# Combinatorial Optimization with Graph Neural Networks

By

Pearl Nibebadaar Sandoh KUURIDONG (pearl.kuuridong@aims.ac.rw)
African Institute for Mathematical Sciences (AIMS), Rwanda
Supervised by: Dr. Olakunle S. ABAWONSE
Co-supervised by: Dr. Donald WOUKENG FEUDJIO
African Institute for Mathematical Sciences (AIMS), Rwanda

June 2026

*AN ESSAY PRESENTED TO AIMS RWANDA IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE AWARD OF MASTER OF SCIENCE IN MATHEMATICAL SCIENCES*

# DECLARATION

This work was carried out at AIMS Rwanda in partial fulfilment of the requirements for a Master of Science Degree.

I hereby declare that except where due acknowledgement is made, this work has never been presented wholly or in part for the award of a degree at AIMS Rwanda or any other University.

Student: Pearl Nibebadaar Sandoh KUURIDONG 

Supervisor: Olakunle S. ABAWONSE 

Co-supervisor: Donald WOUKENG FEUDJIO 

# ACKNOWLEDGEMENTS

Ad majorem Dei gloriam.

I am grateful to AIMS Rwanda and the Mastercard Foundation for the full scholarship that made this possible, and to my supervisors Dr. Olakunle S. Abawonse and Dr. Donald Woukeng Feudjio for their guidance throughout this work.

I would like to thank the researchers whose work this thesis builds upon, in particular the authors of the GCON framework, without whose foundational contributions this work would not have been possible. I am especially grateful to Semih Cantürk for his generous feedback and correspondence, which provided valuable direction at a critical stage of this work.

To the teaching staff at AIMS Rwanda, the lecturers and tutors, and to my family and friends, thank you.

# DEDICATION

To my family, teachers, and to those I have taught.

# Abstract

Combinatorial optimization (CO) problems appear throughout science and industry, yet most are NP-hard or NP-complete, making exact solutions often out of reach. In this essay, we study the Graph Combinatorial Optimization Network (GCON) proposed by Wenkel et al. (2024), an unsupervised GNN framework that equips each node with richer feature representations through a filter bank of aggregation and comparison filters. We reproduce the original results and extend the framework to Maximum Independent Set and Balanced Graph Partitioning, deriving self-supervised loss functions and rule-based decoders for each. Our experiments show that GCON outperforms ScatteringClique on Graph Partitioning and remains competitive on Maximum Independent Set, confirming the modularity and broader applicability of the framework.

# Contents

# List of Abbreviations

| | |
|---|---|
| **CNN** | Convolutional Neural Network |
| **CO** | Combinatorial Optimization |
| **DL** | Deep Learning |
| **GAT** | Graph Attention Network |
| **GCON** | Graph Combinatorial Optimization Network |
| **GNN** | Graph Neural Network |
| **ML** | Machine Learning |
| **MPNN** | Message Passing Neural Network |
| **MLP** | Multi-Layer Perceptron |
| **NP** | Nondeterministic Polynomial time |
| **RL** | Reinforcement Learning |
| **TSP** | Traveling Salesman Problem |

# 1. Introduction

## 1.1 Motivation

Optimization is fundamental to problem solving. In practice and even in theory, it is often the case that finding an exact solution is computationally infeasible. The goal then becomes finding the best possible configuration, or parameters, given the available resources and constraints (Papadimitriou and Steiglitz, 1998). Combinatorial optimization (CO) is a branch of mathematics concerned with finding optimal solutions over discrete feasible sets, drawing on tools from graph theory, theoretical computer science, and mathematical programming.

The remarkable success of machine learning (ML) across a broad range of fields, including computer vision, has naturally led to an interest in applying ML methods to combinatorial optimization (Daigavane et al., 2021). Graphs provide a particularly rich mathematical structure for representing CO problems, as many problems of interest are defined directly on graphs. The success of convolutional neural networks (CNNs) on image data, which can be viewed as a special case of graph-structured data, motivated early attempts to extend CNN-like methods to general graph-structured data (Daigavane et al., 2021). However, the irregular structure of general graphs meant that these methods could not be applied without modification. This motivated the development of Graph Neural Networks (GNNs), which generalize the convolution operation to arbitrary graph structures through a message passing framework known as Message Passing Neural Networks (MPNNs).

GNNs have demonstrated strong performance across a wide range of graph learning tasks. However, the standard MPNNs treat feature smoothness as an inductive bias; expecting adjacent nodes to have similar features. This assumption is problematic in the context of combinatorial optimization, where the graphs considered may not carry inherent node features. It is also the case that the optimal solution often requires distinguishing between adjacent nodes rather than smoothing over them, thus requiring a framework that is more expressive and representative of the underlying combinatorial structure. Additionally, most GNN methods are supervised ML methods (Karalias and Loukas, 2020), thus requiring solutions to these problems before training. In the context of CO this becomes problematic since most problems of interest are either NP-hard or NP-complete (Garey and Johnson, 2002), indicating that finding exact solutions may be computationally infeasible. Hence obtaining labels prior to training may just be impossible.

The Graph Combinatorial Optimization Network (GCON) proposed by Wenkel et al. (2024) addresses these limitations directly. In particular it improves on previous architectures, including the **ScatteringClique** proposed by Min et al. (2022), by introducing comparison filters alongside the standard aggregation filters, allowing the network to detect transition points rather than just smooth regions. In addition to producing more discriminative representations, GCON offers significant speed advantages over exact solvers like Gurobi, particularly on large graph instances. Additionally the GCON trains in a self-supervised fashion, requiring no labeling prior to training.

It turns out that solving these kinds of problems are of particular interest due to their prevalence across a wide range of real-world domains, including distributed computing, scheduling, and

vehicle routing (Cappart et al., 2022, p.2), making combinatorial optimization a field of both theoretical and practical significance.

## 1.2 Related Work

The advent of Graph Neural Networks (GNNs) can be traced back to Hopfield and Tank (1985), where they propose a neural network architecture to solve the Traveling Salesman Problem (TSP). They motivate their architecture with the nervous system of mammals in solving perceptual tasks like pattern recoginition. They argue that parallelism and interconnectedness of neurons in the mammalian nervous system contribute to their impressive speed and computing power in these perceptual tasks. They thus propose a "computational network of extensively interconnected analog neurons" to replicate these properties of the mammalian nervous system. They also propose embedding the discrete decision space of CO problems in a continuous decision space in order to easily formulate the loss functions to make minimization easier.

The success of deep learning (DL) methods on tasks like computer vision (Krizhevsky et al., 2012), natural language processing (Vaswani et al., 2017), and speech recognition (Hinton et al., 2012) led to a renewed interest in applying deep learning methods to solving CO problems. Notably, a significant research direction explored the application of supervised learning methods to solving CO problems (Li et al., 2018; Selsam et al., 2018). However this line of research comes with some limitations. Most CO problems are proven to be either NP-hard or NP-complete, meaning finding exact solutions are usually computationally infeasible, making it very expensive to obtain labels.

Reinforcement Learning (RL) methods have also been explored as a research direction. While Khalil et al. (2017) and Bai et al. (2020) propose some improvements to Q-learning for solving CO problems, Kool et al. (2018), Deudon et al. (2018), Bello et al. (2016), and Yolcu and Póczos (2019) propose some improvements to policy gradient learning for solving CO problems. The challenges with RL as a promising research direction for applying DL methods to solving CO problems, however, include the fact that RL methods are unstable during traning (Nikishin et al., 2018) which can result in poor convergence and inconsistent solution quality.

Unsupervised learning methods, the research direction that the main architecture presented in this essay falls under, have seen many successes in solving CO problems. In particular Min et al. (2022), Karalias and Loukas (2020), Sun et al. (2022), and Zhang et al. (2023) have all proposed some unsupervised learning framework for solving CO problems. Karalias and Loukas (2020) propose a framework that addresses the challenge of constructing well-behaved loss functions and decoding valid solutions, by training a GNN to minimize a probabilistic penalty loss and then using the learned parameters to decode the solutions to the CO problem. Min et al. (2022) propose a geometric scattering-based GNN for the maximum clique problem that remedies the issue of smoothness as an inductive bias in GNNs.

Collectively, these works reflect a broader shift towards unsupervised learning as a more principled and scalable approach to solving CO problems on graphs, moving away from early neural architectures and supervised methods limited by label scarcity, and RL approaches hindered by

training instability.

## 1.3 Objectives

The specific objectives of the thesis are:

- To reproduce the work done by Wenkel et al. (2024).
- To formulate loss functions for two CO problems not considered by Wenkel et al. (2024).
- To test the GCON architecture on these new CO problems.

The rest of this essay is structured as follows: chapter 2 presents the theoretical background that is required for the material that follows. Chapter 3 introduces the architecture proposed by Wenkel et al. (2024), and Chapter 4 details the implementation and experimental results and the extension of GCON to two additional combinatorial optimization problems: Maximum Independent Set and Graph Partitioning. Chapter 5 concludes with a discussion of the limitations of the proposed architecture, the applications of these CO problems, and directions for future work.

# 2. Graphs, Walks, and Neural Networks

This chapter develops the theoretical foundations underlying this work. Section 2.1 introduces the necessary concepts of graph-theory, combinatorial optimization, and Markov chains. Section 2.2 covers Graph Neural Networks, presenting the key architectures and concepts that motivate the GCON framework developed in Chapter 3.

## 2.1 Graphs, Combinatorial Optimization, Markov Chains

**2.1.1 Graph Theory Preliminaries.** Consider the following two instances: the World Wide Web and a friendship network. Both can be naturally modeled using graphs. In the former, web pages are represented as nodes and a hyperlink from one page to another as a directed edge between their corresponding nodes. In the latter, where a connection between two individuals is inherently mutual, it is more natural for edges to carry no direction. It is precisely this kind of relational information that graphs are designed to formalize, encoding entities as vertices and relationships as edges. With this motivation in mind, we now develop the graph-theoretic foundations that underpin the rest of this work.

**Definition 2.1.1.1** (Graphs)**.** A graph $G$ is made up of a vertex set $V(G)$ and an edge set $E(G) \subseteq V(G) \times V(G)$.

**Remark 2.1.1.2.** We use the words *vertex* and *node* interchangeably throughout this work to refer to the elements of $V$. When two vertices share an edge, we say that they are adjacent, and that each one is incident to the edge between them. Whenever the underlying graph $G$ is obvious, we will drop the $G$ and just write $V$ and $E$.

**Remark 2.1.1.3.** Given a graph $G = (V, E)$, if both $V$ and $E$ are finite, we say that $G$ is a finite graph.

**Definition 2.1.1.4** (Subgraph)**.** Given two graphs $G$ and $H$, we say that $H$ is a subgraph of $G$ if $V(H) \subseteq V(G)$ and $E(H) \subseteq E$.

With the basic structure of a graph in place, we now formalize the notion of edge direction, which determines how information flows between vertices.

**Definition 2.1.1.5** (Directed and Undirected Graphs)**.** Given a graph $G$, we say that an edge $e_{ij} \in E$, which is incident with vertices $v_i$ and $v_j$, is undirected if the order of its endpoints does not matter, that is, $e_{ij}$ and $e_{ji}$ refer to the same edge. If on the other hand the order does matter, so that $e_{ij}$ and $e_{ji}$ are treated as distinct, we say the edge is directed. A graph whose edges are all undirected is itself called undirected, and likewise a graph whose edges are all directed is called directed. Unless we say otherwise, the graphs in this work are undirected.

The following definitions deal with different types of graphs.

**Definition 2.1.1.6** (Simple Graph)**.** Given a graph $G = (V, E)$, an edge of the form $\{v, v\}$ for some $v \in V$ is called a loop, that is, an edge that connects a vertex to itself. We say that two edges are parallel when they have the same endpoints. A graph with no loops and no parallel edges is what we call a simple graph. Unless we say otherwise, every graph in this work is taken to be simple.

**Example 2.1.1.7.** We give an illustration of non-simple graphs in Figure 2.1.

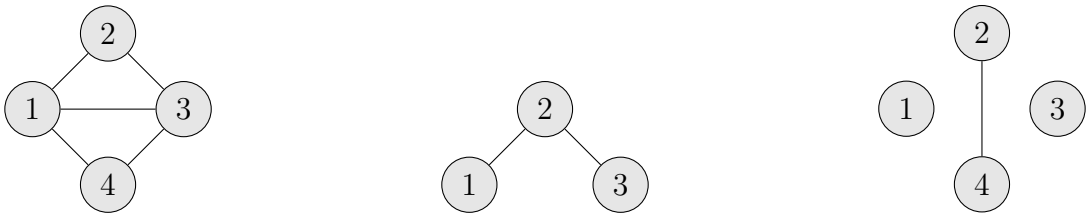

(a) A loop at 1. (b) Parallel edges between 1 and 2.

Figure 2.1: Non-simple graphs.

**Definition 2.1.1.8** (Complement Graph)**.** Given a graph $G = (V, E)$ its complement graph is defined as the graph $\overline{G} = (V, \overline{E})$ where $\overline{E} = \{\{u, v\} : u \neq v,\ \{u, v\} \notin E\}$.

**Definition 2.1.1.9** (Connected Graphs)**.** Given a graph $G$, a path in $G$ is a sequence of distinct vertices $v_0, v_1, \ldots, v_k$ such that $\{v_{i-1}, v_i\} \in E$ for all $i = 1, \ldots, k$. If we add the edge $\{v_k, v_0\} \in E$ to such a path, what we get is a cycle. A graph $G$ is connected when we can find a path between any two distinct vertices $u, v \in V$, and disconnected when we cannot. Unless we say otherwise, the graphs in this work are connected.

**Example 2.1.1.10.** Let $G$ be a graph with vertex set $V = \{1, 2, 3, 4\}$ and edge set $E = \{\{1, 2\}, \{2, 3\}, \{3, 4\}, \{1, 4\}, \{1, 3\}\}$. Figure 2.2 illustrates a simple, connected, undirected graph, $G$, a subgraph $H \subseteq G$, and the corresponding complement $\overline{G}$.




(a) Graph $G$ (b) Subgraph $H \subseteq G$ (c) Complement $\overline{G}$

Figure 2.2: Illustration of a graph, subgraph and complement graph

We now introduce structural terminology that allows us to talk about the local structure of a graph around each vertex, as well as matrix representations that encode both local and global structure algebraically.

**Definition 2.1.1.11** (Neighborhood and Degree of a Vertex)**.** Let $G$ be a graph. The neighborhood of a vertex $v \in V$ is the collection $N(v) = \{u \in V : \{u, v\} \in E\}$ of vertices joined to $v$ by an edge. The degree of $v$, written $\deg(v)$, counts how many edges are incident to $v$; equivalently, $\deg(v) = |N(v)|$.

**Definition 2.1.1.12** (Adjacency Matrix)**.** For a simple, undirected graph $G$ on $n$ vertices, the adjacency matrix is the $n \times n$ matrix $A \in \{0, 1\}^{n \times n}$ whose $(u, v)$ entry equals $1$ when $\{u, v\} \in E$ and $0$ when it does not.

**Remark 2.1.1.13.** It is obvious that for a simple undirected graph the adjacency matrix, $A$, is symmetric, that is $A = A^\top$.

**Definition 2.1.1.14** (Degree Matrix)**.** Given a graph $G$ with $n$ vertices, its degree matrix is the diagonal matrix $\mathbf{D} \in \mathbb{R}^{n \times n}$ where

$$D_{ij} = \begin{cases} \deg(i), & i = j \\ 0, & \text{otherwise.} \end{cases} \tag{2.1.1}$$

**Definition 2.1.1.15** (Graph Laplacian)**.** Let $G$ be a graph on $n$ vertices, with degree matrix $D$ and adjacency matrix $A$. The (unnormalized) graph Laplacian of $G$ is the $n \times n$ real matrix

$$L = D - A. \tag{2.1.2}$$

**Remark 2.1.1.16.** Some properties of the graph Laplacian $L$ for a simple, undirected graph $G$, include that it is symmetric, which follows from the symmetry of both the degree and adjacency matrices.

