FoSR: First-order Spectral Rewiring for addressing Oversquashing in GNNs

Anonymous authors
Paper under double-blind review

Abstract

Graph neural networks (GNNs) are able to leverage the structure of graph data by passing messages along the edges of the graph. While this allows GNNs to learn features depending on the graph structure, for certain graph topologies it leads to inefficient information propagation and a problem known as oversquashing. This has recently been linked with the curvature and spectral gap of the graph. On the other hand, adding edges to the message-passing graph can lead to increasingly similar node representations and a problem known as oversmoothing. We propose a computationally efficient algorithm that prevents oversquashing by systematically adding edges to the graph based on spectral expansion. We combine this with a relational architecture, which lets the GNN preserve the original graph structure and provably prevents oversmoothing. We find experimentally that our algorithm outperforms existing graph rewiring methods in several graph classification tasks.

1 Introduction

Graph neural networks (GNNs) (Gori et al., 2005; Scarselli et al., 2008) are a broad class of models which process graph-structured data by passing messages between nodes of the graph. Due to the versatility of graphs, GNNs have been applied to a variety of domains, such as chemistry, social networks, knowledge graphs, and recommendation systems (Zhou et al., 2020; Wu et al., 2020). GNNs broadly follow a message-passing framework, meaning that each layer of the GNN aggregates the representations of a node and its neighbors, and transforms these features into a new representation for that node. The aggregation function used by the GNN layer is taken to be locally permutation-invariant, since the ordering of the neighbors of a node is arbitrary, and its specific form is a key component of the GNN architecture; varying it gives rise to several common GNN variants (Kipf and Welling, 2017; Veličković et al., 2018; Li et al., 2015; Hamilton et al., 2017; Xu et al., 2019). The output of a GNN can be used for tasks such as graph classification or node classification.

Although GNNs are successful in computing dependencies between nodes of a graph, they have been found to suffer from a limited capacity to capture long-range interactions. For a fixed graph, this is caused by a variety of problems depending on the number of layers in the GNN. Since graph convolutions are local operations, a GNN with a small number of layers can only provide a node with information from nodes close to itself. For a GNN with l layers, the receptive field of a node (the set of nodes it receives messages from) is exactly the ball of radius l about the node. For small values of l, this results in "underreaching", and directly limits which functions the GNN can represent. On a related note, the functions representable by GNNs with l layers are limited to those computable by l steps of the Weisfeiler-Lehman (WL) graph isomorphism test (Morris et al., 2019; Xu et al., 2019; Barceló et al., 2020). On the other hand, increasing the number of layers leads to its own set of problems. In contrast to other architectures that benefit from the expressivity of deeper networks, GNNs experience a decrease in accuracy as the number of layers increases (Li et al., 2018; Chen et al., 2020). This phenomenon has partly been attributed to "oversmoothing", where repeated graph convolutions eventually render node features indistinguishable (Li et al., 2018; Oono and Suzuki, 2020; Cai and Wang, 2020; Zhao and Akoglu, 2020; Rong et al., 2020; Di Giovanni et al., 2022).

Separate from oversmoothing is the problem of "oversquashing" first pointed out by Alon and Yahav (2021). As the number of layers of a GNN increases, information from (potentially) exponentially-growing receptive fields need to be concurrently propagated at each message-passing step. This leads to a bottleneck that causes oversquashing, when an exponential amount of information is squashed

into fixed-size node vectors (Alon and Yahav, 2021). Consequently, for prediction tasks relying on long-range interactions, the GNN can fail. Oversquashing usually occurs when there are enough layers in the GNN to reach any node (the receptive fields are large enough), but few enough that the GNN cannot process all of the necessary relations between nodes. Hence, for a fixed graph, the problems of underreaching, oversquashing, and oversmoothing occur in three different regimes, depending on the number of layers of the GNN.

A common approach to addressing oversquashing is to *rewire* the input graph, making changes to its edges so that it has fewer structural bottlenecks. A simple approach to rewiring is to make the last layer of the GNN fully adjacent, allowing all nodes to interact with one another (Alon and Yahav, 2021). Alternatively, one can make changes to edges of the input graph, feeding the modified graph into all layers of the GNN (Topping et al., 2022; Banerjee et al., 2022). The latter approaches can be viewed as optimizing the spectral gap of the input graph for alleviating structural bottlenecks and improving the overall quality of signal propagation across nodes (see Figure 1).

While these rewiring methods improve the connectivity of the graph, there are drawbacks to making too many modifications to the input. The most obvious problem is that we are losing out on topological information about the original graph. If the structure of the original graph is indeed relevant, adding and removing edges diminishes that benefit to the task. Another issue arises from the smoothing effects of adding edges: If we add too many edges to the input graph, an ordinary GCN will suffer from oversmoothing (Li et al., 2018). In other words, if we use this natural approach to rewiring, we experience a trade-off between oversquashing and oversmoothing. This observation, which does not seem to have been pointed out in earlier works, is the main motivation for the approach that we develop in this work.

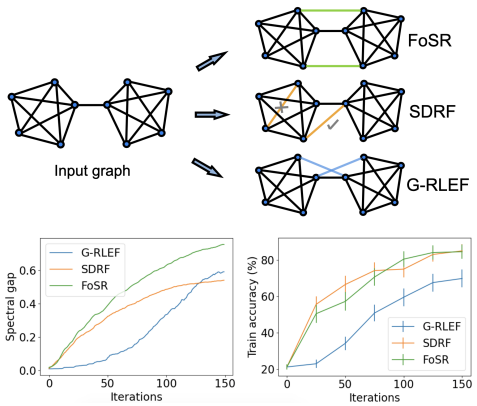

Figure 1: Top: Schematic showing different rewiring methods, FoSR (ours), SDRF (Topping et al., 2022), and G-RLEF (Banerjee et al., 2022) for alleviating structural bottlenecks in the input graph. Our method adds new edges that are labeled differently from the existing ones so that the GNN can distinguish them in training. Bottom: Normalized spectral gap and training accuracy as functions of the number of rewiring iterations for a learning task modeled on the NEIGHBORSMATCH problem for a path-of-cliques input (for details, see Appendix B).

1.1 MAIN CONTRIBUTIONS

This paper presents a new framework for rewiring a graph to reduce oversquashing in GNNs while preventing oversmoothing. Here are our main contributions:

- We introduce a framework for graph rewiring which can be used with any rewiring method that sequentially adds edges. In contrast to previous approaches that only modify the input graph (e.g., Topping et al., 2022; Banerjee et al., 2022; Bober et al., 2022), our solution gives special labels to the added edges. We then use a relational GNN on this new graph, with the relations corresponding to whether the edge was originally in the input graph or added during the rewiring. This allows us to preserve the input graph topology while using the new edges to improve its connectivity. In Theorem 3 we show that this approach also prevents oversmoothing.
- We introduce a new rewiring method, FoSR (First-order Spectral Rewiring) aimed at optimizing the spectral gap of the graph input to the GNN (Algorithm 1). This algorithm computes the first-order change in the spectral gap from adding each edge, and then adds the edge which maximizes this (Theorem 4 and Proposition 5).
- We empirically demonstrate that the proposed method results in faster spectral expansion (a marker of reduced oversquashing) and improved test accuracy against several baselines on several graph classification tasks (see Table 1). Experiments demonstrate that the relational structure preserving the original input graph significantly boosts test accuracy.

