(t-1)}}) \tag{3.9}$$

Note that the output is computed by first predicting a scalar value for each variable and then applying the softmax function over the result. To obtain a new assignment $\alpha^{(t)}$, one variable $x \sim \rho^{(t)}$ is sampled from the predicted distribution and its value from $\alpha^{(t-1)}$ is flipped. In particular, two consecutive assignments will always differ by exactly one variable. Note that π_θ is not recurrent and can not access information of earlier search steps.

The training uses REINFORCE (Williams, 1992) to train the heuristics with policy gradients. To this end, RLSAT uses a simple reward scheme that yields a reward of 1 when a satisfying assignment is found and 0 otherwise. Note that this reward scheme is highly sparse. Especially towards the beginning of training, it requires the randomly initialized GNN to yield some satisfying assignments. Otherwise, the reward signals are constantly zero and not useful for training. To address this, Yolcu and Póczos (2019) deploy *curriculum learning* where the model is initially trained on very small instances which are scheduled to gradually increase in size and difficulty throughout training. With this training procedure, RLSAT can learn heuristics despite the sparse reward function.

Yolcu and Póczos (2019) evaluate RLSAT by learning local search heuristics for various distributions of CNF formulas that arise from reducing different combinatorial problems to SAT. The primary result is that the learned heuristics are highly problem-specific and adapt to the class of CNF instances that are observed during training. The learned heuristics are generally competitive with the classical local

search heuristic WalkSAT (Selman et al., 1993) when both methods are allowed to perform an equal number of search steps. These results do demonstrate that GNN-based Sat heuristics can be tuned towards specific types of inputs in a data-driven manner.

As a caveat, the time needed for each step is orders of magnitude larger when compared to WalkSAT as each variable selection requires a forward pass through an MPGNN. A comparison based on a timeout would therefore still heavily favor classical local search algorithms. In their discussion, Yolcu and Póczos (2019) argue that more efficient architectures and advances in hardware accelerators may address this problem. While efficiency improvements are generally useful, we argue that the main drawback is conceptual: The policy GNN is a comparatively expensive algorithm that extracts rich structural information for all vertices in each search step. This wealth of information is used to change a single variable and is then discarded. The expense of the policy MPGNN is therefore disproportionate to the effect that its actions have in each search step. The sparse reward function requiring curriculum learning is a further limitation that adds complexity to the training procedure.

3.3.4 RUNCSP

In Tönshoff et al. (2021) we propose RUNCSP (**R**ecurrent **U**nsupervised **N**eural Network for **CSP**s), an approach for learning GNN-based heuristics for *binary* CSPs. It uses a recurrent GNN architecture to generate a sequence of assignments for a given CSP instance. Notably, it performs global search in the sense that all variables of the instance can change in each search step. It is not trained with reinforcement learning but with a simple loss based on a relaxed version of the discrete optimization objective.

RUNCSP can be applied to binary CSPs, which are CSPs where all constraints have arity two. The training of RUNCSP further depends on a fixed *constraint language* that specifies the domain and set of relations that can occur in the input instances. Formally, a *constraint language* is a finite set Γ of relations over some fixed domain D, and a CSP instance $\mathcal{I} = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ is a Γ-instance if $\mathcal{D}(X) = D$ for all variables $X \in \mathcal{X}$ and $R^C \in \Gamma$ for all constraints $C \in \mathcal{C}$. Max-CSP(Γ) is the problem of finding an assignment that satisfies the maximum number of constraints of a given Γ-instance. A RUNCSP model is always trained for Max-CSP(Γ) of some fixed constrained language Γ.

The restriction to binary CSPs with fixed constraint language significantly simplifies the graph construction and GNN architecture. Binary CSPs can be directly viewed as graphs where the variables are vertices and the constraints are directed edges. Only finitely many relations can occur in the fixed constraint language. The relation of a constraint can therefore simply be modeled as an edge type (or categorical edge feature). Given the fixed domain, assigning a variable to a value reduces to a standard multi-class classification task over the vertices.

A RUNCSP model takes as input a Γ-instance $\mathcal{I} = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ and outputs a sequence of soft assignments $\varphi^{(1)}, \ldots, \varphi^{(T)}$ for $\mathcal{I}$. A hard assignment $\alpha^{(t)} : \mathcal{X} \to D$ can be obtained by sampling a value for each $X \in \mathcal{X}$ from the distribution $\varphi^{(t)}(X)$ or by deterministically choosing the value with the highest probability. The GNN architecture used by RUNCSP is a recurrent MPGNN that uses an LSTM cell (Hochreiter and Schmidhuber, 1997) as an update function. The model incorporates the constraint relations by applying a relation-specific linear transformation to the messages sent along each directed edge, similar to RGCN (Schlichtkrull et al., 2018). A single message pass is performed in each search step before the (recurrent) vertex embeddings are projected to a soft assignment. Note that this construction allows each variable to change its value in each search step t, thereby allowing RUNCSP to conduct *global* search. This enables the model to parallelize changes to the assignment across the whole instance, speeding up the search.

RUNCSP is trained by directly optimizing a differentiable loss function that measures the quality of soft assignments. For a given instance $\mathcal{I} = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ and soft assignment φ we can obtain a hard assignment $\alpha : X \to D$ by independently sampling a value for each $x \in X$ from the distribution specified by $\varphi(x)$. The probability that any given constraint $(x, y, R) \in C$ is satisfied by α can then be expressed by

$$\Pr_{\alpha \sim \varphi} \left[(\alpha(x), \alpha(y)) \in R \right] = \varphi(x)^T A_R \, \varphi(y) \tag{3.10}$$

where $A_R \in \{0,1\}^{d \times d}$ is the characteristic matrix of the relation R with $(A_R)_{i,j} = 1 \iff (i,j) \in R$. The main objective of RUNCSP is to maximize this probability over all constraints which is achieved by minimizing the corresponding log-likelihood:

$$\mathcal{L}_{\text{CSP}}(\varphi, I) \coloneqq \frac{1}{|C|} \cdot \sum_{(x,y,R) \in \mathcal{C}} - \log\left(\varphi(x)^T A_R \, \varphi(y)\right). \tag{3.11}$$

This is aggregated over all search iterations t. Note that this differentiable loss can optimized using a standard SGD-based optimizer. The training data are randomly generated Γ-instances and no optimal solutions need to be pre-computed. Training converges relatively quickly, usually within minutes, which allows RUNCSP to learn a problem-specific heuristic for the constraint language Γ with moderate compute resources.

In Tönshoff et al. (2021) RUNCSP is evaluated experimentally on a range of binary CSPs such as Max-k-Sat, MaxCut, and graph coloring. On some CSPs, such as Max-2-Sat, the approach can learn heuristics that are competitive state-of-the-art algorithms. For other problems, such as graph coloring, the learned algorithms outperform simple search heuristics, such as greedy search, but do not quite match the best classical algorithmic baselines.

Nonetheless, RUNCSP demonstrates that we can learn non-trivial search algorithms for a range of CSPs with a unified architecture and training methodology. The main drawback of RUNCSP is its training objective which is a simple loss based directly on the soft assignments. While it enables robust and fast training, it does

not fully capture the hardness of the underlying discrete problem. In Section 3.5 we observe that RUNCSP is uncompetitive when compared to a model trained towards the discrete optimization objective with reinforcement learning. Another limitation is the restriction to binary CSPs. Many interesting CSPs, such as SAT, have constraints with an arity greater than two. While such CSPs can always be mapped to binary CSPs, this reduction creates a non-trivial blowup in the instance size. Each RUNCSP model is also bound to a single constraint language that is fixed before training. This further limits the generalization ability of each model. For example, when applying RUNCSP to the graph coloring problem we must train one model for each number of colors since the domain size is fixed with the constraint language. It would be more convenient if an approach could handle variable domain sizes, which would allow us to train and test a single model on mixed instances with any number of colors. The same argument also applies to the constraint relations, which are also pre-determined by the constraint language when training a RUNCSP model.

3.4 ANYCSP

We propose ANYCSP[1], a GNN-based approach for learning heuristics for general constraint satisfaction. Our main objective is to address the key weaknesses of prior work in one scalable method: Firstly, ANYCSP is generic and can be applied to any combinatorial problem within the CSP formalism. More specifically, one ANYCSP model can take every CSP instance as input and map it to a sequence of assignments, even instances with domain sizes, constraint arities, or relations not seen during training. No prior reduction to more specific CSPs, such as binary CSPs or CNF formulas, is required. This generality is achieved with a custom graph representation for general CSPs which is generic and well suited as an input for recurrent MPGNNs. Secondly, we combine a global search action space with reinforcement learning. Given a CSP instance, a trained ANYCSP model iteratively searches for assignments of increasing quality. Crucially, the search is *global*; it allows transitions between any two assignments in a single step. We thereby avoid a key bottleneck of previous neural approaches based on local search, which only change the values of a single or a few variables in each step. Some earlier works, such as RUNCSP, are also based on global search but train with loss functions based on soft assignments which do not fully capture the hardness of the underlying discrete problem. Here, we use RL to train ANYCSP towards the discrete optimization task defined by CSPs with a tailored reward scheme that encourages exploration and prevents the search from getting stuck in local maxima. We utilize policy gradient methods for training to efficiently handle the exponential action spaces associated with global search.

To learn a heuristic with ANYCSP for a specific CSP, such as graph coloring or SAT, all that is needed is a set (or generator) of training instances of the CSP. We can then train an ANYCSP model on these instances to obtain a heuristic tuned

[1] **A**re **N**eural Networks great heuristics? **Y**es, for **CSP**s.

CSP Instance $\mathcal{I}$:

$\mathcal{X} = \{X, Y, Z\}$

$\mathcal{D}(X) = \{1, 2, 3\}$
$\mathcal{D}(Y) = \{1, 2\}$
$\mathcal{D}(Z) = \{1, 2\}$

$C_1 : X \leq Y$
$C_2 : Y \neq Z$

Assignment $\alpha = (2, 1, 2)$

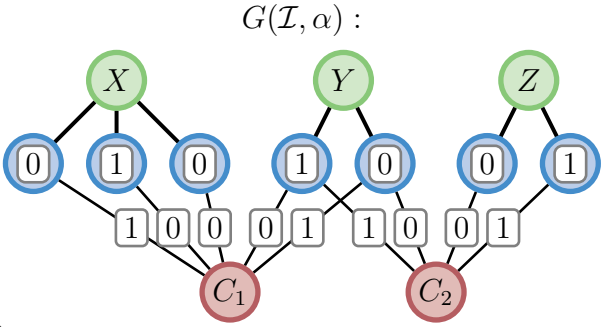


Figure 3.2: Example of the **constraint value graph** $G(\mathcal{I}, \alpha)$ for a given CSP instance $\mathcal{I}$ and an assignment α. The graph contains vertices for the variables, values and constraints of $\mathcal{I}$. Each value is connected to its variable and labeled with the assignment α. Each constraint is connected to the values of its variables. This edge set is labeled such that a label of 1 for edge (C, v) states that choosing value v will satisfy the constraint C if no other variables involved in C change their values.

towards the underlying distribution of CSPs. This procedure is unsupervised and does not depend on the availability of optimal ground-truth solutions, which would be prohibitive in the context of NP-hard problems. Since the model is generic, no architectural changes are necessary before training for a new CSP. We can therefore learn heuristics for any specific CSP in a purely data-driven manner.

In Section 3.5 we evaluate ANYCSP by learning heuristics for a range of important CSPs: Col, Sat, MaxCut and general CSP benchmark instances. We demonstrate that our method achieves a substantial increase in performance over prior GNN approaches and can compete with conventional algorithms. ANYCSP models trained on small random graph coloring problems are on par with state-of-the-art coloring heuristics on structured benchmark instances. On Max-k-Sat, our method scales to test instances 100 times larger than the training data, where it outperforms state-of-the-art search heuristics despite performing 1000 times fewer search iterations. We demonstrate that our global search action space and RL-based training approach are both key factors underlying the strong empirical performance.

In the following, we provide a detailed description of our graph construction, GNN architecture and training procedure.

3.4.1 Graph Construction

We design a novel graph representation for CSPs. The main design objective is a graph construction that can directly model any given CSP instance $\mathcal{I}$. Additionally, the representation should be compact enough to allow instances of significant size to be processed by a GNN.

With every CSP instance $\mathcal{I} = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ we associate a tripartite graph with vertex set $\mathcal{X} \cup \mathcal{V} \cup \mathcal{C}$, where $\mathcal{V}$ is the set of values defined in Section 3.1, and two kinds of edges: *variable edges* (X, v) for all $X \in \mathcal{X}$ and $v \in \mathcal{V}_X$, and *constraint edges* (C, v) for all $C \in \mathcal{C}$ and $v \in \mathcal{V}_X$ for some X in the scope of C. This graph representation is more or less standard; one slightly unusual feature is that we introduce edges from constraints directly to the values and not to the variables. This will be important in the next step, where information about the constraint relations R^C is compactly encoded through a binary labeling of the constraint edges. For each assignment α we introduce a vertex labeling L_V and an edge labeling L_E. The vertex labeling L_V is a binary encoding of α, that is, $L_V(v) = 1$ if $v \in \alpha$ and $L_V(v) = 0$ for each $v \in \mathcal{V} \setminus \alpha$. The edge labeling L_E encodes how changes to α affect each constraint. For every constraint $C \in \mathcal{C}$ and value $(X_i, d) \in \mathcal{V}_{X_i}$ of variable X_i in the scope $(X_1, \ldots, X_k)$ of C we define the edge label to be $L_E(C, v) = 1$ if

$$(\alpha(X_1), \ldots, \alpha(X_{i-1}), d, \alpha(X_{i+1}), \ldots, \alpha(X_k)) \in R^C.$$

and $L_E(C, v) = 0$ otherwise. Intuitively, the edge labels encode for each constraint edge (C, v) whether or not choosing the value v for its variable would satisfy C under the condition that all other variables involved in C retain their current value in α. We call the labeled graph $G(\mathcal{I}, \alpha)$ obtained in this way the *constraint value graph* of $\mathcal{I}$ at α. Figure 3.2 provides a visual example of our construction.

An alternative approach would be a complete encoding of each constraint relation in the graph structure. This could be achieved by adding a fourth partition of vertices to the graph where each vertex corresponds to a tuple in a relation and is connected to its associated constraint and values. However, the constraint relations can grow very large for increasing domain sizes and constraint arities. While this was our initial approach, preliminary experiments showed that it would not be scalable enough to yield competitive heuristics. Our definition of the constraint value graph avoids this issue through a partial encoding of relations based on edge labels that dynamically adapt to the current assignment. This yields a more compact graph with fewer vertices and edges. Note that the graph structure itself is not dynamic as it does not depend on the current assignment α. When searching over a sequence of assignments, the binary node and edge labels have to be updated according to the new assignment in each iteration. We rely on the fact that this update can be computed efficiently. In Section 3.4.8 we provide details on how this is achieved using GPU-accelerated operations.

3.4.2 Architecture

We construct a GNN π_θ that maps constraint value graphs to soft assignments and serves as a trainable policy for our reinforcement learning setup. Here, the real vector θ contains the trainable parameters of π_θ. The input of π_θ in iteration t is the current graph $G(\mathcal{I}, \alpha^{(t-1)})$ and recurrent vertex states $h^{(t-1)}$. The output is a new soft assignment $\varphi^{(t)}$ for $\mathcal{I}$ as well as updated recurrent states:

$$\varphi^{(t)}, h^{(t)} = \pi_\theta\big(G(\mathcal{I}, \alpha^{(t-1)}), h^{(t-1)}\big) \tag{3.12}$$

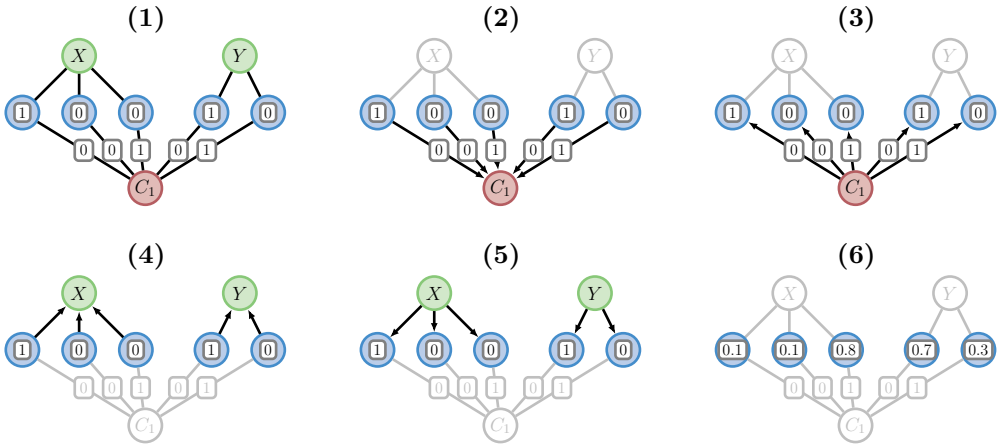


Figure 3.3: Illustration of the message passing scheme in our policy GNN π_θ. The process is performed once in each iteration t. (1) The input graph. (2) Values pass messages to constraints. (3) Constraints pass messages to values. (4) Values pass messages to variables. (5) Variables pass messages to values. (6) Values predict a new soft assignment.

The next assignment $\alpha^{(t)}$ can then be sampled from $\varphi^{(t)}$ before the process is repeated.

Our architecture is a recurrent message passing GNN. The main motivation for using a recurrent architecture is its ability to perform any number of message passes. During inference, we may run models with a timeout during which they perform over 100K message passing while searching for a solution. A recurrent architecture is a natural fit for this setting, as it can reapply the same message passing functions in each step while retaining relevant information about the search trajectory up to step t in the latent states $h^{(t-1)}$.

Furthermore, the architecture of π_θ is heterogeneous since it uses distinct trainable functions for each of the three vertex types in the constraint value graph. In each search step t π_θ performs 4 directed message passes in the following order: (1) values to constraints, (2) constraints to values, (3) values to variables, (4) variables to values. This process is illustrated in Figure 3.3. The first two message passes incorporate the node and edge labels and enable the values to gather information about how changes to the current assignment affect each constraint. The final two message passes allow the values of each domain to negotiate the next variable assignment. Note that this procedure is carried out *once* in each search iteration t. As the recurrent states can carry aggregated information across search iterations we found a single round of message passes per iteration sufficient. We also tested variants with multiple rounds of message passing in each search step t, but these did not perform better.

In the following, we will provide a formal definition of our architecture. The main hyperparameters of π_θ are the latent dimension $d \in \mathbb{N}$ and the aggregation function AGG which we either choose as an element-wise sum, mean or max function. Our GNN is then composed of the following trainable components:

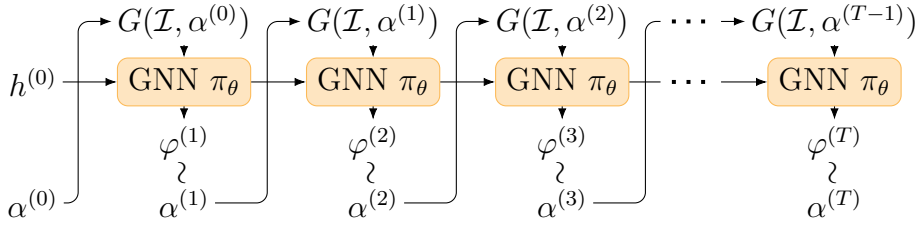


Figure 3.4: Illustration of a run of ANYCSP on a given CSP instance $\mathcal{I}$. We iteratively apply our policy GNN π_θ to the constraint value graph $G(\mathcal{I}, \alpha^{(t-1)})$ of $\mathcal{I}$ and the current assignment $\alpha^{(t-1)}$. From this we obtain a soft assignment $\varphi^{(t)}$ from which the next assignment $\alpha^{(t)}$ is sampled freely with no restrictions to locality.

- A GRU-Cell $\mathbf{G} : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^d$ and its trainable initial state $\mathbf{h} \in \mathbb{R}^d$. This cell is used to update the recurrent value states.
- A value encoder MLP $\mathbf{N} : \mathbb{R}^{d+1} \to \mathbb{R}^d$ which merges the information of the recurrent state and the binary label of each value.
- Two linear perceptrons $\mathbf{M}_\mathcal{V}, \mathbf{M}_\mathcal{C} : \mathbb{R}^d \to \mathbb{R}^{2d}$. These functions are used to generate the messages that are sent from values to constraints and from constraints to values, respectively.
- Three MLPs $\mathbf{U}_\mathcal{V}, \mathbf{U}_\mathcal{C}, \mathbf{U}_\mathcal{X} : \mathbb{R}^d \to \mathbb{R}^d$ for combining aggregated messages for values, constraints and variables, respectively.
- The output MLP $\mathbf{O} : \mathbb{R}^d \to \mathbb{R}$ which generates the logit scores for each value before we apply the domain-wise softmax.

The combined trainable weights of these functions form the parameter vector θ. The MLPs $\mathbf{N}, \mathbf{U}_\mathcal{V}, \mathbf{U}_\mathcal{C}, \mathbf{U}_\mathcal{X}$ and $\mathbf{O}$ all have two layers. The hidden layer is ReLU-activated and has dimension d while the second layer is linear. We also note that $\mathbf{N}, \mathbf{M}_\mathcal{V}, \mathbf{M}_\mathcal{C}, \mathbf{U}_\mathcal{V}, \mathbf{U}_\mathcal{C}$ and $\mathbf{U}_\mathcal{X}$ each apply LayerNorm to their output, which we found to significantly improve convergence during training.

In iteration t we associate a recurrent state $h^{(t)}(v) \in \mathbb{R}^d$ with each value $v \in \mathcal{V}$. These states are passed on from the previous iteration $t-1$ and initialized as $h^{(0)}(v) = \mathbf{h}$. Constraints and variables do not have recurrent states. We did try model variants with stateful constraints and variables but these did not perform better while being slower.

π_θ then performs the following message passing procedure in each iteration t: First, each value $v \in \mathcal{V}$ generates a latent state $x^{(t)}(v)$ by applying the encoder $\mathbf{E}$ to its recurrent state and its binary label:

$$x^{(t)}(v) = \mathbf{N}\Big(\big[h^{(t-1)}(v), L_V^{(t-1)}(v)\big]\Big) \tag{3.13}$$

Here, $[\cdot, \cdot]$ denotes the concatenation of vectors. The latent state is then used to generate two messages for each value by applying the message generation MLP

$\mathbf{M}_{\mathcal{V}}$:

$$m^{(t)}(v,0), m^{(t)}(v,1) = \mathbf{M}_{\mathcal{V}}\big(x^{(t)}(v)\big) \tag{3.14}$$

Note that the output of $\mathbf{M}_{\mathcal{V}}$ has dimension $2d$ and is the stack of both d-dimensional messages. The message $m^{(t)}(v,i)$ is send along all constraint edges (C,v) with label $L_E(C,v) = i$. Hence, the edge labels are incorporated by generating different messages for each label. The constraints aggregate these messages and process the result with their message generation function $\mathbf{M}_{\mathcal{C}}$:

$$y^{(t)}(C) = \mathrm{AGG}\left(\{\!\{m^{(t)}(v, L_E(C,v)) \mid v \in \mathcal{N}(C)\}\!\}\right) \tag{3.15}$$

$$m^{(t)}(C,0), m^{(t)}(C,1) = \mathbf{M}_{\mathcal{C}}\big(y^{(t)}(C)\big) \tag{3.16}$$

These messages are then aggregated by the values that combine the information with their latent state x by applying the update MLP $\mathbf{U}_{\mathcal{V}}$:

$$y^{(t)}(v) = \mathrm{AGG}\left(\{\!\{m^{(t)}(C, L_E(C,v)) \mid C \in \mathcal{N}(v) \cap \mathcal{C}\}\!\}\right) \tag{3.17}$$

$$z^{(t)}(v) = \mathbf{U}_{\mathcal{V}}\big(x^{(t)}(v) + y^{(t)}(v)\big) + x^{(t)}(v) \tag{3.18}$$

Note that we added a residual connection around $\mathbf{U}_{\mathcal{V}}$ for better gradient flow. In the next phase of our message passing procedure values exchange messages with their respective variables. To this end, each variable $X \in \mathcal{X}$ pools the latent states of their respective values and applies $\mathbf{U}_{\mathcal{X}}$ to obtain a variable-level latent representation $z^{(t)}(X)$:

$$z^{(t)}(X) = \mathbf{U}_{\mathcal{X}}\Big(\mathrm{AGG}\left(z^{(t)}(v) \mid v \in D_X\}\!\}\right)\Big) \tag{3.19}$$

This representation is sent back to each value $v \in \mathcal{V}_X$ of X, where it is combined with the value-level latent state by a simple addition. Note that this final message pass needs no aggregation as every value is connected to exactly one variable. The result is used as input to the GRU-Cell $\mathbf{G}$, which updates the recurrent states of the values:

$$h^{(t)}(v) = \mathbf{G}\Big(h^{(t-1)}(v), z^{(t)}(v) + z^{(t)}(X)\Big) \tag{3.20}$$

Finally, π_θ computes a soft assignment $\varphi^{(t)}$ for $\mathcal{I}$. To this end, the MLP $\mathbf{O}$ maps the new recurrent state of each value $v \in \mathcal{V}_X$ of each variable X to a scalar real number $o^{(t)}(v) = \mathbf{O}(h^{(t)}(v))$. We can then apply the softmax function *within each domain* to produce a soft value assignment:

$$\varphi^{(t)}(v) = \frac{\exp\left(o^{(t)}(v)\right)}{\sum_{v' \in \mathcal{V}_X} \exp\left(o^{(t)}(v')\right)} \tag{3.21}$$

This procedure leverages a major strength of our graph construction: By modeling values as vertices, we can directly process arbitrary domains with one GNN. For larger domains, we simply add more value vertices to the graph.

In the appendix, we provide the forward pass of ANYCSP as pseudocode in Algorithm 2.

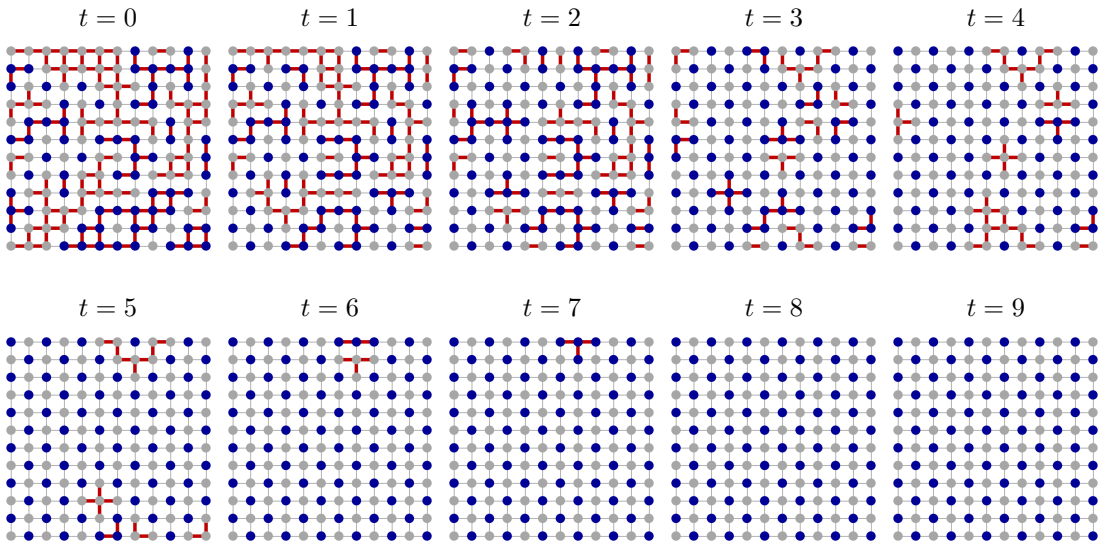


Figure 3.5: A 2-coloring for a grid graph with size 12×12 found by ANYCSP. Conflicting edges are shown in red. The initial assignment at $t = 0$ is random. An optimal solution is found after 8 search steps.

3.4.3 Global Search as a Reinforcement Learning Problem

We deploy the policy GNN π_θ as a trainable search heuristic. Note that a single GNN π_θ can search for solutions on any given CSP instance. ANYCSP takes a CSP instance $\mathcal{I}$ and a parameter $T \in \mathbb{N}$ as input and outputs a sequence $\boldsymbol{\alpha} = \alpha^{(0)}, \ldots, \alpha^{(T)}$ of assignments for $\mathcal{I}$. The initial assignment $\alpha^{(0)}$ is simply drawn uniformly at random. In each iteration $1 \leq t \leq T$ the policy GNN π_θ is applied to the current constraint value graph $G(\mathcal{I}, \alpha^{(t-1)})$ to generate a new soft assignment $\varphi^{(t)}$. The next assignment $\alpha^{(t)} \sim \varphi^{(t)}$ is then sampled from the predicted soft assignment by drawing a new value $\alpha^{(t)}(X)$ for all variables X independently and in parallel without imposing any restrictions on locality. Any number of variables may change their value in each iteration which makes our method a *global* search heuristic. This allows ANYCSP to modify different parts of the solution simultaneously to speed up the search. Figure 3.4 provides a visual illustration of the overall process and Figure 3.5 visualizes a run of a trained ANYCSP model on a 2-coloring problem for a grid graph.

Formally, our action space is the set of all assignments for the input instance, one of which must be chosen as the next assignment in each iteration t. This set is extremely large for many CSPs, with up to 10^{50} assignments to choose from for some of our training instances. Despite this, we found standard policy gradient descent algorithms to be effective and stable during training.

3.4.4 Rewarding Iterative Improvements

We construct a reward scheme that assigns a real-valued reward $r^{(t)}$ to each generated assignment $\alpha^{(t)}$. We can then use policy gradient descent to train a policy π_θ that aims to maximize this reward. Note that the reward scheme does not depend on the knowledge of optimal solutions as it simply rates the assignments that occur in a given trajectory. The reward is based on the maximization task on CSPs aiming to find assignments that satisfy as many constraints as possible. We make no modifications to the reward when training for decision problems, such as graph coloring, since this is just a special case of the maximization task where the optimal assignment satisfies all constraints. This is convenient since we do not have to ensure that all training instances are satisfiable when training for a decision problem.

A simple approach would be to use the quality $Q_{\mathcal{I}}(\alpha^{(t)})$ as a reward, which is the ratio of satisfied constraints in iteration t. However, we found that models trained with this reward tend to get stuck in local maxima and have comparatively poor performance. Intuitively, this simple reward scheme immediately punishes the policy for stepping out of a local maximum causing stagnating behavior.

We, therefore, choose a more sophisticated reward system that avoids this issue. First, in time step $t > 0$ we define the auxiliary variable $q^{(t)} = \max_{t'<t} Q_{\mathcal{I}}(\alpha^{(t')})$, which tracks the highest quality achieved before iteration t. We then define the reward in iteration t as follows:

$$r^{(t)} = \begin{cases} 0 & \text{if } Q_{\mathcal{I}}(\alpha^{(t)}) \leq q^{(t)}, \\ Q_{\mathcal{I}}(\alpha^{(t)}) - q^{(t)} & \text{if } Q_{\mathcal{I}}(\alpha^{(t)}) > q^{(t)}. \end{cases} \tag{3.22}$$

The policy earns a positive reward in iteration t if the new assignment $\alpha^{(t)}$ satisfies more constraints than any assignment generated in the trajectory up to time t. In this case, the reward is the margin of improvement. Note that the reward is 0 in any step in which the new assignment is not an improvement over the previous best regardless of whether the quality of the solution is increasing or decreasing. This reward is designed to encourage π_θ to yield iteratively improving assignments while being agnostic towards how the assignments change between improvements.

Our reward is conceptually similar to that of ECO-DQN (Barrett et al., 2020). The main difference is that we do not add intermediate rewards for reaching local maxima. Inductively, we observe that the total reward over all iterations is given by

$$\sum_{t=1}^{T} r^{(t)} = q^{(T+1)} - Q_{\mathcal{I}}(\alpha^{(0)}). \tag{3.23}$$

For any input instance $\mathcal{I}$ this cumulative reward is maximal (relative to $Q_{\mathcal{I}}(\alpha^{(0)})$) if and only if the highest achieved quality $q^{(T+1)}$ is the optimal quality for $\mathcal{I}$. In Section 3.5.8 we provide an ablation study where we compare our reward scheme to the simpler choice of using $Q_{\mathcal{I}}(\alpha^{(t)})$ directly as a reward.

3.4.5 Markov Decision Process

For a given input $\mathcal{I}$ we model the procedure described so far as a Markov Decision Process $\mathcal{M}(\mathcal{I})$ which will allow us to deploy standard reinforcement learning methods for training: The state in iteration t is given by $s^{(t)} = (\alpha^{(t)}, q^{(t)})$ and contains the current assignment and highest quality achieved before step t. The initial assignment $\alpha^{(0)}$ is drawn uniformly at random and $q^{(0)} = Q_{\mathcal{I}}(\alpha^{(0)})$. The space of actions $\mathcal{A}$ is simply the set of all possible assignments for $\mathcal{I}$. The soft assignments produced by the policy π_θ are distributions over this action space. After the next action is sampled from this distribution, the state transition of the MDP is deterministic and updates the state with the chosen assignment and its quality: The reward $r^{(t)}$ at time t is defined as in Equation 3.22. Formally, the state and action space also contains the recurrent states $h^{(t)}$ which we omit here for clarity.

3.4.6 Training

During training, we assume that some data generating distribution Ω of CSP instances is given. We aim to find a policy that performs well on this distribution of inputs. Ideally, we need to find the set of parameters θ^* which maximizes the expected total reward if we first draw an instance from Ω and then apply the model to it for $T_{\text{train}} \in \mathbb{N}$ steps:

$$\theta^* = \arg\max_\theta \mathop{\mathbf{E}}_{\substack{\mathcal{I}\sim\Omega \\ \boldsymbol{\alpha}\sim\pi_\theta(\mathcal{I})}} \Big[\sum_{t=1}^{T_{\text{train}}} \gamma^{t-1} r^{(t)} \Big] \tag{3.24}$$

The discount factor $\gamma \in [0, 1]$ and the number of search iterations during training T_{train} are both hyperparameters. Starting with randomly initialized parameters θ, we utilize REINFORCE (Williams, 1992) to train π_θ with stochastic policy gradient ascent. REINFORCE is a natural choice for training ANYCSP since it can effectively handle the large size of the action space $\mathcal{A}$. Note that we use vanilla REINFORCE without a baseline or critic network and we sample a single trajectory for every training instance. We found this simple version of the algorithm to be surprisingly robust and effective in our setting.

