Towards a General Recipe for Combinatorial Optimization with Multi-Filter GNNs

Frederik Wenkel*
Dept. of Mathematics & Statistics
Université de Montréal
Mila – Quebec AI Institute
frederik.wenkel@umontreal.ca

Semih Cantürk*
Dept. of Computer Science and Operations Research
Université de Montréal
Mila – Quebec AI Institute
semih.canturk@umontreal.ca

Stefan Horoi
Dept. of Mathematics & Statistics
Université de Montréal
Mila – Quebec AI Institute

Michael Perlmutter†
Dept. of Mathematics
Boise State University

Guy Wolf†
Dept. of Mathematics & Statistics
Université de Montréal
Montreal, QC, Canada

Abstract

Graph neural networks (GNNs) have achieved great success for a variety of tasks such as node classification, graph classification, and link prediction. However, the use of GNNs (and machine learning more generally) to solve combinatorial optimization (CO) problems is much less explored. Here, we introduce GCON, a novel GNN architecture that leverages a complex filter bank and localized attention mechanisms to solve CO problems on graphs. We show how our method differentiates itself from prior GNN-based CO solvers and how it can be effectively applied to the maximum clique, minimum dominating set, and maximum cut problems in a self-supervised learning setting. GCON is competitive across all tasks and consistently outperforms other specialized GNN-based approaches, and is on par with the powerful Gurobi solver on the max-cut problem. We provide an open-source implementation of our work[1].

1 Introduction

Recent years have seen the rapid development of neural networks for graph-structured data [37, 1]. Indeed, graph neural networks (GNNs) achieve state-of-the-art performance for tasks such as node classification and are incorporated in industrial applications such as helping power Google Maps [10] and Amazon's recommender system [35]. Popular message-passing neural networks (MPNNs) typically consider a graph together with a collection of features associated with each node [22, 39, 34]. In each layer, they first perform a local aggregation which effectively smooths the features over neighboring nodes of the graph, and then perform a transformation which learns combinations of the smoothed node features.

Here, we consider leveraging GNNs to solve combinatorial optimization (CO) problems, a somewhat less explored yet emerging application area of graph learning. In particular, we focus on finding (i) the maximum clique, i.e., the largest fully connected subgraph, (ii) the minimum dominating set, i.e., the smallest subset of the vertices which "dominates" the graph in the sense that every vertex is within at most one hop of the dominating set, and (iii) the maximum cut, i.e., a partition of the vertices into two clusters with as many cross-cluster edges as possible.

Notably, these tasks differ significantly from, e.g., node classification in several important ways. The aggregations used in common message passing neural networks are localized averaging operations

*Equal contribution.
†Equal senior author contributions.
[1] https://github.com/WenkelF/copt

F. Wenkel et al., Towards a General Recipe for Combinatorial Optimization with Multi-Filter GNNs. *Proceedings of the Third Learning on Graphs Conference (LoG 2024)*, PMLR 269, Virtual Event, November 26–29, 2024.

that can be interpreted, from the perspective of graph signal processing (GSP) [26, 29], as low-pass filtering. In this manner, they effectively treat smoothness as an inductive bias. While this is a useful heuristic on homophilic social networks such as the ubiquitous Cora, CiteSeer, and PubMed datasets [7], we find that this assumption is not quite appropriate for CO problems. Moreover, in CO problems, it is not reasonable to assume the presence of informative node features. Instead, algorithms are expected to learn directly from the network geometry (taking initial inputs to be constant or features derived from the graph structure, e.g., vertex degree). We also note that the CO problems we are interested in are typically NP-hard. Therefore, it is computationally intractable to obtain sufficient labeled data for supervised learning. As we shall see below, our approach tackles various CO problems with a fully self-supervised framework, using task-specific loss functions.

To address these challenges, we propose the Graph Combinatorial Optimization Network (GCON), a novel GNN that uses a hybrid filter bank consisting of both (i) aggregation operations such as those used in common MPNNs, as well as (ii) comparison operations, which will take the form of band-pass, wavelet filters and aim to capture more intricate aspects of the network geometry. In order to balance the importance of these different filters, we use a localized attention mechanism which may choose to lend more importance to different filters on a node-by-node basis. We compare GCON to various competing methods, such as heuristics, commercial solvers, other specialized GNN-based approaches [18, 25], and recent advances that apply GFlowNets (GFNs) to the above mentioned problems [41]. Our main contributions are:

1. We develop GCON, a novel GNN architecture with a sophisticated filter bank and a localized attention mechanism that is particularly suited for solving CO problems.
2. We evaluate GCON on a variety of graph-CO problems on synthetic graph benchmarks, and demonstrate that our architecture is highly performant.
3. We prove a theorem that helps us interpret the improved performance of our method.

2 Background and Related Work

Many well-known graph-CO problems are either NP-hard or even NP-complete [19], rendering it impossible to efficiently generate exact solutions for large graphs. Instead, one may hope to efficiently generate *approximate solutions*. Almost forty years after Hopfield and Tank [17] first proposed applying deep neural networks to CO problems, there has been a recent revival in achieving this goal via GNNs. Indeed, GNNs are a natural candidate due to their success on a wide variety of tasks such as node classification, graph classification, and link prediction [1, 37]; a comprehensive overview can be found in Cappart et al. [8]. Researchers have explored ways to solve graph-CO problems using both supervised and self-supervised methods as well as reinforcement learning (RL). Unfortunately, supervised methods are often computationally infeasible due to the cost of labeling for NP-hard problems. RL-based solutions, on the other hand, may run into problems arising from the large size of the state space. We thus view self-supervised learning as the most promising family of approaches.

Notably, Tönshoff et al. [32], Karalias and Loukas [18], Min et al. [25] have all used GNNs to solve CO problems in a self-supervised manner. Tönshoff et al. [32] solves binary constraint satisfaction problems using message-passing with recurrent states. Subsequently, Karalias and Loukas [18] proposed a seminal self-supervised learning framework in which a GNN outputs a Bernoulli distribution over the vertex set of the input graph, and a surrogate loss function designed to encourage certain properties in the distribution (e.g., vertices that are assigned high probabilities form a clique) and softly enforce constraints in a differentiable manner. They then propose specific loss instantiations to tackle maximum clique and graph partitioning problems. A similar loss is derived in Min et al. [25], who use an architecture based on the graph scattering transform [14] to obtain promising results for the maximum clique problem. A notable difficulty associated with these methods is that the proposed self-supervised loss functions are highly non-convex. This motivated a separate line of work by Sun et al. [30] which designed an annealed optimization framework that aims to optimize GNNs effectively for such loss functions. We also note the recent work by Zhang et al. [41], which effectively applied generative flow networks [3] to solve CO problems in an iterative manner. Our work is most closely related to Karalias and Loukas [18] and Min et al. [25], relying on the same or closely related self-supervised loss functions, while improving on the learned part of the pipeline, leveraging a more powerful hybrid GNN architecture.

Finally, GCON is related to several lines of work on building neural solvers that can perform well across graph-CO problems. Tönshoff et al. [33] and Boisvert et al. [6] propose building unified representations for constraint satisfaction problems (CSP), Berto et al. [4] and Zhou et al. [42] similarly propose unified methods for vehicle routing problem (VRP) variants, while Drakulic et al. [11] proposes a modular architecture for multi-task learning over a variety of CO problems.

3 Problem Setup

The purpose of this paper is to develop methods for (approximately) solving combinatorial optimization problems on graphs in a scalable and computationally efficient manner.

We let $G = (V, E)$ denote a simple, connected, undirected graph. We assume an arbitrary (but fixed) ordering on the vertices $V = \{v_1, \ldots, v_n\}$. In a slight abuse of notation, if $\mathbf{x}$ is a function defined on the vertices V, we do not distinguish between $\mathbf{x}$ and the vector defined by $x_i = \mathbf{x}[v_i]$. We let $\mathbf{A} \in \mathbb{R}^{n \times n}$ be the adjacency matrix of G, let $\mathbf{D}$ be the corresponding diagonal degree matrix, $\mathbf{D} = \text{diag}(\mathbf{A}\,\mathbf{1})$, and let $\mathbf{P} = \frac{1}{2}(\mathbf{I} + \mathbf{A}\,\mathbf{D}^{-1})$ denote the transition matrix of a lazy random walk (a Markov chain in which, at each step, the walker either stays put or moves to a neighboring vertex, each with probability one half). Finally, we let $\mathbf{X} \in \mathbb{R}^{n \times d}$ denote a matrix of input node features.

We consider the following CO problems in this work, but also note that our framework could be extended to a large variety of graph-CO problems such as vertex cover, maximal independent set or graph coloring by constructing appropriate supervised or self-supervised loss functions:

1. **Maximum Cut:** Find a partition of the vertex set $V = S \sqcup T$ that maximizes the number of edges between S and T, i.e., $|\{\{v_i, v_j\} \in E : v_i \in S, v_j \in T\}|$.
2. **Maximum Clique:** Find a fully connected subgraph (a clique) with as many vertices as possible.
3. **Minimum Dominating Set:** A subset of the vertices $S \subseteq V$ is called a *dominating set* if for every vertex v_i is either an element of S or a neighbor of S, in the sense that $\{v_i, v_j\} \in E$ for some $v_j \in S$. Find a dominating set with as few vertices as possible.

