Can Hybrid Geometric Scattering Networks Help Solve the Maximal Clique Problem?

Anonymous Author(s)
Affiliation
Address
email

Abstract

We propose a geometric scattering-based graph neural network (GNN) for approximating solutions of the NP-hard maximal clique (MC) problem. We construct a loss function with two terms, one which encourages the network to find a large set of nodes and the other which acts as a surrogate for the constraint that the nodes form a clique. We then use this loss to train a novel GNN architecture that outputs a vector representing the probability for each node to be part of the MC and apply a rule-based decoder to make our final prediction. The incorporation of the scattering transform alleviates the so-called oversmoothing problem that is often encountered in GNNs and would degrade the performance of our proposed setup. Our empirical results demonstrate that our method outperforms representative GNN baselines in terms of solution accuracy and inference speed as well as conventional solvers like GUROBI with limited time budgets.

1 Introduction

The success of Graph Neural Networks (GNNs) for a variety of machine learning tasks [Scarselli et al., 2008, Gori et al., 2005, Kipf and Welling, 2016, Gilmer et al., 2017] has sparked interests in using GNNs to solve graph combinatorial optimization (CO) problems [Li et al., 2018a, Karalias and Loukas, 2020]. For example, Joshi et al. [2019] introduces a GNN-based method for approximately solving the travelling salesman problem (TSP), while Karalias and Loukas [2020] approximate solutions of the maximal clique (MC) problem. Solving such problems in an end-to-end fashion via GNNs is challenging for several reasons. First of all, many CO problems are provably either NP-Hard or NP-complete. Therefore, learning CO solvers in a supervised or semi-supervised fashion can be computationally infeasible since the cost of generating the truth labels grows exponentially with the problem size. Second, the solution to the CO problems must satisfy a number of constraints or limitations (e.g. belonging to a clique). Although these constraints can be imposed during training time, there is still no guarantee that these constraints are still satisfied at test time. Furthermore, solving graph combinatorial problems largely depends on the expressive power of GNNs. Many GNNs aggregate information via local averaging, which can be interpreted as a smoothing operation. This degrades the expressive power of GNNs and results in the so-called oversmoothing problem [Li et al., 2018b] where neighboring nodes have similar representations and are difficult to distinguish from each other. This is problematic if a node that is not in the solution set borders many points that are in the solution set.

The purpose of this paper is to use a geometric scattering-based GNN to approximate the solution of the maximal clique (MC) problem, that is to find the largest complete subgraph contained within a large graph G. The use of geometric scattering, rather than a more traditional GNN is motivated by recent work [Wenkel et al., 2022] showing that the geometric scattering transform can help overcome the oversmoothing problem via the use of bandpass wavelet filters in conjunction with GCN-type

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

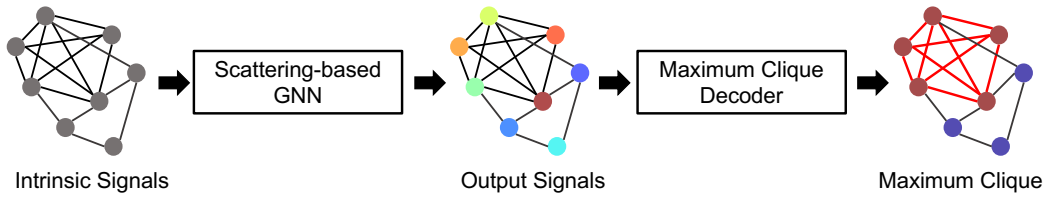

Figure 1: Illustration of our model pipeline. We use a scattering-based GNN to learn a discriminative node representation and use a decoder to extract the MC from the learned representation.

filters. This is particularly important in the context of the MC problem because it is critical to distinguish a point which connects to many members of the clique from an actual member of the clique. As we shall show, our method, which utilizes bandpass wavelet filters is able to better detect the border between the MC and the rest of the graph. This parallels the traditional use of wavelets as edge detectors in image processing[Grossmann, 1988].

Our method uses a two-phase strategy shown in Figure 1. We first use an unsupervised learning model to generate an efficient representation of how likely each node is to be part of the maximal clique. Then, this representations is fed into a constraint-preserving decoder to generate the maximal clique. Ideally, given a graph G with n nodes, we would build a representation $\boldsymbol{p} \in [0,1]^n$ where the nodes in the maximal clique are assigned the value 1 (True) while all other nodes are assigned 0 (False), so that we can 'cut' the MC from the graph via $O(n)$ operations. In practice, we will build a representation which assigns large probabilities to the maximal clique nodes and small probabilities to the other nodes. Figure 2 (a) presents a situation where oversmoothing does not occur. The signal $\boldsymbol{p}$ is much larger on the nodes $(1,2,3,4,6)$, which form the maximal clique, than it is on the other nodes $(5,7,8)$. Therefore, the decoder can easily find the MC by, e.g., setting a threshold. Figure 2 (b), on the other hand, illustrates a situation in which oversmoothing does occur and $\boldsymbol{p}$ has less variability. Indeed, there are no blue colored nodes at all. In this setting, it is nearly impossible to distinguish node 7, which does not belong to the maximal clique, from the nodes $1, 6$, and therefore extracting the MC from $\boldsymbol{p}$ is difficult if not impossible.

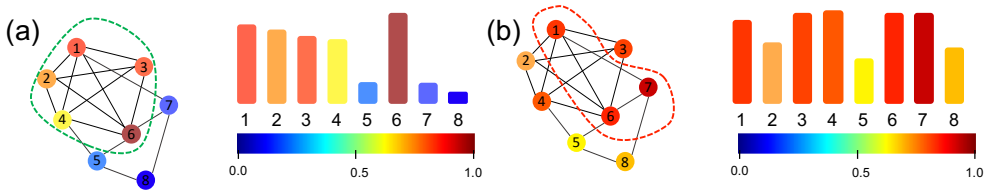

Figure 2: Illustration of the oversmoothing problem. Left: A discriminative representation, which can be interpreted as a probability for each node to belong to the maximal clique or not. Right: Due to oversmoothing problem, the GNN generates a smooth representation, making MC and non-MC nodes difficult to discriminate.