Let us formalize how we apply REINFORCE when training an ANYCSP model. Recall that our action space is extremely large as we choose one assignment from the set of all possible assignments in each step. We can handle this action space efficiently because we model probability distributions over this space as soft assignments from which a new value is sampled independently for every variable. The probability with which a hard assignment α is sampled from a soft assignment φ is therefore given by

$$\mathbf{P}(\alpha|\varphi) = \prod_{X\in\mathcal{X}} \varphi(\alpha(X)). \tag{3.25}$$

Note that sampling one assignment $\alpha \sim \varphi$ and computing its probability according to 3.25 are both efficient operations and are highly parallelizable. These are the only operations on the actions space that we need for training and testing.

In each training step, we independently draw a batch of training instances from Ω. For each such instance $\mathcal{I}$, we first run ANYCSP for T steps to generate sequences of soft assignments $\boldsymbol{\varphi}_\theta = \varphi_\theta^{(1)}, \ldots, \varphi_\theta^{(T)}$ and hard assignments $\boldsymbol{\alpha} = \alpha^{(1)}, \ldots, \alpha^{(T)}$. Note that we added θ as a subscript to the soft assignments to indicate that the parameters in θ have a partial derivative with respect to the probabilities stored in $\varphi_\theta^{(t)}$. We first define G_t as the discounted future reward after step t:

$$G_t = \sum_{k=t}^{T} \gamma^{k-t} r^{(k)} \tag{3.26}$$

Here, $\gamma \in [0, 1]$ is the discount factor that we choose empirically as $\gamma = 0.75$. The purpose of the discount factor is to encourage the policy to earn rewards quickly. Our objective is to find parameters θ that maximize the discounted reward over the whole search:

$$J(\theta) := \mathop{\mathbf{E}}_{\boldsymbol{\alpha} \sim \pi_\theta(\mathcal{I})} \Big[\sum_{t=1}^{T} \gamma^{t-1} r^{(t)} \Big] \tag{3.27}$$

REINFORCE (Williams, 1992) enables us to estimate the policy gradient as follows:

$$\nabla_\theta J(\theta) = \nabla_\theta \sum_{t=1}^{T} G_t \log \mathbf{P}(\alpha^{(t)} | \varphi_\theta^{(t)}) \tag{3.28}$$

$$= \nabla_\theta \sum_{t=1}^{T} \Big(G_t \sum_{X \in \mathcal{X}} \log \big(\varphi_\theta^{(t)}(\alpha^{(t)}(X)) \big) \Big) \tag{3.29}$$

Equation 3.29 simply applies Equation 3.25 to factorize the probability of the global action. Adjusting a model with these gradients can be rephrased into a standard loss minimization task for deep learning:

$$\mathcal{L}(\mathcal{I}, \boldsymbol{\alpha}, \boldsymbol{\varphi}; \theta) = -\sum_{t=1}^{T} \Big(G_t \cdot \sum_{X \in \mathcal{X}} \log \left(\varphi^{(t)}(\alpha^{(t)}(X)) + \epsilon \right) \Big) \tag{3.30}$$

Optimizing this loss is equivalent to training a model with the gradient estimates in Equation 3.29. Note that we added a small $\epsilon = 10^{-5}$ for numerical stability. This loss is averaged over all instances in the current batch and then used for one step of gradient descent with the Adam optimizer. We sample a single trace for each instance in the current batch. The process is repeated in the next training step with a new batch of training instances. Algorithm 1 in the appendix provides our overall training procedure as pseudocode.

This training procedure is simply the standard REINFORCE algorithm applied to our Markov Decision Process. We do not use a baseline or critic network. We initially expected this simple algorithm to be unable to estimate a useful policy gradient given the unusually large size of our action space and hard nature of

our learning problem. Contrary to this expectation REINFORCE is able to train ANYCSP effectively and all experimental results reported in Section 3.5 are based on this simple approach.

3.4.7 Expressivity

Considering the expressivity of the used architecture is critical in the context of neural combinatorial optimization, where the structural details of an input instance are highly important. A well-known theoretical result on MPGNNs is their correspondence to the Weisfeiler-Lehman isomorphism test, as discussed in Section 2.2.1. More specifically, standard MPGNNs can not distinguish more structures than the 1-dimensional Weisfeiler-Lehman test. For node-level tasks, this prohibits an MPGNN from mapping two different nodes with identical n-hop subtrees to different outputs after n message passes. For some graph structures, such as regular graphs, this makes certain combinatorial tasks, such as graph coloring, fundamentally impossible with standard MPGNNs.

Crucially, ANYCSP is **not** limited by 1-WL. Our policy GNN π_θ has access to randomness which has been proven to strengthen GNN expressivity beyond the WL-hierarchy (Abboud et al., 2021a, Sato et al., 2021). In each iteration π_θ predicts a soft assignment. From this we sample a hard assignment and pass it back to the GNN as a binary pattern. This process gives ANYCSP access to randomness in every search iteration. The GNN can predict soft assignments with high variance to break symmetries through random sampling. Empirically this is also demonstrated in the MaxCut experiment in Section 3.5.4. The graphs $G48$ and $G49$ of GSet are both 4-regular toroidal graphs. ANYCSP computes optimal cuts for both graphs, which correspond to conflict-free 2-colorings. This could not be achieved by any function limited by 1-WL.

Our experiments in Section 3.5 also have an interesting connection to the notion of uniform function approximation. As discussed in Section 2.2.2, uniform expressivity aims to characterize the ability of GNNs to robustly approximate a target function for all graph sizes. In our experiments, we usually do have significant distribution shifts from relatively small training instances to much larger benchmark problems in testing. The performance therefore strongly depends on a model's ability to generalize to o.o.d. graph sizes after training. When tuning the hyperparameters of ANYCSP we found max-aggregation to consistently perform best on decision problems while mean worked better for maximization tasks. sum-aggregation proved to be unstable for most problems. We note that Joshi et al. (2020) has also reported similar findings when learning TSP heuristics. These observations generally align with the main results of Rosenbluth et al. (2023), who show that sum-aggregation does not subsume mean- or max-aggregation with respect to uniform function approximation, even though it is strictly more expressive in terms of distinguishing graphs up to bounded size (Xu et al., 2019). Since our focus is on learning heuristics with robust generalization to larger, denser graphs it is

therefore not surprising that both mean- and max-aggregation outperform sum across a range of tasks.

Overall, we believe that these considerations illustrate why the expressivity of GNNs is a multifaceted topic. Designing effective architectures requires balancing different notions of expressivity, such as distinguishing structures and generalizing robustly.

3.4.8 Relabeling Constraint Value Graphs

One critical requirement for ANYCSP is a fast subroutine for recomputing the edge labels L_E given the newly sampled assignment $\alpha^{(t)}$ in step t. Our implementation of this relabeling procedure is based entirely on PyTorch and is GPU-accelerated to maximize performance. Here, we will briefly discuss how this implementation works. Let $\mathcal{I} = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ be a CSP instance and let α be the newly sampled assignment for which we have to compute the edge labels L_E. We first compute the node labels L_V, which are a simple binary encoding of α. Let $C \in \mathcal{C}$ be some constraint with scope $s^C = (X_1, \dots, X_k)$, relation $R^C \in \mathcal{D}(X_1) \times \cdots \times \mathcal{D}(X_k)$ and arity k. For each tuple $r \in R^C$ we compute a score that counts how many of the values occurring in t are currently chosen by α:

$$s(r) = \sum_{i=1}^{k} L_V(r_i) \tag{3.31}$$

For each value $v \in \mathcal{V}_{X_i}$ of each variable X_i in the scope of C we then compute the maximum of $s(t)$ over all tuples $r \in R^C$ with $v \in r$:

$$m(C, v) = \max_{r \in R^C, v \in r} s(r) \tag{3.32}$$

For each value v we observe that $m(C, v) - L_V(v) = k - 1$ if and only if there exists some tuple $r \in R^C$ with $v \in r$ such that for all $v' \in r, v' \neq v$ we already have $v' \in \alpha$. This is equivalent to our original definition of L_E and we can use this case distinction to obtain the edge labels:

$$L_E(C, v) = \begin{cases} 1 & \text{if } m(C, v) - L_V(v) + 1 = k, \\ 0 & \text{otherwise.} \end{cases} \tag{3.33}$$

Our implementation maintains edge lists that connect values and constraint edges to their respective tuples across all constraints. With these edge lists, Equations 3.31 and 3.32 are standard scatter operations, and Equation 3.33 is carried out with standard arithmetic. These functions are fully GPU-supported and allow us to rapidly update all edge labels in parallel. We can optimize this further by allowing relations to be specified in terms of the disallowed tuples (conflicts), rather than the allowed ones. In this case, we can carry out the exact same procedure, except that we swap the labels in the case distinction of Equation 3.33. This optimization is very useful for many problems. SAT in particular has constraints that all forbid

	Model RB	k-Col	3-Sat	Max-k-Sat	MaxCut
d	128	128	128	128	128
AGG	max	max	max	mean	sum
λ	0.75	0.75	0.75	0.75	0.75
T_{train}	40	40	40	40	40
batch size	25	25	25	25	25
lr	$5 \cdot 10^{-6}$	$5 \cdot 10^{-6}$	$5 \cdot 10^{-6}$	$5 \cdot 10^{-6}$	$5 \cdot 10^{-6}$

Table 3.1: Selected hyperparameters for each considered CSP.

exactly one tuple. It is therefore more efficient to work with this one forbidden tuple rather than the $2^k - 1$ allowed tuples.

3.5 Experiments

We evaluate ANYCSP on a wide range of well-known CSPs: Boolean satisfiability (3-Sat) and its maximization version (Max-k-Sat for $k = 3, 4, 5$), graph coloring (k-Col), maximum cut (MaxCut) as well as random CSPs (generated by the so-called Model RB). These problems are of high theoretical and practical importance and are commonly used to benchmark CSP heuristics. We train one ANYCSP model for each of these problems using randomly generated instances. Recall that the process of learning problem-specific heuristics with ANYCSP is purely data-driven as our architecture is generic and can take any CSP instance as input. We cross-compare all models on each others CSPs in Section 3.5.7.

3.5.1 Experimental Setup and Hyperparameters

We implement ANYCSP in PyTorch[2]. The code for relabeling $G(\mathcal{I}, \alpha^{(t)})$ in each iteration t is also fully based on PyTorch and is GPU-compatible. All neural approaches run with one NVIDIA Quadro RTX A6000 GPU with 48GB of memory. All classical approaches run on an Intel Xeon Platinum 8160 CPU (2.1 GHz) and 64GB of RAM. When applicable, we also tune the configuration of the classical algorithms on our validation data to ensure a fair comparison.

Before training (and hyperparameter tuning) we sample fixed validation datasets of 200 instances from the given training distribution of CSP instances. We usually modify the distribution to yield larger instances than those used for training. This favors the selection of models that generalize well to larger instances, which is almost always desirable in our context. During validation we perform $T_{\text{val}} = 200$ search iterations on each validation instance. The metric used for selection is the number of unsatisfied constraints in the best solution averaged over all validation

[2] `https://github.com/toenshoff/ANYCSP`

instances. To save compute resources we perform only 100K training steps with each hyperparameter configuration and only perform the full 500K steps of the training with the best configuration. The aggregation function $\text{AGG} \in \{\text{sum}, \text{mean}, \text{max}\}$ is a key hyperparameter. The choice of aggregation is critical for performance, as we observed max-aggregation to consistently perform best on decision problems but worse on maximization tasks. The hidden dimension is set to $d = 128$. We also validated some models with $d = 64$ but larger models seem to be more capable. We did not increase d further to avoid memory bottlenecks. We tuned the discount factor $\gamma \in \{0.5, 0.75, 0.9, 0.99\}$ and found the value of 0.75 to yield the best results in all of our experiments. The learning rate is initialized as $\text{lr} = 5 \cdot 10^{-6}$ and decays linearly throughout training to a final value of $\text{lr} = 5 \cdot 10^{-7}$. All models train with a batch size of 25. We also considered larger batch sizes of 50 and 100 without improvement. Table 3.1 specifies the final configuration used in each experiment.

We train with the Adam optimizer (Kingma and Ba, 2014) for 500K training steps. For each training distribution Ω we implement data loaders that sample new instances on-the-fly in each training step. In this setup, the number of training instances is virtually unlimited and only bounded by the number of training steps. With our hyperparameters we therefore train each model on 12.5 Million sampled training instances. Training a model takes between 24 and 48 hours, depending on the training data.

3.5.2 Model RB

We evaluate ANYCSP on general CSP benchmark instances generated by the Model RB (Xu and Li, 2003). This model provides a way to generate hard random CSP instances by randomly choosing a number of disallowed tuples of a fixed arity. A class of random CSP instances of Model RB is denoted $\text{RB}(k, n, \alpha, r, p)$ where each instance consists of $n \geq 2$ variables with domain size $d = n^\alpha$ for $a > 0$. Each instance has $m = rn \ln n$ constraints for $r > 0$ of arity $k \geq 2$, with each constraint disallowing $t = pd^k$ randomly selected tuples. The hardest of the Model RB instances occur around the critical value $p_{cr} = 1 - e^{-\alpha/r}$ of p (Xu and Li, 2003).

Data Our training data consists of randomly generated Model RB instances with 30 variables and arity 2. We randomly select $d \in (n^{1/k}, 2n^{1/k}]$ and $m \in [n \log_k d, 2n \log_k d]$ and generate instances with $p = 0.9 p_{cr}$ slightly smaller than the critical values of p to increase the number of satisfiable instances seen during training. To generate one instance we add m constraints, each by randomly selecting a scope of k distinct variables and a corresponding relation of t distinct disallowed tuples. Our validation data contains 200 instances sampled from the exact same distribution. The test dataset is obtained from the XCSP project (Audemard et al., 2020) and contains 50 satisfiable Model RB instances with 50 variables, each with domain size 22 and about 500 constraints of arity 2. More specifically, we use all instances of the `Random-RB-2-50-23f` dataset as our test data. These instances each contain 50 variables, domains with 23 values and roughly 500 constraints.

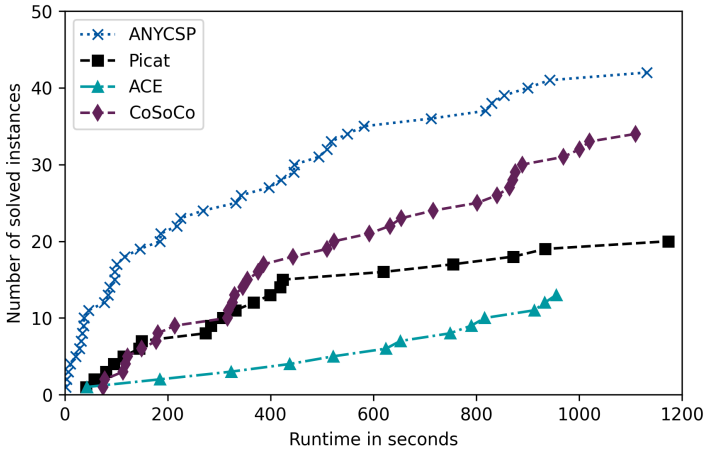


Figure 3.6: Survival plot for RB50. The x-axis gives the wall clock runtime in seconds. The y-axis counts the cumulative number of instances solved within a given time.

Note that the hardness of Model RB problems comes from the dense, random constraint relations chosen at the threshold of satisfiability and even instances with 50 variables are very challenging.

Baselines We use three state-of-the-art CSP-solvers from the XCSP Competition as baselines: Picat (Zhou, 2022), ACE (Lecoutre, 2022), and CoSoCo (Audemard, 2018). Picat is a SAT-based solver and the winner of the XCSP Competition 2022 (Audemard et al., 2020). ACE and CoSoCo are based on constraint propagation. We include CoSoCo because it demonstrated very strong performance specifically on binary Model RB instances in previous CSP Competitions. Indeed, it also is the best-performing baseline in our experiment. No prior neural baseline exists for this problem. All algorithms run once on each instance with a timeout of 20 minutes.

Results Figure 3.8 provides the results on the RB50 dataset. ANYCSP solves the most instances by a substantial margin. The second strongest approach is the CoSoCo solver which solves 34 instances in total, 8 less than ANYCSP. The learned heuristic is therefore a highly competitive search algorithm for random CSP instances. Within the timeout of 20 Minutes, ANYCSP will perform roughly 500K search iterations. Recall that we set $T_{\text{train}} = 40$. Therefore, the learned policy generalizes to searches that are over 10K times longer than those seen during training. This also serves as a demonstration of a GNN that can perform an extremely large number of message passes without losing its ability to compute its target function effectively.

3.5.3 Graph Coloring

We consider the problem of finding a conflict-free vertex coloring for a given graph G and number of colors k. The corresponding CSP instance has variables for each vertex, domains containing the k colors, and one binary "not equal"-constraint for each edge.

Data We train on a distribution Ω_{COL} of graph coloring instances for random graphs with 50 vertices. We mix Erdős-Rényi, Barabási-Albert and random geometric graphs in equal parts and choose the number of colors k from $[3, 10]$. Concretely, we mix the following 3 distributions uniformly:

- Erdős-Rényi graphs with $n = 50$ vertices and edge probability $p \sim U[0.1, 0.3]$
- Barabási-Albert graphs with $n = 50$ vertices and parameter $m \sim U[2, 10]$
- Random geometric graphs with $n = 50$ vertices distributed uniformly at random in a 2-dimensional 1×1 square. The edge threshold radius is drawn uniformly from $r \sim U[0.15, 0.3]$.

For each graph G drawn from this distribution we then choose a number of colors $k \in [3, 10]$ as follows: We first apply a linear time greedy coloring heuristic as implemented by NetworkX (Hagberg et al., 2008) to color the graph without conflict. If the greedy heuristic required k' colors for G, then we pose the problem of coloring G with k colors as the training CSP instance, where k is chosen as:

$$k = \max\{3, \min\{10, k' - 1\}\} \tag{3.34}$$

Intuitively, ANYCSP has to color each graph with 1 color less than the greedy heuristic. Some of these instances are unsatisfiable, which is not a problem for our reward scheme and training procedure. We found this simple method to be very effective at quickly generating graph coloring instances around the threshold of satisfiability with minimal fine-tuning. The 200 validation instances are generated with the same parameters and procedure, except that we increase the number of vertices for all three graph types to $n = 200$.

As test instances we use 100 structured benchmark graphs with known chromatic number $\mathcal{X}(G)$. The instances are obtained from a collection of hard coloring instances commonly used to benchmark heuristics[3]. They are highly structured and come from a wide range of synthetic and real problems. We divide the test graphs into two sets with 50 graphs each: $\text{COL}_{<10}$ contains graphs with $\mathcal{X}(G) < 10$ and $\text{COL}_{\geq 10}$ contains graphs with $\mathcal{X}(G) \geq 10$. The graphs in $\text{COL}_{\geq 10}$ have up to 1K vertices, 19K edges and a chromatic number of up to 73. This experiment tests generalization to larger domains and more complex structures.

[3] `https://sites.google.com/site/graphcoloring/vertex-coloring`

Method	$\text{COL}_{<10}$	$\text{COL}_{\geq 10}$
RUNCSP	33	-
CoSoCo	49	33
Picat	49	38
Greedy	16	15
DSatur	38	28
HybridEA	**50**	**40**
ANYCSP	**50**	**40**

Table 3.2: Results on structured graph coloring instances. We provide the number of instances solved with a 20-minute timeout for both splits, each containing 50 instances with chromatic number less than 10 and at least 10, respectively.

Baselines We compare the performance to three problem-specific heuristics: a simple greedy algorithm, the classic heuristic DSATUR (Brélaz, 1979) and the state-of-the-art heuristic HybridEA (Galinier and Hao, 1999), all implemented efficiently by (Lewis et al., 2012, Lewis, 2015). To evaluate the CSP solvers Picat (Zhou, 2022) and CoSoCo (Audemard, 2018) in this experiment we reduce each coloring instance to a CSP instance in the XCSP3 format. Additionally, we evaluate RUNCSP as a neural baseline on $\text{COL}_{<10}$. Unlike ANYCSP, RUNCSP requires us to fix a domain size before training. Therefore, we must train one RUNCSP model for each tested chromatic number $4 \leq \mathcal{X}(G) \leq 9$ and omit testing on $\text{COL}_{\geq 10}$. For this, we use the same training data distributions as in Tönshoff et al. (2021). We consider a graph solved by RUNCSP if the model trained for the graph's chromatic number is able to find a conflict-free coloring.

Reults Table 3.2 provides the number of solved k-Col instances from both splits. ANYCSP is on par with HybridEA which solves the most instances of all baselines. RUNCSP solves significantly fewer instances than ANYCSP on $\text{COL}_{<10}$ and outperforms only the simple greedy approach. The optimally colored graphs include the largest instance with 73 colors. Since ANYCSP trains with 3 to 10 colors the trained model is able to generalize to significantly larger domains. Note that detailed instance-level results are provided in Tables A.3 and A.4 in the appendix. Out of the 50 instances in $\text{COL}_{\geq 10}$ both ANYCSP and HybridEA solve 40 instances each. In this comparison, 39 instances are solved by both methods. Only two graphs are colored optimally by one approach but not the other resulting in almost identical subsets of solved instances. From a practical standpoint, it would arguably be more desirable if classical and neural approaches would each succeed on different sets of instances, as this would yield a much stronger algorithm when ensembling both types of methods. Here, we do observe that instances that are hard for classical approaches are also hard for ANYCSP and vice versa.

Method	$\|V\|$=800	$\|V\|$=1K	$\|V\|$=2K	$|V|\geq$3K
Greedy	411.44	359.11	737.00	774.25
SDP	245.44	229.22	-	-
RUNCSP	185.89	156.56	357.33	401.00
ECO-DQN	65.11	54.67	157.00	428.25
ECORD	8.67	8.78	39.22	187.75
ANYCSP	**1.22**	**2.44**	**13.11**	**51.63**

Table 3.3: MaxCut results on Gset graphs. The graphs are grouped by their vertex counts and we provide the mean deviation from the best known cut size.

3.5.4 MaxCut

In our MaxCut experiment, we focus on unweighted instances. In this case, MaxCut is identical to maximum 2-Colorability. We can model the MaxCut problem for a given graph G as a CSP instance by using the same reduction we use for graph coloring but with the number of colors fixed at $k = 2$.

Data Our training distribution Ω_{MCUT} consists of Erdős-Rényi graphs with $n = 100$ vertices and an edge probability sampled uniformly from $p \in [0.05, 0.3]$. Our validation data uses the same distribution for p but with $n = 500$ vertices. Our test data is Gset[4] a collection of commonly used MaxCut benchmarks of varying structure with 800 to 10K vertices. We divide the test graphs into groups by the number of vertices (8-9 graphs per group) and report the mean deviation from the best-known cuts obtained by Benlic and Hao (2013) for each method.

Baselines We compare ANYCSP to three neural baselines: RUNCSP, ECO-DQN (Barrett et al., 2020), and ECORD (Barrett et al., 2022). RUNCSP is also trained on Ω_{MCUT}. We train and validate ECO-DQN and ECORD models with the same data that Barrett et al. (2022) used for their Gset experiments as we found this to be better for training their models than our own distribution Ω_{MCUT}. We omit S2V-DQN (Khalil et al., 2017) since ECO-DQN and ECORD have been shown to yield substantially better cuts. We adopt the evaluation setup of ECORD and run the neural methods with 20 parallel runs and a timeout of 180s on all unweighted instances of Gset. The results of a standard greedy construction algorithm and the well-known SDP-based approximation algorithm by Goemans and Williamson (1995) are also included as classical baselines[5] , which runs with a 3-hour timeout for graphs with up to 1K vertices.

Results Table 3.3 provides results for Gset. Note that full instance-level results are provided in Table A.2 in the appendix. ANYCSP outperforms all baselines

[4] `https://web.stanford.edu/~yyye/yyye/Gset`, Yinyu Ye, 2003
[5] `https://github.com/hermish/cvx-graph-algorithms`, Hermish Mehta, 2019

Method	SL50	SL100	SL150	SL200	SL250
RLSAT	**100**	87	67	27	12
PDP	93	79	72	57	61
WalkSAT	**100**	**100**	97	93	87
ProbSAT	**100**	**100**	97	87	92
ANYCSP	**100**	**100**	**100**	**97**	**99**

Table 3.4: Number of solved 3-Sat benchmark instances from SATLIB. For each number of variables there are 100 satisfiable test instances.

across all graph sizes by a large margin. Recall that RUNCSP trains on a soft relaxation of MaxCut while ECO-DQN and ECORD are both neural local search approaches. Neither concept matches the results of our global search approach trained with policy gradients. The margin is particularly significant on the largest graphs with 3k to 10K vertices.

3.5.5 3-SAT

We further conduct experiments on 3-Sat, which is the decision problem for Boolean CNF formulas where all clauses have an arity of three.

Data For 3-Sat we choose the training distribution $\Omega_{3\text{SAT}}$ as uniform random 3-Sat instances with 100 variables. The ratio of clauses to variables is drawn uniformly from the interval $[4, 5]$ which is roughly where the threshold of satisfiability is located for uniform random 3-Sat. For validation, we use formulas with 200 variables and the same density distribution. We test on commonly used benchmark instances for uniform 3-Sat from SATLIB[6]. The test set SLN contains 100 satisfiable instances with $N \in \{50, 100, 150, 200, 250\}$ variables each. The density of these formulas is at the threshold of satisfiability.

Baselines We evaluate two neural baselines: RLSAT (Yolcu and Póczos, 2019) and PDP (Amizadeh et al., 2019). PDP uses an architecture similar to RLSAT but is trained toward a soft differentiable objective, similar to RUNCSP. In our experiments, PDP is also trained on $\Omega_{3\text{SAT}}$. We train RLSAT with the curriculum learning dataset for 3-Sat provided by its authors, since its reward scheme is incompatible with our partially unsatisfiable training instances. We also adopt the experimental setup of RLSAT, which limits the evaluation run by the number of search steps instead of a timeout. The provided code for both PDP and RLSAT is comparatively slow and a timeout would compare implementation details rather than the capability of the learned algorithms. We also evaluate two conventional local search heuristics: The classical WalkSAT algorithm (Selman et al., 1993)

[6] `https://www.cs.ubc.ca/~hoos/SATLIB/benchm.html`

Method	3CNF	4CNF	5CNF
WalkSAT	2145.28	1556.68	1685.10
CCLS	1567.24	1323.14	1315.96
SATLike	1595.86	1188.56	1152.88
ANYCSP	**1537.46**	**1126.44**	**1105.62**

Table 3.5: Results Max-k-Sat instances with 10K variables. For each $k \in \{3, 4, 5\}$ we provide the mean number of unsatisfied clauses over 50 random instances.

based on random walks and a probabilistic approach called probSAT (Balint and Schöning, 2018). Like Yolcu and Póczos (2019), we apply stochastic boosting and run each method 10 times for 10K steps on every instance. PDP is deterministic and only applied once to each formula. Note that our aim is to compare the performance of ANYCSP to prior end-2-end neural approaches and conventional stochastic search algorithms for Sat. We note that ANYCSP can not currently compete with CDCL solvers on structured decision SAT instances, like most known stochastic search heuristics.

Results Table 3.4 provides the number of solved instances for each tested size. All compared approaches do reasonably well on small instances with 50 variables. However, the performance of the two neural baselines drops significantly as the number of variables increases. ANYCSP does not suffer from this issue and even outperforms the classical local search algorithms on the three largest instance sizes considered here.

3.5.6 Max-k-SAT

Data We train on the distribution Ω_{MSAT} of uniform random Max-k-Sat instances with 100 variables and $k \in \{3, 4\}$. Here, the clause/variable ratio is chosen from $[5, 8]$ and $[10, 16]$ for $k = 3$ and $k = 4$, respectively. These formulas are denser than those of Ω_{3SAT} since we aim to train for the maximization task. Our test data for Max-k-Sat consists of uniform random k-CNF formulas generated by us. For each $k \in \{3, 4, 5\}$ we generate 50 instances with 10K variables each. The number of clauses is chosen as 75K for $k = 3$, 150K for $k = 4$ and 300K for $k = 5$. These formulas are therefore 100 times larger than the training data and aim to test the generalization to significantly larger instances as well as unseen arities, since $k = 5$ is not used for training.

Baselines Neural baselines for SAT focus primarily on decision problems. For Max-k-Sat we therefore compare ANYCSP only to classical local search heuristics:

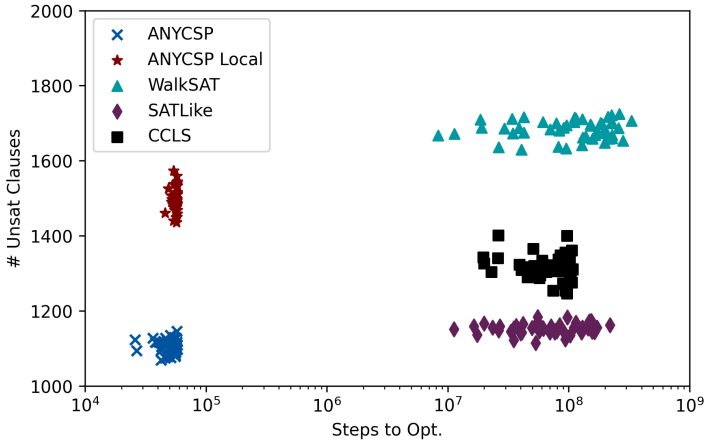


Figure 3.7: Detailed results for Max-5-SAT. For each test instance and each method we plot the number of unsatisfied clauses in the best found solution against the search step in which it was found.

the classical (Max-)WalkSAT (Selman et al., 1993) and two state-of-the-art Max-SAT local search heuristics CCLS (Luo et al., 2015) and SATLike (Cai and Lei, 2020).

Results Table 3.5 provides a comparison. We provide the mean number of unsatisfied clauses after processing each instance with a 20-minute timeout. Remarkably, ANYCSP outperforms all classical baselines by a significant margin. We note that for each tested arity ANYCSP finds the best solution on all 50 test instances. This the case even on Max-5-SAT instances, despite this arity not occurring during training.

We point out that the conventional search heuristics all perform over 100M search steps in the timeout of 20 Minutes. ANYCSP performs less than 100K steps on each instance in this experiment. The GNN cannot match the speed with which classical algorithms iterate, even though it is accelerated by a GPU. Despite this, ANYCSP consistently finds the best solutions. Figure 3.7 evaluates this surprising observation further. We plot the number of unsatisfied clauses in the best-found solution against the search step in which the solution was found (Steps to Opt.) for all methods and all instances of our Max-5-SAT test data. We also provide the results of a modified ANYCSP version (ANYCSP Local as defined in Section 3.5.8) that is only allowed to change one variable at a time and is therefore a local search heuristic. Note that the x-axis is logarithmic as there is a clear dichotomy separating neural and classical approaches: Compared to conventional heuristics ANYCSP performs roughly three orders of magnitude fewer search steps in the same amount of time. When restricted to local search, ANYCSP is unable to overcome this deficit and yields worse results than strong heuristics such as SATLike. However, when ANYCSP leverages global search to parallelize refinements across the whole instance it can find solutions in 100K steps that outperform state-of-the-art local search heuristics after well over 100M iterations.

Ω	RB50	$\text{COL}_{<10}$	Gset800	SL250	Max-5-CNF
$\Omega_{\mathbf{RB}}$	**42**	**50**	655.56	98	6192.18
$\Omega_{\mathbf{COL}}$	15	**50**	868.22	96	5076.16
$\Omega_{\mathbf{MCUT}}$	0	0	**1.22**	0	9048.64
$\Omega_{\mathbf{3SAT}}$	0	19	1213.11	**99**	5001.72
$\Omega_{\mathbf{MSAT}}$	0	15	1217.67	66	**1103.14**

Table 3.6: Cross-Comparison of training distributions on different test datasets.

3.5.7 Cross-Comparison

Recall that every ANYCSP model can take any CSP instance as input. We train on specific distributions of CSPs to obtain problem-specific heuristics. However, a model trained on graph coloring instances can still process 3-Sat formulas and vice versa. Naturally, we expect each model to perform best on the distribution it is trained on, but the universality of our architecture does raise interesting questions of how well models trained on one CSP perform on an entirely different CSP. In this section, we aim to study the transferability of learned heuristics across different CSPs.

Table 3.6 compares all models from our main experiments on each other's test data. For each test dataset, we use the same evaluation metric as the original experiments. First of all, each model does indeed achieve the best results on the test data of the CSP used for training. However, the degree to which each model generalizes to other problems varies substantially. There seems to be a significant compatibility between Model RB and graph coloring. The model trained on Model RB instances solves all coloring instances in $\text{COL}_{<10}$ while the model trained on graph coloring problems solves 15 Model RB benchmark instances. Both of these models also perform very well on decision 3-Sat instances. Curiously, they even outperform the model trained for Max-k-Sat on decision 3-Sat. The training distributions Ω_{3SAT} and Ω_{MSAT} are closely related and one would expect our Max-k-Sat model to do well on 3-Sat as well. While it does solve 66 of the 100 instances, it does not come close to the 96 and 98 instances solved by the policies trained on Ω_{COL} and Ω_{RB}, respectively. This surprising observation can most likely be attributed to the choice of the aggregation function. Like the model trained on Ω_{3SAT}, the Model RB and graph coloring policies both use max-aggregation. The Max-k-Sat policy uses mean-aggregation. In this comparison, choosing the right aggregation function appears to be at least as important as training on similar data. A similar observation also holds for the MaxCut model. It performs as well as random guessing on all problems other than MaxCut. Note that this model uses sum-aggregation since this performed best in validation for the MaxCut problem. However, sum-aggregation is not very robust toward changes in the distribution of inputs. The learned functions are not able to handle larger domains, arities, or degrees than those seen during training and the policy is highly specialized towards MaxCut. On the other hand,

Method	$\text{COL}_{<10}$	$\text{COL}_{\geq 10}$
$\text{ANYCSP}_{\text{loc}}$	49	37
$\text{ANYCSP}_{\text{qual}}$	37	25
ANYCSP	**50**	**40**

(a) Graph Coloring

Method	3CNF	5CNF
$\text{ANYCSP}_{\text{loc}}$	1697.08	1498.46
$\text{ANYCSP}_{\text{qual}}$	1921.70	1471.00
ANYCSP	**1537.46**	**1103.14**

(b) Max-k-Sat

Table 3.7: Ablation results on the graph coloring and Max-k-SAT test instances.

no other model achieves competitive results on MaxCut, indicating that this problem may require a higher degree of specialization.

