SPECTRALMPNN: SPECTRAL GRAPH ARCHITECTURES FOR NEURAL ALGORITHMIC REASONING

Ronald Albert de Araújo

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia de Sistemas e Computação.

Orientadores: Gerson Zaverucha
Aline Marins Paes Carvalho

Rio de Janeiro
Maio de 2025

SPECTRALMPNN: SPECTRAL GRAPH ARCHITECTURES FOR NEURAL ALGORITHMIC REASONING

Ronald Albert de Araújo

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Orientadores: Gerson Zaverucha
Aline Marins Paes Carvalho

Aprovada por: Prof. Gerson Zaverucha
Profa. Aline Marins Paes Carvalho
Prof. Daniel Ratton Figueiredo
Prof. Diego Mesquita

RIO DE JANEIRO, RJ – BRASIL
MAIO DE 2025

de Araújo, Ronald Albert

SpectralMPNN: Spectral Graph Architectures for Neural Algorithmic Reasoning/Ronald Albert de Araújo. – Rio de Janeiro: UFRJ/COPPE, 2025.

XIV, 80 p.: il.; 29, 7cm.

Orientadores: Gerson Zaverucha

Aline Marins Paes Carvalho

Dissertação (mestrado) – UFRJ/COPPE/Programa de Engenharia de Sistemas e Computação, 2025.

Referências Bibliográficas: p. 54 – 67.

1. neural algorithmic reasoning. 2. spectral graph neural networks. 3. message passing. 4. graph signal processing. I. Zaverucha, Gerson *et al.* II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

A todos aqueles que continuam
ao meu lado e continuarão anos
por vir.

Agradecimentos

Este trabalho é indiretamente coescrito por todos os mencionados nesta seção de agradecimentos. Gostaria de iniciar expressando minha profunda gratidão a todos que contribuíram, direta ou indiretamente, para o desenvolvimento desta dissertação.

Em primeiro lugar, gostaria de agradecer aos dois coautores formais, Prof. Gerson Zaverucha e Profa. Aline Paes, que foram fundamentais para a realização desta dissertação. Sua atenção e colaboração durante a orientação foram uma grande fonte de inspiração pessoal, e expresso aqui minha profunda admiração por ambos, como pesquisadores e professores. É uma honra enorme poder dividir esse trabalho com dois grandes pesquisadores na área de Inteligência Artificial e sou extremamente grato por poderem coassinar este trabalho comigo.

Aos meus pais, Ronald Albert e Carla Bitencourt, que sempre estiveram presentes e incentivaram minha educação. Meus alicerces, que me dão tranquilidade em relação ao futuro e que mais me auxiliam, e sempre me auxiliaram, a superar minhas barreiras. Muito obrigado, por terem me fornecido segurança até aqui e espero que ainda possam fornecer muito mais.

À minha irmã, Rebecca Bitencourt, que conheço desde o nascimento e com quem compartilho uma relação muito singular. É ela quem melhor sabe me avaliar de uma perspectiva única e apontar caminhos ainda não explorados. Obrigado por sempre abrir meus olhos para novas possibilidades e por me conhecer tão bem. Acredito que ninguém jamais conseguirá me acessar tão profundamente quanto você.

Ao meu avô, Francisco de Assis Bitencourt, que sempre demonstrou interesse pelo que estou fazendo e nutre grande admiração por mim. Nunca teria conquistado nada se não fosse por seus incentivos e pela crença no meu potencial.

Aos meus amigos mais próximos, João Vitor Esteves e David Medeiros, que, além de compartilharem comigo boa parte da experiência da pós-graduação — ainda que em áreas diferentes —, também me acompanham nos bons momentos da vida. Muito obrigado pelo esforço de me compreenderem e por continuarem ao meu lado.

Aos meus colegas de mestrado e grandes amigos, Anna Barbara Coimbra e Rodrigo Tanajura, que tornaram a experiência da pós-graduação na COPPE mais leve e significativa, compartilhando comigo os desafios e conquistas dessa jornada. Por

mais impressionante que pareça, nossos almoços juntos tiveram um papel fundamental na construção desta dissertação.

Aos meus grandes amigos, Tobias, Carolina, Myllene, Lucas, Hernan, Silvana, Luiza, Julio, Mariana, Guilherme, Gabriel, Juliana, Cecilia, Luiza, Lara, Daniel Marcelo, Geovanna e Marcelle, pela certeza de sempre encontrar a felicidade ao lado de vocês.

À toda minha família, Flavio, Bernardo, Maria Antonia, Marcelo, Eduardo e Paula, pelo apoio desde o meu nascimento até hoje.

A todo o corpo discente do grupo de pesquisa MeLLL-UFF, por valiosas discussões e pelo trabalho excepcional no campo da Inteligência Artificial. Em especial, ao aluno Gabriel Assis, cujo auxílio na operação das máquinas foi fundamental para a realização dos experimentos.

Sou grato à Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) pelo apoio financeiro concedido para a realização deste trabalho.

Gostaria de agradecer também aos professores Daniel Ratton e Diego Mesquita pela presença na banca e pela disponibilidade em avaliar esta dissertação.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

SPECTRALMPNN: ARQUITETURAS ESPECTRAIS EM GRAFOS PARA EXECUÇÃO ALGORÍTMICA

Ronald Albert de Araújo

Maio/2025

Orientadores: Gerson Zaverucha
Aline Marins Paes Carvalho

Programa: Engenharia de Sistemas e Computação

Neural algorithmic reasoning é um subcampo emergente do aprendizado de representações que se concentra no treinamento de modelos neurais para imitar de forma eficaz as execuções de algoritmos clássicos da ciência da computação. Essa área tem demonstrado grande valor tanto em aplicações práticas—onde modelos pré-treinados em dados algorítmicos apresentam desempenho aprimorado—quanto em pesquisas teóricas, ao revelar aspectos arquiteturais das redes neurais que favorecem o computação discreta.A maioria das abordagens existentes restringem as arquiteturas aplicadas ao tipo de Redes Neurais *Message-Passing* em grafos. Essa escolha decorre principalmente da versatilidade das estruturas de grafos, capazes de aproximar uma ampla variedade de estruturas de dados, e de descobertas teóricas recentes que destacam a relação entre a passagem de mensagens e a programação dinâmica. No entanto, existe uma limitação fundamental associada a este paradigma: seu viés inerente para representações suaves no grafo. Esse viés está diretamente relacionado ao fenômeno de over-smoothing, no qual múltiplas rodadas de *message-passing* fazem com que todas as representações de nós convirjam para um mesmo valor. Essa limitação dificulta a aplicação das *Message-Passing GNNs* a grafos heterofílicos, onde nós conectados frequentemente apresentam características distintas, que é o cenário presente em grande parte das tarefas algorítmicas. Por outro lado, Spectral GNNs representam uma classe de redes neurais em grafos que exploram a estrutura do grafo aprendendo filtros diretamente no domínio da Transformada de Fourier no grafo. Isso as permite atuar como filtros adaptativos, possibilitando a propagação de sinais de forma flexível com base na frequência. Nesta dissertação, propomos o

uso de Spectral GNNs para o Neural Algorithmic Reasoning e introduzimos a SpectralMPNN—uma Spectral GNN que combina filtragem adaptativa com uma camada de *message-passing*. A SpectralMPNN foi projetada para manter os vieses indutivos benéficos de *message-passing*, ao mesmo tempo em que incorpora filtragem adaptativa de frequências. Nossos experimentos mostram que essa arquitetura supera os modelos existentes em diversos algoritmos do benchmark CLRS-30.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

SPECTRALMPNN: SPECTRAL GRAPH ARCHITECTURES FOR NEURAL ALGORITHMIC REASONING

Ronald Albert de Araújo

May/2025

Advisors: Gerson Zaverucha
Aline Marins Paes Carvalho

Department: Systems Engineering and Computer Science

Neural algorithmic reasoning is an emerging subfield of representation learning that focuses on training neural models to effectively mimic the execution traces of classical computer science algorithms. This field has shown significant value in practical applications—where models pre-trained on algorithmic data exhibit enhanced performance—and in theoretical research by shedding light on the architectural aspects of neural networks that facilitate discrete reasoning. Most existing approaches to this problem restrict the design space to Message-Passing GNNs. This choice is primarily driven by the versatility of graph structures, which can approximate a wide range of data structures, and by recent theoretical findings highlighting the alignment between message passing and dynamic programming. However, prior studies have underscored a fundamental limitation of message passing: its inherent bias toward smooth representations across the graph. This characteristic is closely linked to the over-smoothing phenomenon, where multiple rounds of message passing cause all node representations to converge to the same value. This limitation poses challenges for applying Message-Passing GNNs to heterophilic graphs, where connected nodes often have distinct features, which is commonly the case for algorithmic tasks. Conversely, Spectral GNNs represent a class of graph neural networks that leverage the graph structure by learning effective filters directly in the Graph Fourier domain. This enables them to function as adaptive filters, allowing flexible frequency-based signal propagation. In this dissertation, we propose using Spectral GNNs for neural algorithmic reasoning and introduce SpectralMPNN—a Spectral GNN that integrates filtering with a Message-Passing layer. SpectralMPNN is designed to retain

the beneficial inductive biases of message passing while incorporating adaptive frequency filtering. Our experiments show that this architecture outperforms existing models across multiple algorithms in the CLRS-30 benchmark.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Over the past decade, Deep Learning has achieved remarkable success across a wide range of domains, starting a real revolution in the field of Artificial Intelligence. In computer vision, image classification, object detection, and segmentation, neural network models can be trained to recognize thousands of different categories and understand images at a very deep level [6], at times even surpassing human-level performance [7]. Similarly, in audio processing, once dominated by traditional signal processing techniques, Deep Learning has led to near-human or even superhuman performance, with models achieving exceptionally high accuracy across a range of tasks [8]. Arguably, the most prominent recent successes have been in the field of Natural Language Processing (NLP), with landmark breakthroughs in machine translation [9] and sentiment analysis [10]. More recently, large-scale textual foundation models, such as the GPT and Gemini series [11, 12], have demonstrated remarkable performance across a wide range of downstream tasks. Pre-trained on massive corpora of unlabeled text, such models can often achieve impressive results even without additional fine-tuning [13]. These models, commonly referred to as Large Language Models (LLMs), have enabled the emergence of a new learning paradigm known as in-context learning [14]. This form of learning occurs at inference time, where the model, leveraging its extensive self-supervised pre-training, is capable of producing high-quality results from only a few, or even zero, example demonstrations provided alongside the input.

Despite the numerous successes of Deep Learning across a wide array of tasks, several avenues for future improvement remain. In particular, recent research has increasingly focused on evaluating and enhancing the reasoning capabilities of neural models [1, 15]. Various definitions have been proposed for both formal and informal reasoning, and several works aim to connect these two forms [16]. Reasoning is commonly understood as the ability to combine previously acquired information in a structured and logically valid manner, progressing from premises to a conclusion through formal rules of inference [17]. Mathematical reasoning [18], coding

capabilities [19] and task adaptation [20] are common benchmarks for reasoning.

Equipping current Deep Learning models with robust reasoning capabilities could help address some of the core limitations of modern LLMs, such as the generation of false information and hallucinations [21]. By framing reasoning as an additional constraint on the space of possible conclusions, limiting outputs to those that are logical consequences and derivable from the given premises, it becomes evident how such mechanisms could mitigate issues like hallucination.

1.1 System 1 and System 2

A widely used framework for analyzing the cognitive abilities of current neural models is the Dual-System Theory, introduced in Daniel Kahneman's *Thinking, Fast and Slow* [22]. System 1, associated with perception tasks, represents fast, intuitive thinking that operates automatically and with minimal cognitive effort. This mode of thinking aligns well with the strengths of Deep Learning models, which typically excel in tasks such as image recognition, language completion, and pattern detection. In contrast, System 2 is characterized by slow, deliberate, and rational thinking, often required for tasks involving reasoning, abstraction, and planning. It is within this domain that current neural models tend to struggle, frequently failing to demonstrate consistent logical reasoning or to generalize contexts not observed during training [15, 23].

Diving deeper into the dichotomy between System 1 and System 2, we encounter the concept of two poles of abstraction [5, 24]. This idea suggests that intelligence arises from the accumulation of small, meaningful units, often referred to as atoms of meaning, acquired through experience. These atoms can be reused and recombined to navigate novel situations. In this view, intelligence is defined as the capacity to recognize similarities, or *analogies*, between new problems and previously encountered patterns, enabling the transfer and application of learned knowledge. These transferable units of knowledge are known as *abstractions*.

The two poles of abstraction is the idea that the *analogies* between *abstractions* and novel problems, can be built from one out of two ways, closely aligned to the dual-system theory.

The first kind of analogy is value-centric analogy [5], where instances are related through measures of similarity. In this form of reasoning, we relate different examples based on shared features or functions, gradually forming abstract prototypes that capture common patterns. This underlies intuitive pattern recognition: rather than memorizing every example, we group similar instances and rely on the value they represent to make fast, effective judgments. When humans perform tasks effortlessly or without conscious deliberation (System 1 thinking), they are often engaging in

value-centric analogy, an ability closely tied to perception. Deep Learning models are particularly well-suited for this kind of analogy-making. By learning latent embedding spaces, they excel at comparing feature representations of known samples to those of novel inputs, enabling local generalization in perceptual tasks.

Cognition involves more than the fast, intuitive categorization enabled by value-centric analogy. A second, more deliberate form of abstraction is known as program-centric (or structure-centric) analogy [5]. This type of reasoning is slower, more precise, and centered on identifying exact structural correspondences.

A useful metaphor comes from software engineering: when developers recognize redundant patterns across different functions or classes, they often refactor by abstracting common logic into reusable components, such as a shared base class or a generalized function. This process does not rely on visual or perceptual similarity but on matching structural patterns, specifically, subgraph isomorphisms, where the structure of operations or relationships is preserved across different representations. Crucially, this kind of comparison cannot be achieved by projecting instances into an arbitrary latent feature space; rather, it requires symbolic, exact matching of structural elements, making it fundamentally different from the continuous similarity-based comparisons used in value-centric analogy.

Program-centric analogy underlies System 2 thinking: it is associated to logic, planning, mathematical reasoning, and formal abstraction. It comes into play whenever we analyze systems composed of discrete, relational structures—whether in symbolic logic, algebra, or causal reasoning. Unlike value-centric analogy, which operates over continuous similarity spaces, program-centric analogy leverages discrete, rule-based networks of relationships to support rigorous, generalizable understanding. As previously mentioned, this is commonly the kind of analogy neural networks fail to perform.

Table 1.1: The two kinds of analogy comparison [5]

Value-centric analogy	Program-centric analogy
Continuous instances	Discrete instances
Instances are associated by distance or similarity measures	Instances are associated by exact structural matching
Abstractions are obtained from averaging instances	Abstractions are obtained from common instances substructures
Perception & Intuition	Reasoning & Planning
Requires a lot of samples to generalize	Naturally generalizes

Certain tasks are considered to closely align with System 2 thinking [20, 23], prompting growing interest within the Deep Learning community in evaluating and enhancing neural models' capabilities on such tasks. One prominent example is algorithmic execution [1], which has led to the emergence of the field known as Neural Algorithmic Reasoning [4]. This area of research aims to connect neural models and

classical algorithms by exploring how neural networks can learn to perform discrete, structured computations in a generalizable and interpretable manner.

1.2 Neural Algorithmic Reasoning

The growing interest in neural models capable of performing discrete computations also raises important questions about which architectural features make these networks more prone to reasoning. This topic is explored in depth in [1], where the authors introduce the concept of Algorithmic Alignment (Definition 3.2), a framework for assessing how well a neural network's structure corresponds to the computational structure of a given algorithmic task. They argue that better algorithmic alignment leads to improved sample complexity (Appendix C), allowing models to learn more efficiently from fewer examples. Moreover, they show that some networks structures align better with algorithmic problems and should perform better in tasks regarding discrete computation.

Since then, Neural Algorithmic Reasoning [4] has presented itself as a promising field for the theoretical study of neural networks, not only due to its emphasis on discrete computation and alignment with reasoning, but also due to the generalization properties inherent to algorithmic processes. When an algorithm is accompanied by a proof of correctness, it offers a strong theoretical guarantee: its behavior will generalize reliably, regardless of the size or structure or size of the input, different from neural networks, which often struggle to generalize beyond the distributions encountered during training.

This property of naturally generalizing *out-of-distribution* [25] makes algorithmic reasoning a valuable lens for investigating which aspects of neural architectures contribute to robust generalization beyond the training distribution. Moreover, out-of-distribution generalization is frequently associated with System 2 cognitive abilities in neural models, those involving deliberate, structured, and reasoning-driven processes [24, 26].

Beyond its theoretical strengths, Neural Algorithmic Reasoning also opens pathways for adapting algorithmic execution to a broader range of real-world inputs. Traditional algorithms operate in an abstract space, requiring their inputs to be well-defined abstract structures. As a result, applying an algorithm to a real-world problem necessitates a preprocessing step that maps the raw, often noisy input data into a suitable abstract representation. Neural models offer the potential to learn this projection automatically, presenting a promising path towards applying algorithms in real-world tasks.

Neural Algorithmic Reasoning has demonstrated strong performance on real-world tasks that resemble algorithm execution [4]. Studies have shown that neural

networks pretrained on algorithmic execution exhibit enhanced performance on real-world problems that require formal reasoning procedures [27–30], highlighting the practical benefits of integrating algorithmic priors into learning models.

1.3 Deep Learning over graphs

Message-passing Graph Neural Networks (GNNs) have become the dominant architecture for Neural Algorithmic Reasoning [31], largely due to the flexibility of graph representations and their inherent inductive biases, which have been shown to facilitate algorithmic execution [1, 32]. However, a well-documented limitation of message-passing GNNs is the *oversmoothing* phenomenon [33], where node representations become indistinguishable as the network depth increases. This effect has been linked to the low-pass filtering nature of these models, which causes the loss of discriminative features when many layers are stacked [34, 35].

Oversmoothing not only limits the model's ability to capture complex and high-frequency patterns but also introduces a depth-vs-information trade-off, shallower networks mitigate *oversmoothing* but struggle to propagate information across distant nodes, harming their ability to encode global structure [36].

A promising alternative to mitigate this issue is Spectral GNNs [37], which operate in the frequency domain using the graph Laplacian's eigendecomposition. Unlike message-passing models that focus on local neighborhoods, Spectral GNNs inherently capture global structural information, an essential trait for algorithmic reasoning tasks that often rely on the overall structure of the input graph.

Because many algorithms rely on structural properties and global patterns to guide decision-making, Spectral GNNs offer a natural inductive bias for such tasks. Yet, to the best of the authors' knowledge, their application remains largely unexplored within the context of Neural Algorithmic Reasoning.

The contribution of this dissertation is twofold. Starting from the hypothesis that inherent locality bias of Message-Passing Graph Neural Networks (MPGNNs) limits their capacity for algorithmic reasoning, we aim to evaluate the extent to which this limitation affects performance and how the aforementioned problem could be solved. As possible pathway to address on the mentioned issue, we investigate the potential performance gains achievable from adopting architectures that more effectively capture the structural properties of input graphs.

While structural graph information is fundamental for algorithmic reasoning, previous works have outlined the resemblance between the message-passing scheme and algorithm execution strategies [1, 32]. In order to efficiently encode global information and retain the useful inductive bias of message-passing, we propose a novel spectral architecture designed specifically for algorithmic tasks: the Spectral

MPNN (Section 4.2). This model leverages the graph Fourier domain to construct spectral filters guided by the learned message-passing embeddings, thus combining the strengths of both paradigms—local feature propagation and global structural awareness.

1.4 Structure of the Dissertation

The thesis is structured as follows: Chapter 2 introduces the foundational concepts of Deep Learning on graphs and graph signal processing, which are essential for understanding the architectures discussed throughout the work. Chapter 3 provides an in-depth overview of the field of Neural Algorithmic Reasoning and presents the motivation for applying Spectral GNNs to this domain. Chapter 5 details the experimental results obtained using the proposed architecture and offers a thorough analysis of the learned embeddings, comparing them across different network types. Finally, Chapter 6 summarizes the main findings and outlines directions for future research.

Chapter 2

Deep Learning over Graphs

Deep Learning is constantly defined as the field of Artificial Intelligence concerned with the study of Deep Neural Networks—computational models originally inspired by the neural mechanisms of the human brain and primarily used for supervised learning tasks.

Originally, such a field received the name of Connectionist Artificial Intelligence, and its history goes back to the 1940s with Warren McCulloch and Walter Pitts's work [38]. They proposed a non-learnable computational model heavily inspired by the biological structure of the neurons, their model, although serving a different purpose from the nowadays Neural Networks is regarded as the very first connection between computational models and the human brain.

The McCulloch-Pitts model of the neuron was designed to describe brain functions in terms of logical clauses. However, it lacked any mechanism for learning from data. The introduction of learnability in early Neural Networks is credited to the work of Donald Hebb, who proposed Hebb's Rule [39] in Neurobiology. This rule suggests that the connection between two neurons strengthens when they are activated simultaneously, forming the basis of learning through association in artificial neural networks.

The Hebbian Rule, originally formulated within the context of neurobiology, inspired the development of learning rules for one of the first and most influential connectionist models in Artificial Intelligence: Rosenblatt's Perceptron [40]. Building upon the idea of the initial Perceptron, researchers hypothesized that stacking sequences of nonlinear activations could yield more powerful and expressive models [41]. This intuition, combined with the development of the backpropagation algorithm [42], which enabled the training of multilayer networks, rendered possible the rise of Deep Learning inside the field of Artificial Intelligence.

The evolution of deep learning has been significantly influenced by the incorporation of inductive biases [43]. These biases guide models to learn more effectively by embedding assumptions about the data, thereby enhancing generalization and

performance across various tasks.

Convolutional Neural Networks (CNNs) [44] introduce inductive biases such as locality and translation invariance, making them particularly effective for image processing tasks. Recurrent Neural Networks (RNNs) [45] and their variants, like Long Short-Term Memory(LSTM) networks [46] , incorporate biases toward sequential data, enabling them to capture temporal dependencies.

Graph Neural Networks (GNNs) [47] extend these concepts by embedding relational inductive biases, allowing models to learn from data represented as graphs, such as social networks or molecular structures [31]. These advancements demonstrate how incorporating specific inductive biases into architectures can lead to more efficient learning and improved performance over a diverse set of data domains.