4 Methodology

We treat each of the CO problems as a self-supervised learning problem. We first compute basic node statistics to serve as input features to the graph. We then use our GNN, described in detail in Section 5, to generate a function/vector $\mathbf{p}$ whose i-th entry $\mathbf{p}_i = \mathbf{p}[v_i]$ is interpreted as the probability that v_i is in the set of interest. We apply a sigmoid in its final layer ensuring that all entries are between 0 and 1. Finally, we use a *rule-based decoder* to construct a candidate solution by subsequently adding nodes ordered by decreasing probability, based on $\mathbf{p}$. Our methodology is illustrated in Fig. 1. In the following subsections, we will explain how to develop suitable loss functions and decoders for each problem of interest.

4.1 Maximum Cut

In this problem, we aim to find the maximum cut $V = S \sqcup T$, and without loss of generality, we choose p_i to be the probability that $v_i \in S$. We then define $\mathbf{y} \coloneqq 2\,\mathbf{p} - \mathbf{1}_n$ (where $\mathbf{1}_n \in \mathbb{R}^n$ is the vector of all 1's) so that the entries of $\mathbf{y}$ lie in $[-1, 1]$ (rather than $[0, 1]$) and use the loss function

$$L(\mathbf{y}) \coloneqq \tfrac{1}{2}\,\mathbf{y}^\top\,\mathbf{A}\,\mathbf{y} = \textstyle\sum_{\{v_i, v_j\} \in E} y_i y_j.$$

After learning $\mathbf{y}$, we then set S to be the set of vertices where $\mathbf{y}$ is greater than or equal to zero and T to be the set of vertices where $\mathbf{y}$ is negative. Intuitively, we note that every edge that connects two nodes in different parts of the cut, i.e., $\text{sgn}(\mathbf{y}_i) = -\,\text{sgn}(\mathbf{y}_j)$, decreases the loss, while the opposite case yields an additive term that increases the loss. For example, if $\mathbf{p} = \mathbf{1}_S$ (where $\mathbf{1}_S$ is the indicator function on $S \subset V$, i.e, $\mathbf{1}_S[v] = 1$ if $v \in S$, and zero otherwise), we have $\mathbf{y} = \mathbf{1}_S - \mathbf{1}_T$, and our loss function is equal to the number of cuts minus the number of intra-cluster edges. Thus, the loss function encourages a node labeling that cuts as many edges as possible.

4.2 Maximum Clique

For the maximum clique, we let $\overline{\mathbf{A}}$ denote the adjacency matrix of the complement graph $\overline{G} = (V, E^c)$, where E^c is the set of all $\{\{v_i, v_j\} \notin E,\, i \neq j\}$ and use the two-part loss function considered in

Min et al. [25] (see also Karalias and Loukas [18]):

$$L(\mathbf{p}) \coloneqq L_1(\mathbf{p}) + \beta L_2(\mathbf{p}) = -\,\mathbf{p}^\top\,\mathbf{A}\,\mathbf{p} + \beta\,\mathbf{p}^\top\,\overline{\mathbf{A}}\,\mathbf{p}\,.$$

To understand this loss function, consider the idealized case where $\mathbf{p} = \mathbf{1}_S$ is the indicator function of some set S. In that case, one may verify that

$$-L_1(\mathbf{1}_S) = \mathbf{1}_S^\top\,\mathbf{A}\,\mathbf{1}_S = \sum_{i,j} \mathbf{A}[i,j]\mathbf{1}_S[v_i]\mathbf{1}_S[v_j]$$

is the number of edges between elements of S. Similarly, we see have that

$$L_2(\mathbf{1}_S) = \mathbf{1}_S^\top\overline{\mathbf{A}}\mathbf{1}_S = \sum_{i,j} \overline{\mathbf{A}}[i,j]\mathbf{1}_S[v_i]\mathbf{1}_S[v_j]$$

is the number of "missing edges" within S, i.e., the number of $\{v_i, v_j\} \in E^c$, $v_i, v_j \in S$. Therefore, in minimizing $L(\mathbf{p})$, we aim to find a subset with as many connections, and as few missing connections, as possible. The hyper-parameter β is used to balance the contributions of L_1 and L_2.

After computing $\mathbf{p}$, we then reorder the vertices $\{v_i\}_{i=1}^n \to \{v_i'\}_{i=1}^n$ so in the new ordering $\mathbf{p}(v_i') \geq \mathbf{p}(v_{i+1}')$. We then build an initial clique by initializing $C^{(1)} = \{v_1'\}$ and iterating over i. At each step, if $C^{(1)} \cup \{v_i'\}$ forms a clique, we add v_i' to $C^{(1)}$, otherwise we do nothing. Note that $C^{(1)}$ is guaranteed to be a clique by construction.

We then repeat this process multiple times to construct additional cliques $C^{(2)}, \ldots, C^{(K)}$. However, on the k-th iteration, we initialize with $C^{(k)} = \{v_k'\}$ and automatically exclude $v_1', \ldots, v_{k-1}'$ from $C^{(k)}$. We then choose the maximal clique C^* to be the $C^{(k)}$ with the largest cardinality. We note that increasing K does increase the computational cost of our decoder; however, running the decoder with different initializations $\{v_k'\}$ can be parallelized in a straightforward manner.

4.3 Minimum Dominating Set

For the minimum dominating set problem, we use the loss function:

$$L(\mathbf{p}) \coloneqq L_1(\mathbf{p}) + \beta L_2(\mathbf{p}) \coloneqq \|\,\mathbf{p}\,\|_1 + \beta \sum_{v_i \in V} \exp\left(\sum_{v_j \in \mathcal{N}_{\underline{v_i}}} \log(1 - \mathbf{p}[v_j])\right),$$

where $\mathcal{N}_{\underline{v}}$ denotes the set of nodes within distance one of v (including v itself). The first loss term promotes the sparsity of the dominating set. To understand the second term, observe as soon as there is at least one $v_j \in \mathcal{N}_{\underline{v_i}}$ with $\mathbf{p}[v_j] \approx 1$ we will have $\exp\left(\sum_{v_j \in \mathcal{N}_{\underline{v_i}}} \log(1 - \mathbf{p}[v_j])\right) \approx 0$. Indeed, we note that, if $\mathbf{p} = \mathbf{1}_S$ is the indicator function of a dominating set S, we have that $\exp\left(\sum_{v_j \in \mathcal{N}_{\underline{v_i}}} \log(1 - \mathbf{p}[v_j])\right) = \exp(-\infty) = 0$, for all v_i. Therefore, $L(\mathbf{1}_S) = \|\mathbf{1}_S\|_1$ would be exactly the cardinality of S.

After learning $\mathbf{p}$, we construct our proposed minimum dominating set in a manner analogous to the max-clique case. We again begin by reordering the vertices so that $\mathbf{p}(v_i') \geq \mathbf{p}(v_{i+1}')$. We then initialize our first dominating set $S^{(1)} = \{v_1'\}$ and iterate over i. At each step, we check if S is a dominating set. If so, we stop. Otherwise, we add v_{i+1}' to $S^{(1)}$ and repeat the process. Similarly to the maximum clique, we repeat this process several times, where we initialize $S^{(k)} = \{v_k'\}$ (automatically excluding $v_1', \ldots, v_{k-1'}$ from the set $S^{(k)}$ considered on the k-th iteration) and take S^* to be the smallest dominating set $S^{(k)}$ found in any iteration.

5 The GCON Architecture

In this section, we introduce the Graph Combinatorial Optimization Network (GCON). A notable feature of our architecture is that it utilizes several different types of filtering operations in each layer. It includes localized aggregation operations similar to those used in common message-passing neural networks (which may be interpreted as low-pass filters from a GSP perspective). We also include comparison operations that extract multi-scale geometric information and consider changes across

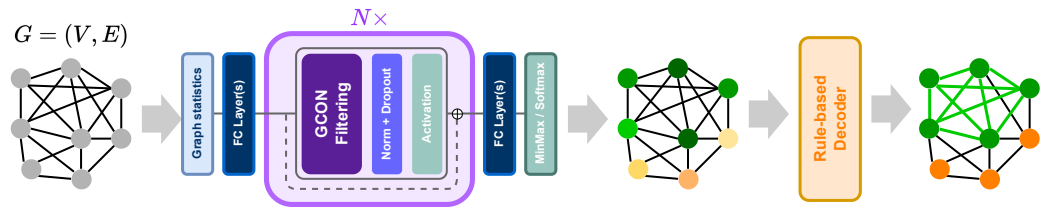

Figure 1: Illustration of our general framework. Graphs are first appended node features derived from graph statistics and then passed through a GNN composed of GCON hybrid layer blocks (detail in Fig. 2(a)) with a min-max or softmax output layer. A rule-based decoder then processes the node-level outputs to determine the set of interest.

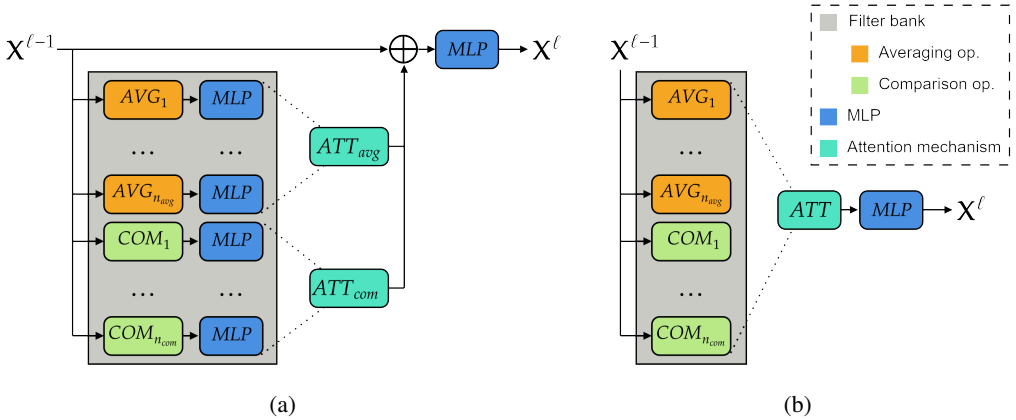

Figure 2: Illustration of the layer-wise update for (a) the new decoupled GCON filter bank and (b) the ScatteringClique filter bank from Min et al. [25].

the different scales. These comparison operations will take the form of wavelets, based on those utilized in the geometric scattering transform [14, 13, 43], and can be interpreted as band-pass filters again from a GSP perspective. The different filters of each type will be combined using a localized attention mechanism that allows the network to focus on different filters at each node, which makes our architecture fundamentally different from many traditional GNN methods that apply the same aggregation at each node of the graph.