1.2 RELATED WORKS

Past approaches to reducing oversquashing have hinged upon choosing a measure of oversquashing, and modifying the edges of the graph to minimize it. Topping et al. (2022) argue that negatively curved edges are responsible for oversquashing drawing on curvature notions from Forman (2003) and Ollivier (2009). They introduce a rewiring method known as *stochastic discrete Ricci Flow* (SDRF), which aims to increase the balanced Forman curvature of negatively curved edges by adding new edges. Bober et al. (2022) extend this line of investigation by considering the same type of rewiring but using different notions of discrete curvature. Banerjee et al. (2022) approach oversquashing from an information-theoeretic viewpoint, measuring it in terms of the spectral gap of the graph and demonstrate empirically that this can increase accuracy for certain graph classification tasks. They propose a rewiring algorithm *greedy random local edge flip* (G-RLEF) motivated by an expander graph construction employing an effective resistance (Lyons and Peres, 2017) based edge sampling strategy. The work of Alon and Yahav (2021) first pointing at oversquashing also introduced an approach to rewiring, where they made the last GNN layer an expander – the complete graph that allows every pair of nodes to connect to each other. They also experimented with making the last layer partially adjacent (randomly including any potential edge). This can be thought of as a form of spectral expansion in the final layer since random graphs have high spectral gap (Friedman, 1991). In contrast to these works, our method gives a practical way of achieving the largest possible increase in the spectral graph with the smallest possible modification of the input graph and in fact preserving the input graph topology via a relational structure.

Although not as closely related, we find it worthwhile also pointing at following works in this general context. Prior to the diagnosis of the oversquashing problem, Klicpera et al. (2019) used graph diffusion to rewire the input graph, improving long-range connectivity for the GNN. Rewiring can also be performed while training a GNN. Arnaiz-Rodríguez et al. (2022) use first-order spectral methods to define a loss function depending on the adjacency matrix, allowing a GNN to learn a rewiring that alleviates oversquashing. We should mention that aside from rewiring the input graph, some works pursue different approaches to solve oversquashing, such as creating positional embeddings for the nodes or edges inspired by the transformer architecture (Vaswani et al., 2017). The most direct generalization of this approach to graphs is using Laplacian embeddings (Kreuzer et al., 2021; Dwivedi and Bresson, 2020). Brüel-Gabrielsson et al. (2022) combine this with adding neighbors to encode the edges which are the result of multiple hops.

2 PRELIMINARIES

2.1 BACKGROUND ON SPECTRAL GRAPH THEORY

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{R})$ be a graph with node set $\mathcal{V}$, $|\mathcal{V}| = n$, edge set $\mathcal{E}$, $|\mathcal{E}| = m$, and relation set $\mathcal{R}$. The set $\mathcal{R}$ is a finite set of relation types, and elements $(u, v, r) \in \mathcal{E}$ consist of a pair of nodes $u, v \in \mathcal{V}$ together with an associated relation type $r \in \mathcal{R}$. When the relation type of an edge is not relevant, we will simply write (u, v) for an edge. For each $v \in \mathcal{V}$ we define $\mathcal{N}(v)$ to consist of all neighbors of v, that is all $u \in \mathcal{V}$ such that there exists an edge $(u, v) \in \mathcal{E}$. For each $r \in \mathcal{R}$ and $v \in \mathcal{V}$, we define $\mathcal{N}_r(v)$ to consist of all neighbors of v of relation type r. The degree d_v of a node $v \in \mathcal{V}$ is the number of neighbors of v. We define the adjacency matrix $A = A(\mathcal{G})$ by $A_{ij} = 1$ if $(i, j) \in \mathcal{E}$, and $A_{ij} = 0$ otherwise. Let $D = D(\mathcal{G})$ denote the diagonal matrix of degrees given by $D_{ii} = d_i$. The normalized Laplacian $L = L(\mathcal{G})$ is defined as $L = I - D^{-1/2}AD^{-1/2}$. We will often add self-loops (edges (i, i) for $i \in \mathcal{V}$) to the graphs we consider, so we define augmented versions of the above matrices corresponding to graphs with self-loops added. If $\mathcal{G}$ is a graph without self-loops, we define its augmented adjacency matrix $\tilde{A} := I + A$, its augmented degree matrix $\tilde{D} = I + D$, and its augmented Laplacian $\tilde{L} = I - \tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}$.

We denote the eigenvalues of the normalized Laplacian L by $0 = \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n \leq 2$. Let $\mathbf{1}$ denote the constant function which assumes the value 1 on each node. Then $D^{1/2}\mathbf{1}$ is an eigenfunction of L with eigenvalue 0. The *spectral gap* of $\mathcal{G}$ is $\lambda_2 - \lambda_1 = \lambda_2$. We say that $\mathcal{G}$ has good spectral expansion if it has a large spectral gap. In Appendix A, we review the relation between the spectral gap and a related measure of graph expansion, the Cheeger constant.

2.2 BACKGROUND ON RELATIONAL GNNS

The framework we propose fundamentally relies on relational GNNs (R-GNNs) (Battaglia et al., 2018), so we review their formulation here. We define a general R-GNN layer by

$$h_v^{(k+1)} = \phi_k\Big(h_v^{(k)}, \sum_{r\in\mathcal{R}}\sum_{u\in\mathcal{N}_r(v)} \psi_{k,r}(h_u^{(k)}, h_v^{(k)})\Big),$$

where the $\psi_{k,r} : \mathbb{R}^{d_k} \times \mathbb{R}^{d_k} \to \mathbb{R}^{d_{k+1}}$ are *message-passing functions*, and the $\phi_k : \mathbb{R}^{d_{k+1}} \to \mathbb{R}^{d_{k+1}}$ are *update functions*. The ϕ_k and $\psi_{k,r}$ can either be fixed mappings or learned during training. All GNN layer types we consider will be special cases of this general R-GNN layer.

As a special case of R-GNNs, R-GCN layers are defined by the update

$$h_v^{(k+1)} = \sigma\Big(W^{(k)}h_v^{(k)} + \sum_{r\in\mathcal{R}}\sum_{u\in\mathcal{N}_r(v)} \tfrac{1}{c_{u,v}} W_r^{(k)} h_u^{(k)}\Big),$$

where σ is a nonlinear activation, $c_{u,v}$ is a normalization factor, and $W^{(k)}$, $W_r^{(k)}$ are relation-specific learned linear maps. One can interpret the $Wh_v^{(k)}$ term as adding self-loops to the original graph, and encoding these self-loops as their own relation. We often take $\sigma = \mathrm{ReLU}$, and $c_{u,v} = \sqrt{(1+d_u)(1+d_v)}$.

Another specialization of R-GNNs are graph convolutional networks (GCNs), defined by

$$h_v^{(k+1)} = \sigma\Big(\sum_{u\in\mathcal{N}(v)\cup\{v\}} \tfrac{1}{c_{u,v}} W^{(k)} h_u^{(k)}\Big).$$

3 RELATIONAL REWIRING OF GNNS

We introduce a graph rewiring framework to improve connectivity while retaining the original graph via a relational structure, which demonstrably allows us to also control the *rate of smoothing*. The rate of smoothing measures the similarity of neighboring node features in the GNN output graph. Adding separate weights for the rewired edges allows the network more flexibility in learning an appropriate rate of smoothing. Our main result in this section, Theorem 3 makes this precise.

3.1 RELATIONAL REWIRING

We incorporate relations into our architecture in the following way. Suppose that we rewire $\mathcal{G} = (\mathcal{V}, \mathcal{E}_1)$ by adding edges, yielding a rewired graph $\mathcal{G}' = (\mathcal{V}, \mathcal{E}_1 \cup \mathcal{E}_2)$. We equip $\mathcal{G}'$ with a relational structure by assigning each edge in $\mathcal{E}_1$ the edge type 1, and each edge in $\mathcal{E}_2$ the edge type 2. This serves to provide the GNN with more flexibility, since it remembers both the original structure of the graph and the rewired structure. In the worst case scenario, if we add too many edges, the GNN may counterbalance this by setting most of the weights $W_2^{(k)}$ to be 0, focusing its attention on the original graph. In a more realistic scenario, the GNN can use the original edge weights $W_1^{(k)}$ to compute important structural features of the graph, and the weights $W_2^{(k)}$ simply to transmit information along the graph. Since the weights assigned to the original and rewired edges are separate, their purposes can be served in parallel without sacrificing the topology of the graph. Finally, this R-GCN model allows us to better regulate the rate of smoothing as we demonstrate next.