This chapter introduces the deep learning concepts fundamental to this dissertation, focusing on two distinct types of neural networks on graphs: Spectral and Spatial Graph Neural Networks. While Spatial GNNs, also known as Message Passing Graph Neural Networks, are the dominant approach for graph-based tasks, we begin by presenting Spectral architectures. This choice is motivated by their foundation in Graph Signal Processing (GSP) theory [48], which is a powerful framework for understanding both Spectral and Spatial methods.

2.1 Spatial Graph Neural Networks

Research on deep learning for graphs has primarily focused on Spatial Graph Neural Networks (Spatial GNNs), commonly called Message Passing Neural Networks (MPNNs). These models learn node embeddings by propagating information from one node to another across the graph, using the representations of the previous layer to update each node's features [31, 47].

In a general, simplified formulation, an MPNN operates by iteratively updating node representations through three key components: message, aggregation, and readout functions [49].

- **Message function** (Ψ): During the message passing phase, each node collects messages from its neighbors. These messages are generated through a message function Ψ, which transforms the representation of each neighboring node (and potentially associated edge features) into a message to be sent. The design of Ψ plays a crucial role in determining what kind of information is communicated across the graph, for example, whether structural patterns, node attributes, or edge relationships are emphasized.

- **Aggregation function** ($\bigoplus$): Next, the aggregation function $\bigoplus$ combines the incoming messages. It is highly desirable that this step possess the prop-

erty of being permutation-invariant [50] to ensure the model generalizes across different node orderings. The choice of aggregation impacts the model's expressiveness: simple functions like summation or mean may fail to distinguish certain graph structures, whereas more complex or learned aggregators can enhance the model's discriminative power [51].

- **Readout function** (Φ): Finally, the readout function Φ uses the aggregated messages, along with the current state of the receiving node, to produce the updated embedding. This function governs how a node integrates external information with its internal state. Its design affects the model's ability to capture and propagate long-range dependencies and hierarchical information across layers.

The general equation for embedding update in message passing models is

$$h_i^t = \Phi\left(\bigoplus_{v_u \in \mathcal{N}(v_i)} \Psi(h_u^{t-1}), h_i^{t-1}\right) \tag{2.1}$$

where Ψ, Φ and $\bigoplus$ represent the message, aggregation, and readout functions, respectively. The term h_i^t denotes the embedding of node v_i at the t-th round of message passing, and $\mathcal{N}(i)$ denotes the set of neighbors of the node v_i. Equation 2.1 can be extended to incorporate information at the edge and the graph level, allowing richer representations of features [52].

One of the most prominent MPNN models is the GCN [53], which applies the following update rule for embedding computations:

$$h_i^t = \sigma\left(\frac{W^t}{|\mathcal{N}(v_i)|} h_i^{t-1} + \sum_{v_u \in \mathcal{N}(v_i)} \frac{W^t}{\sqrt{|\mathcal{N}(v_i)||\mathcal{N}(v_u)|}} h_u^{t-1}\right) \tag{2.2}$$

where W^t represents the learnable weight matrix at layer t and σ is a non-linear activation function. The mapping between the components of a GCN layer and the functions Ψ, Φ and $\bigoplus$ in Equation 2.1 is

$$\begin{aligned} \Psi(h) &= \frac{W^t}{\sqrt{|\mathcal{N}(v_i)||\mathcal{N}(v_u)|}} \cdot h \\ \Phi(h) &= \sigma\left(\frac{W^t}{|\mathcal{N}(v_i)|} h_i^{t-1} + h\right) \\ \bigoplus(\{h_u : v_u \in \mathcal{N}(v_i)\}) &= \sum_{v_u \in \mathcal{N}(v_i)} h_u \end{aligned}$$

By stacking the h_i^t embeddings of each node i into a matrix $H^t \in \mathbb{R}^{n,d}$, the GCN

layer can be implemented as

$$H^t = \sigma\left(D^{-1}\tilde{A}D^{-1}H^{t-1}W^{t-1}\right) \tag{2.3}$$

where D is the diagonal matrix of node degrees and $\tilde{A}$ is the adjacency matrix with self-loops.

The formulation in Equation 2.3 is preferable for dense adjacency matrices $\tilde{A}$, as it allows for fast embedding computation using GPUs.

Worked Example - GCN

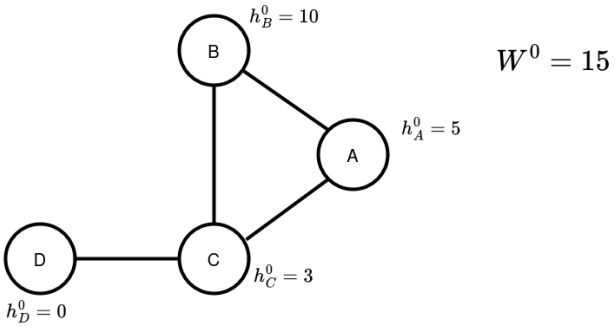


Figure 2.1: Input graph and initial embeddings for the GCN worked example.

For clarity, we will provide a worked example of one execution of the GCN layer, for nodes with one-dimensional embeddings. Figure 2.1, presents the input graph, defined by adjacency matrix A and initial embeddings h^0 for execution of the worked example. Moreover, we will assume the GCN layer with estimated parameter $W^0 = 15$ and non-linear activation function $\sigma = \text{ReLU}$.

$$A = \begin{array}{c} \\ \text{A} \\ \text{B} \\ \text{C} \\ \text{D} \end{array} \begin{array}{c} \begin{array}{cccc} \text{A} & \text{B} & \text{C} & \text{D} \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \end{array}$$

$$h^0 = \begin{array}{c} \begin{array}{cccc} \text{A} & \text{B} & \text{C} & \text{D} \end{array} \\ \begin{bmatrix} 5 & 10 & 3 & 0 \end{bmatrix} \end{array}$$

For calculation simplification all the calculated messages are rounded down to two decimal places.

1. **Node A update:**

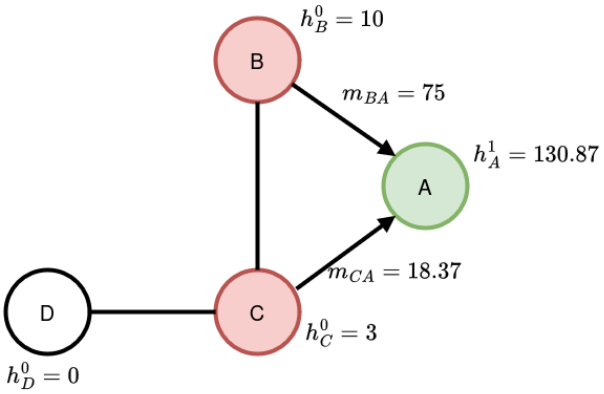


Figure 2.2: Embedding computation of node **A**.

The update of node **A** in the input graph, merges together messages from nodes **B** and **C**, following the GCN update rule (Equation 2.2), its subsequent embedding h_A^1 is

$$h_A^1 = \text{ReLU}\left(\frac{15}{2} \cdot 5 + \left(\frac{15}{\sqrt{2 \cdot 2}} \cdot 10 + \frac{15}{\sqrt{2 \cdot 3}} \cdot 3\right)\right)$$

$$h_A^1 = \text{ReLU}\left(37.5 + 75 + 18.37\right)$$

$$h_A^1 = 130.87$$

as the number of neighbors $\mathcal{N}(\text{A}) = 2$, $\mathcal{N}(\text{B}) = 2$ and $\mathcal{N}(\text{C}) = 3$.

2. **Node B update:**

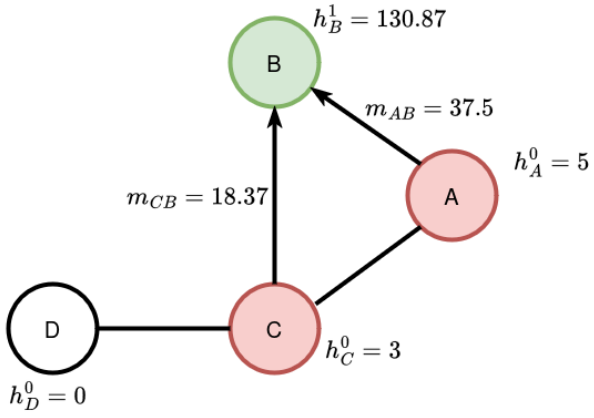


Figure 2.3: Embedding computation of node **B**.

The update of node **B**, merges together messages from nodes **C** and **A**, its

subsequent embedding h_B^1 is

$$h_B^1 = \text{ReLU}\left(\frac{15}{2} \cdot 10 + \left(\frac{15}{\sqrt{2 \cdot 2}} \cdot 5 + \frac{15}{\sqrt{2 \cdot 3}} \cdot 3\right)\right)$$

$$h_B^1 = \text{ReLU}\left(75 + 37.5 + 18.37\right)$$

$$h_B^1 = 130.87$$

as the number of neighbors $\mathcal{N}(\text{B}) = 2$, $\mathcal{N}(\text{A}) = 2$ and $\mathcal{N}(\text{C}) = 3$.

3. **Node C update:**

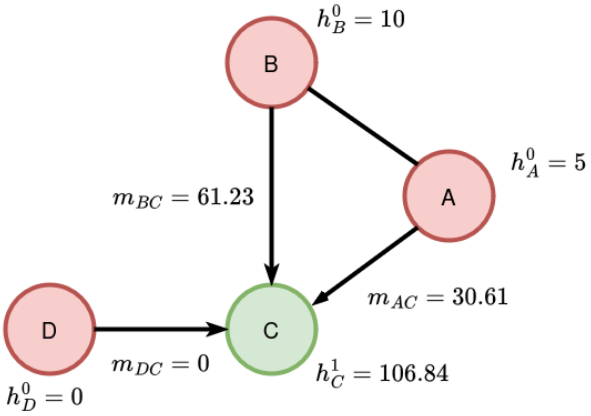


Figure 2.4: Embedding computation of node **C**.

The update of node **C**, merges together messages from nodes **A**, **B** and **D**, its subsequent embedding h_C^1 is

$$h_C^1 = \text{ReLU}\left(\frac{15}{3} \cdot 3 + \left(\frac{15}{\sqrt{3 \cdot 2}} \cdot 5 + \frac{15}{\sqrt{3 \cdot 2}} \cdot 10 + \frac{15}{\sqrt{3 \cdot 1}} \cdot 0\right)\right)$$

$$h_C^1 = \text{ReLU}\left(15 + 30.61 + 61.23 + 0\right)$$

$$h_C^1 = 106.84$$

as the number of neighbors $\mathcal{N}(\text{C}) = 3$, $\mathcal{N}(\text{A}) = 2$, $\mathcal{N}(\text{B}) = 2$ and $\mathcal{N}(\text{D}) = 1$.

4. **Node D update:**

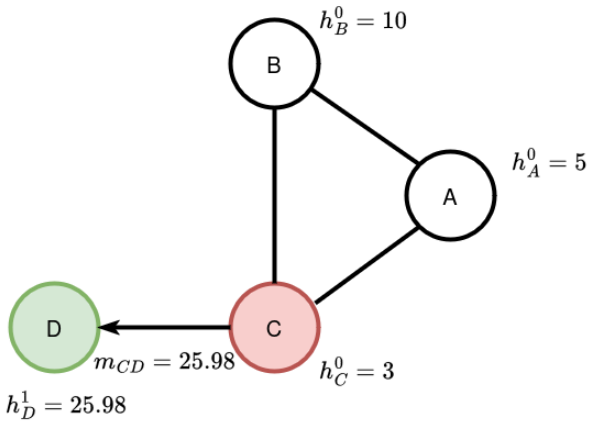


Figure 2.5: Embedding computation of node **D**.

The update of node **D**, receives messages only from node **C** as it is solely neighbor in the graph, its subsequent embedding h_D^1 is

$$h_D^1 = \text{ReLU}\left(15 \cdot 0 + \left(\frac{15}{\sqrt{3 \cdot 1}} \cdot 3\right)\right)$$

$$h_D^1 = \text{ReLU}\left(0 + 25.98\right)$$

$$h_D^1 = 25.98$$

as the number of neighbors $\mathcal{N}(\text{D}) = 1$ and $\mathcal{N}(\text{C}) = 3$

At the end of the first execution layer the calculated embeddings for each node are

$$h^1 = \begin{matrix} \text{A} & \text{B} & \text{C} & \text{D} \end{matrix} \\ \begin{bmatrix} 130.87 & 130.87 & 106.84 & 25.98 \end{bmatrix}$$

The produced embeddings h^1 can then be fed to a subsequent message passing module or be further processed for prediction of a given task.

2.1.1 Over-smoothing in Spatial GNNs

A common phenomenon observed when training Spatial GNNs is over-smoothing, where node representations converge to the same embedding as the number of layers in the network increases [33]. Understanding and mitigating over-smoothing has been a prominent focus in graph learning research, with theoretical works examining it [54–56] and practical attempts to alleviate its effects [57–59].

Over-smoothing places a burden on the depth of Spatial GNNs, reducing the model's ability to capture complex patterns within the graph structure. An MPNN's

ability to encode long-range graph structural information relies on how many message passing layers it possesses, by limiting the GNN's depth, over-smoothing hampers their capacity to effectively encode global information, as capturing such structures relies on multi-hop information propagation [36].

This phenomenon is intrinsically associated with the low-pass filter property of Spatial GNNs [34, 35]. The low-pass filter issue in GNNs refers to their tendency to favor or amplify low-frequency signals (i.e., smooth variations across the graph) while suppressing high-frequency signals (i.e., sharp differences between neighboring nodes). This phenomenon is rooted in how GNNs aggregate information from neighbors of a node during message passing. Notably, the graph Laplacian, described in Section 2.2.1, smallest eigenvalue is always zero, with the corresponding eigenvector consisting of equal values, representing the component with the lowest frequency.

Due to such a characteristic, Spatial GNNs rely on the homophily of input graphs. Homophily is a fundamental property in many real-world datasets: it assumes that connected nodes often belong to the same class or present similar features [60]. The inductive homophily bias of spatial GNNs can place a burden on their performance under heterophilic scenarios (i.e. settings where connected nodes present dissimilar classes). Recent lines of research show how graph-agnostic models, such as an ordinary MLP, can outperform MPNNs in several heterophilic scenarios [61–64].

2.2 Spectral Graph Neural Networks

Spectral GNNs are a distinct approach to the dominant message passing framework for learning on graphs. Unlike Spatial Architectures, defined in the Section 2.1, their definition of convolution on graphs relies on the concepts of Spectral Graph Theory [65]. While convolution in MPNNs works spatially, by aggregating representations in a node's neighborhood, Spectral architectures act directly on the Graph Fourier Transform by attributing different weights to features in the frequency domain.

2.2.1 Graph Signal Processing

Graph Signal Processing (GSP) [48] is the field connecting Signal Processing and Graph Theory. The development of GSP has provided researchers with tools to study and process signals in more general and complex domains, such as social networks, sensor networks, and biological systems, where data can naturally be encoded as graphs.

This advancement was made possible through the study of an object in Graph Theory: the graph Laplacian and its eigenvalue decomposition.

Graph Laplacian

Definition 2.1 (Laplacian Matrix) *[47] Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a given graph with adjacency matrix A and $|\mathcal{V}| = n$, its Laplacian Matrix L is defined as:*

$$L = D - A \tag{2.4}$$

where D is the diagonal degree matrix $D = diag(d(v_1), d(v_2), ..., d(v_n))$

Let x be a vector where $x_i \in \mathbb{R}$ is the element associated to $v_i \in \mathcal{V}$ and $\mathcal{N}(v_i)$ be the neighborhood of node v. A fundamental property of the graph Laplacian L is the following

$$x^T L x = \frac{1}{2} \sum_{v_i \in \mathcal{V}} \sum_{v_j \in \mathcal{N}(v_i)} (x_i - x_j)^2 \tag{2.5}$$

.

A proof for Equation 2.5 can be found on Appendix A Proof A.1.

Equation 2.5 states that $x^T L x$ is the sum of squared differences between adjacent nodes. In other words, $x^T L x$ quantifies how distant signals in adjacent nodes are from each other. Consequently, it serves as a measure of the smoothness of the graph signal, with smaller values indicating smoother signals over the graph.

Eigenvalue Decomposition

The graph Laplacian L admits an eigenvalue decomposition $L = U \Lambda U^T$ [66, 67], where $\Lambda = \text{diag}(\lambda_1, \lambda_2, ..., \lambda_n)$ is a diagonal matrix with the eigenvalues of L and U is a square $n \times n$ matrix where the ith column is u_i the ith eigenvector of L. If L is symmetric (graph is undirected), the eigenvalues $\{\lambda_i\}_{i=1}^n$ are real and usually ordered $\lambda_1 < \lambda_2 < ... < \lambda_n$.

Graph Fourier Transform

Similar to the traditional Fourier Transform, the Graph Fourier Transform (GFT) [68] aims at decomposing a graph signal into a weighted sum of orthogonal bases. From the observation that $\lambda_i = u_i^T L u_i$, we note that the eigenvalues of L represent the frequency/smoothness of the corresponding eigenvectors. Such observation naturally gives rise to a definition of Fourier Transform in graphs.

Definition 2.2 (Graph Fourier Transform) *Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a given graph with graph Laplacian $L = U \Lambda U^T$, the Graph Fourier Transform (GFT) and inverse GFT of a signal x over $\mathcal{G}$ are, respectively*

$$\hat{x} = U^T x, \; x = U \hat{x} \tag{2.6}$$

where $\hat{x}$ are the Fourier coefficients of the signal x.

Definition 2.2 is consistent with the traditional definition of the traditional Fourier Transform as a decomposition of the signal on the eigenfunctions of the 1-dimensional Laplace operator [68].

From the perspective of GFT, filtering a graph signal can be defined in the frequency domain, by a function $g(\lambda)$ which amplifies or attenuates each of the frequency components $\{\lambda_i\}_{i=1}^n$ [69]

$$h = Ug(\Lambda)U^T x \tag{2.7}$$

where h is the output of the filter and $g(\Lambda) = \text{diag}(g(\lambda_1), g(\lambda_2), ..., g(\lambda_n))$

For a more thorough presentation of Graph Signal Processing, see Appendix A and the works [47, 48, 65, 68].

2.2.2 Spectral GNN architectures

The idea of a Spectral GNN's emerges from the definition of a graph filter in Equation 2.7. By having a filter g_θ, parametrized by θ, one could apply gradient-based techniques to optimize a loss function with respect to θ. Different neural network architectures define different structural choices for g_θ and promote different kinds of inductive bias [70, 71].

For example, one of the first networks that works as a graph spectral filter was proposed in [72]. Following the terminology mentioned above, a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is embedded with a node signal $h^0 \in \mathbb{R}^{n \times d}$, where $n = |\mathcal{V}|$ is the number of nodes.

The proposed network follows a construction where each of the layers $l = 1, \ldots, K$ with hidden dimensions $\{d_l\}_{l=0}^K$, generates a hidden representation $h_j^l \in \mathbb{R}^n$ for each feature j in the following way

$$h_j^t = \sigma\left(U \sum_{i=1}^{d_{t-1}} G_{i,j}^t U^T h_i^{t-1}\right),\ j = 1...d_t \tag{2.8}$$

where $G_{i,j}^l$, with $i = 1, \ldots, d_{l-1}$ and $j = 1, \ldots, d_l$, is a diagonal matrix with trainable parameters $\{\theta_i^l\}_{i=1}^n$

$$G_{i,j}^l = [\theta_1^l, \theta_2^l, \ldots, \theta_n^l]I$$

One of the main drawbacks regarding Spectral GNNs is the reliance on eigenvalue decomposition of the Laplacian matrix, which is of complexity $O(n^3)$, where n is the size of the graph [66]. The computational complexity of the eigenvalue decomposition places a burden on wide application of Spectral architectures on larger graphs.

Polynomial filters

To circumvent direct decomposition of the graph Laplacian, current approaches implement filtering in the frequency domain as polynomials of the graph Laplacian L [37, 73]. Polynomial filters are weighted sums of powers of the eigenvalue matrix Λ, defined as

$$g_\theta(\Lambda) = \theta_0 I + \theta_1 \Lambda + +\theta_2 \Lambda^2 + ... + \theta_K \Lambda^K$$

and can be seen as a special case of Equation 2.7, where the filter $g_\theta(\Lambda)$ is a polynomial function.

By noting that

$$\begin{aligned} U g_\theta(\Lambda) U^T &= U(\theta_0 I + \theta_1 \Lambda + \theta_2 \Lambda^2 + ... + \theta_K \Lambda^K) U^T \\ &= \theta_0 U I U^T + \theta_1 U \Lambda U^T + \theta_2 U \Lambda^2 U^T + ... + \theta_K U \Lambda^K U^T \\ &= \theta_0 I + \theta_1 L + \theta_2 L^2 + ... + \theta_K L^k \end{aligned}$$

it is possible to see that the design of polynomial filters in the spectral domain can be done directly as polynomials of the Laplacian L, through $g_\theta(L) = \theta_0 I + \theta_1 L + \theta_2 L^2 + ... + \theta_K L^k$.

However, one major drawback of plain implementation of polynomial filters is the non-orthogonality of the polynomial terms $\{I, L, L^2, \ldots, L^k\}$, which may affect model convergence. One approach, taken by [73], uses Chebyshev expansion to construct a polynomial filter. It has been shown that the orthogonality of Chebyshev terms greatly assists training [70]. By updating the embeddings the same way as Equation 2.8, ChebNet constructs the matrix $G^t_{i,j}$ as

$$G^t_{i,j} = \sum_{i=1}^{K} \theta^t_{i,j,k} T_k(\tilde{L}) \tag{2.9}$$

where $\tilde{L} = \frac{2L}{\lambda_{\max}} - I$ is the scaled Laplacian, and $T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x)$, with $T_0(x) = 1$ and $T_1(x) = x$, the Chebyshev terms. And the j-the feature of embedding $h^t_j \in \mathbb{R}^n$ is calculated as

$$h^t_j = \sum_{i=1}^{d_{t-1}} G^t_{i,j} h^{t-1}_i$$

.

In comparison to Spectral GNNs, the message passing scheme of Spatial GNNs differ in several aspects.