2 Background and Related Work

2.1 Graph Signal Processing

Consider a graph $G = (V, E)$ with a set of nodes (or vertices) $V := \{v_1, \ldots, v_n\}$ and a set of undirected edges $E \subset V \times V$. For a function $x : V \to \mathbb{R}$ defined on V, we will, in minor abuse of notation identify $\boldsymbol{x}$ with the vector where $\boldsymbol{x}[i] = x(v_i)$. We let $\boldsymbol{W} \in \mathbb{R}^{n \times n}$ denote the *adjacency matrix* of G and define the *symmetric normalized grah Laplacian* by

$$\boldsymbol{\mathcal{L}} := \boldsymbol{I}_n - \boldsymbol{D}^{-1/2}\, \boldsymbol{W}\, \boldsymbol{D}^{1/2},$$

where $d_i \coloneqq \sum_{j=1}^{n} W[v_i, v_j]$ is the *degree* of the node v_i and $\boldsymbol{D} \coloneqq \mathrm{diag}(d_1, ..., d_n) \in \mathbb{R}^{n \times n}$ is the degree matrix. We will let $\mathbf{q}_i$ and λ_i denote the (normalized) eigenvectors and eigenvalues of $\boldsymbol{\mathcal{L}}$ with $\boldsymbol{\mathcal{L}}\,\mathbf{q}_i = \lambda_i \mathbf{q}_i$, and make use of the eigendecomposition

$$\boldsymbol{\mathcal{L}} = \boldsymbol{Q}\,\boldsymbol{\Lambda}\,\boldsymbol{Q}^\top,$$

where $\boldsymbol{Q}$ is the orthogonal matrix whose i-th column is the normalized eigenvector $\boldsymbol{q}_i$, and $\boldsymbol{\Lambda} \coloneqq \mathrm{diag}(\lambda_1, ..., \lambda_n)$. Notably, the eigenvectors $\boldsymbol{q}_1, \dots \boldsymbol{q}_n$ can be seen as a generalization of Fourier modes in the context of graph domains with the corresponding eigenvalues representing the increasing frequencies $0 \leq \lambda_1 \leq \cdots \leq \lambda_n \leq 2$. This link may be established by viewing frequency as a measure of variation, in particular the variation of (degree-normalized) Fourier modes across edges is $\lambda_i = \boldsymbol{q}_i^\top\,\boldsymbol{\mathcal{L}}\,\boldsymbol{q}_i = \sum_{\{u,v\} \in E} (\tilde{\boldsymbol{q}}_i[u] - \tilde{\boldsymbol{q}}_i[v])^2$ for $\tilde{\boldsymbol{q}}_i \coloneqq \boldsymbol{D}^{-1/2}\,\boldsymbol{q}_i$. The graph *Fourier transform* of a signal vector $\boldsymbol{x}$ is then defined as by $\hat{\boldsymbol{x}}[i] = \langle \boldsymbol{x}, \boldsymbol{q}_i \rangle$ and the inverse Fourier is given by $\boldsymbol{x} = \sum_{i=1}^{n} \hat{\boldsymbol{x}}[i]\,\boldsymbol{q}_i$. Compactly, we write $\hat{\boldsymbol{x}} = \boldsymbol{Q}^\top \boldsymbol{x}$ and $\boldsymbol{x} = \boldsymbol{Q}\hat{\boldsymbol{x}}$.

In the Euclidean setting, it is known that convolution in the spatial domain corresponds to multiplication in the Fourier domain. This fact has motivated work such as Shuman et al. [2016], which defines the convolution of a signal $\boldsymbol{x}$ with a filter $\boldsymbol{g}$ to be the unique vector verifying $(\widehat{\boldsymbol{g} \star \boldsymbol{x}})[i] = \hat{\boldsymbol{g}}[i]\hat{\boldsymbol{x}}[i]$, which implies that

$$\boldsymbol{g} \star \boldsymbol{x} = \sum_{i=1}^{n} \hat{\boldsymbol{g}}[i]\hat{\boldsymbol{x}}[i]\,\boldsymbol{q}_i = \sum_{i=1}^{n} \hat{\boldsymbol{g}}[i]\langle \boldsymbol{q}_i, \boldsymbol{x} \rangle\,\boldsymbol{q}_i = \boldsymbol{Q}\,\widehat{\boldsymbol{G}}\,\boldsymbol{Q}^\top\,\boldsymbol{x},$$

where $\widehat{\boldsymbol{G}} \coloneqq \mathrm{diag}(\hat{\boldsymbol{g}}) = \mathrm{diag}(\hat{\boldsymbol{g}}[1], \dots, \hat{\boldsymbol{g}}[n])$. When incorporating this notion of convolution into a graph neural network, a common choice [e.g., Defferrard et al., 2016] is to require the coefficients $\hat{\boldsymbol{g}}[i]$ to be polynomials of the eigenvalues $\lambda_i, i \in [n]$, i.e., $\hat{\boldsymbol{g}}[i] \coloneqq \sum_k \gamma_k \lambda_i^k$ in which case we have $\widehat{\boldsymbol{G}} = \sum_k \gamma_k \boldsymbol{\Lambda}^k$. This allows one to implement convolution in the spatial domain by verifying that

$$\boldsymbol{g} \star \boldsymbol{x} = \sum_k \gamma_k \mathcal{L}^k \boldsymbol{x}.$$

In particular, the entire method may be implemented without the need to diagonalize a matrix which is extremely expensive for large graphs.