5.1 The GCON filter bank: Aggregation and Comparison operations

The most basic component of our filter bank is a traditional one-step aggregation somewhat similar to popular message passing neural networks (e.g., Kipf and Welling [22], Hamilton et al. [16]). It has the form $F_1(\mathbf{X}) \coloneqq \mathrm{m}\left(\mathbf{P}\,\mathbf{X}\right)$, where $\mathbf{P} = \frac{1}{2}(\mathbf{I} + \mathbf{A}\,\mathbf{D}^{-1})$ is the lazy random walk matrix.[2] The function $\mathrm{m} \coloneqq \mathrm{m}_{\sigma,\theta}$ can be either a linear layer or multi-layer perceptron (MLP) with activation function σ and learned weights θ. At each node, this process averages information from all direct neighbors of the node, followed by a transformation step that learns new cross-feature combinations. Networks utilizing this setup are commonly referred to as *aggregate-transform* GNNs [39] due to the alternation of *aggregations* $\mathbf{X} \to \mathrm{AGG}(\mathbf{X})$ and *transformations* of node features $\mathrm{AGG}(\mathbf{X}) \to \mathrm{m}(\mathrm{AGG}(\mathbf{X}))$. Additionally, we also consider higher-order aggregations of the form

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

We will refer to the filters F_k as *aggregation operations*.

Next, we add filters inspired by diffusion wavelets [9] that calculate the difference of two different aggregations F_{k_1}, F_{k_2}, where $k_1 \neq k_2$, by setting

$$F_{k_1,k_2}(\mathbf{X}) \coloneqq \mathrm{m}\left((\mathbf{P}^{k_1} - \mathbf{P}^{k_2})\,\mathbf{X}\right). \tag{2}$$

[2]One could readily replace $\mathbf{P}$ with generic graph shift operators $\mathbf{S} \in \mathbb{R}^{n\times n}$, i.e., a matrix where $\mathbf{S}[i,j] = 0$ unless $\{v_i, v_j\} \in E$ or $i = j$. Likewise with the more complex filters $F_k(\mathbf{X})$ and $F_{k_1,k_2}(\mathbf{X})$ defined below.

Filters of this form are fundamentally different from $F_k(\mathbf{X})$ discussed above. From the GSP perspective, $F_{k_1,k_2}(\mathbf{X})$ constitute band-pass filters, whereas $F_k(\mathbf{X})$ constitute low-pass filters. Since the operation $\mathbf{X} \to (\mathbf{P}^{k_1} - \mathbf{P}^{k_2})\,\mathbf{X}$ compares aggregations at two different scales, we will refer to them as *comparison operations*. We denote the set of all our filtering operations as $\mathcal{F} = \mathcal{F}_\mathrm{A} \sqcup \mathcal{F}_\mathrm{C}$, where $\mathcal{F}_\mathrm{A}$ consists of aggregation operations and $\mathcal{F}_\mathrm{C}$ consists of comparison operations. As our filter bank consists of two different types of filters, we will also refer to it as a hybrid filter bank.

The use of the F_{k_1,k_2} is inspired by the geometric scattering transform [43, 14, 27], a handcrafted multi-layer network, which iteratively filters the input node features via dyadic diffusion wavelets of the form $\mathbf{X} \to (\mathbf{P}^{2^{j-1}} - \mathbf{P}^{2^j})\,\mathbf{X}$ and takes vertex-wise absolute values in between wavelet filterings. Notably, the original versions of the geometric scattering transform were handcrafted feature extractors. However, subsequently, Wenkel et al. [36], Tong et al. [31] used the geometric scattering transform as a basis for a fully learned graph neural network. In particular, Tong et al. [31] aimed to learn the optimal diffusion scales (i.e., the powers k_i to which $\mathbf{P}$ is raised) and Wenkel et al. [36] introduced a *hybrid-scattering network* that utilized both aggregation operations (low-pass filters) and comparison operations (band-pass, wavelet filters).

5.2 Attention over filters

We let $\mathbf{X}^{\ell-1}$ denote the input to the ℓ-th layer and use an attention mechanism to determine the importance of information from different filters for each node. For each filter response $\mathbf{H}_f := F(\mathbf{X}^{\ell-1}) \in \mathbb{R}^{n\times d}$, we concatenate it with a matrix of transformed input features $\mathbf{H} := \mathrm{m}(\mathbf{X}^{\ell-1})$, and apply an attention mechanism to their horizontal concatenation $[\mathbf{H} \,\|\, \mathbf{H}_f]$. This takes the form

$$\mathbf{s}_f^A := \sigma\left([\mathbf{H} \,\|\, \mathbf{H}_f]\,\mathbf{a}_A\right), f \in \mathcal{F}_A, \quad \text{and} \quad \mathbf{s}_f^C := \sigma\left([\mathbf{H} \,\|\, \mathbf{H}_f]\,\mathbf{a}_C\right), f \in \mathcal{F}_C$$

where $\mathbf{a}_A, \mathbf{a}_C \in \mathbb{R}^{2d}$ are learned attention vectors for aggregation and comparison filters, respectively. We then use softmax to normalize the importance scores across filters within $\mathcal{F}_\mathrm{A}$ and $\mathcal{F}_\mathrm{C}$ at each vertex, resulting in a normalized score $\bar{\mathbf{s}}_f^\mathrm{A} \in \mathbb{R}^n$ and $\bar{\mathbf{s}}_f^\mathrm{C} \in \mathbb{R}^n$ given by

$$\bar{\mathbf{s}}_f^\mathrm{A}(v) = \mathrm{softmax}\left(\{\mathbf{s}_f[v] : f \in \mathcal{F}_\mathrm{A}\}\right), \quad \bar{\mathbf{s}}_f^\mathrm{C}(v) = \mathrm{softmax}\left(\{\mathbf{s}_f[v] : f \in \mathcal{F}_\mathrm{C}\}\right). \tag{3}$$

Importantly, $\mathbf{s}_f^\mathrm{A}$ and $\mathbf{s}_f^\mathrm{C}$ (and thus each $\bar{\mathbf{s}}_f^\mathrm{A}$ and $\bar{\mathbf{s}}_f^\mathrm{C}$) will take different values for each node. Therefore, they act as localized attention mechanisms, which can up-weight different filters at different vertices.

Next, the filter responses are reweighted according to the attention scores, yielding aggregation and comparison representations $\mathbf{H}_{\mathcal{F}_\mathrm{A}}, \mathbf{H}_{\mathcal{F}_\mathrm{C}} \in \mathbb{R}^{n\times d}$:

$$\mathbf{H}_{\mathcal{F}_\mathrm{A}} = \textstyle\sum_{f\in\mathcal{F}_\mathrm{A}} \left(\bar{\mathbf{s}}_f^\mathrm{A} \circ \mathbf{1}_d\right) \odot \mathbf{H}_f, \quad \text{and} \quad \mathbf{H}_{\mathcal{F}_\mathrm{C}} = \textstyle\sum_{f\in\mathcal{F}_\mathrm{C}} \left(\bar{\mathbf{s}}_f^\mathrm{C} \circ \mathbf{1}_d\right) \odot \mathbf{H}_f,$$

where, $\circ$ and $\odot$ denote the outer and Hadamard product, respectively. Finally, $\mathbf{X}^\ell$ is then given by

$$\mathbf{X}^\ell = \mathrm{MLP}\left(\mathbf{X}^{\ell-1} + \mathbf{H}_{\mathcal{F}_\mathrm{A}} + \mathbf{H}_{\mathcal{F}_\mathrm{C}}\right). \tag{4}$$

Our layer-wise update rule is inspired by the one utilized in Min et al. [25], which also used a localized attention mechanism to balance the contributions of aggregation and comparison operations (i.e., low-pass and band-pass filters). However, our network improves upon Min et al. [25] in several ways. Most importantly, it computes separate normalized attention scores $\bar{\mathbf{s}}_f^\mathrm{A}(v)$ and $\bar{\mathbf{s}}_f^\mathrm{C}(v)$ for the aggregation and the comparison operators, whereas Min et al. [25] uses a single softmax for the entire filter bank $\mathcal{F}$. Additionally, Min et al. [25] omits the learnable function m in Equations 1 and 2, and uses a different layer-wise update in place of Equation 4. For further details, see Appendix A.1.

5.3 Theoretical Analysis

As noted in the previous subsection, one of the improvements we introduce relative to Min et al. [25] is a different method for computing the normalized attention scores. We normalize separately over $\mathcal{F}_\mathrm{A}$ and $\mathcal{F}_\mathrm{C}$, resulting in $\bar{\mathbf{s}}^\mathrm{A}$ and $\bar{\mathbf{s}}^\mathrm{C}$, where Min et al. [25] applies a single attention mechanism over the entire filter bank $\mathcal{F} = \mathcal{F}_\mathrm{A} \sqcup \mathcal{F}_\mathrm{C}$. Below, we provide a theoretical analysis that underlines the advantages of this modification. For clarity, we refer to our method as the *decoupled architecture* since it separates (i.e., decouples) the aggregation operations $\mathcal{F}_A$ and the comparison operations

$\mathcal{F}_C$ when computing the normalized attention scores. We also refer to a variant of our method that computes attention over the entire filter bank together (similar to Min et al. [25]) as the *non-decoupled architecture*. In our analysis of the non-decoupled architecture, we also assume that the function $\mathrm{m}(.)$ is not learned (only applying a nonlinear activation as in Min et al. [25]).