- **Spatial and spectral architectures encode different information**: While spatial GNNs aggregate node features layer by layer, emphasizing locality and restricting a node to capture information within a fixed distance,

spectral GNNs work on the Eigenvalue Decomposition of the graph Laplacian, which naturally carries structural information.

- **Spatial GNNs are biased toward low-pass filtering [34, 35]**: Theoretical works on spatial GNNs (more specifically GCN [53]) have shown how such models bias the learned embeddings towards low-frequency components. Message passing models work by iteratively multiplying the augmented adjacency matrix, which resembles a low-pass filter on graphs [35]; this issue also leads to the phenomenon of oversmoothing in spatial GNNs [33]. However, constructing spectral GNNs allows them to learn arbitrary filters on graphs, filtering out the appropriate frequencies for the task at hand.

- **Spectral architectures are less scalable**: The time complexity of the eigenvalue decomposition of the Laplacian graph places a burden on the general applicability of spectral GNNs. Current approaches attempt to avoid explicit computation of the Fourier features by approximating the graph Laplacian decomposition with polynomial filters [37, 73].

2.3 Permutation-Equivariance in Graph Neural Networks

A desirable property of modules that leverage the graph structure for embedding computation is permutation equivariance [50]. Unlike lists, a graph's structure, defined by its adjacency matrix, is invariant to any particular ordering of the nodes.

Therefore, any reasonable model operating on a graph should produce node embeddings that are invariant to the order in which the nodes are listed.

Definition 2.3 (Permutation Equivariance in GNNs) *Consider a network module $\mathcal{N}$ operating on a graph with N nodes. The module receives as input an adjacency matrix $A \in \{0,1\}^{N\times N}$ and an embedding matrix $H^t \in \mathbb{R}^{N\times d}$, and produces an output embedding matrix $H^{t+1} \in \mathbb{R}^{N\times d}$. The module $\mathcal{N}$ is said to be permutation equivariant if, for any permutation matrix $P \in \{0,1\}^{N\times N}$,*

$$PH^{t+1} = \mathcal{N}(PAP^T, PH^t)$$

Permutation equivariance ensures that the learned representations depend only on the graph topology and features—not on how data is arranged in memory and serves as additional inductive bias for tasks being executed over graphs [50].

Any architecture fitting into the design space of Spatial GNNs (Equation 2.1) [49] or the update function of Spectral GNNs (Equation 2.7) has the property of

being permutation equivariant over the set of nodes.

2.4 Out-of-Distribution (OOD) generalization

The techniques discussed earlier in this chapter have demonstrated performance surpassing human-level capabilities on specific tasks [74, 75]. However, these models are typically evaluated in controlled environments that may not fully reflect the distributions encountered during real-world deployment. As a result, there is a growing body of research focused on enhancing model robustness to distribution shifts.

Out-of-Distribution (OOD) generalization refers to the ability of a model to be robust to distribution shifts on test data [25]. Despite the importance of OOD generalization, most standard machine learning algorithms are theoretically grounded in the *i.i.d.* assumption, presuming that the train and test samples are independent and identically distributed, violating such a established assumption requires the development of novel techniques that extend beyond traditional ML.

The problem with the current optimization framework is that it is solely focused on minimizing an error metric from training samples, consequently exploiting spurious correlations and confounding factors, without necessarily capturing the underlying true relationships among the data. From this perspective, OOD generalization can be seen as attempt to drive models towards capturing the relationships that exist under any environment, i.e. the causal relationships.

Given the interest in OOD generalization, many recent works have attempted to formalize it into a theoretical framework [76, 77]. One of the most prominent approaches relies on the theory of Invariant Learning, i.e. a predictor which is invariant across different environments. By focusing on the aspects of the predictors that do not change with environments, models are able to better generalize to unseen environments.

The invariance assumption can be expressed mathematically in various ways [78–80]. A common feature across all works attempting to formalize Invariant Learning is its intrinsic connection to the theory of causation. In fact, invariance can be viewed as a relaxation of causality. By assuming that the Structural Equation Model (SEM) B remains the same across all environments Assumption 2.1 and treating environments as interventions on covariate variables, the problem of Invariant Learning reduces to the issue of identifiability [81] in Causal Inference.

Assumption 2.1 *[82] The environment-specific structural equation model:*

$$Y^e \leftarrow f_Y(X^e_{pa(Y)}, \epsilon_Y) \tag{2.10}$$

remains the same across all possible environments $e \in \mathcal{E}$, that is f_Y and $pa(Y)$ are environment independent functions and ϵ_Y has the same distributions for all environments.

By allowing Y^e and X^e distributions to change with the environments $e \in \mathcal{E}$, the interest relies in modeling the function f_Y and identifying the parents of Y in the causal graph $pa(Y)$. In other words, from a causal perspective, OOD generalization involves modeling the interventional distribution $P(Y|do(X = x))$, allowing the model to remain robust under changes to the distribution of input variables across environments.

Beyond its theoretical formalization, successful Out-of-Distribution (OOD) generalization fundamentally relies on identifying the true causal relationships between exogenous and endogenous variables, rather than exploiting spurious correlations present in the data. The main topic of this dissertation, Neural Algorithmic Reasoning, addresses OOD Generalization with the objective to learn such causal and invariant structures over algorithmic tasks. The specific way in which OOD Generalization is explored in Neural Algorithmic Reasoning is described Section 3.2.2.

Chapter 3

Neural Algorithmic Reasoning

The current chapter addresses the field of Neural Algorithmic Reasoning [4], the main topic of this dissertation. Neural Algorithmic Reasoning (NAR) is concerned with the task of embedding an artificial neural system with the ability to execute algorithms. More specifically, the kind of computation found on undergraduate classical algorithms textbooks [83], for the purposes of this work coined here as "Classical Computation", is a form of breaking a complex and intricate problem into smaller simple problems with abstract inputs.

3.1 Algorithmic Alignment [1]

One of the landmark papers of the field, "*What Can Neural Networks Reason About?*" [1] discourses about the issue of Algorithmic Alignment and which architectural aspects of Neural Networks contribute to it. The authors main argument relies on the (contestable, however reasonable) assumption that reasoning processes resemble algorithm execution. Therefore, a network exhibiting good Algorithmic Alignment would only need to learn small and simple algorithm steps to thrive on reasoning tasks.

Moreover, intuitively and as seen in previous works [84, 85], there exists an intrinsic relation between reasoning and the generalization ability of current artificial neural networks. The article builds upon this idea by showing that the Algorithmic Alignment improves sample complexity as defined in the PAC Learning framework [86].

Definition 3.1 (PAC Learning and Sample Complexity [86]) *For a fixed error parameter $\epsilon > 0$ and failure probability $\delta \in [0, 1]$, a function g is said to be PAC-learnable if there exists a learning algorithm $\mathcal{A}$ that can generate a function $f = \mathcal{A}(\{(x_i, g(x_i))\}_{i=0}^{M})$ from M i.i.d. samples $\{(x_i, g(x_i))\}_{i=0}^{M}$ drawn from distribu-*

tion $\mathcal{D}$. The function g is (M, ϵ, δ)-learnable with $\mathcal{A}$ if

$$\mathbb{P}_{x\sim\mathcal{D}}(\|f(x) - g(x)\| < \epsilon) < 1 - \delta$$

The sample complexity $\mathcal{C}_{\mathcal{A}}(g, \epsilon, \delta)$ is the minimum M for which g is (M, ϵ, δ)-learnable with $\mathcal{A}$.

The PAC-learning framework (Definition 3.1) serves as a foundation for defining a measure of Algorithmic Alignment. Formally, [1] asserts that neural networks with a limited number of modules can simulate an algorithm's execution with low sample complexity.

Definition 3.2 (Algorithmic Alignment [1]) *Given a fixed sample size M, let g be a function decomposable into n modules, $f_1 \circ f_2 \circ ... \circ f_n$, and let $\mathcal{N}$ be a neural network composed of n corresponding modules, $\mathcal{N}_1, ..., \mathcal{N}_n$. Then, $\mathcal{N}$ (M, ϵ, δ)-algorithmically aligns with g if there exist learning procedures $\{\mathcal{A}_i\}_{i=0}^n$ for the $\mathcal{N}_i$'s such that*

$$n \times \mathcal{C}_{\mathcal{A}_i}(f_i, \epsilon, \delta) \leq M$$

Different choices of learning procedures $\{\mathcal{A}_i\}_{i=0}^n$ and network modules $\{\mathcal{N}_i\}_{i=0}^n$ can lead to better Algorithmic Alignment, that is, smaller M.

From the definition of Algorithmic Alignment, the authors of [1] prove that an algorithmic procedure g can be PAC-learned, under the assumption of a network $\mathcal{N}$ algorithmically aligned to g. More generally, they show that under some mild conditions, an algorithmically aligned model achieves reasonable sample complexity, i.e. good generalization.

Theorem 3.1 (Algorithmic Alignment improves Sample Complexity [1]) *For a fixed error parameter $\epsilon > 0$ and failure probability $\delta \in [0, 1]$. Let $\{x_i, g(x_i)\}_{i=1}^M \sim \mathcal{D}$ be a dataset sampled from distribution $\mathcal{D}$. Suppose $\mathcal{N}_1, ..., \mathcal{N}_n$ are the network's $\mathcal{N}$ MLP modules and $\mathcal{N}$ and g (M, ϵ, δ)-algorithmically align through functions $f_1, f_2, ..., f_n$. Under the listed conditions, g is $(M, O(\epsilon), O(\delta))$-learnable by $\mathcal{N}$.*

1. ***Algorithm Stability**: Let $\mathcal{A}$ be the learning algorithm for $\mathcal{N}_i$'s. Assume $f = \mathcal{A}(\{x_i, g(x_i)\}_{i=1}^M)$ and $\hat{f} = \mathcal{A}(\{\hat{x}_i, g(x_i)\}_{i=1}^M)$. For any x, $\left\|f(x) - \hat{f}(x)\right\| \leq L_0 \max_i \|x_i - \hat{x}_i\|$, for some L_0.*

2. ***Sequential Learning**: The network modules are trained sequentially, that is, the first module $\mathcal{N}_1$ has input samples $\{\hat{x}_i^1, f_1(x_i^1)\}_{i=1}^M$. For subsequent modules, $j > 1$, the input $\hat{x}_i^j$ of module $\mathcal{N}_j$ is the output of the previous modules, and the labels are the ground-truth correct labels $(f_1 \circ f_2 \circ ... \circ f_j)(x_i^1)$.*

3. ***Lipschitzness:*** *The learned functions* $\hat{f}_j$ *satisfy* $\left\|\hat{f}_j(x) - \hat{f}_j(\hat{x})\right\| \leq L_1 \|x - \hat{x}\|$, *for some* L_1.

The proof for the previous proposition is provided in Appendix C, as well as in the work where it was originally stated [1].

Theorem 3.1 provides a guarantee that an algorithmic procedure g can be effectively learned by a network $\mathcal{N}$ if they are both algorithmically aligned. Moreover, it not only provides a guarantee of learnability, but also states that the algorithmic alignment value M defines the sample complexity in the PAC learning framework, that is, with fixed ϵ and δ better algorithmic alignment leads to better sample complexity of the algorithmic procedure g.

3.2 The CLRS Algorithmic Reasoning Benchmark [2]

Historically, benchmark datasets have been of fundamental importance in the field of Deep Learning, serving as standardized tools for evaluating and comparing novel state-of-the-art models. Examples range from image modeling, such as the ImageNet [87] and MS COCO [88], to text translation [89] and general reinforcement learning [90].

Benchmark datasets are crucial not only for providing a common reference point so as to compare different deep learning models. They also play a role in driving the development of the field as a whole. By creating handcrafted datasets, the research community is able to tackle the specific aspects of problems where Neural Networks struggle. A notable recent example is the Abstraction and Reasoning Corpus [20], which attempts to encode the challenges of Artificial General Intelligence by evaluating the efficiency of skill acquirement from models.

Aiming at the implementation of such a benchmark for Neural Algorithmic Reasoning, the newly proposed CLRS30 Benchmark [2] has been developed. The dataset consists of sample executions from 30 cherry-picked algorithms from the seminal book *Introduction to Algorithms* [83]. These algorithms are divided in a taxonomy of 8 categories:

Sorting: Insertion Sort, Bubble Sort, Heapsort [91] and Quicksort [92].

Searching: Minimum, Binary Search and Quickselect [93]

Divide and Conquer: Find Maximum Subarray [94].

Greedy Algorithms: Activity selection [95] and Task Scheduling[96].

Dynamic Programming: Matrix chain multiplication, Longest Common Subsequence and Optimal Binary Search Tree [97].

Graphs: Depth and Breadth first searches, Topological Sorting [98], Articulation Points, Bridges, Kosaraju's strongly-connected components algorithm [97], Kruskal's and Prim's Minimum Spanning Tree algorithms [99, 100], Bellman-Ford and Dijkstra's algorithm for shortest path [101, 102] and the Floyd-Warshall algorithm for all-pairs shortest path [103].

Strings: Naive string matcher and Knuth-Morris-Pratt (KMP) string matcher [104].

Geometry: Segment intersection and Graham's and Jarvis' planar convex hull algorithms [105, 106].

3.2.1 Structure of the Dataset

Graphs are highly general and versatile algebraic structures, capable of representing a wide variety of data and relationships. This flexibility enables them to encode numerous data structures commonly used in traditional algorithms. Furthermore, theoretical results have demonstrated that graph neural networks align well with the principles of dynamic programming [32]. As a conclusion from these observations, the developers of CLRS30 [2] opted to implement the dataset using a graph-oriented approach.

Algorithms are represented as a set of vertices and edges, with information (features) over them which describes the steps of the algorithm's execution. Every feature over the graph respect the following categorization:

Stage: Each feature is associated with a specific stage, representing a step in the algorithm's trajectory. The information required for algorithm execution is classified into three types: *inputs*, *outputs*, and *hints*. Features categorized as *inputs* or *outputs* correspond to single-step information, while *hints* provide a time-series of states that capture the evolution of the data structure throughout the algorithm's execution.

Location: Each feature exists in either *node*, *edge* or *graph* level.

Type: Each feature belongs to one of five data types, these types define the appropriate way to encode and decode it, as well as the specific loss function to be applied for its prediction.

scalar: Floating-point scalar value; i.e. edge weights, positions.

categorical: Categorical feature over a determined number of possible classes, each node is attributed a one-hot vector with a single 1 in the index associated to the class.

mask: Categorical feature over two classes, each node is attributed a value of 0 or 1.

mask_one: Categorical feature over two classes, each node is attributed a value of 0 or 1, with the constraint that a single node has value 1 and all the others are 0.

pointer: Categorical feature over the n nodes.

3.2.2 How the CLRS30 benchmark addresses OOD generalization

As stated earlier, the concerns of Neural Algorithmic Reasoning go beyond accurate predictions of input-output pairs [1], the issues lie in the ability of neural systems to precisely mimic algorithmic execution, i.e. processing of discrete information.

Moreover, it has been empirically seen that plain prediction of input-output pairs limits the model's generalization exclusively to in-distribution samples [107–109]. For instance, a neural network trained to perform sorting on arrays within a specific length range in the training dataset often struggles when tasked with inputs that fall outside this range.

Generalization is not a concern in classical computer science. If provided with proof-of-correctness, algorithms can guarantee generalization to any input in agreement with the expected input structure. Ideally, the same property should be observed in neural models mimicking algorithmic execution.

The mechanism to better approximate neural networks and algorithms, beyond input-output prediction, is through *hint* features present in the CLRS30 dataset. *Hints* are the kinds of features in the dataset that capture intermediate states throughout algorithmic execution. Unlike *inputs* and *outputs*, *hints* include an additional dimension for **time** or **steps**, and can thus be seen as a time series of states an algorithm goes through while it is executing.

Such *hint* features are presented as intermediate targets the network is trained to predict, successful prediction of *hints* is a great indicator the network has captured the underlying reasoning process of the presented algorithm.

In addition, it is theoretically reasonable to believe that predicting *hints* constrains the optimized parameters of neural networks to align more closely with the actual algorithmic execution, steering them away from confounding factors and spurious correlations present in input-output pairs. Such observation has also been

empirically demonstrated in [110–112].

3.2.3 Example: *Breadth-first search*

The *Breadth-first search* algorithm [113] is one of the simplest and most fundamental algorithms for graph traversal. Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and a source node $s \in \mathcal{V}$ as inputs, BFS explores the graph by systematically visiting nodes in layers. Starting with the source node, it adds all its neighbors to a queue. Nodes are then explored in the order they are added to the queue, with the earliest added nodes being explored first.

In the first iteration, all neighbors of the source node s are enqueued. In each subsequent iteration, a node $v \in \mathcal{V}$ is dequeued, and its neighbors are then enqueued. This process continues until every node in the graph has been visited. A pseudocode of the algorithm is available at Algorithm 1 in Appendix D.

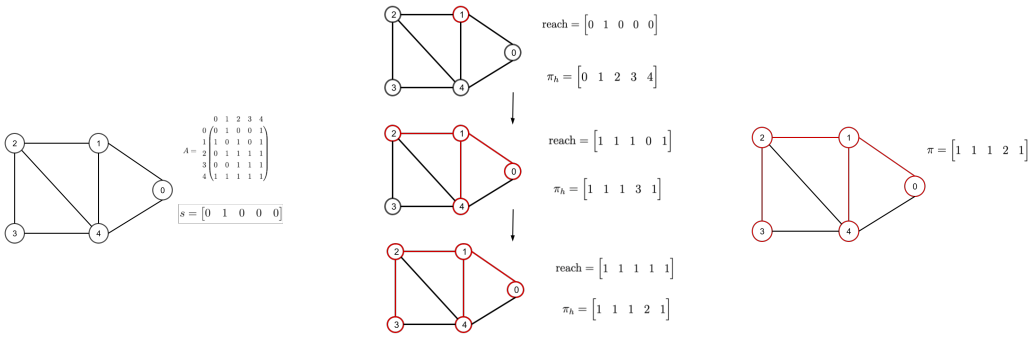
Breadth-first search algorithm is represented within the CLRS30 framework with the following features and their specification is presented in Table 3.1:

- s: The source node from where the search is going to start.
- A: The adjacency matrix of the input graph.
- π: A list with one entry for each of the nodes, indicating a node's predecessor in the graph.
- reach: A binary-valued list with one entry for each of the nodes, with 0's for nodes that haven't been reach yet and 1's for all the nodes which have already been reached.
- π_h: A list with one entry for each of the nodes, indicating a node's predecessor in the graph. Such feature differ from the original π in the sense that it exists in the *hints* level and is updated along the algorithm's trajectory

Feature	**Stage**	**Location**	**Type**
s	*inputs*	*nodes*	*mask_one*
A	*inputs*	*edges*	*mask*
π	*outputs*	*nodes*	*pointer*
reach	*hints*	*nodes*	*mask*
π_h	*hints*	*nodes*	*pointer*

Table 3.1: Specification of the *Breadth-first search* algorithm.

A worked example of the BFS algorithm is shown in Figure 3.1. The algorithm begins with the standard inputs for Breadth-First Search, the graph, represented



(a) BFS *inputs* (b) BFS *hints* (c) BFS *outputs*

Figure 3.1: BFS Worked Example

by its adjacency matrix A, and the source node s. In the CLRS30 framework, the source node s is of type *mask_ one*, meaning it is encoded as a binary vector of length equal to the number of nodes in the graph. This vector contains all zeros except for a single one at the index corresponding to the source node.

During the execution, the algorithm maintains two auxiliary lists, reach and π_h, both categorized as *hints* in the CLRS30 framework, as shown in Figure 3.1. The reach list is of type *mask*, and is represented as a binary vector, entries corresponding to visited nodes are set to 1, while unvisited nodes remain at 0. The π_h list stores the parent (or predecessor) of each node according to the breadth-first search traversal. It is of type *pointer*, meaning it is a list of integers where each entry indicates the index of the parent node in the input graph.

The execution traces of the *hints* for the input graph presented in Figure 3.1a is

$$\text{reach}^1 = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \end{bmatrix} \rightarrow \text{reach}^2 = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 \end{bmatrix} \rightarrow \text{reach}^3 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

$$\pi_h^1 = \begin{bmatrix} 0 & 1 & 2 & 3 & 4 \end{bmatrix} \rightarrow \pi_h^2 = \begin{bmatrix} 1 & 1 & 1 & 3 & 1 \end{bmatrix} \rightarrow \pi_h^3 = \begin{bmatrix} 1 & 1 & 1 & 2 & 1 \end{bmatrix}$$

The output of the algorithm is a list of parent nodes for each node in the graph, following the BFS traversal. This output is stored in a separate list, denoted by π, which matches the final state of π_h at the end of the algorithm's execution.

A single datapoint in the CLRS30 benchmark consists of three components: *inputs*, *hints*, and *outputs*. In the case of the input graph shown in Figure 3.1a, this datapoint would be represented by the following data structure:

$$inputs: \begin{cases} A = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} \\ s = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \end{bmatrix} \end{cases}$$

$$hints: \begin{cases} \text{reach} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} \\ \pi_h = \begin{bmatrix} 0 & 1 & 2 & 3 & 4 \\ 1 & 1 & 1 & 3 & 1 \\ 1 & 1 & 1 & 2 & 1 \end{bmatrix} \end{cases}$$

$$outputs: \begin{cases} \pi = \begin{bmatrix} 1 & 1 & 1 & 2 & 1 \end{bmatrix} \end{cases}$$

3.2.4 Encode-Process-Decode

An important aspect for solving the CLRS30 benchmark is the kind of architecture being applied to the problem. Graph Neural Networks from Chapter 2 arise as natural models for such tasks, as many algorithms within the dataset are inherently designed to operate on graph-structured data.

However, plain application of a GNN does not fully fulfill the goals of Neural Algorithmic Reasoning. This limitation stems from the nature of the data in the CLRS30 benchmark, where inputs and intermediate states correspond to algorithms operating in a discrete domain. Neural networks, on the other hand, are designed to process continuous vector representations, and a growing body of research highlights the suboptimal performance of neural networks when working directly with discrete data [114].

Furthermore, we would like models trained for algorithmic tasks, to be generalizable over algorithms and over tasks that closely resemble algorithmic execution [4]. In a way, that a single model could be able to execute many distinct algorithms [115] while also being fine-tuned to handle real-world tasks across a diverse range of inputs.