Overall, we can conclude that some of our trained models generalize well beyond their training distribution to entirely different CSPs. However, this is strongly dependent on the specific instance distributions and the robustness of the used aggregation functions.

3.5.8 Impact of the Action Space and Reward

We further provide an empirical ablation study for two major design choices of ANYCSP: Firstly, we want to study the benefit of our exponentially sized action space when compared to a more conventional local search setting. Secondly, we aim to validate the reward scheme we constructed in Section 3.4.4. To this end, we will evaluate two modified versions of ANYCSP:

(1) $\text{ANYCSP}_{\text{loc}}$: A version of ANYCSP designed to be a local search heuristic. We modify π_θ such that the softmax over the scores in $o^{(t)}(v)$ is not performed separately within each domain but overall values in the disjoint union of domains $\mathcal{V}$:

$$\varphi^{(t+1)}(v) = \frac{\exp\left(o^{(t)}(v)\right)}{\sum_{v' \in \mathcal{V}} \exp\left(o^{(t)}(v')\right)} \tag{3.35}$$

The output $\varphi^{(t+1)}$ of π_θ in iteration $t+1$ is therefore not a soft assignment but a probability distribution over the disjoint union of domains. To obtain a new hard assignment $\alpha^{(t+1)}$ we sample a single value $v \in \mathcal{C}$ from this distribution and set it as the value for its respective variable X_v with $v \in \mathcal{V}_{X_v}$:

$$v \sim \varphi^{(t+1)} \tag{3.36}$$

$$\alpha^{(t+1)} = \alpha^{(t)}[X_v{=}v] \tag{3.37}$$

All variables other than X_v remain unchanged in iteration $t+1$. With this modification $\text{ANYCSP}_{\text{loc}}$ becomes a local search heuristic that only changes one variable at a time. The remaining architecture and training procedure are identical to ANYCSP, including the reward scheme.

Method	$\|V\|$=800	$\|V\|$=1K	$\|V\|$=2K	$|V|\geq$3K
ANYCSP$_{\text{loc}}$	14.22	26.00	84.00	197.50
ANYCSP$_{\text{qual}}$	30.11	12.89	42.56	200.75
ANYCSP	**1.22**	**2.44**	**13.11**	**51.63**

Table 3.8: Ablation results for MaxCut on the Gset instances.

(2) ANYCSP$_{\text{qual}}$: A version of ANYCSP trained by using the quality $Q_{\mathcal{I}}(\alpha^{(t)})$ of the current assignment as a reward. For this configuration to train well, we found it helpful to use the quality of the initial assignment as a baseline:

$$r^{(t)} = Q_{\mathcal{I}}(\alpha^{(t)}) - Q_{\mathcal{I}}(\alpha^{(0)}) \tag{3.38}$$

Without the subtractive baseline, we found the training to be very unstable. Note that the baseline does not solve the fundamental problem of the reward scheme: A heuristic can not leave a local maximum without being immediately punished for doing so. Here, we will study how this issue actually affects performance empirically.

We perform our ablation experiments on the graph coloring, MaxCut and Max-k-Sat problems. For each problem, we train both modifications with the same training data and hyperparameters as ANYCSP.

Table 3.7a contains the results of our ablation study for k-Col. For graph coloring, ANYCSP$_{\text{qual}}$ performs significantly worse than the other two versions of our method. Compared to our main baselines it only outperforms RUNCSP and the simple greedy approach. ANYCSP$_{\text{loc}}$ actually performs reasonably well, as it only solves four graphs less than ANYCSP across all 100 test instances. In this experiment, the reward scheme seems to contribute more to the performance than the global search action space. However, only the combination of both in ANYCSP yields the best results. Table 3.8 provides a comparison for the MaxCut problem. There is no clear hierarchy between ANYCSP$_{\text{loc}}$ and ANYCSP$_{\text{qual}}$ on this CSP. However, both ablation versions perform significantly worse than ANYCSP. Finally, Table 3.7b contains the ablation results for Max-k-Sat. As for the MaxCut problem, the two modified versions yield similar results but perform far worse than ANYCSP.

Figure 3.8 investigates the differences on the Max-k-Sat problem further. We plot how the number of unsatisfied clauses in the best-found solution evolves throughout the 60K search steps performed by ANYCSP in 20 minutes. The curves are averaged over all 50 instances in out Max-5-SAT test data. Both ANYCSP$_{\text{loc}}$ and ANYCSP$_{\text{qual}}$ are unable to converge to solutions as good as those found by ANYCSP but for different reasons. ANYCSP$_{\text{loc}}$ converges slowly but steadily. Due to the slow convergence compared to ANYCSP, it is not able to find equivalent solutions in the same amount of time. ANYCSP$_{\text{qual}}$ initially converges as fast as ANYCSP. This is expected since this version also performs global search and can refine the whole solution in parallel. However, the solution quality tapers off significantly earlier compared to ANYCSP and remains virtually constant after

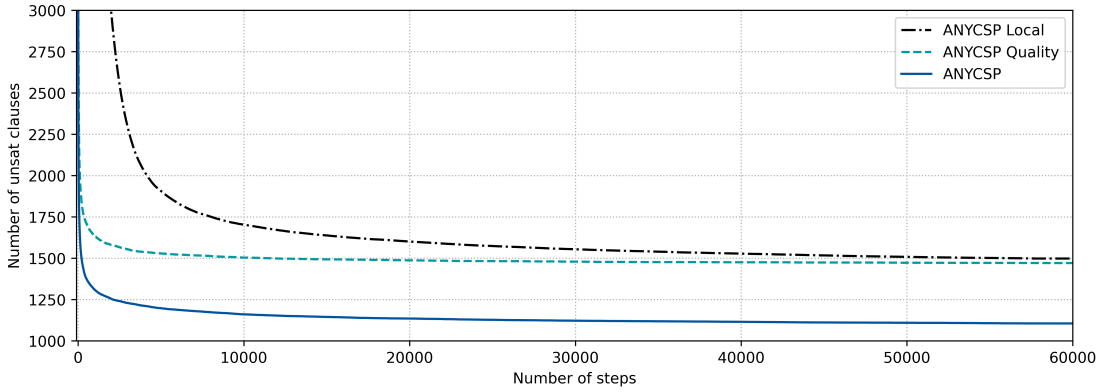


Figure 3.8: Number of unsat clauses in the best found solution on our MAX-5-SAT test instances. We plot the average over 50 instances as a function in the number of search steps t. We compare ANYCSP and its two ablation versions.

20K search steps. This is the expected problem our reward scheme intends to solve: During training, $\text{ANYCSP}_{\text{qual}}$ can not leave local maxima without being punished for doing so by the simple reward scheme. This inhibits exploration and encourages stagnation. After 60K steps, $\text{ANYCSP}_{\text{loc}}$ actually catches up to $\text{ANYCSP}_{\text{qual}}$ and both ablation versions yield similar results once the 20-minute timeout is reached.

Overall, the experiments and ablation study suggest that our main design choices are crucial to consistently obtaining strong search heuristics:

(1) A global search action space is necessary to refine the whole solution in parallel and speed up the search. Without this advantage, GNN-based heuristics can not compensate for their comparatively high computational cost.

(2) A well-chosen reward scheme that encourages exploration is equally important. Without it, global search simply gets stuck faster than local search. A simple reward proportional to the quality is not suitable in this regard.

ANYCSP combines these insights in one generic architecture for all CSPs.

3.6 Discussion

We have introduced ANYCSP, a method for neural combinatorial optimization to learn heuristics for any CSP through a purely data-driven process. The main contributions are two-fold: Firstly, we provide a generic graph representation and architecture that can directly process any CSP instance. Our experiments demonstrate that our generic approach outperforms specialized models for individual CSPs. Secondly, we demonstrate that reinforcement learning with standard policy gradient methods is sufficient to train GNNs as global search heuristics for combinatorial problems, despite the large action space and hard learning task. We also show that this training procedure is key to the strong empirical performance

of ANYCSP. Let us emphasize that these two contributions are loosely coupled. In particular, training for global search with policy gradients can be applied to architectures other than ours, such as the problem-specific models used in prior work.

One potential direction for future work includes widening the scope of the approach even further: Weighted CSPs are a natural extension of the CSP formalism and could be incorporated through node features and adjustments to the reward. Variables with real-valued domains may be another viable extension as policy gradient methods are also applicable to infinite continuous action spaces. In preliminary experiments, we also tested variants of ANYCSP for constraint *optimization* problems, where some optimization objective has to be maximized in addition to satisfying all constraints of a CSP. Maximum independent set is a simple example of a COP. In these experiments, we observed rather poor performance and no tested reward scheme was able to effectively balance both the constraints and the objective of a given instance. Whether ANYCSP can be effectively extended to COPs therefore remains an open problem.

Besides extending our approach to more problems, gaining deeper insights into the heuristics already learned by ANYCSP would be useful. Are they parallelized versions of classical local search algorithms or do they represent fundamentally different search strategies? How much structural information of the problem instance is stored in the model's latent states and how is this information translated into actions? Can the learned algorithms be distilled into smaller models for faster inference or even translated into simple CPU-friendly rules not based on compute-hungry linear algebra? Overall, many intriguing questions remain to be answered in future work.

Finally, let us emphasize that neural combinatorial optimization is still a relatively young field of research that continues to grow alongside the field of graph learning. The scope of algorithms that can be expressed and learned by GNNs widens steadily as advances are made toward more powerful architectures, improved scalability, and more effective training recipes. While the results shown here are promising, they should be viewed as early work toward integrating deep learning into the evolving landscape of combinatorial optimization, with much potential for further advances.

Graph Learning with 1D Convolutions and Random Walks

4.1 Introduction

One of our core research objectives is to add novel architectures to the methodological toolbox of graph learning. In this chapter, we present CRaWl (**C**NNs for **Ra**ndom **W**a**l**ks) (Tönshoff et al., 2023b) to explore alternative approaches for learning on graphs with neural networks. A key aim of this work is to design an architecture that is not based on the message passing paradigm without compromising on permutation equivariance, scalability or strong empirical performance.

While being fully compatible with MPGNNs in the sense that CRaWl also iteratively computes node embeddings and thus can potentially be integrated into an MPGNN architecture, CRaWl is based on a fundamentally different idea, and the features it extracts are provably different from those of MPGNNs. For a given input graph, CRaWl samples a set of random walks and extracts sequential features that fully describe the subgraphs visible within a sliding window over these walks. Each CRaWl-layer processes these features with standard 1D convolutions and the extracted information is pooled into the nodes to update latent node embeddings. Compared to other methods based on counting or sampling small subgraphs, for example, graphlet kernels (Shervashidze et al., 2009a), CRaWl's method of sampling small subgraphs in a sliding window on random walks has the advantage that even in sparse graphs it usually yields meaningful subgraph patterns. Its way of encoding the local graph structure and processing this encoding gives CRaWl an edge over subgraph GNNs (Frasca et al., 2022) and random-walk-based neural networks such as RAW-GNN (Jin et al., 2022).

We carry out a comprehensive theoretical analysis of the expressivity of CRaWl. Our main theorem shows that the expressivity of CRaWl is incomparable with MPGNNs in terms of distinguishing graphs: (1) there are graphs that can be distinguished by CRaWl, but not by MPGNNs, and (2) there are graphs that can be distinguished by MPGNNs, but not by CRaWl. Notably, the result can be extended to the full Weisfeiler-Leman hierarchy and hence to higher-order GNNs: for every k there are graphs that can be distinguished by CRaWl, but not by k-WL. In particular, this means that the result also applies to the variety

of GNN-extensions whose expressivity is situated within the Weisfeiler Leman hierarchy (e.g. Barceló et al., 2021, Cotta et al., 2021, Maron et al., 2019).

We further conduct a comprehensive experimental study, demonstrating that empirically CRaWl is on par with advanced MPGNN architectures and Graph Transformers on major graph learning benchmark datasets. We thereby validate empirically that designing an architecture with strong performance requires neither message passing nor characterizability in terms of the WL hierarchy.

4.2 Related Work

The primary baseline of CRaWl are standard MPGNN architectures such as GCN (Kipf and Welling, 2017), GIN (Xu et al., 2019), GAT (Veličković et al., 2017), GraphSage (Hamilton et al., 2017), and PNA (Corso et al., 2020) as well as Graph Transformers such as SAN (Kreuzer et al., 2021), SAT (Chen et al., 2022), Graph ViT (He et al., 2023), and GPS (Rampášek et al., 2022). We introduced both classes of architecture in depth in Section 2.1 and Section 2.4, respectively. Here we will provide an overview of the remaining related work that is connected to CRaWl either through common motivation, similar features, or shared architectural components.

Extensions of Message Passing The theoretical expressivity of MPGNNs is bounded by the 1-dimensional Weisfeiler-Leman algorithm (Theorem 2.1). A well-known class of extensions that address this limitation are higher-order GNNs (Morris et al., 2019), which are as expressive as higher-dimensional variants of the WL algorithm as discussed in Section 2.2.1.1. Beyond such higher-order GNNs, many other extensions of standard MPGNNs have been proposed, motivated by theoretical properties or empirical performance. In HIMP (Fey et al., 2020), the backbone of a molecular graph is extracted and then two GNNs are run in parallel on the backbone and the full molecule graph. This allows HIMP to capture structural features that are otherwise neglected. An approach with strong empirical performance is GINE+ (Brossard et al., 2020). It is based on GIN (Xu et al., 2019) and aggregates information from higher-order neighborhoods, allowing it to detect small substructures such as cycles. Beaini et al. (2021) proposed DGN, which incorporates directional awareness into message passing. Bodnar et al. (2021) propose CW Networks, which incorporate regular Cell Complexes into GNNs to construct architectures that are not less powerful than 3-WL.

Structural Encodings Structural encodings (SE) are an alternative approach to strengthening expressivity by adding pre-computed node and edge features that directly encode structural information. Random Walk Structural Encodings (RWSE) (Dwivedi et al., 2021) are a popular example that we already discussed in Section 2.3. Several other works propose different encodings. Explicit counts of fixed substructures such as cycles or small cliques have been added to the node and

edge features by Bouritsas et al. (2020) (GSN). Similarly, Sankar et al. (2017), Lee et al. (2019), and Peng et al. (2020) added the frequencies of motifs, i.e., common connected induced subgraphs, to improve the predictions of GNNs. Sankar et al. (2020) introduce motif-based regularization, a framework that improves multiple MPGNNs. In (Jiang et al., 2022) the authors introduce sparse motifs that help to reduce over-smoothing.

Convolutional Neural Networks A few previous approaches utilize conventional CNNs (not to be confused with *Graph* Convolutional Neural Networks) in the context of end-to-end graph learning. Patchy-SAN (Niepert et al., 2016) normalizes graphs in such a way that they can be interpreted by CNN layers. Zhang et al. (2018) proposed a pooling strategy based on sorting intermediate node embeddings and presented DGCNN which applies a 1D CNN to the sorted embeddings of a graph. Yuan and Ji (2021) used a 1D CNN layer and attention for neighborhood aggregation to compute node embeddings.

Random-Walk-Based GNNs Several architectures based on random walks have been proposed. Nikolentzos and Vazirgiannis (2020) study a differentiable version of the random walk kernel and integrate it into a GNN architecture that learns a number of small graphs against which the differentiable random walk kernel compares the nodes of the graph. In Geerts (2020b), the ℓ-walk MPGNN aggregates node features along random walks. RAW-GNN (Jin et al., 2022) and pathGCN (Eliasof et al., 2022) learn to aggregate information along random walks based on the walk distance. This aggregation allows RAW-GNN to tackle node predictions in a heterophilic domain. Similar distance-based aggregations are also possible in a multi-hop MPGNN proposed by Ma et al. (2020). Note that these approaches do not encode the structure traversed by the walk at all, instead, the methods focus solely on proximity. In contrast, CRaWl explicitly extracts and uses local structure. Ivanov and Burnaev (2018) proposed a structure-aware approach by encoding node identity along anonymous random walks to learn graph representations for SVMs. Wang et al. (2021) extend the concept of anonymous random walks to temporal graphs. They also use encodings of node identity as features which are processed by an RNN to solve temporal link prediction tasks. In a related approach, Yin et al. (2022) learn subgraph representations based on random walks sampled around a set of query vertices. Their features are based on histograms of walk distances and are processed by RNNs. They apply their approach to link prediction tasks and demonstrate performance superior to standard MPGNNs. The previous two methods share conceptual parallels with CRaWl, but there are several core differences: The features of CRaWl encode both the identity and adjacency between vertices (instead of just identity) to provide an explicit and complete encoding of subgraph structure. Furthermore, our approach does not just compute an embedding of a subgraph around query vertices. Instead, we use CRaWl to embed all vertices at once with a more parallelizable CNN in a manner that is compatible with message passing, Virtual Nodes, and other layers based on iteratively refining vertex embeddings. These differences enable

CRaWl to tackle tasks where local structural features are of high importance and a bottleneck of standard GNN architectures such as graph classification in the molecular domain.

Subgraph GNNs Through its encoding of the complete local structure around a random walk, CRaWl processes induced subgraphs, and in that regard is similar to Subgraph GNNs (Frasca et al., 2022) and Structure-Aware Transformers (SAT) (Chen et al., 2022). Prior work in this framework includes Nested Graph Neural Networks (Zhang and Li, 2021), Equivariant Subgraph Aggregation Networks (Bevilacqua et al., 2022), and "GNN As Kernel" (Zhao et al., 2022). The main difference to our model is the way local structure is made available to the model. Both Subgraph GNNs and SAT use a standard MPGNN to process subgraphs, therefore keeping some limitations although they are more expressive than 1-WL. In contrast, CRaWl defines binary encodings of the (fixed-size) subgraph structure which are then embedded by an MLP which in theory can encode every function on the compact input domain. On the other hand, our subgraph representation is not deterministic since it depends on the order in which the random walk traverses the subgraph, while always providing complete structural information. Also, the chosen subgraph is random, depending on the concrete random walk. Due to those differences, the expressivity of MPGNNs and CRaWl is incomparable as formally shown in Section 4.4.

Graph Kernels A different way to learn on graph data is to use similarity measures on graphs with graph kernels (Kriege et al., 2020) which are usually based on counting substructures in a given graph such as graphlets, label sequences, or subtrees. We discussed the connection between MPGNNs and the Weisfeiler-Leman graph kernel (Shervashidze et al., 2011) in Section 2.2.1.2. The graphlet kernel (Shervashidze et al., 2009b) is based on counting the occurrences of all k-node subgraphs up to some fixed size $k \in \mathbb{N}$. It is therefore conceptually related to CRaWl, which also encodes the distribution of local subgraphs in its features. These are high-level parallels, however, since the manner in which CRaWl samples and processes subgraphs is significantly different.

4.3 Method

CRaWl processes random walks with convolutional neural networks to extract information about local and global graph structure. For a given input graph, we initially sample a large set of (relatively long) random walks. For each walk, we construct a sequential feature representation that encodes the structure of all contiguous walk segments up to a predetermined size (walklets). These features are well suited for processing with a 1D CNN, which maps each walklet to an embedding. These walklet embeddings are then used to update latent node states. By repeating this process, we allow long-range information about the graph structure to flow along each random walk.

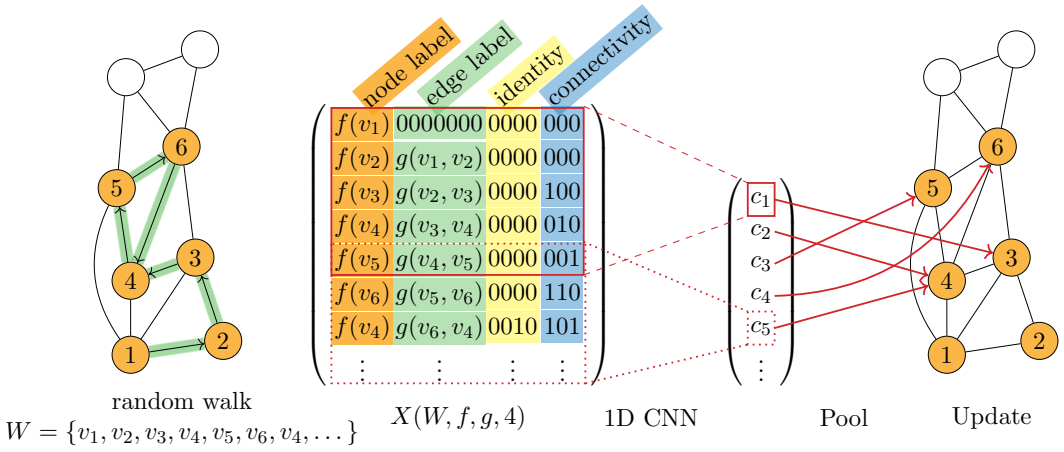


Figure 4.1: Example of the information flow in a CRAWL layer for one walk $W \in \Omega$. For W we compute the feature matrix X based on node embeddings f, edge embeddings g, and a window size of $s=4$. To this matrix, we apply a 1D CNN with receptive field $r=5$ and pool the output into the nodes to update their embeddings.

4.3.1 Random Walk Features

A walk W of length $\ell \in \mathbb{N}$ in a graph $G = (V, E)$ is a sequence of nodes $W = (v_0, \dots, v_\ell) \in V^{\ell+1}$ with $(v_{i-1}, v_i) \in E$ for all $i \in [\ell]$. A random walk in a graph is obtained by starting at some initial node $v_0 \in V$ and iteratively sampling the next node v_{i+1} randomly from the neighbors $N_G(v_i)$ of the current node v_i. In our main experiments, we choose the next node uniformly among $N_G(v_i) \setminus \{v_{i-1}\}$, i.e. following a non-backtracking uniform random walk strategy. In general, the applied random walk strategy is a hyperparameter of CRAWL and we perform an ablation study on their effectiveness in Section 4.5.5. There, we compare uniform random walks with and without backtracking. Another common random walk distribution are pq-Walks as used in node2vec (Grover and Leskovec, 2016).

We call contiguous segments $W[i:j] \coloneqq (w_i, \dots, w_j)$ of a walk $W = (w_0, \dots, w_\ell)$ *walklets*. The *center* of a walklet $(w_i, \dots, w_j)$ of even length $j-i$ is the node $w_{(i+j)/2}$. For each walklet $w = (w_i, \dots, w_j)$, we denote by $G[w]$ the subgraph induced by G on the set $\{w_i, \dots, w_j\}$. Note that $G[w]$ is connected as it contains all edges $w_k w_{k+1}$ for $i \leq k < j$, and it may contain additional edges. Also note that the w_k are not necessarily distinct, especially if backtracking uniform random walks are used.

For a walk W and a local window size s, we construct a matrix representation that encodes the structure of all walklets of size $s+1$ in W as well as the node and edge features occurring along the walk. Given a walk $W \in V^\ell$ of length $\ell - 1$ in a featured graph $G = (V, E, x, e)$, a d-dimensional node embedding $x : V \to \mathbb{R}^d$, a d'-dimensional edge embedding $e : E \to \mathbb{R}^{d'}$, and a local window size $s > 0$ we define the *walk feature matrix* $X(W, x, e, s) \in \mathbb{R}^{\ell \times d_X}$ with feature dimension

$d_X = d + d' + s + (s-1)$ as

$$X(W, x, e, s) = (x_W \, e_W \, I_W^s \, A_W^s).$$

For ease of notation, the first dimensions of each of the matrices x_W, e_W, I_W^s, A_W^s is indexed from 0 to $\ell - 1$. Here, the *node feature sequence* $x_W \in \mathbb{R}^{\ell \times d}$ and the *edge feature sequence* $e_W \in \mathbb{R}^{\ell \times d'}$ are defined as the concatenation of node and edge features, respectively. Formally,

$$(x_W)_{i,\text{-}} = x(v_i) \qquad \text{and} \qquad (e_W)_{i,\text{-}} = \begin{cases} \mathbf{0}, & \text{if } i = 0 \\ e(v_{i-1}, v_i), & \text{else.} \end{cases}$$

We define the *local identity relation* $I_W^s \in \{0,1\}^{\ell \times s}$ and the *local adjacency relation* $A_W^s \in \{0,1\}^{\ell \times (s-1)}$ as

$$(I_W^s)_{i,j} = \begin{cases} 1, & \text{if } i-j \geq 0 \wedge v_i = v_{i-j} \\ 0, & \text{else} \end{cases} \quad \text{and}$$

$$(A_W^s)_{i,j} = \begin{cases} 1, & \text{if } i-j \geq 1 \wedge v_i v_{i-j-1} \in E \\ 0, & \text{else.} \end{cases}$$

The matrices I_W^s and A_W^s are binary matrices that contain one row for every node v_i in the walk W. The bitstring for v_i in I_W^s encodes which of the s predecessors of v_i in W are identical to v_i, that is, where the random walk looped or backtracked. Micali and Zhu (2016) showed that from the distribution of random walks of bounded length with such identity features, one can reconstruct any graph locally (up to some radius r depending on the length of the walks). Similarly, A_W^s stores which of the predecessors are adjacent to v_i, indicating edges in the induced subgraph. The immediate predecessor v_{i-1} is always adjacent to v_i and is thus omitted in A_W^s. Note that we do not leverage edge labels of edges not on the walk, only the existence of such edges within the local window is encoded in A_W^s.

For any walklet $w = W[i : i+s]$, the restriction of the walk feature matrix to rows $i, \ldots, i+s$ contains a full description of the induced subgraph $G[w]$. Hence, when we apply a CNN with a receptive field of size at most $s+1$ to the walk feature matrix, the CNN filter has full access to the subgraph induced by the walklet within its scope. Figure 4.1 depicts an example of a walk feature matrix and its use in a CRAWL layer.

CRAWL initially samples m random walks $\Omega = \{W_1, \ldots, W_m\}$ of length ℓ from the input graph. By stacking the individual feature matrices for each walk, we get the *walk feature tensor* $X(\Omega, f, g, s) \in \mathbb{R}^{m \times \ell \times d_X}$ defined as

$$X(\Omega, f, g, s)_i = X(W_i, x, e, s).$$

This stack of walk feature matrices can then be processed in parallel by a CNN. The values for m and ℓ are not fixed hyperparameters of the model but instead can be chosen at runtime. By default, we start one walk at every node, i.e., $m = |V|$. We

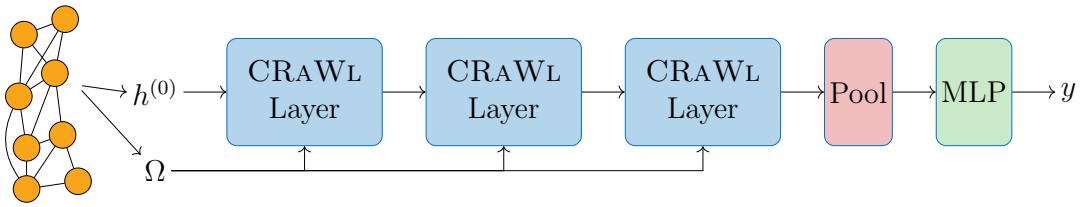


Figure 4.2: Architecture of a 3-layer graph-level CRaWl network. Each layer iteratively updates a vertex embedding by processing the features observed along a set of random walks Ω sampled from the input graph.

noted that reducing the number of walks during training can help against overfitting and of course, is a way to reduce the memory footprint which is important for large graphs. If we choose to use fewer random walks, we sample m starting nodes uniformly at random. We always make sure that $m\ell \gg |V|$, ensuring that with high probability each node appears on multiple paths. We typically choose $\ell \geq 50$ during training. During inference, we choose a larger ℓ of up to 150 which improves the predictions. The window size s has to be at least 6 for molecular datasets to capture organic rings. By default, we thus use $s = 8$.

4.3.2 Architecture

A CRaWl network iteratively updates latent embeddings for each node. Let $G = (V, E, x, e)$ be a graph with initial node and edge feature maps $x : V \to \mathbb{R}^{d_V}$ and $e : E \to \mathbb{R}^{d_E}$. The function $h^{(t)} : V \to \mathbb{R}^{d^{(t)}}$ stores the output of the t-th layer of CRaWl and the initial node features are stored in $h^{(0)} = x$. In principle, the output dimension $d^{(t)}$ is an independent hyperparameter for each layer. In practice, we use the same size d for the output node embeddings of all layers for simplicity.

The t-th layer takes as input the walk feature tensor $X^{(t)} = X(\Omega, h^{(t-1)}, e, s)$ using $h^{(t-1)}$ as its input node embedding and the graph's edge features e. This walk feature tensor is then processed by a convolutional network $\mathrm{CNN}^{(t)}$ based on 1D CNNs. The first dimension of $X^{(t)}$ of size m is viewed as the batch dimension. The convolutional filters move along the second dimension (and therefore along each walk) while the third dimension contains the feature channels. Each $\mathrm{CNN}^{(t)}$ module consists of three convolutional layers combined with ReLU activations and batch normalization, as described in Section 4.3.3. The stack of operations has a receptive field of $s+1$ and effectively applies an MLP to each subsection of this length in the walk feature matrices. The output of $\mathrm{CNN}^{(t)}$ is a tensor $C^{(t)} \in \mathbb{R}^{m \times (\ell - s) \times d}$. Note that the second dimension is $\ell - s$ instead of ℓ as no padding is used in the convolutions. Through its receptive field, the CNN operates on walklets of size $s+1$ and produces embeddings for those. We pool those embeddings into the nodes of the graph by averaging for each node $v \in V$ all embeddings of walklets centered at v. Let $\Omega = \{W_1, \ldots, W_m\}$ be a set of walks with $W_i = (v_{i,1}, \ldots, v_{i,\ell})$. Then $C^{(t)}_{i,j-s/2} \in \mathbb{R}^d$ is the embedding computed by $\mathrm{CNN}^{(t)}$ for the walklet $w = W_i[j - \frac{s}{2} : j + \frac{s}{2}]$ centered at $v_{i,j}$ and the pooling operation is

given by

$$p^{(t)}(v) = \underset{(i,j)\in \text{center}(\Omega,s,v)}{\text{mean}} C^{(t)}_{i,j-s/2}\,.$$

Here, $\text{center}(\Omega, s, v)$ is the set of walklets of length $s+1$ in Ω in which v occurs as center. We restrict s to be even such that the center is always well-defined. The output of the pooling step is a vector $p^{(t)}(v) \in \mathbb{R}^d$ for each v. This vector is then processed by a trainable MLP $\mathbf{U}^{(t)}$ with a single hidden layer of dimension $2d$ to compute the next latent vertex embedding

$$h^{(t)}(v) = \mathbf{U}^{(t)}\big(p^{(t)}(v)\big).$$

Figure 4.1 gives an overview over the elements of a CRAWL layer. Our architecture stacks multiple CRAWL layers into one end-to-end trainable neural network, as illustrated in Figure 4.2. After the final layer we apply batch normalization before performing a global readout step with either sum- or mean-pooling to obtain a graph-level latent embedding. Finally, a simple feedforward neural network is used to produce a graph-level output which can then be used in classification and regression tasks.

Since CRAWL layers are based on iteratively updating latent node embeddings, they are fully compatible with conventional message passing layers and related techniques such as Virtual Nodes (Gilmer et al., 2017, see Section 2.4). In our experiments, we use Virtual Nodes whenever this increases validation performance. Note that we view the Virtual Node as an intermediate readout step that is placed between CRAWL layers to allow for global communication between nodes. No additional node is actually added to the graph and, most importantly, the "virtual node" does not occur in the random walks sampled by CRAWL. Combining CRAWL with message passing layers is left as future work.

We implemented CRAWL in PyTorch (Paszke et al., 2019, Fey and Lenssen, 2019)[1]. Note that for a given graph we sample all m walks in parallel with a GPU-accelerated random walk generator. This process is fast enough to be done on-the-fly for each new batch instead of using precomputed random walks. Roughly 10% of the total runtime during training is spent on sampling walks.

4.3.3 Convolution Module

Here, we describe the architecture used for the 1D CNN network $\text{CNN}^{(t)}$ in each layer t. Let $\text{Conv1D}(d, d', k)$ be a standard 1D convolution module with input feature dimension d, output feature dimension d', kernel size k, and no bias. This module has $d \cdot d' \cdot k$ trainable parameters. The term scales poorly for larger hidden dimensions d, since the square of this dimension is scaled with an additional factor of k, which we typically set to 9 or more.

[1] `https://github.com/toenshoff/CRaWl`

To address this issue we leverage *Depthwise Separable Convolutions*, as suggested by (Chollet, 2017). This method is most commonly applied to 2D data in computer vision, but it can also be utilized for 1D convolutions. It decomposes one convolution with kernel size k into two convolutions: The first convolution is a standard 1D convolution with window size 1. The second convolution is a depthwise convolution with kernel size k, which convolves each channel individually and therefore only requires $k \cdot d'$ parameters. The second convolution is succeeded by batch normalization and a ReLU activation function. Note that there is no non-linearity between the two convolutions. These operations effectively simulate a standard convolution with kernel size k but require substantially less memory and runtime.

After the ReLU activation, we apply an additional (standard) convolution with window size 1, followed by another ReLU non-linearity. This final convolution increases the expressivity of our convolution module which could otherwise only learn linearly separable functions. This would limit its ability to distinguish the binary patterns that encode identity and adjacency. The full stack of operations effectively applies a 2-layer MLP to each sliding window position of the walk feature tensor. Overall, $\text{CNN}^{(t)}$ is composed of the following operations:

$$\text{Conv1D}(d, d', 1) \to \text{Conv1D}^{dw}(d', d', k) \to \text{BatchNorm} \to \text{ReLU} \to \text{Conv1D}(d', d', 1) \to \text{ReLU}$$

Here, Conv1D is a standard 1D convolution and Conv1D^{dw} is a depthwise convolution. The total number of parameters of one such module (without the affine transformation of the BatchNorm module) is equal to $dd' + kd + {d'}^2$.

4.3.4 Asymptotic Runtime

Let us compare the asymptotic runtime of CRaWl to that of expressive MPGNNs and Graph Transformers. Neglecting the hidden dimension d, the runtime of applying the CNN in each CRaWl layer is $\mathcal{O}(m \cdot \ell \cdot s^2)$. The window size s is squared since it determines both the size of the walk features but also the receptive field of the CNN. The walk feature tensor uses memory in $\mathcal{O}(m \cdot \ell \cdot (s + d))$. Our default choice for training is $m = |V|$ and we usually choose ℓ such that $|V| \cdot \ell > |E|$. Therefore, the required time exceeds that of a standard MPGNN layer running in $\mathcal{O}(|V| + |E|)$. Let us emphasize that this is a common drawback of architectures which are not bounded by 1-WL. For example, precomputing structural features with GSN (Bouritsas et al., 2020) has a worst-case runtime in $\mathcal{O}(|V|^k)$, where k is the size of the considered subgraphs. Similarly, higher-order k-GNNs (Morris et al., 2020a) have a runtime in $\mathcal{O}(|V|^k)$ while methods based on Graph Transformers (see Section 2.4.2) usually have a time and space requirement in $\mathcal{O}(|V|^2)$. While random node initialization (Abboud et al., 2021b, Sato et al., 2020) also strengthens the expressive power of MPGNNs beyond 1-WL, its empirical advantage is limited. We compare empirical runtime measurements in Table 4.5.