To understand the decoupled architecture, we note that the low-pass filters F_k and band-pass filters F_{k_1,k_2} constitute fundamentally different operations and capture different types of information. While low-pass filters use a weighted averaging of the features of the node's neighbors, band-pass filters *compare* two such averages of neighborhoods at different scales. As different types of information may be useful at different nodes, it is important that our MLP has access to both families of filterings.

However, we find that when grouped together, the outputs of the low-pass filters tend to have a larger magnitude than the band-pass filters. Therefore, the MLP is incapable of extracting band-pass information in the non-decoupled architecture. In the remainder of this section, we prove a theorem illustrating this phenomenon as a complement to our empirical evidence presented later. We will prove a result demonstrating that, in the non-decoupled architecture, the low-pass information will dominate the band-pass information as long as the receptive fields of the filters are sufficiently large. For simplicity, we will assume that we have a single node feature so that the input to our layer is a signal $\mathbf{x} \in \mathbb{R}^n$. Moreover, for $f \in \mathcal{F}$, we will write $\mathbf{h}_f[v]$ to denote the response of f at v.

Definition 1 (Band-dominant representation)**.** Consider a graph $G = (V, E)$, a non-negative input signal $\mathbf{x} \in \mathbb{R}^n$, and a filter bank $\mathcal{F} = \mathcal{F}_\mathrm{A} \sqcup \mathcal{F}_\mathrm{C}$. For $c > 1$, we refer to the representation at a node $v \in V$ $\mathbf{x} \mapsto (\bar{\mathbf{s}}_f[v], \mathbf{h}_f[v])_{f\in\mathcal{F}}$ as c-band-dominant if

$$\max\{\bar{\mathbf{s}}_f[v] : f \in \mathcal{F}_\mathrm{C}\} = c \cdot \max\{\bar{\mathbf{s}}_f[v] : f \in \mathcal{F}_\mathrm{A}\}.$$

A band-dominant representation may be interpreted as a situation where the model is paying more attention to band-pass filter information at a specific node than to low-pass filter information. We now show that, under certain assumptions, the model may nevertheless primarily use the low-pass information. We provide the proof of this statement in the Appendix A.2.

Theorem 1. *Consider the non-decoupled architecture and a c-band-dominant representation $\mathbf{x} \mapsto (\bar{\mathbf{s}}_f, \mathbf{h}_f)_{f\in\mathcal{F}}$ at a node $v \in V$ for a non-negative input signal $\mathbf{x} \geq 0$. Then, if all filters have sufficiently large scales, the low-pass filter responses will always dominate the band-pass filter responses in the sense that*

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

where $\epsilon > 0$ can be made arbitrarily small if the scales of the filters are sufficiently large.

For adequate choices of filters, we can therefore expect the magnitude of filter responses of low-pass filters to be significantly larger than the band-pass filter responses. This leads us to the conclusion that the non-decoupled attention limits the ability of the network to leverage diverse information. Our *decoupled* architecture addresses this shortcoming by (i) learning the functions $\mathrm{m}(\cdot)$ for each filter in Equations 1 and 2 to individually *re-balance* the filter responses, and (ii) *decoupling* low-pass from band-pass filters to prevent the former from dominating the latter.

6 Experiments

Data & Benchmarks: Our choice of benchmark datasets is based on those used in previous works exploring GNNs for CO problems [18, 41, 30]. For the maximum clique problem, we use the RB model from Xu et al. [38] to generate challenging synthetic graphs as proposed by Karalias and Loukas [18]. For MDS and maximum cut, we follow Zhang et al. [41] and Sun et al. [30], generating Barabási–Albert (BA) graphs [2] on *small* (200-300 vertices) and *large* (800-1200 vertices) scales.

For benchmarks, we use Gurobi [15], a state-of-the-art program solver that outperforms all deep learning-based methods in Karalias and Loukas [18] and Zhang et al. [41], alongside deep-learning-based solvers and two heuristic approaches: Greedy (see Appendix F for the specific algorithms) and mean-field annealing (MFA) [5], an extension of simulated annealing (SA) [23] that replaces the Markov process formulation of SA with a mean-field to guide the search procedure.

We note that while Gurobi is very accurate when given large amounts of time, on tighter time budgets it is unable to find optimal solutions [18]. As for deep-learning-based solvers, we compare against

Karalias and Loukas [18] (Erdős), Zhang et al. [41] (GFN), Min et al. [25] (ScatteringClique), and Sun et al. [30] (Anneal). The Erdős models and ScatteringClique represent our main benchmarks since our model & training framework, a GNN-based encoder-decoder with customized message-passing layers, is most analogous to them. Anneal represents a different avenue in that it proposes a novel training framework with an annealed loss function while borrowing Erdős's architecture. For a fairer comparison, we also compare against larger variants of Erdős and Anneal with the number of layers and hidden dimensions set to match GCON at each task; denoted Erdős-large and Anneal-large respectively. GFN is a generative flow network [3] based solver that represents the state-of-the-art deep learning method on various problems, despite beingless scalable than GNN-based solvers.

As input node features to GCON, we compute a collection of node-level statistics; we find that this approach is superior to using one-hot encodings (akin to Erdős/Anneal) or random features. For BA datasets we use node degree, eccentricity, cluster coefficient, and triangle counts; we drop eccentricity on RB graphs due to its computational cost. These features are then mapped to the hidden dimension of our GNN layers with a linear layer or a shallow MLP.

Table 1: Performance comparison of GCON with other baselines on *small* datasets. Average of three runs with standard deviation listed. The best deep learning-based method is highlighted in **bold**, and second best is underlined. † indicates Gurobi outperforms all deep learning methods on a given task.

Method	Type	MCut size ↑ BA-small	Time ↓	MClique size ↑ RB-small	Time ↓	MDS size ↓ BA-small	Time ↓
GUROBI	OR	732.47†	13:04	19.05†	1:55	27.89†	1:47
GREEDY	H	684.53 ± 1.17	0:13	13.54 ± 0.10	0:25	37.39 ± 0.19	2:13
MFA	H	719.78 ± 1.08	1:36	14.81 ± 3.64	0:27	36.23 ± 0.13	2:56
GFN	SSL	700.23 ± 0.60	2:57	**16.22 ± 0.06**	0:42	**29.14 ± 0.68**	2:20
ANNEAL	SSL	704.28 ± 1.32	0:05	13.06 ± 0.15	3:42	66.52 ± 35.1	1:50
ANNEAL-large	SSL	719.57 ± 1.56	0:08	13.27 ± 0.12	3:10	32.45 ± 0.49	1:20
ERDŐS	SSL-GNN	704.72 ± 1.39	0:05	13.09 ± 0.19	3:42	69.92 ± 40.8	1:50
ERDŐS-large	SSL-GNN	720.04 ± 1.23	0:08	13.21 ± 0.13	3:10	32.48 ± 0.29	1:20
ScatteringClique	SSL-GNN	724.45 ± 1.46	0:18	15.80 ± 0.12	4:06	31.07 ± 0.12	1:11
GCON (Ours)	SSL-GNN	**727.09 ± 1.49**	0:22	15.87 ± 0.15	4:09	30.26 ± 0.30	1:15

Table 2: Performance comparison of GCON with other baselines on *large* datasets. Average of three runs with standard deviation listed. The best deep learning-based method is highlighted in **bold**, and second best is underlined. † indicates Gurobi outperforms all deep learning methods on a given task.

Method	Type	MCut size ↑ BA-large	Time ↓	MClique size ↑ RB-large	Time ↓	MDS size ↓ BA-large	Time ↓
GUROBI	OR	2915.29	1:05:29	33.89†	16:40	103.80†	13:48
GREEDY	H	2781.30 ± 3.59	3:07	27.05 ± 0.26	0:25	141.95 ± 0.34	35:01
MFA	H	2929.65 ± 4.14	7:16	28.56 ± 0.19	2:19	167.05 ± 1.59	36:31
GFN	SSL	2826.64 ± 19.1	21:20	**31.73 ± 1.50**	4:50	113.77 ± 1.94	32:12
ANNEAL	SSL	2858.85 ± 1.31	0:05	24.86 ± 0.63	4:12	317.30 ± 137	31:42
ANNEAL-large	SSL	2878.73 ± 21.4	0:08	24.06 ± 0.48	4:20	131.89 ± 2.48	14:29
ERDŐS	SSL-GNN	2858.72 ± 2.78	0:05	25.20 ± 0.14	4:13	274.09 ± 95.0	30:28
ERDŐS-large	SSL-GNN	2881.48 ± 11.8	0.08	23.92 ± 0.66	4:19	134.39 ± 2.83	14:29
ScatteringClique	SSL-GNN	2952.03 ± 4.85	0:19	29.36 ± 0.38	4:21	121.26 ± 1.68	14:52
GCON (Ours)	SSL-GNN	**2961.19 ± 3.58**	0:27	29.46 ± 0.51	4:57	**113.47 ± 0.63**	11:24

Table 3: Ablation study comparing our decoupled GCON layer to the non-decoupled version and common message-passing layers from the literature for the MCut problem on the BA-small dataset. Average of three runs with standard deviation listed.