**Proposition 2.1.1.17.** For an undirected graph $G$, the graph Laplacian $L$ has the following property:

$$(\mathbf{x}^\top L \mathbf{x}) = \sum_{\{i,k\} \in E} (x_i - x_k)^2. \tag{2.1.3}$$

*Proof.* Expanding $L = D - A$ and distributing, we get

$$x^\top L x = x^\top (D - A) x = x^\top D x - x^\top A x.$$

Writing each quadratic form as a double sum over the entries of $D$ and $A$,

$$x^\top L x = \sum_{i=1}^{n} \left( \sum_{k=1}^{n} x_k D_{ki} \right) x_i - \sum_{i=1}^{n} \left( \sum_{k=1}^{n} x_k A_{ki} \right) x_i.$$

Since $D$ is diagonal with $D_{ii} = \deg(i)$, the inner sum of the first term reduces to $x_i D_{ii}$. In the second term, the inner sum reduces to a sum over the neighbors of $i$ since $A_{ki}$ is nonzero only when $k \in \mathcal{N}(i)$. Therefore,

$$x^\top L x = \sum_{i=1}^{n} x_i^2 \, \deg(i) - \sum_{i=1}^{n} \sum_{k \in \mathcal{N}(i)} x_k x_i.$$

Using $\deg(i) = |\mathcal{N}(i)|$, we get that the above expression becomes

$$x^\top Lx = \sum_{i=1}^{n} \sum_{k \in \mathcal{N}(i)} \left(x_i^2 - x_k x_i\right).$$

By symmetry, the expression above equals $\sum_{i=1}^{n} \sum_{k \in \mathcal{N}(i)} (x_k^2 - x_k x_i)$. Averaging the two forms,

$$\begin{aligned} x^\top Lx &= \tfrac{1}{2} \sum_{i=1}^{n} \sum_{k \in \mathcal{N}(i)} \left(x_i^2 - 2x_i x_k + x_k^2\right) \\ &= \tfrac{1}{2} \sum_{i=1}^{n} \sum_{k \in \mathcal{N}(i)} (x_i - x_k)^2 . \end{aligned}$$

Again by symmetry we get

$$x^\top Lx = \sum_{\{i,k\} \in E} (x_i - x_k)^2 .$$

□

**Remark 2.1.1.18.** Proposition 2.1.1.17 directly implies that for an undirected, simple, connected graph $G$, its corresponding graph Laplacian is positive semi-definite.

**Example 2.1.1.19.** Consider the graph $G$ from Example 2.2a. The adjacency matrix, degree matrix and graph Laplacian of $G$ are given by

$$A = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix}, \qquad D = \begin{pmatrix} 3 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 2 \end{pmatrix}, \qquad L = \begin{pmatrix} 3 & -1 & -1 & -1 \\ -1 & 2 & -1 & 0 \\ -1 & -1 & 3 & -1 \\ -1 & 0 & -1 & 2 \end{pmatrix} \tag{2.1.4}$$

The following illustrative example motivates the use of the Graph Laplacian as a message passing operator:

**Example 2.1.1.20.** Consider a simple, connected graph $G$, with Laplacian $L$. For simplicity, suppose that we have some feature vector $\mathrm{x} \in \mathbb{R}^n$, where $x_i$ represents the feature stored at node $i$. Then

$$\begin{aligned} (Lx)_i &= \sum_{k=1}^{n} L_{ik} x_k = \sum_{k=1}^{n} (D - A)_{ik} x_k \\ &= \sum_{k=1}^{n} D_{ik} x_k - \sum_{k=1}^{n} A_{ik} x_k = D_{ii} x_i - \sum_{k \in N(i)} x_k . \end{aligned} \tag{2.1.5}$$

To see this concretely, take the graph from Example 2.2a with $\mathrm{x} = (1, 2, 3, 4)^\top$. Then

$$L\mathrm{x} = \begin{pmatrix} 3 - 2 - 3 - 4 \\ -1 + 4 - 3 \\ -1 - 2 + 9 - 4 \\ -1 - 3 + 8 \end{pmatrix} = \begin{pmatrix} -6 \\ 0 \\ 2 \\ 4 \end{pmatrix}. \tag{2.1.6}$$

For instance, $(L\mathrm{x})_2 = 0$ since $\mathcal{N}(2) = \{1, 3\}$ and $2 \cdot 2 - (1 + 3) = 0$.

**Remark 2.1.1.21.** This shows that applying $L$ to a feature vector aggregates information from the immediate neighborhood of each node, making it a natural message passing operator. Repeated application of $L$, as in the polynomial filters $L^k x$, extends this aggregation to $k$-hop neighborhoods, allowing the network to capture increasingly global structural information with each additional power.

**2.1.2 Combinatorial Optimization.** In any university setting, it is of particular interest to schedule exams such that no student faces a conflict between two courses. Finding an optimal such schedule is a natural instance of a combinatorial optimization problem. Such problems belong to a class known as NP-hard problems, for which no efficient algorithm is known in general. In this section, we develop the theoretical framework surrounding combinatorial optimization that underlies the rest of this work.

**Definition 2.1.2.1** (Combinatorial Optimization Instance (Cappart et al., 2022, sec. 2.2))**.** An instance of a combinatorial optimization problem is a tuple $(\Omega, F, w)$, where $\Omega$ is a finite set, $F \subseteq \mathbb{P}(\Omega)$ is the set of feasible solutions, and $c : \mathbb{P}(\Omega) \to \mathbb{R}$ is a cost function with $c(S) = \sum_{\omega \in S} w(\omega)$ for $S \in F$.

**Remark 2.1.2.2.** $\mathbb{P}(\Omega)$ is the power set on a finite set $\Omega$.

**Remark 2.1.2.3.** A combinatorial optimization problem refers to the general problem class, whereas an instance specifies a concrete input. Consider the Traveling Salesman Problem (TSP), which asks for the shortest route visiting a set of cities exactly once. An instance of TSP might consist of five cities $A$,$B$,$C$,$D$,$E$ with specified distances between each pair, and the task is to find the optimal tour for that specific configuration.

Most CO problems of interest have a corresponding *decision problem*. A decision problem asks, given an instance, whether a solution satisfying some condition exists, making the solution to a decision problem either "yes" or "no". For instance, the optimization version of TSP asks for the shortest tour visiting all cities, whereas its decision version asks: does there exist a tour of length at most $k$, for some given bound $k$?

The main technique for showing that any two decision problems $L_1, L_2$ are related is to reduce one to the other. This is done by giving a transformation that maps any instance of $L_1$ to an equivalent instance of $L_2$. This concept of reducibility allows us to convert any algorithm that solves $L_2$ to a corresponding algorithm for $L_1$.

**Definition 2.1.2.4** (Polynomial-Time Reducibility)**.** A decision problem $L_1$ is polynomial-time reducible to a decision problem $L_2$, written $L_1 \leq_P L_2$, if there exists an algorithm, $f : L_1 \to L_2$ which is polynomial-time in the worst case, such that

$$x \in L_1 \iff f(x) \in L_1. \tag{2.1.7}$$

**Remark 2.1.2.5** (Properties of Polynomial-Time Reducibility)**.** We give some basic properties of the relation $\leq_P$.

1. $\leq_P$ is not symmetric. This follows directly from the fact that a polynomial-time algorithm $f : L_1 \to L_2$ does not necessarily yield a corresponding algorithm $f^{-1} : L_2 \to L_1$ which is also polynomial-time.

2. $\leq_P$ is reflexive. This follows directly from the fact that the identity map $\mathrm{id} : L_1 \to L_1$ is a polynomial-time algorithm satisfying $x \in L_1 \iff \mathrm{id}(x) \in L_1$.

3. $\leq_P$ is transitive. This follows directly from the fact that the composition of two polynomial-time algorithms $f : L_1 \to L_2$ and $g : L_2 \to L_3$ yields a polynomial-time algorithm $g \circ f : L_1 \to L_3$ satisfying $x \in L_1 \iff (g \circ f)(x) \in L_3$.

Taken together, it then means that $\leq_P$ is not an equivalence relation.

**Remark 2.1.2.6** (Nondeterministic Algorithm)**.** A nondeterministic algorithm is, loosely speaking, an idealized model of computation in which the algorithm is allowed to "guess" a solution for the input and then check, deterministically, whether this solution is correct. We say the algorithm accepts the input whenever at least one such guess passes the verification step.

This is in contrast with a deterministic algorithm, where the input alone pins down the entire execution, leaving no room for guessing along the way.

Having developed the necessary preliminaries, we now turn to the complexity-theoretic setting in which the problems studied in this work are situated. We introduce the class NP and the related notions of NP-hardness and NP-completeness, which together provide the vocabulary used throughout the remainder of this essay.

**Definition 2.1.2.7** (NP)**.** A decision problem $L$ belongs to the class $\mathbf{NP}$ if and only if it is accepted by a nondeterministic algorithm operating in polynomial time.

**Definition 2.1.2.8** (NP-Complete)**.** A decision problem $L$ is NP-complete if it belongs to the class $\mathbf{NP}$ and every other problem in $\mathbf{NP}$ reduces to it.

**Definition 2.1.2.9** (NP-Hard)**.** A decision problem $L$ is NP-hard if every problem in $\mathbf{NP}$ reduces to it. Note that $L$ itself is not required to be in $\mathbf{NP}$.

**Example 2.1.2.10** (NP-completeness of Maximum Clique (Karp, 2009))**.** The Maximum Clique problem provides a natural illustration of the distinction between the two notions. Given a graph $G$, one may ask about its clique structure in two complementary ways. The decision version asks whether $G$ contains a clique of size at least $k$ for some given integer $k$, and is NP-complete: given a candidate subset of vertices, one can verify in polynomial time that the required edges are present, placing the problem in $\mathbf{NP}$, yet no efficient algorithm is known for determining whether such a clique exists in general. The optimization version, by contrast, asks for the size of the largest clique in $G$, and is NP-hard: beyond exhibiting a clique of a given size, one must also rule out the existence of any larger one, a guarantee that does not appear to be checkable in polynomial time. The same combinatorial object thus gives rise to two problems of subtly different character, and Maximum Clique is among the problems we consider later in this essay.

**2.1.3 Markov Chains.** We now turn to Markov chains, a mathematical framework for modeling processes that evolve in a memoryless fashion: where the system goes next depends only on where it currently is, not on how it got there. The propagation of information across a graph in a GNN fits naturally into this picture. At each step, a node's features are determined by those of its

neighbors, with no reference to earlier states. Our goal in this section is to develop the theory of discrete-time Markov chains, working our way up to the Ergodic Theorem. We will lean on this result in Chapter 3 to justify the convergence of the shift operator underlying the GCON filter bank.

**Definition 2.1.3.1** (State and State Space)**.** Let $I$ be a countable set. We call every element $i \in I$ a state and $I$ a state-space.

**Definition 2.1.3.2** (Distribution)**.** Given a probability space $(\Omega, \mathcal{F}, \mathbb{P})$ and a state-space $I$, we call a row vector $\lambda = (\lambda_i : i \in I)$ a distribution of a random variable $X$, if

$$\lambda_i = \mathbb{P}(X = i). \tag{2.1.8}$$

We now define terminology that allows us to talk about how these processes evolve over time.

**Definition 2.1.3.3** (Transition Probability and Transition Matrix)**.** Given a sequence of random variables $(X_n)_{n\geq 0}$ and a state space $I$, the transition probability from state $i$ to state $j$ is defined as

$$P_{ij} = \mathbb{P}(X_{n+1} = j \mid X_n = i). \tag{2.1.9}$$

The matrix $P = (P_{ij})_{i,j\in I}$ is called the transition matrix of the Markov chain. Under this convention, $P$ is a row stochastic matrix, meaning each row sums to one:

$$\sum_{j\in I} P_{ij} = 1 \quad \forall i \in I. \tag{2.1.10}$$

**Remark 2.1.3.4.** The transition matrix defined above only encodes 1-step transitions, where the chain moves from state $i$ at time $k$ to state $j$ at time $k+1$. However, it is natural to ask about the probability of transitioning from state $i$ to state $j$ in exactly $n$ steps. The $n$-step transition probabilities $P_{ij}^{(n)}$ capture this, and together form the $n$-step transition matrix $P^{(n)} = (P_{ij}^{(n)})_{i,j\in I}$.

**Remark 2.1.3.5.** An alternative convention defines the transition probability column-wise in which case $P_{ij}$ represents the probability of transitioning from state $j$ to state $i$. This convention yields a column stochastic matrix where each column sums to one. The two conventions are transposes of each other.

With all these definitions in place, we now formally define the main framework for discussing evolving states over discrete time: the Discrete-Time Markov Chain (DTMC).

**Definition 2.1.3.6** (Discrete-Time Markov Chains)**.** Given a probability space $(\Omega, \mathcal{F}, \mathbb{P})$ and a state-space $I$, a sequence of random variables $(X_n)_{n\geq 0}$ is said to be a discrete-time Markov chain with initial distribution $\lambda \in \mathbb{R}^{|I|}$ and transition matrix $P$ if

1. $X_0$ has distribution $\lambda$, and
2. $\mathbb{P}(X_{n+1} = i_{n+1} \mid X_0 = i_0, \ldots, X_n = i_n) = \mathbb{P}(X_{n+1} = i_{n+1} \mid X_n = i_n) = P_{i_n, i_{n+1}}$.

**Remark 2.1.3.7.** Property 2 above is known as the Markov Property.

In particular the Chapman-Kolmogorov equation (Theorem 2.1.3.8) allows us to establish a relationship between the $n$-step transition matrix of some Markov chain and powers of the 1-step transition matrix. We state, without proof, the Chapman-Kolmogorov equation below:

**Theorem 2.1.3.8** (Chapman-Kolmogorov Equation)**.** *Let $(X_n)_{n\geq 0}$ be a discrete-time Markov chain with transition matrix $P$. Then for all $n, m \geq 0$ and all states $i, j \in I$,*

$$P_{ij}^{(n+m)} = \sum_{k\in I} P_{ik}^{(n)} P_{kj}^{(m)}. \tag{2.1.11}$$

**Remark 2.1.3.9.** It follows from Theorem 2.1.3.8 that the $(n+m)$-step transition matrix satisfies

$$P^{(n+m)} = P^{(n)} \cdot P^{(m)}. \tag{2.1.12}$$

To see this, observe that the $ij$-th entry of $P^{(n+m)}$ is precisely the $(n+m)$-step transition probability $P_{ij}^{(n+m)}$. By the Chapman-Kolmogorov equation,

$$P_{ij}^{(n+m)} = \sum_{k\in I} P_{ik}^{(n)} P_{kj}^{(m)}, \tag{2.1.13}$$

which is exactly the $ij$-th entry of the matrix product $P^{(n)} \cdot P^{(m)}$. Since this holds for all $i, j \in I$, the result follows.

**Proposition 2.1.3.10.** Given a Markov chain $(X_n)_{n\geq 1}$ with transition matrix $P$, the $n$-step transition matrix satisfies $P^{(n)} = P^n$ for all $n \geq 1$.

*Proof.* We proceed by induction on $n$.

For the base case of $n = 1$, we notice that this holds trivially since $P^{(1)} = P = P^1$ by definition.