4.4 Expressivity

In this section, we report on theoretical results comparing the expressivity of CRaWl with that of standard MPGNNs. We prove that the expressivity of the two architectures is incomparable, confirming our intuition that CRaWl detects fundamentally different features of graphs than approaches based on message passing. This gives a strong theoretical argument for the use of CRaWl as an alternative graph learning architecture, ideally in combination with GNNs, to get the best of both worlds.

Intuitively, the additional strength of CRaWl has two different roots:

(i) CRaWl detects small subgraphs (of size determined by the window size hyperparameter s), whereas MPGNNs (at least in their basic form) cannot even detect triangles;

(ii) CRaWl detects long-range interactions (determined by the window size and path length hyperparameters), whereas MPGNNs are local with a locality radius determined by the number of layers (which is typically smaller).

While (ii) does play a role in practice, in our theoretical analysis we focus on (i). This has the additional benefit that our results are independent of the number of layers, making them applicable even to recurrent MPGNNs where the number of iterations can be chosen at runtime. Let us remark that (i) makes CRaWl similar to network analysis techniques based on motif detection (Alon, 2007) and graph kernels based on counting subgraphs, such as the graphlet kernel (Shervashidze et al., 2009b) (even though the implementation of these ideas is very different in CRaWl).

Conversely, the intuitive advantage MPGNNs have over CRaWl is that locally, they perform a breadth-first search, whereas CRaWl searches along independent walks. For example, MPGNNs immediately detect the degree of every node, and then on subsequent layers, the degree distributions of the neighbors, and their neighbors, et cetera. While in principle, CRaWl can also detect degrees by walks that are returning to the same node multiple times, it becomes increasingly unlikely and in fact impossible if the degree is larger than the walk length. Interestingly, large degrees are not the only features that MPGNNs detect, but CRaWl does not. We separate MPGNNs from CRaWl even on graphs of maximum degree 3.

As discussed in Section 2.2.1, the expressivity of MPGNNs corresponds exactly to that of the 1-dimensional Weisfeiler-Leman algorithm (1-WL), in the sense that two graphs are distinguished by 1-WL if and only if they can be distinguished by an MPGNN. It is also known that higher-dimensional versions of WL characterize the expressivity of higher-order MPGNNs (k-GNNs) (Morris et al., 2019, see Section 2.2.1.1). Our main theorem states that the expressivity of CRaWl and k-WL is incomparable.

Theorem 4.1.

(1) For every $k \geq 1$ there are graphs of maximum degree 3 that are distinguishable by CRaWl *with window size and walk length $O(k^2)$, but not by k-WL (and hence not by k-GNNs).*

(2) There are graphs of maximum degree 3 that are distinguishable by 1-WL (and hence by MPGNNs), but not by CRaWl *with window size and path length $O(n)$, where n is the number of vertices of the graph.*

We formally prove this theorem in Sections 4.4.1 and 4.4.2. To make the theorem precise, we first need to define what it actually means that CRaWl, as a randomized algorithm, *distinguishes* two graphs. To this end, we give a formal definition based on the total variation distance of the distribution of walk feature matrices that we observe in the graphs. Our proof of assertion (1) assumes that the CNN filters used to process the features within a window on a walk can be arbitrary multi-layer perceptrons and hence share their universality. Once this is settled, we can prove (1) by using the seminal Cai-Fürer-Immerman construction (Cai et al., 1992) providing families of graphs indistinguishable by the WL algorithm. The proof of (2) is based on an analysis of random walks on a long path based on the gambler's ruin problem. Before we give the proof, let us briefly comment on the necessary window size and simple extensions of the theorem.

In practice, k-GNNs of order at most $k = 3$ are feasible. Thus, the window size and path length $O(k^2)$ required by CRaWl models to exceed the expressivity of GNNs of that order (according to Assertion (1) of the theorem) is very moderate. It can also be shown that CRaWl with a window size polynomial in the size of the graphlets is strictly more expressive than graphlet kernels. This can be proved similarly to Assertion (1) of Theorem 4.1.

Let us remark that the expressivity of MPGNNs can be considerably strengthened by adding a random node initialization (Abboud et al., 2021b, Sato et al., 2020). The same can be done for CRaWl, but so far the need for such a strengthening (at the cost of a higher variance) did not arise. Besides expressivity, a fundamental aspect of machine learning on graphs is the permutation invariance of graph-level representations. So let us briefly comment on the invariance of CRaWl's computations. A CRaWl model represents a random variable mapping graphs into a feature space $\mathbb{R}^d$, and this random variable is invariant, which means that it only depends on the isomorphism type of the input graph. By independently sampling more walks we can reduce the variance of the random variable and converge to the invariant "true" function value. Note that this invariance of CRaWl is the same as for other graph learning architectures, in particular, MPGNNs with random node initialization (see Abboud et al., 2021b).

Notation Let us first recall the setting and introduce some additional notation. Throughout the proofs, we assume that the graphs are undirected, simple (that is, without self-loops and parallel edges) and unlabeled. All results can easily be

extended to (vertex and edge) labeled graphs.[2] In fact, the (harder) inexpressivity results only become stronger by restricting them to the subclass of unlabeled graphs. We further assume that graphs have no isolated nodes, which enables us to start a random walk from every node. This makes the setup cleaner and avoids tedious case distinctions, but again is no serious restriction. Formally, for a graph $G = (V(G), E(G))$ and a set $X \subseteq V(G)$, the *induced subgraph* $G[X]$ is the graph with node set X and edge set $\{vw \in E(G) \mid v, w \in X\}$. A walk of length ℓ in G is a sequence $W = (w_0, \dots, w_\ell) \in V(G)^{\ell+1}$ such that $w_{i-1}w_i \in E(G)$ for $1 \leq i \leq \ell$. The walk is *non-backtracking* if for $1 < i < \ell$ we have $w_{i+1} \neq w_{i-1}$ unless the degree of vertex w_i is 1.

Distinguishing Graphs Before we prove the theorem, let us precisely specify what it means that CRaWl distinguishes two graphs. Recall that CRaWl has three (walk related) hyperparameters:

(1) the *window size* s;

(2) the *walk length* ℓ;

(3) the *sample size* m.

Recall furthermore that with every walk $W = (w_0, \dots, w_\ell)$ we associate a *walk feature matrix* $X \in \mathbb{R}^{(\ell+1)\times(d+d'+s+(s-1))}$. For $0 \leq i \leq \ell$, the first d entries of the i-th row of X describe the current embedding of the node w_i, the next d' entries the embedding of the edge $w_{i-1}w_i$ (0 for $i = 0$), the following s entries are indicators for the equalities between w_i and the nodes w_{i-j} for $j = 1, \dots, s$ (1 if $w_i = w_{i-j}$, 0 if $i - j < 0$ or $w_i \neq w_{i-j}$), and the remaining $s - 1$ entries are indicators for the adjacencies between w_i and the nodes w_{i-j} for $j = 2, \dots, s$ (1 if w_i, w_{i-j} are adjacent in G, 0 if $i - j < 0$ or w_i, w_{i-j} are non-adjacent; note that w_i, w_{i-1} are always be adjacent because W is a walk in G). Since on unlabeled graphs initial node and edge embeddings are the same for all nodes and edges by definition, those embeddings cannot contribute to expressivity. Thus, we can safely ignore these embeddings and focus on the subgraph features encoded in the last $2s - 1$ columns. For simplicity, we regard X as an $(\ell + 1) \times (2s - 1)$ matrix with only these features in the following. We denote the entries of the matrix X by $X_{i,j}$ and the rows by $X_{i,-}$. So $X_{i,-} = (X_{i,1}, \dots, X_{i,2s-1}) \in \{0, 1\}^{2s-1}$. We denote the walk feature matrix of a walk W in a graph G by $X(G, W)$. The following observation formalizing the connection between walk-induced subgraphs and rows in the matrix X is immediate from the definitions.

Observation 1. For all walks $W = (w_1, \dots, w_\ell), W' = (w'_1, \dots, w'_\ell)$ in graphs G, G' with feature matrices $X \coloneqq X(G, W), X' \coloneqq X(G', W')$, we have:

(1) if $X_{i-j,-} = X'_{i-j,-}$ for $j = 0, \dots, s - 1$ then the mapping $w_{i-j} \mapsto w'_{i-j}$ for $j = 0, \dots, s$ is an isomorphism from the induced subgraph $G[\{w_{i-j} \mid j = 0, \dots, s\}]$ to the induced subgraph $G'[\{w'_{i-j} \mid j = 0, \dots, s\}]$;

[2] It is possible to simulate directed edges and parallel edges through edge labels and loops through node labels.

(2) if the mapping $w_{i-j} \mapsto w'_{i-j}$ for $j = 0, \dots, 2s-1$ is an isomorphism from the induced subgraph $G[\{w_{i-j} \mid j = 0, \dots, 2s-1\}]$ to the induced subgraph $G'[\{w'_{i-j} \mid j = 0, \dots, 2s-1\}]$, then $X_{i-j,-} = X'_{i-j,-}$ for $j = 0, \dots, s-1$.

The reason that we need to include the vertices $w_{i-2s+1}, \dots, w_{i-s}$ and $w'_{i-2s+1}, \dots, w'_{i-s}$ into the subgraphs in (2) is that row $X_{i-s+1,-}$ of the feature matrix records edges and equalities between w_{i-s+1} and $w_{i-2s+1}, \dots, w_{i-s}$.

For every graph G we denote the distribution of random walks on G starting from a node chosen uniformly at random by $\mathcal{W}(G)$ and $\mathcal{W}_{nb}(G)$ for the non-backtracking walks. We let $\mathcal{X}(G)$ and $\mathcal{X}_{nb}(G)$ be the push-forward distributions on $\{0,1\}^{(\ell+1)\times(2s-1)}$, that is, for every $X \in \{0,1\}^{(\ell+1)\times(2s-1)}$ we let

$$\Pr_{\mathcal{X}(G)}(X) = \Pr_{\mathcal{W}(G)}\big(\{W \mid X(G,W) = X\}\big)$$

A CRaWl run on G takes m samples from $\mathcal{X}(G)$. So to distinguish two graphs G, G', CRaWl must detect that the distributions $\mathcal{X}(G), \mathcal{X}(G')$ are distinct using m samples.

As a warm-up, let us prove the following simple result.

Theorem 4.2. *Let G be a cycle of length n and G' the disjoint union of two cycles of length $n/2$. Then G and G' cannot be distinguished by* CRaWl *with window size $s < n/2$ (for any choice of parameters ℓ and m).*

Proof. With a window size smaller than the length of the shortest cycle, the graph CRaWl sees in its window is always a path. Thus, for every walk W in either G or G' the feature matrix $X(W)$ only depends on the backtracking pattern of W. This means that $\mathcal{X}(G) = \mathcal{X}(G')$. □

It is worth noting that the graphs G, G' of Theorem 4.2 can be distinguished by 2-WL[3] (the 2-dimensional Weisfeiler-Leman algorithm), but not by 1-WL.

Proving that two graphs G, G' have identical feature-matrix distributions $\mathcal{X}(G) = \mathcal{X}(G')$ as in Theorem 4.2 is the ultimate way of proving that they are not distinguishable by CRaWl. Yet, for more interesting graphs, we rarely have identical feature-matrix distributions. However, if the distributions are sufficiently close, we will still not be able to distinguish them. This will be the case for Theorem 4.1 (2). To quantify closeness, we use the *total variation distance* of the distributions. Recall that the total variation distance between two probability distributions $\mathcal{D}, \mathcal{D}'$ on the same finite sample space Ω is

$$\operatorname{dist}_{TV}(\mathcal{D}, \mathcal{D}') \coloneqq \max_{S \subseteq \Omega} |\Pr_{\mathcal{D}}(S) - \Pr_{\mathcal{D}'}(S)|.$$

[3]Within this section we denote with k-WL the "folklore" k-WL algorithm studied by Cai et al. (1992). This was denoted with k-FWL in Section 2.2.1.1

It is known that the total variation distance is half the ℓ_1-distance between the distributions, that is,

$$\begin{aligned}\mathrm{dist}_{TV}(\mathcal{D}, \mathcal{D}') &= \frac{1}{2}\|\mathcal{D} - \mathcal{D}'\|_1 \\ &= \frac{1}{2}\sum_{\omega\in\Omega} |\Pr_{\mathcal{D}}(\{\omega\}) - \Pr_{\mathcal{D}'}(\{\omega\})|.\end{aligned}$$

Let $\varepsilon > 0$. We say that two graphs G, G' are *ε-indistinguishable* by CRaWl with window size s, walk length ℓ, and sample size m if

$$\mathrm{dist}_{TV}(\mathcal{X}(G), \mathcal{X}(G')) < \frac{\varepsilon}{m}. \tag{4.1}$$

The rationale behind this definition is that if $\mathrm{dist}_{TV}(\mathcal{X}(G), \mathcal{X}(G')) < \frac{\varepsilon}{m}$ then for every property of feature matrices that CRaWl may want to use to distinguish the graphs, the expected numbers of samples with this property that CRaWl sees in both graphs are close together (assuming ε is small).

Often, we want to make asymptotic statements, where we have two families of graphs $(G_n)_{n\geq 1}$ and $(G'_n)_{n\geq 1}$, typically of order $|G_n| = |G'_n| = \Theta(n)$, and classes S, L, M of functions, such as the class $O(\log n)$ of logarithmic functions or the class $n^{O(1)}$ of polynomial functions. We say that $(G_n)_{n\geq 1}$ and $(G'_n)_{n\geq 1}$ are *indistinguishable* by CRaWl with window size S, walk length L, and sample size M if for all $\varepsilon > 0$ and all $s \in S, \ell \in L, m \in M$ there is an n such that G_n, G'_n are ε-indistinguishable by CRaWl with window size $s(n)$, walk length $\ell(n)$, and sample size $m(n)$.

We could make similar definitions for distinguishability, but we omit them here because there is an asymmetry between distinguishability and indistinguishability. To understand this, note that our notion of indistinguishability only looks at the features that are extracted on the random walks and not on how these features are processed by the 1D CNNs. Clearly, this ignores an important aspect of the CRaWl architecture. However, we would like to argue that for the results presented here, this simplification is justified; if anything it makes the results stronger. Let us first look at the inexpressibility results (Theorem 4.1 (2) and Theorems 4.2 and 4.5). A prerequisite for CRaWl to distinguish two graphs is that the distributions of features observed on these graphs are sufficiently different, so our notion of indistinguishability that just refers to the distributions of features yields a stronger notion of indistinguishability (indistinguishability in the feature-distribution sense implies indistinguishability in CRaWl). This means that our indistinguishability results also apply to the actual CRaWl architecture.

For the distinguishability, the implication goes the other way and we need to be more careful. Our proof of Theorem 1 (1) (restated as Theorem 4.3 in the next section) does take this into account (see the discussion immediately after the statement of Theorem 4.3).

4.4.1 Proof of Theorem 4.1 (1)

Here is a precise quantitative version of the first part of Theorem 4.1.

Theorem 4.3. *For all $k \geq 1$ there are graphs G_k, H_k of order $\mathcal{O}(k)$ such that* CRaWl *with window size $s = \mathcal{O}(k^2)$, walk length $\ell = \mathcal{O}(k^2)$, and sample size m distinguishes G_k and H_k with probability at least $1 - (\frac{1}{2})^m$ while k-WL (and hence k-GNNs) fails to distinguish G_k and H_k.*

To prove the theorem we need to be careful with our theoretical model of distinguishability in CRaWl. It turns out that all we ever need to consider in the proof is the features observed within a single window of CRaWl. These features can be processed at once by the filter of the CNN. We assume that these filters can be arbitrary multilayer perceptrons (MLP) and hence share the universality property of MLPs that have at least one hidden layer: Any continuous function on the compact domain $[0, 1]^d$ (where d is the number of all features in the feature matrix X within a single window) can be approximated to arbitrary precision. While this is a strong assumption, it is common in expressivity results, e.g. (Xu et al., 2019).

In order to find pairs of graphs that k-WL is unable to distinguish, we do not need to know any details about the Weisfeiler-Leman algorithm (the interested reader is referred to (Grohe, 2021b, Kiefer, 2020)). Instead, we can use the following well-known inexpressibility result as a black box.

Theorem 4.4 (Cai et al. (1992))**.** *For all $k \geq 1$ there are graphs G_k, H_k such that $|G_k| = |H_k| = O(k)$, the graphs G_k and H_k are connected and 3-regular, and k-WL cannot distinguish G_k and H_k.*

Proof of Theorem 4.3. We fix $k \geq 1$. Let G_k, H_k be non-isomorphic graphs of order $O(k)$ that are connected, 3-regular, and not distinguishable by k-WL. Such graphs exist by Theorem 4.4.

Let $n := |G_k| = |H_k|$, and note that the number of edges of G_k and H_k is $(3/2)n$, because they are 3-regular. It is a well-known fact that the *cover time* of a connected graph of order n with m edges, that is, the expected time it takes a random walk starting from a random node to visit all nodes of the graph is bounded from above by $4nm$ (Aleliunas et al., 1979). Thus, in our graphs G_k, H_k the cover time is $6n^2$. Therefore, by Markov's inequality, a walk of length $12n^2$ visits all nodes with probability at least $1/2$.

We set the walk length ℓ and the window size s to $12n^2$. Since n is $O(k)$, this is $O(k^2)$.

Consider a single walk $W = (w_0, \ldots, w_\ell)$ of length ℓ in a graph $F \in \{G_k, H_k\}$, and assume that it visits all nodes, which happens with probability at least $1/2$ in either graph. Then the feature matrix $X = X(F, W)$, which we see at once in our window of size $s = \ell$, gives us a representation of the graph F. Of course, this representation depends on the specific order in which the walk traverses the

graph. What this means is that if for two graphs F, F' and walks W, W' visiting all nodes of their respective graphs we have $X(F, W) = X(F', W')$, then the graphs F, F' are isomorphic. This follows immediately from Observation 1. Note that the converse does not hold.

Since the above will rarely happen, we now define three sets of feature matrices that we may see:

$$\begin{aligned} R_G &\coloneqq \{X(G_k, W) \mid W \text{ walk on } G_k \text{ visiting every vertex}\}, \\ R_H &\coloneqq \{X(H_k, W) \mid W \text{ walk on } H_k \text{ visiting every vertex}\}, \\ S &\coloneqq \{X(F, W) \mid F \in \{G_k, H_k\}, W \text{ walk on } F \text{ not visiting every vertex}\}. \end{aligned}$$

Observe that because we chose the walk length ℓ and the window size s large enough we have:

- for every walk on a graph $F \in \{G_k, H_k\}$, the feature matrix X that we see belongs to one of the three sets, and with probability at least $1/2$ it belongs to R_G or R_H;
- the three sets R_G, R_H, S are mutually disjoint. For the non-isomorphic graphs G_k, H_k we can never have $X(G_k, W) = X(H_k, W')$ such that a walk visiting all nodes will end up in the corresponding set R_G or R_H. Through the equality features one can count the number of nodes visited in W, distinguishing S from both R_G and R_H.

Thus, if we sample a walk from G_k, with probability at least $1/2$ we see a feature matrix in R_G, and if we sample from H_k, we can never see a matrix in R_G. Let $f : [0,1]^{(\ell+1)\times(2s-1)} \to [0,1]$ be a continuous function that maps every feature matrix $X \in \{0,1\}^{(\ell+1)\times(2s-1)}$ to 1 if $X \in R_G$ and to 0 if $X \in R_H \cup S$.

Let us now run CRaWl on a graph $F \in \{G_k, H_k\}$. Since the window size s equals the walk length ℓ, CRaWl sees the full feature matrix $X(F, W)$ in a single window. Thus, by the universality of the CNN filters, it can approximate the function f that checks whether a given feature matrix belongs to R_G up to an additive error of $1/3$, allowing us to simply use f as we can distinguish both cases independent of the error due to the approximation. We know that if we run CRaWl on $F = G_k$, with probability at least $1/2$ it holds that $X(F, W) \in R_G$ and thus $f(X(F, W)) = 1$. Otherwise, $f(X(F, W)) = 0$ with probability 1 as the three sets are disjoint. Hence, a CRaWl run based on a single random walk W distinguishes the two graphs G_k and H_k with probability at least $1/2$. By exploiting that the error is one-sided and repeating the above argument m times, one can distinguish the two graphs with probability $1 - (\frac{1}{2})^m$ using m independent random walks. By simply relying on the universality of MLPs in this final step, the size of the CNN filter is exponential in k. This is not a limitation in this context, since k-WL also scales exponentially in k.

□

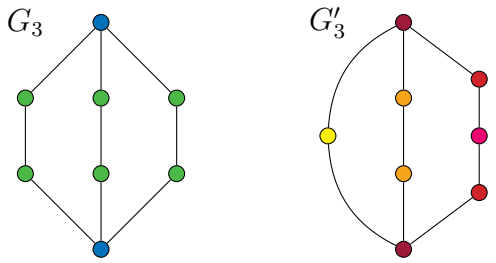


Figure 4.3: The graphs G_3 and G'_3 in the proof of Theorem 4.5 with their stable coloring computed by 1-WL

We note that while the proof assumed that the random walk covered the whole graph and in addition the window size is chosen to encompass the complete walk, this is neither feasible nor necessary in practice. Instead, the theorem states that using a window size and path length of $O(k^2)$, CRaWl can detect differences in graphs which are inaccessible to k-GNNs. This is applicable to graphs of arbitrary size n where the path of fixed size $O(k^2)$ (typically $k \leq 3$ as k-GNNs scale in n^k) is no longer able to cover the whole graph. In contrast to the theorem where a single walk (covering the whole graph) could suffice to tell apart two graphs, one needs multiple walks to distinguish the difference in the distributions of walks on real-world graphs as outlined earlier in this section.

4.4.2 Proof of Theorem 4.1 (2)

For the results so far, it has not mattered if we allowed backtracking or not. Here, it makes a big difference. For the non-backtracking version, we obtain a stronger result with an easier proof. The following theorem is a precise quantitative statement of Theorem 4.1 (2).

Theorem 4.5. *There are families $(G_n)_{n\geq 1}$, $(G'_n)_{n\geq 1}$ of graphs of order $|G_n| = |G'_n| = 3n - 1$ with the following properties:*

(1) For all $n \geq 1$, 1-WL distinguishes G_n and G'_n.

(2) $(G_n)_{n\geq 1}$, $(G'_n)_{n\geq 1}$ are indistinguishable by the non-backtracking version of CRaWl *with window size $s(n) = o(n)$ (regardless of the walk length and sample size).*

(3) $(G_n)_{n\geq 1}$ $(G'_n)_{n\geq 1}$ are indistinguishable by CRaWl *with walk length $\ell(n) = O(n)$, and samples size $m(n) = n^{O(1)}$ (regardless of the window size).*

Proof. The graphs G_n and G'_n both consist of three internally disjoint paths with the same end nodes x and y. In G_n the lengths of all three paths is n. In G'_n, the length of the paths is $n - 1, n, n + 1$ (see Figure 4.3).

It is easy to see that 1-WL distinguishes the two graphs.

To prove assertion (2), let $s := 2n - 3$ be the window size. Then the length of the shortest cycle in G_n, G'_n is $s + 2$. Now consider a non-backtracking walk $W = (w_1, \ldots, w_\ell)$ in either G_n or G'_n (of arbitrary length ℓ). Intuitively, CRaWl only sees a simple path within its window size independent of which graph W is sampled from and can thus not distinguish G_n and G'_n. Formally, for all i and j with $i - s \leq j < i$ we have $w_i \neq w_j$, and unless $j = i - 1$, there is no edge between w_i and w_j. Thus, $X(W) = X(W')$ for all walks W' of the same length ℓ, and since it does not matter which of the two graphs G_n, G'_n the walks are from, it follows that $\mathcal{X}_{nb}(G_n) = \mathcal{X}_{nb}(G'_n)$.

To prove assertion (3) we use the fact that random walks of length $O(n)$ are very unlikely to traverse a path of length at least $n - 1$ from x to y. It is well known that the expected traversal time is $\Theta(n^2)$ (this follows from the analysis of the gambler's ruin problem). However, this does not suffice for us. We need to bound the probability that a walk of length $O(n)$ is a traversal. Using a standard, Chernoff type tail bound, it is straightforward to prove that for every constant $c \geq 0$ there is a constant $d \geq 1$ such that the probability that a random walk of length cn in either G_n or G'_n visits both x and y is at most $\exp(-n/d)$. As only walks visiting both x and y can differentiate between the two graphs, this gives us an upper bound of $\exp(-n/d)$ for the total variation distance between $\mathcal{X}(G_n)$ and $\mathcal{X}(G'_n)$. □

Let us note that a bound on the walk length in assertion (3) of the previous theorem is necessary because the backtracking version of CRaWl with sufficiently long paths does seem to have the ability to distinguish the graphs G_n, G'_n even with constant size windows. The intuitive argument is as follows: we first observe that, by going back and forth between a node and all its neighbors within its window, CRaWl can distinguish the two degree-3 nodes x, y from the remaining degree-2 nodes. Thus, the feature matrix reflects traversal times between degree-3 nodes, and the distribution of traversal times is different in G_n and G'_n. With sufficiently many samples, CRaWl can detect this. We leave it as future research to work out the quantitative details of this argument.

4.5 Experiments

We evaluate CRaWl on a range of standard graph learning datasets (Dwivedi et al., 2020, Hu et al., 2020c, Dwivedi et al., 2022, Morris et al., 2020b). Our goal is to understand how CRaWl performs on real-world and synthetic learning tasks and how its main hyperparameters affect the results. Our main baselines are numerous MPGNN and Graph Transformer architectures that have been proposed in recent years (see Section 4.2). We report values as provided in the literature and official leaderboards. For a fair and direct comparison, we exclude results of ensemble methods and models pre-trained on additional data.

4.5.1 Benchmark Datasets

From the OGB project (Hu et al., 2020c), we use the molecular property prediction dataset MOLPCBA with more than 400k molecules. Each of its 128 binary targets states whether a molecule is active towards a particular bioassay. The dataset is adapted from MoleculeNet (Wu et al., 2018) and represents molecules as graphs of atoms. It contains multi-dimensional node and edge features that encode information such as atomic number and chirality. Additionally, it provides a train/val/test split that separates structurally different types of molecules for a more realistic experimental setting. On MOLPCBA, the performance is measured in terms of the average precision (AP). Further, we evaluate on four datasets from Dwivedi et al. (2020). The first dataset ZINC is a molecular regression dataset with 12K molecules. The aim is to predict the *constrained solubility*, a chemical property that depends strongly on the distribution of cycles on the molecular structure. The node label is the atomic number and the edge labels specify the bond type. The datasets CIFAR10 and MNIST are graph classification datasets derived from corresponding image classification tasks and contain 60k and 70k graphs, respectively. The original images are modeled as networks of super-pixels. Both datasets are 10-class classification problems. We conduct additional experiments on the Long-Range Graph Benchmark (LRGB) (Dwivedi et al., 2022), which we discuss in more depth in Section 5.1. We use three LRGB datasets: PascalVOC-SP, Peptides-Func, and Peptides-Struct. To apply CRaWl to the node-level task of PascalVOC-SP we omit the pooling step and directly pass each vertex embedding into an MLP classifier.

Experimental Setup We adopt the training procedure specified by Dwivedi et al. (2020). In particular, the learning rate is initialized as 10^{-3} and decays with a factor of 0.5 if the performance on the validation set stagnates for 10 epochs. The training stops once the learning rate falls below 10^{-6}. Dwivedi et al. (2020) also specify that networks need to stay within parameter budgets of either 100k or 500k parameters. This ensures a fairer comparison between different methods. For MNIST, and CIFAR10 we train CRaWl models with the smaller budget of 100k since more baseline results are available in the literature. For MOLPCBA, we train for 60 epochs and decay the learning rate once with a factor of 0.1 after epoch 50. On the LRGB datasets PascalVOC-SP, Peptides-Func, and Peptides-Struct we use the 500k parameter budget and found it helpful to switch to a cosine annealing schedule for the learning rate. For all datasets we use a walk length of $\ell = 50$ during training. For evaluation, we increase this number to $\ell = 150$, except for MOLPCBA where we use $\ell = 100$ for efficiency. The window size s was chosen to be 8 for all but the LRGB datasets where we increased it to 16. All hyperparameters and the exact number of trainable parameters are provided in Table B.1 in the appendix. We observed that our default setting of $\ell = 50$ and $s = 8$ to be a robust initial setting across all datasets. When we observed overfitting, decreasing the number of walks turned out to be a simple and effective measure to improve performance. In molecular datasets a window size of at least 6 is required to detect rings and in our ablation study in Section 4.5.5 we observe a clear performance drop on the

	Method	ZINC Test MAE	MNIST Test Acc (%)	CIFAR10 Test Acc (%)	MOLPCBA Test AP
MPGNN	GIN	0.526 ± 0.051	96.485 ± 0.252	55.255 ± 1.527	0.2703 ± 0.0023
	GraphSage	0.398 ± 0.002	97.312 ± 0.097	65.767 ± 0.308	-
	GAT	0.384 ± 0.007	95.535 ± 0.205	64.223 ± 0.455	-
	GCN	0.367 ± 0.011	90.705 ± 0.218	55.710 ± 0.381	0.2483 ± 0.0037
	GatedGCN	0.214 ± 0.006	97.340 ± 0.143	67.312 ± 0.311	-
	PNA	0.142 ± 0.010	97.940 ± 0.120	70.350 ± 0.630	0.2838 ± 0.0035
Other	3WLGNN	0.303 ± 0.068	95.075 ± 0.961	59.175 ± 1.593	-
	GINE+	-	-	-	**0.2917 ± 0.0015**
	DGN	0.168 ± 0.003	-	**72.840 ± 0.420**	-
	HIMP	0.151 ± 0.006	-	-	-
	GSN	0.108 ± 0.018	-	-	-
	SAT	0.094 ± 0.008	-	-	-
	SAN	0.139 ± 0.006	-	-	0.2765 ± 0.0042
	CIN	**0.079 ± 0.006**	-	-	-
	Nested GIN	-	-	-	0.2832 ± 0.0041
	GIN-AK+	0.080 ± 0.001	-	72.190 ± 0.130	**0.2930 ± 0.0044**
	GPS	**0.070 ± 0.004**	**98.051 ± 0.126**	**72.298 ± 0.356**	0.2907 ± 0.0028
	GRIT	**0.059 ± 0.002**	**98.108 ± 0.111**	**76.468 ± 0.881**	-
Our	CRaWl	0.085 ± 0.004	**97.944 ± 0.050**	69.013 ± 0.259	**0.2986 ± 0.0025**

Table 4.1: Performance achieved on ZINC, MNIST, CIFAR10, and MOLPCBA. Baseline results are organized into standard MPGNNs and "Others", where the latter includes higher-order GNNs, methods with structural encodings, and Graph Transformers. We highlight the **first**, **second** and **third** best result, respectively.

molecular dataset ZINC when reducing s below 6. For each dataset we report the mean and standard deviation across several models trained with different random seeds. We follow the standardized procedure for each dataset and average over 10 models for MOLPCBA, 5 models for ZINC, CIFAR10, and MNIST and 4 models for PascalVOC-SP, Peptides-Func, and Peptides-Struct. During inference, the output of each model depends on the sampled random walks. Thus, each model's performance is given by the average performance over 10 evaluations based on different random walks.

Results Table 4.1 provides results on ZINC, MNIST, CIFAR10, and MOLPCBA. On the ZINC dataset, CRaWl achieves an MAE of 0.085. This is approximately a 40% improvement over the current best standard MPGNN (PNA) on ZINC. The result is on par (within standard deviation) of Cellular GNNs (CIN) (Bodnar et al., 2021) and "GNN as Kernel" (GIN-AK+) (Zhao et al., 2022), which are among the strongest approaches on ZINC. On MNIST and CIFAR10, CRaWl is on par with the strongest MPGNN baselines, such as PNA (Corso et al., 2020). On MOLPCBA, CRaWl yields state-of-the-art results. We note that the baselines include architectures strictly stronger than 1-WL, such as Nested GIN (Zhang and

Method	PascalVOC-SP Test F1	Peptides-Func Test AP	Peptides-Struct Test MAE
GCN	0.2078 ± 0.0031	0.6860 ± 0.0050	**0.2460 ± 0.0007**
GINE	0.2718 ± 0.0054	0.6621 ± 0.0067	**0.2473 ± 0.0017**
GatedGCN	**0.3880 ± 0.0040**	0.6765 ± 0.0047	0.2477 ± 0.0009
GPS	**0.4440 ± 0.0065**	0.6534 ± 0.0091	0.2509 ± 0.0014
DRew	0.3314 ± 0.0024*	**0.7150 ± 0.0044**	0.2536 ± 0.0015
Exphormer	0.3960 ± 0.0027*	0.6527 ± 0.0043	0.2481 ± 0.0007
GRIT	-	**0.6988 ± 0.0082**	**0.2460 ± 0.0012**
Graph ViT	-	0.6942 ± 0.0075	**0.2449 ± 0.0016**
G-MLPMixer	-	0.6921 ± 0.0054	0.2475 ± 0.0015
CRaWl	**0.4588 ± 0.0079**	**0.7074 ± 0.0032**	0.2506 ± 0.0022

Table 4.2: Performance on the PascalVOC-SP, Peptides-Func, and Peptides-Struct datasets from the Long-Range Graph Benchmark. We highlight the **first**, **second** and **third** best result, respectively. * indicates that results obtained with unnormalized features.

Li, 2021) and 3WLGNN (Maron et al., 2019). CRaWl yields better results, indicating that the WL-incomparability does not keep our method from outperforming higher-order MPGNNs.

Table 4.2 provides the results obtained for the LRGB datasets. Note that the baseline results for MPGNNs and GPS are the improved reevaluated results from Section 5.1. CRaWl achieves state-of-the-art performance on PascalVOC-SP outperforming all baselines. On Peptides-Func the results are competitive with state-of-the-art baselines. On Peptides-Struct the performance of CRaWl matches that of GPS, although neither outperforms simpler baselines such as GCN. Note that the relative margins on this dataset are small across all baselines.