3.2 RATE OF SMOOTHING FOR R-GCN WITH REWIRING

In this section, we analyze the smoothing effects of repeatedly applying R-GCN layers with rewiring. Here, we consider R-GCN layers $\varphi : \mathbb{R}^{n\times p} \to \mathbb{R}^{n\times p}$ of the form

$$\varphi(X) = X\Theta + \sum_{r\in\mathcal{R}} D^{-1/2}A_r D^{-1/2} X\Theta_r, \tag{1}$$

where $\Theta, \Theta_r \in \mathbb{R}^{p\times p}$ are weight matrices, A_r is the adjacency matrix of the r-th relation, and D is the matrix of degrees of $\mathcal{G}$. For a vanilla GCN without relations or residual connections, one problem with rewiring the graph is that adding too many edges will result in oversmoothing (Di Giovanni et al., 2022). Indeed, adding edges to a graph will by definition increase its Cheeger constant. For input graphs with a high spectral gap (and hence high Cheeger constant), a GCN will produce similar

representations for each node (Xu et al., 2018; Oono and Suzuki, 2020). As an extreme case, if we replace the graph with a complete graph, the GCN layer will assign each node the same representation. We will show that this does not happen for an R-GCN layer, allowing us to add more edges without the adverse side effect of oversmoothing.

Definition 1. Let $\mathcal{G}$ be a connected graph with adjacency matrix A and normalized Laplacian L. For $i \in \mathcal{V}$, let d_i denote the degree of node i. Given a scalar field $f \in \mathbb{R}^n$, its *Dirichlet energy* with respect to $\mathcal{G}$ is defined as

$$\mathscr{E}(f) := \tfrac{1}{2}\textstyle\sum_{i,j} A_{i,j}\left(\frac{f_i}{\sqrt{d_i}} - \frac{f_j}{\sqrt{d_j}}\right)^2 = f^T L f.$$

For a vector field $X \in \mathbb{R}^{n\times p}$, we define

$$\mathscr{E}(X) := \tfrac{1}{2}\textstyle\sum_{i,j,k} A_{i,j}\left(\frac{X_{i,k}}{\sqrt{d_i}} - \frac{X_{j,k}}{\sqrt{d_j}}\right)^2 = \operatorname{Tr}(X^T L X).$$

The Dirichlet energy of a unit-norm function on the nodes of a graph is a measure of how "non-smooth" it is (Chung and Graham, 1997). The Dirichlet energy of a function f is small when $f_i/\sqrt{d_i}$ is close in value to $f_j/\sqrt{d_j}$ for all edges (i, j). In the case where $\mathcal{G}$ is d-regular (all nodes have degree d), this reduces to the familiar notion of adjacent nodes having representations close in value. Hence, we make the following definition:

Definition 2. Let $\mathcal{G}$ be a graph and $\varphi : \mathbb{R}^{n\times p} \to \mathbb{R}^{n\times p}$ be a mapping. We define the *rate of smoothing* of φ with respect to $\mathcal{G}$ as

$$RS_{\mathcal{G}}(\varphi) := 1 - \left(\frac{\sup_{X:\mathscr{E}(X)\neq 0} \mathscr{E}(\varphi(X))/\mathscr{E}(X)}{\sup_{X:X\neq 0} \|\varphi(X)\|_F^2/\|X\|_F^2}\right)^{1/2}.$$

The numerator of the fraction above indicates the rate of decay (or expansion) of the Dirichlet energy upon applying φ. The denominator is a normalization factor to account for φ scaling matrices in Frobenius norm. By defining this as a ratio of two norms, we can estimate them separately which gives us a good theoretical handle of smoothing. Note that if φ is linear,

$$RS_{\mathcal{G}}(\varphi) := 1 - \left(\frac{\sup_{\mathscr{E}(X)=1} \mathscr{E}(\varphi(X))}{\sup_{\|X\|_F=1} \|\varphi(X)\|_F^2}\right)^{1/2},$$

since $\|\cdot\|_F^2$ and $\mathscr{E}$ are quadratic forms. The following theorem shows that R-GCN layers are flexible in their ability to choose an appropriate rate of smoothing for a graph:

Theorem 3. *Let $\mathcal{G}_1 = (\mathcal{V}, \mathcal{E}_1)$ be a graph and $\mathcal{G}_2 = (\mathcal{V}, \mathcal{E}_1 \cup \mathcal{E}_2)$ be a rewiring of $\mathcal{G}_1$. Consider an R-GCN layer φ defined as in (1), with relations $r_1 = \mathcal{E}_1$, $r_2 = \mathcal{E}_2$. Then for any $\lambda \in [0, \lambda_2(L(\mathcal{G}_2))]$, there exist values of $\Theta, \Theta_1, \Theta_2$ for which φ smooths with rate $RS_{\mathcal{G}_2}(\varphi) = \lambda$ with respect to $\mathcal{G}_2$.*

Proof. The map φ is given by

$$\varphi(X; \Theta, \Theta_1, \Theta_2) = X\Theta + D^{-1/2}A_1D^{-1/2}X\Theta_1 + D^{-1/2}A_2D^{-1/2}X\Theta_2.$$

Here D is the degree matrix of $\mathcal{G}_2$. Fix $\alpha \in [0, 1]$, and take $\Theta_1 = \Theta_2 = \alpha I$ and $\Theta = I$. Then

$$\begin{aligned}\varphi(X) &= X + \alpha D^{-1/2}A_1D^{-1/2}X + \alpha D^{-1/2}A_2D^{-1/2}X \\ &= X + \alpha D^{-1/2}AD^{-1/2}X = (1-\alpha)X + \alpha D^{-1/2}AD^{-1/2}X = I - \alpha L,\end{aligned}$$

where A is the adjacency matrix of $\mathcal{G}_2$, and L is the normalized Laplacian of $\mathcal{G}_2$. We will show that φ smooths with rate $\alpha\lambda_2(L)$. Let $L = U\Sigma U^{-1}$ be an orthogonal diagonalization of L. Let $\lambda_1, \ldots, \lambda_n$ be the eigenvalues of L in ascending order and recall that $0 \le \lambda_i \le 2$ for all i with $\lambda_1 = 0$. We have

$$\begin{aligned}\mathscr{E}(\varphi(X)) &= \operatorname{Tr}(X^TL(I-\alpha L)^2X) = \operatorname{Tr}(X^TU\Sigma(I-\alpha\Sigma)^2U^TX) = \textstyle\sum_{i,j}\lambda_i(1-\alpha\lambda_i)^2((U^TX)_{i,j})^2 \\ &\overset{(a)}{\le} \textstyle\sum_{i,j}\lambda_i(1-\alpha\lambda_2)^2((U^TX)_{i,j})^2 = (1-\alpha\lambda_2)^2\operatorname{Tr}(X^TU\Sigma U^TX) = (1-\alpha\lambda_2)^2\mathscr{E}(X),\end{aligned}$$

with equality in (a) when $(U^T X)_{i,j} = 0$ for $i \neq 2$. Hence,

$$\sup_{\mathscr{E}(X)\neq 0} \frac{\mathscr{E}(\varphi(X))}{\mathscr{E}(X)} = (1-\alpha\lambda_2)^2. \tag{2}$$

Next, we compute

$$\begin{aligned}\|\varphi(X)\|_F^2 &= \mathrm{Tr}(X^T(I-\alpha L)^2 X) = \mathrm{Tr}(X^T U(I-\alpha\Sigma)^2 U X) \\ &= \textstyle\sum_{i,j}(1-\alpha\lambda_i)^2((UX)_{i,j})^2 \overset{(b)}{\leq} \sum_{i,j}((UX)_{i,j})^2 = \|UX\|_F^2 = \|X\|_F^2,\end{aligned}$$

with equality in (b) when $(UX)_{i,j} = 0$ for $i \neq 1$. Hence

$$\sup_{X\neq 0} \frac{\|\varphi(X)\|_F^2}{\|X\|_F^2} = 1. \tag{3}$$

Combining (2) and (3) yields $RS(\varphi) = \alpha\lambda_2 = \alpha\lambda_2(L)$. Since this holds for any $\alpha \in [0,1]$, the result follows. □

4 FoSR: First order Spectral Rewiring

In order to use the R-GCN framework, we need to make a choice of rewiring algorithm that is capable of adding edges without removing any. We propose a first-order spectral rewiring algorithm (FoSR) with the goal of improving graph connectivity. The *rate of spectral expansion*, i.e., the rate at which the spectral gap improves as a function of the rewiring iterations is one of the key determinants of oversquashing (Banerjee et al., 2022). Compared to existing approaches such as SDRF (Topping et al., 2022) and G-RLEF (Banerjee et al., 2022), the proposed algorithm has a faster rate of spectral expansion. See Figure 1 for an illustration.