2.2 Graph Convolutional Networks

In Kipf and Welling [2016], the authors set $\hat{\boldsymbol{g}}[i] \coloneqq \theta(2 - \lambda_i)$, where θ is a real number, which yields

$$\boldsymbol{g}_\theta \star \boldsymbol{x} = \theta(2\,\boldsymbol{I}_n - \boldsymbol{\mathcal{L}}) = \theta\left(\boldsymbol{I}_n + \boldsymbol{D}^{-1/2}\,\boldsymbol{W}\,\boldsymbol{D}^{-1/2}\right)\boldsymbol{x}\,. \tag{1}$$

As the eigenvalues of the above convolutional filter lie in [0,2] and could lead to vanishing or exploding gradients, Kipf and Welling [2016] apply a renormalization trick which replaces $\boldsymbol{I}_n + \boldsymbol{D}^{-1/2}\,\boldsymbol{W}\,\boldsymbol{D}^{-1/2}$ with

$$\boldsymbol{A} \coloneqq (\boldsymbol{D} + \boldsymbol{I})^{-1/2}(\boldsymbol{W} + \boldsymbol{I})(\boldsymbol{D} + \boldsymbol{I})^{-1/2}.$$

The layer-wise propagation rule of GCN is then defined by $\boldsymbol{x}_j^\ell = \sigma\big(\sum_{i=1}^{N_{\ell-1}} \theta_{ij}^\ell \boldsymbol{A}\,\boldsymbol{x}_i^{\ell-1}\big)$, where $\boldsymbol{x}_i^{\ell-1} \in \mathbb{R}^n$ is the i-th feature vector in layer $\ell - 1$, θ_{ij}^ℓ is a trainable parameter and $\sigma(.)$ is a nonlinear activation function, $\boldsymbol{x}_j^\ell$ is the j-th feature activation vector in layer ℓ. We can further write this compactly in matrix notation, this gives

$$\boldsymbol{X}^\ell = \sigma\left(\boldsymbol{A}\,\boldsymbol{X}^{\ell-1}\,\boldsymbol{\Theta}^\ell\right), \tag{2}$$

where $\boldsymbol{\Theta}^\ell \in \mathbb{R}^{N_{\ell-1} \times N_\ell}$ is the weight-matrix of layer ℓ and $\boldsymbol{X}^\ell \in \mathbb{R}^{n \times N_\ell}$ contains the activations output by layer ℓ. The GCN model updates node features at every node by averaging over the node features of the node itself and its neighbors, which enforces similarity throughout node neighborhoods. This makes nodes increasingly difficult to discriminate for 'deeper' models and is widely referred to as the oversmoothing problem [Li et al., 2018b]. From a graph spectral theory perspective, oversmoothing is related to lowpass filtering as the filter in Eq. 1 puts larger weights on the low-frequency spectrum (as $0 \leq \lambda_i \leq 2$). This motivates the idea of using GCN-type lowpass filters in conjunction with bandpass filters, that can be implemented, for example, using graph scattering [Min et al., 2020], which we discuss in the following.

2.3 Graph Scattering

The graph scattering transform [Gama et al., 2019, Zou and Lerman, 2019, Gao et al., 2019] is a wavelet-based model for machine learning on graphs. Unlike the one-hop localized low-pass filters used in GCN, which aim to promote smoothness between neighboring nodes, these wavelets are band-pass filters, which in turn incorporate long-range dependencies through the large spatial support of the used aggregations. The scattering transform is based upon raising the lazy random walk matrix

$$\boldsymbol{P} \coloneqq \frac{1}{2}\left(\boldsymbol{I}_n + \boldsymbol{W}\,\boldsymbol{D}^{-1}\right)$$

to different powers in order to capture the diffusion geometry of the graph G at various time scales. In particular, subtracting such powers allows us to detect changes in these diffusion geometries. Following the lead of Coifman and Maggioni [2006], for $k \in \mathbb{N}_0$, we define a wavelet matrix $\boldsymbol{\Psi}_k \in \mathbb{R}^{n\times n}$ at scale 2^k by

$$\boldsymbol{\Psi}_0 \coloneqq \boldsymbol{I}_n - \boldsymbol{P}, \quad \boldsymbol{\Psi}_k \coloneqq \boldsymbol{P}^{2^{k-1}} - \boldsymbol{P}^{2^k}, \quad k \geq 1. \tag{3}$$

Intuitively, at every node, these diffusion wavelets can be seen as a comparison operation that calculates the difference of the averaged features of two neighborhoods of different sizes (namely sizes 2^{k-1} and 2^k). The geometric scattering transform is constructed through an alternating cascade on wavelet filterings and pointwise nonlinearities. Towards this end, we let $\sigma : \mathbb{R} \to \mathbb{R}$ be a nonlinear function which is nonexpansive in the sense that $|\sigma(x) - \sigma(y)| \leq |x - y|$, and for a scattering path $p \coloneqq (k_1, \ldots, k_m)$, we define

$$\boldsymbol{U}_p\boldsymbol{x} \coloneqq \boldsymbol{\Psi}_{k_m} \circ \sigma \circ \boldsymbol{\Psi}_{k_{m-1}} \cdots \sigma \circ \boldsymbol{\Psi}_{k_2} \circ \sigma \circ \boldsymbol{\Psi}_{k_1}\boldsymbol{x}. \tag{4}$$