Suppose now that for some $n \in \mathbb{N}$, $P^{(n)} = P^n$. Then by Theorem 2.1.3.8,

$$\begin{aligned} P^{(n+1)} &= P^{(n)} \cdot P^{(1)} \\ &= P^n \cdot P \\ &= P^{n+1}, \end{aligned} \tag{2.1.14}$$

Hence, by the principle of mathematical induction, $P^{(n)} = P^n$ for all $n \geq 1$. □

**Example 2.1.3.11.** Consider a particle moving randomly on the graph $G$ from Example 2.2a, illustrated in Figure 2.3.

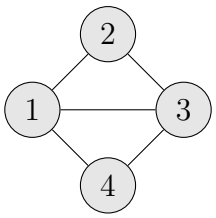



Figure 2.3: The graph $G$ from Example 2.2a.

At each step, the particle moves to a neighboring node chosen uniformly at random, a process known as a *random walk*. So if the particle is at node $1$, which has $\deg(1) = 3$, it moves to each of nodes $2$, $3$, $4$ with probability $\frac{1}{3}$. If it is at node $2$, which has $\deg(2) = 2$, it moves to each of nodes $1$, $3$ with probability $\frac{1}{2}$. The transition matrix of this random walk is

$$P = \begin{pmatrix} 0 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ \frac{1}{3} & \frac{1}{3} & 0 & \frac{1}{3} \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 \end{pmatrix}. \tag{2.1.15}$$

Each row sums to one, confirming that $P$ is row stochastic. By the proposition above, the $2$-step transition matrix is $P^{(2)} = P^2$, so for instance $P^{(2)}_{14} = P^2_{14} = \frac{1}{9}$ gives the probability of the particle moving from node $1$ to node $4$ in exactly two steps.

A natural question arising from the study of Markov chains concerns their long-run behavior. In particular, one may ask whether the distribution of $X_n$ stabilizes as $n \to \infty$. This motivates the following two definitions.

**Definition 2.1.3.12** (Limiting Distribution)**.** Given a Markov chain $(X_n)_{n\geq 0}$ with transition matrix $P$. We say that a distribution, $\pi$, is a limiting distribution of $(X_n)_{n\geq 0}$ if for any initial distribution $\alpha$

$$\lim_{n\to\infty} \alpha P^n = \pi. \tag{2.1.16}$$

**Definition 2.1.3.13** (Stationary Distribution)**.** Given a Markov chain $(X_n)_{n\geq 0}$ with transition matrix $P$. We say that a distribution, $\lambda$, is a stationary distribution of $(X_n)_{n\geq 0}$ if

$$\lambda P = \lambda. \tag{2.1.17}$$

Together, these two concepts allow us to answer a practically important question about a Markov chain: where does it end up in the long run? A stationary distribution describes what the chain looks like once it has "settled down", while a limiting distribution describes the distribution the chain converges to over time, regardless of where it started. The following proposition and example let us see the relationship between these two concepts:

**Proposition 2.1.3.14.** For a given Markov chain $(X_n)_{n\geq 0}$ with a limiting distribution $\pi$, $\pi$ is also a stationary distribution.

*Proof.* Suppose that $(X_n)_{n\geq 0}$ is a Markov chain with transition matrix $P$ and limiting distribution $\pi$. Then for any initial distribution $\alpha$,

$$\lim_{n\to\infty} \alpha P^n = \pi.$$

Then,

$$\pi P = \left(\lim_{n\to\infty} \alpha P^n\right) P$$

$$= \lim_{n\to\infty} \alpha P^{n+1}$$
$$= \pi.$$

□

The converse of the statement in Proposition 2.1.3.14 is not necessarily true and is illustrated by the following example.

**Example 2.1.3.15.** Consider a random walk on the path graph $G$ with vertex set $V = \{1, 2, 3\}$ and edge set $E = \{\{1, 2\}, \{2, 3\}\}$, illustrated in the figure below.

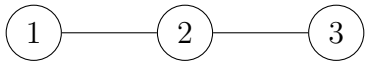



Figure 2.4: A path graph on three vertices.

The transition matrix of this random walk is

$$P = \begin{pmatrix} 0 & 1 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} \\ 0 & 1 & 0 \end{pmatrix}.$$

One can verify that $\pi = \left(\frac{1}{4}, \frac{1}{2}, \frac{1}{4}\right)$ is a stationary distribution, satisfying $\pi P = \pi$. However, $\pi$ is not a limiting distribution. To see this, observe that

$$P^2 = \begin{pmatrix} \frac{1}{2} & 0 & \frac{1}{2} \\ 0 & 1 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} \end{pmatrix}, \quad P^3 = \begin{pmatrix} 0 & 1 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} \\ 0 & 1 & 0 \end{pmatrix} = P.$$

Since $P^3 = P$, the powers of $P$ oscillate indefinitely between $P$ and $P^2$ and never converge, confirming that no limiting distribution exists.

The existence and uniqueness of a stationary distribution depends on structural properties of the chain. We now introduce two such properties.

**Definition 2.1.3.16** (Irreducible Markov Chain)**.** A Markov chain $(X_n)_{n\geq 0}$ with transition matrix $P$ and state space $I$ is said to be irreducible if for every pair of states $i, j \in I$ there exists $n \geq 1$ such that $P_{ij}^{(n)} > 0$.

**Definition 2.1.3.17** (Aperiodic Markov Chain)**.** The period $d$ of a state $i \in I$ is defined as

$$d(i) := \gcd\{n \in \mathbb{N} : P_{ii}^{(n)} > 0\}. \tag{2.1.18}$$

A Markov chain $(X_n)_{n\geq 0}$ is called aperiodic if every state has period $1$.

**Remark 2.1.3.18.** A finite-state Markov chain $(X_n)_{n\geq 0}$ is called ergodic if it is irreducible and aperiodic.

We now state without proof the following theorem which is important for the justification in 3.

**Theorem 2.1.3.19** (Fundamental Limit Theorem for Ergodic Markov Chains)**.** *Let $(X_n)_{n\geq 0}$ be a finite-state ergodic Markov chain with transition matrix $P$. Then there exists a unique stationary distribution, $\pi$, of the chain which is also a limiting distribution of the chain.*

We summarize all these concepts in the following example:

**Example 2.1.3.20.** Consider the random walk on $G$ from Example 2.1.3.11 with transition matrix $P$. Since $G$ is connected, then the chain is irreducible. It is easy to verify that each state has period $1$, and hence the chain is aperiodic.

Being irreducible and aperiodic on a finite state space, the chain is ergodic. By Theorem 2.1.3.19, there exists a unique stationary distribution $\pi$ satisfying $\pi P = \pi$. One can verify that

$$\pi = \left(\frac{3}{10}, \frac{2}{10}, \frac{3}{10}, \frac{2}{10}\right) \tag{2.1.19}$$

is the unique stationary distribution.

**Remark 2.1.3.21.** Throughout this work, the random walk matrix $\mathbf{P} = \frac{1}{2}(\mathbf{I} + \mathbf{A}\mathbf{D}^{-1})$ adopted in Chapter 3 follows the column-stochastic convention, in which $P_{ij}$ denotes the probability of transitioning from state $j$ to state $i$. Under this convention, the stationary distribution satisfies $P\pi = \pi$ for a column vector $\pi$, which is the transpose of the row-stochastic formulation $\pi P = \pi$. All definitions and results in this section are stated in the row-stochastic convention and carry over to the column-stochastic setting by transposition.

## 2.2 Graph Neural Networks

The success of deep learning across domains such as computer vision has led to growing interest in applying Multi-Layer Perceptrons (MLPs) to CO problems which are, for the most part, naturally expressed as graphs (Cappart et al., 2022). However, a few peculiar challenges arise when applying MLPs to general graph-structured data without modification.

Machine learning models typically operate on rectangular or grid-like arrays as input. While a graph's node features can be encoded as a feature matrix $\mathbf{X} \in \mathbb{R}^{n\times d}$, with $n$ being the number of nodes and $d$ the feature dimension, and connectivity can be encoded via the adjacency matrix, doing so naively introduces a fundamental problem. Since relabeling the nodes of a graph does not alter the graph itself, any meaningful representation should reflect this invariance. In practice, however, a relabeling induces a different adjacency matrix and a different feature matrix, for which an MLP will learn an entirely different representation. This is particularly problematic since a graph with $n$ nodes admits $n!$ possible labelings, each yielding a distinct matrix representation. As an MLP has no mechanism to recognise these as equivalent, it cannot learn a representation that generalises across different presentations of the same underlying graph. This property is known as permutation invariance, and is the central challenge in applying standard MLPs to graph-structured data.

A further challenge arises from the sparsity of real-world combinatorial optimization instances (Cappart et al., 2022). Such graphs tend to be large, with most node pairs sharing no edge. Representing them as adjacency matrices requires $O(n^2)$ space and time complexity, the vast majority of which corresponds to absent edges. This makes direct MLP-based approaches both memory-intensive and computationally wasteful, and thus ill-suited to scale to the large instances that arise in practice.

In the sections that follow, we detail the implementation of a standard Graph Neural Network (GNN) to see how they address these challenges raised. We also consider a GNN-variant known as the Graph Attention Network (GAT) that is directly relevant to this thesis.

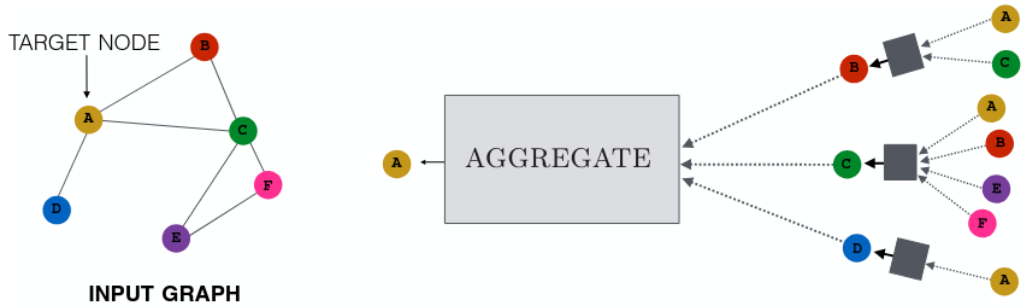



Figure 2.5: Message passing framework in GNNs (Hamilton, 2020).

**2.2.1 Message-Passing Graph Neural Networks.** In this section, we describe the Message-Passing Neural Network (MPNN) framework, the original work of Gilmer et al. (2017). We also discuss the challenges raised in the preceding section that the MPNN addresses, and note some limitations that motivate the GAT discussed in Section 2.2.2.

Like was mentioned in the preceding section, the key challenges in applying MLPs to graph-structured data are finding a permutation equivariant representation of the graph and handling the sparsity of real-world graphs. The MPNN addresses these by having each node aggregate neighboring information using an operation that is commutative or at least insensitive to the order of its operands, such as a sum or maximum, and use it to update its own representation. Since each node only aggregates over its immediate neighbors, computation scales with the number of edges rather than $n^2$, making the approach suitable for the sparse graphs that arise in practice. The MPNN architecture can be thought of as consisting of two parts: a feature transformation step and the message-passing layers. We now proceed to describe each part of the MPNN architecture:

**Feature Transformation.** The feature transformation step is a preprocessing operation applied before message passing begins. Given an input graph $G$ with feature matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$, a shared MLP is applied independently to each node's features. This produces an initial per-node embedding

$$\mathbf{h}_v^{(0)} = \mathrm{MLP}(\mathbf{x}_v) \in \mathbb{R}^{d'}, \qquad v \in V. \tag{2.2.1}$$

Intuitively, the MLP learns a nonlinear transformation of the raw features into a space the network finds more useful for the task at hand. Combinations of features that contribute to reducing the

loss are emphasized, while those that do not are suppressed. We note that the output dimension $d'$ of the MLP need not equal $d$, and so the resulting embeddings $\mathbf{h}_v^{(0)}$ may live in a space of different dimension than the input features.

**Message Passing.** In the second stage, the graph structure is exploited. We organize this stage into $K$ layers indexed by $k = 1, \ldots, K$. At each layer $k$, every node $v$ aggregates the feature representations of its neighbors, and uses the result to update its own representation. The operations of a single layer are given by

$$\mathbf{m}_v^{(k)} = \bigoplus_{u \in \mathcal{N}(v)} M^{(k)}\big(\mathbf{h}_v^{(k-1)}, \mathbf{h}_u^{(k-1)}\big)\,, \tag{2.2.2}$$

$$\mathbf{h}_v^{(k)} = U^{(k)}\big(\mathbf{h}_v^{(k-1)}, \mathbf{m}_v^{(k)}\big)\,, \tag{2.2.3}$$

where $\mathbf{m}_v^{(k)}$ is the aggregated message at node $v$, and $\mathbf{h}_v^{(k)}$ is the updated representation of $v$ after layer $k$. The operator $\bigoplus$ denotes an aggregation function, usually a sum or an average which are invariant to the labeling of the nodes. $M^{(k)}$ is a learned message function that produces, for each pair $(v, u)$ with $u \in \mathcal{N}(v)$, a vector capturing the information that $u$ should pass to $v$ at layer $k$. The simplest choice is to take the neighbor's previous embedding directly, that is, $M^{(k)}(\mathbf{h}_v, \mathbf{h}_u) = \mathbf{h}_u$. A more expressive choice is a learned linear map $M^{(k)}(\mathbf{h}_v, \mathbf{h}_u) = W^{(k)}\mathbf{h}_u$ for some weight matrix $W^{(k)}$, and the most general choice is a small MLP applied to the concatenation of the two embeddings. $U^{(k)}$ is a learned update function that combines $v$'s previous embedding with the aggregated message to produce $\mathbf{h}_v^{(k)}$. A typical instantiation is $U^{(k)}(\mathbf{h}_v, \mathbf{m}_v) = \sigma\Big(W_1^{(k)}\mathbf{h}_v + W_2^{(k)}\mathbf{m}_v\Big)$, where $W_1^{(k)}$ and $W_2^{(k)}$ are learned weight matrices and $\sigma$ is a nonlinear activation. As with $M^{(k)}$, $U^{(k)}$ may also be implemented as a small MLP applied to the concatenation $[\mathbf{h}_v \,\|\, \mathbf{m}_v]$.

This formulation directly addresses some of the challenges raised in the previous section. Since aggregation at each node is performed only over $\mathcal{N}(v)$, the actual neighbors of $v$, the computation at each layer scales with the number of edges rather than with $n^2$, addressing the sparsity challenge. Furthermore, since $\bigoplus$ is permutation-invariant, the learned representations do not depend on the particular labeling of the nodes, addressing the permutation invariance challenge. After $K$ layers, every node $v$ carries a representation $\mathbf{h}_v^{(K)}$ that encodes information from its $K$-hop neighborhood, on which predictions can then be done.

**Remark 2.2.1.1.** Once the $K$ message-passing layers have been applied, each node $v$ carries a final embedding $\mathbf{h}_v^{(K)}$ encoding information from its $K$-hop neighborhood. To produce a per-node prediction, this embedding is passed through a node-wise classifier, which is typically a shared MLP applied independently to each $\mathbf{h}_v^{(K)}$. Since this classifier is applied to one node at a time and uses the same parameters for every node, the permutation invariance established by the message-passing step is preserved. The full pipeline therefore consists of a preprocessing MLP, $K$ message-passing layers, and a per-node MLP classifier, trained end-to-end against a standard loss for node classification.

We now consider an example to illustrate message-passing:

**Example 2.2.1.2.** Consider the graph $G$ from Example 2.2a, illustrated in Figure 2.6. Suppose each node stores a single scalar feature, so $d = 1$, and take

$$\mathbf{x}_1 = 1, \quad \mathbf{x}_2 = 2, \quad \mathbf{x}_3 = 3, \quad \mathbf{x}_4 = 4. \tag{2.2.4}$$

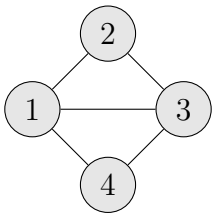



Figure 2.6: The graph $G$ from Example 2.2a, used to illustrate message passing.