Our algorithm FoSR follows a similar philosophy as Chan and Akoglu (2016) optimizing the spectral gap of the input graph by sequentially adding edges such that the rate of spectral expansion is maximized. At each step, we wish to add the edge that maximizes the spectral gap of the resulting graph. However, computing the spectral gap for each edge addition is expensive as it requires us to compute eigenvalues for $O(n^2)$ different matrices. Instead of computing this quantity directly, we use a first-order approximation of the spectral gap based on matrix perturbation theory. The following theorem determines the first-order change of the eigenvalues of a perturbed symmetric matrix:

Theorem 4. *For symmetric matrices $M \in \mathbb{R}^{n\times n}$ with distinct eigenvalues, the i-th largest eigenvalue λ_i satisfies*

$$\nabla_M \lambda_i(M) = x_i x_i^T,$$

where x_i denotes the (normalized) eigenvector for the i-th largest eigenvalue of M.

We provide a proof in Appendix C. Stated differently, the previous theorem allows us to make the linear approximation

$$\lambda_2(M+\delta M) \approx \lambda_2(M) + \mathrm{Tr}(\nabla\lambda_2^T(\delta M)) = \lambda_2(M) + x_2^T(\delta M)x_2. \tag{4}$$

Let us apply the approximation (4) to the spectrum of a graph $\mathcal{G}$. We wish to sequentially add an edge (u,v) in a way that maximizes the spectral gap of the resulting graph. Equivalently, the second-largest eigenvalue of $D^{-1/2}AD^{-1/2}$ should be minimized, since this matrix is equal to $I-L$. Using Theorem 4 for the matrix $D^{-1/2}AD^{-1/2}$, we obtain the following result.

Proposition 5. *The first-order change in $\lambda_2 = \lambda_2(D^{-1/2}AD^{-1/2})$ from adding the edge (u,v) is*

$$\frac{2x_u x_v}{(\sqrt{1+d_u})(\sqrt{1+d_v})} + 2\lambda_2 x_u^2\left(\frac{\sqrt{d_u}}{\sqrt{1+d_u}} - 1\right) + 2\lambda_2 x_v^2\left(\frac{\sqrt{d_v}}{\sqrt{1+d_v}} - 1\right), \tag{5}$$

where x denotes the second eigenvector of $D^{-1/2}AD^{-1/2}$, and x_u denotes the u-th entry of x.

We provide a proof in Appendix C. We choose to minimize the first term in (5)

$$\frac{2x_u x_v}{\sqrt{(1+d_u)(1+d_v)}}, \tag{6}$$

since this is simpler and cheaper to compute. To see why this approximation is reasonable, note that

$$\frac{1}{(\sqrt{1+d_u})(\sqrt{1+d_v})} \sim (d_u d_v)^{-1/2}$$

as $d_u, d_v \to \infty$, while

$$\frac{\sqrt{d_u}}{\sqrt{1+d_u}} - 1 \sim d_u^{-2}.$$

So if the degrees of the nodes are sufficiently large and comparable in size, the first term in (5) dominates.

For each iteration that it runs, FoSR computes an approximation of the second eigenvector $x \in \mathbb{R}^n$ of $D^{-1/2}AD^{-1/2}$ and chooses an edge (u, v) which minimizes (6). To produce an initial approximation of x, we use power iteration (rather than computing a full eigendecomposition). Let $d \in \mathbb{R}^n$ denote the vector of degrees of $\mathcal{G}$. Let $\sqrt{d}$ denote the entry-wise square root of d, i.e., with components $\sqrt{d_i}$. Recall that $\sqrt{d}$ is the first eigenvector of $D^{-1/2}AD^{-1/2}$. Since x is the second eigenvector of $D^{-1/2}AD^{-1/2}$, we may approximate it by repeatedly applying $D^{-1/2}AD^{-1/2}$ and then subtracting the component in the direction of the first eigenvector. This map is given by

$$x \mapsto D^{-1/2}AD^{-1/2}x - \frac{\langle x, \sqrt{d}\rangle}{2m} d.$$

After applying this map, we normalize x back to a unit vector,

$$x \mapsto \frac{x}{\|x\|}.$$

FoSR operates by repeatedly alternating between adding an edge and computing a new approximation of x via power iteration. Importantly, our algorithm avoids computing a full eigendecomposition of $\mathcal{G}$ by only approximating its second eigenvector. We compute increasingly accurate estimates of this eigenvector using power iteration. The steps of our method are outlined in Algorithm 1.

Algorithm 1 FoSR: First-order Spectral Rewiring

Input: $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, iteration count k, initial number of power iterations r
Output: Rewired graph $\mathcal{G}' = (\mathcal{V}, \mathcal{E}')$
Initialize $x \in \mathbb{R}^n$ arbitrarily
for $i = 1, 2, \cdots, r$ **do**
    $x \leftarrow D^{-1/2}AD^{-1/2}x - \frac{\langle x, \sqrt{d}\rangle}{2m} d$ ▷ Approximate second eigenvector before rewiring
    $x \leftarrow \frac{x}{\|x\|_2}$
end for
for $i = 1, 2, \cdots, k$ **do**
    Add edge (i, j) which minimizes $\frac{x_i x_j}{\sqrt{(1+d_i)(1+d_j)}}$
    $x \leftarrow D^{-1/2}AD^{-1/2}x - \frac{\langle x, \sqrt{d}\rangle}{2m} d$ ▷ Power iteration to update second eigenvector
    $x \leftarrow \frac{x}{\|x\|_2}$
end for

The number of edge additions k and the initial number of power iterations r are hyperparameters. The number of power iterations r only needs to be chosen large enough to produce an initial approximation of the eigenvector x. In practice, we found that taking r between 5 and 10 is sufficient. The proper choice of iteration count k is problem-specific, and can be chosen to be such that the spectral gap increases sufficiently. We tried multiple configurations of k when training models.

Computational complexity Taking advantage of the sparsity of $\mathcal{G}$, FoSR requires $O(m)$ floating-point operations for each power iteration and $O(n^2)$ operations for each edge added. The $O(n^2)$ operations come from having to search over all node pairs (i, j) for the minimal value of $y_i y_j$, where $y_i = x_i/\sqrt{1+d_i}$. This results in a total cost of $O(kn^2)$. However, we can often compute the minimal value of $y_i y_j$ in a faster way, by choosing $i = \arg\min_k y_k$ and $j = \arg\max_k y_k$ if $\min_k y_k \leq 0$ and

the (i, j) chosen in this way is not already an edge. If this is possible, the edge addition only takes $O(m)$ operations. More generally, we can relax the minimization by choosing $i = \arg\min_k y_k$ and $j = \arg\max_{k \notin \mathcal{N}(i)} y_k$ when $\min_k y_k \leq 0$. We can use a similar relaxation when $\min_k y_k > 0$; take $i = \arg\min_k y_k$ and $j = \arg\min_{k \notin \mathcal{N}(i)} y_k$. If we use the above relaxations, the total cost is $O(km)$.