In order to overcome such difficulties with plain application of GNNs, the usual framework used within the CLRS30 is the Encode-Process-Decode [116], where in-

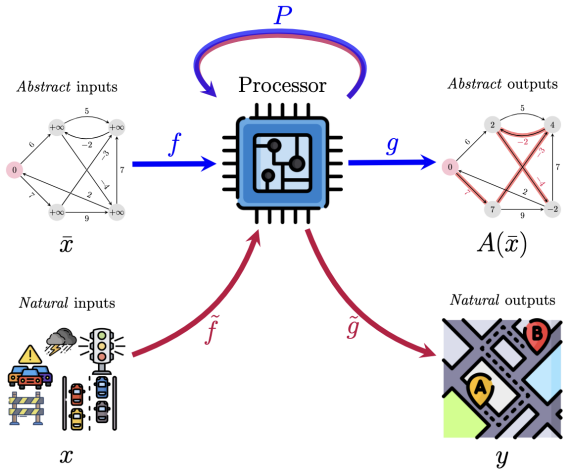


Figure 3.2: **The Encode-Process-Decode paradigm**. Image from [4]

puts are initially transformed into continuous vector representations by an **Encoder** model. These representations are then passed to a **Processor**, typically a GNN, which processes the data. Finally, the **Decoder** converts the outputs back into the discrete states of the algorithm.

Such a framework provides the flexibility to independently design and optimize each component—**Encoder**, **Processor**, and **Decoder**—based on the specific requirements of the task. If the interest relies on developing a single model capable of executing a diverse set of algorithms $\mathcal{A} = \{A, B, C, ...\}$, this framework allows you to develop algorithm specific encoder and decoders, respectively $f_{\mathcal{A}} = \{f_A, f_B, f_C, ...\}$, $g_{\mathcal{A}} = \{g_A, g_B, g_C, ...\}$, and a single **Processor** P capable of processing inputs from every encoder $f \in f_{\mathcal{A}}$ and providing representations for every decoder $g \in g_{\mathcal{A}}$ [115].

On the other hand, applying models trained on algorithmic executions to natural inputs is also possible. Suppose a model $g(P(f(x)))$ trained to predict x, where x is an algorithmic input, now suppose $\tilde{x}$ is a natural input in a task that closely resembles the algorithm generating x. By fixing the processor P and carefully constructing encoder $\tilde{f}$ and decoder $\tilde{g}$ for processing raw data input $\tilde{x}$, the model could be fine-tuned for this downstream task [30, 117–119].

Encoder and Decoder networks

There is no restriction as to the kind of network used for encoding or decoding the algorithmic states. However, so as to keep the models as simple as possible and place focus on the **Processor**, usually such models are plain matrix multiplications with no non-linearities (linear regressions). All of the **Encoder** and **Decoder** models throughout this dissertation follow the same implementation from the open github repository `https://github.com/google-deepmind/clrs` of [2] with changes only to match the distinct frameworks used.

The **Encoder** network f, is composed of $k = 1, 2, ..., n$ distinct linear layers W_k one for each of the algorithm input features $x_{\text{input},k}$, f then works by summing up each of these linear transformations

$$f(x_{\text{input}}) = \sum_{k=0}^{n} W_k x_k$$

The latent generated features $h = f(x_{\text{input}})$ of arbitrary dimension will the be fed to the **Processor**.

One important remark, is that GNNs can process features in the node, edge and graph levels, and algorithmic features also exist in all three levels as stated in Section 3.2.1. However, while a graph is composed of n nodes it has at most $\binom{n}{2}$ edges. Therefore, due to dimension mismatches these latent features cannot be summed, and we maintain separate representations for nodes, edges and graph features.

$$h_i = f^{\text{node}}(x_{\text{input, i}}) = \sum_{k=0}^{n} W_k^{\text{node}} x_{k,i}, \forall i \in \mathcal{V}$$

$$h_{ij} = f^{\text{edge}}(x_{\text{input},ij}) = \sum_{k=0}^{n} W_k^{\text{edge}} x_{k,ij}, \forall i, j \in \mathcal{V}$$

$$h_{\text{graph}} = f^{\text{graph}}(x_{\text{graph}}) = \sum_{k=0}^{n} W_k^{\text{graph}} x_k$$

where i, j are indices over the set of nodes $\mathcal{V}$ with cardinality $|\mathcal{V}| = N$. The encoding process generates three latent tensors of arbitrary latent dimension d and sizes (N, d), (N, N, d) and (d), for nodes, edges and the graph respectively.

Analogously, **Decoder** models work in a similar way. For a given algorithm, there exists one decoder g_i for each predicted feature i. Such networks are linear transformations of the **Processor**'s output p.

$$g_i = W_i p$$

In the **Decoder** phase, the processed features $p_i^t, p_{i,j}^t$ are decoded back into discrete states, so as to predict next step hints $x_{\text{hints},t}$ and algorithm outputs x_{output}:

$$\hat{x}_{\text{hints},t}, \hat{x}_{\text{output}} = g(\{p_i^t, p_{i,j}^t | \forall i, j \in \mathcal{V}\})$$

The generated hints $\hat{x}_{\text{hints},t}$ are then aggregated with the inputs and provided as inputs for the next step prediction.

As each of the features need to be predicted, the aggregation happening in the **Encoder** does not show up in the **Decoder**. The important matter at play with the

Decoder models is the output dimension, which needs to be equal to 1 for features of type *scalar* and *mask*, equal to the number of nodes N for features of type *mask_ - one* and *pointer* and equal to the arbitrary number of classes for features of type *categorical*, this is so that the output dimension matches the target distributions for cross-entropy loss (mean-squared error for the case of *scalar*'s).

Processor Network

Arguably the most important component in Encode-Process-Decode framework [116], is the **Processor** network P. While **Encoder** f and **Decoder** g networks work as components mainly for projection between continuous and discrete states, the **Processor** bears the majority of the computational workload associated with algorithmic execution. Moreover, it serves as the primary design space for developing new architectures tailored to neural algorithmic reasoning.

Graph neural architectures are the commonly chosen model for the **Processor**. This is in part due to the structural generality of graphs and their capability of representing different kinds of data structures, thus providing the model with additional structural inductive bias when computing embeddings. Additionally, theoretical works outline the similarities between the message passing scheme and algorithmic implementation techniques [1, 32].

From the encoded features h_i^t, h_{ij}^t, h_g^t at step t, provided from **Encoder** network. The process continues by using these embeddings as inputs to the **Processor** network P, which is typically implemented as a GNN message passing layer [31].

$$p_i^t, p_{i,j}^t = P(h_i^t, h_{ij}^t, h_g^t, p_i^{t-1}, A^t) \tag{3.1}$$

where A^t is the adjacency matrix at step t and $p_i^t, p_{i,j}^t$ are the node and edge embeddings from the **Processor** network.

Figure 3.3 presents a diagram illustrating the entire process of algorithmic execution within the CLRS30 framework. To summarize the process: at each time step, inputs and predicted hints are concatenated, and each feature undergoes an **Encoder** network, typically a linear projection, which generates a vector of arbitrary dimension. The embedding produced by the encoder is the summation of the projections of each feature. Next, the **Processor** network receives the encoder output to generate features related to the graph information. The output of the **Processor** is then passed to a **Decoder** network, commonly implemented as a linear layer, which projects the **Processor** features back into the original feature space. Like the **Encoder**, the **Decoder** must also be feature-specific, as each feature belongs to a different *type*, as defined in Section 3.2.1, and requires distinct reprojection to ensure sound predictions.

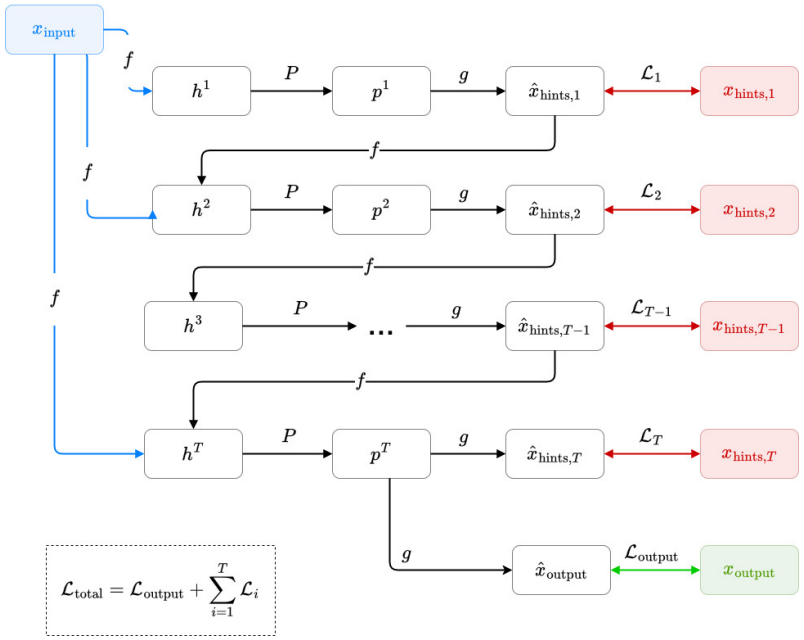


Figure 3.3: **Framework for Neural algorithmic execution within the CLRS30 benchmark**[2]. f, P and g denote the Encoder, Processor and Decoder network, respectively. $\{\mathcal{L}_i\}_{i=1}^{T}$ denote the loss function value for each step and $\mathcal{L}_{\text{out}}$ denote the output loss. x denotes the datapoint and its different features are indicated with subscripts. Information from the dataset is represented by the colored boxes in the diagram, and information generated by the network is represented by black boxes.

Inference Example

For illustration matters, we will present a worked example of an inference following the Neural Algorithmic Reasoning framework, presented in Figure 3.3. The inference example will work on the prediction of the datapoint for Breadth-first search presented in Section 3.2.3.

In this example, we will predict the next step *hints* and *outputs* for timestep $t = 2$. Two key points should be emphasized:

- A single timestep prediction is not sufficient for the entire datapoint prediction. To effectively generate outputs, the same process must be repeated over an arbitrary number of timesteps, ideally predicting all *hints*.
- We assume the prediction starts at timestep $t = 1$. At timestep $t = 0$, no *hints* are available, and the input to the **Processor** network consists solely of the projection of the *inputs*.

Encoder : For simplicity, the prediction will focus solely on node features, though the framework also supports the encoding of edge and graph features.

At timestep $t = 1$, the input for the **Encoder** network consists of the *input*

features, in the case of the BFS worked example, we have as *inputs* over nodes:

$$s = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \end{bmatrix}^T$$

The linear projection $W_s \in \mathbb{R}^{1 \times n}$, applied to the inputs, generates the encoded features $H_s \in \mathbb{R}^{5 \times n}$, where 5 is the number of nodes in the graph and n is the arbitrary embedding dimension.

Assuming the prediction of *hints* from timestep 0 to 1 was able to predict the ground-truth *hints*. The same process happens for the first rows of reach and π_h. The features reach^1 and π_h^1 undergo the same linear projection as s

$$\text{reach}^1 = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \end{bmatrix}^T$$
$$\pi_h^1 = \begin{bmatrix} 0 & 1 & 2 & 3 & 4 \end{bmatrix}^T$$

This process generates encoded features $H_{\text{reach}} \in \mathbb{R}^{5 \times n}$ and $H_{\pi_h} \in \mathbb{R}^{5 \times n}$. The output for the encoder $H \in \mathbb{R}^{5 \times n}$, at timestep $t = 1$ is then:

$$H = H_s + H_{\text{reach}} + H_{\pi_h}$$

Processor : The **Processor** network receives the calculated embeddings H, generated by the **Encoder** network, alongside the provided adjacency matrix A. The generated the processed features $p \in \mathbb{R}^{5 \times n}$ are

$$p = P(H, A)$$

For simplicity and better understanding, we are assuming the processor only receives the current embeddings generated by the **Encoder** and the adjacency matrix A. However, it can receive a wider range of inputs, such as last step embeddings, as presented in Equation 3.1.

Decoder : For prediction of outputs and next step *hints* the processed feature p go through additional processing. In this example, The *outputs* are of *type pointer*, therefore, to predict π, the linear layer $G_\pi \in \mathbb{R}^{n \times 5}$ needs to project the arbitrary embedding dimension n into the number of nodes (5):

$$g_\pi = G_\pi p$$

The generated $g_\pi \in \mathbb{R}^{5 \times 5}$, can be interpreted as a vector of logits for each node. The values of g_π can then be further processed by a softmax function, so that

the output $\hat{x}_{output}$ becomes a distribution over the nodes. The prediction selects the node with the highest probability.

The same process is applied to the feature π_s, generating the next step hints $\hat{x}_{\pi_s,2}$ at $t = 2$, which will be fed back to the network for next step prediction.

The feature reach requires special attention, as it is of *type mask* over nodes, the prediction assigns each node one of two labels. The **Decoder** linear layer for this task has dimensions $G_{\text{reach}} \in \mathbb{R}^{5\times1}$ and the applied function for generation of reach[2] is the sigmoid function.

Chapter 4

SpectralMPNN

This chapter introduces SpectralMPNN, a novel architecture designed for neural algorithmic reasoning. It combines the structural encoding capabilities of spectral graph architectures with key inductive biases from the message passing framework [1, 32], aiming to combine the strengths of both approaches.

The chapter is organized into two sections. The first motivates the use of spectral architectures for neural algorithmic reasoning, highlighting their potential advantages and identifying core aspects of message passing that we seek to integrate into spectral models. The second section presents two specific architectures, SpectralMPNN and PolySpectralMPNN, which define their spectral filters as functions of the output of message passing, therefore unifying the advantages of both approaches.

4.1 On the defense of Spectral architectures for algorithmic execution

Spatial GNNs are often the preferred choice for algorithmic execution [1, 112], this is attributed to the generality of graphs and their capacity to represent different data structures algorithms run over, and also theoretical works that present similarities between the message passing scheme and algorithm implementation techniques [1, 32].

As mentioned in Section 2.1.1, MPNNs excel at encoding local information; however, they often fail to capture global graph structure [36, 120]. The over-smoothing and over-squashing phenomenon are related to this characteristic. As over-smoothing limits the number of layers in MPNNs, it prevents the network from being capable of differentiating nodes within larger structures or encoding their structural role [121]. Still, even in sufficiently expressive GNNs less prone to over-smoothing, the compression of information over many hops places a burden on the network's capacity to attend to the graph's global properties [122].

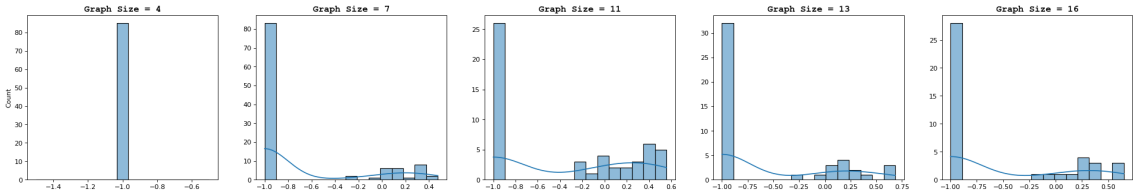


Figure 4.1: Distribution of Homophily for the Articulation Points classes in different graph sizes.

It should be intuitive, that while using a graph to represent a data structure, both the graph's topology and the information propagated through multiple hops are crucial for algorithmic execution. One of the strategies employed in [2] to enhance algorithmic performance involves disregarding the original graph structure and instead running MPNNs on a fully connected graph. We believe the improved performance observed with such an architecture in certain algorithms stems from the increased reliance of these algorithms on global information.

Moreover, one of the assumptions spatial GNNs rely on for their success, is that signals over a graph consist of low-frequency components plus high-frequency noise, and the low-frequency components carry sufficient information for the task at hand [35]. This assumption does not hold for algorithmic reasoning or the CLRS benchmark [2], as practitioners have complete control over the data-generating process, the input features are not subject to any kind of noise.

CLRS30 Homophily

The ovesmoothing and low-pass filter properties of Spatial GNNs make their performance be heavily reliant on the graph's homophily. Homophily is a fundamental property in many real-world datasets: it assumes that connected nodes often belong to the same class or present similar features [60].

The inductive homophily bias of spatial GNNs can place a burden on their performance under heterophilic scenarios (i.e. settings where connected nodes present dissimilar classes). Recent lines of research show how graph-agnostic models, such as an ordinary MLP, can outperform MPNNs in several heterophilic scenarios [61–64].

Algorithmic outputs and hints are not necessarily homophilic. For instance, Tarjan's algorithm for detecting articulation points [123], implemented in the CLRS30 dataset, identifies nodes whose removal increases the number of connected components in a graph. From a machine learning perspective, this task can be viewed as a binary classification problem, where class 0 indicates a non-articulation point and class 1 indicates an articulation point.

By calculating the unbiased homophily [3] (described in Appendix E), for graphs with articulation points with different number of nodes, when can see that the

mode of the distribution is in the minimum value attained by the measure, thus most graphs do not have edges between articulation points, and such labels can be rendered as heterophilic.

Appendix F presents the same plot of distributions for algorithms Task Scheduling, Activity Selector, Jarvis March and Graham Scan, which have outputs as *mask* type, as described in Section 3.2.1. We can see that for such algorithms the probability density function is not as concentrated in the mode of -1, however for most graph sizes the mode for unbiased homophily is still the minimum value (-1).

4.1.1 The Message Passing inductive bias

As previously mentioned, graph neural architectures are the commonly chosen model for neural algorithmic execution. This is in part due to the structural generality of graphs and their capability of representing different kinds of data structures. However, this is not the only reason as to why these architectures are the usual choice.

Recent theoretical works [1, 32] have drawn on the similarities between the message passing scheme (Equation 2.1) and the execution of algorithms based on Dynamic Programming [124]. More specifically, the work of [32] shows that the message passing in GNNs and the update rule in the Bellman-Ford algorithm share a common structure, called the integral transform.

The connection between Dynamic Programming (DP) and message passing Neural Networks (MPNNs) becomes clearer when we view DP as a process of recursively solving subproblems and aggregating their solutions to construct the final result for the original problem instance. In this perspective, each subproblem can be represented as a node in an arbitrary graph, and the DP update rule closely mirrors the message passing scheme described in Equation 2.1.

A striking example of this similarity is the Bellman-Ford algorithm, a well-known method for computing single-source shortest paths. Its update rule is given by:

$$d_{v_i} \leftarrow \min\left(d_{v_i}, \min_{v_u \in \mathcal{N}(v_i)} d_{v_u} + w_{(v_u, v_i)}\right)$$

where d_{v_i} represents the shortest known distance from the source to node v_i, and $w_{(v_u,v_i)}$ is the weight of the edge connecting node v_u to v_i.

This update rule strongly resembles the message passing framework, where information from neighboring nodes is aggregated to update each node's state. Specifically, the Bellman-Ford algorithm propagates distance information across the graph in an iterative manner, akin to how MPNNs update node embeddings based on local neighborhood features.

One major drawback of applying spectral GNNs to algorithmic reasoning is the loss of the inductive bias inherent in the message passing scheme. Our approach to addressing this challenge is to design spectral GNNs that retain the inductive bias of traditional MPNNs while still functioning as general frequency filters.

Building on this idea, we propose the *SpectralMPNN*, a spectral architecture that directly filters graph signals in the frequency domain while using the message passing update to design arbitrary filters.

4.2 Spectral MPNN

In this section we introduce the *Spectral MPNN*, an spectral GNN tailored for algorithmic reasoning, where filters are adaptively constructed from the message passing scheme. While there exist previous works who have combined spectral and spatial architectures [125], for general learning in graph data, they typically merge their outputs either by summation or stacking their layers in a network. The main distinction is that in our approach the message passing layer is directly designing the filter being applied in the spectral component.

The underlying idea of the *Spectral MPNN* is to construct spectral domain filters that are adapted to local information gathered during the message passing phase. In a more general framework, the filters being applied in the spectral domain, are functions of the embeddings generated during message passing.

$$\begin{aligned} H^t_{\text{MPNN}} &\leftarrow \text{MPNN}(H^{t-1}_{\text{node}}, H^{t-1}_{\text{edge}}, A) \\ H^t_{\text{node}} &\leftarrow U g_\theta(\Lambda, H^t_{\text{MPNN}}) U^T x \end{aligned} \tag{4.1}$$

Here, the learnable spectral filter g_θ, parameterized by θ, is conditioned on both the eigenvalues Λ of the graph Laplacian and the output H^t_{MPNN} from the message passing layer. In the next subsection, we introduce a specific architecture for g_θ. However, we argue that any design following the general formulation of Equation 4.1 should be capable of preserving the inductive bias inherent to MPNNs [1, 32].

4.2.1 Update function

The update function, generally described , takes as input node embeddings $H^t_{\text{node}} \in \mathbb{R}^{n,d}$, edge embeddings $H^t_{\text{edge}} \in \mathbb{R}^{n,n,d}$ and adjacency matrix $A \in \{0,1\}^{n,n}$, and proceeds in the following way:

$$\begin{aligned} H^t_{\text{MPNN}} &\leftarrow \text{MPNN}(H^{t-1}_{\text{node}}, H^{t-1}_{\text{edge}}, A) \\ U, \Lambda &\leftarrow \text{LaplacianEVD}(A) \end{aligned}$$

where LaplacianEVD$(-)$ of A is the eigenvalue decomposition of the Laplacian matrix of the graph defined by A, and MPNN$(-,-,-)$ is the traditional MPNN, as described in Equation 2.1, with $\bigoplus = \max$, $\Phi = \emptyset$ and $\Psi = \sum$.

The MPNN output $H^{t+1}_{\text{MPNN}} \in \mathbb{R}^{n,d}$ is then projected onto a lower dimension d_{lower} by matrix $W \in \mathbb{R}^{d,d_{\text{lower}}}$, where d_{lower} defines the number of applied filters

$$H^t_{\text{filter}} \leftarrow W \times H^t_{\text{MPNN}} \tag{4.2}$$

where $\times$ denotes matrix multiplication.