For simplicity, we skip the feature transformation step and set $\mathbf{h}_v^{(0)} = \mathbf{x}_v$ for each $v \in V$. We further take $M^{(1)}(\mathbf{h}_v, \mathbf{h}_u) = \mathbf{h}_u$, $\bigoplus$ = mean, and $U^{(1)}(\mathbf{h}_v, \mathbf{m}_v) = \mathbf{m}_v$. Under these choices, a single message-passing layer reduces to replacing each node's feature with the mean of its neighbors' features.

Node $1$ has neighbors $\mathcal{N}(1) = \{2, 3, 4\}$, so

$$\mathbf{h}_1^{(1)} = \frac{1}{3}(\mathbf{h}_2^{(0)} + \mathbf{h}_3^{(0)} + \mathbf{h}_4^{(0)}) = \frac{1}{3}(2 + 3 + 4) = 3. \tag{2.2.5}$$

Similarly, $\mathcal{N}(2) = \{1, 3\}$, $\mathcal{N}(3) = \{1, 2, 4\}$, and $\mathcal{N}(4) = \{1, 3\}$, giving

$$\mathbf{h}_2^{(1)} = 2, \qquad \mathbf{h}_3^{(1)} = \frac{7}{3}, \qquad \mathbf{h}_4^{(1)} = 2. \tag{2.2.6}$$

Notice that $\mathbf{h}_2^{(1)} = \mathbf{h}_4^{(1)}$. This is no coincidence: nodes $2$ and $4$ share the same neighborhood, $\mathcal{N}(2) = \mathcal{N}(4) = \{1, 3\}$, and under uniform aggregation they receive identical updates regardless of their own initial features.

We now discuss some limitations of standard MPNNs that motivate other GNN architectures.

**Limitations.** Despite these strengths, the MPNN framework has some structural limitations. The standard MPNN aggregation treats all neighboring nodes equally, assigning the same importance to every neighbor regardless of how relevant their features are to the central node. In the context of CO, this poses an inherent problem since there could exist nodes at transition points which need to be distinguished from the rest of their neighborhood, yet uniform aggregation washes out exactly the kind of feature contrast that would make this distinction visible. Section 2.2.2 introduces the Graph Attention Network (GAT), which addresses this shortcoming by learning, rather than fixing, the relative importance of each neighbor.

**2.2.2 Graph Attention Networks.** As noted in Section 2.2.1, the standard MPNN aggregates information from all neighboring nodes equally, assigning the same importance to every neighbor regardless of how relevant their features are to the central node. In general, however, not all

neighbors contribute equally informative signals: some may carry features that are highly relevant to the task at hand, while others may be largely uninformative or even introduce noise into the aggregation. For instance, in graphs exhibiting heterophily, where connected nodes often belong to different classes, uniformly averaging neighbor features can dilute the very signal the model needs to recover. Graph Attention Networks (GATs) introduced by Veličković et al. (2018) address this situation. They learn to assign different importance scores to different neighbors during aggregation.

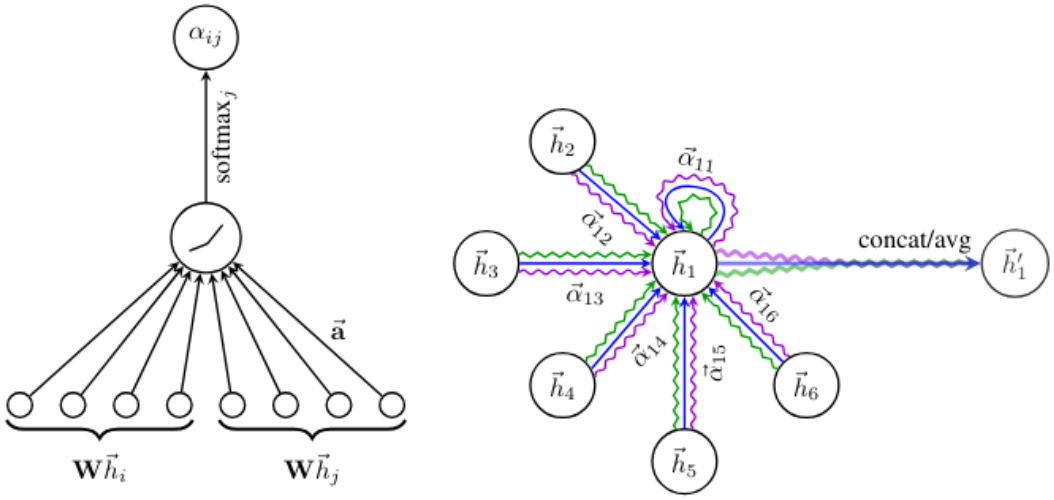



Figure 2.7: Graph Attention Network mechanism (Veličković et al., 2018).

**Attention Scores and Weights.** Given a graph $G = (V, E)$, we represent the features of a single node $v \in V$ by $h_v \in \mathbb{R}^d$, where $d$ is the number of features stored at each node. For a given layer $\ell$ in the GAT framework, we let $h_v^{(\ell-1)} \in \mathbb{R}^d$ denote the input feature vector of node $v$ to that layer.

Much like the MPNN, the GAT first does some feature transformation; unlike the standard MPNN, however, this transformation is learned anew at each layer. At layer $\ell$, it is parameterized by a single weight matrix $W^{(\ell)} \in \mathbb{R}^{d' \times d}$, and the transformed feature vector of node $v$ is given by

$$W^{(\ell)} h_v^{(\ell-1)} \in \mathbb{R}^{d'}. \tag{2.2.7}$$

An attention mechanism $a : \mathbb{R}^{d'} \times \mathbb{R}^{d'} \to \mathbb{R}$, parameterized by a learnable vector $\mathbf{a}^{(\ell)} \in \mathbb{R}^{2d'}$, then enables the network to compute attention scores between pairs of nodes. In its most general form, the mechanism computes scores without regard for the underlying graph structure, evaluating a score between a given node and every other node in the graph. Restricting the computation of attention scores to pairs of nodes that are connected in $G$ is referred to as *masked attention*, and is the formulation adopted in the original GAT.

Under masked attention, the score between nodes $v_i$ and $v_j$ such that $\{v_i, v_j\} \in E$ is defined as

$$e_{ij}^{(\ell)} := \mathbf{a}^{(\ell)\top} \left[ W^{(\ell)} h_i^{(\ell-1)} \,\|\, W^{(\ell)} h_j^{(\ell-1)} \right] \in \mathbb{R}, \tag{2.2.8}$$

where $\|$ denotes vertical concatenation, so that $W^{(\ell)}h_i^{(\ell-1)} \,\|\, W^{(\ell)}h_j^{(\ell-1)} \in \mathbb{R}^{2d'}$. To make scores comparable across the (possibly differently sized) neighborhoods of different nodes, the raw scores are normalized over each node's neighborhood $\mathcal{N}(i)$ via a softmax, yielding the *attention weights*

$$\alpha_{ij}^{(\ell)} := \frac{\exp\big(e_{ij}^{(\ell)}\big)}{\sum_{k\in\mathcal{N}(i)} \exp\big(e_{ik}^{(\ell)}\big)}. \tag{2.2.9}$$

The updated representation of node $v_i$ at layer $\ell$ is then computed as a weighted sum of its neighbors' transformed features,

$$h_i^{(\ell)} = \sigma\left(\sum_{j\in\mathcal{N}(i)} \alpha_{ij}^{(\ell)}\, W^{(\ell)}h_j^{(\ell-1)}\right), \tag{2.2.10}$$

where $\sigma$ is a nonlinear activation function, and $i \in \mathcal{N}(i)$.

**Multi-Head Attention.** A single attention mechanism, as described above, can be unstable to train and may capture only a narrow view of how neighbors relate to a given node. To address this, the GAT employs *multi-head attention*, in which $K$ independent attention mechanisms are applied in parallel at each layer. Each head $k \in \{1, \dots, K\}$ has its own learnable weight matrix $W^{(\ell),k}$ and attention vector $\mathbf{a}^{(\ell),k}$, and therefore produces its own set of attention weights $\alpha_{ij}^{(\ell),k}$ and a corresponding updated representation of node $v_i$.

The $K$ resulting representations must then be combined into a single output for node $v_i$ at layer $\ell$. At intermediate layers, the representations from each head are concatenated, yielding

$$h_i^{(\ell)} = \Big\|_{k=1}^{K} \sigma\left(\sum_{j\in\mathcal{N}(i)} \alpha_{ij}^{(\ell),k}\, W^{(\ell),k}h_j^{(\ell-1)}\right), \tag{2.2.11}$$

so that the output dimension at that layer becomes $Kd'$ rather than $d'$. At the final (prediction) layer, however, concatenation is no longer appropriate, since the output must match the dimension expected by the downstream task. In this case, the representations from the $K$ heads are averaged instead, and the nonlinearity is applied only after averaging:

$$h_i^{(\ell)} = \sigma\left(\frac{1}{K}\sum_{k=1}^{K}\sum_{j\in\mathcal{N}(i)} \alpha_{ij}^{(\ell),k}\, W^{(\ell),k}h_j^{(\ell-1)}\right). \tag{2.2.12}$$

# 3. Graph Combinatorial Optimization Network

This chapter presents the Graph Combinatorial Optimization Network (GCON) architecture proposed by Wenkel et al. (2024). We describe the two main components of the architecture: the filter bank and the attention over filters. We also present the statement and proof of a theorem that justifies the empirical gains of GCON over other benchmarks. We note that all operations are described for a single layer of the network.

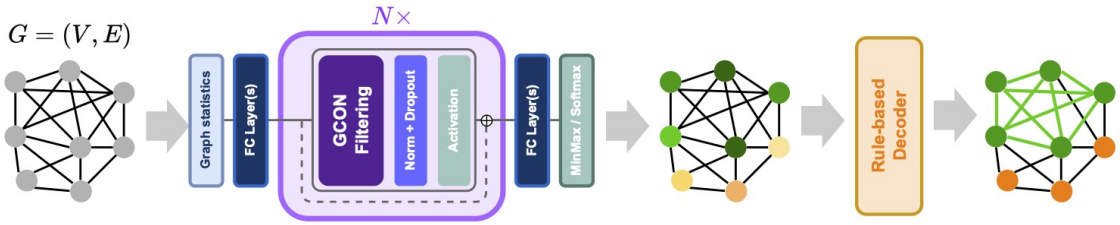



Figure 3.1: GCON Architecture (Wenkel et al., 2024).

## 3.1 The GCON Filter Bank

The GCON filter bank forms the backbone of the feature extraction pipeline of the GCON architecture. As has been established in previous chapters, smoothness as an inductive bias is ill-suited in the context of CO problems, since these problems and their graphs do not often come with inherent node features. The filter bank remedies this by allowing for operators that both aggregate the features as well as compare them, thus capturing complementary aspects of the graph structure. In this section we describe the filter bank and the two classes of filters it accommodates.

Unless otherwise cited, all definitions and theorems in this section are from Wenkel et al. (2024, sec. 5).

**3.1.1 Aggregation and Comparison Filters.** The filter bank, denoted by $\mathcal{F}$, of the GCON framework is partitioned into two filter classes: the **aggregation filter class**, $\mathcal{F}_A$, and the **comparison filter class**, $\mathcal{F}_C$. Before describing the elements of each of these classes of the filter bank, we state an important definition.

**Definition 3.1.1.1** (Shift Operators)**.** Given a graph $G = (V, E)$, a matrix $S \in \mathbb{R}^{n\times n}$ is said to be a shift operator of $G$ if $S_{ij}$ is non-zero for adjacent vertices $i, j \in V\, i \neq j$ and $S_{ij} = 0$ for non-adjacent vertices $i, j \in V\, i \neq j$.

**Remark 3.1.1.2.** The above definition puts no restriction on the elements of the diagonal $S_{ii}$ of a shift operator $S$ hence $S_{ii}$ could be any element of $\mathbb{R}$.

**Example 3.1.1.3.** Let $G = (V, E)$ be a simple, connected, undirected graph. Then the graph Laplacian $L$, introduced in Section 2.1, is a valid shift operator on $G$. Additionally, the lazy random walk

$$\mathbf{P} = \frac{1}{2}\left(\mathbf{I} + \mathbf{A}\mathbf{D}^{-1}\right), \tag{3.1.1}$$

is also a valid shift operator on $G$.

**Remark 3.1.1.4.** For adjacent nodes $i, j$ we have that $P_{ij} = \frac{1}{d_j}$, where $d_j$ is the degree of node $j$, and the diagonal entries satisfy $P_{ii} = \frac{1}{2}$ for all $i \in V$. $\mathbf{P}$ defines a lazy random walk on $G$ since at each step, the walker either moves to a neighboring node with probability $0.5$ or stays put with $0.5$. We also note that $\mathbf{P}$ is a column-stochastic matrix.

We now discuss the different components of the GCON architecture.

### Aggregation Filters

Given some node $i$ in our graph of interest, aggregation filters are not only able to smooth features of neighboring nodes, but also extend this smoothing operation to multi-hop neighborhoods, capturing feature information up to $k$ hops away after $k$ successive aggregations.

More specifically, aggregation filters are defined as

$$F_k(\mathbf{X}) := \mathrm{m}\left(\mathbf{P}^k\mathbf{X}\right), \quad k \geq 1 \tag{3.1.2}$$

where $k$ is a hyperparameter that encodes how many hops of neighborhood information are aggregated and $\mathbf{X}$ is the feature matrix. $\mathrm{m}$ on the other hand is a learned transformation that projects $\mathbf{P}^k\mathbf{X}$ into a more expressive feature space.

### Comparison Filters

While aggregation filters smooth features in some $k-$neighborhood of each node $i$ in a graph, comparison filters, on the other hand, compare the features in two different neighborhoods of each node. In particular, comparison filters are of the form

$$F_{k_1,k_2}(\mathbf{X}) := \mathrm{m}\left(\left(\mathbf{P}^{k_1} - \mathbf{P}^{k_2}\right)\mathbf{X}\right), \quad k_1 \neq k_2. \tag{3.1.3}$$

Like the aggregation filters, $k_1$ and $k_2$ are hyperparameters to be specified, $\mathrm{m}$ is a learned transformation and $\mathbf{X}$, the feature matrix of the input graph. It is clear to see that comparison filters first compute aggregations of features over two distinct neighborhoods of each node, then take their difference. In doing so, they capture feature information that differs across these neighborhoods which is the advantage the GCON framework has over typical MPNNs in solving CO problems.

## 3.2 Attention over Filters

In the previous section, we introduced the filter bank, which forms the backbone of the GCON feature extraction pipeline. We now describe the attention mechanism over filters, which allows

the network to learn the importance of each filter relative to the others in the same filter class, at each node. Notably, while Min et al. (2022) also employs an attention mechanism to balance filter contributions, GCON departs from this by computing separate attention scores for each filter class. As we shall see, this choice has theoretical backing explaining why it affords the model greater node-wise adaptability. Algorithm 1 details how this is implemented.