3 Model

Our method is centered around three main components, (i) a hybrid scattering-GCN model M (Sec. 3.1-3.3) that transforms a small set of simple node-level statistics represented by the matrix $\boldsymbol{X} \in \mathbb{R}^{n\times d}$ to a probability vector $\boldsymbol{p} \in [0,1]^n$ representing the probabilities that each node is part of the maximal clique, (ii) an easy-to-optimize unsupervised loss function L (Sec. 3.4) that is small for "good" probability vectors $\boldsymbol{p}^\star$; and (iii) a rule-based decoder D (Sec. 3.5) that maps the probability vector to the maximal clique $C^\star \subset V$ of G.

$$\boldsymbol{X} \xrightarrow{M} \boldsymbol{p} \xleftrightarrow{L} \boldsymbol{p}^\star \xrightarrow{D} C^\star$$

3.1 Embedding Module

The input of our model is a node feature matrix $\boldsymbol{X} \in \mathbb{R}^{n\times d}$, containing d features for each node. In our experiments, we set $d = 3$ and let the features be the eccentricity, the clustering coefficient, and the logarithm of the degree of each vertex. The encoder transforms the node features to d_h-dimensional embeddings $\boldsymbol{H}^0$ using a multi-layer perceptrone (MLP) $\mathrm{m}_{\text{emb}} : \mathbb{R}^d \to \mathbb{R}^{d_h}$, i.e.,

$$\boldsymbol{H}^0 \coloneqq \mathrm{m}_{\text{emb}}(\boldsymbol{X}).$$

We will use $\boldsymbol{H}^0$ as the input to the diffusion model introduced in the next subsection.

3.2 Diffusion Module

The aggregation module consists of a cascade of $K \in \mathbb{N}$ aggregation (or diffusion) layers with operations that are chosen for each node via an attention mechanism. Our network is based upon the framework introduced in Wenkel et al. [2022]. However, our method differs from Wenkel et al. [2022] by storing every intermediate node representation $\boldsymbol{H}^\ell, 0 \leq \ell \leq K$, in a list of *readouts*.

In each layer ℓ, every node has access to node representations from set a of filters $\mathcal{F}$ that contains a selection of lowpass and bandpass filters. Similar to Wenkel et al. [2022], our bandpass filters $f_{\text{low},r}$ are modified GCN filters that have the form

$$f_{\text{low},r}(\boldsymbol{H}^{\ell-1}) = \boldsymbol{A}^r\,\boldsymbol{H}^{\ell-1} \tag{5}$$

and are parameterized by the power $r \geq 1$ of the matrix $\boldsymbol{A}$. Similarly, we define the scattering filter $f_{\text{band},k}$ of order k according to

$$f_{\text{band},k}(\boldsymbol{H}^{\ell-1}) = \boldsymbol{\Psi}_k \, \boldsymbol{H}^{\ell-1} \,. \tag{6}$$

Next, we want to obtain data-driven scores $s_f(v)$ that determine the importance of filter $f \in \mathcal{F}$ for node $v \in V$. We set $\boldsymbol{H}_f^\ell \coloneqq f(\boldsymbol{H}^{\ell-1})$ and calculate the scores via an attention mechanism, setting

$$\boldsymbol{s}_f^\ell \coloneqq \sigma\left(\boldsymbol{H}_f^\ell \,\|\, \boldsymbol{H}^{\ell-1}\right) \boldsymbol{a}^\ell, \tag{7}$$

which, at every node, calculates the dot product of the learnable attention vector $\boldsymbol{a}^\ell \in \mathbb{R}^{2d_h}$ with the filtered node representation of the node concatenated to its previous representation. We normalize the scores across the filters using the softmax function, i.e., $\alpha_f(v) = \mathrm{softmax}_{\mathcal{F}}(s_f(v))$, and use the corresponding attention vectors $\boldsymbol{\alpha}_f^\ell \in \mathbb{R}^n$, $f \in \mathcal{F}$ to update node representations via

$$\boldsymbol{H}_{\text{agg}}^\ell \coloneqq \sum_{f \in \mathcal{F}} \boldsymbol{\alpha}_f^\ell \odot \boldsymbol{H}_f^\ell \,. \tag{8}$$

Notably, the softmax function enforces that every node is likely to focus on one single filter. We note that the method is compatible with multi-head attention but omit further details as it is not necessary for the present experiments. Finally, we transform the aggregated node representations via an MLP $\mathrm{m}^\ell : \mathbb{R}^{d_h} \to \mathbb{R}^{d_h}$, i.e., $\boldsymbol{H}^\ell \coloneqq \mathrm{m}^\ell(\boldsymbol{H}_{\text{agg}}^\ell)$.

3.3 Output Module

In the output module, we combine the information from the readouts $\boldsymbol{H}^\ell, 0 \leq \ell \leq K$. We first concatenate the readouts horizontally to

$$\boldsymbol{H}_{\text{cat}} \coloneqq \|_{\ell=0}^{K} \boldsymbol{H}^\ell,$$

and then transform $\boldsymbol{H}_{\text{cat}}$ to a vector $\boldsymbol{h} \in \mathbb{R}^n$ using an MLP, i.e.,

$$\boldsymbol{h} \coloneqq \mathrm{m}_{\text{out}}(\boldsymbol{H}_{\text{cat}}).$$

Finally, to obtain a probability vector, we apply min-max normalization, i.e.,

$$\boldsymbol{p} \coloneqq (\boldsymbol{h} - \min(\boldsymbol{h}) \cdot \mathbf{1}_n)/(\max(\boldsymbol{h}) - \min(\boldsymbol{h})),$$

and interpret $\boldsymbol{p}[i]$ as the probability that the i-th node is part of the maximal clique.