5 Experiments

We compare our rewiring methods to existing ones to demonstrate the efficacy of relational rewiring as well as rewiring informed by spectral expansion on several graph classification tasks.

Datasets We consider graph classification tasks REDDIT-BINARY, IMDB-BINARY, MUTAG, ENZYMES, PROTEINS, COLLAB from the TUDataset (Morris et al., 2020) where the topology of the graphs in relation to the task has been identified to require long-range interactions. While certain node classification tasks have also been considered in previous works, these have been found to be tractable with nearest neighbor information (Brockschmidt, 2020).

Compared methods We focus on approaches that preprocess the input graph by adding edges. Diffusion Improves Graph Learning (DIGL) (Klicpera et al., 2019) is a diffusion-based rewiring scheme that computes a kernel evaluation of the adjacency matrix, followed by sparsification. SDRF (Topping et al., 2022) surgically rewires a graph by adding edges to support other edges with low curvature, which are locations of bottlenecks. For SDRF, we include results for both the original method (configured to only add edges), and our relational method (again only add edges, and include the added edges with their own relation).

Experimental details We test each rewiring algorithm on an R-GCN (Schlichtkrull et al., 2018), with the relations varying depending on the rewiring method. For rewirings which do not explicitly assign relations to edges, we use two relations: one for the default edge type, and one for self-loops. In other words, we simply use the rewired graph and add residual connections to the architecture. For each task, we use the same GNN architecture for all rewiring methods to illustrate how the choice of rewiring affects test performance. That is, we hold the number of layers, hidden dimension, and dropout probability constant across all rewiring methods for a given dataset. The hyperparameter values are reported in Appendix D.1. However, for each dataset, we separately tune all hyperparamters specific to the rewiring method (such as the number of iterations for FoSR and SDRF). We train all models with an Adam optimizer with a learning rate of 10^{-3} and a scheduler which reduces the learning rate by a factor of 10 after 10 epochs with no improvement in validation loss. For FoSR and SDRF, we only tuned the number of rewiring iterations. For DIGL (Klicpera et al., 2019), we tuned the sparsification threshold (ϵ) and teleport probability (α).

For optimizing hyperparamters and evaluating each configuration, we first select a test set consisting of 10 percent of the graphs and a development set consisting of the other 90 percent of the graphs. We determine accuracies of each configuration using 100 random train/validation splits of the development set consisting of 80 and 10 percent of the graphs, respectively. When training the GNNs, we use a stopping patience of 100 epochs based on the validation loss. We evaluate each hyperparameter configuration using the validation accuracy, so the test set is only used once to generate results for the best hyperparamter configurations. For the test results, we calculate 95 percent confidence intervals for the hyperparameters with the best validation accuracy using the 100 runs.

Results Table 1 shows the results of our experiments on various GNN layer types. Shown is the test accuracy. For each rewiring method, using a relational GNN type improves classification accuracy. The boost in accuracy from using an R-GNN is greatest for the rewiring methods which surgically add edges (FoSR and SDRF). Out of all of the rewiring methods, FoSR consistently achieves the best classification accuracy when relational rewiring is used. When relational rewiring is not used, FoSR still outperforms DIGL and SDRF for most datasets. Our results are also very competitive against results reported for the DiffWire methods by Arnaiz-Rodríguez et al. (2022, Table 1), although they are using a different type of architecture that is not directly comparable to preprocessing methods like ours or the SDRF that sequentially adds edges.

Table 1: Results of rewiring methods for GCN and GIN comparing standard and relational. The best results in each setting are highlighted in bold font and best across settings are highlighted red.

GCN

Rewiring	REDDIT-BINARY	IMDB-BINARY	MUTAG	ENZYMES	PROTEINS	COLLAB
None	68.255 ± 0.560	49.770 ± 0.417	72.150 ± 1.246	27.667 ± 0.594	70.982 ± 0.376	33.784 ± 0.249
DIGL	49.980 ± 0.347	$\mathbf{49.910} \pm 0.429$	71.350 ± 1.220	27.517 ± 0.537	70.607 ± 0.373	15.530 ± 0.150
SDRF	68.620 ± 0.434	49.400 ± 0.461	71.050 ± 0.955	$\mathbf{28.367} \pm 0.599$	70.920 ± 0.404	33.448 ± 0.241
FoSR	$\mathbf{70.330} \pm 0.371$	49.660 ± 0.441	$\mathbf{80.000} \pm 0.803$	25.067 ± 0.507	$\mathbf{73.420} \pm 0.414$	$\mathbf{33.836} \pm 0.298$

R-GCN

Rewiring	REDDIT-BINARY	IMDB-BINARY	MUTAG	ENZYMES	PROTEINS	COLLAB
None	49.850 ± 0.333	50.012 ± 0.468	69.250 ± 1.064	28.600 ± 0.605	69.518 ± 0.370	33.602 ± 0.534
DIGL	49.995 ± 0.316	49.670 ± 0.430	73.400 ± 1.024	28.283 ± 0.619	68.232 ± 0.434	16.926 ± 0.735
SDRF	58.620 ± 0.330	53.640 ± 0.532	72.300 ± 1.130	33.483 ± 0.635	69.107 ± 0.387	67.990 ± 0.197
FoSR	$\mathbf{76.590} \pm 0.271$	$\mathbf{64.050} \pm 0.573$	$\mathbf{84.450} \pm 0.774$	$\mathbf{35.633} \pm 0.587$	$\mathbf{73.795} \pm 0.353$	$\mathbf{70.650} \pm 0.246$

GIN

Rewiring	REDDIT-BINARY	IMDB-BINARY	MUTAG	ENZYMES	PROTEINS	COLLAB
None	86.785 ± 0.539	70.180 ± 0.506	77.700 ± 1.838	33.800 ± 0.569	70.804 ± 0.422	72.992 ± 0.196
DIGL	76.035 ± 0.395	64.390 ± 0.463	$\mathbf{79.700} \pm 1.097$	35.717 ± 0.611	70.759 ± 0.395	54.504 ± 0.209
SDRF	86.440 ± 0.301	69.720 ± 0.588	78.400 ± 1.430	$\mathbf{35.817} \pm 0.558$	69.813 ± 0.404	72.958 ± 0.214
FoSR	$\mathbf{87.350} \pm 0.305$	$\mathbf{71.210} \pm 0.469$	78.000 ± 1.131	29.200 ± 0.702	$\mathbf{75.107} \pm 0.417$	$\mathbf{73.278} \pm 0.212$

R-GIN

Rewiring	REDDIT-BINARY	IMDB-BINARY	MUTAG	ENZYMES	PROTEINS	COLLAB
None	87.965 ± 0.288	68.889 ± 0.445	83.050 ± 0.734	39.017 ± 0.595	70.500 ± 0.413	75.544 ± 0.165
DIGL	74.425 ± 0.369	63.930 ± 0.483	81.450 ± 0.759	37.600 ± 0.611	71.312 ± 0.386	54.714 ± 0.212
SDRF	86.825 ± 0.267	70.210 ± 0.411	82.700 ± 0.909	39.583 ± 0.680	70.696 ± 0.416	76.480 ± 0.198
FoSR	$\mathbf{89.665} \pm 0.212$	$\mathbf{71.810} \pm 0.449$	$\mathbf{86.150} \pm 0.761$	$\mathbf{45.550} \pm 0.642$	$\mathbf{74.670} \pm 0.353$	$\mathbf{76.806} \pm 0.230$

6 Conclusions

We proposed an efficient graph rewiring method for preventing oversquashing based on iterative first-order maximization of the spectral gap. Further, we identified a shortcoming of existing rewiring approaches that can cause oversmoothing and propose a relational rewiring method to overcome this with theoretical guarantees. Experiments on several graph classification benchmarks demonstrate that the proposed methods can significantly improve the performance of GNNs. In future work it will be interesting to study the effects of oversmoothing and oversquashing in relation to training dynamics and investigate rewiring strategies that aid training.