Convolution	MCut size ↑ BA-small	MClique size ↑ RB-small	MDS size ↓ BA-small
GCON (Ours, decoupled)	**727.09 ± 1.49**	**15.87 ± 0.15**	**30.26 ± 0.30**
GCON (Ours, non-decoupled)	725.11 ± 1.77	15.78 ± 0.23	39.17 ± 5.47
ScatteringClique [25]	724.45 ± 1.46	15.80 ± 0.12	31.07 ± 0.20
GCN [22]	684.24 ± 0.20	15.24 ± 0.16	48.43 ± 1.66
GIN [39]	691.86 ± 0.88	14.48 ± 0.33	33.96 ± 0.71

Results & Analysis: In Tables 1 and 2, we report the mean result of the objective sizes, in addition to inference times for *small* and *large* datasets respectively. OR refers to algorithmic solvers, H refers to heuristics while SSL refers to the self-supervised learning algorithms we compete against. SSL-GNN denotes self-supervised methods with specialized GNN layers. OR results are as reported in Zhang et al. [41], while we evaluate all H & SSL baselines on our synthetic datasets using identical data splits across multiple seeds. More information on our experimental setup is available in Appendix B.

GCON attains the most striking results for the maximum cut problem (MCut). In both BA-small and BA-large, we obtain the best results amongst all non-OR methods: On BA-small, we obtain a max-cut size of 727.09, a mere five short of the Gurobi solver. On BA-large, the results are arguably even more impressive: We not only attain the best result amongst non-OR methods, but also surpass the Gurobi solver by almost 45 (due to the fact that Gurobi is unable to reach an exact solution within the time constraint). The success of GCON is further underlined by the fact that the inference time is fairly robust to the growing graph sizes in MCut, taking just 27 seconds to evaluate on 500 graph, whereas it takes the Gurobi solver more than an hour to do so.

On the maximum clique (MClique) and minimum dominating set (MDS) benchmarks, GCON remains very competitive. We surpass GFN by a small margin to attain the best result for MDS on large graphs, while outperforming the GNN-based solvers ScatteringClique, Erdős and Anneal (including the -large variants) as well as the heuristic methods on all MClique and MDS benchmarks. The iterative GFlowNet algorithm proves particularly useful for MClique, and represents the best SSL method for both MClique experiments as well as MDS-BA-small. We should point out that further gains for our algorithm may be possible here by integrating the optimization framework proposed in Anneal [30] into our framework. In the meantime, by outperforming our main baselines Erdős and ScatteringClique on all benchmarks tested on, we demonstrate the power of GCON as a model framework for a wide array of graph-CO problems.

Finally, we perform an ablation study on the message-passing layers in order to demonstrate the architectural gains of the GCON layer, presented in Table 3. We compare our GCON with a variant *without* the decoupled filter bank as well as ScatteringClique and two well-established baseline MPNNs, GCN [22] and GIN [39], the latter of which forms the backbone of the Erdős' GNN architecture. We use identical depth, width, and training procedure for all models. GCON provides a clear improvement of about three cut-edges on average compared to ScatteringClique, and more than 35 compared to the best MPNN baseline (GIN) thanks to our hybrid layer. Decoupling the filter bank provides an additional two-edge improvement on average. We note that these gains are further amplified on the large counterparts. Similar to the Erdős' GNN, we use skip-connections and batch normalization between layers and perform minimal hyperparameter tuning on several components like the number of layers, layer normalization, and the use of skip-connections within the hybrid layer. See Appendix B for further details on hyperparameter tuning.

7 Conclusion

We have introduced GCON, a novel GNN that uses a sophisticated pair of decoupled filter banks and a localized attention mechanism for solving several well-known CO problems. GCON can be trained by self-supervised loss functions, which allow us to estimate the probability $\mathbf{p}(v)$ that each node is in the set of interest and then determine our solution by utilizing rule-based decoders. We then demonstrate the effectiveness of our method compared to a variety of common GNNs and other SSL methods, obtaining leading performance for the max-cut problem. We note that in the future, it would be interesting to (a) combine our architectural improvements with better optimization frameworks to leverage the power of our model better, and (b) extend GCON to other CO problems such as MIS and graph coloring to further validate its efficacy.

Limitations and future work. We found our framework to be more difficult to optimize well on larger datasets. This insight corroborates our understanding that the self-supervised loss functions associated with graph-CO problems are highly non-convex and typically challenging to optimize. This motivates future work that incorporates the annealed optimization framework from Sun et al. [30] into GCON for superior performance. Additionally, the primary challenge regarding extending GCON to other graph-CO problems would be to develop appropriate self-supervised losses or efficient methods of producing labeled data to allow for supervised learning approaches.

Acknowledgements and Disclosure of Funding

This work was partially funded by Fin-ML CREATE graduate studies scholarship for PhD [Frederik Wenkel]; Bourse en intelligence artificielle des Études supérieures et postdoctorales (ESP) 2023-2024 [Semih Cantürk]; NSF OIA 2242769 [Michael Perlmutter]; Canada CIFAR AI Chair, IVADO (Institut de valorisation des données) grant PRF-2019-3583139727, FRQNT (Fonds de recherche du Québec - Nature et technologies) grant 299376 and NSERC Discovery grant 03267 [Guy Wolf]; NSF DMS grant 2327211 [Michael Perlmutter and Guy Wolf]. This research was also enabled in part by compute resources provided by Mila (mila.quebec). The content provided here is solely the responsibility of the authors and does not necessarily represent the official views of the funding agencies.

References

[1] Sergi Abadal, Akshay Jain, Robert Guirado, Jorge López-Alonso, and Eduard Alarcón. Computing graph neural networks: A survey from algorithms to accelerators. *ACM Computing Surveys (CSUR)*, 54(9):1–38, 2021. 1, 2

[2] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999. doi: 10.1126/science.286.5439.509. URL https://www.science.org/doi/abs/10.1126/science.286.5439.509. 7, 15

[3] Emmanuel Bengio, Moksh Jain, Maksym Korablyov, Doina Precup, and Yoshua Bengio. Flow network based generative models for non-iterative diverse candidate generation. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 27381–27394. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/e614f646836aaed9f89ce58e837e2310-Paper.pdf. 2, 8

[4] Federico Berto, Chuanbo Hua, Nayeli Gast Zepeda, André Hottung, Niels Wouda, Leon Lan, Junyoung Park, Kevin Tierney, and Jinkyoo Park. Routefinder: Towards foundation models for vehicle routing problems, 2024. URL https://arxiv.org/abs/2406.15007. 3

[5] Griff Bilbro, Reinhold Mann, Thomas Miller, Wesley Snyder, David van den Bout, and Mark White. Optimization by mean field annealing. In D. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 1. Morgan-Kaufmann, 1988. URL https://proceedings.neurips.cc/paper_files/paper/1988/file/ec5decca5ed3d6b8079e2e7e7bacc9f2-Paper.pdf. 7

[6] Léo Boisvert, Hélène Verhaeghe, and Quentin Cappart. Towards a generic representation of combinatorial problems for learning-based approaches. In Bistra Dilkina, editor, *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 99–108, Cham, 2024. Springer Nature Switzerland. 3

[7] A. Bojchevski and S. Günnemann. Deep gaussian embedding of graphs: Unsupervised inductive learning via ranking. In *Proc. of ICLR*, 2018. 2

[8] Quentin Cappart, Didier ChÃ©telat, Elias B. Khalil, Andrea Lodi, Christopher Morris, and Petar Velickovic. Combinatorial optimization and reasoning with graph neural networks. *Journal of Machine Learning Research*, 24(130):1–61, 2023. URL http://jmlr.org/papers/v24/21-0449.html. 2

[9] Ronald R Coifman and Mauro Maggioni. Diffusion wavelets. *Applied and computational harmonic analysis*, 21(1):53–94, 2006. 5

[10] Austin Derrow-Pinion, Jennifer She, David Wong, Oliver Lange, Todd Hester, Luis Perez, Marc Nunkesser, Seongjae Lee, Xueying Guo, Brett Wiltshire, et al. Eta prediction with graph neural networks in google maps. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pages 3767–3776, 2021. 1

[11] Darko Drakulic, Sofia Michel, and Jean-Marc Andreoli. Goal: A generalist combinatorial optimization agent learning, 2024. URL https://arxiv.org/abs/2406.15079. 3

[12] Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019. 15

[13] Fernando Gama, Alejandro Ribeiro, and Joan Bruna. Diffusion scattering transforms on graphs. In *International Conference on Learning Representations*, 2018. 5

[14] Feng Gao, Guy Wolf, and Matthew Hirn. Geometric scattering for graph data analysis. In *International Conference on Machine Learning*, pages 2122–2131. PMLR, 2019. 2, 5, 6

[15] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2023. URL https://www.gurobi.com. 7

[16] William L Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 1025–1035, 2017. 5

[17] J. J. Hopfield and D. W. Tank. "neural" computation of decisions in optimization problems. *Biological Cybernetics*, 52(3):141–152, Jul 1985. ISSN 1432-0770. doi: 10.1007/BF00339943. URL https://doi.org/10.1007/BF00339943. 2

[18] Nikolaos Karalias and Andreas Loukas. Erdos goes neural: an unsupervised learning framework for combinatorial optimization on graphs. *Advances in Neural Information Processing Systems*, 33:6659–6672, 2020. 2, 4, 7, 8, 15, 16

[19] Richard Karp. Reducibility among combinatorial problems. volume 40, pages 85–103, 01 1972. ISBN 978-3-540-68274-5. doi: 10.1007/978-3-540-68279-0_8. 2