3.4 Training Loss

We now derive an unsupervised training loss for the maximal clique problem. Letting Ω denote the set of cliques, we note that the task of finding the maximal clique is equivalent to identifying the clique $C \in \Omega$ that contains the most edges, i.e., maximizes the objective function

$$L^\star(C) \coloneqq \sum_{u,v \in C} w_{u,v}.$$

Our approach relies on producing a vector $\boldsymbol{p}$ such that $\boldsymbol{p}_v \approx 1$ if v is in the maximal clique and $\boldsymbol{p}_v \approx 0$ otherwise. We consider the vector $\boldsymbol{x} \in \mathbb{R}^n$ where $\boldsymbol{x}_v = 1$ if v is in the maximal clique and $\boldsymbol{x}_v = 0$ otherwise. We then model $\boldsymbol{x}_v$ as a Bernoulli random variable where $\mathbb{P}(\boldsymbol{x}_v = 1) = \mathbf{p}_v$. Our goal then becomes to maximize the expectation

$$\mathbb{E}\left[L^\star(C)\right] = \mathbb{E}\left[\sum_{(u,v) \in E} w_{u,v} \, \boldsymbol{x}_u \, \boldsymbol{x}_v\right] = \sum_{(u,v) \in E} w_{u,v} \, \boldsymbol{p}_u \, \boldsymbol{p}_v = \boldsymbol{p}^\top \boldsymbol{W} \, \boldsymbol{p} \,. \tag{9}$$

Maximizing (9) encourages $\boldsymbol{p}$ to be large on the support of the largest clique. However, we also the support of $\boldsymbol{p}$ to be concentrated within a clique. This naturally motivates the constrained optimization problem of minimizing

$$L_1(\boldsymbol{p}) \coloneqq -\boldsymbol{p}^\top \boldsymbol{W} \, \boldsymbol{p} \quad \text{subject to: } \mathrm{supp}(\boldsymbol{p}) \text{ is contained in a clique.} \tag{10}$$

However, the constraint that $\mathbf{p}$ is contained in a clique is hard to enforce directly. In order to construct a surrogate, we consider the complement graph $\overline{G} = (\overline{V}, \overline{E})$. The vertices of $\overline{G}$ are taken to be the same as G, i.e., $\overline{V} = V$, while the edges are defined by the rule that for $u \neq v$ there is an edge between u and v in $\overline{G}$ if and only if there is *not* an edge between them in G, i.e., $\overline{E} = \{(u, v) \in V \times V : u \neq v\} \backslash E$. We denote the adjacency matrix of $\overline{G}$ by $\overline{\boldsymbol{W}} = (\overline{w}_{u,v})_{u,v \in V}$, and note that

$$\overline{w}_{u,v} \coloneqq \begin{cases} 1 & \text{if } (u,v) \notin E \text{ and } u \neq v, \\ 0 & \text{otherwise.} \end{cases}$$

In particular, $\overline{\boldsymbol{W}}$ has zeros on the diagonal and can be computed from $\boldsymbol{W}$ via $\overline{\boldsymbol{W}} = \mathbf{1}_{n \times n} - (\boldsymbol{I} + \boldsymbol{W})$. The following Lemma indicates that a second loss term $L_2(\boldsymbol{p}) \coloneqq \boldsymbol{p}^\top \, \overline{\boldsymbol{W}} \, \boldsymbol{p}$ can be used to ensure that mass is primarily concentrated in a set of nodes that form a clique.

Lemma 1. *Consider a graph $G = (V, E)$, a signal $\boldsymbol{p} \geq 0$ and define* $\operatorname{supp}(\boldsymbol{p}) \coloneqq \{v \in V : \boldsymbol{p}_v > 0\}$. *Then, $L_2(\boldsymbol{p}) = \boldsymbol{p}^\top \, \overline{\boldsymbol{W}} \, \boldsymbol{p} = 0$ if and only if there exists a clique $C \subset V$ such that the support of $\mathbf{p}$ is contained in C.*

Proof. We first assume $\boldsymbol{p}^\top \, \overline{\boldsymbol{W}} \, \boldsymbol{p} = 0$ and suppose for the sake of a contradiction that there exists no clique $C \subset V$ with $\operatorname{supp}(\boldsymbol{p}) \subset C$. Then, there exist at least two nodes $u_0, v_0 \in \operatorname{supp}(\boldsymbol{p})$ with $\{u_0, v_0\} \notin E$. Now,

$$\boldsymbol{p}^\top \, \overline{\boldsymbol{W}} \, \boldsymbol{p} = \sum_{\{u,v\} \notin E} \boldsymbol{p}_u \, \boldsymbol{p}_v \geq \boldsymbol{p}_{u_0} \, \boldsymbol{p}_{v_0} > 0,$$

which is a contradiction.

For the opposite direction, we assume that there exists a clique $C \subset V$ such that $\operatorname{supp}(\boldsymbol{p}) \subset C$. Hence, every pair of distinct nodes in $\operatorname{supp}(\boldsymbol{p})$ is connected by an edge and consequently $\overline{w}_{u,v} = 0$ for all $u, v \in \operatorname{supp}(\boldsymbol{p})$. Thus,

$$\boldsymbol{p}^\top \, \overline{\boldsymbol{W}} \, \boldsymbol{p} = \sum_{u,v \in V} \overline{w}_{u,v} \, \boldsymbol{p}_u \, \boldsymbol{p}_v = \sum_{u,v \in \operatorname{supp}(\boldsymbol{p})} \overline{w}_{u,v} \, \boldsymbol{p}_u \, \boldsymbol{p}_v = 0.$$

□

In light of (9) and Lemma 1, we now define our training loss as

$$L(\boldsymbol{p}) \coloneqq L_1(\boldsymbol{p}) + \beta L_2(\boldsymbol{p}) = -\boldsymbol{p}^T \, \boldsymbol{W} \, \boldsymbol{p} + \beta \, \boldsymbol{p}^T \, \overline{\boldsymbol{W}} \, \boldsymbol{p} \, . \tag{11}$$

We note that the loss function decomposes in two terms. The first term, L_1, encourages the allocation of mass at highly connected nodes, while the second term L_2 encourages most of the mass of $\mathbf{p}$ to be contained in a clique. β is a hyper-parameter which is used to balance the contributions of the two terms.