From the matrix H^{t+1}_{filter}, the process proceeds in the same manner as defined by the Specformer architecture [126]. Constructing d_{lower} different signal filters, Let $\lambda_i \in \mathbb{R}^{d_{\text{lower}}}$ be the i-th column of $H^{t+1}_{\text{filter}} = [\lambda_1||\lambda_2||...||\lambda_{d_{\text{lower}}}]$.

$$S_i \leftarrow U\text{diag}(\lambda_i)U^T \qquad\qquad \hat{S} \leftarrow \text{FFN}([I||S_1||...||S_{d_{\text{lower}}}])$$

Each of the S_i's is coined by [126] as the different learnable bases. They can be seen as a distinct filters being applied to the graph signal. The concatenation of the d_{lower} different bases lead to $\hat{S} \in \mathbb{R}^{n,n,d_{\text{lower}}+1}$, and a Feed-Forward Neural Network FFN : $\mathbb{R}^{d_{\text{lower}}} \to \mathbb{R}^d$ is used for additional processing.

Following the Graph Convolution as implemented by [126], the learned bases are of the same dimension as the original signal and each feature receives application of a distinct filter $\hat{S}_{:,:,i}$.

$$H^t_{:,i} \leftarrow \hat{S}_{:,:,i} \times H^{t-1}_{:,i}$$

where $H^t_{:,i}$ is the i-th feature of the node embeddings H^t_{node}.

By directly decomposing the graph Laplacian L, the SpectralMPNN efficiently encodes global information within the graph, whereas MPNNs are limited to encoding localized information. Furthermore, by designing the filter based on message passing information, the network preserves the fundamental inductive bias essential for Neural Algorithmic Reasoning (Section 4.1.1).

Permutation Equivariance

As as an architecture over graphs, it is extremely desirable that the Spectral MPNN possess the property of being permutation equivariant over the set of nodes [50] (Section 2.3). We see in the following theorem that such a property is preserved in the proposed architecture.

Theorem 4.1 (Permutation Equivariance of SpectralMPNN) *The update*

function applied by the SpectralMPNN update function is permutation equivariant over the set of nodes.

Step 1 It should be rather clear that the validity of this theorem relies on the MPNN layer being a permutation equivariant function, which is a property possessed by every architecture following the design space of Equation 2.1 [49].

Therefore, for the Permutation Equivariance of the SpectralMPNN to be validy, it must hold the MPNN layer, that[1]:

$$PH^t_{\text{MPNN}} = \text{MPNN}(PH^{t-1}_{\text{node}}, PAP^T)$$

where P is any permutation matrix (matrix with exactly one entry 1 in each row and each column). Throughout the remainder of this proof, we demonstrate that applying P to node embeddings H^{t-1}_{node} and adjacency matrix A is preserved by the SpectralMPNN update function.

Step 2 The construction of H^{t+1}_{filter} applies a linear transformation separately for each of the node embeddings

$$H^{t+1}_{\text{filter}} = WH^t_{\text{MPNN}}$$

Since the linear transformation is applied exclusively on the last dimension of H^t_{MPNN}, we have that

$$PH^{t+1}_{\text{filter}} = W(PH^t_{\text{MPNN}})$$

And so the generation of H^{t+1}_{filter} is permutation-equivariant.

Step 3 Since the generation of H^{t+1}_{filter} is permutation-equivariant, its column vectors $\{\lambda_i\}_{i=1}^{d_{\text{lower}}}$ must also be permutation-equivariant:

$$PH^{t+1}_{\text{filter}} = [P\lambda_1||P\lambda_2||...||P\lambda_{d_{\text{lower}}}]$$

Each matrix S_i is constructed by turning each of the λ_i vectors into its diagonal matrix $\text{diag}(\lambda_i)$. Applying P to λ_i, we obtain

$$\text{diag}(P\lambda_i) = P\text{diag}(\lambda_i)P^T$$

Additionaly, the Laplacian matrix L transforms as PLP^T, since it is computed from the permuted adjacency matrix PAP^T. The eigenvalue decomposition of the Laplacian then satisfies:

$$PUP^T, P\Sigma P^T = \text{LaplacianEVD}(PAP^T)$$

[1] We omit edge embeddings from the MPNN function, however the definition is similar.

where U is the eigenvector matrix of L.

Using this, we compute each of th S_i's

$$\begin{aligned} PUP^T\text{diag}(P\lambda_i)(PUP^T)^T &= PUP^TP\text{diag}(\lambda_i)P^T(PUP^T)^T \\ &= PU\text{diag}(\lambda_i)UP^T \\ &= PS_iP^T \end{aligned}$$

Thus, the filters S_i retain their permutation-equivariance.

Step 4 The filters $\{S_i\}_{i=1}^{d_{\text{lower}}}$ are concatenated and processed through an FFN to form $\hat{S}$:

$$\hat{S} = \text{FFN}([I||PS_1P^T||...||PS_{d_{\text{lower}}}P^T])$$

Since the FFN operates independently on each feature dimension and does not alter the first two dimensions, the 2-d matrices from indexing the last dimension of $\hat{S}$, remain permuted $P\hat{S}_{:,:,i}P^T$.

Step 5 Finally, the update function applies a distinct filter $\hat{S}_{:,:,i}$ to each feature of the node embeddings:

$$\begin{aligned} P\hat{S}_{:,:,i}P^TPH^{t-1}_{:,i} &= P\hat{S}_{:,:,i}H^{t-1}_{:,i} \\ &= PH^t_{:,i} \end{aligned}$$

Thus, the generated embeddings $PH^t_{:,i}$ are permuted in the same way as the original embeddings $PH^{t-1}_{:,i}$ and the adjacency matrix PAP^T.

However, the enhanced expressive power comes at a computational cost. Since the SpectralMPNN relies on the eigendecomposition of an $n \times n$ matrix, its running time is governed by $O(n^3)$ complexity, placing a burden in scaling to larger problem instances. To address this limitation, we introduce an alternative model, the Polynomial SpectralMPNN, described in the following section.

4.2.2 PolySpectralMPNN

Due to the computational constraints of the Laplacian matrix eigendecomposition, the Spectral MPNN does not scale efficiently.

To overcome this issue, we propose a distinct architecture, which relies on the polynomial graph filters, described in Section 2.2.2. The Polynomial SpectralMPNN (PolySpectralMPNN) is restricted exclusively to polynomial filters of the Laplacian eigenvalues and is also guided by a message passing layer.

More specifically, we implement a ChebNet [73] with the weights $\theta^l_{i,j,k}$ in Equation 2.9 calculated from the message passing layer.

$$H^t_{\text{MPNN}} \leftarrow \text{MPNN}(H^{t-1}_{\text{node}}, H^{t-1}_{\text{edge}}, A)$$
$$\theta^t_{i,j,k} \leftarrow \sum_{l=1}^{d_{t-1}} \omega_{i,l,k} H^t_{j,l}$$

where $\omega_{i,l,k}$ are the learnable parameters and $H^t_{j,l}$ is the entry at the j-th row and l-th column of H^t_{MPNN}.

The updated embeddings H^t_{node} follow the usual ChebNet [73] update equation (Equations 2.8 and 2.9).

Instead of learning the coefficients for the Chebyshev terms $\theta^t_{i,j,k}$ directly via gradient descent, we define them as a function of the message passing layer. This introduces an additional inductive bias, which is particularly beneficial for algorithmic reasoning.

While the Polynomial SpectralMPNN significantly improves scalability for larger graphs, this comes at the cost of reduced performance and expressiveness, as the filter is constrained to polynomial functions of the graph frequencies.

Chapter 5

Experiments and Results

This chapter evaluates the performance of our proposed model, SpectralMPNN (Chapter 4), and its ability to achieve fair results under different algorithms over the CLRS-30 benchmark [2].

This section aims to answer the following research questions:

1. Can spectral architectures match or outperform spatial models in algorithmic reasoning tasks?

2. How do the learned embeddings of each model decompose into the Fourier basis? Do spectral architectures preserve smoothness in their learned representations?

3. Does incorporating a message passing step in spectral architectures enhance their performance?

We believe all three research questions are of fundamental importance toward wide application of spectral architectures to algorithmic reasoning. While research question 1 aims to evaluate the performance of the proposed model when faced against state-of-the-art ones.

Research questions 2 and 3 investigate the theoretical properties of SpectralMPNN and PolySpectralMPNN architectures when applied to Neural Algorithmic Reasoning.

Analyzing the smoothness of learned representations within the graph allows us to assess the role of high-frequency components in algorithmic execution. Spatial GNNs inherently produce smooth embeddings due to their low-pass filtering nature [35]. In contrast, Spectral GNNs can retain high-frequency signals, and their performance gains suggest that this ability is crucial for capturing the intricate structures required in algorithmic tasks. Thus, improvements in spectral models may stem from their capacity to preserve these high-frequency components.

Moreover, previous research [32, 112] has demonstrated that the inductive bias of message passing networks can facilitate algorithmic reasoning. However, the necessity of this additional step in spectral models remains unexplored. To address this, we conduct an ablation study by re-implementing the proposed SpectralMPNNs (Section 4.2) without the message passing step, evaluating any potential decrease in performance.

The implemented code uses the PyTorch and PyTorch Lightning frameworks for neural network implementation and training [127]. The entire code is open source and is available in the github repository `https://github.com/ronaldalbrt/algo_reasoning`.

5.1 Experimental details

All of the experiments were performed with an NVIDIA GeForce RTX 3090, which provides 24GB of GPU memory.

5.1.1 Hyperparameters for training.

The embedding dimension used $d = 128$ across all experiments. We train with a batch size of 32, except for algorithms Floyd-Warshall, Matrix chain multiplication and Optimal binary search tree where memory constraints restricted us to using a batch size of 16, using the AdamW optimizer [128] with learning rate 10^{-3}, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-}8$ and weight decay of 10^{-2} (default input parameters defined by PyTorch's AdamW optimizer). For learning stability, we also used gradient clipping by norm [129] with the gradient clipping threshold set as 1, as well as Layer Normalization [130] at the end of every **Processor** P.

Each model was trained for a total of a 100 epochs, and the best performing model on the validation set over all epochs is the one selected.

5.1.2 Dataset generation

The dataset is generated in a online manner, with algorithms being executed on random inputs during batch construction. We follow the same dataset generation pipeline as described in [115].

Given a problem size n at random, generate an input represented as a graph with n nodes, with node, edge and graph features matching the algorithm's specification.

- For graph algorithms (Section 3.2), inputs graphs of size n are generated with connection probability p being sampled randomly from the list $[0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]$. The connection probability samples a

random variable $e_{v,u} \sim \text{Bernoulli}(p)$ for every pair of node (v, u) defining whether the edge exists. This is the widely known Erdos-Rényi model, $\text{ER}(n, p)$ [131].

- For string algorithms (Section 3.2), the problem size n is fixed at 20 and we vary the size of the pattern $1 \leq m \leq \frac{n}{2}$ to be searched. In such algorithms the first $n - m$ nodes represent the string and the remaining m nodes represent the pattern to be matched over the input string.

- For sorting, searching and divide-and-conquer algorithms, the problem size n simply defines the size of the array to be sorted.

- For geometry algorithms, n defines the number of points in convex hull finding algorithms (Graham scan and Jarvis march) and for Segments intersect n is fixed at 4 defining the start and end coordinates of both segments.

- For greedy algorithms, n defines the number of activities and tasks in Activity selection and Task Scheduling, respectively.

The algorithm is then executed in the given *inputs*, generating intermediate *hints* and *outputs*.

To induce diversity in the training data, the training input size n_{train} of a batch is randomly sampled from the list $[4, 7, 11, 13, 16]$, while the validation input size is fixed $n_{\text{validation}} = 16$ and the test input size is larger $n_{\text{test}} = 64$, to account for the OOD generalization (Section 3.2.2).

5.1.3 Methodology for model comparison

In order to compare model performance across all algorithm in the CLRS30 dataset, each model was trained 5 times, yielding 5 individual scores. For each run, we computed micro-F1 (or accuracy) [132] over the *output* features (Section 3.2.1).

We follow the methodology described in [2], deriving the same win/tie/loss employed in their experiment.

- Let $\mu(\mathcal{M})$ and $\sigma(\mathcal{M})$ denote mean and standard deviation, respectively, of $\mathcal{M}$ model's micro-F1 scores for a given algorithm.

- It is the case that model $\mathcal{M}_1$ outperforms model $\mathcal{M}_2$, if $\mu(\mathcal{M}_1) - \sigma(\mathcal{M}_1) \geq \mu(\mathcal{M}_2)$

- The model wins on a given algorithm, if it outperforms every other tested model on such algorithm.

Table 5.1: Averaged accuracy results for each of the tested models. Here results are averaged over groups of algorithms as in [2]. Number of algorithms per group are expressed next to the group's name.

Algorithm	MPNN [31]	GAT [134]	SpectralMPNN	PolySpectralMPNN
Divide & Conquer (1)	$15.00\% \pm 6.8$	$10.31\% \pm 3.9$	$23.12\% \pm 4.5$	$\mathbf{25.93}\% \pm 7.9$
Dyn. Programming (3)	$12.18\% \pm 4.2$	$\mathbf{17.88}\% \pm 6.9$	$11.61\% \pm 5.6$	$12.13\% \pm 5.9$
Geometry (3)	$86.57\% \pm 11.20$	$55.48\% \pm 12.90$	$\mathbf{91.89}\% \pm 5.8$	$96.48\% \pm 7.0$
Graphs (12)	$\mathbf{74.48}\% \pm 4.6$	$55.45\% \pm 6.8$	$70.83\% \pm 4.2$	$68.39\% \pm 6.7$
Greedy (2)	$85.34\% \pm 2.5$	$73.30\% \pm 3.9$	$\mathbf{86.89}\% \pm 5.0$	$85.48\% \pm 1.2$
Search (3)	$41.45\% \pm 4.0$	$30.83\% \pm 3.7$	$\mathbf{45.82}\% \pm 3.3$	$42.91\% \pm 4.3$
Sorting (4)	$16.75\% \pm 9.5$	$15.43\% \pm 6.1$	$\mathbf{39.87}\% \pm 14.3$	$30.40\% \pm 17.2$
Strings (2)	$28.12\% \pm 13.8$	$4.6\% \pm 3.6$	$\mathbf{36.06}\% \pm 14.2$	$25.00\% \pm 11.3$
Avg. Accuracy	44.99%	32.92%	**50.76%**	47.91%
Avg. STD	7.2%	**6%**	7.1%	7.0%
Win/Tie/Loss counts.	4/12/14	0/4/26	**4/18/8**	2/14/14

- The model loses on a given algorithm, if it is outperformed by any other tested model on such algorithm.
- Otherwise, the model is ties on a given algorithm.

As there exist 30 distinct algorithms in the CLRS30 dataset, the number of wins, losses and ties for each model must sum to 30.

Additionally, we employ the one-sided non-parametric test Mann-Whitney U [133], for testing whether one of two random variables stochastically dominates the other (For a definition of Stochastic Dominance, see Appendix G).

As baselines, we employ the traditional MPNN [31] and Graph Attention Network (GAT) [134], both models have been tested in Neural Algorithmic Reasoning in [2] and are employed against the proposed models SpectralMPNN and PolySpectralMPNN

5.2 Results

Table 5.1 presents the results grouped by algorithm types, following the CLRS30 [2] taxonomy, while Table 5.3 focus on per-algorithm results. The proposed SpectralMPNN model significantly outperformed the other three tested models. Additionally, despite its inherent expressiveness limitations, the Polynomial SpectralMPNN remained highly competitive against the baselines, exhibiting only a slight decrease in performance. The Graph Attention Network (GAT) achieved the lowest standard deviation across runs, although its overall accuracy remained consistently lower.

Table 5.2: Pairwise performance comparisons among models using the Mann-Whitney U test. For each comparison, the number of times each model outperformed the other and the number of ties are reported. Bold numbers indicate the model that performed better more frequently.

Test	MPNN	GAT	SpectralMPNN	PolySpectralMPNN	Ties
MPNN vs GAT	**16**	2	n/a	n/a	12
MPNN vs SpectralMPNN	4	n/a	**7**	n/a	19
MPNN vs PolySpectralMPNN	**5**	n/a	n/a	**5**	20
GAT vs SpectralMPNN	n/a	2	**22**	n/a	6
GAT vs PolySpectralMPNN	n/a	1	n/a	**19**	10
SpectralMPNN vs PolySpectralMPNN	n/a	n/a	**7**	3	20

The most significant improvements from applying spectral architectures were observed in Sorting and String algorithms. The performance boost in Sorting algorithms can be attributed to the nature of their output: these algorithms produce pointers that classify nodes by indicating their predecessors in the array. Predicting node permutations in this form is inherently heterophilic, as each node must correspond to a distinct node. Consequently, the ground-truth prediction for every node in the graph is unique.

Although SpectralMPNN excels in several domains, the traditional message passing approaches (MPNN and GAT) outperform both spectral models in the Graphs and Dynamic Programming categories. While all models exhibit similar performance in Dynamic Programming, the baseline MPNN significantly surpasses the proposed architectures on graph-based algorithms. We attribute this difference primarily to the Topological Sort algorithm (see Table 5.3), where the traditional MPNN consistently delivers superior results.

Table 5.2 presents the Mann-Whitney U test [133] results, showing the number of algorithms for which one model's performance was statistically superior to the other's. For instance, in the MPNN vs. SpectralMPNN comparison, SpectralMPNN was significantly better on seven algorithms, while MPNN outperformed on four, with twenty ties. Similarly, in the MPNN vs. Polynomial SpectralMPNN comparison, both models were superior on five algorithms each (with twenty ties). Finally, SpectralMPNN outperformed Polynomial SpectralMPNN on seven algorithms compared to three, with twenty ties. GAT was extensively outperformed by all other tested models regarding the statistical test.

To further investigate whether the improved performance of SpectralMPNN stems from its ability to adaptively filter signals, we analyze the spectral decomposition of the learned network embeddings. Specifically, we evaluate whether the embeddings learned by spectral architectures are less driven toward smooth features compared to the baseline.

Table 5.3: Accuracy results for each of the tested models on all 30 algorithms.

Algorithm	MPNN [31]	GAT [134]	SpectralMPNN	PolySpectralMPNN
Activity Selector	**89.58**% ± 4.6	63.58% ± 7.2	84.44% ± 6.9	87.85% ± 1.9
Articulation Points	61.39% ± 3.0	47.50% ± 15.77	80.85 ± 1.6	**90.95**% ± 6.0
Bellman Ford	**98.02**% ± 0.4	82.01% ± 1.3	96.80% ± 0.7	95.05% ± 0.8
BFS	99.37% ± 0.4	99.61% ± 0.6	**99.70**% ± 0.3	93.88% ± 9.0
Binary Search	27.50% ± 11.4	1.8% ± 2.5	25.63% ± 7.5	**30.63**% ± 11.4
Bridges	79.94% ± 16.0	35.41% ± 20.90	**83.02**% ± 7.5	58.27% ± 26.4
Bubble Sort	22.92% ± 13.0	20.06% ± 9.38	**46.46**% ± 9.0	35.14% ± 16.2
DAG Shortest Path	**98.95**% ± 0.4	93.72% ± 3.03	85.87% ± 7.1	97.71% ± 1.5
DFS	**19.50**% ± 10.2	15.79% ± 2.45	18.07% ± 6.4	18.80% ± 10.4
Dijkstra	**98.02**% ± 0.5	75.18% ± 2.59	97.61% ± 0.5	96.62% ± 0.7
Maximum Subarray	15.00% ± 6.8	10.31% ± 3.9	23.13% ± 4.6	**25.94**% ± 7.9
Floyd Warshall	11.40% ± 2.4	6.97% ± 1.8	10.83% ± 1.6	**12.84**% ± 3.1
Graham Scan	**97.51**% ± 0.7	89.38% ± 4.06	95.38% ± 2.1	93.56% ± 5.8
Heapsort	21.63% ± 13.9	13.60% ± 5.02	**70.87**% ± 23.1	22.43% ± 23.3
Insertion Sort	6.73% ± 1.5	18.30% ± 5.95	21.71% ± 8.0	**22.87**% ± 15.6
Jarvis March	64.33% ± 31.03	50.54% ± 20.87	82.49% ± 13.51	**86.54**% ± 2.4
KMP Matcher	23.13% ± 17.5	5.62% ± 4.14	**35.25**% ± 17.8	28.13% ± 15.6
LCS Length	28.51% ± 10.0	45.68% ± 18.61	26.24% ± 14.3	**28.98**% ± 16.5
Matrix Chain Order	4.50% ± 0.5	4.92% ± 0.04	**5.09**% ± 0.3	4.74% ± 0.2
Minimum	96.88% ± 2.8	89.38% ± 6.0	**98.75**% ± 1.5	96.25% ± 3.1
MST Kruskal	83.26% ± 2.6	77.69% ± 7.0	85.54% ± 1.4	**88.86**% ± 2.2
MST Prim	**90.55**% ± 1.7	52.51% ± 7.0	89.80% ± 2.1	85.76% ± 4.2
Naive String Matcher	33.13% ± 10.2	3.75% ± 3.0	**36.88**% ± 10.7	21.88% ± 7.1
Optimal BST	**3.55**% ± 2.2	3.03 ± 1.8	3.53% ± 2.5	2.69% ± 1.2
Quickselect	0.00% ± 0.0	1.25 ± 2.5	**13.09**% ± 1.1	0.00% ± 0.0
Quicksort	15.74% ± 9.8	9.78 ± 4.25	20.46% ± 17.3	**32.11**% ± 19.0
Segments Intersect	**97.86**% ± 1.9	86.64 ± 5.68	97.80% ± 2.0	**96.22**% ± 1.3
SCC	46.92% ± 11.49	26.51 ± 14.01	**56.94**% ± 6.5	32.60% ± 6.2
Task Scheduling	81.11% ± 0.4	82.97 ± 0.6	**89.35**% ± 3.1	83.11% ± 0.5
Topological Sort	**99.11**% ± 0.5	30.61 ± 1.0	47.64% ± 14.3	69.08% ± 13.8
Avg. Accuracy	54.93%	41.47%	**58.05**%	55.43%
Avg. STD	**6.3**	6.4	6.6	7.1

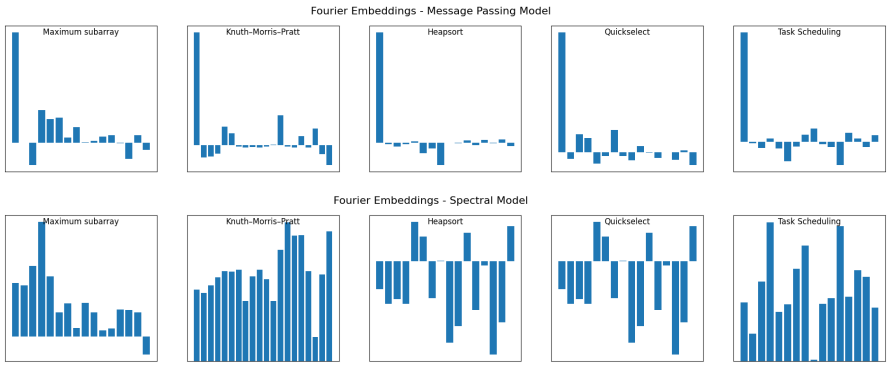


Figure 5.1: Fourier decomposition of learned representations of five algorithms

5.2.1 Analysis of the Fourier Features

To understand the representational differences between spectral and message passing architectures, we analyzed the Fourier decomposition of their learned representations. Figure 5.1 visualizes the average Fourier features, computed across batches, execution steps, and embedding dimensions, for five algorithms where spectral architectures demonstrated notable performance improvements (Table 5.3).