[20] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *The Bell System Technical Journal*, 49(2):291–307, 1970. doi: 10.1002/j.1538-7305.1970.tb01770.x. 19

[21] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017. URL https://arxiv.org/abs/1412.6980. 15

[22] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *the 4th International Conference on Learning Representations (ICLR)*, 2016. 1, 5, 8, 9

[23] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983. doi: 10.1126/science.220.4598.671. URL https://www.science.org/doi/abs/10.1126/science.220.4598.671. 7

[24] Ilya Loshchilov and Frank Hutter. SGDR: stochastic gradient descent with restarts. *CoRR*, abs/1608.03983, 2016. URL http://arxiv.org/abs/1608.03983. 15

[25] Yimeng Min, Frederik Wenkel, Michael Perlmutter, and Guy Wolf. Can hybrid geometric scattering networks help solve the maximum clique problem? *Advances in Neural Information Processing Systems*, 35:22713–22724, 2022. 2, 4, 5, 6, 7, 8, 13

[26] Antonio Ortega, Pascal Frossard, Jelena Kovačević, José MF Moura, and Pierre Vandergheynst. Graph signal processing: Overview, challenges, and applications. *Proceedings of the IEEE*, 106 (5):808–828, 2018. 2

[27] Michael Perlmutter, Alexander Tong, Feng Gao, Guy Wolf, and Matthew Hirn. Understanding graph neural networks with generalized geometric scattering transforms. *SIAM Journal on Mathematics of Data Science*, 5(4):873–898, 2023. 6

[28] Ryan A. Rossi, David F. Gleich, Assefaw H. Gebremedhin, and Md. Mostofa Ali Patwary. Fast maximum clique algorithms for large graphs. In *Proceedings of the 23rd International Conference on World Wide Web*, WWW '14 Companion, page 365–366, New York, NY, USA, 2014. Association for Computing Machinery. ISBN 9781450327459. doi: 10.1145/2567948.2577283. URL https://doi.org/10.1145/2567948.2577283. 20

[29] David I Shuman, Sunil K Narang, Pascal Frossard, Antonio Ortega, and Pierre Vandergheynst. The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *IEEE signal processing magazine*, 30(3):83–98, 2013. 2

[30] Haoran Sun, Etash K. Guha, and Hanjun Dai. Annealed training for combinatorial optimization on graphs, 2022. 2, 7, 8, 9, 15, 16, 20

[31] Alexander Tong, Frederik Wenkel, Dhananjay Bhaskar, Kincaid Macdonald, Jackson Grady, Michael Perlmutter, Smita Krishnaswamy, and Guy Wolf. Learnable filters for geometric scattering modules. *arXiv preprint arXiv:2208.07458*, 2022. 6

[32] Jan Tönshoff, Martin Ritzert, Hinrikus Wolf, and Martin Grohe. RUN-CSP: unsupervised learning of message passing networks for binary constraint satisfaction problems. *CoRR*, abs/1909.08387, 2019. URL http://arxiv.org/abs/1909.08387. 2

[33] Jan Tönshoff, Berke Kisin, Jakob Lindner, and Martin Grohe. One model, any csp: Graph neural networks as fast global search heuristics for constraint satisfaction. In Edith Elkind, editor, *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI-23*, pages 4280–4288. International Joint Conferences on Artificial Intelligence Organization, 8 2023. doi: 10.24963/ijcai.2023/476. URL https://doi.org/10.24963/ijcai.2023/476. Main Track. 3

[34] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, Yoshua Bengio, et al. Graph attention networks. *stat*, 1050(20):10–48550, 2017. 1

[35] Zichen Wang and Vassilis N. Ioannidis. How aws uses graph neural networks to meet customer needs, 2022. URL https://www.amazon.science/blog/how-aws-uses-graph-neural-networks-to-meet-customer-needs. 1

[36] Frederik Wenkel, Yimeng Min, Matthew Hirn, Michael Perlmutter, and Guy Wolf. Overcoming oversmoothness in graph convolutional networks via hybrid scattering networks. *arXiv preprint arXiv:2201.08932*, 2022. 6

[37] Shiwen Wu, Fei Sun, Wentao Zhang, Xu Xie, and Bin Cui. Graph neural networks in recommender systems: a survey. *ACM Computing Surveys*, 55(5):1–37, 2022. 1, 2

[38] Ke Xu, Frédéric Boussemart, Fred Hemery, and Christophe Lecoutre. Random constraint satisfaction: Easy generation of hard (satisfiable) instances. *Artificial Intelligence*, 171(8): 514–534, 2007. ISSN 0004-3702. doi: https://doi.org/10.1016/j.artint.2007.04.001. URL https://www.sciencedirect.com/science/article/pii/S0004370207000653. 7, 15

[39] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2018. 1, 5, 8, 9

[40] Jiaxuan You, Rex Ying, and Jure Leskovec. Design space for graph neural networks. In *NeurIPS*, 2020. 15

[41] Dinghuai Zhang, Hanjun Dai, Nikolay Malkin, Aaron Courville, Yoshua Bengio, and Ling Pan. Let the flows tell: Solving graph combinatorial optimization problems with gflownets. *arXiv preprint arXiv:2305.17010*, 2023. 2, 7, 8, 9, 15, 16

[42] Jianan Zhou, Zhiguang Cao, Yaoxin Wu, Wen Song, Yining Ma, Jie Zhang, and Xu Chi. MVMoE: Multi-task vehicle routing solver with mixture-of-experts. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp, editors, *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 61804–61824. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/zhou24c.html. 3

[43] Dongmian Zou and Gilad Lerman. Graph convolutional neural networks via scattering. *Applied and Computational Harmonic Analysis*, 49(3):1046–1074, 2020. 5, 6

Our appendix is organized as follows. In Appendix A, we provide further details on the baseline architecture Min et al. [25] and the proof of Theorem 1. In Appendix B, we give further details on our experimental setup and hyperparameters. In Appendices C and D, we conduct and discuss further experiments related to timing and model generalization respectively. Appendix E presents the MIP formulations for the graph-CO tasks that were employed in Gurobi. Finally, in Appendix F we provide the greedy heuristic algorithms used as baselines.

A Further Theoretical Analysis

A.1 Further details on the baseline architecture [25]

The GNN-based method most closely related to our work is Min et al. [25], which also utilized a complex filter bank consisting of low-pass and band-pass filters. As alluded to in Section 5.2, our method builds on Min et al. [25] in certain ways. (i) We utilize a learnable function m in our filtering operations; (ii) we use separate attention mechanisms for the aggregation operations $\mathcal{F}_A$ and the comparison operations $\mathcal{F}_C$; and (iii) our layer-wise update rule takes a different form. To clarify these differences, below, we give a more detailed recollection of the architecture from Min et al. [25].

In Min et al. [25], similar to our method, the input features $\mathbf{X}^{\ell-1}$ of the ℓth layer were filtered by each filter from a filter bank $\mathcal{F}$ that decomposes into low-pass and band-pass filters $\mathcal{F} = \mathcal{F}_{\mathrm{A}} \sqcup \mathcal{F}_{\mathrm{C}}$. The filter responses were also derived according to Equation 1 and 2. However, in Min et al. [25], they use a simple, non-learned function $\mathrm{m}(.)$ that is just an activation function $\mathrm{m} = \sigma$. By contrast, we use learned functions.

Next, Min et al. [25] used an attention mechanism to determine the importance of information of different filters for each individual node. For each filter response $\mathbf{H}_f \coloneqq F(\mathbf{X}) \in \mathbb{R}^{n\times d}$ of $F \in \mathcal{F}$, and input features $\mathbf{X}$, this takes the form

$$\mathbf{s}_f \coloneqq \sigma\left([\mathbf{X} \,\|\, \mathbf{H}_f]\,\mathbf{a}\right) \in \mathbb{R}^{n\times 1},$$

where $\|$ denotes horizontal concatenations. However, differing from our method, when computing the normalized attention scores the softmax is applied over the entire filter bank. This results in a single set of normalized attention scores given by $\bar{\mathbf{s}}_f \in \mathbb{R}^n$ where

$$\bar{\mathbf{s}}_f(v) = \text{softmax}\left(\{\mathbf{s}_f(v) : f \in \mathcal{F}\}\right)$$

whereas in GCON, we compute sepearate normalized attention scores $\bar{\mathbf{s}}_f^A$ and $\bar{\mathbf{s}}_f^C$ for the aggregation and comparison operators. Next, the filter responses are re-weighted according to the attention scores, yielding $\mathbf{H}_{\mathcal{F}} \in \mathbb{R}^{n\times d}$, where

$$\mathbf{H}_{\mathcal{F}} = \textstyle\sum_{f\in\mathcal{F}} \left(\bar{\mathbf{s}}_f \circ \mathbf{1}_d\right) \odot \mathbf{H}_f,$$

where, as in Section 5.2, $\circ$ and $\odot$ denote the outer and Hadamard product, respectively. The final output of each block is then generated by simply applying an MLP to the $\mathbf{H}_{\mathcal{F}}$, i.e.,

$$\mathbf{X}^{\ell} \coloneqq \text{MLP}(\mathbf{H}_{\mathcal{F}}), \tag{5}$$

instead of the update rule considered in Equation 4, which adds $\mathbf{H}_{\mathcal{F}_A}$ and $\mathbf{H}_{\mathcal{F}_C}$ to $\mathbf{X}^{\ell-1}$ before applying the MLP.

A.2 Proof of Theorem 1

We start by stating a copy of Theorem 1 presented in Section 5.3 of the main text.