**Algorithm 1** GCON Attention over Filters (Single Layer)

**Require:** Input features $\mathbf{X}^{\ell-1} \in \mathbb{R}^{n\times d}$, filter bank $\mathcal{F} = \mathcal{F}_A \sqcup \mathcal{F}_C$, learnable attention vectors $\mathbf{a}_A, \mathbf{a}_C \in \mathbb{R}^{2d}$

**Ensure:** Updated node features $\mathbf{X}^{\ell} \in \mathbb{R}^{n\times d}$

$\mathbf{H} \leftarrow \mathrm{m}(\mathbf{X}^{\ell-1})$ ▷ transformed input

**for** each filter $f \in \mathcal{F}$ **do**

  $\mathbf{H}_f \leftarrow f(\mathbf{X}^{\ell-1})$ ▷ filter response

  $\mathbf{Z}_f \leftarrow [\mathbf{H} \,\|\, \mathbf{H}_f]$ ▷ concatenation, $\mathbf{Z}_f \in \mathbb{R}^{n\times 2d}$

  **if** $f \in \mathcal{F}_A$ **then**

    $s_f^A \leftarrow \sigma(\mathbf{Z}_f\, \mathbf{a}_A)$ ▷ attention score, aggregation class

  **else if** $f \in \mathcal{F}_C$ **then**

    $s_f^C \leftarrow \sigma(\mathbf{Z}_f\, \mathbf{a}_C)$ ▷ attention score, comparison class

  **end if**

**end for**

**for** every $v \in V$ **do** ▷ normalize within each class

  $\bar{s}_f^A(v) \leftarrow \sigma(\{s_f^A[v] : f \in \mathcal{F}_A\})$

  $\bar{s}_f^C(v) \leftarrow \sigma(\{s_f^C[v] : f \in \mathcal{F}_C\})$ ▷ $\sigma$ is $\mathrm{softmax}$

**end for**

$\mathbf{H}_{\mathcal{F}_A} \leftarrow \sum_{f\in\mathcal{F}_A} \left(\bar{\mathbf{s}}_f^A \circ \mathbf{1}_d\right) \odot \mathbf{H}_f$ ▷ aggregate within $\mathcal{F}_A$

$\mathbf{H}_{\mathcal{F}_C} \leftarrow \sum_{f\in\mathcal{F}_C} \left(\bar{\mathbf{s}}_f^C \circ \mathbf{1}_d\right) \odot \mathbf{H}_f$ ▷ aggregate within $\mathcal{F}_C$

▷ $\circ$ is the outer product

▷ $\odot$ is the Hadamard product

$\mathbf{X}^{\ell} \leftarrow \mathrm{MLP}\left(\mathbf{X}^{\ell-1} + \mathbf{H}_{\mathcal{F}_A} + \mathbf{H}_{\mathcal{F}_C}\right)$ ▷ update node features

Intuitively, each layer $\ell$ accepts the node features $\mathbf{X}^{\ell-1}$ from the previous layer as input, and applies a learned transformation $\mathrm{m}$ to obtain a base feature matrix $\mathbf{H}$. The network then computes filter responses (aggregations or comparisons of node features), giving it access to the distinct structural views the filter bank provides. Before computing the attention score for each filter $f$, $\mathbf{H}$ is concatenated with the filter response $\mathbf{H}_f$, so that the network has simultaneous access to the base representation and the particular view of the graph that $f$ captures. A learnable attention vector then produces, for each node, a score reflecting the relative importance of that filter within its class. These scores are normalized via softmax, ensuring they lie in $(0, 1)$ and sum to $1$ within each filter class $\mathcal{F}_A$ and $\mathcal{F}_C$ separately.

For each filter in each class, the filter response at each node is rescaled by the corresponding normalized attention score. This reweighting is what allows the network to emphasize the most relevant filters differently for each node. The reweighted responses are then summed within each filter class separately, yielding two class-level representations $\mathbf{H}_{\mathcal{F}_A}$ and $\mathbf{H}_{\mathcal{F}_C}$. The final output

of the layer is then computed by passing the sum of the input features $\mathbf{X}^{\ell-1}$, the aggregation representation $\mathbf{H}_{\mathcal{F}_A}$, and the comparison representation $\mathbf{H}_{\mathcal{F}_C}$ through an MLP, giving the updated node features $\mathbf{X}^{\ell}$.

## 3.3 Theoretical Justification

In the previous sections, we described the two main components of the GCON architecture: the filter bank in 3.1 and the attention mechanism over filters in 3.2 which is similar to the architecture proposed by Min et al. (2022). We have also noted that the attention scores are computed separately over the different filter classes, which in that case is different from the network from Min et al. (2022). A natural question arises, then, is whether the design choice in the GCON architecture, is theoretically justified. In this section, we present a theorem from Wenkel et al. (2024, sec. 5.3) that answers this question affirmatively. The theorem shows that under a condition on the relative dominance of comparison filter attention scores, the aggregation filter responses will always dominate when filter scales are sufficiently large. This provides theoretical grounding for why the decoupled attention design affords the model greater node-wise adaptability.

### 3.3.1 Preliminary Definitions.

**Definition 3.3.1.1** (Band-dominant representation (Wenkel et al., 2024, sec. 5.3))**.** Take a graph $G = (V, E)$ together with a non-negative input signal $\mathbf{x} \in \mathbb{R}^n$ and a filter bank $\mathcal{F} = \mathcal{F}_A \sqcup \mathcal{F}_C$. We call the representation $\mathbf{x} \mapsto (\bar{\mathbf{s}}_f[v], \mathbf{h}_f[v])_{f\in\mathcal{F}}$, where $\bar{\mathbf{s}}_f[v]$ is the normalized attention score of $f \in \mathcal{F}$ at node $v$ and $\mathbf{h}_f[v]$ is the result of the filter $f \in \mathcal{F}$ at a node $v$, *$c$-band dominant* when the largest comparison attention score at $v$ is exactly $c$ times the largest aggregation attention score, that is,

$$\max\{\bar{\mathbf{s}}_f[v] : f \in \mathcal{F}_C\} = c \cdot \max\{\bar{\mathbf{s}}_f[v] : f \in \mathcal{F}_A\}, \tag{3.3.1}$$

for some $c > 1$.

**Remark 3.3.1.2.** The concept of band-dominance encodes the idea that at some node, the filter with the highest importance is a comparison filter. This means that at that node it is better to compare its features with the features in its neighborhood, rather than smoothing over the features.

### 3.3.2 Main Theorem.

We now present the statement of the theorem justifying the empirical gains of GCON over other benchmarks and its proof in Appendix A.4.

**Theorem 3.3.2.1** ((Wenkel et al., 2024, sec. 5.3))**.** *Take the non-decoupled architecture and suppose the representation $(\bar{\mathbf{s}}_f[v], \mathbf{h}_f[v])_{f\in\mathcal{F}}$ at a node $v \in V$ is $c$-band dominant for a non-negative input signal $\mathbf{x} \in \mathbb{R}^n$ with $x_i \geq 0$ for every $i$, where $\bar{\mathbf{s}}_f[v]$ is the normalized attention score of $f \in \mathcal{F}$ at node $v$ and $\mathbf{h}_f[v]$ is the result of the filter $f \in \mathcal{F}$ at node $v$. Then, provided every filter is run at a sufficiently large scale, the aggregation filter responses always dominate the comparison filter responses, in the sense that*

$$\sum_{f\in\mathcal{F}_C} \bar{\mathbf{s}}_f[v]\, \mathbf{h}_f[v] \;\leq\; \epsilon \cdot c \cdot \sum_{f\in\mathcal{F}_A} \bar{\mathbf{s}}_f[v]\, \mathbf{h}_f[v]. \tag{3.3.2}$$

# 4. Experiments and Results

This chapter presents the implementation and experimental evaluation of the GCON framework across five combinatorial optimization problems. We begin by introducing the three problems considered in the original work of Wenkel et al. (2024): Maximum Cut, Maximum Clique, and Minimum Dominating Set (MDS). We present their associated loss functions and decoding procedures as given by the authors. We then extend the framework to three additional problems: Maximum Independent Set (MIS) and Balanced Graph Partitioning (GP). For each problem, we formulate a differentiable self-supervised loss function that encodes the combinatorial constraints as soft penalties, and pair it with a rule-based decoder that extracts a valid discrete solution from the continuous node-level outputs of the network.

All problems are treated within the unified GCON framework described in Chapter 3: for all but the GP problem, the network outputs a probability $p_i \in [0,1]$ for each node $i$, interpreted as the likelihood that $i$ belongs to the set of interest. For the GP problem into $k$ partitions, the network instead outputs a single vector $p_i \in [0,1]^k$ for each node $v_i$. The loss functions and decoders are the only components that differ across tasks, making the framework modular and readily extensible to new problems.

## 4.1 Problem Formulations

### 4.1.1 Maximum Cut.

**Definition 4.1.1.1** (Maximum Cut, Wenkel et al. (2024, sec. 3))**.** For a graph $G = (V, E)$ Maximum Cut problem asks to find a partition of the vertex set, $V = S \sqcup T$, in a way that maximizes the number of edges crossing between $S$ and $T$.

To cast this as an unsupervised learning problem, we recall that the GCON outputs a probability $p_i \in [0,1]$ for each vertex $v_i$, representing the likelihood that $v_i$ belongs to the set of interest. In the maximum cut setting, we have two sets rather than one, so we interpret $p_i$ as the probability that $v_i \in S$, and redefine the output as $\mathbf{y} := 2\mathbf{p} - \mathbf{1}_n$ so that $\mathbf{y} \in [-1,1]^n$, allowing us to cleanly assign vertices to either partition. The loss function used is:

$$L(\mathbf{y}) := \sum_{\{i,j\} \in E} y_i y_j \tag{4.1.1}$$

After learning $\mathbf{y}$, we assign vertices whose $\mathbf{y}$ value is greater than or equal to $0$ to the set $S$ and vertices whose $\mathbf{y}$ value is less than $0$ to set $T$.

With this formulation, $L$ is consistent with the goal of the problem since every edge connecting two vertices in different partitions decreases the loss. That is to say for $\{v_i, v_j\} \in E$ if $v_i \in S$ and $v_j \in T$ then $\text{sgn}(y_i) = -\text{sgn}(y_j)$, which yields a negative contribution to the loss. Conversely, edges within the same partition yield a positive contribution $y_i y_j > 0$, increasing the loss.

### 4.1.2 Maximum Clique.

**Definition 4.1.2.1** (Maximum Clique, Wenkel et al. (2024, sec. 3))**.** Given a graph $G = (V, E)$ the Maximum Clique problem asks to find a fully connected subgraph $C \subseteq V$ with the most number of vertices.

To cast this as an unsupervised learning problem, we let $p_i \in [0, 1]$ denote the probability that $v_i$ belongs to the clique. Let the complement graph $\overline{G} = (V, \overline{E})$ have adjacency matrix $\overline{\mathbf{A}}$, where $\overline{E} = \{\{v_i, v_j\} \notin E,\, i \neq j\}$. We define the following loss function:

$$L(\mathbf{p}) := -\sum_{(i,j)\in E} p_i p_j + \beta \sum_{\substack{(i,j)\notin E \\ i\neq j}} p_i p_j \tag{4.1.2}$$

Once $\mathbf{p}$ has been learned, we decode the solution as follows. We first reorder the vertices $\{v_i\}_{i=1}^n \to \{v_i'\}_{i=1}^n$ in descending order of their assigned probability, so that $\mathbf{p}(v_i') \geq \mathbf{p}(v_{i+1}')$. Starting with $C^{(1)} = \{v_1'\}$, we iterate over the remaining vertices in order: whenever $C^{(1)} \cup \{v_i'\}$ is still a clique, we include $v_i'$, and we skip it otherwise. We then repeat this construction $K$ times, each time seeding $C^{(k)} = \{v_k'\}$ and discarding $v_1', \ldots, v_{k-1}'$ from consideration. Our final answer is the $C^{(k)}$ of largest size.

To understand this loss function, notice that with the model minimizing the loss, it tries to push the first term $-\sum_{(i,j)\in E} p_i p_j$ to be as small as possible, which in this case is towards $-1$ for each summand, thus trying to simultaneously assign adjacent vertices to be in the solution set. The second term acts as the soft constraint the model has to satisfy, thus trying to minimize $\sum_{\substack{(i,j)\notin E \\ i\neq j}} p_i p_j$ towards $0$, thereby minimizing the number of non-adjacent vertices being assigned to the solution set.

### 4.1.3 Minimum Dominating Set.

**Definition 4.1.3.1** (Minimum Dominating Set, Wenkel et al. (2024, sec. 3))**.** Given a graph $G = (V, E)$ the Minimum Dominating Set problem seeks to find the smallest subset $S \subseteq V$ such that every node in $V$ is either in $S$ or has a neighbor in $S$.

To cast this as an unsupervised learning problem, we let $p_i \in [0, 1]$ denote the probability that $v_i$ belongs to the dominating set. We define the loss function as:

$$L(\mathbf{p}) := \|\mathbf{p}\|_1 + \beta \sum_{v_i \in V} \exp\left(\sum_{v_j \in \mathcal{N}_{v_i}} \log(1 - p_j)\right) \tag{4.1.3}$$

where $\mathcal{N}_v$ denotes the neighborhood of $v$, including $v$ itself.

The role of the second term in this loss function becomes clear once we consider what happens without it. The network could then minimise $\|\mathbf{p}\|_1$ all the way to zero and return the

empty set, which fails to be a dominating set since no vertex ends up being dominated. To prevent this degenerate outcome, the second term imposes a coverage penalty on each vertex that has not been adequately represented in the solution. For a fixed vertex $v_i \in V$, the quantity $\exp\left(\sum_{v_j \in \mathcal{N}_{v_i}} \log(1-p_j)\right)$ remains large whenever none of the vertices in $\mathcal{N}_{v_i}$ (including $v_i$ itself) carries a probability close to 1, and falls to nearly zero as soon as at least one $v_j \in \mathcal{N}_{v_i}$ satisfies $p_j \approx 1$. In effect, the second term drives the network to ensure that every vertex is either placed in the dominating set or has a neighbour that is, which is precisely the domination requirement.

Once $\mathbf{p}$ has been learned, we reorder the vertices in descending order of their assigned probability, so that $\mathbf{p}(v'_i) \geq \mathbf{p}(v'_{i+1})$. Beginning with $S^{(1)} = \{v'_1\}$, we proceed through the ordering, one vertex at a time: at each step we check whether $S^{(1)}$ is already a dominating set, terminating if it is and otherwise appending $v'_{i+1}$ before continuing. The same construction is then carried out $K$ times in total, with the $k$-th run seeded at $S^{(k)} = \{v'_k\}$ and $v'_1, \ldots, v'_{k-1}$ removed from consideration. We report the smallest dominating set obtained across all iterations as our solution.

We now turn to our own extensions of the framework, formulating loss functions and decoders for two problems not considered in the original work: Maximum Independent Set and Balanced Graph Partitioning. For MIS, we formulate two loss functions, $L_{\text{lin}}$ and $L_{\text{QUBO}}$, which induce qualitatively different training dynamics; we evaluate both and compare their behaviour directly.

### 4.1.4 Maximum Independent Set.

**Definition 4.1.4.1** (Maximum Independent Set)**.** Given a graph $G = (V, E)$, find the largest subset $S \subseteq V$ such that no two vertices in $S$ are adjacent, that is, for all $i, j \in S$, $(i, j) \notin E$.

To cast this as an unsupervised learning problem, we let $p_i \in [0, 1]$ denote the probability that $v_i$ belongs to the independent set. A direct unconstrained relaxation of the constrained optimization problem given in Appendix A.1 is given as:

$$L_{\text{lin}}(\mathbf{p}) := -\sum_{v_i \in V} p_i + \beta \sum_{(v_i, v_j) \in E} p_i p_j. \tag{4.1.4}$$

While the first term drives the model to maximise the number of vertices assigned to the independent set. However, this alone admits the trivial solution: the model simply assigns probability 1 to every vertex, satisfying the objective without producing a valid independent set, unless we have a graph of only isolated nodes. To address this, we introduce a second term which enforces the constraint of non-adjacent vertices being in the set. This second term penalises the model, weighted by a factor of $\beta$, whenever a pair of neighbouring vertices are simultaneously assigned to the set.