3.5 Decoder

In order to extract the MC from $\boldsymbol{p}$, we use a greedy decoder detailed in in Alg. 1. Our algorithm creates a set $\Omega = \{\hat{C}_j\}_{j=1}^{\kappa}$ of κ cliques and then selects the $\hat{C}_j$ with largest cardinality. To construct the $\hat{C}_j$, we first create a sorting function $\pi : V \to \{1, \ldots, |V|\}$ which puts the nodes in descending order, so

$$\mathbf{p}_{\pi^{-1}(1)} \geq \cdots \geq \mathbf{p}_{\pi^{-1}(n)}.$$

Each $\hat{C}_j$ starts as a single vertex $\pi^{-1}(j)$. We then consider $\tau - 1$, additional candidate, $\pi^{-1}(j + 1), \ldots, \pi^{-1}(\tau)$ and accept each candidate vertex $v_{\text{canditate}}$ if $\hat{C}_j \cup \{v_{\text{candidate}}\}$ forms a clique. This is tested by verifying that $\boldsymbol{x}^\top \, \overline{\boldsymbol{W}} \, \boldsymbol{x} = 0$ for

$$\boldsymbol{x} \coloneqq \sum_{v \in \hat{C}_j \cup \{v_{\text{candidate}}\}} \mathbf{1}_v.$$

We remark that in order for our decoder to predict true maximal clique exactly, we must choose the parameters τ and κ such that there exists a j, $1 \leq j \leq \kappa$, such that $j \leq \pi^{-1}(i) \leq j + \tau$ for all v_i in the MC. On simple datasets, such as IMDB, it suffices to set $\kappa = 1$. More complex datasets

Algorithm 1 Maximal Clique Decoder

Input: Probability vector $\boldsymbol{p}$, graph G with n nodes, number of samplers κ, sample length τ.
Output: Predicted maximal clique $\hat{C}$ of size $|\hat{C}|$.
Order nodes via permutation $\pi : V \to [n]$ such that $\boldsymbol{p}_{\pi^{-1}(1)} \geq \boldsymbol{p}_{\pi^{-1}(2)} \geq \cdots \geq \boldsymbol{p}_{\pi^{-1}(n)}$.
for $j = 1$ **to** κ **do**
  $\hat{C}_j \leftarrow \{\pi^{-1}(j)\}$
  for $i = 2$ **to** τ **do**
    if $\hat{C}_j \cup \{\pi^{-1}(j+i)\}$ forms a clique **then**
      $\hat{C}_j \leftarrow \hat{C}_j \cup \{\pi^{-1}(j+i)\}$
    end if
  end for
end for
$\hat{\Omega} \leftarrow \{\hat{C}_1, \ldots, \hat{C}_\kappa\}$
$\hat{C} \leftarrow \arg\max_{C \in \hat{\Omega}} |C|$

such as Twitter require larger values of κ. Importantly, we note that the $\hat{C}_j$ can be computed in parallel, so increasing the value of κ does not create major scalability issues. We note that setting the hyperparameters of the decoder requires some prior knowledge of the largest sizes of the maximal cliques present in the data, which we tune using the validation set.

4 Results

We empirically experiment on three popular graph learning baselines, namely IMDB, COLLAB [Yanardag and Vishwanathan, 2015] and TWITTER [Yan et al., 2008]. We compare both against other neural network based methods and against an integer-programming-based solver. For the the neural networks, Erdös' GNN [Karalias and Loukas, 2020] and RUN-CSP [Toenshoff et al., 2019], the authors provide two implementations, one optimized towards computational speed (fast), and the other optimized for best approximation score. The integer-programming-based solver, GUROBI 9.0[Gurobi Optimization, LLC, 2022], is in general more accurate than neural network based methods. However, it is slower on larger graphs. In order to highlight the contribution of our hybrid, scattering model, we also provide results for a pure lowpass model, which only uses GCN-type layers [Kipf and Welling, 2016] followed by our decoder module. We provide our results in Tab. 1, where we track each model's test approximation score and the average time needed to classify one graph of the dataset. Dataset statistics can be found in Tab. 2. For a given graph, the approximation score is computed by dividing the size of the clique found by the algorithm by the true MC. So if the MC of a graph has size 10 and the algorithm returns a clique of size 9, it would be given a score of 90%. We then average these scores over all graphs in each dataset.

For the IMDB dataset, our hybrid approach and most other methods (except RUN-CSP) solve the MC problem with a perfect (100%) approximation score. However, our model significantly outperforms the other neural network-based models (Erdös' GNN and RUN-CSP) with respect to computation time (although the highly optimized GUROBI solver is slightly faster than our method on this dataset due to the small graph size). On COLLAB, our model almost reaches perfect performance (99.8%) in 0.035 S/G, outperforming the other neural network based methods in both speed and approximation score. Some versions of the Gurobi solver are able to achieve 99.9% and 100% approximation scores, however, they are slower than our method on this dataset. Lastly, we experimented on the TWITTER dataset, the most challenging of the datasets considered here. Among neural networks, RUN-CSP (accurate) does exceptionally well at 98.7% approximation score our method is second best with 95%, while running faster than all compared neural networks, in particular, 3 times faster than RUN-CSP.