Theorem. *Consider the non-decoupled architecture and a c-band-dominant representation* $\mathbf{x} \mapsto (\bar{\mathbf{s}}_f, \mathbf{h}_f)_{f\in\mathcal{F}}$ *at a node* $v \in V$ *for a non-negative input signal* $\mathbf{x} \geq 0$*. Then, if all filters have sufficiently large scales, the low-pass filter responses will always dominate the band-pass filter responses in the sense that*

$$\sum_{f\in\mathcal{F}_C} \bar{\mathbf{s}}_f[v]\,\mathbf{h}_f[v] \leq \epsilon c \sum_{f\in\mathcal{F}_A} \bar{\mathbf{s}}_f[v]\,\mathbf{h}_f[v], \tag{6}$$

where $\epsilon > 0$ *can be made arbitrarily small if the scales of the filters are sufficiently large.*

Proof. Let $\epsilon > 0$, let $v \in V$ be fixed and set $p_\infty := \| \mathbf{x} \|_1 \, \mathbf{d}[v] / \| \mathbf{d} \|_1$, where $\mathbf{d} \in \mathbb{R}^n$ is the degree vector. As $k \to \infty$, it is known that $\mathbf{P}^k(\mathbf{x} / \|\mathbf{x}\|_1)$ converges to the stationary distribution $\mathbf{d} / \| \mathbf{d} \|_1$, which implies that $F_k(\mathbf{x}) = \mathbf{P}^k \, \mathbf{x}$ converges to $\| \mathbf{x} \|_1 \, \mathbf{d} / \| \mathbf{d} \|_1$. This in turn implies that $\lim_{k_1,k_2 \to \infty} F_{k_1,k_2}(\mathbf{x}) = 0$.

Therefore, for all $\delta > 0$, there exists some $K := K(\delta) \in \mathbb{N}$ such that if all of the filters in $f \in \mathcal{F}_\mathrm{A}$ have the form $f = F_k$, $k \geq K$, and if all of the filters in $f \in \mathcal{F}_\mathrm{C}$ have the form $f = F_{k_1,k_2}$, $k_1, k_2 \geq K$, then we have that

$$|\mathbf{h}_f[v] - p_\infty| \leq \delta \text{ for all } f \in \mathcal{F}_\mathrm{A}$$

and

$$|\mathbf{h}_f[v]| \leq \delta \text{ for all } f \in \mathcal{F}_\mathrm{C}.$$

We next set $\tau := \max\{\bar{\mathbf{s}}_f[v] : f \in \mathcal{F}_\mathrm{C}\}$, so that the left-hand-side of Equation 6 can be estimated from above as

$$\text{LHS} = \textstyle\sum_{f \in \mathcal{F}_\mathrm{C}} \bar{\mathbf{s}}_f[v] \, \mathbf{h}_f[v] \leq \sum_{f \in \mathcal{F}_\mathrm{C}} \tau\delta = |\mathcal{F}_\mathrm{C}|\tau\delta.$$

At the same time, if we let $f_\mathrm{A}^{\max}$ denote the filter from $\mathcal{F}_\mathrm{A}$ corresponding to the largest value of $\bar{\mathbf{s}}_f$, we can recall the definition of a c-band dominate representation to estimate the right-hand-side from below by

$$\text{RHS} \geq \epsilon \cdot c \cdot \bar{\mathbf{s}}_{f_\mathrm{A}^{\max}} \mathbf{h}_{f_\mathrm{A}^{\max}}[v] \geq \epsilon \cdot c \cdot \frac{\tau}{c} \left(p_\infty - \delta\right) = \epsilon \cdot \tau \left(p_\infty - \delta\right).$$

Hence, Equation 6 will be true if

$$|\mathcal{F}_\mathrm{C}| \leq \frac{p_\infty - \delta}{\delta} \epsilon. \tag{7}$$

We now recall that for filters with sufficiently large scales (i.e., for sufficiently large K), δ can be arbitrarily small such that the inequality will hold. To confirm this, we note that $g(\delta) := (p_\infty - \delta)/\delta$ is strictly increasing when decreasing $0 < \delta < p_\infty$ and tends to infinity as $\delta \downarrow 0$. Thus, for sufficiently large K we can choose δ small enough such that Equation 7 holds. □

B Experimental setup & hyperparameters

Our experimental framework is built on PyTorch Geometric [12] and GraphGym [40]. All experiments are conducted using a single GPU and 4 CPUs; most experiments (e.g. MClique experiments on -small graphs) ran within several minutes to an hour, while the largest (e.g. MDS on BA-large using 16-layer-256-width GCON) took several hours to converge.

We perform all timing experiments in Tables 1 and 2 following the setup in Zhang et al. [41], using an NVIDIA V100 GPU for fair comparison of the inference time estimates on 500 samples. We report worst-case results, i.e., use a batch size of 1; assuming that Zhang et al. [41] have done so despite not listing an explicit batch size for the timing experiments. Additional timing experiments comparing the GCON layer with other baselines can be found in Appendix C.

As mentioned, the self-supervised loss functions associated with combinatorial graph problems are difficult to optimize well, due to their highly non-convex nature. We thus conducted hyperparameter optimization on several main components of our model. For each task, a separate set of hyperparameter tuning experiments were conducted for the number of layers, layer width, and activation, as well as layer normalization for layers and learning rate where appropriate.

In addition to the configurations listed in Table 4, all tasks used Adam optimizer [21] and a cosine annealing scheduler [24] with a 5-epoch warm-up period. Other hyperparameters shared by all tasks include the use of batch normalization and dropout with 0.3 probability.

Below in Table 4, GSN refers to graph size normalization where features are divided by the graph size after each layer, as per Karalias and Loukas [18]. L2 refers to L2 normalization. Additionally, two different skip connection strategies are considered, denoted *skipsum* and *stack-concat*. Skipsum refers to summing the output of the previous layer to the output features; in stack-concat no actual skip connections are used, but all intermediate GCON layer outputs are concatenated after the final GNN layer and passed to the post-GNN fully-connected layers accordingly. We found that stack-concat performs better overall, but skipsum is particularly helpful when the GNN inner dimension is large.

Table 4: Overview of model hyperparameters associated with the reported results for each dataset.

	MCut BA-small	**MCut BA-large**	**MClique RB-small**	**MClique RB-large**	**MDS BA-small**	**MDS BA-large**
# Pre-GNN layers	1	4	1	1	1	1
# GNN (Hybrid) layers	16	16	20	20	16	16
# Post-GNN layers	1	1	2	2	1	1
Hybrid layer width	32	32	32	32	256	256
Hybrid layer normalization	None	L2	GSN	GSN	L2	L2
Hybrid layer activation	ELU	ELU	GELU	GELU	GELU	GELU
MLP activation	LReLU (0.3)	LReLU (0.3)	LReLU (0.01)	LReLU (0.01)	LReLU (0.3)	GELU
Skip connection	stack-concat	skipsum	stack-concat	stack-concat	stack-concat	skipsum
Skip conn. in Hybrid layer	Yes	Yes	Yes	Yes	No	No
Learning rate	1e-3	3e-3	1e-3	1e-3	3e-3	3e-3
# Epochs	200	400	100	100	200	200
# Batch size	256	256	8	8	256	256
Decoder K	N/A	N/A	10	10	1	1

Further discussion on benchmarks. Our main benchmarking pipeline (Tables 1 and 2) parallel Zhang et al. [41] to a considerable extent in that we evaluate GCON on the same tasks (MCut, MClique and MDS), using datasets drawn from identical distributions, i.e. RB [38] and BA [2] graphs generated with identical parameters; we also use their GFN model and several of the baselines from their study as methods to compare against in our study. Gurobi results in Tables 1 and 2 are as reported in Zhang et al. [41], while we evaluate all H & SSL baselines on our synthetic datasets using identical data splits across multiple seeds. We adapted the Greedy and MFA methods from Anneal [30] for MClique and MDS, while implementing them ourselves for MCut, since it was a problem not considered in Sun et al. [30].

We additionally provide a brief discussion on differences between our results and those in Zhang et al. [41] to allow for better interpretability of our results.

- We observed that model depth is particularly important for GNN-based methods on MCut and MDS, as we see our deeper GNN variants Erdős-large and Anneal-large outperform their

shallower counterparts significantly. We conjecture that Zhang et al. [41] did not evaluate on deeper variants of ERDŐS/ANNEAL on MCut, which may explain why these methods underperform in their paper. On the other hand, MClique did not benefit substantially from model depth, where the -large and base models performed very similarly. Shallow models particularly struggled on MDS, and exhibited extreme variance across runs.

- Annealing was less helpful in our case than suggested in Sun et al. [30] and Zhang et al. [41] on MClique. This is likely due to the fact that we were able to optimize Erdős' GNN better, as both Erdős and Anneal results converged to values similar to those corresponding to Anneal in Zhang et al. [41]. We also noted that Erdős' and Anneal were difficult to tune for MDS in general, and fell well short of the results presented in Zhang et al. [41]. This applied to MClique as well, albeit to a lesser extent. We believe their implementation of Erdős' GNN is based on Sun et al. [30], which diverge significantly from that of Karalias and Loukas [18], e.g. does not use masking. For consistency across all tasks, we modeled our implementation on Karalias and Loukas [18]; but note that the Sun et al. [30] implementation performed marginally better for MClique (though still significantly worse than GCON).
- Heuristic results were very consistent with those in Zhang et al. [41] for MClique and MDS, but our MFA implementation for MCut was seemingly more powerful than theirs – while the Greedy results were similar (with a 0.5% drop on BA-small and 0.7% rise on BA-large), our MFA results improved over theirs by 2.2% and 3.4% respectively. While we do not think these differences make any substantial difference, we still report them for completeness, particularly as we do not have access to the exact implementations nor parameters used in their work.