Overall, CRaWl performs very well on a variety of datasets across several domains and usually yields significantly better results than standard MPGNNs. It is also highly competitive with architectures that are more expressive than 1-WL, such as Graph Transformers with structural encodings and higher-order GNNs.

4.5.2 Synthetic Data

We conduct additional experiments on the CSL dataset (Dwivedi et al., 2020), which is a synthetic dataset consisting of 4-regular *Cyclic Skip Link* graphs (Murphy et al., 2019). The task is to classify the graphs by their isomorphism class. Figure 4.4 provides an example of two skip-link graphs. Since all such graphs are 4-regular this task is unsolvable for standard MPGNNs when no node features are provided.

Experimental Setup The evaluation on CSL is based on 5-fold cross-validation. We use the 5-fold split Dwivedi et al. (2020) provided in their repository. In each

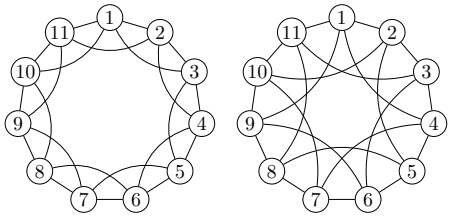


Figure 4.4: Two cyclic skip-link graphs with 11 nodes and a skip distance of 2 and 3 respectively.

Method	CSL Test Acc (%)	CSL+Lap Test Acc (%)
GIN	≤ 10.0	99.333 ± 1.333
GraphSage	≤ 10.0	99.933 ± 0.467
GAT	≤ 10.0	99.933 ± 0.467
GCN	≤ 10.0	**100.000 ± 0.000**
3WLGNN	95.700 ± 14.850	30.533 ± 9.863
GatedGCN	≤ 10.0	99.600 ± 1.083
CRaWl	**100.000 ± 0.000**	**100.000 ± 0.000**

Table 4.3: Test accuracy achieved on CSL with and without Laplacian eigenvectors (Dwivedi et al., 2020).

training run, three folds are used for training and one is used for validation and model selection. After training, the remaining fold is used for testing.

Results The results on CSL are reported in Table 4.3. We consider two variants of CSL, the pure task and an easier variant in which LapPE node features (see Section 2.3) are additionally provided. Already on the pure task, CRaWl achieves an accuracy of 100% as none of the 5 CRaWl models misclassified a single graph in the test folds. 3WLGNN is also theoretically capable of solving the pure task and achieves almost 100% accuracy, while MPGNNs cannot distinguish the pure 4-regular CSL graphs and achieve at most 10% accuracy. With Laplacian features that essentially encode the solution, all approaches (except 3WLGNN) achieve results close to 100%. Thus, the CSL dataset showcases the high theoretical expressivity of CRaWl.

4.5.3 Counting Substructures

We conduct additional experiments on the task of counting various subgraphs in synthetic graphs. To this end, we utilize a benchmark dataset proposed by Chen et al. (2020b). This dataset constitutes a graph-level regression problem that aims to infer how often each of the following subgraphs occurs in a given graph: Triangles, Tailed Triangles, 3-Stars, and 4-Cycles. The same dataset was used by Zhao et al. (2022) to evaluate GNN-AK, a state-of-the-art subgraph GNN. This experiment therefore offers a fine-grained comparison between both approaches.

We train CRaWl with 6 layers, hidden dimension 256, and window size 8 for 500 epochs and average the results over 4 models trained with different random seeds. Table 4.4 provides the test results. We report the MAE for each target subgraph. The baseline results were obtained from Zhao et al. (2022). Note that standard deviations were not reported.

As expected, both GNN-AK and CRaWl yield significantly better results on subgraph counting than simple MPGNNs like GCN and GIN. CRaWl yields the best results when counting stars, while GNN-AK achieves the lowest MAE on

Method	Triangle	Tailed Triangle	3-Star	4-Cycle
GCN	0.4186	0.3248	0.1798	0.2822
GIN	0.3569	0.2373	0.0224	0.2185
PNA	0.3532	0.2648	0.1278	0.2430
GCN-AK+	0.0137	0.0134	0.0174	0.0174
GIN-AK+	0.0137	**0.0112**	0.0150	0.0150
PNA-AK+	**0.0118**	0.0138	0.0166	**0.0132**
CRaWl	0.0208 ± 0.0022	0.0197 ± 0.0019	**0.0095 ± 0.0015**	0.0322 ± 0.0013

Table 4.4: Test MAE for various subgraph counting tasks.

Dataset	CRaWl	GCN	GatedGCN	Transformer	SAN	GPS	GIN-AK+
ZINC	10.0s	12.8s	10.7s	27.78s	106.0s	14.0s	9.4s
CIFAR10	265.0	109.7s	154.2s	-	-	62.0s	241.1s
MOLPCBA	458.4s	-	-	-	883.0s	-	-
PASCALVOC-SP	47.8s	8.8s	12.0s	13.0s	179.0s	17.0s	-
PEPTIDES-FUNC	18.5s	3.0s	3.3s	5.8s	49.1s	6.0s	-
PEPTIDES-STRUCT	18.4s	2.6s	3.3s	5.9s	49.7s	-	-

Table 4.5: Average time per epoch for CRaWl. The reported times are averaged over a training run and include the time used to perform a validation run after each training epoch. We also provide reported training times for several baselines as reported in the literature (Dwivedi et al., 2020, 2022, Zhao et al., 2022).

triangles and 4-cycles. Therefore, CRaWl and subgraph GNNs like GNN-AK seem to have different strengths and weaknesses. This overall result aligns with our main experiment in Table 4.1, where CRaWl outperforms subgraph GNNs on molecular data while GIN-AK+ yields better results on vision datasets.

4.5.4 Runtime

Table 4.5 provides runtime per epoch observed during training. All experiments were run on a machine with 64GB RAM, an Intel Xeon 8160 CPU, and an Nvidia Tesla V100 GPU with 16GB GPU memory. Note that absolute training times for the baselines are sparsely reported and depend on implementation details and hardware. We report these values as a rough guideline of how long the training of standard MPGNNs and stronger architectures usually takes. From the table we observe that the runtime of CRaWl does not differ much from common MPGNNs. While simpler architectures like GCN can run faster on larger datasets, CRaWl has no significant runtime disadvantage when compared to more advanced architectures like Graph Transformers or subgraph GNNs.

Features	Walks	ZINC (MAE)	MOLPCBA (AP)	CSL (Acc)
x+e	UN	0.19768 ± 0.01159	0.28364 ± 0.00201	0.06000 ± 0.04422
x+e	NB	0.15475 ± 0.00350	0.29613 ± 0.00209	0.06000 ± 0.04422
x+e+A	UN	0.10039 ± 0.00514	-	0.97467 ± 0.02587
x+e+A	NB	0.08656 ± 0.00310	-	0.99933 ± 0.00133
x+e+I	UN	0.10940 ± 0.00698	-	0.70733 ± 0.07658
x+e+I	NB	0.09345 ± 0.00219	-	0.97133 ± 0.00859
x+e+I+A	UN	0.09368 ± 0.00232	0.28522 ± 0.00317	0.96267 ± 0.02037
x+e+I+A	NB	0.08456 ± 0.00352	0.29863 ± 0.00249	1.00000 ± 0.00000

Table 4.6: Results of our ablation study: Node features x, edge features e, adjacency encoding A, and identity encoding I. Walk strategies: no-backtrack (NB) and uniform (UN).

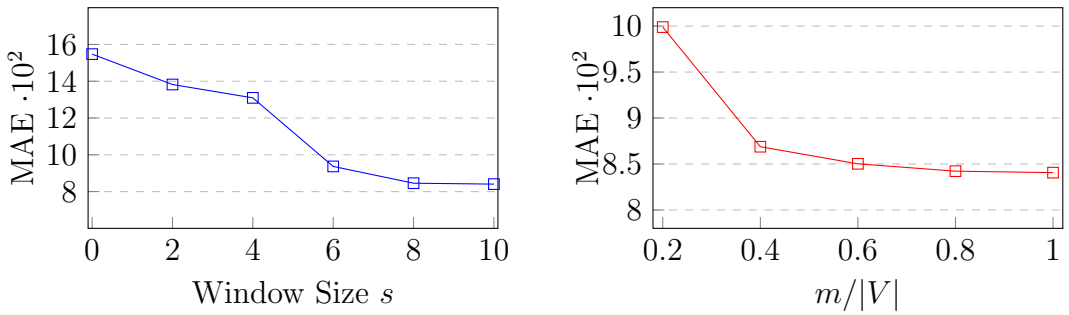
4.5.5 Ablation Study

We perform an ablation study to understand how the key aspects of CRaWl influence the empirical performance. We aim to answer three main questions:

- How useful are the identity and adjacency features we construct for the walks?
- How do different strategies for sampling random walks impact the performance?
- How do window size s and the number of walks m influence the performance?

Here, we use the ZINC, MOLPCBA, and CSL datasets to answer these questions empirically. We trained multiple versions of CRaWl with varying amounts of structural features used in the walk feature matrices. The simplest version only uses the sequences of node and edge features without any structural information. For ZINC and CSL, we also train intermediate versions using either the identity or the adjacency encoding, but not both. We omit these for MOLPCBA to save computational resources. Finally, we measure the performance of the standard CRaWl architecture, where both encodings are incorporated into the walk feature matrices. For each version, we compute the performance with both walk strategies. On each dataset, the experimental setup and hyperparameters are identical to those used in the previous experiments on both datasets. In particular, we train five models with different seeds and provide the average performance as well as the standard deviation across models. Note that we repeat the experiment independently for each walk strategy. Switching walk strategies between training and evaluation yields poor results.

Table 4.6 reports the performance of each studied version of CRaWl. On ZINC, the networks without any structural encoding yield the worst predictions. Adding either the adjacency or the identity encoding improves the results substantially. The



(a) Test MAE for different numbers window sizes s.

(b) Test MAE for different numbers of walks m.

Figure 4.5: Ablation study on the ZINC dataset. We study the influence of s and m on the test MAE.

best results are obtained when both encodings are utilized and non-backtracking walks are used. On MOLPCBA, the best performance is also obtained with full structural encodings. However, the improvement over the version without the encodings is only marginal. Again, non-backtracking walks perform significantly better than uniform walks. On CSL, the only version to achieve a perfect accuracy of 100% is the one with all structural encodings and non-backtracking walks. Note that the version without any encodings can only guess on CSL since this dataset has no node features (we are not using the LapPE features in this experiment).

Overall, the structural encodings of the walk feature matrices yield a measurable performance increase on all three datasets. However, the margin of the improvement varies significantly and depends on the specific dataset. For some tasks such as MOLPCBA, CRaWl yields highly competitive results even when only the sequences of node and edge features are considered in the walk feature matrices.

Finally, the non-backtracking walks consistently outperform the uniform walks. This could be attributed to their ability to traverse sparse substructures quickly. On sparse graphs, such as molecules, uniform walks will backtrack often. Thus, the subgraphs induced by windows of a fixed size s on uniform walks tend to be smaller than for non-backtracking walks. This limits the method's power to evaluate local substructures and long-range dependencies. On all three datasets used in this ablation study, these effects seem to cause a significant loss in performance.

Influence of s and m

Let us further study how the values of the window size s and the number of sampled walks m influence the empirical performance of CRaWl. Generally, we expect performance to improve as both parameters increase: Larger window sizes s allow the model to view larger structural features as the expected size of the induced subgraph increases, while a larger number of walks ensures a more reliable traversal of important features. To test this hypothesis, we conduct an additional ablation study on the ZINC dataset.

First, we evaluate the effect of s on the model performance. To this end, we train models with varying window sizes of $s \in \{0, 2, 4, 6, 8, 10\}$ while all other parameters remain as in our main experiment where we used $s = 8$. We train 3 models for each s and provide the mean test MAE as a function of s in Figure 4.5a. We observe that the performance improves monotonically as s increases. The decrease in MAE is especially significant between $s = 4$ and $s = 6$. This jump can be explained by the importance of benzene rings in organic molecules and in the ZINC dataset in particular. A window of size $s < 6$ is insufficient to detect these structures, which explains the strong improvement around this threshold. The performance saturates around $s = 8$ and is not improved further for $s = 10$. Secondly, we aim to quantify the effect of m. We use the trained models from our main experiment and evaluate them on the test split of ZINC with different numbers of walks m. We vary the number of walks $m = p \cdot |V|$ where $p \in \{0.2, 0.4, 0.6, 0.8, 1.0\}$ and $|V|$ is the number of vertices in a given test graph. All other parameters are unchanged. Figure 4.5b provides the mean test MAE as a function of p. The error decreases monotonically for larger values of m, as predicted. The improvement from $m = 0.8|V|$ to $m = |V|$ is marginal. We note that the walk length ℓ has a similar effect on the performance as the product $m \cdot \ell$ determines how densely a graph is covered in walks. In this ablation study the walk length is fixed at $\ell = 150$ during inference, as in our main experiment. Overall, we conclude that our hypothesis seems to hold as larger values for s and m seem to improve performance.

4.6 Discussion

We have presented the CRaWl architecture that utilizes random walks and 1D CNNs to learn on graphs. We demonstrated the effectiveness of this approach on a variety of graph learning tasks on which it can outperform or at least match state-of-the-art GNNs. On a theoretical level, the expressivity of CRaWl is incomparable to the Weisfeiler-Leman hierarchy which bounds standard MPGNNs. By construction, CRaWl can detect arbitrary substructures up to the size of its local window. Thus CRaWl is able to extract useful features from highly symmetrical graphs such as the 4-regular graphs of CSL on which pure MPGNNs fail due to the lack of expressivity. This way random walks allow CRaWl to escape the Weisfeiler-Leman hierarchy and solve tasks that are impossible for MPGNNs.

CRaWl can be viewed as an attempt to process random walks and the structures they induce with end-to-end neural networks. The strong empirical performance demonstrates the potential of this general approach. However, many variations remain to be explored, including different walk strategies, variations in the walk features, and alternative pooling functions for pooling walklet embeddings into nodes or edges. Beyond plain 1D-CNNs, other deep learning architectures for sequential data, such as LSTMs or sequence Transformers, could be used to process random walks.

Since our CRaWl layer is compatible with both MPGNNs and Graph Transformers, it raises the question of how a combination of all three techniques would perform as

the three approaches focus on different aspects of the graph structure. Through a unified architecture that runs all three models in parallel, we would thus anticipate a performance gain due to synergies.

To conclude, we demonstrated that effective graph learning architectures do not necessarily require message passing, Transformers, or adherence to the Weisfeiler-Leman hierarchy. We believe that the strong empirical results illustrate that exploring new approaches with provably distinct properties can only strengthen the available ensemble of methods.

Long-Range Interactions in Theory and Practice

In this chapter, we study graph learning architectures that enable long-range interactions (LRI) between vertices, such as Virtual Nodes (VN) and Graph Transformers (GT). The general aim of these methods is to overcome the limited receptive field of standard MPGNNs on tasks where this constitutes a performance bottleneck (see Section 2.4). While this has long been a subject of study in graph learning, it recently received much attention due to the growing popularity of Graph Transformers. It is therefore important to understand how different LRI-enabling architectures affect learning performance. For many real-world graph learning tasks, it is challenging to quantify how strongly the target function depends on LRI. Comparing architectures in this context therefore requires careful experimentation and a grounded theoretical understanding. Working towards these goals is the main aim of this chapter.

The content of this chapter is structured as follows: We provide an experimental re-evaluation of popular benchmark datasets for LRI in Section 5.1, based on Tönshoff et al. (2024). There, the main result is that prior work overestimates the advantage of Graph Transformers over standard MPGNNs on these specific datasets. We further suggest specific improvements to the experimental setup to mitigate these issues. In Section 5.2 we present results on the uniform expressivity of self-attention and Virtual Nodes from Rosenbluth et al. (2024). A key result of this work is that MPGNNs with Virtual Node and Graph Transformers such as GPS (Rampášek et al., 2022) are incomparable in the sense that neither architecture subsumes the other in terms of uniform function approximation.

5.1 Reassessing the Long-Range Graph Benchmark

The evaluation of graph learning techniques relies strongly upon empirical comparisons in benchmark datasets. While simple LRI-enabling techniques like Virtual Nodes have been known to improve performance on some popular benchmark datasets (Hu et al., 2020c), the rapid development of new Graph Transformer architectures necessitates datasets that are specifically designed to compare the proposed approaches in a fine-grained manner. To this end, Dwivedi et al. (2022) introduced the Long-Range Graph Benchmark (LRGB) which is a collection of five datasets:

- *Peptides-func* and *Peptides-struct* are graph-level classification and regression tasks, respectively. Their aim is to predict various properties of peptides which are modelled as molecular graphs.
- *PascalVOC-SP* and *COCO-SP* model semantic image segmentation as a node-classification task on superpixel graphs.
- *PCQM-Contact* is a link prediction task on molecular graphs. The task is to predict pairs of vertices that are distant in the graph but in contact in 3D space.

Table 5.1 provides the main statistics of each dataset. The five LRGB datasets contain graphs that are larger and of higher diameter than those used in prior datasets for similar prediction tasks. A core hypothesis of LRGB is that these learning tasks depend strongly on long-range interactions and are therefore suitable for comparing LRI-enabling architectures empirically. The experiments provided by Dwivedi et al. (2022) report a strong performance advantage of Graph Transformers over the MPGNN architectures GCN (Kipf and Welling, 2017), GINE (Hu et al., 2020b), and GatedGCN (Bresson and Laurent, 2017), in accordance with the expectations. Subsequently, newer work on LRI-enabling architectures, especially Graph Transformers, frequently use LRGB datasets to conduct experimental comparisons (Rampášek et al., 2022, Shirzad et al., 2023, Ma et al., 2023, He et al., 2023).

Our contribution is three-fold[1]: First, we show that the three MPGNN baselines GCN, GINE, and GatedGCN all profit massively from further hyperparameter tuning, reducing and even closing the gap to Graph Transformers on multiple datasets. In fact, GCN yields state-of-the-art results on Peptides-Struct, surpassing several newer Graph Transformers. On this dataset in particular, most of the performance boost is due to a deep readout function instead of a linear one, again highlighting the importance of hyperparameters. Second, we show that on the vision datasets PascalVOC-SP and COCO-SP normalization of the input features

[1]Source code: `https://github.com/toenshoff/LRGB`

Dataset	Peptides-Func	Peptides-Struct	PascalVOC-SP	COCO-SP	PCQM-Contact
Task	Graph Cls.	Graph Reg.	Node Cls.	Node Cls.	Link Pred.
#Graphs	15,535	15,535	11,355	123,286	529,434
Avg. Nodes	150.94	150.94	479.40	476.88	30.14
Avg. Diameter	56.99	56.99	27.62	27.39	9.86

Table 5.1: Overview of the five Long-Range Graph Benchmark datasets.

is highly beneficial. We argue that, as in the vision domain, feature normalization should be the default setting. Third and last we take a closer look at the MRR metric used to evaluate PCQM-Contact. There, we demonstrate different filtering strategies have a major impact on the results and must be implemented exactly to specification to facilitate reliable comparisons.

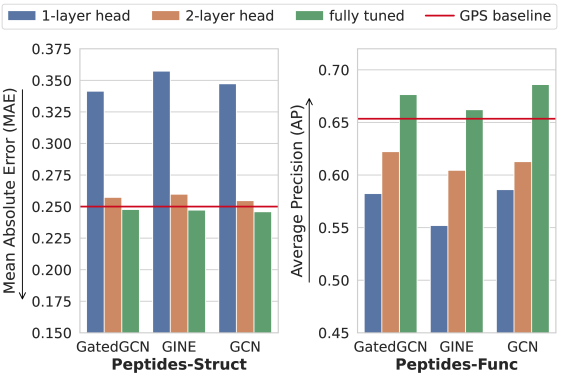
5.1.1 Experimental Setup

Our primary focus is to reevaluate the MPGNN architectures for which Dwivedi et al. (2022) originally reported baseline results, namely GCN (Kipf and Welling, 2017), GINE (Hu et al., 2020b), and GatedGCN (Bresson and Laurent, 2017). We additionally reevaluate the reported results for GPS (Rampášek et al., 2022) to quantify how our experimental changes affect the results of Graph Transformers. Many other Graph Transformer architectures have also reported results on LRGB datasets, including Exphormer (Shirzad et al., 2023), GRIT (Ma et al., 2023) and Graph ViT / GraphMLPMixer (He et al., 2023). Several architectures not based on Transformers have also been evaluated on LRGB, such as CRaWl (Tönshoff et al., 2023b) and DRew (Gutteridge et al., 2023). We will include their reported results in our comparison. While our focus here is to reevaluate standard MPGNNs on the LRGB datasets, we also provide results for models with Virtual Nodes in Section 5.2.4.

Hyperparameters We argue that the baseline results reported by Dwivedi et al. (2022) are not representative for MPGNNs and suffer from suboptimal hyperparameters. We provide new results for the same MPGNN architectures that are obtained after a straightforward hyperparameter sweep. We tune the main hyperparameters (such as depth, dropout rate, ...) in pre-defined ranges while strictly adhering to the official 500k parameter budget. In particular, we looked at networks with 6 to 10 layers, varied the number of layers in the prediction head (i.e. the readout function) from 1 to 3, and also considered the dropout and learning rate of the network. Note that we also view the usage of positional or structural encoding (none / LapPE (Dwivedi and Bresson, 2021) / RWSE (Dwivedi et al., 2021)) as a hyperparameter that is tuned for each method, including all MPGNNs. For GPS we additionally evaluated the internal MPGNN architecture and whether to use BatchNorm or LayerNorm. All tuned hyperparameters and their respective sweep budgets are specified in detail in Table 5.2. The final model

	Method	PEPTIDES-FUNC Test AP ↑	rel imp	PEPTIDES-STRUCT Test MAE ↓	rel imp
LRGB	GCN	0.5930 ± 0.0023		0.3496 ± 0.0013	
	GINE	0.5498 ± 0.0079		0.3547 ± 0.0045	
	GatedGCN	0.6069 ± 0.0035		0.3357 ± 0.0006	
	Transformer	0.6326 ± 0.0126		0.2529 ± 0.0016	
	SAN	0.6439 ± 0.0075		0.2545 ± 0.0012	
	GPS	0.6535 ± 0.0041		0.2500 ± 0.0005	
Ours	GCN	0.6860 ± 0.0050	+16%	**0.2460 ± 0.0007**	+30%
	GINE	0.6621 ± 0.0067	+20%	0.2473 ± 0.0017	+30%
	GatedGCN	0.6765 ± 0.0047	+11%	0.2477 ± 0.0009	+26%
	GPS	0.6534 ± 0.0091	± 0%	0.2509 ± 0.0014	± 0%
Others	CRaWl	0.7074 ± 0.0032		0.2506 ± 0.0022	
	DRew	**0.7150 ± 0.0044**		0.2536 ± 0.0015	
	Exphormer	0.6527 ± 0.0043		0.2481 ± 0.0007	
	GRIT	0.6988 ± 0.0082		**0.2460 ± 0.0012**	
	Graph ViT	0.6942 ± 0.0075		**0.2449 ± 0.0016**	
	G-MLPMixer	0.6921 ± 0.0054		0.2475 ± 0.0015	

(a) Previous and updated results on the peptides datasets. Best results (within stdev) in **bold**.



(b) Exchanging the linear prediction head by an MLP accounts for most of the additional performance of all three MPGNNs, especially on Peptides-Struct.

Figure 5.1: On both Peptides datasets, all three MPGNNs surpass GPS. On Peptides-Struct a basic GCN model even achieves SOTA results.

configurations after tuning are provided in Tables C.1 to C.5 in the appendix. We make some additional setup changes based on preliminary experiments. All models are trained with an AdamW optimizer using a cosine annealing learning rate schedule and linear warmup. This differs from Dwivedi et al. (2022), who optimized the MPGNN models with a "Reduce on Plateau" schedule and instead matches the learning rate schedule of GPS (Rampášek et al., 2022). We set the weight decay to 0.0 in all five datasets and switch to slightly larger batch sizes to speed up convergence. We also choose GeLU (Hendrycks and Gimpel, 2016) as our default activation function. Furthermore, we change the prediction head for graph-level tasks such that all hidden layers have the same hidden dimension as the GNN itself. These were previously configured to become more narrow with depth, but we could not observe any clear benefit from this design choice. Last, all MPGNN models use proper skip connections that go around the entire GNN layer. The original LRGB results use an implementation of GCN as provided by GraphGym (You et al., 2020). The skip connections in this implementation do not skip the actual non-linearity at the end of each GCN layer, possibly hindering the flow of gradients. We reimplement GCN with skip connections that go around the non-linearity. Note that these additional tweaks are **not** used in our ablation studies in Figure 5.1b and Figure 5.2b when training the intermediate models where we only change the head depth and normalization, respectively. There, we use identical model configurations to those used in the literature.

Feature Normalization The vision datasets PascalVOC-SP and COCO-SP have multi-dimensional node and edge features with values spanning different orders of magnitude for different feature channels. Passing this input to a neural network without channel-wise normalization can cause poorly conditioned activations. While feature normalization is standard practice in deep learning and computer vision in particular, neither Dwivedi et al. (2022) nor any subsequent works using LRGB utilize it, except CRaWl (Tönshoff et al., 2023b). We apply channel-wise linear

Hyperparameter	Range	Default
#layers	$[6, 8, 10]$	8
head depth	[1, 2, 3]	2
PE/SE	[none, LapPE, RWSE]	none
lr	$[0.001, 0.0005, 0.0001]$	0.001
dropout	$[0.0, 0.1, 0.2]$	0.1
normalization (GPS only)	[LayerNorm, BatchNorm]	BatchNorm
MPGNN (GPS only)	[GCN, GINE, GatedGCN]	GatedGCN

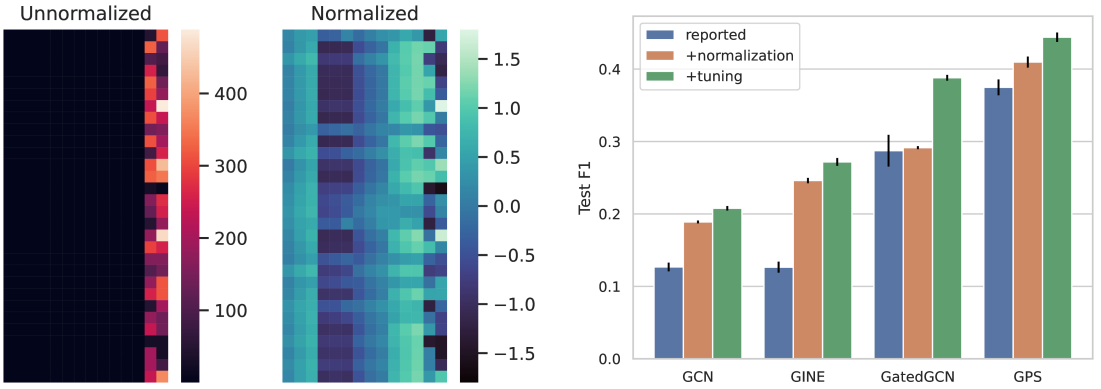
Table 5.2: The hyperprameter ranges used in our experiments.

normalization to all node and edge features as visualized in Figure 5.2a. Specifically, we compute the channel-wise mean $\mu \in \mathbb{R}^d$ and standard deviation $\sigma \in \mathbb{R}^d$ on the train split, where d is the feature dimension. Each feature vector $x \in \mathbb{R}^d$ is then normalized through the channel-wise transformation $\tilde{x} = \frac{x-\mu}{\sigma}$. These transformed features are then passed as input to the model. We show that all models (baselines and GPS) profit from this procedure in Figure 5.2b.

Link Prediction Metrics The evaluation metric on the link-prediction dataset PCQM-Contact (Dwivedi et al., 2022) is the Mean Reciprocal Rank (MRR) in a *filtered* setting, as defined by Bordes et al. (2013). For predicted edge scores the MRR measures how a given true edge (h, t) is ranked compared to all possible candidate edges (h, x) of the same head. As there might be multiple true tails t for each head h, the *filtered* MRR removes those other true tails (false negatives) from the list of candidates before computing the metric. This filtering avoids erroneously low MRR values due to the model preferring other true edges and is common in link-prediction tasks. Even though Dwivedi et al. (2022) explicitly define the metric to be the filtered MRR, the provided code computes the *raw* MRR, i.e. keeping other true tails in the list. We report results on PCQM-Contact in a corrected filtered setting. We additionally provide results with an extended filtering procedure where self-loops of the form (h, h) are also removed from the set of candidates since these are semantically meaningless and never positive in the underlying prediction task. This is impactful as the scoring function used by Dwivedi et al. (2022) is based on a symmetric dot-product and therefore exhibits a strong bias towards self-loops.

5.1.2 Results

Here, we will present and discuss the main results of our reevaluation and provide additional ablation studies to isolate specific causes for the performance shifts. An overview of all measured performance metrics is also given in Table C.6 in the appendix.



(a) Vertex feature matrices ($|V| \times 14$) on PascalVOC-SP before and after channel-wise normalization.

(b) The effects of feature normalization and hyperparameter tuning on PascalVOC-SP.

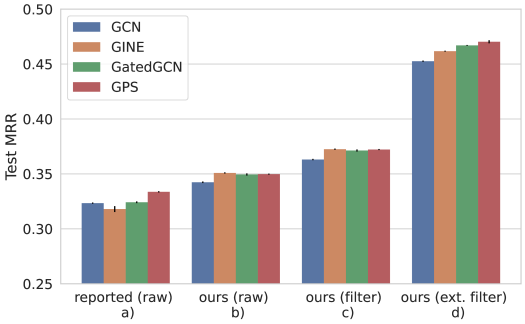
Figure 5.2: On PascalVOC-SP and COCO-SP feature normalization and further tuning improves performance across all compared methods.

Peptides-Func and Peptides-Struct Table 5.1a provides the results obtained on the test splits of the Peptides-Func and Peptides-Struct. For the MPGNN baselines we observe considerable improvements on both datasets as all three MPGNNs outperform GPS after tuning. The average precision on Peptides-Func increased relatively by around 10% to 20%. GCN achieves a score of 68.60%, which is competitive with newer Graph Transformers such as GRIT or Graph ViT. The improvement on Peptides-Struct is even more significant with a relative reduction of the MAE of 30%, fully closing the gap to recently proposed GTs. Surprisingly, a simple GCN is all you need to match the best known results on Peptides-Struct. The results for GPS effectively stayed the same as in the original paper (Rampášek et al., 2022). Those values thus seem to be representative for GPS.

We observed that the key hyperparameter underlying the improvements of all three MPGNNs is the depth of the prediction head. To show this Figure 5.1b contains an ablation where we exchanged the linear prediction head configured by Dwivedi et al. (2022) with a 2-layer perceptron, keeping all other hyperparameters the same. While the benefit on Peptides-Func is considerable and highly significant, on Peptides-Struct the head depth accounts for almost the complete performance gap between MPGNNs and GTs. GPS' performance with linear and deeper prediction heads is largely unchanged. For example, our GPS configurations in Table 5.1a use a 2-layer prediction head. Our results indicate that the prediction targets of both datasets appear to depend non-linearly on global graph information. In this case, MPGNNs with linear prediction heads are unable to model the target function. Graph Transformers are not as sensitive to linear prediction heads, since each layer can process global graph information with a deep feedforward network. However, we would argue that switching to a deep predictive head represents a simpler and computationally cheaper solution to the same issue.

	Method	PascalVOC-SP Test F1 ↑	rel imp	COCO-SP Test F1 ↑	rel imp
LRGB	GCN	0.1268 ± 0.0060*		0.0841 ± 0.0010*	
	GINE	0.1265 ± 0.0076*		0.1339 ± 0.0044*	
	GatedGCN	0.2873 ± 0.0219*		0.2641 ± 0.0045*	
	Transformer	0.2694 ± 0.0098*		0.2618 ± 0.0031*	
	SAN	0.3230 ± 0.0039*		0.2592 ± 0.0158*	
	GPS	0.3748 ± 0.0109*		0.3412 ± 0.0044*	
Ours	GCN	0.2078 ± 0.0031	+64%	0.1338 ± 0.0007	+59%
	GINE	0.2718 ± 0.0054	+115%	0.2125 ± 0.0009	+59%
	GatedGCN	0.3880 ± 0.0040	+35%	0.2922 ± 0.0018	+11%
	GPS	0.4440 ± 0.0065	+18%	**0.3884 ± 0.0055**	+13%
Others	CRaWl	**0.4588 ± 0.0079**		-	
	DRew	0.3314 ± 0.0024*		-	
	Exphormer	0.3960 ± 0.0027*		0.3430 ±0.0008*	

(a) Tuning results on vision datasets PascalVOC-SP and COCO-SP. *No normalization used.



(b) Results on PCQM-Contact. For our own models we provide the MRR scores with varying levels of filtering.

PascalVOC-SP and COCO-SP Table 5.3a provides the results obtained on the test splits of the superpixel datasets PascalVOC-SP and COCO-SP. We observe significant improvements for all evaluated methods. On PascalVOC-SP the F1 score of GatedGCN increases to 38.80% which exceeds the original performance reported for GPS by Rampášek et al. (2022). GPS also improves significantly to 44.40% F1. This is only one percentage point below the results achieved by CRaWl, which was the only reported result with normalized features. The previously large performance gap between GPS and CRaWl is therefore primarily explained by GPS processing raw input signals. On COCO-SP, we observe similar results. Here GPS sets a new state-of-the-art F1 score of 38.84%.

Note that these improvements are achieved entirely through data normalization and hyperparameter tuning. Figure 5.2b provides an ablation on the individual effect of normalization. We train intermediate models with configurations identical to those used by Dwivedi et al. (2022) and Rampášek et al. (2022), but with feature normalization. For GatedGCN we observe a slight performance increase but a large reduction in the variance across random seeds. For the remaining methods, including GPS, normalization of node and edge features already accounts for at least half of the observed performance gain, emphasizing its importance in practice.