We further consider a quadratic variant of the node term, yielding the loss

$$L_{\text{QUBO}}(\mathbf{p}) := -\sum_{v_i \in V} p_i^2 + \beta \sum_{(v_i, v_j) \in E} p_i p_j. \tag{4.1.5}$$

On binary inputs the two losses coincide, since $p_i^2 = p_i$ for $p_i \in 0, 1$. Both formulations share the same structure: the first term rewards large independent sets while the second penalises

adjacent vertices being simultaneously assigned to the solution. The key distinction lies in how they treat intermediate probabilities on the relaxed domain $[0, 1]$. Since $p_i^2 < p_i$ for $p_i \in (0, 1)$, the quadratic node term contributes less to reducing the loss at fractional values than the linear term does, imposing a harsher implicit penalty on indecisive assignments. $L_{\text{QUBO}}$ thus exerts stronger pressure on the network to commit to near-binary outputs during training. We report results for both formulations in Section 4.3.

**Remark 4.1.4.2.** The formulation in (4.1.5) corresponds to a variant of the Quadratic Unconstrained Binary Optimisation (QUBO) used in the codebase of Cantürk et al. (2026).

On decoding the solutions, after learning $\mathbf{p}$, we construct the independent set as follows. We reorder the vertices $\{v_i\}_{i=1}^n \to \{v_i'\}_{i=1}^n$ so that $\mathbf{p}(v_i') \geq \mathbf{p}(v_{i+1}')$. We initialize $C^{(1)} = \{v_1'\}$ and iterate over the vertices: at each step, if $v_i'$ is adjacent to any vertex already in $C^{(1)}$, we do not add it to the set, otherwise we do. We repeat this process $K$ times, initializing $C^{(k)} = \{v_k'\}$ on the $k$-th iteration and automatically excluding $v_1', \ldots, v_{k-1}'$ from $C^{(k)}$. The solution is given as the largest valid independent set $C^*$ found across all $K$ iterations.

### 4.1.5 Balanced Graph Partitioning.

**Definition 4.1.5.1** (Balanced Graph Partitioning)**.** Given a graph $G = (V, E)$ with $n$ vertices, a balanced partition seeks to partition the vertex set $V = \bigsqcup_{t=1}^{k} V_i$ into $k$ partitions so that each partition satisfies $|V_t| \approx n/k \; \forall t$ and the number of edges with endpoints in different partitons is minimized.

**Remark 4.1.5.2.** For the remainder of this work, we refer to the Balanced Graph Partitioning problem simply as Graph Partitioning (GP).

To cast this as an unsupervised learning problem, we formulate the following relaxation of the constrained optimization problem given in Appendix A.1:

$$L(\mathbf{P}) = \sum_{(v_i,v_j)\in E} \|\mathbf{p}_i - \mathbf{p}_j\|_2^2 + \beta \sum_{\substack{(v_i,v_j)\notin E \\ i\neq j}} \left(1 - \frac{1}{2}\|\mathbf{p}_i - \mathbf{p}_j\|_2^2\right)^2 + \gamma \sum_{c=0}^{k-1} \left(\sum_{v_i\in V} p_{ic} - \frac{n}{k}\right)^2 \tag{4.1.6}$$

The loss function directly encodes the objectives and constraints of the GP problem, with each term corresponding to a distinct requirement. The first term minimizes the distance between the partition assignment vectors of adjacent nodes, encouraging connected vertices to be placed in the same partition. The second term penalizes non-adjacent nodes whose assignment vectors are similar, pushing them toward distinct partitions. The third term enforces the balance constraint by penalizing deviations from the target partition size $n/k$.

Given the learned matrix $\mathbf{P}$, we decode the partition assignment of each node $v_i$ by taking the argmax over its softmax output: $\hat{c}_i = \operatorname{argmax}_c p_{ic}$. Since the network outputs a proper probability distribution over the $k$ partitions by construction, this procedure always returns a valid $k$-partition of $V$ without any post-processing.

## 4.2 Setup

**4.2.1 Datasets.** Since the combinatorial optimization problems considered in this work are NP-hard, generating sufficiently large labeled datasets for supervised learning is computationally intractable. We therefore follow the approach of Wenkel et al. (2024) and evaluate all methods on synthetic graphs, where the underlying generative model is controlled and instances can be produced at scale.

Rather than initializing each vertex with random or one-hot features, we follow Wenkel et al. (2024) and equip each graph with node features derived from structural graph statistics, which provide the model with meaningful information about the local and global geometry of the graph. For each vertex, we compute its degree, clustering coefficient, and triangle count; on the BA datasets, we additionally include the eccentricity, which we omit on RB graphs due to its computational cost. These features are then mapped to the hidden dimension of the GNN through a linear layer or a shallow MLP prior to the message-passing layers.

We make use of the Barabási-Albert (BA), RB, and Stochastic Block Model (SBM) generative models from the literature to produce the synthetic graph instances used throughout our experiments. More information can be found about these models in Appendix A.3.

**4.2.2 Baselines and Evaluation Metrics.** For each extension problem, we compare GCON against problem-specific baselines. For Maximum Independent Set, we compare against the GMIN greedy heuristic, which repeatedly selects the minimum-degree vertex in the remaining graph and removes it along with its neighbors. For Graph Partitioning, we compare against two baselines: a greedy balanced $k$-partition heuristic that assigns nodes in descending degree order to whichever non-full partition contains the most of their already-assigned neighbors, and spectral partitioning, the classical relaxation-based heuristic described in Algorithm 3. Spectral partitioning serves as a strong reference baseline since it is provably near-optimal for recovering planted communities in SBM-generated graphs. The full algorithms for each greedy heuristic are provided in Appendix A.2.

In addition to the non-learned baselines above, we compare GCON against **ScatteringClique**, the GNN architecture proposed by Min et al. (2022). Like GCON, ScatteringClique employs a hybrid filter bank consisting of aggregation and comparison filters combined through a localized attention mechanism. The two architectures differ primarily in how the attention scores are computed: Min et al. (2022) compute a single set of attention scores over the entire filter bank, whereas Wenkel et al. (2024) introduce separate attention vectors for the aggregation and comparison filters, decoupling the two filter classes. ScatteringClique is evaluated under conditions identical to GCON, using the same dataset, loss function, decoder, and training procedure.

For evaluation, we use problem-specific metrics computed as averages over the test set. For the Maximum Independent Set problem, we report the average independent set size across test graphs, with larger values indicating better performance. For the Graph Partitioning problem, we report the average cut fraction, defined as the ratio of edges crossing partitions to the total number of edges with smaller values indicating better performance.

All experiments are conducted on synthetic graphs generated using data generative models. This

is in line with the data benchmarking setup of Karalias and Loukas (2020) as well as Wenkel et al. (2024). The MIS experiments use RB graphs (see sec. 4.2.1) whereas the GP experiments use the SBM graphs. Both sets of experiments use graphs with $50 - 100$ nodes per instance, generated to a total dataset of $5000$ graphs. The dataset is split into $3000$ training, $1000$ validation, and $1000$ test graphs. For both problems, we adopt a $5$-fold cross-validation split assigning three folds to training, one fold to validation, and one fold to testing. However, due to computational constraints, only a single fold was used in our experiments, in the sense that there was no rotation of the folds. We follow Wenkel et al. (2024) and use node degree, local clustering coefficient, and triangle count as input features, adding eccentricity on BA graphs only, due to its computational cost on RB and SBM instances.

## 4.3 Results

In this section, we present our experimental results in two parts. First, we report our reproduction of the GCON benchmarks from Wenkel et al. (2024) on the three combinatorial optimization problems considered in the original paper: Maximum Cut, Maximum Clique, and Minimum Dominating Set. Due to computational constraints, we conduct a single run per problem rather than averaging over three runs as in the original work; nevertheless, our results closely match those reported by Wenkel et al. (2024), confirming the reproducibility of their findings. Second, we extend the evaluation of GCON to two additional combinatorial optimization problems not considered in the original paper, namely the Maximum Independent Set, which the original authors explicitly identified as a promising direction for future work, and the Graph Partitioning problem, specifically into $3$ partitions, in order to assess the generality of the architecture beyond the tasks for which it was originally designed.

Table 4.1: Reproduction results for GCON on small datasets (single run, seed=0), compared against results reported by Wenkel et al. (2024). MCut and MDS use BA-small graphs; MClique uses RB-small graphs. ↑ indicates higher is better, ↓ indicates lower is better.

| Task | GCON (Local Run) | GCON (Reported Figures) |
|---|---|---|
| MCut size ↑ | $726.59$ | $727.09 \pm 1.49$ |
| MClique size ↑ | $15.91$ | $15.87 \pm 0.15$ |
| MDS size ↓ | $30.44$ | $30.26 \pm 0.30$ |

**4.3.1 Reproduction Results.** Table 4.1 presents our reproduction results for GCON on the three original CO problems considered by Wenkel et al. (2024). For each task, the reported value is the average objective size obtained by the trained model across all graphs in the corresponding test set, following the evaluation protocol of Wenkel et al. (2024). Across all three tasks, our results are consistent with those reported in the original paper, falling within or very close to the reported standard deviation ranges. This confirms that our implementation faithfully reproduces the GCON framework, and provides a reliable basis for the extensions to additional CO problems presented in the following section.

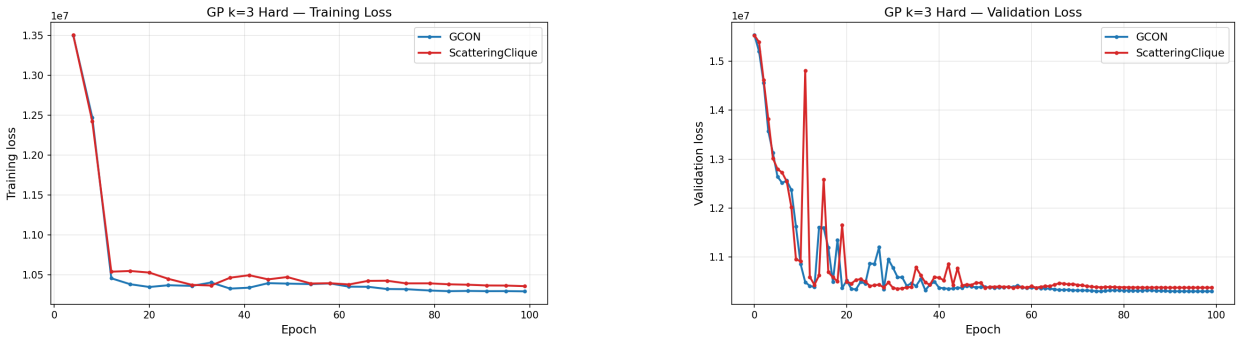
**4.3.2 Extension Results.** Building on the reproduction results above, we now evaluate GCON on the two new problems, starting with Graph Partitioning. For each problem, we compare GCON against ScatteringClique, introduced in Section 4.2.2. This comparison serves as a direct test of whether the decoupled attention of GCON, justified theoretically in Section 3.3, also yields empirical gains on problems beyond those considered by Wenkel et al. (2024).

### Graph Partitioning

Table 4.2: Graph Partitioning ($k = 3$, hard SBM) test-set results (single run). Lower is better.

| **Method** | **Cut Fraction** $\downarrow$ |
|---|---|
| Greedy | 0.632 |
| ScatteringClique | 0.436 |
| GCON | 0.332 |
| Spectral | 0.044 |

While the spectral baseline achieves the lowest cut fraction on this instance, cut fraction alone does not tell the full story, since the GP problem imposes two competing requirements: minimizing inter-partition edges and maintaining balanced partition sizes. As shown in Figure 4.2, the spectral baseline produces a severely imbalanced partition ($P0 = 65$, $P1 = 9$, $P2 = 1$), effectively collapsing almost all nodes into a single partition and thus failing the balance constraint entirely. Both learned methods recover well-balanced partitions, with GCON achieving ($P0 = 24$, $P1 = 27$, $P2 = 24$) and ScatteringClique achieving ($P0 = 25$, $P1 = 20$, $P2 = 30$), suggesting that the balance term in $L(\mathbf{P})$ is critical for producing solutions that genuinely respect the problem constraints.




(a) Training loss curves.

(b) Validation loss curves.

Figure 4.1: Training and validation loss curves for GCON and ScatteringClique on the hard SBM Graph Partitioning task ($k = 3$).

Despite the weak community structure of the hard SBM setting ($p_{in}/p_{out} \approx 2.3$), both architectures converge rapidly within the first fifteen epochs, as shown in Figure 4.1. GCON achieves a consistently lower training loss than ScatteringClique for the remainder of training. On the

validation set, Both architectures show some instability in earlier epochs, with the instability of ScatteringClique being more pronounced than that of GCON. Overall, for the most part the GCON loss remains lower than the ScatteringClique loss, even as both losses eventually converge. That both models train stably under these conditions suggests that the loss function provides a sufficiently strong learning signal even when the community structure is difficult to detect.

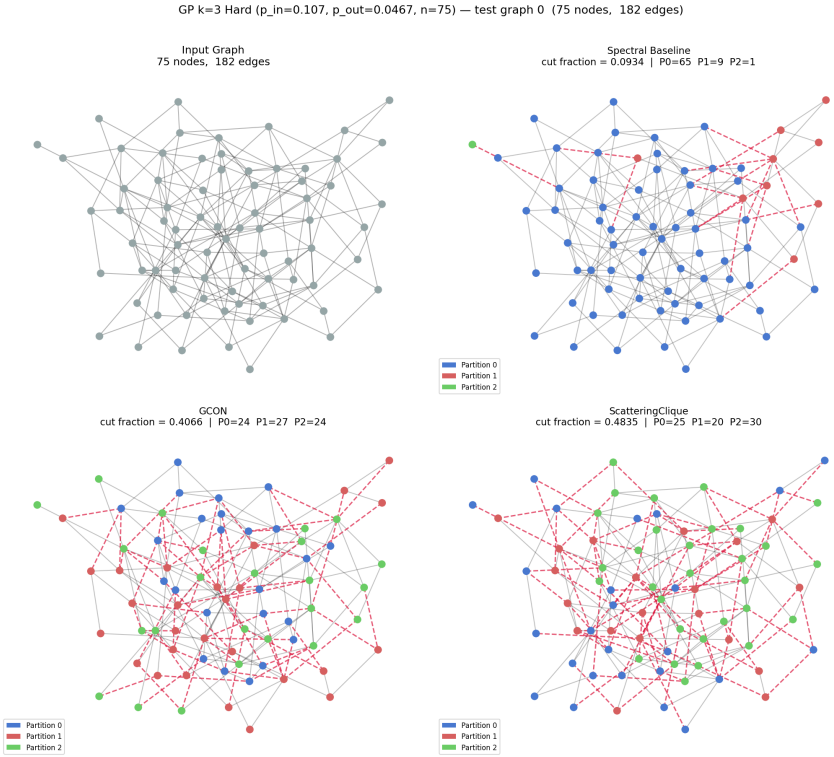



Figure 4.2: Graph Partitioning ($k = 3$, hard SBM) solutions on a representative test graph. Each colour denotes a distinct partition; dashed edges indicate cut edges crossing partition boundaries.