The Fourier features are presented as a bar graph, ordered by eigenvalue, with the left-most features associated to the lowest eigenvalue and the right-most features associated to the highest eigenvalue.

From Figure 5.1, we observe that spectral architectures exhibit a reduced tendency to be driven toward non-smooth features. In contrast, representations learned by message passing models are strongly aligned with the eigenvector of the Laplacian matrix corresponding to the zero eigenvalue, which represents the smoothest feature over the graph. However, representations learned by spectral architectures are not necessarily as smooth, often showing high similarity with eigenvectors associated with larger eigenvalues, capturing higher-frequency components.

5.2.2 Ablation Study - Message Passing Layer

In addition, we conducted ablation studies on each algorithm to analyze the impact of the message passing layer in the architecture. To implement SpectralMPNN without message passing while keeping all other components unchanged, we modify the computation of the filter matrix H^{t+1}_{filter} (Equation 4.2) as follows:

$$H^{t}_{\text{filter}} \leftarrow W \times \boldsymbol{H^{t-1}_{\text{node}}}$$

This differs from the original formulation in Section 4.2.1, where the equation used the output of the message passing layer H^{t}_{MPNN}, instead of the input node embeddings, H^{t-1}_{node}.

Table 5.4: Averaged accuracy results for each of the tested models. Here results are averaged over groups of algorithms as in [2], for per-algorithm results, see Table 5.5. Number of algorithms per group are expressed next to the group's name.

Algorithm	SpectralMPNN w/o MP	SpectralMPNN
Divide & Conquer (1)	$11.87\% \pm 5.7$	$\mathbf{23.12}\% \pm 4.5$
Dyn. Programming (3)	$\mathbf{24.21}\% \pm 12.77$	$11.61\% \pm 5.6$
Geometry (3)	$68.68\% \pm 16.58$	$\mathbf{91.89}\% \pm 5.8$
Graphs (12)	$54.11\% \pm 7.0$	$\mathbf{70.83}\% \pm 4.2$
Greedy (2)	$68.00\% \pm 3.7$	$\mathbf{86.89}\% \pm 5.0$
Search (3)	$28.54\% \pm 3.1$	$\mathbf{45.82}\% \pm 3.3$
Sorting (4)	$36.36\% \pm 14.17$	$\mathbf{39.87}\% \pm 14.3$
Strings (2)	$1.5\% \pm 1.7$	$\mathbf{36.06}\% \pm 14.2$
Avg. Accuracy	36.67%	**50.76**%
Avg. STD	8.1%	**7.1**%

Results grouped by algorithm type, following the CLRS30 taxonomy, are presented in Table 5.4, while per-algorithm results can be found in Table 5.5.

The results from the implementation without message passing are consistently outperformed by those of the traditional architecture described in Section 4.2.1. This reinforces the previous conclusion that message passing is a crucial component of algorithmic execution.

Interestingly, the only algorithm group that showed improvement after removing the message passing layer was Dynamic Programming, contradicting previous findings that highlight the similarity between MP layers and DP update rules [1, 32]. In particular, the architecture without message passing achieved notable gains on the Optimal BST and LCS Length algorithms (Table 5.5), both of which belong to the Dynamic Programming category.

Nonetheless, dynamic programming appears as a key component in various algorithms across the CLRS30 dataset, even in those not classified under the Dynamic Programming group. The consistently superior performance of SpectralMPNN with message passing further reinforces previous findings, highlighting how the layer's local embedding mechanism provides a crucial inductive bias for algorithmic reasoning.

Table 5.5: Accuracy results for the conducted Ablation study per algorithm.

Algorithm	SpectralMPNN w/o MP	SpectralMPNN
Activity Selector	54.82% ± 6.5	**84.44**% ± 7.0
Articulation Points	40.77% ± 35.4	**93.22**% ± 5.1
Bellman Ford	90.62% ± 2.5	**96.80**% ± 0.7
BFS	94.55% ± 5.3	**99.70**% ± 0.3
Binary Search	6.25% ± 0.0	**25.63**% ± 7.5
Bridges	37.17% ± 6.7	**83.02**% ± 7.5
Bubble Sort	28.54% ± 7.8	**46.46**% ± 9.0
DAG Shortest Path	78.36% ± 5.8	**85.87**% ± 7.1
DFS	11.33% ± 4.3	**18.07**% ± 6.4
Dijkstra	92.96% ± 2.4	**97.61**% ± 0.5
Find Maximum Subarray Kadane	11.87% ± 5.7	**23.13**% ± 4.6
Floyd Warshall	3.71% ± 0.2	**10.83**% ± 1.6
Graham Scan	79.08% ± 11.7	**95.38**% ± 2.1
Heapsort	16.31% ± 10.8	**70.87**% ± 23.1
Insertion Sort	**55.61**% ± 24.5	21.71% ± 8.0
Jarvis March	58.85% ± 14.0	**82.49**% ± 13.5
KMP Matcher	0.63% ± 1.3	**35.25**% ± 17.8
LCS Length	**39.25**% ± 19.8	26.24% ± 14.3
Matrix Chain Order	**5.75**% ± 0.8	5.09% ± 0.3
Minimum	79.38% ± 9.4	**98.75**% ± 1.5
MST Kruskal	**87.71**% ± 1.7	85.54% ± 1.4
MST Prim	81.48% ± 2.0	**89.80**% ± 2.1
Naive String Matcher	2.50% ± 2.3	**36.88**% ± 10.7
Optimal BST	**27.66**% ± 17.7	3.53% ± 2.5
Quickselect	0.00% ± 0.0	**13.09**% ± 1.1
Quicksort	**45.01**% ± 13.6	20.46% ± 17.3
Strongly Connected Components	24.30% ± 6.9	**56.94**% ± 6.5
Segments Intersect	68.13% ± 24.0	**97.81**% ± 2.1
Task Scheduling	81.19% ± 1.0	**89.35**% ± 3.1
Topological Sort	30.61% ± 10.3	**47.64**% ± 14.3
Avg. Accuracy	44.47%	**58.05**%
Avg. STD	8.4%	**6.6**%

Chapter 6

Conclusion

In this work, we demonstrate that the low-pass filtering nature of message passing GNNs hinders algorithmic execution and that preserving the non-smooth components of graph features is essential for effective algorithmic reasoning. Specifically, we argue that the locality bias in message passing models limits their ability to capture the full structure of the graph—information that is crucial for accurately mimicking algorithmic steps.

To address this limitation, we propose two spectral architectures tailored for algorithmic reasoning, integrating filters guided by a message passing operation. We demonstrate that these architectures remain competitive with the state-of-the-art baseline and further argue that their improved performance stems from the adaptive filtering capability of spectral models, in contrast to the inherent low-pass filtering of traditional MPNNs [34, 35].

While this study is still an initial exploration of Spectral GNNs for Neural Algorithmic Reasoning, we believe that further investigation through the lens of Graph Signal Processing holds significant promise. In particular, we recognize the need for deeper theoretical work on the alignment between algorithms and graph filtering. A substantial body of research already explores the connection between Spectral Graph Theory and Graph Algorithms [135], and we believe that establishing a solid theoretical foundation for Spectral GNNs in Neural Algorithmic Reasoning could greatly benefit from this existing framework.

Additionally, scalability becomes a challenge when applying spectral architectures to larger graphs, as real-world graphs are significantly larger than the 64-node graphs used for testing. Future research should focus on developing scalable neural network approaches that can effectively capture global graph structure while maintaining computational efficiency.

Future work could also investigate the applicability of Spectral models to language-based tasks by leveraging CLRS30-Text [136], a benchmark that translates CLRS30 datapoints into natural language descriptions suitable for processing

by state-of-the-art LLMs. In particular, [137] demonstrates that combining the outputs of a graph network trained on the original CLRS30 dataset with a large language model can enhance performance on algorithmic reasoning tasks expressed in natural language. The improved performance of Spectral MPNN (Chapter 4) is expected to better guide the LLM's outputs for algorithmic execution.

In conclusion, this dissertation underscores the advantages of efficiently encoding global structural information, rather than relying solely on local graph features, for algorithmic tasks. It also brings to light a key bottleneck in discrete reasoning with neural networks: the locality bias and low-pass filtering behavior inherent in many widely used graph models. We argue that these limitations significantly hinder their performance on tasks requiring formal reasoning.

References

[1] XU, K., LI, J., ZHANG, M., et al. "What Can Neural Networks Reason About?" 2020. Disponível em: <https://arxiv.org/abs/1905.13211>.

[2] VELIČKOVIĆ, P., BADIA, A. P., BUDDEN, D., et al. "The CLRS Algorithmic Reasoning Benchmark". 2022. Disponível em: <https://arxiv.org/abs/2205.15659>.

[3] MIRONOV, M., PROKHORENKOVA, L. "Revisiting Graph Homophily Measures". 2024. Disponível em: <https://arxiv.org/abs/2412.09663>.

[4] VELIČKOVIĆ, P., BLUNDELL, C. "Neural algorithmic reasoning", *Patterns*, v. 2, n. 7, pp. 100273, jul. 2021. ISSN: 2666-3899. doi: 10.1016/j.patter.2021.100273. Disponível em: <http://dx.doi.org/10.1016/j.patter.2021.100273>.

[5] CHOLLET, F. *Deep Learning with Python*. 1st ed. USA, Manning Publications Co., 2017. ISBN: 1617294438.

[6] KRIZHEVSKY, A., SUTSKEVER, I., HINTON, G. E. "ImageNet Classification with Deep Convolutional Neural Networks". In: Pereira, F., Burges, C., Bottou, L., et al. (Eds.), *Advances in Neural Information Processing Systems*, v. 25. Curran Associates, Inc., 2012. Disponível em: <https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf>.

[7] HE, K., ZHANG, X., REN, S., et al. "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification". In: *2015 IEEE International Conference on Computer Vision (ICCV)*, pp. 1026–1034, 2015. doi: 10.1109/ICCV.2015.123.

[8] LEE, H., PHAM, P., LARGMAN, Y., et al. "Unsupervised feature learning for audio classification using convolutional deep belief networks". In: Bengio, Y., Schuurmans, D., Lafferty, J., et al. (Eds.), *Advances in Neural Information Processing Systems*, v. 22. Curran Associates, Inc., 2009.

Disponível em: <https://proceedings.neurips.cc/paper_files/paper/2009/file/a113c1ecd3cace2237256f4c712f61b5-Paper.pdf>.

[9] VASWANI, A., SHAZEER, N., PARMAR, N., et al. "Attention Is All You Need". 2023. Disponível em: <https://arxiv.org/abs/1706.03762>.

[10] BATRA, H., PUNN, N. S., SONBHADRA, S. K., et al. "BERT-Based Sentiment Analysis: A Software Engineering Perspective". In: *Database and Expert Systems Applications*, p. 138–148, Springer International Publishing, 2021. ISBN: 9783030864729. doi: 10.1007/978-3-030-86472-9_13. Disponível em: <http://dx.doi.org/10.1007/978-3-030-86472-9_13>.

[11] OPENAI, ACHIAM, J., ADLER, S., et al. "GPT-4 Technical Report". 2024. Disponível em: <https://arxiv.org/abs/2303.08774>.

[12] TEAM, G., ANIL, R., BORGEAUD, S., et al. "Gemini: A Family of Highly Capable Multimodal Models". 2024. Disponível em: <https://arxiv.org/abs/2312.11805>.

[13] BOMMASANI, R., HUDSON, D. A., ADELI, E., et al. "On the Opportunities and Risks of Foundation Models". 2022. Disponível em: <https://arxiv.org/abs/2108.07258>.

[14] BROWN, T. B., MANN, B., RYDER, N., et al. "Language Models are Few-Shot Learners". 2020. Disponível em: <https://arxiv.org/abs/2005.14165>.

[15] PLAAT, A., WONG, A., VERBERNE, S., et al. "Reasoning with Large Language Models, a Survey". 2024. Disponível em: <https://arxiv.org/abs/2407.11511>.

[16] TEIG, N., SCHERER, R. "Bringing Formal and Informal Reasoning Together—A New Era of Assessment?" *Frontiers in Psychology*, v. 7, 07 2016. doi: 10.3389/fpsyg.2016.01097.

[17] PROUDFOOT, M. *The Routledge Dictionary of Philosophy*. New York, NY, Routledge, 2010.

[18] SAXTON, D., GREFENSTETTE, E., HILL, F., et al. "Analysing Mathematical Reasoning Abilities of Neural Models". 2019. Disponível em: <https://arxiv.org/abs/1904.01557>.

[19] CHEN, M., TWOREK, J., JUN, H., et al. "Evaluating Large Language Models Trained on Code". 2021. Disponível em: <https://arxiv.org/abs/2107.03374>.

[20] CHOLLET, F. "On the Measure of Intelligence". 2019. Disponível em: <https://arxiv.org/abs/1911.01547>.

[21] HUANG, L., YU, W., MA, W., et al. "A Survey on Hallucination in Large Language Models: Principles, Taxonomy, Challenges, and Open Questions", *ACM Transactions on Information Systems*, v. 43, n. 2, pp. 1–55, jan. 2025. ISSN: 1558-2868. doi: 10.1145/3703155. Disponível em: <http://dx.doi.org/10.1145/3703155>.

[22] KAHNEMAN, D. *Thinking, Fast and Slow*. New York: New York, Farrar, Straus and Giroux, 2011.

[23] LI, Z.-Z., ZHANG, D., ZHANG, M.-L., et al. "From System 1 to System 2: A Survey of Reasoning Large Language Models". 2025. Disponível em: <https://arxiv.org/abs/2502.17419>.

[24] CHOLLET, F. "It's Not About Scale, It's About Abstraction", The 17th Annual AGI Conference, 2024. Disponível em: <https://www.youtube.com/watch?v=s7_NlkBwdj8>.

[25] LIU, J., SHEN, Z., HE, Y., et al. "Towards Out-Of-Distribution Generalization: A Survey". 2023. Disponível em: <https://arxiv.org/abs/2108.13624>.

[26] VALIANT, L. "Out-of-Distribution Generalization as Reasoning: Are LLMs Competitive?" Emerging Generalization Settings, 2024. Disponível em: <https://www.youtube.com/watch?v=rvLUo0xiSxg>.

[27] BEURER-KELLNER, L., VECHEV, M., VANBEVER, L., et al. "Learning to Configure Computer Networks with Neural Algorithmic Reasoning". 2022. Disponível em: <https://arxiv.org/abs/2211.01980>.

[28] NUMEROSO, D., BACCIU, D., VELIČKOVIĆ, P. "Dual Algorithmic Reasoning". 2023. Disponível em: <https://arxiv.org/abs/2302.04496>.

[29] VELIČKOVIĆ, P., BOŠNJAK, M., KIPF, T., et al. "Reasoning-Modulated Representations". 2022. Disponível em: <https://arxiv.org/abs/2107.08881>.

[30] ALQIAM, A. A., YAO, Y., WANG, Z., et al. "Transferable Neural WAN TE for Changing Topologies". In: *Proceedings of the ACM SIGCOMM 2024 Conference*, ACM SIGCOMM '24, p. 86–102, New York, NY, USA, 2024. Association for Computing Machinery. ISBN: 9798400706141. doi: 10.1145/3651890.3672237. Disponível em: <https://doi.org/10.1145/3651890.3672237>.

[31] GILMER, J., SCHOENHOLZ, S. S., RILEY, P. F., et al. "Neural Message Passing for Quantum Chemistry". 2017. Disponível em: <https://arxiv.org/abs/1704.01212>.

[32] DUDZIK, A., VELIČKOVIĆ, P. "Graph Neural Networks are Dynamic Programmers". 2022. Disponível em: <https://arxiv.org/abs/2203.15544>.

[33] RUSCH, T. K., BRONSTEIN, M. M., MISHRA, S. "A Survey on Oversmoothing in Graph Neural Networks". 2023. Disponível em: <https://arxiv.org/abs/2303.10993>.

[34] LI, Q., HAN, Z., WU, X.-M. "Deeper Insights into Graph Convolutional Networks for Semi-Supervised Learning". 2018. Disponível em: <https://arxiv.org/abs/1801.07606>.

[35] NT, H., MAEHARA, T. "Revisiting Graph Neural Networks: All We Have is Low-Pass Filters". 2019. Disponível em: <https://arxiv.org/abs/1905.09550>.

[36] RAMPÁŠEK, L., GALKIN, M., DWIVEDI, V. P., et al. "Recipe for a General, Powerful, Scalable Graph Transformer". 2023. Disponível em: <https://arxiv.org/abs/2205.12454>.

[37] WANG, X., ZHANG, M. "How Powerful are Spectral Graph Neural Networks". 2022. Disponível em: <https://arxiv.org/abs/2205.11172>.

[38] MCCULLOCH, W. S., PITTS, W. "A logical calculus of the ideas immanent in nervous activity", *The bulletin of mathematical biophysics*, v. 5, n. 4, pp. 115–133, Dec 1943. ISSN: 1522-9602. doi: 10.1007/BF02478259. Disponível em: <https://doi.org/10.1007/BF02478259>.

[39] HEBB, D. O. *The Organization of Behavior: A Neuropsychological Theory*. New York, Wiley, 1949.

[40] ROSENBLATT, F. "The perceptron: a probabilistic model for information storage and organization in the brain." *Psychological review*, v. 65 6,

pp. 386–408, 1958. Disponível em: <https://api.semanticscholar.org/CorpusID:12781225>.

[41] CYBENKO, G. "Approximation by superpositions of a sigmoidal function", *Mathematics of Control, Signals and Systems*, v. 2, n. 4, pp. 303–314, Dec 1989. ISSN: 1435-568X. doi: 10.1007/BF02551274. Disponível em: <https://doi.org/10.1007/BF02551274>.

[42] RUMELHART, D. E., HINTON, G. E., WILLIAMS, R. J. "Learning representations by back-propagating errors", *Nature*, v. 323, n. 6088, pp. 533–536, Oct 1986. ISSN: 1476-4687. doi: 10.1038/323533a0. Disponível em: <https://doi.org/10.1038/323533a0>.

[43] GOYAL, A., BENGIO, Y. "Inductive biases for deep learning of higher-level cognition", *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, v. 478, n. 2266, pp. 20210068, 2022. doi: 10.1098/rspa.2021.0068. Disponível em: <https://royalsocietypublishing.org/doi/abs/10.1098/rspa.2021.0068>.

[44] LECUN, Y., BOSER, B., DENKER, J., et al. "Handwritten Digit Recognition with a Back-Propagation Network". In: Touretzky, D. (Ed.), *Advances in Neural Information Processing Systems*, v. 2. Morgan-Kaufmann, 1989. Disponível em: <https://proceedings.neurips.cc/paper_files/paper/1989/file/53c3bce66e43be4f209556518c2fcb54-Paper.pdf>.

[45] GRAVES, A., LIWICKI, M., FERNÁNDEZ, S., et al. "A Novel Connectionist System for Unconstrained Handwriting Recognition", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, v. 31, n. 5, pp. 855–868, 2009. doi: 10.1109/TPAMI.2008.137.

[46] HOCHREITER, S., SCHMIDHUBER, J. "Long Short-Term Memory", *Neural Computation*, v. 9, n. 8, pp. 1735–1780, 1997. doi: 10.1162/neco.1997.9.8.1735.

[47] MA, Y., TANG, J. *Deep Learning on Graphs*. Cambridge, United Kingdom., Cambridge University Press, 2021.

[48] CVETKOVIĆ, D., DOOB, M., SACHS, H. *Spectra of Graphs: Theory and Application*. Pure and applied mathematics : a series of monographs and textbooks. New York, Academic Press, 1980. ISBN: 9780121951504. Disponível em: <https://books.google.com.br/books?id=4u7uAAAAMAAJ>.

[49] YOU, J., YING, R., LESKOVEC, J. "Design Space for Graph Neural Networks". 2021. Disponível em: <https://arxiv.org/abs/2011.08843>.

[50] BRONSTEIN, M. M., BRUNA, J., COHEN, T., et al. "Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges". 2021. Disponível em: <https://arxiv.org/abs/2104.13478>.

[51] XU, K., HU, W., LESKOVEC, J., et al. "How Powerful are Graph Neural Networks?" 2019. Disponível em: <https://arxiv.org/abs/1810.00826>.

[52] WANG, Y., SUN, Y., LIU, Z., et al. "Dynamic Graph CNN for Learning on Point Clouds". 2019. Disponível em: <https://arxiv.org/abs/1801.07829>.

[53] KIPF, T. N., WELLING, M. "Semi-Supervised Classification with Graph Convolutional Networks". 2017. Disponível em: <https://arxiv.org/abs/1609.02907>.

[54] GIOVANNI, F. D., ROWBOTTOM, J., CHAMBERLAIN, B. P., et al. "Understanding convolution on graphs via energies". 2023. Disponível em: <https://arxiv.org/abs/2206.10991>.

[55] ZHAO, W., WANG, C., HAN, C., et al. "Comprehensive Analysis of Oversmoothing in Graph Neural Networks from Markov Chains Perspective". 2023. Disponível em: <https://arxiv.org/abs/2211.06605>.

[56] CAI, C., WANG, Y. "A Note on Over-Smoothing for Graph Neural Networks". 2020. Disponível em: <https://arxiv.org/abs/2006.13318>.