C Timing studies

We also performed a set of timing experiments to provide an overview of the scalability of the GCON layer compared to ScatteringClique and GCN, two GNN layers benchmarked against in this study. Using the configurations denoted in Table 4, we ran our framework on 500 test graphs for each dataset. We see that even when the considered datasets are identical, as in the case of MCut and MDS (both tasks use the BA datasets), both the duration and scalability of the test runs are largely dependent on the task-specific decoder: The MCut decoder is faster than the MDS decoder across all three models; perhaps more remarkably, while the MCut decoder scales very well from BA-small to BA-large, the runtimes for the MDS decoder increase by 1000+% for all models between the two datasets. This implies that for GNN-based neural solvers, larger efficiency gains are likely to be made by focusing on optimizing the decoder architectures than the GNN layers themselves.

Decoder scalability aside, our GCON layer maintains similar runtimes with ScatteringClique and GCN across all experiments, indicating the overall scalability of the GCON model. We note that there is *some* overhead associated with our filters and attention mechanism such that GCN remains the fastest model in most cases, one exception is MDS on BA-large where our layer is faster than both baselines.

Table 5: Timing comparison of GCON, ScatteringClique and GCN on Small and Large datasets for the MCut and MClique tasks. The times are based on test datasets of 500 graphs.

Model	**Small**	**Large**	**% Increase (s)**
	MCut – BA	**MCut – BA**	
GCON	0:22	0:27	22.73
ScatteringClique	0:18	0:19	5.56
GCN	0:14	0:14	0.00
	MDS – BA	**MDS – BA**	
GCON	1:15	11:24	812.00
ScatteringClique	1:11	14:52	1156.34
GCN	1:14	13:36	1002.70
	MClique – RB	**MClique – RB**	
GCON	4:09	4:57	19.28
ScatteringClique	4:06	4:21	6.10
GCN	3:13	4:04	26.42

D Generalization study

We also present a brief study on the transferability of our GCON architecture. As every graph-CO problem in our study has a corresponding *small* and *large* benchmark dataset, we trained a GCON model on each, and tested the respective models on the *other* dataset (e.g. for MCut, we trained a GCON on BA-small and tested it on BA-large, and vice versa), and measured the percentage change in performance compared to the in-distribution case (e.g. GCON both trained and tested on BA-small). We repeated the study for GFN and Erdős-large as well for comparison. Our results are presented in Table 6, while Table 7 shows average changes across several groupings such as type of generalization and generalization per graph-CO problem.

Both GCON and GFN prove to be quite transferable across all our benchmarks, with less than 6% average drop in performance for GCON and only a 2% drop for GFN. Nevertheless, the generalizability patterns of the two methods are considerably different. GCON is consistently better when generalizing from *small* to *large* datasets while it struggles more in the opposite direction; these trends are completely reversed for GFN, which does not exhibit any negative transfer when generalizing from *large* to *small*. When looking at individual CO problems, GCON is particularly robust on MCut in both directions while GFN is more robust for MClique and MDS. Erdős-large is on par with both methods on MCut and MClique, but is vastly inferior with huge performance drops in MDS. We note that despite the marginal drops, GCON still comfortably outperforms most in-distribution baselines even in the transfer learning setting.

Table 6: Generalization results between *small* and *large* graphs for GCON and GFN across all tasks. Average of three runs listed.

Task	Model	Train Data	Test Data	Base Perf.	Transfer Perf.	% Change
MCut	GFN	BA-large	BA-small	700.23	704.99	0.68%
MCut	GCON	BA-large	BA-small	727.09	723.45	-0.50%
MCut	ERDŐS-large	BA-large	BA-small	720.04	664.82	-7.67%
MCut	GFN	BA-small	BA-large	2826.64	2569.44	-9.10%
MCut	GCON	BA-small	BA-large	2961.19	2951.18	-0.34%
MCut	ERDŐS-large	BA-small	BA-large	2881.48	2895.51	0.49%
MClique	GFN	RB-large	RB-small	16.22	16.26	0.25%
MClique	GCON	RB-large	RB-small	15.87	13.62	-14.17%
MClique	ERDŐS-large	RB-large	RB-small	13.21	13.00	-1.56%
MClique	GFN	RB-small	RB-large	31.73	30.16	-4.95%
MClique	GCON	RB-small	RB-large	29.46	28.57	-3.01%
MClique	ERDŐS-large	RB-small	RB-large	23.91	26.27	9.85%
MDS	GFN	BA-large	BA-small	29.14	29.13	0.02%
MDS	GCON	BA-large	BA-small	30.26	33.89	-11.99%
MDS	ERDŐS-large	BA-large	BA-small	32.48	46.82	-44.14%
MDS	GFN	BA-small	BA-large	113.77	112.89	0.77%
MDS	GCON	BA-small	BA-large	113.47	118.70	-4.60%
MDS	ERDŐS-large	BA-small	BA-large	134.39	362.68	-169.87%

Table 7: Aggregated % performance changes reported in Table 6 for GCON and GFN.

Group	GFN	GCON	ERDŐS-large
Overall	-2.05%	-5.77%	-35.48%
Small → Large	-4.42%	-2.65%	-53.18%
Large → Small	0.32%	-8.89%	-17.79%
MCut	-4.21%	-0.42%	-3.59%
MClique	-2.35%	-8.59%	4.15%
MDS	0.40%	-8.30%	-107.01%

E MIP formulations

MCut

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

MClique

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

MDS

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

F Greedy heuristic algorithms

Algorithm 1 Heuristic algorithm for MCut, inspired by Kernighan and Lin [20]

```
function MCutHeuristic(V, E)        ▷ Input: vertices V, edges E; Output: set of cut edges S
    (S_1, S_2) ← RandomPartition(V)        ▷ Randomly partition vertices into two sets
    for each v ∈ V do                      ▷ Iteratively optimize the partition
        c_1 ← CountEdges(v, S_1, E)        ▷ Edges from v to nodes in S_1
        c_2 ← CountEdges(v, S_2, E)        ▷ Edges from v to nodes in S_2
        if v ∈ S_1 and c_1 > c_2 then      ▷ Move v to S_2 if it improves the cut
            Move(v, S_1, S_2)
        else if v ∈ S_2 and c_2 > c_1 then ▷ Move v to S_1 if it improves the cut
            Move(v, S_2, S_1)
        end if
    end for
    S ← {}                                 ▷ Initialize the set of cut edges
    for each {v_i, v_j} ∈ E do             ▷ Identify edges crossing the cut
        if (v_i ∈ S_1 and v_j ∈ S_2) or (v_i ∈ S_2 and v_j ∈ S_1) then
            Add({v_i, v_j}, S)
        end if
    end for
    return S                               ▷ Return the set of edges in the maximum cut
end function
```

Algorithm 2 Heuristic algorithm for MClique [28], implementation based on Sun et al. [30]

function MCLIQUEHEURISTIC(V, E) ▷ Input: vertices V, edges E; Output: clique S
  $S \leftarrow \{\}$ ▷ Initialize the clique as an empty set
  $vList \leftarrow$ SORTBYDEGREE($V, E, descending$) ▷ Sort vertices by descending degree
  for $v_i \in vList$ **do** ▷ Iterate through vertices in sorted order
    if $S = \{\}$ **then** ▷ If the clique is empty, start with the first vertex
      $S \leftarrow \{v_i\}$
    end if
    if MAINTAINSCLIQUE(v_i, S, E) **then** ▷ Check if adding v_i to S maintains a valid clique
      ADD(v_i, S) ▷ Add v_i to the clique
    end if
  end for
  return S ▷ Return the maximum clique found
end function

Algorithm 3 Heuristic algorithm for MDS, based on Sun et al. [30]

function MDSHEURISTIC(V, A) ▷ Input: vertices V, adjacency matrix A; Output: dominating set S
  $S \leftarrow \{\}$ ▷ Initialize the dominating set as empty
  $vList \leftarrow$ SORTBYDEGREE($V, A, descending$) ▷ Sort vertices by descending degree
  for $v_i \in vList$ **do** ▷ Initialize probability of each node being in the dominating set
    $p_i \leftarrow 0.5$
  end for
  for $v_i \in vList$ **do** ▷ Initialize probability of node being "uncovered" by the dominating set
    $u_i \leftarrow (1 - p_i) \cdot \prod_{j=1}^{n} \left(A_{ij} \cdot (1 - p_j) + (1 - A_{ij})\right)$
  end for
  for $v_i \in vList$ **do** ▷ Iterate over sorted vertices
    $potential \leftarrow \frac{10}{1-p_i} \cdot \left(u_i + \sum_{j \in \mathcal{N}(i)} u_j\right)$ ▷ Calculate potential of v_i
    if $potential > 1$ **then** ▷ Add v_i to the dominating set if potential exceeds threshold
      ADD(v_i, S) ▷ Add v_i to dominating set S
      $p_i \leftarrow 1$ ▷ Set probability of v_i being in the dominating set to 1
      $u_i \leftarrow 0$ ▷ Mark v_i as covered
      $u_j \leftarrow 0 \quad \forall j \in \mathcal{N}(i)$ ▷ Mark neighbors of v_i as covered
    else
      $p_i \leftarrow 0$ ▷ Set v_i as non-dominant
      $u_i \leftarrow \frac{u_i}{1-p_i}$ ▷ Adjust uncover probability of v_i
      $u_j \leftarrow u_j \cdot (1 - A_{ij}) + A_{ij} \cdot \frac{u_j}{1-p_i} \quad \forall j$ ▷ Adjust neighbors' uncover probabilities
    end if
  end for
  return S ▷ Return the dominating set
end function