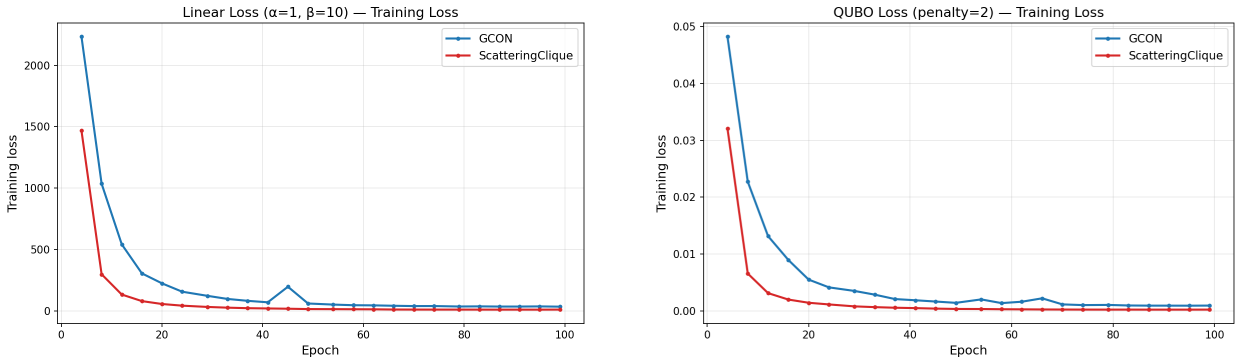
### Maximum Independent Set

Table 4.3: MIS test-set results on RB graphs (single run, seed=0). Average independent set size; higher is better.

| **Method** | $L_{\text{lin}} \uparrow$ | $L_{\text{QUBO}} \uparrow$ |
|---|---|---|
| Greedy (GMIN) | 6.664 | 6.664 |
| GCON | 5.336 | 5.778 |
| ScatteringClique | 5.537 | 4.861 |

Both GNN architectures fall short of the GMIN greedy heuristic in terms of average independent set size, though the margin is modest. A more interesting picture emerges when comparing across loss formulations: under $L_{\text{lin}}$, GCON ($5.336$) performs worse than ScatteringClique ($5.537$), yet

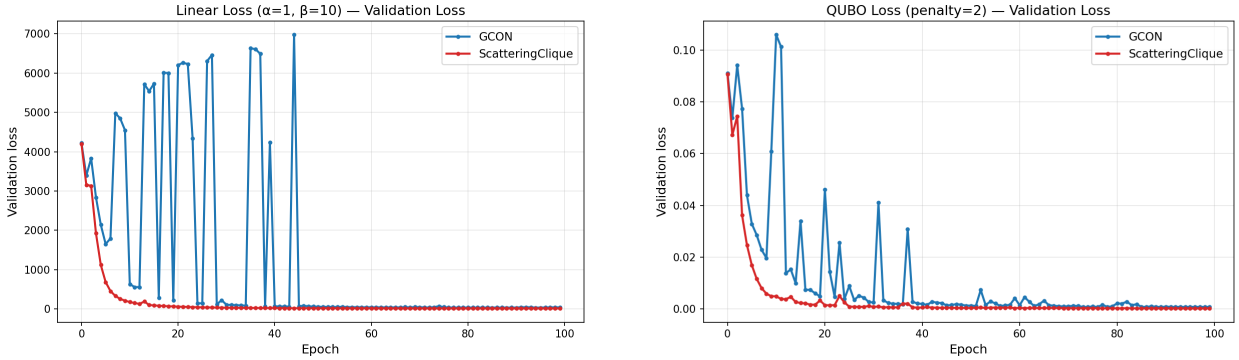
under $L_{\mathrm{QUBO}}$ the ranking reverses, with GCON $(5.778)$ substantially outperforming ScatteringClique $(4.861)$. This reversal indicates that the choice of loss function has a significant and architecture-dependent effect on solution quality, one that would not be apparent from evaluating either formulation in isolation.




(a) $L_{\mathrm{lin}}$ training loss.

(b) $L_{\mathrm{QUBO}}$ training loss.

Figure 4.3: MIS training loss curves for GCON and ScatteringClique under $L_{\mathrm{lin}}$ (left) and $L_{\mathrm{QUBO}}$ (right) on RB graphs.

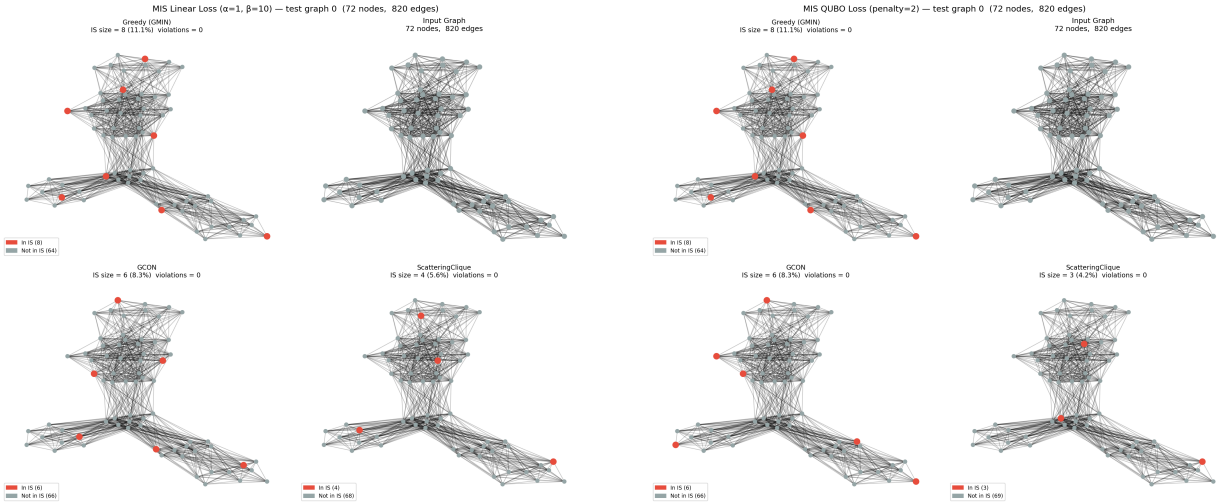



(a) $L_{\mathrm{lin}}$ validation loss.

(b) $L_{\mathrm{QUBO}}$ validation loss.

Figure 4.4: MIS validation loss curves for GCON and ScatteringClique under $L_{\mathrm{lin}}$ (left) and $L_{\mathrm{QUBO}}$ (right) on RB graphs.

The loss curves provide further insight into this result. Under both formulations, both architectures converge smoothly on the training set. The key distinction lies in the validation dynamics: ScatteringClique converges rapidly and stably on the validation set under both loss functions, whereas GCON exhibits pronounced and repeated spikes throughout the earlier epochs, indicating difficulty in generalizing during the initial phase of training. Despite this instability, GCON's validation loss eventually converges to a level comparable to that of ScatteringClique under both formulations, suggesting that the early instability does not prevent the model from ultimately finding a reasonable solution.




(a) Solutions under $L_{\text{lin}}$.

(b) Solutions under $L_{\text{QUBO}}$.

Figure 4.5: MIS solutions on a representative test graph (72 nodes, 820 edges) under $L_{\text{lin}}$ (left) and $L_{\text{QUBO}}$ (right). Red nodes are in the independent set; all solutions have zero constraint violations.

Figure 4.5 shows the solutions produced on a representative test graph of 72 nodes and 820 edges. All methods produce valid independent sets with zero constraint violations under both loss functions. On this particular instance, GCON recovers an independent set of size $6$ under both $L_{\text{lin}}$ and $L_{\text{QUBO}}$, while ScatteringClique finds $4$ and $3$ nodes respectively, and the greedy baseline finds $8$. We note that this single instance is not necessarily representative of the average test-set behaviour reported in Table 4.3: in particular, GCON outperforms ScatteringClique on this graph under $L_{\text{lin}}$, whereas the average results show the opposite. This highlights the variance inherent in single-instance evaluations and underscores the importance of reporting aggregate metrics over the full test set.

# 5. Conclusion & Future Work

## 5.1 Limitations

The limitations of this work are largely computational. All experiments were restricted to small-scale graphs, single-seed runs, and trial-and-error hyperparameter selection, none of which we expect to be inconsequential given the nature of self-supervised CO losses.

One thing our experiments did make clear is that the choice of loss function matters more than one might expect. On MIS, simply switching from $L_{\text{lin}}$ to $L_{\text{QUBO}}$ was enough to reverse the relative ranking of the two architectures, suggesting that loss design deserves careful thought when extending GCON to new problems.

## 5.2 Applications

The problems studied in this work arise naturally in a wide range of real-world settings. We highlight one concrete application of some of the problems discussed to illustrate the practical relevance of finding good approximate solutions efficiently.

**Graph Partitioning (Çatalyürek et al., 2023):** In distributed database systems, large datasets must be split across multiple machines called shards. We model this as a graph $G = (V, E)$, where vertices represent data records and edges connect records that are accessed together by the same query. Since involving multiple shards in a single query increases latency and processing time, we want to assign records evenly across shards while minimizing the number of edges crossing partition boundaries. This is precisely the Balanced Graph Partitioning problem.

**Maximum Independent Set (Chaudhry et al., 2013):** In a WiFi network, routers must be assigned to broadcast on one of a limited number of frequency channels. If two nearby routers share the same channel simultaneously, their signals interfere and degrade the connection quality. We model the network as a graph $G = (V, E)$, where vertices represent routers and edges connect pairs that are close enough to interfere with each other. Finding the largest set of routers that can safely share the same channel simultaneously is precisely the Maximum Independent Set problem.

**Maximum Clique (Bomze et al., 2026):** In data center management, servers are interconnected through a high-speed network, which could be a physical cable. To sustain critical operations under potential disruptions, we want to identify the largest group of servers that are all directly connected to one another, ensuring that every pair in the group can communicate without relying on any intermediate server. We model the data center as a graph $G = (V, E)$, where vertices represent servers and edges represent direct high-speed connections. Finding the largest such group is precisely the Maximum Clique problem.

**Minimum Dominating Set (Sun et al., 2019):** In wireless sensor networks, sensors are randomly distributed across a region and communicate either directly or through intermediate nodes. Since sensors run on limited battery power, we want to select the smallest subset of

sensors to remain active such that every inactive sensor is within communication range of at least one active sensor, allowing the remaining sensors to sleep and conserve energy. We model the network as a graph $G = (V, E)$, where vertices represent sensors and edges connect pairs of sensors within communication range of each other. Finding the smallest such subset of active sensors is precisely the Minimum Dominating Set problem.

## 5.3 Future Work

Several directions remain open for extending this work. On the theoretical side, it would be interesting to investigate whether the dominance result of Theorem 3.3.2.1 holds for shift operators beyond the lazy random walk matrix, such as the unnormalised or symmetric normalised Laplacian. It is not obvious that it should, and the answer either way would be informative.

On the empirical side, the modular structure of GCON makes it a natural fit for additional CO problems such as Minimum Clique Cover a nd Graph Coloring. Our MIS results also point to loss design as a direction worth pursuing more carefully. The authors of Wenkel et al. (2024) have noted in personal correspondence that QUBO formulations tend to induce a more favourable optimization landscape than the losses used in the original paper, and understanding precisely why this is the case strikes us as both an interesting and practically useful question.

For Graph Partitioning, a natural extension would be to move beyond unweighted graphs and consider settings where edges carry weights reflecting the cost of cutting a connection. Finally, incorporating richer node features beyond the structural statistics used here could improve the expressive power of the model more broadly, and seems like a direction worth exploring.

# References


Bai, Y., D. Xu, A. Wang, K. Gu, X. Wu, A. Marinovic, C. Ro, Y. Sun, and W. Wang (2020). Fast detection of maximum common subgraph via deep q-learning. *arXiv preprint arXiv:2002.03129 16*, 1235–1243.

Barabási, A.-L. and R. Albert (1999). Emergence of scaling in random networks. *science 286*(5439), 509–512.

Bello, I., H. Pham, Q. V. Le, M. Norouzi, and S. Bengio (2016). Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940*.

Bomze, I., C. Faccio, F. Rinaldi, and G. Spisso (2026). The maximum clique problem under adversarial uncertainty: a min-max approach. *arXiv preprint arXiv:2601.05644*.

Cantürk, S. et al. (2026). COPT-MT source code. https://github.com/semihcanturk/COPT-MT. Accessed: May 2026.

Cappart, Q., D. Chételat, E. Khalil, A. Lodi, C. Morris, and P. Veličković (2022). Combinatorial optimization and reasoning with graph neural networks.

Çatalyürek, Ü., K. Devine, M. Faraj, L. Gottesbüren, T. Heuer, H. Meyerhenke, P. Sanders, S. Schlag, C. Schulz, D. Seemaier, et al. (2023). More recent advances in (hyper) graph partitioning. *ACM Computing Surveys 55*(12), 1–38.

Chaudhry, A. U., J. W. Chinneck, and R. H. Hafez (2013). On the number of channels required for interference-free wireless mesh networks. *EURASIP Journal on Wireless Communications and Networking 2013*(1), 229.

Daigavane, A., B. Ravindran, and G. Aggarwal (2021). Understanding convolutions on graphs. *Distill*. https://distill.pub/2021/understanding-gnns.

Deudon, M., P. Cournut, A. Lacoste, Y. Adulyasak, and L.-M. Rousseau (2018). Learning heuristics for the tsp by policy gradient. In *International conference on the integration of constraint programming, artificial intelligence, and operations research*, pp. 170–181. Springer.

Garey, M. R. and D. S. Johnson (2002). *Computers and intractability*, Volume 29. wh freeman New York.

Gilmer, J., S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl (2017). Neural message passing for quantum chemistry. In *International conference on machine learning*, pp. 1263–1272. Pmlr.

Hamilton, W. L. (2020). *Graph Representation Learning*, Volume 14 of *Synthesis Lectures on Artificial Intelligence and Machine Learning*. Morgan & Claypool Publishers.

Hinton, G., L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, et al. (2012). Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal processing magazine 29*(6), 82–97.

Holland, P. W., K. B. Laskey, and S. Leinhardt (1983). Stochastic blockmodels: First steps. *Social networks 5*(2), 109–137.

Hopfield, J. J. and D. W. Tank (1985). "Neural" Computation of Decisions in Optimization Problems. *Biological cybernetics 52*(3), 141–152.

Karalias, N. and A. Loukas (2020). Erdos Goes Neural: an Unsupervised Learning Framework for Combinatorial Optimization on Graphs. *CoRR abs/2006.10643*.

Karp, R. M. (2009). Reducibility among combinatorial problems. In *50 Years of Integer Programming 1958-2008: from the Early Years to the State-of-the-Art*, pp. 219–241. Springer.

Khalil, E., H. Dai, Y. Zhang, B. Dilkina, and L. Song (2017). Learning combinatorial optimization algorithms over graphs. *Advances in neural information processing systems 30*.

Kool, W., H. Van Hoof, and M. Welling (2018). Attention, learn to solve routing problems! *arXiv preprint arXiv:1803.08475*.

Krizhevsky, A., I. Sutskever, and G. E. Hinton (2012). Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems 25*.

Li, Z., Q. Chen, and V. Koltun (2018). Combinatorial optimization with graph convolutional networks and guided tree search. *Advances in neural information processing systems 31*.

Min, Y., F. Wenkel, M. Perlmutter, and G. Wolf (2022). Can hybrid geometric scattering networks help solve the maximum clique problem? *Advances in Neural Information Processing Systems 35*, 22713–22724.

Nikishin, E., P. Izmailov, B. Athiwaratkun, D. Podoprikhin, T. Garipov, P. Shvechikov, D. Vetrov, and A. G. Wilson (2018). Improving stability in deep reinforcement learning with weight averaging. In *Uncertainty in artificial intelligence workshop on uncertainty in Deep learning*.

Papadimitriou, C. H. and K. Steiglitz (1998). *Combinatorial Optimization: Algorithms and Complexity*. Courier Corporation.

Selsam, D., M. Lamm, B. Bünz, P. Liang, L. de Moura, and D. L. Dill (2018). Learning a sat solver from single-bit supervision. *arXiv preprint arXiv:1802.03685*.

Sun, H., E. K. Guha, and H. Dai (2022). Annealed training for combinatorial optimization on graphs. *arXiv preprint arXiv:2207.11542*.

Sun, X., Y. Yang, and M. Ma (2019). Minimum connected dominating set algorithms for ad hoc sensor networks. *Sensors 19*(8), 1919.

Vaswani, A., N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin (2017). Attention is all you need. *Advances in neural information processing systems 30*.