[57] RUSCH, T. K., CHAMBERLAIN, B. P., ROWBOTTOM, J., et al. "Graph-Coupled Oscillator Networks". 2022. Disponível em: <https://arxiv.org/abs/2202.02296>.

[58] LI, G., MÜLLER, M., THABET, A., et al. "DeepGCNs: Can GCNs Go as Deep as CNNs?" 2019. Disponível em: <https://arxiv.org/abs/1904.03751>.

[59] RONG, Y., HUANG, W., XU, T., et al. "DropEdge: Towards Deep Graph Convolutional Networks on Node Classification". 2020. Disponível em: <https://arxiv.org/abs/1907.10903>.

[60] MCPHERSON, M., SMITH-LOVIN, L., COOK, J. M. "Birds of a Feather: Homophily in Social Networks", *Annual Review of Sociology*, v. 27, pp. 415–444, 2001. ISSN: 03600572, 15452115. Disponível em: <http://www.jstor.org/stable/2678628>.

[61] ZHU, J., YAN, Y., ZHAO, L., et al. "Beyond Homophily in Graph Neural Networks: Current Limitations and Effective Designs". 2020. Disponível em: <https://arxiv.org/abs/2006.11468>.

[62] LUAN, S., HUA, C., LU, Q., et al. "The Heterophilic Graph Learning Handbook: Benchmarks, Models, Theoretical Analysis, Applications and Challenges". 2024. Disponível em: <https://arxiv.org/abs/2407.09618>.

[63] LUAN, S., HUA, C., LU, Q., et al. "Revisiting Heterophily For Graph Neural Networks". 2022. Disponível em: <https://arxiv.org/abs/2210.07606>.

[64] PLATONOV, O., KUZNEDELEV, D., DISKIN, M., et al. "A critical look at the evaluation of GNNs under heterophily: Are we really making progress?" 2024. Disponível em: <https://arxiv.org/abs/2302.11640>.

[65] SPIELMAN, D. A. "Spectral Graph Theory and its Applications". In: *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS'07)*, pp. 29–38, 2007. doi: 10.1109/FOCS.2007.56.

[66] GOLUB, G., VAN LOAN, C. *Matrix Computations*. Johns Hopkins Studies in the Mathematical Sciences. Baltimore, Maryland, USA, Johns Hopkins University Press, 2013. ISBN: 9781421407944.

[67] MARSDEN, A. "Eigenvalues of the Laplacian and their relationship to the connectedness", *University of Chicago, REU*, 2013. Disponível em: <https://api.semanticscholar.org/CorpusID:17239810>.

[68] SHUMAN, D. I., NARANG, S. K., FROSSARD, P., et al. "The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains", *IEEE Signal Processing Magazine*, v. 30, n. 3, pp. 83–98, maio 2013. ISSN: 1053-5888. doi: 10.1109/msp.2012.2235192. Disponível em: <http://dx.doi.org/10.1109/MSP.2012.2235192>.

[69] KENLAY, H., THANOU, D., DONG, X. "On The Stability of Polynomial Spectral Graph Filters", *2020 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 5350–5354, 2020. doi: 10.1109/ICASSP40776.2020.9054072.

[70] WU, Z., PAN, S., CHEN, F., et al. "A Comprehensive Survey on Graph Neural Networks", *IEEE Transactions on Neural Networks and Learning Systems*, v. 32, n. 1, pp. 4–24, jan. 2021. ISSN: 2162-2388. doi: 10.1109/tnnls.2020.

2978386. Disponível em: <http://dx.doi.org/10.1109/TNNLS.2020.2978386>.

[71] BO, D., WANG, X., LIU, Y., et al. "A Survey on Spectral Graph Neural Networks". 2023. Disponível em: <https://arxiv.org/abs/2302.05631>.

[72] BRUNA, J., ZAREMBA, W., SZLAM, A., et al. "Spectral Networks and Locally Connected Networks on Graphs". 2014. Disponível em: <https://arxiv.org/abs/1312.6203>.

[73] DEFFERRARD, M., BRESSON, X., VANDERGHEYNST, P. "Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering". 2017. Disponível em: <https://arxiv.org/abs/1606.09375>.

[74] SILVER, D., HUANG, A., MADDISON, C. J., et al. "Mastering the game of Go with deep neural networks and tree search", *Nature*, v. 529, n. 7587, pp. 484–489, Jan 2016. ISSN: 1476-4687. doi: 10.1038/nature16961. Disponível em: <https://doi.org/10.1038/nature16961>.

[75] HE, K., ZHANG, X., REN, S., et al. "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification". 2015. Disponível em: <https://arxiv.org/abs/1502.01852>.

[76] YE, H., XIE, C., CAI, T., et al. "Towards a Theoretical Framework of Out-of-Distribution Generalization". 2021. Disponível em: <https://arxiv.org/abs/2106.04496>.

[77] PETERS, J., BÜHLMANN, P., MEINSHAUSEN, N. "Causal inference using invariant prediction: identification and confidence intervals". 2015. Disponível em: <https://arxiv.org/abs/1501.01332>.

[78] ARJOVSKY, M., BOTTOU, L., GULRAJANI, I., et al. "Invariant Risk Minimization". 2020. Disponível em: <https://arxiv.org/abs/1907.02893>.

[79] CHANG, S., ZHANG, Y., YU, M., et al. "Invariant Rationalization". 2020. Disponível em: <https://arxiv.org/abs/2003.09772>.

[80] PETERS, J., BÜHLMANN, P., MEINSHAUSEN, N. "Causal inference using invariant prediction: identification and confidence intervals". 2015. Disponível em: <https://arxiv.org/abs/1501.01332>.

[81] PEARL, J. *Causality: models, reasoning, and inference*. USA, Cambridge University Press, 2000. ISBN: 0521773628.

[82] BÜHLMANN, P. "Invariance, Causality and Robustness". 2018. Disponível em: <https://arxiv.org/abs/1812.08233>.

[83] CORMEN, T., LEISERSON, C., RIVEST, R., et al. *Introduction to Algorithms, fourth edition.* 4th ed. , The MIT Press. ISBN: 0262033844.

[84] CHOLLET, F. "On the Measure of Intelligence". 2019. Disponível em: <https://arxiv.org/abs/1911.01547>.

[85] BOBER-IRIZAR, M., BANERJEE, S. "Neural networks for abstraction and reasoning: Towards broad generalization in machines". 2024. Disponível em: <https://arxiv.org/abs/2402.03507>.

[86] MOHRI, M., ROSTAMIZADEH, A., TALWALKAR, A. *Foundations of Machine Learning.* The MIT Press, 2012. ISBN: 026201825X.

[87] DENG, J., DONG, W., SOCHER, R., et al. "ImageNet: A large-scale hierarchical image database". In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255, 2009. doi: 10.1109/CVPR.2009.5206848.

[88] LIN, T.-Y., MAIRE, M., BELONGIE, S., et al. "Microsoft COCO: Common Objects in Context". 2015. Disponível em: <https://arxiv.org/abs/1405.0312>.

[89] BOJAR, O., BUCK, C., FEDERMANN, C., et al. "Findings of the 2014 Workshop on Statistical Machine Translation". In: Bojar, O., Buck, C., Federmann, C., et al. (Eds.), *Proceedings of the Ninth Workshop on Statistical Machine Translation*, pp. 12–58, Baltimore, Maryland, USA, jun. 2014. Association for Computational Linguistics. doi: 10.3115/v1/W14-3302. Disponível em: <https://aclanthology.org/W14-3302/>.

[90] DULAC-ARNOLD, G., LEVINE, N., MANKOWITZ, D. J., et al. "An empirical investigation of the challenges of real-world reinforcement learning". 2021. Disponível em: <https://arxiv.org/abs/2003.11881>.

[91] WILLIAMS, J. W. J. "Algorithm 232: Heapsort", *Communications of the ACM*, v. 7, n. 6, pp. 347–348, 1964.

[92] HOARE, C. A. R. "Quicksort", *The Computer Journal*, v. 5, n. 1, pp. 10–16, 01 1962. ISSN: 0010-4620. doi: 10.1093/comjnl/5.1.10. Disponível em: <https://doi.org/10.1093/comjnl/5.1.10>.

[93] HOARE, C. A. R. "Algorithm 65: find", *Commun. ACM*, v. 4, n. 7, pp. 321–322, jul. 1961. ISSN: 0001-0782. doi: 10.1145/366622.366647. Disponível em: <https://doi.org/10.1145/366622.366647>.

[94] BENTLEY, J. "Programming pearls: algorithm design techniques", *Commun. ACM*, v. 27, n. 9, pp. 865–873, set. 1984. ISSN: 0001-0782. doi: 10.1145/358234.381162. Disponível em: <https://doi.org/10.1145/358234.381162>.

[95] GAVRIL, F. "Algorithms for Minimum Coloring, Maximum Clique, Minimum Covering by Cliques, and Maximum Independent Set of a Chordal Graph", *SIAM Journal on Computing*, v. 1, n. 2, pp. 180–187, 1972. doi: 10.1137/0201013. Disponível em: <https://doi.org/10.1137/0201013>.

[96] LAWLER, E. L., LENSTRA, J. K., RINNOOY, A. H., et al. *The traveling salesman problem: a guided tour of combinatorial optimization.* Wiley Series in Discrete Mathematics & Optimization. Chichester, England, John Wiley & Sons, ago. 1985.

[97] AHO, A. V., HOPCROFT, J. E. *The Design and Analysis of Computer Algorithms*. 1st ed. USA, Addison-Wesley Longman Publishing Co., Inc., 1974. ISBN: 0201000296.

[98] WELLS, M. B. "Review: Donald E. Knuth, The Art of Computer Programming, Volume 1. Fundamental Algorithms and Volume 2. Seminumerical Algorithms", *Bulletin of the American Mathematical Society*, v. 79, n. 3, pp. 501 – 509, 1973.

[99] KRUSKAL, J. B. "On the shortest spanning subtree of a graph and the traveling salesman problem". In: *Proceedings of the American Mathematical society*, 1956. Disponível em: <https://api.semanticscholar.org/CorpusID:120068278>.

[100] PRIM, R. C. "Shortest connection networks and some generalizations", *The Bell System Technical Journal*, v. 36, n. 6, pp. 1389–1401, 1957. doi: 10.1002/j.1538-7305.1957.tb01515.x.

[101] BELLMAN, R. "On a Routing Problem", *Quarterly of Applied Mathematics*, v. 16, pp. 87–90, 1958. Disponível em: <https://api.semanticscholar.org/CorpusID:123639971>.

[102] DIJKSTRA, E. W. "A note on two problems in connexion with graphs", *Numerische Mathematik*, v. 1, n. 1, pp. 269–271, Dec 1959. ISSN: 0945-

3245. doi: 10.1007/BF01386390. Disponível em: <https://doi.org/10.1007/BF01386390>.

[103] FLOYD, R. W. "Algorithm 97: Shortest path", *Commun. ACM*, v. 5, n. 6, pp. 345, jun. 1962. ISSN: 0001-0782. doi: 10.1145/367766.368168. Disponível em: <https://doi.org/10.1145/367766.368168>.

[104] KNUTH, D. E., MORRIS, JR., J. H., PRATT, V. R. "Fast Pattern Matching in Strings", *SIAM Journal on Computing*, v. 6, n. 2, pp. 323–350, 1977. doi: 10.1137/0206024. Disponível em: <https://doi.org/10.1137/0206024>.

[105] GRAHAM, R. L. "An Efficient Algorithm for Determining the Convex Hull of a Finite Planar Set", *Inf. Process. Lett.*, v. 1, pp. 132–133, 1972. Disponível em: <https://api.semanticscholar.org/CorpusID:45778703>.

[106] JARVIS, R. "On the identification of the convex hull of a finite set of points in the plane", *Information Processing Letters*, v. 2, n. 1, pp. 18–21, 1973. ISSN: 0020-0190. doi: https://doi.org/10.1016/0020-0190(73)90020-3. Disponível em: <https://www.sciencedirect.com/science/article/pii/0020019073900203>.

[107] VELIČKOVIĆ, P., YING, R., PADOVANO, M., et al. "Neural Execution of Graph Algorithms". 2020. Disponível em: <https://arxiv.org/abs/1910.10593>.

[108] XU, K., ZHANG, M., LI, J., et al. "How Neural Networks Extrapolate: From Feedforward to Graph Neural Networks". 2021. Disponível em: <https://arxiv.org/abs/2009.11848>.

[109] BEVILACQUA, B., ZHOU, Y., RIBEIRO, B. "Size-Invariant Graph Representations for Graph Classification Extrapolations". 2021. Disponível em: <https://arxiv.org/abs/2103.05045>.

[110] GEORGIEV, D., LIÒ, P. "Neural Bipartite Matching". 2024. Disponível em: <https://arxiv.org/abs/2005.11304>.

[111] DEAC, A., BACON, P.-L., TANG, J. "Graph neural induction of value iteration". 2020. Disponível em: <https://arxiv.org/abs/2009.12604>.

[112] VELIČKOVIĆ, P., BUESING, L., OVERLAN, M. C., et al. "Pointer Graph Networks". 2020. Disponível em: <https://arxiv.org/abs/2006.06380>.

[113] F., M. E. "The shortest path through a maze", *Proc. International Symposium on the Theory of Switching*, pp. 285–292, 1959. Disponível em: <https://cir.nii.ac.jp/crid/1570854174823707776>.

[114] LAKE, B. M., ULLMAN, T. D., TENENBAUM, J. B., et al. "Building Machines That Learn and Think Like People", 2016. Disponível em: <https://arxiv.org/abs/1604.00289>.

[115] IBARZ, B., KURIN, V., PAPAMAKARIOS, G., et al. "A Generalist Neural Algorithmic Learner". 2022. Disponível em: <https://arxiv.org/abs/2209.11142>.

[116] HAMRICK, J. B., ALLEN, K. R., BAPST, V., et al. "Relational inductive bias for physical construction in humans and machines". 2018. Disponível em: <https://arxiv.org/abs/1806.01203>.

[117] DEAC, A., VELIČKOVIĆ, P., MILINKOVIĆ, O., et al. "XLVIN: eXecuted Latent Value Iteration Nets". 2020. Disponível em: <https://arxiv.org/abs/2010.13146>.

[118] NUMEROSO, D., BACCIU, D., VELIČKOVIĆ, P. "Dual Algorithmic Reasoning". 2023. Disponível em: <https://arxiv.org/abs/2302.04496>.

[119] VELIČKOVIĆ, P., BOŠNJAK, M., KIPF, T., et al. "Reasoning-Modulated Representations". In: Rieck, B., Pascanu, R. (Eds.), *Proceedings of the First Learning on Graphs Conference*, v. 198, *Proceedings of Machine Learning Research*, pp. 50:1–50:17. PMLR, 09–12 Dec 2022. Disponível em: <https://proceedings.mlr.press/v198/velickovic22a.html>.

[120] ALON, U., YAHAV, E. "On the Bottleneck of Graph Neural Networks and its Practical Implications". 2021. Disponível em: <https://arxiv.org/abs/2006.05205>.

[121] BUFFELLI, D., VANDIN, F. "The Impact of Global Structural Information in Graph Neural Networks Applications", *Data*, v. 7, n. 1, 2022. ISSN: 2306-5729. doi: 10.3390/data7010010. Disponível em: <https://www.mdpi.com/2306-5729/7/1/10>.

[122] GIOVANNI, F. D., RUSCH, T. K., BRONSTEIN, M. M., et al. "How does over-squashing affect the power of GNNs?" 2024. Disponível em: <https://arxiv.org/abs/2306.03589>.

[123] TARJAN, R. "Depth-first search and linear graph algorithms". In: *12th Annual Symposium on Switching and Automata Theory (swat 1971)*, pp. 114–121, 1971. doi: 10.1109/SWAT.1971.10.

[124] BELLMAN, R. *Dynamic Programming*. USA, Princeton University Press, 2010. ISBN: 0691146683.

[125] GEISLER, S., KOSMALA, A., HERBST, D., et al. "Spatio-Spectral Graph Neural Networks". 2024. Disponível em: <https://arxiv.org/abs/2405.19121>.

[126] BO, D., SHI, C., WANG, L., et al. "Specformer: Spectral Graph Neural Networks Meet Transformers". 2023. Disponível em: <https://arxiv.org/abs/2303.01028>.

[127] PASZKE, A., GROSS, S., CHINTALA, S., et al. "Automatic differentiation in PyTorch", 2017.

[128] LOSHCHILOV, I., HUTTER, F. "Decoupled Weight Decay Regularization". 2019. Disponível em: <https://arxiv.org/abs/1711.05101>.

[129] PASCANU, R., MIKOLOV, T., BENGIO, Y. "On the difficulty of training Recurrent Neural Networks". 2013. Disponível em: <https://arxiv.org/abs/1211.5063>.

[130] BA, J. L., KIROS, J. R., HINTON, G. E. "Layer Normalization". 2016. Disponível em: <https://arxiv.org/abs/1607.06450>.

[131] ERDÖS, P., RÉNYI, A. "On the evolution of random graphs". In: *The Structure and Dynamics of Networks*, pp. 38–82, Princeton, Princeton University Press, 2006. ISBN: 9781400841356. doi: doi:10.1515/9781400841356.38. Disponível em: <https://doi.org/10.1515/9781400841356.38>.

[132] OPITZ, J. "A Closer Look at Classification Evaluation Metrics and a Critical Reflection of Common Evaluation Practice", *Transactions of the Association for Computational Linguistics*, v. 12, pp. 820–836, 06 2024. ISSN: 2307-387X. doi: 10.1162/tacl_a_00675. Disponível em: <https://doi.org/10.1162/tacl_a_00675>.

[133] MANN, H. B., WHITNEY, D. R. "On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other", *The Annals of Mathematical Statistics*, v. 18, n. 1, pp. 50 – 60, 1947. doi: 10.1214/aoms/1177730491. Disponível em: <https://doi.org/10.1214/aoms/1177730491>.

[134] VELICKOVIC, P., CUCURULL, G., CASANOVA, A., et al. "Graph Attention Networks". In: *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. Disponível em: <https://openreview.net/forum?id=rJXMpikCZ>.

[135] MERRIS, R. "Laplacian matrices of graphs: a survey", *Linear Algebra and its Applications*, v. 197-198, pp. 143–176, 1994. ISSN: 0024-3795. doi: https://doi.org/10.1016/0024-3795(94)90486-3. Disponível em: <https://www.sciencedirect.com/science/article/pii/0024379594904863>.

[136] MARKEEVA, L., MCLEISH, S., IBARZ, B., et al. "The CLRS-Text Algorithmic Reasoning Language Benchmark". 2024. Disponível em: <https://arxiv.org/abs/2406.04229>.

[137] BOUNSI, W., IBARZ, B., DUDZIK, A., et al. "Transformers meet Neural Algorithmic Reasoners". 2024. Disponível em: <https://arxiv.org/abs/2406.09308>.

[138] PLATONOV, O., KUZNEDELEV, D., BABENKO, A., et al. "Characterizing Graph Datasets for Node Classification: Homophily-Heterophily Dichotomy and Beyond". 2024. Disponível em: <https://arxiv.org/abs/2209.06177>.

Appendix A

Graph Signal Processing

Graph Signal Processing (GSP) extends traditional signal processing techniques from Euclidean domains to non-Euclidean domains that can be represented as graphs [48]. The Fourier Transform is a technique for decomposing a signal, over a given domain, into its frequency components, and it is arguably the most fundamental tool in traditional Signal Processing. Extending signal processing to graphs naturally begins with developing an analogue of the Fourier Transform for these irregular domains. This extension is known as the Graph Fourier Transform (GFT).

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a graph, then a signal over $\mathcal{G}$ is function $x : \mathcal{V} \to \mathbb{R}$ assigning scalar values to nodes, the signal is usually represented by an N-dimensional vector, where $N = |\mathcal{V}|$ and each index corresponds to a node.

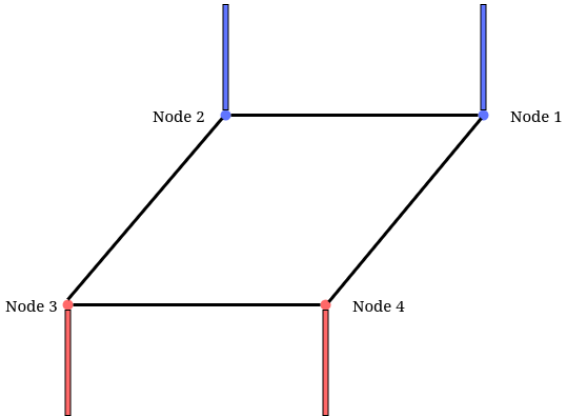


Figure A.1: Example of a signal over a graph, specifically $x = [1, 1, -1, -1]$.

One of the fundamental tools for analyzing signals in graph domains is the graph Laplacian L, defined in Section 2.2.1. As previously discussed, it has the useful property of quantifying the smoothness of a signal x, as shown in Equation 2.5:

$$x^T L x = \frac{1}{2} \sum_{v_i \in \mathcal{V}} \sum_{v_j \in \mathcal{N}(v_i)} (x_i - x_j)^2$$

This expression measures how similar the signal values are between adjacent nodes. When neighboring nodes have similar values, $x^T Lx$ is small, indicating a smooth signal over the graph. In contrast, if neighboring node values differ significantly, $x^T Lx$ will be larger, reflecting a less smooth signal.