PCQM-Contact Figure 5.3b plots the MRR scores obtained on the test split with various evaluation settings as described in Section 5.1.1. First, we provide the results originally reported for LRGB in the literature (a). Recall that these values are obtained in a raw setting with false negatives present. We then provide results obtained after training our own model with new hyperparameters (chosen based on the raw MRR) in b). We still use the raw MRR for evaluation in b) to measure the impact of hyperparameter tuning. Tuning yields an absolute improvement of around 3%. The previously reported slight performance edge of GPS is not observable in this setting after tuning. In subplot c) we measure the MRR of our models in the filtered setting. Note that these values are based on the exact same predictions as in b), but false negatives are removed. The measured MRR increases by roughly 3% when compared to the raw setting. This shift could erroneously be interpreted as a significant improvement when comparing to literature values obtained in a raw setting. In d) we evaluate our models (still using the same

predictions) in an extended filtered setting where we additionally remove self-loops from the candidate pool. Compared to the filtered MRR in c) the MRR metric increases by about 10 percentage points, indicating that self-loops strongly affect the results. Note that in d) GPS again slightly outperforms the MPGNN baselines, in contrast to b) and c). This means that GPS' predictions seem to suffer overproportionally when self-loops are not filtered. Therefore, the specific choice of how negative samples are filtered on PCQM-Contact can directly affect the ranking of compared methods and must be considered and implemented with care.

5.1.3 Discussion

In our experiments, we observed considerable performance gains for all three MPGNN baselines. First, this indicates that extensive baseline tuning is important for properly assessing one's own method, especially on relatively recent datasets. Second, only on the two superpixel datasets do Graph Transformers exhibit clear performance benefits against MPGNNs, indicating that either there are ways to solve the other tasks without long-range interactions or Graph Transformers are not inherently better at exploiting such long-range dependencies.

Going forward, additional benchmark datasets that test for LRI would be highly useful to further evaluate existing and future architectures. A stronger focus on synthetic tasks may be helpful to overcome the caveats of the LRGB datasets, where it is unclear how strongly the proposed real-world tasks truly depend on LRI. Classical algorithmic tasks such as path-finding or vertex coloring could be used to define target functions that, by construction, require LRI on synthetic high-diameter graphs.

To evaluate the potential virtues of Graph Transformers it may further be important to evaluate models on datasets that are large-scale, rather than long-range. When Vision Transformers were initially shown to outperform CNNs on certain vision tasks, this was only achieved when pre-training on massive datasets with over 100 Million images (Dosovitskiy et al., 2021). While more data-efficient Vision Transformers have since been developed (Touvron et al., 2021), the general consensus that Transformers are "data-hungry" prevails across domains. Many popular benchmark datasets in graph learning are comparatively small and often contain less than 100K graphs. These datasets may therefore simply be to small to observe the potential scaling benefits that Transformers have demonstrated in other domains. Recent work has begun to investigate the scaling of MPGNNs and Graph Transformers to new large-scale datasets (Beaini et al., 2024, Sypetkowski et al., 2024). This research direction may open up new perspectives on the strengths and weaknesses of the many architectures that have been proposed.

5.2 Expressivity of Self-Attention and Virtual Nodes

Shortly after the emergence of Graph Transformers, the expressivity of such architectures became a natural subject of study with early work by Kreuzer et al. (2021). Like many theoretical papers in graph machine learning (including e.g. Morris et al., 2020a, Xu et al., 2019, Abboud et al., 2021b), the analysis uses the non-uniform setting where the size and structure of the neural networks may depend on the size of the input graphs. Kreuzer et al. (2021) prove that Graph Transformers can approximate every function when provided with precomputed positional encodings such as LapPE (Dwivedi and Bresson, 2021). In Section 5.2.2 we argue that it is not the transformer architecture, but rather the combination of positional encoding and non-uniformity that enable universal function approximation and show that the same result holds for shallow MPGNNs and even MLPs. All three architectures are therefore equally expressive in a non-uniform setting when given sufficiently powerful PEs.

We then compare LRI-enabling architectures under the lens of uniform function approximation in Section 5.2.3. Here, a single network must work for graphs of all sizes. While non-uniform expressivity only guarantees the existence of a model that works well for graphs up to a fixed size, uniform expressivity implies the existence of a model that generalizes to out-of-distribution graph sizes. Uniform function approximation therefore represents a significantly stronger notion of expressivity. We focus on a comparison between MPGNNs with Virtual Nodes (MPGNNs+VN) and Graph Transformers (GTs), where we use GPS (Rampášek et al., 2022) as a prototypical and generic GT architecture.

First, we show that uniform universal approximation is impossible, both for GTs and MPGNNs+VN even when using positional encodings. We then focus on the differences between the architectures and show that neither can emulate the other, by providing functions that can be expressed by GTs but not by MPGNNs+VN and vice versa. These results focus on model architecture and hold regardless of whether positional encodings like LapPE are used or not. Our theory shows that due to softmax-attention eventually being a weighted average, Graph Transformers cannot uniformly solve tasks that require unbounded aggregation. In contrast, Virtual Nodes that internally perform sum-aggregation over all nodes can by definition sum over the nodes in a graph. Based on this observation, we show that the function $|V|^2$ differentiates MPGNNs+VN from GTs. Inversely, we show that a Virtual Node is not sufficient to emulate the intricate computation of self-attention uniformly and provide a function that GTs can express but MPGNNs+VN cannot. A fundamental difference between self-attention and VNs is that in self-attention the updating node can access each of the other nodes individually while with VNs it can only access their aggregated value. However, as proven by (Grohe and Rosenbluth, 2024), that difference alone does not imply an expressivity advantage. Proving that neither MPGNN+VN nor GT subsumes the other is the main technical contribution presented in this section.

We complement our theoretical findings with an empirical study on synthetic datasets and observe that a uniformly expressive model can (sometimes) be learned, and a uniform-inexpressive model does not generalize in terms of graph size. On real-world datasets, we observe mixed results when comparing MPGNNs+VN and GTs, with neither architecture clearly outperforming the other. This loosely aligns with our main result of both architectures being incomparable.

5.2.1 Related Work

Cai et al. (2023) explore differences between Graph Transformers and MPGNNs+VN and prove that with only constant blowup in depth, MPGNN+VN can approximate GTs based on linear variations of self-attention, such as Performer (Choromanski et al., 2021). They also show that with a constant number of wide (i.e. $O(n^d)$) layers, MPGNN+VN can approximate a full self-attention layer. Furthermore, they present an explicit construction using $O(n)$ MPGNN+VN layers of constant width to simulate a self-attention layer, albeit with strong assumptions on the node features. Our theoretical results are perpendicular to theirs. While Cai et al. (2023) show that MPGNNs+VN can mimic GTs in the non-uniform setting, we analyze the architectures in the uniform setting. We note that their first result about linear Graph Transformers also holds in the uniform setting, implying that MPGNN+VN architectures subsume linear Graph Transformers.

Major works in function approximation of Transformers are (Yun et al., 2020) for general Transformers and (Kreuzer et al., 2021) for Graph Transformers. Both prove universal function approximation in a non-uniform setting. Kim et al. (2022) show that using (orthogonal) random node identifiers and encoding both nodes and edges as tokens, a standard Transformer subsumes 2-WL and thus most pure MPGNN architectures. Note that MPGNNs with random node identifiers are universal approximators for probabilistic functions on graphs (Abboud et al., 2021b).

There are a few works that analyze expressivity in the uniform setting. In Barceló et al. (2020) and Barceló et al. (2021) they characterize the expressive power of GNNs in terms of logic and in the second paper take homomorphism counts as structural encodings into account. Rosenbluth et al. (2023) show that sum-aggregation GNNs do not subsume other aggregations in the uniform setting – as opposed to the non-uniform setting where sum-aggregation GNNs can express anything a GNN can express. Morris et al. (2023) analyze the VC dimension of GNNs in the uniform setting. Other than that, essentially all theoretical papers assume the non-uniform setting.

5.2.2 Non-Uniform Function Approximation on Graphs

In this section, we present universality results that are more or less immediate consequences of MLPs being universal function approximations. Nevertheless, we

believe that it is useful to discuss these results to clarify that the universality stated in the Graph Transformer literature is not a unique property of Transformers. This has also been discussed in Müller et al. (2024).

Injective Positional Encodings All universality results discussed here rely on the usage of sufficiently powerful positional encodings. Let x_{PE} be some positional encoding as defined in Section 2.3. We further define $M_{x_{\text{PE}},G} = \{\!\{x_{\text{PE}}(G, v) : v \in V(G)\}\!\}$ as the multiset of vectors that x_{PE} assigns to the vertices of G. We say that π is *injective up to isomorphism and order* n if for all graphs G, H of order at most n it holds that:

$$G \ncong H \implies M_{x_{\text{PE}},G} \neq M_{x_{\text{PE}},H}. \tag{5.1}$$

That is, two graphs map to the same multiset of vertex embeddings only if they are isomorphic. It has been shown that LapPE, as defined by Kreuzer et al. (2021), has this property when considering the first n eigenvalues. Note that the converse of Equation 5.1 is not required to hold as PEs, such as LapPE, are often *not* equivariant.

Based on the positional encoding LapPE being injective up to isomorphism and order n, Kreuzer et al. (2021) proved that there exists a Transformer network T that maps the positional encodings of graphs G, H of order at most n to the same output if and only if G and H are isomorphic. This is a consequence of the injectivity of the positional encoding x_{PE} combined with the universality of Transformer networks (Yun et al., 2020). Having realized this, it is clear that the existence of such an "isomorphism network" is not unique to Transformers, but shared by other sufficiently powerful architectures including MPGNNs and even MLPs.

Proposition 5.1. *2-layer MLPs and 1-layer MPGNNs are universal function approximators when given access to a positional encoding x_{PE} that is injective up to isomorphism and order n.*

We give formal statements and proofs for both architectures in Rosenbluth et al. (2024). As research on the expressivity of GNNs has often focused on the power of GNNs to distinguish graphs, we note that universal function approximation and graph isomorphism testing have been shown to be equivalent (Chen et al., 2019). Thus, both 2-layer MLPs and 1-layer MPGNNs with injective positional encodings are able to distinguish all pairs of graphs. This fact does not contradict the well-known limitation of MPGNNs to be at most as powerful as the Weisfeiler-Leman algorithm for distinguishing graphs (Morris et al., 2019, Xu et al., 2019, see Section 2.2.1). To the contrary, Proposition 5.1 aligns with the fact that if the positional encoding x_{PE} is used as an initial coloring for each graph, then Weisfeiler-Leman distinguishes any two non-isomorphic graphs instantly after zero rounds of refinement. The caveat here is that there are cases where the algorithm also distinguishes isomorphic graphs G, H with $M_{x_{\text{PE}},G} \neq M_{x_{\text{PE}},H}$.

Note that the proofs for Proposition 5.1 as well as the proof of the universality of Graph Transformers (Kreuzer et al., 2021) share the same principle: Essentially they just implement giant look-up tables for finitely many inputs. Hence, they do not provide any deep insights into the relative strengths and weaknesses of Graph Transformers and MPGNNs.

5.2.3 Uniform Function Approximation with Self-Attention and Virtual Nodes

In Section 2.2.2 we examined the uniform expressivity of different aggregation functions. There, the focus was on node-level regression tasks. Here, all results concern graph-level regression tasks. Let us therefore revisit the definition of uniform function approximation in this context. Formally, fix some feature domain $D \subseteq \mathbb{R}^d$ and some class of graph-level GNNs C. Let $f^* \in \mathcal{F}_{D\to\mathbb{R}}$ be some target function for graph-level regression. We say that C *uniformly additively approximates* f^* if and only if for all $\epsilon > 0$ there exists $f \in C$:

$$\forall G \in \mathcal{G}_D : \ |f(G) - f^*(G)| \leq \epsilon \tag{5.2}$$

We denote this by $C \approx f^*$. Intuitively, uniform approximation states that a single model "works" for graphs of all sizes, unlike non-uniform approximation which allows having different functions for graphs of different sizes. Throughout this section, approximation always means uniform additive approximation and we use the term "approximates" synonymously with "expresses".

For two classes of GNNs C and C' we say that C *subsumes* C' if $C \approx f'$ holds for all $f' \in C'$ and denote this with $C \geq^D C'$, as in Section 2.2.2. If the subsumption holds only for graphs featured with a subset $D' \subseteq D$ we notate it as $C \geq^{D'} C'$, if it does not hold for graphs featured with a subset $D' \subseteq D$ we notate it as $C \not\geq^{D'} C'$.

In the following, we examine the inability of MPGNNs+VN and GPS to uniformly express various target functions. We rely on the exact model definitions provided in Section 2.4. Throughout this section, we assume that piecewise linear activation functions are used for all MLPs in both architectures. We further assume that the pooling functions used by virtual nodes and readout modules are restricted to sum-, mean-, or max-aggregations, as is common in practice. Note that throughout the section we consider GPS models whose layers incorporate neither BatchNorm nor LayerNorm. BatchNorm does not increase GPS expressivity, and LayerNorm would not help GPS in the tasks we define. This restriction therefore does not affect our inexpressivity results for GPS. We show that the inexpressivity holds true even when standard positional or structural encodings are used, such as LapPE and RWSE (see Section 2.3). Such encodings are commonly used in practice and have the following properties: They are polynomial-time computable and their range is bounded. The former assumption is used in Theorem 5.2 and the latter in Theorem 5.4. In contrast to Section 5.2.2, injective PEs do not guarantee expressivity in the uniform setting. Clearly, the proof logic based on indexing all

inputs is not applicable in an infinite input-domain setting. In addition, commonly used PEs are not injective for input graphs of unbounded size. The complete proofs of the lemmas and theorems in this section can be found in Rosenbluth et al. (2024).

5.2.3.1 GPS And MPGNNs+VN Are Not Universal Approximators

For the sake of completeness, we show that none of the considered models is a universal function approximator in the uniform setting.

Theorem 5.2. *Let h be the characteristic function of an NP-hard problem on graphs, for example: $h(G) = 1$ if G is 3-colorable and $h(G) = 0$ otherwise. Let $x_{PE} : \mathcal{G}_* \to \mathbb{R}$ be a positional encoding function that is computable in polynomial time. Then, assuming PTIME $\neq$ NPTIME we have that GPS $\not\approx h$ and MPGNNs+VN $\not\approx h$ even when the input graph is featured with positional encodings computed by x_{PE}.*

The proof of this theorem is straightforward and relies on the fact that the runtime of both considered architectures is polynomial in the size of the graph. Therefore, neither class of models can compute a function that is not in PTIME.

Corollary 5.3. *GPS and MPGNNs+VN cannot approximate every computable function on graphs, not even every graph function in NPTIME, even when the input graph is featured with PTIME-computable positional encodings.*

This result generalizes to any architecture with polynomial runtime. Achieving universality in a uniform setting is therefore not a viable objective for GNN architecture design.

5.2.3.2 GPS Does Not Subsume MPGNNs+VN

In the following, we construct a target function that is expressible by an MPGNN+VN network but can not be approximated uniformly by GPS. This holds true even when the input graphs to the GPS are featured with bounded positional encodings. Let x_{PE} be a positional encoding function with bounded range $[-1, 1]^d$. Define a family of graphs $\{G_n\}_{n \in \mathbb{N}_{>0}}$, where $G_n = (V_n, E_n, x_n)$ is defined as follows: $V_n = \{v_1, \ldots, v_n\}$; $E_n = \emptyset$, and $\forall v \in V_n \; x_n(v) \coloneqq x_{\text{PE}}(G_n, v)$.

Theorem 5.4. *Let $f : \mathcal{G}_{[-1,1]} \to \mathbb{R}$ be a graph embedding such that for every n it holds that $f(G_n) = n^2$. Then, no GPS network can approximate f for all graphs G_n. Formally, GPS $\not\approx f$.*

The proof's main argument is that for every GPS network (with fixed weights), if the input graphs are of bounded degree and bounded features, then the latent vertex embeddings computed by each layer are also bounded by some constant. The result of the final readout function, which is assumed to use sum-, mean-

or max-pooling and a piecewise-linear MLP, is therefore in $\mathcal{O}(n)$ and can not approximate the target function n^2 with bounded error for all graph sizes $n \in \mathbb{N}$.

In contrast, a 1-layer MPGNN+VN is able to exactly compute f as follows: The sum-aggregation of the virtual node is able to compute n (aggregating a 1 from every node) and propagate this back to each node. Then the sum-readout simply sums up the values of all nodes to reach n^2. When combined with Theorem 5.4 we obtain the following corollary:

Corollary 5.5. *GPS* $\not\geq^{[-1,1]}$ *MPGNNs+VN.*

The boundedness of the self-attention operation prevents it from approximating functions that grow unbounded with the graph size, some of which can be expressed with sum-based virtual nodes. Graph Transformers are therefore no silver bullet for graph learning, as simple functions such as regressing $|V|^2$ can not be approximated robustly with architectures such as GPS.

5.2.3.3 MPGNNs+VN Do Not Subsume GPS

We further prove that there is a target function that is expressible by GPS but not by any MPGNN+VN network. To this end, we define a family of featured graphs $\{G_{l,r}\}_{l,r\in\mathbb{N}_{>0}}$, where $G_{l,r} = (V_{l,r}, E_{l,r}, x_{l,r})$ is defined as follows:

$$V_{l,r} = \{u_1, \dots, u_l, w_1, \dots, w_{l\cdot r}\},\ E_{l,r} = \emptyset,\ \text{and}$$
$$x_{l,r} : V_{l,r} \to \{1,2\}^2,\ v \mapsto \begin{cases} (2,1) & \text{if } v \in \{u_1, \dots, u_l\} \\ (2,2) & \text{if } v \in \{w_1, \dots, w_{l\cdot r}\}. \end{cases}$$

Let some MPGNN+VN network N be given. We first characterize the functions that can be expressed by N when processing $G_{l,r}$, which are piecewise-rational in l and r by construction. Formally, the following holds true:

Lemma 5.6. *There exists a finite set of functions* $F = \{f_1, \dots, f_k\}$, $f_i : \mathbb{N}\times\mathbb{N} \to \mathbb{R}$, *such that:*

(1) $\forall f \in F :\ f(l,r) = \sum_{i=0}^{m+1} \sum_{j=0}^{i} \sum_{x=0}^{m+1} \sum_{y=0}^{x} a_{i,j,x,y}^{(f)} l^i r^j \frac{r^y}{(1+r)^x}$ *for some real coefficients* $a_{i,j,x,y}^{(f)} \in \mathbb{R}$

(2) $\forall l \in \mathbb{N}_{>0}\ \forall r \in \mathbb{N}_{>0}\ \exists f \in F : f_N(G_{l,r}) = f(l,r)$

The proof of Lemma 5.6 relies on the assumption that the model is composed of MLPs with piecewise-linear activation functions. Given that the Virtual Node can perform either sum-, mean- or max-aggregation one can show inductively that the model computes a function that is a piecewise composition of the terms in Lemma 5.6 (1).

Next, we define a target function $h(G_{l,r})$ and characterize the gap that necessarily exists between it and any function of the kind described in Lemma 5.6 (1). For $l \in \mathbb{N}_{>0}, r \in \mathbb{N}_{>0}$, we let $h(G_{l,r}) \coloneqq l\big(\frac{3+2re^9}{1+re^9} + r\frac{3+2re^{12}}{1+re^{12}}\big)$. For this specific target function, it can be shown that the following holds true:

Lemma 5.7. *Let* $f(l,r) = \sum_{i=0}^{m+1}\sum_{j=0}^{i}\sum_{x=0}^{m+1}\sum_{y=0}^{x} a_{i,j,x,y} l^i r^j \frac{r^y}{(1+r)^x}$ *for some coefficients* $a_{i,j,x,y}$, *then*

$$\exists r_0 : \forall r > r_0 \ \lim_{l\to\infty} |f(l,r) - h(G_{l,r})| = \infty$$

Essentially, Lemma 5.7 states that the type of function defined in Lemma 5.6 (1) can not uniformly approximate $h(G_{l,r})$ when both parameters r and l are unbounded. As Lemma 5.6 asserts that any MPGNN+VN model computes a piecewise composition of finitely many such functions, we obtain that h can not be approximated uniformly by any MPGNN+VN:

Theorem 5.8. *No MPGNN+VN can approximate* h *for all graphs* $G_{l,r}$, $l, r \in \mathbb{N}_{>0}$. *Formally, MPGNNs+VN* $\not\approx h$.

In contrast, there does exist a GPS network that computes $h(G_{l,r})$ exactly for all $l \in \mathbb{N}_{>0}, r \in \mathbb{N}_{>0}$. A specific set of parameters that achieves this is provided in Rosenbluth et al. (2024). By combining this observation with Theorem 5.8 we obtain that the MPGNN+VN architecture does not subsume GPS in terms of uniform approximation:

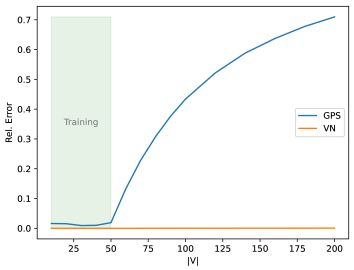
Corollary 5.9. *MPGNNs+VN* $\not\geq^{\{1,2\}}$ *GPS.*

Overall, Corollary 5.5 and Corollary 5.9 state that the two compared architectures are incomparable in the sense that neither subsumes the other with respect to uniform function approximation.

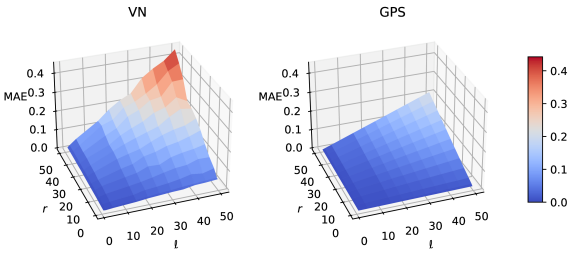
5.2.4 Experiments

We perform two kinds of experiments[2]. First, we empirically verify the difference between GPS and MPGNN+VN architectures using synthetic datasets based on Theorems 5.4 and 5.8. Even when an architecture can theoretically express a target function uniformly, it is not guaranteed that SGD-based optimization can learn an expressing model from data. To investigate this, our synthetic experiments test if a uniformly expressive architecture actually yields models with improved o.o.d. generalization to larger graph sizes after training. Second, we perform experiments on real-world datasets and show that neither architecture strictly outperforms the other overall. On some datasets, a simple Virtual Node is enough to achieve state-of-the-art performance when compared to Graph Transformers. While these experiments are more loosely related to our theoretical results, they

[2] `https://github.com/toenshoff/VN-vs-GT`



(a) Test performance of MPGNN+VN and GPS when computing the square function. We report the relative error. The graphs in the training data were restricted to $|V| \leq 50$.



(b) Test performance of MPGNN+VN and GPS when computing the function $h(G_{l,r})$ from Section 5.2.3.3. The training data is restricted to $l \in [1, 10], r \in [1, 5]$, where l is the number of type-1 vertices and $l \cdot r$ is the number of type-2 vertices.

Figure 5.4: Synthetic experiments confirming the theoretical differences between models in practice

offer another view leading to the general message: Neither GPS nor MPGNN+VN performs strictly better than the other across all learning tasks, and a broad range of architectures should generally be considered.

5.2.4.1 Synthetic Datasets

MPGNN+VN $\not\leq$ GPS From Corollary 5.5 we know that the simple function where the value $|V|^2$ has to be predicted as a graph-level regression target can be uniformly expressed by an MPGNN with VN but not by a Graph Transformer. In order to verify this empirically, we design a dataset consisting of 100k random Erdős-Rényi graphs with size $10 \leq |V| \leq 50$ and a mean degree sampled uniformly from the interval $[1, 5]$. For testing, we use larger graphs of size up to 200 as we are interested in the generalization behavior of the trained models. We train MPGNN+VN and GPS architectures that are identical in width (256 hidden dimensions) and depth (3 layers). For both models, we use a standard GCN layer (Kipf and Welling, 2017) as a message passing module. To the GPS model, we additionally provide LapPE encodings. Figure 5.4a shows the results on the test data. MPGNN+VN has learned to closely predict the target function even for graphs outside the training distribution. GPS does achieve a low relative error for graphs with up to 50 vertices. However, for sizes unseen during training the performance deteriorates rapidly. Thus, GPS was unable to learn a function that generalizes across graph sizes which is in line with Corollary 5.5.

GPS $\not\leq$ MPGNN+VN To empirically study the converse scenario, we generate another synthetic dataset consisting of 100k random graphs $G_{l,r}$ with vertex features as defined in Section 5.2.3.3 and the target function being the graph-level regression objective $h(G_{l,r})$ from Lemma 5.7. Even though the formal definition specifies an empty edge set, we use random Erdős-Rényi graphs with a mean degree in $[5, 10]$, as in the previous experiment. This does not affect the theoretical ability of GPS

Table 5.3: Performance of VN-equipped models compared to standard MPGNNs and Graph Transformers. **First**, **second**, and **third** best values are colored. * indicates methods lacking feature normalization on the computer vision datasets (see Section 5.1).

	Method	PascalVOC-SP Test F1 ↑	COCO-SP Test F1 ↑	Peptides-Func Test AP ↑	Peptides-Struct Test MAE ↓	OGB-Code2 Test F1 ↑	OGB-MolPCBA Test AP ↑
MPGNN	GCN	0.2078 ± 0.0031	0.1338 ± 0.0007	**0.6860 ± 0.0050**	**0.2460 ± 0.0007**	0.1507 ± 0.0018	0.2483 ± 0.0037
	GINE / GIN	0.2718 ± 0.0054	0.2125 ± 0.0009	0.6621 ± 0.0067	**0.2473 ± 0.0017**	0.1495 ± 0.0023	0.2703 ± 0.0023
	GatedGCN	0.3880 ± 0.0040	0.2922 ± 0.0018	0.6765 ± 0.0047	0.2477 ± 0.0009	0.1606 ± 0.0015	**0.3066 ± 0.0013**
GTs	SAN	0.3230 ± 0.0039*	0.2592 ± 0.0158*	0.6439 ± 0.0075	0.2545 ± 0.0012	-	0.2765 ± 0.0042
	GPS	**0.4440 ± 0.0065**	**0.3884 ± 0.0055**	0.6534 ± 0.0091	0.2509 ± 0.0014	**0.1894 ± 0.0024**	0.2907 ± 0.0028
	LGI-GT	-	-	-	-	**0.1948 ± 0.0024**	**0.3040 ± 0.0029**
	Exphormer	**0.3975 ± 0.0037***	**0.3455 ± 0.0009***	0.6527 ± 0.0043	0.2481 ± 0.0007	-	-
VN	GCN-VN	0.2950 ± 0.0058	0.2072 ± 0.0043	**0.6732 ± 0.0066**	0.2505 ± 0.0022	-	-
	GatedGCN-VN	**0.4477 ± 0.0137**	**0.3244 ± 0.0025**	**0.6823 ± 0.0069**	**0.2475 ± 0.0018**	**0.1855 ± 0.0018**	**0.3141 ± 0.0019**

to express the target function $h(G_{l,r})$ but it does allow us to test if the model can learn to ignore the edges in a data-driven manner. During training, we restrict the data distribution to $l \in [1, 10]$ and $r \in [1, 5]$ and increase the range to $l, r \in [1, 50]$ for testing. Again, we train 3-layer MPGNN+VN and GPS models based on GCN layers of identical width. Figure 5.4b provides the experiment's results. For both models, the MAE increases roughly linearly in both l and r. The error of the MPGNN+VN model increases with a significantly steeper slope when compared to that of the GPS model which generalizes better to values of r and l outside the training distribution. This observation generally aligns with Corollary 5.9.

Through these experiments on synthetic data, we showed that the theoretical findings from Section 5.2.3 extend to practice. In both cases, the uniformly expressing architecture was indeed able to learn a model with better o.o.d generalization to larger graphs. Especially in the first case, we observe that a simple function on the number of nodes of a graph was not learned robustly by GPS but was learned by the uniformly expressive MPGNN+VN architecture.

5.2.4.2 Benchmark Datasets

Finally, we compare MPGNNs+VN to Graph Transformers to contrast the two paradigms of global information exchange. For this, we evaluate MPGNNs+VN on a range of real-world benchmark datasets from LRGB (Dwivedi et al., 2022) and OGB (Hu et al., 2020c). Concretely, we use the two graph-level datasets Peptides-Func and Peptides-Struct and the two node-level superpixel datasets Pascal-VOC and COCO from LRGB and the two graph-level datasets code2 and MOLPCBA from OGB. For the LRGB datasets, we use the updated baseline results from Section 5.1 and adopt an identical setting to evaluate MPGNN+VN models. For all datasets, we add LapPE or RWSE features if these improve validation performance. The hyperparameters used are provided in the appendix in Table C.7.

Results Our results are provided in Table 5.3. We can see that on the Peptides datasets, and especially Peptides-Struct, all models are very close and neither Graph

Transformers nor MPGNNs+VN hold a significant edge over simple MPGNNs. On the superpixel datasets PascalVOC-SP and COCO-SP the performance of MPGNNs is significantly strengthened by adding a Virtual Node. On PascalVOC-SP the GatedGCN+VN model is on par with GPS, indicating that the Virtual Node yields similar performance benefits as the self-attention in GPS which also relies on GatedGCN as its MPGNN module on this dataset. On COCO-SP the improvement of adding a VN to MPGNNs is also significant, although not enough to fully close the gap to GPS. The results on ogbg-code2 are similar and adding Virtual Nodes to MPGNNs significantly boosts the performance to almost the level of the GTs. Last, we provide new results for GatedGCN on MOLPCBA (with and without Virtual Nodes) and observe state-of-the-art performance. Note that these models use RWSE features, which improve performance significantly on this molecular task. Virtual Nodes were able to further boost the already very good performance of GatedGCN by about 1% AP which is a highly significant gain over all other methods and even reaches the level of Graphormer (not shown) which was pre-trained on additional data.

Overall, the comparison yields mixed results with neither Graph Transformers nor MPGNNs+VN clearly outperforming the other. This adds to our main theoretical results which state that neither architecture subsumes the other in terms of uniform expressivity. Furthermore, MPGNNs+VN do yield the best results on some datasets, showing that Virtual Nodes can generally compete with self-attention as a method of global information exchange on real-world learning tasks while being significantly more efficient.

5.2.5 Discussion

In the non-uniform setting, we formally proved that strong positional encodings imply universality not only for GTs but also for MPGNNs and even MLPs. We thus looked into uniform approximation where a single model has to work for all graph sizes. There we showed that GTs and MPGNNs with Virtual Nodes express different sets of functions. The core observations were that GTs are unable to perform unbounded aggregation (e.g. counting the nodes) and on the other hand self-attention can compute functions that can not be expressed uniformly by the simple computation of a Virtual Node.

Already for expressivity reasons, the global computation method should therefore be carefully chosen for a given learning task. Through synthetic experiments, we showed that this difference also extends to practice. On real-world datasets from LRGB and OGB we observe that simple and efficient Virtual Nodes are generally competitive with Graph Transformers. We believe that our findings highlight the virtues of uniform function approximation as a notion of expressivity, as it was able to capture the relative strengths and weaknesses of the compared methods, while an analysis in the non-uniform setting was not.

Conclusion

In this thesis, we explored practical and theoretical aspects of the development and analysis of Graph Neural Networks. Let us summarize the main contributions that were presented.

We introduced ANYCSP, a GNN-based approach for learning heuristics for CSPs in a data-driven manner. A key contribution is a general graph representation and GNN architecture that captures all CSPs in a unified approach. We demonstrated that reinforcement learning with standard policy gradient methods is sufficient to learn strong heuristics, despite the large, challenging action space. ANYCSP achieves superior performance compared to prior neural methods and is shown to be competitive with classical solvers on a range of complex combinatorial problems. This highlights the efficacy of GNNs in learning sophisticated algorithms, emphasizing their potential to solve hard tasks in the context of combinatorial optimization.

We further presented CRaWl as a novel GNN architecture that departs from the traditional message passing framework. By utilizing 1D convolutions on sequential features obtained through random walks, CRaWl achieves an expressivity that is provably incomparable to the entire Weisfeiler-Leman hierarchy and therefore most MPGNN architectures. CRaWl has shown strong empirical performance across various tasks, providing a new direction for GNN architecture design beyond the paradigm of message passing.

We conducted an empirical evaluation and theoretical analyses of GNN architectures that aim to capture long-range dependencies in graphs. Our experimental study re-evaluated the performance of standard MPGNNs on long-range benchmark datasets, revealing that they perform significantly better than previously believed. Additionally, we contributed a theoretical comparison between Graph Transformers and MPGNNs with Virtual Nodes in terms of uniform function approximation. There, we prove that neither architecture subsumes the other in terms of uniform expressivity. These results demonstrate that both architectures have their relative strengths and weaknesses and highlight why a wide range of architectures should generally be considered when selecting a method for a specific learning task.

Overall, we believe that the presented work underscores the power and versatility of GNNs as well as the large range of architectural variations that are worth exploring to improve expressivity and empirical performance.

6.1 Directions for Future Work

Finally, let us discuss promising directions for future GNN research that have not yet been covered in the chapter-wise discussions but fit into the general context of the presented work.

The continued development of high-quality benchmark datasets is an essential cornerstone of deep learning research. The study of GNNs is no exception and exploring yet more benchmark tasks for principled comparisons is crucial. In Chapter 5 we already discussed the importance of novel large-scale datasets as a key direction for future work toward understanding potential scaling laws governing various GNN architectures. However, it is not just the study of large-scale or long-range tasks that would benefit from more datasets but practically all aspects of understanding GNNs. For example, a potential focus of new benchmarks could be a direct investigation of generalization by constructing learning tasks with sharp distribution shifts from training to testing with respect to graph size and structure. Generally, for a field that relies as strongly on empirical evaluations as machine learning, the importance of creating and curating quality datasets can not be understated.

A class of GNNs that is arguably understudied are recurrent MPGNNs. Recurrent architectures are not limited to a fixed number of message passes and can therefore propagate information across arbitrary distances if issues such as over-smoothing can be avoided. The main caveat is that the training of recurrent neural networks with backpropagation through time is often more brittle when compared to standard feedforward models. However, in light of the strong results obtained with a recurrent MPGNN in Chapter 3 we argue that this class of GNNs remains underexplored in both theory and practice and may offer a straightforward solution to long-range concerns. Furthermore, RNNs have recently experienced a methodological revival for sequential data with novel RNN architectures such as S4 (Gu et al., 2022), Mamba (Gu and Dao, 2023) or xLSTM (Beck et al., 2024) that have set out to challenge the dominance of Transformers for language modeling tasks. These new architectural ideas may be adaptable to construct improved recurrent MPGNN models.

Further studying the aspect of *learnability* in the context of GNNs may be another key direction of future work. Many theoretical results on the expressivity of GNNs make statements on whether or not there exists some parameterization of a given GNN model that computes a certain target function. Few formal results exist on when such an expressing parameterization can reliably be learned with stochastic gradient descent. Empirical experimentation with synthetic and real-world datasets can provide some insights into this question. However, gaining a more rigorous

formal understanding of the conditions under which commonly used optimizers learn specific parameterizations would be a crucial step toward bridging the gap between the theory and the practice of GNNs.