Reproducibility Statement The computer implementation of the proposed methods along with scripts to re-run our experiments are made publicly available on `https://anonymous.4open.science/r/FoSR-0CE3/`. The experimental settings are described in detail in Section 5 and Appendix D.1, which also details the compute infrastructure and selected hyperparameter values.

References

N. Alon and V. D. Milman. Eigenvalues, expanders and superconcentrators. In *25th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 320–322, 1984.

U. Alon and E. Yahav. On the bottleneck of graph neural networks and its practical implications. In *International Conference on Learning Representations*, 2021.

A. Arnaiz-Rodríguez, A. Begga, F. Escolano, and N. Oliver. DiffWire: Inductive graph rewiring via the Lovász bound. *arXiv preprint arXiv:2206.07369*, 2022.

P. K. Banerjee, K. Karhadkar, Y. G. Wang, U. Alon, and G. Montúfar. Oversquashing in GNNs through the lens of information contraction and graph expansion. *arXiv preprint arXiv:2208.03471*, 2022. 58th Annual Allerton Conference on Communication, Control, and Computing.

P. Barceló, E. Kostylev, M. Monet, J. Pérez, J. Reutter, and J.-P. Silva. The logical expressiveness of graph neural networks. In *International Conference on Learning Representations*, 2020.

P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

J. Bober, A. Monod, E. Saucan, and K. N. Webster. Rewiring networks for graph neural network training using discrete geometry, 2022. URL `https://arxiv.org/abs/2207.08026`.

M. Brockschmidt. GNN-film: Graph neural networks with feature-wise linear modulation. In *International Conference on Machine Learning*, pages 1144–1152. PMLR, 2020.

R. Brüel-Gabrielsson, M. Yurochkin, and J. Solomon. Rewiring with positional encodings for graph neural networks, 2022. URL `https://arxiv.org/abs/2201.12674`.

C. Cai and Y. Wang. A note on over-smoothing for graph neural networks. *arXiv preprint arXiv:2006.13318*, 2020.

H. Chan and L. Akoglu. Optimizing network robustness by edge rewiring: a general framework. *Data Mining and Knowledge Discovery*, 30(5):1395–1425, 2016.

J. Cheeger. A lower bound for the smallest eigenvalue of the Laplacian. *Problems in Analysis*, 625 (195-199):110, 1970.

D. Chen, Y. Lin, W. Li, P. Li, J. Zhou, and X. Sun. Measuring and relieving the over-smoothing problem for graph neural networks from the topological view. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 3438–3445, 2020.

F. R. Chung and F. C. Graham. *Spectral graph theory*. Number 92. American Mathematical Society, 1997.

F. Di Giovanni, J. Rowbottom, B. P. Chamberlain, T. Markovich, and M. M. Bronstein. Graph neural networks as gradient flows. *arXiv preprint arXiv:2206.10991*, 2022.

V. P. Dwivedi and X. Bresson. A generalization of transformer networks to graphs. *arXiv preprint arXiv:2012.09699*, 2020. AAAI 2021 Workshop on Deep Learning on Graphs: Methods and Applications (DLG-AAAI 2021).

M. Fey and J. E. Lenssen. Fast graph representation learning with Pytorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

R. Forman. Bochner's method for cell complexes and combinatorial Ricci curvature. *Discrete and Computational Geometry*, 29(3):323–374, 2003.

J. Friedman. On the second eigenvalue and random walks in random d-regular graphs. *Combinatorica*, 11(4):331–362, 1991.

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

W. Hamilton, Z. Ying, and J. Leskovec. Inductive representation learning on large graphs. *Advances in Neural Information processing Systems*, 30:1024–1034, 2017.

C. R. Harris, K. J. Millman, S. J. Van Der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, et al. Array programming with NumPy. *Nature*, 585(7825):357–362, 2020.

J. D. Hunter. Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, 9(03): 90–95, 2007.

T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017.

J. Klicpera, S. Weißenberger, and S. Günnemann. Diffusion improves graph learning. *Advances in Neural Information processing Systems*, 33, 2019.

D. Kreuzer, D. Beaini, W. Hamilton, V. Létourneau, and P. Tossou. Rethinking graph transformers with spectral attention. *Advances in Neural Information Processing Systems*, 34:21618–21629, 2021.

Q. Li, Z. Han, and X.-M. Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*, 2015.

R. Lyons and Y. Peres. *Probability on trees and networks*, volume 42. Cambridge University Press, 2017.

C. Morris, M. Ritzert, M. Fey, W. L. Hamilton, J. E. Lenssen, G. Rattan, and M. Grohe. Weisfeiler and Leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 4602–4609, 2019.

C. Morris, N. M. Kriege, F. Bause, K. Kersting, P. Mutzel, and M. Neumann. Tudataset: A collection of benchmark datasets for learning with graphs. In *ICML 2020 Workshop on Graph Representation Learning and Beyond (GRL+ 2020)*, 2020. URL `http://eprints.cs.univie.ac.at/6469/`.

Y. Ollivier. Ricci curvature of Markov chains on metric spaces. *Journal of Functional Analysis*, 256 (3):810–864, 2009.

K. Oono and T. Suzuki. Graph neural networks exponentially lose expressive power for node classification. In *International Conference on Learning Representations*, 2020.

A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems*, 32, 2019.

Y. Rong, W. Huang, T. Xu, and J. Huang. Dropedge: Towards deep graph convolutional networks on node classification. In *International Conference on Learning Representations*, 2020.

F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2008.

M. Schlichtkrull, T. N. Kipf, P. Bloem, R. v. d. Berg, I. Titov, and M. Welling. Modeling relational data with graph convolutional networks. In *European Semantic Web Conference*, pages 593–607. Springer, 2018.

G. Stewart and J. Sun. *Matrix Perturbation Theory*. Computer Science and Scientific Computing. Elsevier Science, 1990.

J. Topping, F. D. Giovanni, B. P. Chamberlain, X. Dong, and M. M. Bronstein. Understanding over-squashing and bottlenecks on graphs via curvature. In *International Conference on Learning Representations*, 2022.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems*, 30, 2017.

P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018.

Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1):4–24, 2020.

K. Xu, C. Li, Y. Tian, T. Sonobe, K.-i. Kawarabayashi, and S. Jegelka. Representation learning on graphs with jumping knowledge networks. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, pages 5453–5462, 2018.

K. Xu, W. Hu, J. Leskovec, and S. Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019.

L. Zhao and L. Akoglu. Pairnorm: Tackling oversmoothing in gnns. In *International Conference on Learning Representations*, 2020.

J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun. Graph neural networks: A review of methods and applications. *AI Open*, 1:57–81, 2020.

APPENDIX

A SPECTRAL GAP AND THE CHEEGER CONSTANT

A classic measure of well-connectedness of a graph is the *isoperimetric ratio* or the *Cheeger constant* (Chung and Graham, 1997)

$$h(\mathcal{G}) := \min_{\varnothing \neq \mathcal{S} \subset \mathcal{V}} \frac{|\partial \mathcal{S}|}{\min\big(\operatorname{vol} \mathcal{S}, \operatorname{vol} \bar{\mathcal{S}}\big)}, \tag{7}$$

where $\operatorname{vol} \mathcal{S} = \sum_{v \in \mathcal{S}} d_v$, $\bar{\mathcal{S}} = \mathcal{V} \setminus \mathcal{S}$ and $\partial \mathcal{S} = \{(u,v) \colon u \in \mathcal{S}, v \in \bar{\mathcal{S}}, (u,v) \in \mathcal{E}\}$ is the edge boundary of $\mathcal{S} \subset \mathcal{V}$. Note that $0 \leq h(\mathcal{G}) \leq 1$ for all $\mathcal{S} \subset \mathcal{V}$. $\mathcal{G}$ is connected if and only if $h(\mathcal{G}) > 0$. The same is true for the spectral gap $\lambda_2(L)$. Intuitively, when $h(\mathcal{G})$ is large, $\mathcal{G}$ has no structural bottlenecks in the sense that every part of $\mathcal{G}$ is connected to the rest of it by a large fraction of its edges.