Proof A.1 (Proof of Equation 2.5) *The proof starts from the observation, that applying the Graph Laplacian L to an arbitrary signal x is*

$$\begin{aligned} Lx &= (D - A)x \\ &= Dx - Ax \end{aligned}$$

Specifically, the i-th entry of Lx is

$$\begin{aligned} (Lx)_i &= d(v_i) \cdot x_i - \sum_{j=1}^{n} A_{i,j} \cdot x_j \\ &= \sum_{v_j \in \mathcal{N}(v_i)} x_i - x_j \end{aligned}$$

From $(Lx)_i = \sum_{v_j \in \mathcal{N}(v_i)} x_i - x_j$, we calculate the value of $x^T Lx$:

$$\begin{aligned} x^T Lx &= \sum_{v_i \in \mathcal{V}} x_i \cdot (Lx)_i \\ &= \sum_{v_i \in \mathcal{V}} x_i \cdot \sum_{v_j \in \mathcal{N}(v_i)} x_i - x_j \\ &= \sum_{v_i \in \mathcal{V}} \sum_{v_j \in \mathcal{N}(v_i)} x_i \cdot (x_i - x_j) \\ &= \sum_{v_i \in \mathcal{V}} \sum_{v_j \in \mathcal{N}(v_i)} (\frac{1}{2} x_i^2 - x_i \cdot x_j + \frac{1}{2} x_j^2) \\ &= \frac{1}{2} \sum_{v_i \in \mathcal{V}} \sum_{v_j \in \mathcal{N}(v_i)} (x_i - x_j)^2 \end{aligned}$$

The concept of signal smoothness on graphs provides the foundation for defining a graph-based analogue of the Fourier Transform, known as the Graph Fourier Transform (GFT). In this framework, the eigenvectors of the graph Laplacian serve as the Fourier basis, while the corresponding eigenvalues represent the frequency components of the graph signal. This formulation is inspired by the analogy between the graph Laplacian L and the Laplace-Beltrami operator [68], whose eigenfunctions form the Fourier basis in classical signal processing.

Let $u_1, ..., u_N$ be the set of orthonormal eigenvectors of the graph Laplacian $L = U^T \Lambda U$. By treating the ith column of U as u_i, the Graph Fourier Transform $\hat{x}$

of a signal x is defined as

$$\hat{x} = \sum_{i=1}^{N} u_1^T x = U^T x$$

Moreover, let $\lambda_1, ..., \lambda_n$ be the set of eigenvalues associated to the eigenvectors $u_1, ..., u_N$, respectively, then

$$u_i^T L u_i = \lambda_i u_i^T u_i = \lambda_i$$

the eigenvalue λ_i measures the smoothness of the associated eigenvector u_i of norm 1. Therefore, the eigenvectors associated to small eigenvalues vary slowly across the graph, in contrast, eigenvectors associated to higher eigenvalues exhibit high difference between two connected nodes.

Appendix B

Structural Equation Model

Definition B.1 (Structural Equation Model) *[81] A Structural Equation Model (SEM) governing the random vector $X = (X_1, ..., X_n)$ is a set of equations of the folowing form:*

$$S_i : X_i \leftarrow f_{X_i}(pa(X_i), \epsilon_i) \tag{B.1}$$

where $pa(X_i) \subseteq \{X_1, ..., X_n\} \backslash \{X_i\}$ are the set of parents of X_i, i.e. the causes of X_i, and ϵ_i are independent noise random variables.

Any Structural Equation Model, if well formulated, induces a directed acyclic graph, i.e. the causal graph. The graph is constructed by having a single node for each random variable $\{X_i\}_{i=1}^n$ and an edge from $X_j \rightarrow X_i$ iff $X_j \in pa(X_i)$.

It's worth noting that possession of the entire SEM governing a set of random variables $\{X_1, ..., X_n\}$, gives one complete understanding of the underlying data generating process. Moreover, by *running* the structural equations according to the topological order of the generated DAG, it's possible to generate samples from the joint distribution $P(X)$.

The usage the sign $\leftarrow$ for denoting value attribution, instead of the usual equal sign ($=$), is at the core of what kind of information SEM's are trying to convey. As equality is symmetric ($A = B \iff B = A$) it is not a suitable relation to model causality, since one should be able to convey the information that A is a cause of B, but B is not a cause of A.

Now the relation $\leftarrow$ is asymmetric, and most importantly time-dependent, in the sense that every $X_i \in pa(X_i)$ should be generated prior to the generation of Y. This is the main distinction between structural equations and algebraic equations, where the latter is defined by its sets of solutions and the former defined by the solutions of each individual equation [81].

Appendix C

Proof of Theorem 3.1

For readability purposes, Theorem 3.1 will be restated here.

Theorem C.1 (Algorithmic Alignment improves Sample Complexity [1]) *For a fixed error parameter $\epsilon > 0$ and failure probability $\delta \in [0, 1]$. Let $\{x_i, g(x_i)\}_{i=1}^{M} \sim \mathcal{D}$ be a dataset sampled from distribution $\mathcal{D}$. Suppose $\mathcal{N}_1, ..., \mathcal{N}_n$ are the network's $\mathcal{N}$ MLP modules and $\mathcal{N}$ and g (M, ϵ, δ)-algorithmically align through functions $f_1, f_2, ..., f_n$. Under the listed conditions, g is $(M, O(\epsilon), O(\delta))$-learnable by $\mathcal{N}$.*

1. ***Algorithm Stability**: Let $\mathcal{A}$ be the learning algorithm for $\mathcal{N}_i$'s. Assume $f = \mathcal{A}(\{x_i, g(x_i)\}_{i=1}^{M})$ and $\hat{f} = \mathcal{A}(\{\hat{x}_i, g(x_i)\}_{i=1}^{M})$. For any x, $\left\| f(x) - \hat{f}(x) \right\| \leq L_0 \max_i \|x_i - \hat{x}_i\|$, for some L_0.*

2. ***Sequential Learning**: The network modules are trained sequentially, that is, the first module $\mathcal{N}_1$ has input samples $\{\hat{x}_i^1, f_1(x_i^1)\}_{i=1}^{M}$. For subsequent modules, $j > 1$, the input $\hat{x}_i^j$ of module $\mathcal{N}_j$ is the output of the previous modules, and the labels are the ground-truth correct labels $(f_1 \circ f_2 \circ ... \circ f_j)(x_i^1)$.*

3. ***Lipschitzness**: The learned functions $\hat{f}_j$ satisfy $\left\| \hat{f}_j(x) - \hat{f}_j(\hat{x}) \right\| \leq L_1 \|x - \hat{x}\|$, for some L_1.*

The proof for Theorem 3.1 follows an inductive strategy, and the proof follows by showing that, under the assumptions, the error between module $\mathcal{N}_j$ and f_j is still restricted to $O(\epsilon)$ with probability $1 - O(\delta)$, after the appliedtransformations $\mathcal{N}_1, \mathcal{N}_2, ..., \mathcal{N}_{j-1}$.

Assuming a sequential learning paradigm, where each module $\mathcal{N}_j$ is learned previous to the subsequent one $\mathcal{N}_{j+1}$ and the input to the subsequent module is the output of the previous one. Treating the network as the composition of all modules $\mathcal{N} = \mathcal{N}_1 \circ \mathcal{N}_2 \circ ... \circ \mathcal{N}_n$ and the algorithmic procedure $g = f_1 \circ f_2 \circ ... \circ f_n$. The goal is to bound the learned representation $\|\mathcal{N}(x) - g(x)\|$ with high probability, by assuming learnability on the intermediary functions $f_1, f_2, ..., f_n$.

Inductive Proof

Base Case: Assuming g equals a single function f_1, and $\mathcal{N}_1$ is the module following learning procedure $\mathcal{A}$, to learn g. From the assumption that $\mathcal{N}_1$ and g (M, ϵ, δ)-algorithmically align, we know that f_1 is (M, ϵ, δ)-learnable with $\mathcal{A}$. Thus, the proof of the base case is straightforward, and

$$\mathbb{P}_{x\sim\mathcal{D}}(\|\mathcal{N}_1(x) - g(x)\| < \epsilon) < 1 - \delta$$

holds, showing that single function g's are $(M, O(\epsilon), O(\delta))$-learnable.

Inductive step : Our goal in the inductive step is to show that, if for a fixed k, the error $\|(\mathcal{N}_1 \circ ... \circ \mathcal{N}_k)(x) - (f_1 \circ ... \circ f_k)(x)\|$ is bounded by $O(\epsilon)$ with error probability $O(\delta)$, then the same holds for the subsequent $k+1$ module.

Let be $z = (f_1 \circ f_2 \circ ... \circ f_k)(x)$, the output generated by the k previous modules, and $\hat{z} = (\mathcal{N}_1 \circ \mathcal{N}_2 \circ ... \circ \mathcal{N}_k)(x)$ the output generated by the network up until module k. The goal is to show that the bound $\|\mathcal{N}_{k+1}(x) - f_{k+1}(x)\|$ holds from the theorem's assumptions and the inductive hypothesis. From the triangle inequality of the norm, we have that

$$\begin{aligned}\|\mathcal{N}_{k+1}(\hat{z}) - f_{k+1}(z)\| &= \|\mathcal{N}_{k+1}(\hat{z}) - \mathcal{N}_{k+1}(z) + \mathcal{N}_{k+1}(z) - f_{k+1}(z)\| \\ &\leq \|\mathcal{N}_{k+1}(\hat{z}) - \mathcal{N}_{k+1}(z)\| + \|\mathcal{N}_{k+1}(z) - f_{k+1}(z)\|\end{aligned}$$

The first summand in the RHS of the inequation can then be bounded by the Lipschitzness assumption of $\mathcal{N}$, thus

$$\|\mathcal{N}_{k+1}(\hat{z}) - \mathcal{N}_{k+1}(z)\| \leq L_1 \|\hat{z} - z\|$$

From the inductive step, the term $\|\hat{z} - z\| \leq O(\epsilon)$ with error probability $O(\delta)$, therefore, with error probability $O(\delta)$ it holds that

$$\|\mathcal{N}_{k+1}(\hat{z}) - \mathcal{N}_{k+1}(z)\| \leq L_1 O(\epsilon)$$

$$\|\mathcal{N}_{k+1}(\hat{z}) - \mathcal{N}_{k+1}(z)\| \leq O(\epsilon)$$

For the second summand, we rely on the theorem's assumption that f_k is (M, ϵ, δ)-learnable, by $\tilde{\mathcal{N}}_{k+1} = \mathcal{A}(\{(z_i, f_{k+1}(z_i))\}_{i=0}^{M}$ on the correct inputs. Moreover, the module $\mathcal{N}_{k+1} = \mathcal{A}(\{(\hat{z}_i, f_(k+1)(z_i))\}_{i=0}^{M}$ is generated from the perturbed samples and from the **Algorithm Stability** assumption of Theo-

rem 3.1, it is possible to bound $\left\|\mathcal{N}_{k+1}(z) - \tilde{\mathcal{N}}_{k+1}(z)\right\|$. Then,

$$\begin{aligned}\|\mathcal{N}_{k+1}(z) - f_{k+1}(z)\| &= \left\|\mathcal{N}_{k+1}(z) - \tilde{\mathcal{N}}_{k+1}(z) + \tilde{\mathcal{N}}_{k+1}(z) - f_{k+1}(z)\right\| \\ &\leq \left\|\mathcal{N}_{k+1}(z) - \tilde{\mathcal{N}}_{k+1}(z)\right\| + \left\|\tilde{\mathcal{N}}_{k+1}(z) - f_{k+1}(z)\right\|\end{aligned}$$

The first term $\left\|\mathcal{N}_{k+1}(z) - \tilde{\mathcal{N}}_{k+1}(z)\right\|$ is bounded by $L_0 \max_i \|z_i - \hat{z}_i\|$ from the **Algorithm Stability** assumption, and the second term $\left\|\tilde{\mathcal{N}}_{k+1}(z) - f_{k+1}(z)\right\|$ is bounded by ϵ with probability $1 - \delta$ from PAC-learnability of f_{k+1}. Thus, with probability $1 - \delta$, we have

$$\|\mathcal{N}_{k+1}(z) - f_{k+1}(z)\| \leq L_0 \max_i \|z_i - \hat{z}_i\| + \epsilon$$

From the induction hypothesis:

$$P_{x\sim\mathcal{D}}(\|z_i - \hat{z}_i\| \leq O(\epsilon)) \leq 1 - O(\delta)$$

where $z_i = (f_1 \circ f_2 \circ ... \circ f_k)(x_i)$ and $\hat{z}_i = (\mathcal{N}_1 \circ \mathcal{N}_2 \circ ... \circ \mathcal{N}_k)(x_i)$. The probability of the bad event $\max_i \|z_i - \hat{z}_i\| \leq O(\epsilon)$ can be bounded, with the assistance of the union bound:

$$\begin{aligned}P_{x\sim\mathcal{D}}\left(\max_i \|z_i - \hat{z}_i\| \leq O(\epsilon)\right) &= P_{x\sim\mathcal{D}}\left(\cup_{i=0}^{M}\left(\|z_i - \hat{z}_i\| \leq O(\epsilon)\right)\right) \\ &\leq \sum_{i=0}^{M} P_{x\sim\mathcal{D}}\left(\|z_i - \hat{z}_i\| \leq O(\epsilon)\right)\end{aligned}$$

Guaranteeing that $\max_i \|z_i - \hat{z}_i\| \leq O(\epsilon)$ holds with probability at most $1 - O(\delta)$.

Combining the bounds for the two summands in the original expression, the following holds with probability at most $1 - O(\delta)$:

$$\|\mathcal{N}_{k+1}(\hat{z}) - f_{k+1}(z)\| \leq O(\epsilon) + O(\epsilon) + \epsilon = O(\epsilon)$$

Completing the proof.

Appendix D

Breadth-first search pseudo-code

Algorithm 1: Breadth-first search

Data: Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ (represented as adjacency matrix A) and a source vertex $s \in \mathcal{V}$.
Result: π, a list such that $\pi[v]$ is the predecessor of v in $\mathcal{G}$
foreach $v \in \mathcal{V}$ **do**
    reach$[v] \leftarrow 0$;
    $\pi[v] \leftarrow v$
end
reach$[s] \leftarrow 1$;
do
    prev_reach $\leftarrow$ reach;
    foreach $v_1 \in \mathcal{V}$ **do**
        foreach $v_2 \in \mathcal{V} : v_2 \neq v_1$ **do**
            if $A[v_1, v_2] = 1$ & $prev\_reach[v_1] = 1$ **then**
                if $\pi[v_2] = j$ & $v_2 \neq s$ **then**
                    $\pi[v_2] \leftarrow v_1$;
                end
                reach$[v_2] \leftarrow 1$;
            end
        end
    end
while $prev\_reach[v] \neq reach[v], \forall v \in \mathcal{V}$;

Appendix E

Unbiased Homophily [3]

The work of [3] extends the theoretical framework of graph homophily by introducing a novel measure, **Unbiased Homophily**, which fulfills six desirable properties proposed by [138]. This measure is designed for graphs with categorical node labels and provides a fair quantification of homophily, free from biases introduced by the number of labels or class imbalance .

An edge is defined as *homophilic* if it connects two nodes with the same label, and *heterophilic* otherwise. The measure proposed in [3] satisfies the following six properties:

Property 1: Monotonicity. If an *homophilic* edge is added or an *heterophilic* edge is removed, the homophily measure of the overral graph must increase.

Property 2: Minimal Agreement. The homophily measure should require that graphs with only *heterophilic* edges attain a predefined lower bound value $R_{\min}$ for all other possible values of the measure.

Property 3: Maximal Agreement. The homophily measure should require that graphs with only *homophilic* edges attain a predefined upper bound value $R_{\max}$ for all other possible values of the measure.

Property 4: Empty class tolerance. The property requires that if the number of labels is increased by dummy labels, not present in the graph, the homophily measure should not change. Namely, labels not present in the graph should not contribute to the measure.

Property 5: Constant baseline. If the node's labels are completely independent of the graph's structure, then the attained homophily measure should be equal to a baseline constant value R_{base}.

Property 6: Class symmetry. Any reordering or renaming of the labels should not change the attained value for homophily.

To construct a measure that satisfies all these properties, the authors introduce the concept of a *normalized class adjacency matrix* $C_{\mathcal{G}}$, for a given graph $\mathcal{G}$. The **Unbiased Homophily** formulation is built on the definition of the *normalized class adjacency matrix*.

Definition E.1 (Normalized class adjacency matrix) *[3] For a given undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where each node $v \in \mathcal{V}$ is assigned a categorical label $y_v \in M$, with M the label set and $m = |M|$ its cardinality. The entries $\{c_{ij}\}_{i,j=1}^{m}$ of the $m \times m$ normalized class adjacency matrix $C_{\mathcal{G}}$ are*

$$c_{ii} = \frac{|\{(u,v) \in \mathcal{E} : y_u = y_v = i\}|}{|\mathcal{E}|}$$
$$c_{ij} = \frac{|\{(u,v) \in \mathcal{E} : (y_u, y_v) = (i,j)\}|}{2|\mathcal{E}|},\ i \neq j$$

Here, c_{ii} represents the proportion of homophilic edges within class i, and c_{ij} (for $i \neq j$) corresponds to the proportion of heterophilic edges between classes i and j.

From the Definition E.1, the authors of [3] were able to define the **Unbiased Homphily**, a measure which attends to all six desirable homophily properties.

Definition E.2 (Unbiased Homophily) *Given the normalized class adjacency matrix $C_{\mathcal{G}}$, the Unbiased Homophily $h(C_{\mathcal{G}})$ is defined as*

$$h(C_{\mathcal{G}}) = \frac{\sum_{j=1}^{m}\sum_{i=1}^{j}\sqrt{c_{ii}c_{jj}} - c_{ij}}{\sum_{j=1}^{m}\sum_{i=1}^{j}\sqrt{c_{ii}c_{jj}} + c_{ij}} \tag{E.1}$$

Following Definition E.2, the lower and upper bound $R_{\min}, R_{\max}$ and baseline R_{base} values for h are -1, 1 and 0, respectively.

Our interpretation starts from the observation that if the graph's structure and node labeling are independent, the value of c_{ij} is expected to be equal to $\sqrt{c_{ii}c_{jj}}$, leading to a value of 0 in the baseline case. Therefore, the numerator $\sqrt{c_{ii}c_{jj}} - c_{ij}$ should be interpreted as the difference between expected and observed *heterophilic* edges between labels i and j. The summation over this difference is positive in the case the number of expected *heterophilic* edges surpasses the number of observed ones (i.e., when homophily is present), and is negative otherwise. The denominator acts as a normalization factor, ensuring that the value of h remains within the bounded interval $[-1, 1]$.

The **Unbiased Homophily** measure satisfies all six desirable properties in *almost* all cases. An exception arises when the normalized class adjacency matrix contains at most one non-zero diagonal entry. In such cases, the **monotonicity** and **minimal agreement** properties may no longer hold. To address this edge case,

the authors of [3] propose a slight modification to the original definition (Definition E.2) to address the extreme case.

For a more thorough presentation of Unbiased Homophily, see [3].

Appendix F

Unbiased Homophily for mask outputs

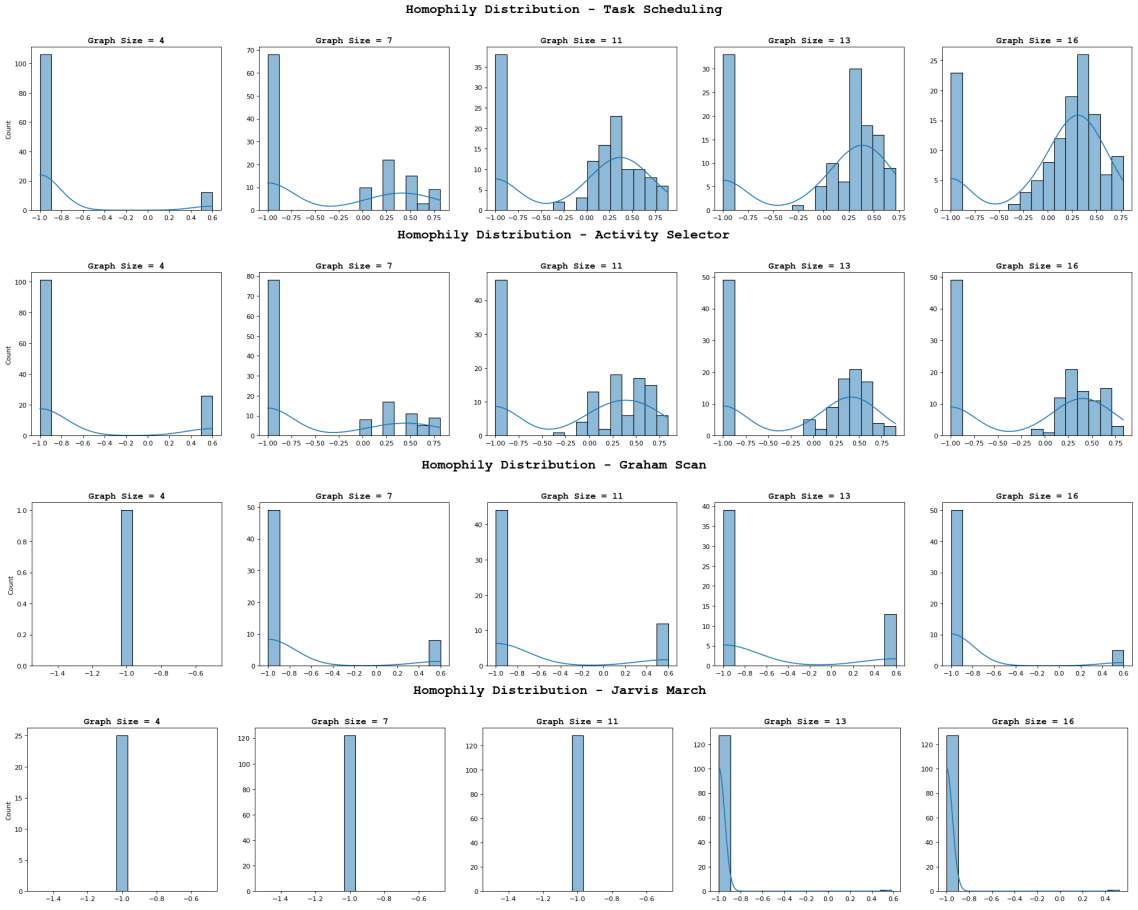


Figure F.1: Distribution of Homophily for distinct algorithms.

Appendix G

Stochastic dominance

Definition G.1 (Stochastic Dominance [133]) *Let A and B be two random variables, with cumulative distribution functions F_A and F_B, respectively, with the same support set $\mathcal{X}$. We say that A stochastically dominates B if*

$$F_A(x) > F_B(x),\ \forall x \in \mathcal{X}$$

The Mann Whitney-U test [133], employed in Section 5.1.3, is a hypothesis test on two random variables, A and B, with the following null (H_0) and altenative (H_1) hypothesis.

$$\begin{cases} H_0 : A \text{ and } B \text{ come from the same distribution.} \\ H_1 : A \text{ stochastically dominates } B \end{cases}$$