In conclusion, the wealth of applications and angles for theoretical analysis leaves no shortage of promising future directions for studying Graph Neural Networks. Undoubtedly, the field is wide.

Bibliography

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

Thomas Gaudelet, Ben Day, Arian R Jamasb, Jyothish Soman, Cristian Regep, Gertrude Liu, Jeremy B R Hayter, Richard Vickers, Charles Roberts, Jian Tang, David Roblin, Tom L Blundell, Michael M Bronstein, and Jake P Taylor-King. Utilizing graph machine learning within drug discovery and development. *Briefings in Bioinformatics*, 22(6):bbab159, 05 2021. ISSN 1477-4054. doi: 10.1093/bib/bbab159. URL https://doi.org/10.1093/bib/bbab159.

Remi Lam, Alvaro Sanchez-Gonzalez, Matthew Willson, Peter Wirnsberger, Meire Fortunato, Ferran Alet, Suman Ravuri, Timo Ewalds, Zach Eaton-Rosen, Weihua Hu, Alexander Merose, Stephan Hoyer, George Holland, Oriol Vinyals, Jacklynn Stott, Alexander Pritzel, Shakir Mohamed, and Peter Battaglia. Learning skillful medium-range global weather forecasting. *Science*, 382(6677):1416–1421, 2023. doi: 10.1126/science.adi2336. URL https://www.science.org/doi/abs/10.1126/science.adi2336.

Quentin Cappart, Didier Chételat, Elias B. Khalil, Andrea Lodi, Christopher Morris, and Petar Veličković. Combinatorial optimization and reasoning with graph neural networks. In Zhi-Hua Zhou, editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pages 4348–4355. International Joint Conferences on Artificial Intelligence Organization, 8 2021. doi: 10.24963/ijcai.2021/595. URL https://doi.org/10.24963/ijcai.2021/595. Survey Track.

Simon Ståhlberg, Blai Bonet, and Hector Geffner. Learning general optimal policies with graph neural networks: Expressive power, transparency, and limits. *Proceedings of the International Conference on Automated Planning and Scheduling*, 32(1):629–637, Jun. 2022. doi: 10.1609/icaps.v32i1.19851. URL https://ojs.aaai.org/index.php/ICAPS/article/view/19851.

Shiwen Wu, Fei Sun, Wentao Zhang, Xu Xie, and Bin Cui. Graph neural networks in recommender systems: A survey. *ACM Comput. Surv.*, 55(5), dec 2022. ISSN 0360-0300. doi: 10.1145/3535101. URL https://doi.org/10.1145/3535101.

Yang Liu, Xiang Ao, Zidi Qin, Jianfeng Chi, Jinghua Feng, Hao Yang, and Qing He. Pick and choose: a gnn-based imbalanced learning approach for fraud detection. In *Proceedings of the web conference 2021*, pages 3168–3177, 2021.

Michihiro Yasunaga, Hongyu Ren, Antoine Bosselut, Percy Liang, and Jure Leskovec. Qa-gnn: Reasoning with language models and knowledge graphs for question answering. *arXiv preprint arXiv:2104.06378*, 2021.

Milan Cvitkovic. Supervised learning on relational databases with graph neural networks. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009. doi: 10.1109/TNN.2008.2005605.

Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, 1989. doi: 10.1162/neco.1989.1.4.541.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep sets. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL `https://proceedings.neurips.cc/paper_files/paper/2017/file/f22e4747da1aa27e363d86d40ff442fe-Paper.pdf`.

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

Minjie Wang, Da Zheng, Zihao Ye, Quan Gan, Mufei Li, Xiang Song, Jinjing Zhou, Chao Ma, Lingfan Yu, Yu Gai, Tianjun Xiao, Tong He, George Karypis, Jinyang Li, and Zheng Zhang. Deep graph library: A graph-centric, highly-performant package for graph neural networks. *arXiv preprint arXiv:1909.01315*, 2019.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.

Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. Graphsaint: Graph sampling based inductive learning method. *arXiv preprint arXiv:1907.04931*, 2019.

Ziniu Hu, Yuxiao Dong, Kuansan Wang, and Yizhou Sun. Heterogeneous graph transformer. In *Proceedings of the web conference 2020*, pages 2704–2710, 2020a.

Hanqing Zeng, Muhan Zhang, Yinglong Xia, Ajitesh Srivastava, Andrey Malevich, Rajgopal Kannan, Viktor Prasanna, Long Jin, and Ren Chen. Decoupling the depth and scope of graph neural networks. *Advances in Neural Information Processing Systems*, 34:19665–19679, 2021.

Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI Open*, 1:57–81, 2020. ISSN 2666-6510. doi: https://doi.org/10.1016/j.aiopen.2021.01.001. URL https://www.sciencedirect.com/science/article/pii/S2666651021000012.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*, 2017.

David K Hammond, Pierre Vandergheynst, and Rémi Gribonval. Wavelets on graphs via spectral graph theory. *Applied and Computational Harmonic Analysis*, 30(2):129–150, 2011.

Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

Uri Alon and Eran Yahav. On the bottleneck of graph neural networks and its practical implications. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=i80OPhOCVH2.

Guohao Li, Matthias Muller, Ali Thabet, and Bernard Ghanem. Deepgcns: Can gcns go as deep as cnns? In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 9267–9276, 2019.

Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. Simple and deep graph convolutional networks. In *International conference on machine learning*, pages 1725–1735. PMLR, 2020a.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *Proceedings of the Seventh International Conference on Learning Representations (ICLR)*, 2019.

Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay Pande, and Jure Leskovec. Strategies for pre-training graph neural networks. In *International Conference on Learning Representations*, 2020b. URL https://openreview.net/forum?id=HJlWWJSFDH.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1409.0473.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

Shaked Brody, Uri Alon, and Eran Yahav. How attentive are graph attention networks? In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=F72ximsx7C1.

Yunsheng Shi, Zhengjie Huang, Shikun Feng, Hui Zhong, Wenjing Wang, and Yu Sun. Masked label prediction: Unified message passing model for semi-supervised classification. In Zhi-Hua Zhou, editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pages 1548–1554. International Joint Conferences on Artificial Intelligence Organization, 8 2021. doi: 10.24963/ijcai.2021/214. URL https://doi.org/10.24963/ijcai.2021/214. Main Track.

Vijay Prakash Dwivedi and Xavier Bresson. A generalization of transformer networks to graphs. *AAAI Workshop on Deep Learning on Graphs: Methods and Applications*, 2021.

Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 11 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL https://doi.org/10.1162/neco.1997.9.8.1735.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

Alessandro Sperduti and Antonina Starita. Supervised neural networks for the classification of structures. *IEEE Transactions on Neural Networks*, 8(3): 714 – 735, 1997. doi: 10.1109/72.572108. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-0031145983&doi=10.1109%2f72.572108&partnerID=40&md5=7e42838d416dd063240a58829d5eaa4d. Cited by: 489.

M. Gori, G. Monfardini, and F. Scarselli. A new model for learning in graph domains. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 2, pages 729–734 vol. 2, 2005. doi: 10.1109/IJCNN.2005.1555942.

Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*, 2015.

Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *The semantic web: 15th international conference, ESWC 2018, Heraklion, Crete, Greece, June 3–7, 2018, proceedings 15*, pages 593–607. Springer, 2018.

Wang Xiao, Ji Houye, Shi Chuan, Wang Bai, Cui Peng, Yu P., and Ye Yanfang. Heterogeneous graph attention network. *WWW*, 2019.

Chuan Shi, Xiao Wang, and S Yu Philip. *Heterogeneous Graph Representation Learning and Applications*. Springer, 2022.

Qingsong Lv, Ming Ding, Qiang Liu, Yuxiang Chen, Wenzheng Feng, Siming He, Chang Zhou, Jianguo Jiang, Yuxiao Dong, and Jie Tang. Are we really making much progress? revisiting, benchmarking and refining heterogeneous graph neural networks. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, KDD '21, page 1150–1160, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450383325. doi: 10.1145/3447548.3467350. URL https://doi.org/10.1145/3447548.3467350.

Christopher Morris, Martin Ritzert, Matthias Fey, William L. Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and Leman go neural: Higher-order graph neural networks. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence (AAAI)*, pages 4602–4609, 2019.

Pablo Barceló, Egor V Kostylev, Mikael Monet, Jorge Pérez, Juan Reutter, and Juan-Pablo Silva. The logical expressiveness of graph neural networks. In *8th International Conference on Learning Representations (ICLR 2020)*, 2020.

M. Grohe. The logic of graph neural networks. In *Symposium on Logic in Computer Science*, pages 1–17, 2021a.

Martin Grohe. The descriptive complexity of graph neural networks. In *2023 38th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–14. IEEE, 2023.

Eran Rosenbluth, Jan Tönshoff, and Martin Grohe. Some might say all you need is sum. In Edith Elkind, editor, *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI-23*, pages 4172–4179. International Joint Conferences on Artificial Intelligence Organization, 8 2023. doi: 10.24963/ijcai.2023/464. URL https://doi.org/10.24963/ijcai.2023/464. Main Track.

Christopher Morris, Floris Geerts, Jan Tönshoff, and Martin Grohe. WL meet VC. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 25275–25302. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/morris23a.html.

Laszlo Babai and Ludik Kucera. Canonical labelling of graphs in linear average time. In *20th Annual Symposium on Foundations of Computer Science (sfcs 1979)*, pages 39–46, 1979. doi: 10.1109/SFCS.1979.8.

J. Cai, M. Fürer, and N. Immerman. An optimal lower bound on the number of variables for graph identification. *Combinatorica*, 12:389–410, 1992.

Martin Grohe and Martin Otto. Pebble games and linear equations. *The Journal of Symbolic Logic*, 80(3):797–844, 2015.

Ningyuan Teresa Huang and Soledad Villar. A short tutorial on the weisfeiler-lehman test and its variants. In *ICASSP 2021 - 2021 IEEE International*

Conference on Acoustics, Speech and Signal Processing (ICASSP), pages 8533–8537, 2021. doi: 10.1109/ICASSP39728.2021.9413523.

Haggai Maron, Heli Ben-Hamu, Hadar Serviansky, and Yaron Lipman. Provably powerful graph networks. pages 2153–2164, 2019.

Floris Geerts. The expressive power of kth-order invariant graph networks. *arXiv preprint arXiv:2007.12035*, 2020a.

Waiss Azizian and Marc Lelarge. Expressive power of invariant and equivariant graph neural networks. *arXiv preprint arXiv:2006.15646*, 2020.

Christopher Morris, Gaurav Rattan, Sandra Kiefer, and Siamak Ravanbakhsh. Speqnets: Sparsity-aware permutation-equivariant graph networks. In *International Conference on Machine Learning*, pages 16017–16042. PMLR, 2022.

Nino Shervashidze, Pascal Schweitzer, Erik Jan Van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(9), 2011.

Gabriele Corso, Luca Cavalleri, Dominique Beaini, Pietro Liò, and Petar Velickovic. Principal neighbourhood aggregation for graph nets. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL `https://proceedings.neurips.cc/paper/2020/hash/99cad265a1768cc2dd013f0e740300ae-Abstract.html`.

V. N. Vapnik. *The Nature of Statistical Learning Theory*. Springer, 1995.

M. Mohri, A. Rostamizadeh, and A. Talwalkar. *Foundations of Machine Learning*. MIT Press, 2018.

Ladislav Rampášek, Michael Galkin, Vijay Prakash Dwivedi, Anh Tuan Luu, Guy Wolf, and Dominique Beaini. Recipe for a general, powerful, scalable graph transformer. *Advances in Neural Information Processing Systems*, 35: 14501–14515, 2022.

Vijay Prakash Dwivedi, Anh Tuan Luu, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Graph neural networks with learnable structural and positional representations. In *International Conference on Learning Representations*, 2021.

Giorgos Bouritsas, Fabrizio Frasca, Stefanos Zafeiriou, and Michael M Bronstein. Improving graph neural network expressivity via subgraph isomorphism counting. *arXiv preprint arXiv:2006.09252*, 2020.

Aravind Sankar, Xinyang Zhang, and Kevin Chen-Chuan Chang. Motif-based convolutional neural network on graphs. *arXiv preprint arXiv:1711.05697*, 2017.

John Boaz Lee, Ryan A Rossi, Xiangnan Kong, Sungchul Kim, Eunyee Koh, and Anup Rao. Graph convolutional networks with motif-based attention. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pages 499–508, 2019.

Hao Peng, Jianxin Li, Qiran Gong, Yuanxin Ning, Senzhang Wang, and Lifang He. Motif-matching based subgraph-level attentional convolutional network for graph classification. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 5387–5394, 2020.

Vijay Prakash Dwivedi, Chaitanya K Joshi, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Benchmarking graph neural networks. *arXiv preprint arXiv:2003.00982*, 2020.

Devin Kreuzer, Dominique Beaini, Will Hamilton, Vincent Létourneau, and Prudencio Tossou. Rethinking graph transformers with spectral attention. *Advances in Neural Information Processing Systems*, 34:21618–21629, 2021.

Haorui Wang, Haoteng Yin, Muhan Zhang, and Pan Li. Equivariant and stable positional encoding for more powerful graph neural networks. In *International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=e95i1IHcWj`.

Derek Lim, Joshua David Robinson, Lingxiao Zhao, Tess Smidt, Suvrit Sra, Haggai Maron, and Stefanie Jegelka. Sign and basis invariant networks for spectral graph representation learning. In *The Eleventh International Conference on Learning Representations*, 2023. URL `https://openreview.net/forum?id=Q-UHqMorzil`.

Yinan Huang, William Lu, Joshua Robinson, Yu Yang, Muhan Zhang, Stefanie Jegelka, and Pan Li. On the stability of expressive positional encodings for graphs. In *The Twelfth International Conference on Learning Representations*, 2024. URL `https://openreview.net/forum?id=xAqcJ9XoTf`.

Rickard Brüel Gabrielsson, Mikhail Yurochkin, and Justin Solomon. Rewiring with positional encodings for graph neural networks. *Transactions on Machine Learning Research*, 2023. ISSN 2835-8856. URL `https://openreview.net/forum?id=dn3ZkqG2YV`.

Qingyun Sun, Jianxin Li, Haonan Yuan, Xingcheng Fu, Hao Peng, Cheng Ji, Qian Li, and Philip S Yu. Position-aware structure learning for graph topology-imbalance by relieving under-reaching and over-squashing. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, pages 1848–1857, 2022.

Francesco Di Giovanni, Lorenzo Giusti, Federico Barbero, Giulia Luise, Pietro Lio, and Michael M. Bronstein. On over-squashing in message passing neural networks: The impact of width, depth, and topology. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *Proceedings of the 40th International Conference on Machine*

Learning, volume 202 of *Proceedings of Machine Learning Research*, pages 7865–7885. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/di-giovanni23a.html.

Kenta Oono and Taiji Suzuki. Graph neural networks exponentially lose expressive power for node classification. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=S1ldO2EFPr.

Chen Cai and Yusu Wang. A note on over-smoothing for graph neural networks. *arXiv preprint arXiv:2006.13318*, 2020.

Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020c. URL https://proceedings.neurips.cc/paper/2020/hash/fb60d411a5c5b72b2e7d3527cfc84fd0-Abstract.html.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=YicbFdNTTy.

Anurag Arnab, Mostafa Dehghani, Georg Heigold, Chen Sun, Mario Lučić, and Cordelia Schmid. Vivit: A video vision transformer. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 6836–6846, 2021.

Alexei Baevski, Yuhao Zhou, Abdelrahman Mohamed, and Michael Auli. wav2vec 2.0: A framework for self-supervised learning of speech representations. *Advances in neural information processing systems*, 33:12449–12460, 2020.

Luis Müller, Mikhail Galkin, Christopher Morris, and Ladislav Rampášek. Attending to graph transformers. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856. URL https://openreview.net/forum?id=HhbqHBBrfZ.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. pmlr, 2015.

Hamed Shirzad, Ameya Velingker, Balaji Venkatachalam, Danica J Sutherland, and Ali Kemal Sinop. Exphormer: Sparse transformers for graphs. In *International Conference on Machine Learning*, 2023.

Shuo Yin and Guoqiang Zhong. Lgi-gt: Graph transformers with local and global operators interleaving. In Edith Elkind, editor, *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI-23*, pages 4504–4512. International Joint Conferences on Artificial Intelligence Organization, 8 2023. doi: 10.24963/ijcai.2023/501. URL https://doi.org/10.24963/ijcai.2023/501. Main Track.

Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and Tie-Yan Liu. Do transformers really perform badly for graph representation? *Advances in Neural Information Processing Systems*, 34: 28877–28888, 2021.

Weihua Hu, Matthias Fey, Hongyu Ren, Maho Nakata, Yuxiao Dong, and Jure Leskovec. Ogb-lsc: A large-scale challenge for machine learning on graphs. *arXiv preprint arXiv:2103.09430*, 2021.

Liheng Ma, Chen Lin, Derek Lim, Adriana Romero-Soriano, K. Dokania, Mark Coates, Philip H.S. Torr, and Ser-Nam Lim. Graph Inductive Biases in Transformers without Message Passing. In *Proc. Int. Conf. Mach. Learn.*, 2023.

Dexiong Chen, Leslie O'Bray, and Karsten Borgwardt. Structure-aware transformer for graph representation learning. In *International Conference on Machine Learning*, pages 3469–3489. PMLR, 2022.

Xiaoxin He, Bryan Hooi, Thomas Laurent, Adam Perold, Yann Lecun, and Xavier Bresson. A generalization of ViT/MLP-mixer to graphs. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 12724–12745. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/he23a.html.

Ilya O Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Andreas Steiner, Daniel Keysers, Jakob Uszkoreit, et al. Mlp-mixer: An all-mlp architecture for vision. *Advances in neural information processing systems*, 34:24261–24272, 2021.

Jinwoo Kim, Dat Nguyen, Seonwoo Min, Sungjun Cho, Moontae Lee, Honglak Lee, and Seunghoon Hong. Pure transformers are powerful graph learners. *Advances in Neural Information Processing Systems*, 35:14582–14595, 2022.

Ralph Abboud, Radoslav Dimitrov, and Ismail Ilkan Ceylan. Shortest path networks for graph property prediction. In *The First Learning on Graphs Conference*, 2022. URL https://openreview.net/forum?id=mWzWvMxuFg1.

Adrián Arnaiz-Rodríguez, Ahmed Begga, Francisco Escolano, and Nuria M Oliver. Diffwire: Inductive graph rewiring via the lovász bound. In *The First Learning on Graphs Conference*, 2022. URL https://openreview.net/forum?id=IXvfIex0mX6f.

Kedar Karhadkar, Pradeep Kr. Banerjee, and Guido Montufar. FoSR: First-order spectral rewiring for addressing oversquashing in GNNs. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=3YjQfCLdrzz.

Benjamin Gutteridge, Xiaowen Dong, Michael M. Bronstein, and Francesco Di Giovanni. DRew: Dynamically rewired message passing with delay. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 12252–12267. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/gutteridge23a.html.

Jan Tönshoff, Berke Kisin, Jakob Lindner, and Martin Grohe. One model, any csp: Graph neural networks as fast global search heuristics for constraint satisfaction. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI-23*, pages 4280–4288, 8 2023a. doi: 10.24963/ijcai.2023/476. URL https://doi.org/10.24963/ijcai.2023/476.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. *Advances in neural information processing systems*, 28, 2015.

Elias Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. *Advances in neural information processing systems*, 30, 2017.

Daniel Selsam, Matthew Lamm, Benedikt Bünz, Percy Liang, Leonardo de Moura, and David L Dill. Learning a sat solver from single-bit supervision. *arXiv preprint arXiv:1802.03685*, 2018.

Maxime Gasse, Didier Chételat, Nicola Ferroni, Laurent Charlin, and Andrea Lodi. Exact combinatorial optimization with graph convolutional neural networks. *Advances in neural information processing systems*, 32, 2019.

Jan Tönshoff, Martin Ritzert, Hinrikus Wolf, and Martin Grohe. Graph neural networks for maximum constraint satisfaction. *Frontiers in Artificial Intelligence*, 3, 2021. ISSN 2624-8212. doi: 10.3389/frai.2020.580607. URL https://www.frontiersin.org/article/10.3389/frai.2020.580607.

S. Russell, S.J. Russell, P. Norvig, and E. Davis. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 4th edition, 2020. ISBN 9780136042594.

A. Biere, M. Heule, H. Van Maaren, and T. Walsh. *Handbook of Satisfiability*. IOS Press, 2nd edition, 2021. ISBN 978-1-64368-160-3.

Jan Peters, Sethu Vijayakumar, and Stefan Schaal. Reinforcement learning for humanoid robotics. In *Proceedings of the third IEEE-RAS international conference on humanoid robots*, pages 1–20, 2003.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744, 2022.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

Christopher John Cornish Hellaby Watkins. Learning from delayed rewards. 1989.

Longxin Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 8:293–321, 1992. URL `https://api.semanticscholar.org/CorpusID:3248358`.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256, 1992.

Thomas Degris, Martha White, and Richard S Sutton. Off-policy actor-critic. *arXiv preprint arXiv:1205.4839*, 2012.

Hanjun Dai, Bo Dai, and Le Song. Discriminative embeddings of latent variable models for structured data. In *International conference on machine learning*, pages 2702–2711. PMLR, 2016.

Martin Riedmiller. Neural fitted q iteration – first experiences with a data efficient neural reinforcement learning method. In João Gama, Rui Camacho, Pavel B. Brazdil, Alípio Mário Jorge, and Luís Torgo, editors, *Machine Learning: ECML 2005*, pages 317–328, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. ISBN 978-3-540-31692-3.

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

Thomas D Barrett, Christopher WF Parsonson, and Alexandre Laterre. Learning to solve combinatorial graph partitioning problems via efficient exploration. *arXiv preprint arXiv:2205.14105*, 2022.

Emre Yolcu and Barnabás Póczos. Learning local search heuristics for boolean satisfiability. *Advances in Neural Information Processing Systems*, 32, 2019.

Bart Selman, Henry A Kautz, Bram Cohen, et al. Local search strategies for satisfiability testing. *Cliques, coloring, and satisfiability*, 26:521–532, 1993.

Ralph Abboud, İsmail İlkan Ceylan, Martin Grohe, and Thomas Lukasiewicz. The surprising power of graph neural networks with random node initialization. In Zhi-Hua Zhou, editor, *Proceedings of the 30th International Joint Conference on Artificial Intelligence*, pages 2112–2118, 2021a. doi: 10.24963/ijcai.2021/291.

Ryoma Sato, Makoto Yamada, and Hisashi Kashima. Random features strengthen graph neural networks. In *Proceedings of the 2021 SIAM International Conference on Data Mining (SDM)*, pages 333–341. SIAM, 2021.

Chaitanya K. Joshi, Quentin Cappart, Louis-Martin Rousseau, Thomas Laurent, and Xavier Bresson. Learning TSP requires rethinking generalization. *CoRR*, abs/2006.07054, 2020. URL `https://arxiv.org/abs/2006.07054`.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Ke Xu and Wei Li. Many hard examples in exact phase transitions. *Science Direct Working Paper No S1574-034X (04)*, pages 70228–8, 2003.

Gilles Audemard, Frédéric Boussemart, Christophe Lecoutre, Cédric Piette, and Olivier Roussel. Xcsp3 and its ecosystem. *Constraints*, 25(1):47–69, 2020.

Neng-Fa Zhou. An xcsp3 solver in picat. In *XCSP3 Competition 2022 Proceedings*, XCSP3 Competition, pages 79–81, 2022.

Christophe Lecoutre. Ace a generic constraint solver. In *XCSP3 Competition 2022 Proceedings*, XCSP3 Competition, pages 58–59, 2022.

Gilles Audemard. Cosoco 1.12. In *XCSP3 Competition 2018 Proceedings*, XCSP3 Competition, pages 78–79, 2018.

Aric Hagberg, Pieter Swart, and Daniel S Chult. Exploring network structure, dynamics, and function using networkx. Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008.

Daniel Brélaz. New methods to color the vertices of a graph. *Communications of the ACM*, 22(4):251–256, 1979.

Philippe Galinier and Jin-Kao Hao. Hybrid evolutionary algorithms for graph coloring. *Journal of combinatorial optimization*, 3(4):379–397, 1999.

Rhyd Lewis, Jonathan Thompson, Christine Mumford, and Jonathan Gillard. A wide-ranging computational comparison of high-performance graph colouring algorithms. *Computers & Operations Research*, 39(9):1933–1950, 2012.

Rhyd Lewis. *A guide to graph colouring*, volume 7. Springer, 2015.

Una Benlic and Jin-Kao Hao. Breakout local search for the max-cutproblem. *Engineering Applications of Artificial Intelligence*, 26(3):1162 – 1173, 2013. ISSN 0952-1976. doi: https://doi.org/10.1016/j.engappai.2012.09.001. URL `http://www.sciencedirect.com/science/article/pii/S0952197612002175`.

Michel X Goemans and David P Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM (JACM)*, 42(6):1115–1145, 1995.

Saeed Amizadeh, Sergiy Matusevych, and Markus Weimer. Pdp: A general neural framework for learning constraint satisfaction solvers. *arXiv preprint arXiv:1903.01969*, 2019.

Adrian Balint and Uwe Schöning. probsat. In *Proceedings of SAT Competition 2018: Solver and Benchmark Descriptions*. Department of Computer Science, University of Helsinki, 2018.

Chuan Luo, Shaowei Cai, Wei Wu, Zhong Jie, and Kaile Su. Ccls: An efficient local search algorithm for weighted maximum satisfiability. *IEEE Transactions on Computers*, 64(7):1830–1843, 2015. doi: 10.1109/TC.2014.2346196.

Shaowei Cai and Zhendong Lei. Old techniques in new ways: Clause weighting, unit propagation and hybridization for maximum satisfiability. *Artificial Intelligence*, 287:103354, 2020.

Jan Tönshoff, Martin Ritzert, Hinrikus Wolf, and Martin Grohe. Walking out of the weisfeiler leman hierarchy: Graph learning beyond message passing. *Transactions on Machine Learning Research*, 2023b. ISSN 2835-8856. URL `https://openreview.net/forum?id=vgXnEyeWVY`.

Nino Shervashidze, SVN Vishwanathan, Tobias Petri, Kurt Mehlhorn, and Karsten Borgwardt. Efficient graphlet kernels for large graph comparison. In *Artificial intelligence and statistics*, pages 488–495. PMLR, 2009a.

Fabrizio Frasca, Beatrice Bevilacqua, Michael M. Bronstein, and Haggai Maron. Understanding and extending subgraph GNNs by rethinking their symmetries. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022. URL `https://openreview.net/forum?id=sc7bBHAmcN`.

Di Jin, Rui Wang, Meng Ge, Dongxiao He, Xiang Li, Wei Lin, and Weixiong Zhang. Raw-gnn: Random walk aggregation based graph neural network. In Lud De Raedt, editor, *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22*, pages 2108–2114. International Joint Conferences on Artificial Intelligence Organization, 7 2022. doi: 10.24963/ijcai.2022/293. URL `https://doi.org/10.24963/ijcai.2022/293`. Main Track.

Pablo Barceló, Floris Geerts, Juan L. Reutter, and Maksimilian Ryschkov. Graph neural networks with local graph parameters. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 25280–25293, 2021.

Leonardo Cotta, Christopher Morris, and Bruno Ribeiro. Reconstruction for powerful graph representations. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 1713–1726, 2021.

Matthias Fey, Jan-Gin Yuen, and Frank Weichert. Hierarchical inter-message passing for learning on molecular graphs. *arXiv preprint arXiv:2006.12179*, 2020.

Rémy Brossard, Oriel Frigo, and David Dehaene. Graph convolutions that can finally model local structure. *arXiv preprint arXiv:2011.15069*, 2020.

Dominique Beaini, Saro Passaro, Vincent Létourneau, William L. Hamilton, Gabriele Corso, and Pietro Lió. Directional graph networks. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 748–758. PMLR, 2021. URL `http://proceedings.mlr.press/v139/beani21a.html`.

Cristian Bodnar, Fabrizio Frasca, Nina Otter, Yuguang Wang, Pietro Liò, Guido F Montufar, and Michael Bronstein. Weisfeiler and lehman go cellular: Cw networks. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 2625–2640. Curran Associates, Inc., 2021. URL `https://proceedings.neurips.cc/paper/2021/file/157792e4abb490f99dbd738483e0d2d4-Paper.pdf`.

Aravind Sankar, Junting Wang, Adit Krishnan, and Hari Sundaram. Beyond localized graph neural networks: An attributed motif regularization framework. In Claudia Plant, Haixun Wang, Alfredo Cuzzocrea, Carlo Zaniolo, and Xindong Wu, editors, *20th IEEE International Conference on Data Mining, ICDM 2020, Sorrento, Italy, November 17-20, 2020*, pages 472–481. IEEE, 2020. doi: 10.1109/ICDM50108.2020.00056. URL `https://doi.org/10.1109/ICDM50108.2020.00056`.

Xuan Jiang, Zhiyong Yang, Peisong Wen, Li Su, and Qingming Huang. A sparse-motif ensemble graph convolutional network against over-smoothing. In Lud De Raedt, editor, *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22*, pages 2094–2100. International Joint Conferences on Artificial Intelligence Organization, 7 2022. doi: 10.24963/ijcai.2022/291. URL `https://doi.org/10.24963/ijcai.2022/291`. Main Track.

M. Niepert, M. Ahmed, and K. Kutzkov. Learning convolutional neural networks for graphs. In *International Conference on Machine Learning*, pages 2014–2023, 2016.

Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. An end-to-end deep learning architecture for graph classification. In *AAAI*, pages 4438–4445, 2018.

Hao Yuan and Shuiwang Ji. Node2seq: Towards trainable convolutions in graph neural networks. *arXiv preprint arXiv:2101.01849*, 2021.

Giannis Nikolentzos and Michalis Vazirgiannis. Random walk graph neural networks. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

Floris Geerts. Walk message passing neural networks and second-order graph neural networks. *arXiv preprint arXiv:2006.09499*, 2020b.

Moshe Eliasof, Eldad Haber, and Eran Treister. pathgcn: Learning general graph spatial operators from paths. In *International Conference on Machine Learning*, pages 5878–5891. PMLR, 2022.

Zheng Ma, Junyu Xuan, Yu Guang Wang, Ming Li, and Pietro Liò. Path integral based convolution and pooling for graph neural networks. *Advances in Neural Information Processing Systems*, 33:16421–16433, 2020.

Sergey Ivanov and Evgeny Burnaev. Anonymous walk embeddings. In *International conference on machine learning*, pages 2186–2195. PMLR, 2018.

Yanbang Wang, Yen-Yu Chang, Yunyu Liu, Jure Leskovec, and Pan Li. Inductive representation learning in temporal networks via causal anonymous walks. *arXiv preprint arXiv:2101.05974*, 2021.

Haoteng Yin, Muhan Zhang, Yanbang Wang, Jianguo Wang, and Pan Li. Algorithm and system co-design for efficient subgraph-based graph representation learning. *arXiv preprint arXiv:2202.13538*, 2022.

Muhan Zhang and Pan Li. Nested graph neural networks. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 15734–15747. Curran Associates, Inc., 2021. URL `https://proceedings.neurips.cc/paper/2021/file/8462a7c229aea03dde69da754c3bbcc4-Paper.pdf`.

Beatrice Bevilacqua, Fabrizio Frasca, Derek Lim, Balasubramaniam Srinivasan, Chen Cai, Gopinath Balamurugan, Michael M. Bronstein, and Haggai Maron. Equivariant subgraph aggregation networks. In *International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=dFbKQaRk15w`.

Lingxiao Zhao, Wei Jin, Leman Akoglu, and Neil Shah. From stars to subgraphs: Uplifting any GNN with local structure awareness. In *International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=Mspk_WYKoEH`.

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

N. Shervashidze, S. Vishwanathan, T. Petri, K. Mehlhorn, and K. Borgwardt. Efficient graphlet kernels for large graph comparison. In *Artificial Intelligence and Statistics*, pages 488–495, 2009b.

Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In Balaji Krishnapuram, Mohak Shah, Alexander J. Smola, Charu C. Aggarwal, Dou Shen, and Rajeev Rastogi, editors, *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, pages 855–864. ACM, 2016. doi: 10.1145/2939672.2939754.

Silvio Micali and Zeyuan Allen Zhu. Reconstructing markov processes from independent and anonymous experiments. *Discrete Applied Mathematics*, 200: 108–122, 2016.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

Francois Chollet. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

Christopher Morris, Gaurav Rattan, and Petra Mutzel. Weisfeiler and leman go sparse: Towards scalable higher-order graph embeddings. *Advances in Neural Information Processing Systems*, 33:21824–21840, 2020a.

Ralph Abboud, İsmail İlkan Ceylan, Martin Grohe, and Thomas Lukasiewicz. The surprising power of graph neural networks with random node initialization. In Zhi-Hua Zhou, editor, *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, 2021b. doi: 10.24963/ijcai.2021/291. URL `https://doi.org/10.24963/ijcai.2021/291`.

Ryoma Sato, Makoto Yamada, and Hisashi Kashima. Random features strengthen graph neural networks. *CoRR*, abs/2002.03155, 2020.

Uri Alon. Network motifs: theory and experimental approaches. *Nature Reviews Genetics*, 8:450–461, 2007. doi: 10.1038/nrg2102.

Martin Grohe. The logic of graph neural networks. pages 1–17, 2021b. doi: 10.1109/LICS52264.2021.9470677. URL https://doi.org/10.1109/LICS52264.2021.9470677.

Sandra Kiefer. The Weisfeiler-Leman algorithm: An exploration of its power. *ACM SIGLOG News*, 7(3):5–27, 2020.

R. Aleliunas, R.M. Karp, R.J. Lipton, L. Lovász, and C. Rackoff. Random walks, universal traversal sequences, and the complexity of maze problem. In *FOCS79*, pages 218–223, 1979.

Vijay Prakash Dwivedi, Ladislav Rampášek, Michael Galkin, Ali Parviz, Guy Wolf, Anh Tuan Luu, and Dominique Beaini. Long range graph benchmark. *Advances in Neural Information Processing Systems*, 35:22326–22340, 2022.

Christopher Morris, Nils M. Kriege, Franka Bause, Kristian Kersting, Petra Mutzel, and Marion Neumann. TUDataset: A collection of benchmark datasets for learning with graphs. In *ICML 2020 Workshop on Graph Representation Learning and Beyond (GRL+ 2020)*, 2020b. URL www.graphlearning.io.