Veličković, P., G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio (2018). Graph attention networks.

Wenkel, F., S. Cantürk, S. Horoi, M. Perlmutter, and G. Wolf (2024). Towards a General Recipe for Combinatorial Optimization with Multi-Filter GNNs. *arXiv preprint arXiv:2405.20543*.

Xu, K., F. Boussemart, F. Hemery, and C. Lecoutre (2007). Random constraint satisfaction: Easy generation of hard (satisfiable) instances. *Artificial Intelligence 171*(8), 514–534.

Yolcu, E. and B. Póczos (2019). Learning local search heuristics for boolean satisfiability. *Advances in Neural Information Processing Systems 32*.

Zhang, D., H. Dai, N. Malkin, A. C. Courville, Y. Bengio, and L. Pan (2023). Let the flows tell: Solving graph combinatorial problems with gflownets. *Advances in neural information processing systems 36*, 11952–11969.

# Appendix A

## A.1 Constrained Optimization Formulations

**Maximum Cut** (Wenkel et al., 2024, Appendix E)

$$\begin{aligned} \text{Maximize} \quad & \sum_{(i,j)\in E} x_i + x_j - 2x_i x_j \\ \text{subject to} \quad & x_i \in \{0,1\} \quad \forall i \in V \end{aligned}$$

**Maximum Clique** (Wenkel et al., 2024, Appendix E)

$$\begin{aligned} \text{Maximize} \quad & \sum_{i\in V} x_i \\ \text{subject to} \quad & x_i + x_j \leq 1 \quad \forall (i,j) \notin E, \\ & x_i \in \{0,1\} \quad \forall i \in V \end{aligned}$$

**Minimum Dominating Set** (Wenkel et al., 2024, Appendix E)

$$\begin{aligned} \text{Minimize} \quad & \sum_{i\in V} x_i \\ \text{subject to} \quad & x_i + \sum_{j\in\mathcal{N}(i)} x_j \geq 1 \quad \forall i \in V, \\ & x_i \in \{0,1\} \quad \forall i \in V \end{aligned}$$

**Maximum Independent Set**

$$\begin{aligned} \text{Maximize} \quad & \sum_{i\in V} x_i \\ \text{subject to} \quad & x_i + x_j \leq 1 \quad \forall (i,j) \in E, \\ & x_i \in \{0,1\} \quad \forall i \in V. \end{aligned}$$

**Balanced Graph Partitioning**

$$\begin{aligned} \text{Minimize} \quad & \sum_{t=1}^{k} \sum_{(i,j)\in E} (x_i^t - x_j^t)^2 \\ \text{subject to} \quad & \sum_{t=1}^{k} x_i^t = 1 \quad \forall i \in V \\ \text{and} \quad & \sum_{i=1}^{n} x_i^t = \frac{n}{k} \quad \text{for } 1 \leq t \leq k \end{aligned}$$

## A.2 Baseline Heuristics

**Algorithm 2** Greedy Balanced Bisection for Graph Partitioning

**Require:** Graph $G = (V, E)$
**Ensure:** Partition labels labels : $V \to \{0, 1\}$
labels$[v] \leftarrow -1$ for all $v \in V$ ▷ $-1$ denotes unassigned
$c_0 \leftarrow 0$, $c_1 \leftarrow 0$ ▷ partition size counters
cap $\leftarrow \lceil |V|/2 \rceil$ ▷ maximum size per partition
sort $V$ in descending order of degree
**for** each $v \in V$ in sorted order **do**
    $e_0 \leftarrow |\{u \in \mathcal{N}(v) : \text{labels}[u] = 0\}|$
    $e_1 \leftarrow |\{u \in \mathcal{N}(v) : \text{labels}[u] = 1\}|$
    **if** $c_0 \geq$ cap **then** ▷ partition $0$ is full
        $p \leftarrow 1$
    **else if** $c_1 \geq$ cap **then** ▷ partition $1$ is full
        $p \leftarrow 0$
    **else if** $e_1 > e_0$ **then** ▷ join the side with more assigned neighbors
        $p \leftarrow 1$
    **else**
        $p \leftarrow 0$
    **end if**
    labels$[v] \leftarrow p$
    $c_p \leftarrow c_p + 1$
**end for**
**return** labels

**Algorithm 3** Spectral Graph Partitioning

**Require:** Adjacency matrix $A \in \{0, 1\}^{n \times n}$, number of partitions $k$
**Ensure:** A $k$-way partition $\{P_1, \ldots, P_k\}$ of the vertices $V$
Compute degree matrix $D$ where $D_{ii} = \sum_j A_{ij}$
Compute graph Laplacian $L \leftarrow D - A$
Compute the $k$ smallest eigenvectors $u_1, u_2, \ldots, u_k$ of $L$ ▷ Discard $u_1$ (constant vector); use $u_2, \ldots, u_k$ for $k > 2$
Form embedding matrix $U \in \mathbb{R}^{n \times (k-1)}$ with columns $u_2, \ldots, u_k$
Run $k$-means clustering on the rows of $U$ to obtain $k$ clusters
**return** partition $\{P_1, \ldots, P_k\}$ corresponding to the $k$ clusters

**Algorithm 4** GMIN Greedy Heuristic for Maximum Independent Set

**Require:** Graph $G = (V, E)$
**Ensure:** Independent set $\mathcal{I} \subseteq V$
$\mathcal{I} \leftarrow \emptyset$
$R \leftarrow V$ ▷ $R$ tracks the remaining nodes
**while** $R \neq \emptyset$ **do**
  $v^* \leftarrow \arg\min_{v \in R} \deg_R(v)$ ▷ select the minimum degree node in the remaining graph
  $\mathcal{I} \leftarrow \mathcal{I} \cup \{v^*\}$
  $R \leftarrow R \setminus (\{v^*\} \cup \mathcal{N}(v^*))$ ▷ remove $v^*$ and all its neighbors
**end while**
**return** $\mathcal{I}$

## A.3 Graph Generating Models

We describe each of the graph generating models used to generate hard instances for the problems in this work.

1. **Barabási-Albert (BA) graphs**: The BA graphs are generated with the preferential attachment mechanism proposed by Barabási and Albert (1999). The user only has to specify a single parameter $m$, and the operation starts from a base graph of $m_0 \geq m$ nodes. At each iteration a new node is added and linked to $m$ existing nodes, with each existing node $i$ being chosen proportional to its current degree $k_i$. This produces scale-free graphs with a heavy-tailed degree distribution. Wenkel et al. (2024) use BA graphs for the Maximum Cut and Minimum Dominating Set problems.

2. **RB graphs**: RB graphs are derived from a random constraint satisfaction model Xu et al. (2007). Given $n$ variables, the model randomly places $m = rn \ln n$ constraints between pairs of variables, where $r > 0$ is a parameter controlling the number of constraints. The resulting constraint graph has $n$ nodes and $m$ edges. A key property of the model is that instances generated near the phase transition threshold are provably hard. We use RB graphs for the Maximum Clique and Maximum Independent Set problems, following Wenkel et al. (2024).

3. **Stochastic Block Model (SBM) graphs**: SBM graphs are generated according to the model of Holland et al. (1983). Given $n$ vertices partitioned into $k$ disjoint blocks (communities) of approximately balanced size, each pair of vertices $(i, j)$ is connected independently with probability $p_{\text{in}}$ if $i$ and $j$ belong to the same block, and with probability $p_{\text{out}}$ otherwise. Choosing $p_{\text{in}} > p_{\text{out}}$ yields graphs with an inherent community structure, where edges concentrate within blocks rather than across them. This property makes SBM a natural generative model for evaluating methods on the Graph Partitioning problem, as the planted block assignment provides a meaningful ground-truth partition. We use SBM graphs for the Graph Partitioning problem.

## A.4 Proof of Theorem 3.3.2.1

We restate the theorem before we give its proof:

**Theorem A.4.0.1** ((Wenkel et al., 2024, sec. 5.3)). *Take the non-decoupled architecture and suppose the representation $(\bar{\mathbf{s}}_f[v], \mathbf{h}_f[v])_{f\in\mathcal{F}}$ at a node $v \in V$ is $c$-band dominant for a non-negative input signal $\mathbf{x} \in \mathbb{R}^n$ with $x_i \geq 0$ for every $i$, where $\bar{\mathbf{s}}_f[v]$ is the normalized attention score of $f \in \mathcal{F}$ at node $v$ and $\mathbf{h}_f[v]$ is the result of the filter $f \in \mathcal{F}$ at node $v$. Then, provided every filter is run at a sufficiently large scale, the aggregation filter responses always dominate the comparison filter responses, in the sense that*

$$\sum_{f\in\mathcal{F}_C} \bar{\mathbf{s}}_f[v]\,\mathbf{h}_f[v] \;\leq\; \epsilon \cdot c \cdot \sum_{f\in\mathcal{F}_A} \bar{\mathbf{s}}_f[v]\,\mathbf{h}_f[v]. \tag{A.4.1}$$

*Proof (Wenkel et al., 2024, Appendix A.1).* Consider a graph $G = (V, E)$ on $n$ vertices. Let $\epsilon > 0$ and fix a node $v \in V$. Suppose that the representation $(\bar{\mathbf{s}}_f[v], \mathbf{h}_f[v])_{f\in\mathcal{F}}$ is $c-$band dominant.

For simplicity, consider the feature vector $\mathbf{x} \in \mathbb{R}^n$ and the lazy random walk $\mathbf{P} = \frac{1}{2}(\mathbf{I} + AD^{-1})$ on $G$. Let $\mathbf{d} \in \mathbb{R}^n$ be the degree vector and define

$$p_\infty := \|\mathbf{x}\|_1 \frac{\mathbf{d}[v]}{\|\mathbf{d}\|_1}, \tag{A.4.2}$$

as the stationary value of the transition matrix at $v$.

Since $G$ is a finite and connected graph, the random walk on $G$ is irreducible. Additionally, since $\mathbf{P}$ is a lazy random walk, with every state having a self-loop probability of $\frac{1}{2}$, then $G$ is aperiodic.

Since $G$ is finite, irreducible, and aperiodic Markov chain then $G$ is ergodic hence by Theorem 2.1.3.19 and the fact that $\mathbf{P}$ is column-stochastic there exists a unique stationary distribution $\pi$, which is a column vector, such that for any initial distribution $\lambda$,

$$\lim_{n\to\infty} \mathbf{P}^n\lambda = \pi. \tag{A.4.3}$$

We claim that for $G$, $\dfrac{\mathbf{d}}{\|\mathbf{d}\|_1}$ is the unique stationary distribution to which the chain converges. To prove this, it suffices to show that $\dfrac{\mathbf{d}}{\|\mathbf{d}\|_1}$ is indeed a stationary distribution.

We have

$$\begin{aligned}
\mathbf{P} \cdot \frac{\mathbf{d}}{\|\mathbf{d}\|_1} &= \frac{1}{2}(\mathbf{I} + AD^{-1})\frac{\mathbf{d}}{\|\mathbf{d}\|_1} \\
&= \frac{1}{2\|\mathbf{d}\|_1}(\mathbf{d} + AD^{-1}\mathbf{d}) \\
&= \frac{1}{2\|\mathbf{d}\|_1}(\mathbf{d} + A\mathbf{1})
\end{aligned}$$

$$= \frac{1}{2\|\mathbf{d}\|_1}(\mathbf{d}+\mathbf{d})$$
$$= \frac{\mathbf{d}}{\|\mathbf{d}\|_1}$$

where $\mathbf{1}$ is the all-ones vector.

Thus we have that for some $F_k \in \mathcal{F}_A$ and $F_{k_1,k_2} \in \mathcal{F}_C$,

$$F_k(\mathbf{x}) := \mathbf{P}^k\mathbf{x} \to \|\mathbf{x}\|_1 \frac{\mathbf{d}}{\|\mathbf{d}\|_1} \tag{A.4.4}$$

and

$$F_{k_1,k_2}(\mathbf{x}) := \left(\mathbf{P}^{k_1} - \mathbf{P}^{k_2}\right)\mathbf{x} \to 0. \tag{A.4.5}$$

Hence for any $\delta > 0$, there exists $K := K(\delta)$ such that all filters $f$ with scale $k \geq K$ satisfy:

$$|\mathbf{h}_f[v] - p_\infty| \leq \delta \quad \text{if } f \in \mathcal{F}_A, \tag{A.4.6}$$

and

$$|\mathbf{h}_f[v]| \leq \delta \quad \text{if } f \in \mathcal{F}_C, \tag{A.4.7}$$

where $\mathbf{h}_f[v] := f(\mathbf{x})[v]$ is the response of the filter $f \in \mathcal{F}$ at $v \in V$.

Set $\tau := \max\{\bar{\mathbf{s}}_f[v] : f \in \mathcal{F}_C\}$, where $\bar{\mathbf{s}}_f[v]$ is the normalized attention score of $f \in \mathcal{F}$ at $v \in V$. Then since each comparison filter satisfies $|\mathbf{h}_f[v]| \leq \delta$ we have:

$$\sum_{f\in\mathcal{F}_C} \bar{\mathbf{s}}_f[v]\,\mathbf{h}_f[v] \;\leq\; \sum_{f\in\mathcal{F}_C} \bar{\mathbf{s}}_f[v]\,\delta \;\leq\; \sum_{f\in\mathcal{F}_C} \tau\,\delta \;\leq\; |\mathcal{F}_C|\,\tau\delta. \tag{A.4.8}$$

Let $f_A^{\max}$ be the aggregation filter achieving the largest importance $\bar{\mathbf{s}}_f[v]$ at node $v$. Then since, by hypothesis, the representation at $v$ is $c$-band dominant and the input signal $\mathbf{x}$ is non-negative, and $\mathbf{h}_{f_A^{\max}}[v] \geq p_\infty - \delta$ from, Equation A.4.5, then

$$\epsilon \cdot c \cdot \sum_{f\in\mathcal{F}_A} \bar{\mathbf{s}}_f[v]\mathbf{h}_f[v] \geq \epsilon \cdot c \cdot \bar{\mathbf{s}}_{f_A^{\max}}\mathbf{h}_{f_A^{\max}}[v] \geq \epsilon \cdot \tau\,(p_\infty - \delta). \tag{A.4.9}$$

Notice that for Equation A.4.1 to hold then

$$|\mathcal{F}_C| \leq \frac{p_\infty - \delta}{\delta}\,\epsilon. \tag{A.4.10}$$

Since $g(\delta) := \dfrac{(p_\infty - \delta)}{\delta} \to \infty$ as $\delta \to 0$, then it is possible to choose $K := K(\delta)$ large enough that makes $\delta$ small enough that Equation A.4.10 is satisfied.

□

## A.5 Graph Statistics

Let $G = (V, E)$ be a simple, connected, undirected graph with adjacency matrix $A$, and let $d$ denote the shortest path distance, which is the minimum number of edges on any path connecting any two nodes, on $V$. For each node $v \in V$, we define the following graph statistics.

The **eccentricity** of $v$ is the maximum shortest path distance from $v$ to any other node:

$$\varepsilon(v) := \max_{u \in V} d(u, v). \tag{A.5.1}$$

The **triangle count** of $v$ is the number of triangles in $G$ that contain $v$:

$$t(v) := \frac{1}{2} \sum_{u,w \in N(v)} A_{uw}. \tag{A.5.2}$$

The **clustering coefficient** of $v$ measures the fraction of pairs of neighbors of $v$ that are themselves connected:

$$C(v) := \frac{t(v)}{\binom{\deg(v)}{2}}, \tag{A.5.3}$$

where $\binom{\deg(v)}{2}$ is the number of pairs of neighbors of $v$. We set $C(v) = 0$ when $\deg(v) \leq 1$. The clustering coefficient can be thought of as the probability that any two neighbors of $v$ are adjacent.