Figure 3 illustrates the differences between our hybrid scattering model and the low-pass GCN-based model[Kipf and Welling, 2016] and helps explain why the scattering-based model performs better across all three datasets. It shows that the probability vector $\boldsymbol{p}$ generated by our hybrid scattering model is a more discriminative representation than the one produced by the low-pass model. When using the low-pass model, most nodes are assigned high probabilities (marked in red). The decoder cannot tell which nodes have higher priorities given such a smooth output. This leads to the decoder accepting nodes which are not part of the MC. If the new clique is not a sub-clique of the MC, this

DATASET	IMDB	COLLAB	TWITTER
SCATTERINGCLIQUE	**<u>1.000</u>** (8E-3)	**<u>0.998</u>** $\pm$ 0.011 (0.035)	**0.950** $\pm$ 0.062(0.12)
GCN (LOWPASS)	0.956 $\pm$ 0.109 (4E-3)	0.981 $\pm$ 0.085 (0.033)	0.887 $\pm$ 0.111(0.11)
ERDŐS (FAST)	**1.000** (0.08)	0.982 $\pm$ 0.063 (0.10)	0.924 $\pm$ 0.133 (0.17)
ERDŐS (ACCURATE)	1.000 (0.10)	**0.990** $\pm$ 0.042 (0.15)	0.942 $\pm$ 0.111 (0.42)
RUN-CSP (FAST)	0.823 $\pm$ 0.191 (0.11)	0.912 $\pm$ 0.188 (0.14)	0.909 $\pm$ 0.145 (0.21)
RUN-CSP (ACCURATE)	0.957 $\pm$ 0.089 (0.12)	0.987 $\pm$ 0.074 (0.19)	**<u>0.987</u>** $\pm$ 0.063 (0.39)
GUROBI 9.0 (0.1S)	1.000 (1E-3)	0.982 $\pm$ 0.101 (0.05)	0.803 $\pm$ 0.258 (0.21)
GUROBI 9.0 (0.5S)	1.000 (1E-3)	0.997 $\pm$ 0.035 (0.06)	0.996 $\pm$ 0.019 (0.34)
GUROBI 9.0 (1S)	1.000 (1E-3)	0.999 $\pm$ 0.015 (0.06)	1.000 (0.34)
GUROBI 9.0 (5S)	1.000 (1E-3)	1.000 (0.06)	1.000 (0.35)

Table 1: Maximal clique test approximation ratio and average prediction time measured in seconds per graph (in brackets) on test graphs for the different baseline methods compared on across datasets. We set the number of samplers, κ, equal to 1, 3 and 10 on IMDB, COLLAB and TWITTER, respectively. We mark the two best neural network models with highest prediction accuracy bold and underlined and the best one. In case of the same performance, we use lower computation time as a tiebreaker. We provide results for GUROBI with 4 different time budgets.

Dataset	# Graphs	Avg. # Nodes	Avg. # Edges	Avg. MC Size
IMDB	$1,000$	19.8 ± 10.0	96.5 ± 105.6	10.2 ± 5.3
COLLAB	$5,000$	74.5 ± 62.3	2457.2 ± 6438.9	41.5 ± 48.5
TWITTER	973	131.8 ± 64.4	1709.3 ± 1559.7	15.2 ± 7.1

Table 2: Dataset statistics of IMDB, COLLAB and TWITTER.

in turn can lead to the decoder rejecting nodes which *are* part of the MC. The scattering model on the other hand produces a less smooth representation. The nodes outside the MC are assigned lower probabilities so that our decoder can successfully find the MC. This is consistent with previous work [Wenkel et al., 2022] showing that the graph scattering transform helps overcome the oversmoothness problem in node classification. We emphasize that both of these models are trained with the same loss function and the same rule-based decoder to predict the maximal clique. Therefore, the differences illustrated in Figure 3 are the direct result using scattering. More comparison between the scattering model and low-pass model can be found in the appendix.

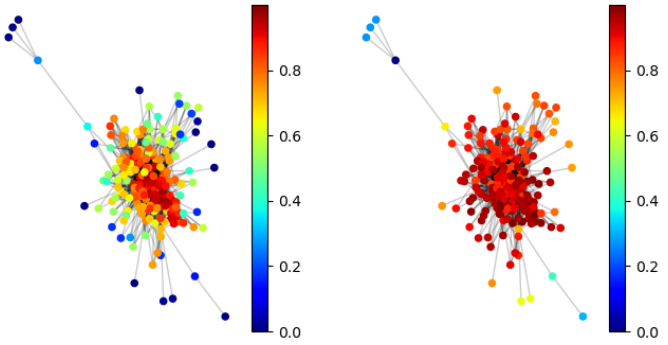

Figure 3: Comparison of the output probability vector $\boldsymbol{p}$ for our hybrid scattering model (left) and the low-pass model (right) on a graph taken from the TWITTER dataset with ground truth maximal clique size of 10. Our hybrid scattering model yields followed by our decoder yields the correct clique, while the lowpass model only concludes with a clique of size 8.

DATASET	EASY	MEDIUM	HARD
SCATTERINGCLIQUE	0.993 ±0.017 (**0.050**)	0.935 ± 0.102 (**0.021**)	0.846 ± 0.177 (**0.184**)
GUROBI 9.0 (0.1S)	0.991 (0.42)	0.806 ± 0.101 (0.41)	0.742 ± 0.258 (0.42)
GUROBI 9.0 (0.2S)	0.998 (0.49)	0.878 ± 0.035 (0.51G)	0.843 ± 0.019 (0.48)
GUROBI 9.0 (0.5S)	1.000 (0.55)	0.977 ± 0.015 (0.68)	0.962 (0.60)
GUROBI 9.0 (1S)	1.000 (0.65)	0.991 (0.91)	0.989 (0.70)

Table 3: Maximal clique test approximation ratio and average prediction time (measured in seconds per graph) on test graphs for our hybrid scattering model compared to the GUROBI solver with different time budgets. We use 1 sampler for the easy and medium difficulty and 10 samplers on hard dataset. Fastest computation time is marked in bold, while we underline the GUROBI solver with lowest time budget that outperforms our model.