Zhenqin Wu, Bharath Ramsundar, Evan N Feinberg, Joseph Gomes, Caleb Geniesse, Aneesh S Pappu, Karl Leswing, and Vijay Pande. MoleculeNet: a benchmark for molecular machine learning. *Chemical science*, 9(2):513–530, 2018.

Ryan Murphy, Balasubramaniam Srinivasan, Vinayak Rao, and Bruno Ribeiro. Relational pooling for graph representations. In *International Conference on Machine Learning*, pages 4663–4673. PMLR, 2019.

Zhengdao Chen, Lei Chen, Soledad Villar, and Joan Bruna. Can graph neural networks count substructures? In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 10383–10395. Curran Associates, Inc., 2020b. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/75877cb75154206c4e65e76b88a12712-Paper.pdf.

Jan Tönshoff, Martin Ritzert, Eran Rosenbluth, and Martin Grohe. Where did the gap go? reassessing the long-range graph benchmark. 2024. URL https://openreview.net/forum?id=Nm0WX86sKv.

Eran Rosenbluth, Jan Tönshoff, Martin Ritzert, Berke Kisin, and Martin Grohe. Distinguished in uniform: Self-attention vs. virtual nodes. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=AcSChDWL6V.

Xavier Bresson and Thomas Laurent. Residual gated graph convnets. *arXiv preprint arXiv:1711.07553*, 2017.

Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

Jiaxuan You, Zhitao Ying, and Jure Leskovec. Design space for graph neural networks. *Advances in Neural Information Processing Systems*, 33:17009–17021, 2020.

Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In C.J. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc., 2013. URL https://proceedings.neurips.cc/paper_files/paper/2013/file/1cecc7a77928ca8133fa24680a88d2f9-Paper.pdf.

Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. In *International conference on machine learning*, pages 10347–10357. PMLR, 2021.

Dominique Beaini, Shenyang Huang, Joao Alex Cunha, Zhiyi Li, Gabriela Moisescu-Pareja, Oleksandr Dymov, Samuel Maddrell-Mander, Callum McLean, Frederik Wenkel, Luis Müller, Jama Hussein Mohamud, Ali Parviz, Michael Craig, Michał Koziarski, Jiarui Lu, Zhaocheng Zhu, Cristian Gabellini, Kerstin Klaser, Josef Dean, Cas Wognum, Maciej Sypetkowski, Guillaume Rabusseau, Reihaneh Rabbany, Jian Tang, Christopher Morris, Mirco Ravanelli, Guy Wolf, Prudencio Tossou, Hadrien Mary, Therence Bois, Andrew William Fitzgibbon, Blazej Banaszewski, Chad Martin, and Dominic Masters. Towards foundational models for molecular learning on large-scale multi-task datasets. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=Zc2aIcucwc.

Maciej Sypetkowski, Frederik Wenkel, Farimah Poursafaei, Nia Dickson, Karush Suri, Philip Fradkin, and Dominique Beaini. ON THE SCALABILITY OF GNNS FOR MOLECULAR GRAPHS. In *ICLR 2024 Workshop on Navigating and Addressing Data Problems for Foundation Models*, 2024. URL https://openreview.net/forum?id=WSqiNe1o0w.

Martin Grohe and Eran Rosenbluth. Are targeted messages more effective?, 2024.

Chen Cai, Truong Son Hy, Rose Yu, and Yusu Wang. On the connection between mpnn and graph transformer. *arXiv preprint arXiv:2301.11956*, 2023.

Krzysztof Marcin Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Quincy Davis, Afroz Mohiuddin, Lukasz Kaiser, David Benjamin Belanger, Lucy J Colwell, and Adrian Weller. Rethinking attention with performers. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=Ua6zuk0WRH.

Chulhee Yun, Srinadh Bhojanapalli, Ankit Singh Rawat, Sashank Reddi, and Sanjiv Kumar. Are transformers universal approximators of sequence-to-sequence functions? In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=ByxRM0Ntvr.

Pablo Barceló, Floris Geerts, Juan Reutter, and Maksimilian Ryschkov. Graph neural networks with local graph parameters. *Advances in Neural Information Processing Systems*, 34:25280–25293, 2021.

Zhengdao Chen, Soledad Villar, Lei Chen, and Joan Bruna. On the equivalence between graph isomorphism testing and function approximation with gnns. In *Neural Information Processing Systems*, 2019. URL https://api.semanticscholar.org/CorpusID:168169990.

Albert Gu, Karan Goel, and Christopher Re. Efficiently modeling long sequences with structured state spaces. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=uYLFoz1vlAC.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.

Maximilian Beck, Korbinian Pöppel, Markus Spanring, Andreas Auer, Oleksandra Prudnikova, Michael Kopp, Günter Klambauer, Johannes Brandstetter, and Sepp Hochreiter. xlstm: Extended long short-term memory. *arXiv preprint arXiv:2405.04517*, 2024.

Appendix Chapter 3

Table A.1: URLs and versions of used software.

Software	Version	URL
Picat	3.26	http://picat-lang.org/
ACE	2.0	https://github.com/xcsp3team/ace
CoSoCo	1.12	https://www.cril.univ-artois.fr/XCSP18/files/cosoco-competition.tgz
PyCSP3	2.0	https://github.com/xcsp3team/pycsp3
RUNCSP	1.0	https://github.com/toenshoff/RUNCSP-PyTorch
Greedy + DSATUR + HybridEA (COL)	2017	http://rhydlewis.eu/resources/gCol.zip
ECORD	commit:3463463	https://github.com/tomdbar/ecord
ECO-DQN	commit:134df73	https://github.com/tomdbar/eco-dqn/
Greedy + SDP (MaxCut)	0.1.2	https://github.com/hermish/cvx-graph-algorithms
RLSAT	commit:292e55c	https://github.com/emreyolcu/sat
PDP	commit:1fca34d	https://github.com/microsoft/PDP-Solver
WalkSAT	56	https://gitlab.com/HenryKautz/Walksat
ProbSAT	v2014	https://github.com/satcompetition/2018/blob/master/solvers/probSAT.zip
MaxWalkSAT	2013	https://github.com/stechu/Maxwalksat
SATLike	3.0	http://lcs.ios.ac.cn/~caisw/MaxSAT.html
CCLS	2015	http://lcs.ios.ac.cn/~caisw/MaxSAT.html

Algorithm 1 Training ANYCSP

Input: Initial parameters θ, training distribution Ω, train_steps $\in \mathbb{N}$, batch_size $\in \mathbb{N}$, $T_{\text{train}} \in \mathbb{N}$, lr > 0, $\gamma \in (0, 1]$

Output: Trained parameters θ

for $s \in \{1, \dots, \text{train_steps}\}$ **do**
  for $i \in \{1, \dots, \text{batch_size}\}$ **do**
    $\mathcal{I} \sim \Omega$ {Sample training instance.}
    $\boldsymbol{\varphi}_\theta, \boldsymbol{\alpha}, \mathbf{r} \leftarrow \pi_\theta(\mathcal{I}, T_{\text{train}})$ {Apply policy network.}
    for $t \in \{1, \dots, T_{\text{train}}\}$ **do**
      $G_t \leftarrow \sum_{k=t}^{T} \gamma^{k-t} r^{(k)}$
    end for
    $\nabla_\theta J_i \leftarrow \nabla_\theta \sum_{t=1}^{T} \left(G_t \sum_{X \in \mathcal{X}} \log\left(\varphi_\theta^{(t)}(\alpha^{(t)}(X)) + \epsilon\right)\right)$ {Policy gradient for i-th instance in batch.}
  end for
  $\theta \leftarrow \theta + \frac{\text{lr}}{\text{batch_size}} \sum_i \nabla_\theta J_i$ {Average gradients and ascent.}
end for

return θ

Algorithm 2 Forward Pass of π_θ. All inner for-loops are parallelized.

Input: CSP instance $\mathcal{I} = (\mathcal{X}, \mathcal{D}, \mathcal{C})$, Number of steps $T \in \mathbb{N}$.
Output: Soft Assignments $\boldsymbol{\varphi} = \varphi^{(1)}, \ldots, \varphi^{(T)}$, Assignments $\boldsymbol{\alpha} = \alpha^{(1)}, \ldots, \alpha^{(T)}$, Rewards $\boldsymbol{r} = r^{(1)}, \ldots, r^{(T)}$

for $X \in \mathcal{X}$ **do**
  $\alpha^{(0)}(X) \sim \mathcal{D}(X)$ {Sample initial assignment uniformly.}
end for
$L_V^{(0)}, L_E^{(0)} \leftarrow \text{LABEL}(\mathcal{I}, \alpha^{(0)})$ {Get vertex + edge labels.}
$q^{(1)} \leftarrow Q_{\mathcal{I}}(\alpha^{(0)})\}$ {Init. best prior quality.}
for $v \in \mathcal{V}$ **do**
  $h^{(0)}(v) \leftarrow \mathbf{h}$ {$\mathbf{h}$ is the learned initial state.}
end for
for $t \in \{1, \ldots, T\}$ **do**
  for $v \in \mathcal{V}$ **do**
    $x^{(t)}(v) \leftarrow \mathbf{N}\Big(\big[h^{(t-1)}(v), L_V^{(t-1)}(v)\big]\Big)$ {Values generate latent state.}
    $m^{(t)}(v,0), m^{(t)}(v,1) \leftarrow \mathbf{M}_{\mathcal{V}}\big(x^{(t)}(v)\big)$ {Values generate two messages.}
  end for
  for $C \in \mathcal{C}$ **do**
    $y^{(t)}(C) = \underset{v \in \mathcal{N}(C)}{\text{AGG}}\, m^{(t)}\big(v, L_E(C,v)\big)$ {Constraints receive messages.}
    $m^{(t)}(C,0), m^{(t)}(C,1) = \mathbf{M}_{\mathcal{C}}\big(y^{(t)}(C)\big)$ {Constraints generate messages.}
  end for
  for $v \in \mathcal{V}$ **do**
    $y^{(t)}(v) = \underset{C \in \mathcal{N}(v) \cap \mathcal{C}}{\text{AGG}}\, m^{(t)}(C, L_E(C,v))$ {Values receive messages from constraints.}
    $z^{(t)}(v) = \mathbf{U}_{\mathcal{V}}\big(x^{(t)}(v) + y^{(t)}(v)\big) + x^{(t)}(v)$ {Values combine messages with their latent state.}
  end for
  for $X \in \mathcal{X}$ **do**
    $z^{(t)}(X) = \mathbf{U}_{\mathcal{X}}\Big(\underset{v \in D_X}{\text{AGG}}\, z^{(t)}(v)\Big)$ {Variables receive states from values.}
  end for
  for $v \in \mathcal{V}$ **do**
    $h^{(t)}(v) \leftarrow \mathbf{G}\Big(h^{(t-1)}(v), z^{(t)}(v) + z^{(t)}(X)\Big)$ {Update recurrent states.}
    $o^{(t)}(v) \leftarrow \mathbf{O}(h^{(t)}(v))$ {Values predict scores.}
  end for
  for $v \in \mathcal{V}$ **do**
    $\varphi^{(t)}(v) \leftarrow \frac{\exp(o^{(t)}(v))}{\sum_{v' \in \mathcal{V}(X_v)} \exp(o^{(t)}(v'))}$ {Apply softmax within each domain}
  end for
  $\alpha^{(t)} \sim \varphi^{(t)}$ {Sample next assignment.}
  $L_V^{(t)}, L_E^{(t)} \leftarrow \text{LABEL}(\mathcal{I}, \alpha^{(t)})$ {Relabel graph.}
  $r^{(t)} \leftarrow \max\{Q_{\mathcal{I}}(\alpha^{(t)}) - q^{(t)}, 0\}$ {Get Reward.}
  $q^{(t+1)} \leftarrow \max\{q^{(t)}, Q_{\mathcal{I}}(\alpha^{(t)})\}$ {Update best prior quality.}
end for

return $\boldsymbol{\theta}$, $\boldsymbol{\alpha}$, $\boldsymbol{r}$

Table A.2: Extended MaxCut results on Gset graphs. We provide the deviation from the best known cut size.

Graph	$\|V\|$	$\|E\|$	Greedy	SDP	RUN-CSP	ECO-DQN	ECORD	ANYCSP
G1	800	19176	675	346	242	90	**0**	**0**
G2	800	19176	570	343	226	103	**0**	1
G3	800	19176	607	326	207	85	10	**1**
G4	800	19176	587	330	222	55	**0**	**0**
G5	800	19176	593	344	225	67	**0**	1
G14	800	4694	164	145	132	46	15	**3**
G15	800	4661	185	126	146	46	16	**0**
G16	800	4672	158	128	141	45	20	**3**
G17	800	4667	164	121	132	49	17	**0**
G22	2000	19990	1019	-	365	157	33	**6**
G23	2000	19990	972	-	349	181	21	**6**
G24	2000	19990	965	-	350	177	39	**9**
G25	2000	19990	946	-	321	164	32	**4**
G26	2000	19990	1001	-	337	194	23	**8**
G35	2000	11778	442	-	371	134	46	**19**
G36	2000	11766	438	-	358	141	53	**22**
G37	2000	11785	415	-	379	146	54	**22**
G38	2000	11779	435	-	386	119	52	**22**
G43	1000	9990	473	213	156	66	**0**	**0**
G44	1000	9990	532	294	143	57	**0**	**0**
G45	1000	9990	463	329	135	52	**0**	1
G46	1000	9990	461	264	159	69	**1**	**1**
G47	1000	9990	455	289	158	33	6	**1**
G48	3000	6000	**0**	-	**0**	**0**	**0**	**0**
G49	3000	6000	**0**	-	**0**	72	**0**	**0**
G50	3000	6000	**0**	-	2	32	12	**0**
G51	1000	5909	196	152	175	49	6	**4**
G52	1000	5916	214	163	170	53	21	**5**
G53	1000	5914	208	167	191	50	20	**5**
G54	1000	5916	230	192	182	63	25	**5**
G55	5000	12498	1102	-	218	405	211	**69**
G58	5000	29570	1058	-	1014	560	223	**56**
G60	7000	17148	1562	-	279	836	367	**71**
G63	7000	41459	1384	-	1410	763	399	**74**
G70	10 000	9999	1088	-	285	758	290	**143**

Table A.3: Extended results on structured Vertex Coloring instances with chromatic number less than 10.

Graph	$\|V\|$	$\|E\|$	$X(G)$	ANYCSP		RUNCSP		PICAT		CoSoCo		HybridEA			Dsatur		Greedy	
				Solved	Time	Solved	Time	Solved	Time	Solved	Time	Solved	Cols	Time	Solved	Cols	Solved	Cols
1-FullIns_3.col	30	100	4	True	0.52	True	0.13	True	0.09	True	0.07	True	4	0	True	4	False	5
1-FullIns_4.col	93	593	5	True	0.52	False	TO	True	0.1	True	0.07	True	5	0	True	5	False	6
1-FullIns_5.col	282	3247	6	True	0.53	False	TO	True	0.19	True	0.11	True	6	0	True	6	False	9
1-Insertions_4.col	67	232	5	True	0.51	True	0.13	True	0.08	True	0.01	True	5	0	True	5	True	5
1-Insertions_5.col	202	1227	6	True	0.52	True	0.21	True	0.11	True	0.12	True	6	0	True	6	True	6
1-Insertions_6.col	607	6337	7	True	0.52	True	3.42	True	0.35	True	0.09	True	7	0	True	7	False	8
2-FullIns_3.col	52	201	5	True	0.51	True	0.13	True	0.08	True	0.01	True	5	0	True	5	True	5
2-FullIns_4.col	212	1621	6	True	0.52	False	TO	True	0.16	True	0.03	True	6	0	True	6	False	8
2-FullIns_5.col	852	12201	7	True	0.53	False	TO	True	0.73	True	0.19	True	7	10	True	7	False	10
2-Insertions_3.col	37	72	4	True	0.51	True	0.14	True	0.07	True	0.04	True	4	0	True	4	True	4
2-Insertions_4.col	149	541	5	True	0.52	True	0.15	True	0.08	True	0.02	True	5	0	True	5	False	6
2-Insertions_5.col	597	3936	6	True	0.52	True	0.84	True	0.17	True	0.06	True	6	0	True	6	False	8
3-FullIns_3.col	80	346	6	True	0.51	True	0.15	True	0.08	True	0.01	True	6	0	True	6	False	7
3-FullIns_4.col	405	3524	7	True	0.52	False	TO	True	0.17	True	0.05	True	7	0	True	7	False	8
3-Insertions_3.col	56	110	4	True	0.51	True	0.14	True	0.07	True	0.01	True	4	0	True	4	True	4
3-Insertions_4.col	281	1046	5	True	0.52	True	0.17	True	0.1	True	0.02	True	5	0	True	5	True	5
3-Insertions_5.col	1406	9695	6	True	0.52	True	7.52	True	0.52	True	0.25	True	6	20	True	6	False	8
4-FullIns_3.col	114	541	7	True	0.52	True	0.26	True	0.12	True	0.01	True	7	0	True	7	True	7
4-FullIns_4.col	690	6650	8	True	0.52	False	TO	True	0.31	True	0.09	True	8	0	True	8	False	11
4-Insertions_3.col	79	156	4	True	0.51	True	0.14	True	0.06	True	0.01	True	4	0	True	4	False	5
4-Insertions_4.col	475	1795	5	True	0.52	True	0.21	True	0.09	True	0.03	True	5	0	True	5	False	6
5-FullIns_3.col	154	792	8	True	0.52	False	TO	True	0.08	True	0.02	True	8	0	True	8	True	8
5-FullIns_4.col	1085	11395	9	True	0.51	False	TO	True	0.75	True	0.19	True	9	20	True	9	False	12
DSJC125.1.col	125	736	5	True	0.95	False	TO	True	0.13	True	0.05	True	5	0	False	6	False	9
DSJC250.1.col	250	3218	8	True	8.20	False	TO	False	TO	False	TO	True	8	30	False	11	False	13
ash331GPIA.col	662	4181	4	True	0.58	True	1.07	True	0.13	True	0.07	True	4	10	False	5	False	6
ash608GPIA.col	1216	7844	4	True	0.57	True	1.66	True	0.25	True	0.19	True	4	30	False	5	False	6
ash958GPIA.col	1916	12506	4	True	0.62	False	TO	True	0.45	True	0.24	True	4	110	False	6	False	6
games120.col	120	638	9	True	0.52	True	0.23	True	0.09	True	0.01	True	9	0	True	9	True	9
le450_5a.col	450	5714	5	True	32.04	False	TO	True	0.31	True	0.96	True	5	80	False	10	False	13
le450_5b.col	450	5734	5	True	6.64	False	TO	True	0.22	True	1.47	True	5	110	False	10	False	14
le450_5c.col	450	9803	5	True	4.02	True	1	True	0.32	True	0.68	True	5	40	False	11	False	17
le450_5d.col	450	9757	5	True	0.74	True	1.1	True	0.33	True	0.15	True	5	30	False	12	False	16
miles250.col	128	387	8	True	0.52	True	0.18	True	0.08	True	0.36	True	8	0	True	8	False	10
mug100_1.col	100	166	4	True	0.52	True	0.15	True	1.9	True	0.01	True	4	0	True	4	True	4
mug100_25.col	100	166	4	True	0.52	True	0.15	True	1.85	True	0.55	True	4	0	True	4	True	4
mug88_1.col	88	146	4	True	0.51	True	0.15	True	0.08	True	0.24	True	4	0	True	4	True	4
mug88_25.col	88	146	4	True	0.51	True	0.15	True	0.07	True	0.46	True	4	0	True	4	True	4
myciel3.col	11	20	4	True	0.50	True	0.14	True	0.08	True	0.79	True	4	0	True	4	True	4
myciel4.col	23	71	5	True	0.51	True	0.14	True	0.07	True	0.68	True	5	0	True	5	True	5
myciel5.col	47	236	6	True	0.51	True	0.14	True	0.07	True	0.23	True	6	0	True	6	True	6
myciel6.col	95	755	7	True	0.51	True	0.21	True	0.19	True	0.35	True	7	0	True	7	False	8
myciel7.col	191	2360	8	True	0.51	True	3.42	True	0.19	True	0.04	True	8	0	True	8	False	9
queen5_5.col	25	160	5	True	0.57	True	0.14	True	0.07	True	0.3	True	5	0	True	5	False	7
queen6_6.col	36	290	7	True	0.57	False	TO	True	0.17	True	0.16	True	7	0	False	9	False	10
queen7_7.col	49	476	7	True	3.16	True	0.2	True	0.08	True	0.23	True	7	0	False	10	False	12
queen8_8.col	64	728	9	True	6.35	False	TO	True	0.12	True	1.18	True	9	0	False	11	False	13
r125.1.col	125	209	5	True	0.52	True	0.14	True	0.08	True	0.07	True	5	0	True	5	False	6
r250.1.col	250	867	8	True	0.53	False	TO	True	0.1	True	0.08	True	8	0	True	8	False	9
will199GPIA.col	701	6772	7	True	0.57	False	TO	True	0.26	True	0.1	True	7	0	True	7	False	12

Table A.4: Extended results on structured Vertex Coloring instances with chromatic number at least 10.

Graph	$\|V\|$	$\|E\|$	$X(G)$	ANYCSP		PICAT		CoSoCo		HybridEA			Dsatur		Greedy	
				Solved	Time	Solved	Time	Solved	Time	Solved	Cols	Time	Solved	Cols	Solved	Cols
DSJC125.5.col	125	3891	17	True	186.92	False	TO	False	TO	True	17	400	False	22	False	25
DSJC125.9.col	125	6961	44	True	23.4	True	52.65	False	TO	True	44	40	False	51	False	56
DSJR500.1.col	500	3555	12	True	0.56	True	0.37	True	0.05	True	12	0	True	12	False	14
anna.col	138	493	11	True	0.52	True	0.11	True	0.02	True	11	0	True	11	False	13
david.col	87	406	11	True	0.51	True	0.1	True	0.02	True	11	0	True	11	False	12
flat300_28_0.col	300	21695	28	False	TO	False	TO	False	TO	False	31	TO	False	42	False	46
fpsol2.i.1.col	496	11654	65	True	1.14	True	37.88	True	0.35	True	65	0	True	65	True	65
fpsol2.i.2.col	451	8691	30	True	0.67	True	5.67	True	0.14	True	30	0	True	30	True	30
fpsol2.i.3.col	425	8688	30	True	0.67	True	5.71	True	0.14	True	30	0	True	30	True	30
homer.col	561	1628	13	True	0.51	True	0.17	False	TO	True	13	0	True	13	False	14
huck.col	74	301	11	True	0.51	True	0.09	True	0.01	True	11	0	True	11	True	11
inithx.i.1.col	864	18707	54	True	1.07	True	52.22	True	0.41	True	54	0	True	54	True	54
inithx.i.2.col	645	13979	31	True	0.62	True	12.77	True	0.22	True	31	0	True	31	True	31
inithx.i.3.col	621	13969	31	True	0.66	True	12.51	True	0.23	True	31	0	True	31	True	31
jean.col	80	254	10	True	0.52	True	0.09	True	0.01	True	10	0	True	10	True	10
le450_15a.col	450	8168	15	True	4.21	True	1.39	True	1078.32	True	15	2720	False	16	False	21
le450_15b.col	450	8169	15	True	1.97	True	1.29	True	0.13	True	15	100	False	16	False	22
le450_15c.col	450	16680	15	True	44.86	False	TO	False	TO	False	16	TO	False	24	False	29
le450_15d.col	450	16750	15	False	TO	False	TO	False	TO	True	15	3190	False	24	False	33
le450_25a.col	450	8260	25	True	0.55	True	3.21	True	0.23	True	25	0	True	25	False	28
le450_25b.col	450	8263	25	True	0.55	True	3.25	True	0.2	True	25	0	True	25	False	27
le450_25c.col	450	17343	25	False	TO	False	TO	False	TO	False	26	TO	False	29	False	34
le450_25d.col	450	17425	25	False	TO	False	TO	False	TO	False	26	TO	False	28	False	38
miles1000.col	128	3216	42	True	0.69	True	1.11	True	0.12	True	42	0	True	42	False	46
miles1500.col	128	5198	73	True	0.72	True	10.02	True	0.27	True	73	0	True	73	False	75
miles500.col	128	1170	20	True	0.52	True	0.19	True	0.02	True	20	0	True	20	False	22
miles750.col	128	2113	31	True	0.71	True	0.39	True	0.08	True	31	0	True	31	False	33
mulsol.i.1.col	197	3925	49	True	0.61	True	2.07	True	0.1	True	49	0	True	49	True	49
mulsol.i.2.col	188	3885	31	True	0.58	True	0.9	True	0.17	True	31	0	True	31	True	31
mulsol.i.3.col	184	3916	31	True	0.55	True	0.93	True	0.13	True	31	0	True	31	True	31
mulsol.i.4.col	185	3946	31	True	0.57	True	0.92	True	0.07	True	31	0	True	31	True	31
mulsol.i.5.col	186	3973	31	True	0.55	True	0.95	True	0.06	True	31	0	True	31	True	31
queen10_10.col	100	2940	11	True	16.95	True	30.73	True	351.24	True	11	0	False	14	False	15
queen11_11.col	121	3960	11	False	TO	False	TO	False	TO	False	12	TO	False	15	False	16
queen12_12.col	144	5192	12	False	TO	False	TO	False	TO	False	13	TO	False	17	False	18
queen13_13.col	169	6656	13	False	TO	False	TO	False	TO	False	14	TO	False	18	False	20
queen14_14.col	196	4186	14	False	TO	False	TO	False	TO	False	15	TO	False	19	False	20
queen15_15.col	225	5180	15	False	TO	False	TO	False	TO	False	16	TO	False	22	False	22
queen16_16.col	256	12640	16	False	TO	False	TO	False	TO	False	17	TO	False	22	False	23
queen8_12.col	96	1368	12	True	0.6	True	0.14	True	0.93	True	12	0	False	13	False	16
queen9_9.col	81	1056	10	True	4.97	True	0.21	True	47.81	True	10	0	False	13	False	14
r1000.1.col	1000	14378	20	True	0.73	True	11.11	True	0.28	True	20	10	True	20	False	26
r125.1c.col	125	7501	46	True	0.73	True	14.05	True	1.14	True	46	0	True	46	False	51
r125.5.col	125	3838	36	True	0.78	True	1.45	True	21.86	True	36	30	False	38	False	43
r250.5.col	250	14849	65	True	36.41	True	56.35	False	TO	True	65	442170	False	68	False	78
school1.col	385	19095	14	True	0.79	True	6.42	False	TO	True	14	0	False	17	False	44
school1_nsh.col	352	14612	14	True	0.8	True	2.9	False	TO	True	14	20	False	27	False	38
zeroin.i.1.col	211	4100	49	True	0.66	True	2.41	True	0.08	True	49	0	True	49	True	49
zeroin.i.2.col	211	3541	30	True	0.56	True	0.8	True	0.06	True	30	0	True	30	True	30
zeroin.i.3.col	206	3540	3	True	0.54	True	0.78	True	0.05	True	30	0	True	30	False	31

Appendix Chapter 4

Table B.1: Hyperparameters used in our main experiments.

	ZINC	CIFAR10	MNIST	CSL	MOLPCBA	PASCAL VOC-SP	PEPTIDES -FUNC	PEPTIDES -STRUCT
L	3	3	3	2	5	6	6	6
d	147	75	75	90	400	100	100	100
s	8	8	8	8	8	16	16	16
pool	sum	mean	mean	mean	mean	sum	sum	sum
out	mlp	mlp	mlp	mlp	linear	mlp	mlp	mlp
ℓ_{train}	50	50	50	50	50	50	50	50
ℓ_{eval}	150	150	150	150	100	150	150	150
$p*$	1	1	1	1	0.2	0.2	0.2	0.2
walk strat.	nb	nb	nb	nb	nb	nb	nb	nb
r_{val}	2	2	2	5	2	2	2	2
r_{test}	10	10	10	10	10	5	5	5
dropout	0.0	0.0	0.0	0.0	0.25	0.1	0.1	0.2
learning rate	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001
patience	10	10	10	20	-	-	-	-
epochs	-	-	-	-	60	300	500	500
batch size	50	50	50	50	100	50	50	50
virtual node	Yes	No	No	No	Yes	Yes	Yes	Yes
#Params	497,743	109,660	109,360	104,140	6,115,728	495,021	510,910	511,011

Table B.2: Statistics for our main datasets.

	#Graphs	Avg. $\|V\|$	Avg. $\|E\|$	#Classes/#Tasks
ZINC	10000 / 1000 / 1000	23.1	49.8	1
CIFAR10	45000 / 5000 / 10000	117.6	1129.8	10
MNIST	55000 / 5000 / 10000	80.98	785.39	10
CSL	150	41	82	10
MOLPCBA	350343 / 43793 / 43793	25.6	55.4	128
PASCALVOC-SP	8498/1428/1429	479.40	2710.48	21
PEPTIDES-FUNC	10873/2331/2331	150.94	307.30	10
PEPTIDES-STRUCT	10873/2331/2331	150.94	307.30	11

Appendix Chapter 5

Tables C.1 to C.5 provide the tuned hyperparameters used in Section 5.1 and Table C.6 provides the complete results on the LRGB test splits. Tables C.7 and C.8 further provide the hyperparameters used to obtain the results in Section 5.2.4.2.

Table C.1: Hyperparameters on Peptides-Func

	GCN	GINE	GatedGCN	GPS
lr	0.001	0.001	0.001	0.001
dropout	0.1	0.1	0.1	0.1
#layers	6	8	10	6
hidden dim.	235	160	95	76
head depth	3	3	3	2
PE/SE	RWSE	RWSE	RWSE	LapPE
batch size	200	200	200	200
#epochs	250	250	250	250
norm	-	-	-	BatchNorm
MPNN	-	-	-	GatedGCN
#Param.	486k	491k	493k	479k

Table C.2: Hyperparameters on Peptides-Struct

	GCN	GINE	GatedGCN	GPS
lr	0.001	0.001	0.001	0.001
dropout	0.1	0.1	0.1	0.1
#layers	6	10	8	8
hidden dim.	235	145	100	64
head depth	3	3	3	2
PE/SE	LapPE	LapPE	LapPE	LapPE
batch size	200	200	200	200
#epochs	250	250	250	250
norm	-	-	-	BatchNorm
MPNN	-	-	-	GatedGCN
#Param.	488k	492k	445k	452k

Table C.3: Hyperparameters on PascalVOC-SP

	GCN	GINE	GatedGCN	GPS
lr	0.001	0.001	0.001	0.001
dropout	0.0	0.2	0.2	0.1
#layers	10	10	10	8
hidden dim.	200	145	95	68
head depth	3	2	2	2
PE/SE	RWSE	none	none	LapPE
batch size	50	50	50	50
#epochs	200	200	200	200
norm	-	-	-	BatchNorm
MPNN	-	-	-	GatedGCN
#Param.	490k	450k	473k	501k

Table C.4: Hyperparameters on COCO-SP

	GCN	GINE	GatedGCN	GPS
lr	0.001	0.001	0.001	0.001
dropout	0.1	0.1	0.1	0.1
#layers	6	6	8	8
hidden dim.	280	195	105	68
head depth	1	1	1	1
PE/SE	none	none	none	none
batch size	50	50	50	50
#epochs	200	200	200	200
norm	-	-	-	LayerNorm
MPNN	-	-	-	GatedGCN
#Param.	500k	478k	459k	500k

Table C.5: Hyperparameters on PCQM-Contact

	GCN	GINE	GatedGCN	GPS
lr	0.001	0.001	0.001	0.001
dropout	0.1	0.1	0.1	0.0
#layers	8	8	8	6
hidden dim.	215	160	105	76
head depth	1	1	1	1
PE/SE	LapPE	LapPE	LapPE	LapPE
batch size	500	500	500	500
#epochs	150	150	150	150
norm	-	-	-	LayerNorm
MPNN	-	-	-	GatedGCN
#Param.	456k	466k	477k	478k

Table C.6: Performance of our models on the Long-Range Graph Benchmark.

Method	PASCALVOC-SP Test F1 ↑	COCO-SP Test F1 ↑	PEPTIDES-FUNC Test AP ↑	PEPTIDES-STRUCT Test MAE ↓	PCQM-CONTACT Test MRR ↑		
					raw	filter	ext. filter
GCN	0.2078 ± 0.0031	0.1338 ± 0.0007	0.6860 ± 0.0050	0.2460 ± 0.0007	0.3424 ± 0.0007	0.3631 ± 0.0006	0.4526 ± 0.0006
GINE	0.2718 ± 0.0054	0.2125 ± 0.0009	0.6621 ± 0.0067	0.2473 ± 0.0017	0.3509 ± 0.0006	0.3725 ± 0.0006	0.4617 ± 0.0005
GatedGCN	0.3880 ± 0.0040	0.2922 ± 0.0018	0.6765 ± 0.0047	0.2477 ± 0.0009	0.3495 ± 0.0010	0.3714 ± 0.0010	0.4670 ± 0.0004
GPS	0.4440 ± 0.0065	0.3884 ± 0.0055	0.6534 ± 0.0091	0.2509 ± 0.0014	0.3498 ± 0.0005	0.3722 ± 0.0005	0.4703 ± 0.0014

Table C.7: Hyperparameters for GatedGCN+VN.

	PascalVOC-SP	COCO-SP	Peptides-Func	Peptides-Struct	ogbg-code2	ogbg-molpcba
lr	0.001	0.001	0.001	0.001	0.001	0.001
dropout	0.1	0.1	0.1	0.1	0.3	0.4
#layers	10	6	8	10	5	6
hidden dim.	70	90	80	70	256	1024
head depth	2	1	2	2	1	1
PE/SE	none	none	RWSE	LapPE	none	RWSE
VN pooling	mean	mean	mean	mean	mean	sum
batch size	50	50	200	200	32	512
#epochs	200	200	250	250	50	60
#Param.	478K	453K	486K	466K	1.20M	57M
time/epoch	66s	206s	24s	28s	502s	487s

Table C.8: Hyperparameters for GCN+VN.

	PascalVOC-SP	COCO-SP	Peptides-Func	Peptides-Struct
lr	0.001	0.001	0.001	0.001
dropout	0.2	0.1	0.1	0.1
#layers	8	10	10	8
hidden dim.	105	95	95	105
head depth	2	1	3	2
PE/SE	none	none	RWSE	LapPE
VN pooling	mean	mean	mean	mean
batch size	50	50	200	200
#epochs	200	200	250	250
#Param.	460k	465k	487k	474k
time/epoch	35s	156s	18s	19s