The discrete Cheeger inequality (Cheeger, 1970; Alon and Milman, 1984) shows that the spectral gap of $\mathcal{G}$ provides an estimate of its Cheeger constant:

$$\frac{\lambda_2}{2} \leq h(\mathcal{G}) \leq \sqrt{2\lambda_2}. \tag{8}$$

In particular, the spectral gap is bounded away from zero if and only if $h(\mathcal{G})$ is bounded away from zero.

B DETAILS ON FIGURE 1

Alon and Yahav (2021) introduced the synthetic benchmark NEIGHBORSMATCH to test the impact of oversquashing in GNNs. Given an input graph $\mathcal{G}$, suppose that we wish to predict the label for a node T; see Figure 2(a). The correct label is the label of the green node that has the same number of blue neighbors as T. For the example in Figure 2(a), the answer is B which happens to reside at the opposite end of the graph. A correct prediction of the label for this example will require a number of GNN layers l that is equal to the diameter of the graph.

We consider a learning task modeled on the NEIGHBORSMATCH problem. Given an input graph $\mathcal{G}$, a target node T, and a subset $\mathcal{S}$ of the nodes of $\mathcal{G}$, we assign each node in $\mathcal{S}$ a different random $|\mathcal{S}|$-dimensional one-hot vector encoding the number of blue neighbors. Likewise, we represent the label of T as a random $|\mathcal{S}|$-dimensional one-hot vector and the goal is to predict the node T' $\in \mathcal{S}$ with the same label as T. We take the input $\mathcal{G}$ to be a path-of-cliques, which comprises of three cliques each of size 10 connected in a path by two edges; see Figure 2(b). The set $\mathcal{S}$ consists of the first 9 nodes in clique-1, and the target T is the last node of clique-3. The range of the interaction, i.e., the maximum distance between T and a node in $\mathcal{S}$ is 5. We trained a graph attention network (Veličković et al., 2018) with 6 layers each of width 64 on a training dataset comprising of 10000 copies of G, where each copy has a different mapping matching the target.

Figure 1 (bottom) shows the evolution of the normalized spectral gap and training accuracy as a function of the number of rewiring iterations for three algorithms: our FoSR, SDRF (Topping et al., 2022), and G-RLEF (Banerjee et al., 2022). SDRF is motivated from the idea that negatively curved edges lead to oversquashing. The notion of curvature, which can be construed as a "local" Cheeger constant, plays a key role in the SDRF rewiring process. SDRF cycles between adding a supporting edge around a negatively curved edge and removing a redundant positively-curved edge; see Figure 1 (top). Since the addition and removal of edges are done independently of each other, SDRF can potentially disconnect a connected input graph. G-RLEF, on the other hand, seeks to improve the "global" Cheeger constant $h(\mathcal{G})$ sampling edges according to inverse triangle counts (a proxy for curvature) and using a local edge flip mechanism so that the input graph is never disconnected. In our experiments, we configured SDRF to only add edges. For all three algorithms, we observe that both the normalized spectral gap and training accuracy increase monotonically in the number of rewiring steps until they saturate at around 150 iterations. The rate of spectral expansion i.e., the rate at which the spectral gap improves as a function of the rewiring iterations is the fastest for FoSR.

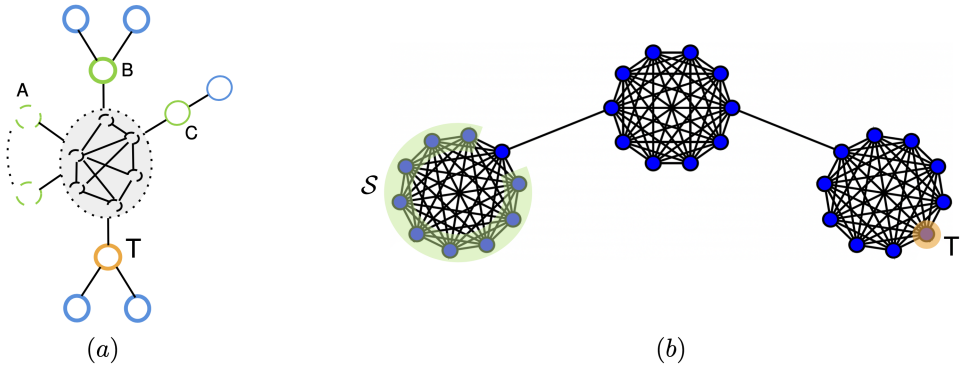

Figure 2: Learning task modeled on the NEIGHBORSMATCH problem.

C DETAILS ON SECTION 4

Proof of Theorem 4. We refer to Stewart and Sun (1990, Theorem 2.3) for a proof that λ_i is a continuously differentiable function and a computation of its gradient for general classes of matrices. We provide a calculation for our case here.

Fix a symmetric matrix $M_0 \in \mathbb{R}^{n\times n}$ and let $M_0 = U\Sigma U^{-1}$ be an orthonormal diagonalization of M_0 with the diagonal entries of Σ sorted in descending order. Fix $j, k \in [n]$. Let $E_{j,k}$ denote the matrix whose (j, k) entry is 1 and remaining entries are 0. Define a curve $\gamma : \mathbb{R} \to \mathbb{R}^{n\times n}$ by

$$\gamma(t) := M_0 + tUE_{j,k}U^{-1}.$$

Then

$$\lambda_i(\gamma(t)) = \lambda_i(U(\Sigma + tE_{j,k})U^{-1}) = \lambda_i(\Sigma + tE_{j,k}) = \lambda_i + \delta_{i,j}\delta_{i,k}t, \tag{9}$$

where $\delta_{i,j}$ is equal to 1 if $i = j$ and 0 otherwise. The final equality follows since the i-th diagonal entry of Σ is λ_i, and adding an off-diagonal entry does not change its eigenvalues. For sufficiently small t the order of the eigenvalues does not change. By the chain rule,

$$\frac{d}{dt}\Big|_{t=0} \lambda_i(\gamma(t)) = (d\lambda_i)_{M_0}(\dot{\gamma}(0)).$$

By equation (9), the left-hand side of the above equation is equal to $\delta_{i,j}\delta_{i,k}$. So

$$\delta_{i,j} = (d\lambda_i)_{M_0}(\dot{\gamma}(0)) = (d\lambda_i)_{M_0}(UE_{j,k}U^{-1}).$$

By the definition of the gradient,

$$\mathrm{Tr}(UE_{j,k}U^{-1}\nabla\lambda_i(M_0)) = (d\lambda_i)_{M_0}(UE_{j,k}U^{-1}) = \delta_{i,j}\delta_{i,k} = \mathrm{Tr}(E_{i,i}E_{j,k}).$$

Rewriting the left-hand side of the above equation gives us

$$\mathrm{Tr}(U^{-1}\nabla\lambda_i(M_0)UE_{j,k}) = \mathrm{Tr}(E_{i,i}E_{j,k}).$$

Since this holds for all $j, k \in [n]$ and the $E_{j,k}$ span $\mathbb{R}^{n\times n}$, this implies that

$$\begin{aligned} U^{-1}\nabla\lambda_i(M_0)U &= E_{i,i} \\ \nabla\lambda_i(M_0) &= UE_{i,i}U^{-1} \\ \nabla\lambda_i(M_0) &= x_i x_i^T, \end{aligned}$$

as desired. □

Proof of Proposition 5. Let A be the adjacency matrix of a graph G, and let A_N be the normalized adjacency matrix. Let x be an eigenvector of A_N with eigenvalue λ. Suppose that we add an edge (u, v) to G. Let δA_N denote the entry-wise change in the normalized adjacency matrix from adding the edge (u, v). If $i \neq u$ and $j \neq v$, then

$$(\delta A_N)_{ij} = 0.$$

If $i \neq u$ and $j = v$, then

$$(\delta A_N)_{ij} = \frac{A_{ij}}{\sqrt{d_i}}\left(\frac{1}{\sqrt{1+d_j}} - \frac{1}{\sqrt{d_j}}\right).$$