To further explore how our model is compares to the current state-of-the-art commercial solver GUROBI 9.0, we introduce a new dataset that contains three classes of graphs with different difficulties (easy, medium and hard) for the maximal clique prediction. The dataset generation is based on the approach in Xu et al. [2005], where we set the hardness parameter to 0.2, 0.5 and 0.8, respectively, to generate the easy, medium and hard cases, as shown in Table 4. Each hardness class case contain 1,000 graphs. We notice that our model outperforms GUROBI under limited time budget. The results are shown in Table 3. Notably, the scattering model is very efficient when dealing with medium hardness cases, where our model is 20 times faster than GUROBI when achieving similar approximation score.

Dataset	# Graphs	Avg # Nodes	Avg # Edges	Avg. MC Size
Easy	$1,000$	209.0 ± 49.9	2293.6 ± 1147.8	16.0 ± 3.6
Medium	$1,000$	187.6 ± 62.7	3378.2 ± 2077.9	17.7 ± 8.2
Hard	$1,000$	181.3 ± 62.2	4165.3 ± 2780.2	21.0 ± 9.0

Table 4: Dataset statistics of easy, medium and hard cases.

Overall, we remark that our hybrid model provides considerable speedup compared to both the other neural networks and to GUROBI for large graphs. We also note that our model takes fewer parameters than the Erdös' GNN model, which uses MLPs with 512 hidden units for each layer, while in our model, whereas the MLPs used in our method have either 8, 16, or 20 hidden units on IMDB, COLLAB, and TWITTER, respectively. This in turn suggests that the use of both GCN and scattering filters leads to a powerful feature extractors that help learn informative node representations with relatively few parameters.

5 Conclusion

In this paper, we show the expressive power of GNNs can be critical for solving graph CO problems. Our results suggest that low-pass models that generally enforce smoothness over graph neighborhoods due to oversmoothing, make the nodes indistinguishable, which leads to unfavorable node representations for the MC problem. Inspired by the geometric scattering transform, we propose a novel scattering-based GNN to overcome the oversmoothing problem. We further construct a novel two-term loss function which encourages our network both to find a large set of nodes and to find a set of nodes which are contained within a clique, which, in conjunction mimic the otherwise hard to optimize problem of finding maximal cliques. Our scattering model is able to generate efficient representations that enable us to learn probabilities for each node to be part of the maximal clique. Based on these node representations we approximate the correct maximal cliques using a constraint-preserving greedy decoder. Overall, our performance is competitive with other neural networks-based methods and commercial solvers in terms of time and accuracy.

References

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008.

Marco Gori, Gabriele Monfardini, and Franco Scarselli. A new model for learning in graph domains. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 2, pages 729–734. IEEE, 2005.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International conference on machine learning*, pages 1263–1272. PMLR, 2017.

Zhuwen Li, Qifeng Chen, and Vladlen Koltun. Combinatorial optimization with graph convolutional networks and guided tree search. *arXiv preprint arXiv:1810.10659*, 2018a.

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.

Chaitanya K Joshi, Thomas Laurent, and Xavier Bresson. An efficient graph convolutional network technique for the travelling salesman problem. *arXiv preprint arXiv:1906.01227*, 2019.

Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *Thirty-Second AAAI conference on artificial intelligence*, 2018b.

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.

Alex Grossmann. Wavelet transforms and edge detection. In *Stochastic processes in physics and engineering*, pages 149–157. Springer, 1988.

David I Shuman, Benjamin Ricaud, and Pierre Vandergheynst. Vertex-frequency analysis on graphs. *Applied and Computational Harmonic Analysis*, 40(2):260–291, 2016.

Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. *Advances in neural information processing systems*, 29, 2016.

Yimeng Min, Frederik Wenkel, and Guy Wolf. Scattering gcn: Overcoming oversmoothness in graph convolutional networks. *Advances in Neural Information Processing Systems*, 33, 2020.

Fernando Gama, Alejandro Ribeiro, and Joan Bruna. Diffusion scattering transforms on graphs. In *International Conference on Learning Representations*, 2019.

Dongmian Zou and Gilad Lerman. Graph convolutional neural networks via scattering. *Applied and Computational Harmonic Analysis*, 49(3)(3):1046–1074, 2019.

Feng Gao, Guy Wolf, and Matthew Hirn. Geometric scattering for graph data analysis. In *Proceedings of the 36th International Conference on Machine Learning, PMLR*, volume 97, pages 2122–2131, 2019.

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

Pinar Yanardag and SVN Vishwanathan. Deep graph kernels. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1365–1374, 2015.

Xifeng Yan, Hong Cheng, Jiawei Han, and Philip S Yu. Mining significant graph patterns by leap search. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 433–444, 2008.

Jan Toenshoff, Martin Ritzert, Hinrikus Wolf, and Martin Grohe. Run-csp: Unsupervised learning of message passing networks for binary constraint satisfaction problems. *arXiv preprint arXiv:1909.08387*, 2019.

Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2022. URL `https://www.gurobi.com`.

Ke Xu, Fr´ ed´ eric Boussemart, Fred Hemery, and Christophe Lecoutre. A simple model to generate hard satisfiable instances. In *Proceedings of the 19th international joint conference on Artificial intelligence*, pages 337–342, 2005.

A Checklist

1. Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? Yes
2. Have you read the ethics review guidelines and ensured that your paper conforms to them? Yes
3. Did you discuss any potential negative societal impacts of your work? No
4. Did you describe the limitations of your work? No
5. Did you state the full set of assumptions of all theoretical results? Yes
6. Did you include complete proofs of all theoretical results? Yes
7. Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? We will include a link to an anonymous github repository in the supplementary material.
8. Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? Yes
9. Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? Yes, we report the standard deviation of our model.
10. Did you include the amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? No