If $i = u$ and $j \neq v$, then

$$(\delta A_N)_{ij} = \frac{A_{ij}}{\sqrt{d_j}}\left(\frac{1}{\sqrt{1+d_i}} - \frac{1}{\sqrt{d_i}}\right).$$

If $i = u$ and $j = v$, then

$$(\delta A_N)_{ij} = \frac{1}{(1+\sqrt{d_i})(1+\sqrt{d_j})}.$$

By Theorem 4, the first-order change in $\lambda_2(A_N)$ is given by

$$\begin{aligned}
x^T(\delta A_N)x =& \sum_{i=u}\sum_{j=v}(\delta A_N)_{ij}x_ix_j + \sum_{i=u}\sum_{j\neq v}(\delta A_N)_{ij}x_ix_j + \sum_{i=v}\sum_{j=u}(\delta A_N)_{ij}x_ix_j + \sum_{i=v}\sum_{j\neq u}(\delta A_N)_{ij}x_ix_j \\
&+ \sum_{i\neq u,v}\sum_{j=u}(\delta A_N)_{ij}x_ix_j + \sum_{i\neq u,v}\sum_{j=v}(\delta A_N)_{ij}x_ix_j + \sum_{i\neq u,v}\sum_{j\neq u,v}(\delta A_N)_{ij}x_ix_j \\
=& \frac{x_ux_v}{(\sqrt{1+d_u})(\sqrt{1+d_v})} + \sum_{j\neq v}\frac{A_{uj}x_ux_j}{\sqrt{d_j}}\left(\frac{1}{\sqrt{1+d_u}} - \frac{1}{\sqrt{d_u}}\right) \\
&+ \frac{x_ux_v}{(\sqrt{1+d_u})(\sqrt{1+d_v})} + \sum_{j\neq u}\frac{A_{vj}x_vx_j}{\sqrt{d_j}}\left(\frac{1}{\sqrt{1+d_v}} - \frac{1}{\sqrt{d_v}}\right) \\
&+ \sum_{i\neq u,v}\frac{A_{iu}x_ix_u}{\sqrt{d_i}}\left(\frac{1}{\sqrt{1+d_u}} - \frac{1}{\sqrt{d_u}}\right) + \sum_{i\neq u,v}\frac{A_{iv}x_ix_v}{\sqrt{d_i}}\left(\frac{1}{\sqrt{1+d_v}} - \frac{1}{\sqrt{d_v}}\right) \\
=& \frac{2x_ux_v}{(\sqrt{1+d_u})(\sqrt{1+d_v})} + 2\sum_{i\neq u}\frac{A_{iv}x_ix_v}{\sqrt{d_i}}\left(\frac{1}{\sqrt{1+d_v}} - \frac{1}{\sqrt{d_v}}\right) \\
&+ 2\sum_{i\neq v}\frac{A_{iu}x_ix_u}{\sqrt{d_i}}\left(\frac{1}{\sqrt{1+d_u}} - \frac{1}{\sqrt{d_u}}\right) \\
=& \frac{2x_ux_v}{(\sqrt{1+d_u})(\sqrt{1+d_v})} + 2\sum_{i\neq u}(A_N)_{iv}x_ix_v\left(\frac{\sqrt{d_v}}{\sqrt{1+d_v}} - 1\right) \\
&+ 2\sum_{i\neq v}(A_N)_{iu}x_ix_u\left(\frac{\sqrt{d_u}}{\sqrt{1+d_u}} - 1\right) \\
=& \frac{2x_ux_v}{(\sqrt{1+d_u})(\sqrt{1+d_v})} + 2\sum_{i}(A_N)_{iv}x_ix_v\left(\frac{\sqrt{d_v}}{\sqrt{1+d_v}} - 1\right) \\
&+ 2\sum_{i}(A_N)_{iu}x_ix_u\left(\frac{\sqrt{d_u}}{\sqrt{1+d_u}} - 1\right)
\end{aligned}$$

Since x is an eigenvector of A_N with eigenvalue λ, we may write the above expression as

$$\frac{2x_ux_v}{(\sqrt{1+d_u})(\sqrt{1+d_v})} + 2\lambda x_v^2\left(\frac{\sqrt{d_v}}{\sqrt{1+d_v}} - 1\right) + 2\lambda x_u^2\left(\frac{\sqrt{d_u}}{\sqrt{1+d_u}} - 1\right).$$

Applying this equality to the second eigenvector of A_N yields the result. □

D Details on the experiments from Section 5

The computer implementation of the proposed methods along with scripts to re-run our experiments are made publicly available on `https://anonymous.4open.science/r/FoSR-0CE3/`. In the following we list details on hyperparameters, libraries, and the compute infrastructure we used in these experiments.

D.1 HYPERPARAMETERS

Table 2: Common hyperparameters

Dropout	0.5
Number of layers	4
Hidden dimension	64
Learning rate	10^{-3}
Stopping patience	100 epochs

Table 3: Rewiring iteration counts for FoSR and SDRF

FoSR

Architecture	REDDIT-BINARY	IMDB-BINARY	MUTAG	ENZYMES	PROTEINS	COLLAB
GCN	5	5	40	10	20	10
R-GCN	5	20	40	40	5	5
GIN	10	20	20	5	10	20
R-GIN	40	20	5	40	20	10

SDRF

Architecture	REDDIT-BINARY	IMDB-BINARY	MUTAG	ENZYMES	PROTEINS	COLLAB
GCN	5	20	5	5	40	5
R-GCN	40	5	40	5	20	20
GIN	5	10	5	5	20	40
R-GIN	5	40	5	5	5	20

Table 4: DIGL hyperparameters

Teleport probability (α)

Architecture	REDDIT-BINARY	IMDB-BINARY	MUTAG	ENZYMES	PROTEINS	COLLAB
GCN	0.15	0.05	0.15	0.15	0.15	0.05
R-GCN	0.15	0.05	0.05	0.15	0.05	0.05
GIN	0.05	0.15	0.05	0.15	0.15	0.15
R-GIN	0.05	0.15	0.05	0.05	0.05	0.05

Sparsification threshold (ϵ)

Architecture	REDDIT-BINARY	IMDB-BINARY	MUTAG	ENZYMES	PROTEINS	COLLAB
GCN	10^{-3}	10^{-4}	10^{-3}	10^{-4}	10^{-4}	10^{-4}
R-GCN	10^{-4}	10^{-3}	10^{-4}	10^{-3}	10^{-3}	10^{-3}
GIN	10^{-3}	10^{-4}	10^{-4}	10^{-3}	10^{-3}	10^{-4}
R-GIN	10^{-3}	10^{-4}	10^{-3}	10^{-4}	10^{-4}	10^{-3}

D.2 COMPUTER INFRASTRUCTURE AND LIBRARIES

In this section, we provide details of our implementation. All the experiments were implemented in Python using PyTorch (Paszke et al., 2019), NumPy (Harris et al., 2020), PyG (PyTorch Geometric) (Fey and Lenssen, 2019), with plots created using Matplotlib (Hunter, 2007). PyTorch, PyG and NumPy are made available under the BSD license, and Matplotlib under the PSF license.

We conducted all our experiments on a local server with 2x 22-Core/44-Thread Intel Xeon Scalable Gold 6152 processor (2.1/3.7 GHz) and 8x NVIDIA GeForce RTX 2080 Ti graphics card (11 GB, GDDR6).