On the Trade-off between Over-smoothing and Over-squashing in Deep Graph Neural Networks

Jhony H Giraldo, Konstantinos Skianis, Thierry Bouwmans, Fragkiskos D. Malliaros

On the Trade-off between Over-smoothing and Over-squashing in Deep Graph Neural Networks

Jhony H. Giraldo
LTCI, Télécom Paris - Institut Polytechnique de Paris
Palaiseau, France
jhony.giraldo@telecom-paris.fr

Konstantinos Skianis
BLUAI
Athens, Greece
skianis.konstantinos@gmail.com

Thierry Bouwmans
Laboratoire MIA, La Rochelle Université
La Rochelle, France
tbouwman@univ-lr.fr

Fragkiskos D. Malliaros
Université Paris-Saclay, CentraleSupélec, Inria
Gif-sur-Yvette, France
fragkiskos.malliaros@centralesupelec.fr

ABSTRACT

Graph Neural Networks (GNNs) have succeeded in various computer science applications, yet deep GNNs underperform their shallow counterparts despite deep learning's success in other domains. Over-smoothing and over-squashing are key challenges when stacking graph convolutional layers, hindering deep representation learning and information propagation from distant nodes. Our work reveals that over-smoothing and over-squashing are intrinsically related to the spectral gap of the graph Laplacian, resulting in an inevitable trade-off between these two issues, as they cannot be alleviated simultaneously. To achieve a suitable compromise, we propose adding and removing edges as a viable approach. We introduce the Stochastic Jost and Liu Curvature Rewiring (SJLR) algorithm, which is computationally efficient and preserves fundamental properties compared to previous curvature-based methods. Unlike existing approaches, SJLR performs edge addition and removal during GNN training while maintaining the graph unchanged during testing. Comprehensive comparisons demonstrate SJLR's competitive performance in addressing over-smoothing and over-squashing.

CCS CONCEPTS

• **Computing methodologies** → **Machine learning algorithms**;
• **Computer systems organization** → **Neural networks**.

KEYWORDS

Graph neural networks, over-smoothing, over-squashing, curvature

ACM Reference Format:
Jhony H. Giraldo, Konstantinos Skianis, Thierry Bouwmans, and Fragkiskos D. Malliaros. 2023. On the Trade-off between Over-smoothing and Over-squashing in Deep Graph Neural Networks. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management (CIKM '23), October 21–25, 2023, Birmingham, United Kingdom.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3583780.3614997

CIKM '23, October 21–25, 2023, Birmingham, United Kingdom

ACM ISBN 979-8-4007-0124-5/23/10…$15.00
https://doi.org/10.1145/3583780.3614997

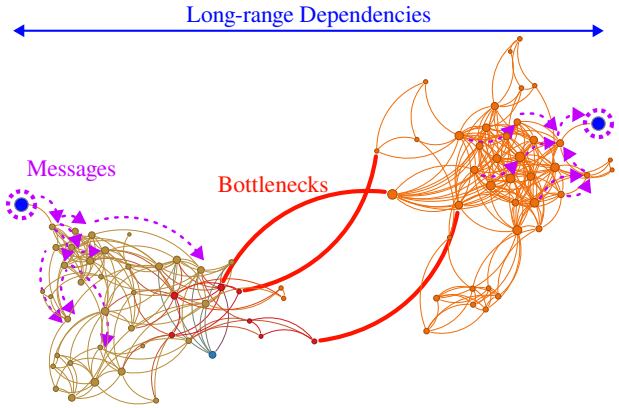

Figure 1: Long-range dependencies in graph neural networks.

1 INTRODUCTION

Graph representation learning is a growing research area, offering a versatile tool for modeling structured data. In this context, Graph Neural Networks (GNNs) have gained considerable attention from the research community [3, 23, 29, 50, 52]. GNNs extend Convolutional Neural Networks (CNNs) [31] to graph-structured data, enabling powerful models to capture complex dependencies between graph nodes. GNNs find applications in diverse domains, including semi-supervised learning [29], social network analysis [49], misinformation detection [2], materials modeling [13], drug discovery [16, 59], and computer vision [7, 21, 24, 32, 41].

Although GNNs show great promise in modeling graph-structured data, they are not immune to common neural network limitations, such as over-fitting and vanishing gradients [32]. Two intrinsic limitations of GNNs, over-smoothing [33] and over-squashing [1], remain poorly understood. These issues arise when stacking multiple graph convolutional layers, leading to degraded node representations and distorted information from distant nodes. Over-smoothing results from node features becoming more similar with increasing convolutional layers [33], while over-squashing occurs due to large information compression through bottleneck edges [1]. These issues are particularly relevant in graphs with large diameters and long-range dependencies between nodes [14], as illustrated in Fig. 1, where an exponentially growing number of messages is compressed as they traverse through bottleneck edges.

Various methods have been proposed to address these challenges [1, 10, 35, 39, 42, 48], yet their relationship remains unformally analyzed [27].

In this work, we establish a fundamental topological relationship between over-smoothing and over-squashing in deep GNNs. We leverage the properties of the random walk matrix and the spectral gap of the Laplacian matrix to investigate the phenomenon of over-smoothing, demonstrating how node representations exponentially converge to a stationary distribution [11]. Similarly, building upon the insights from Topping *et al.* [48], we establish a close connection between over-squashing and the spectral gap. We employ the Cheeger inequality [6] to highlight the inherent trade-off between over-smoothing and over-squashing, emphasizing that improving one aspect invariably worsens the other. To navigate this trade-off, we propose the *Stochastic Jost and Liu Curvature Rewiring* (SJLR) algorithm. The proposed algorithm introduces the Jost and Liu Curvature (JLC), a computationally efficient approximation of Ollivier's Ricci curvature [38] as presented in [26]. Unlike previous methods, SJLR dynamically adds and removes edges during GNN training to mitigate both over-smoothing and over-squashing, while ensuring the graph remains unchanged during evaluation. Notably, the JLC metric offers a less computationally complex alternative to the Balanced Forman Curvature (BFC) proposed by Topping *et al.* [48], while preserving important theoretical properties. To evaluate the effectiveness of SJLR, we conduct extensive benchmarking experiments on both homophilous and heterophyllous graph datasets. The results demonstrate the competitive performance of SJLR in addressing over-smoothing and over-squashing, highlighting its potential as a promising solution in the field of deep GNNs.

This work makes the following main contributions:

(1) The establishment of a significant topological relationship between over-smoothing and over-squashing, offering valuable theoretical insights into the behavior of deep GNNs.
(2) The introduction of SJLR, a novel rewiring algorithm specifically designed to address both over-smoothing and over-squashing. Notably, SJLR stands out as the first algorithm to perform edge removal and addition exclusively during training, ensuring that the original graph remains unchanged.
(3) Extensive experimentation to evaluate the effectiveness and properties of SJLR. Through comprehensive benchmarking and analysis, we provide empirical evidence supporting the performance and benefits of SJLR in mitigating over-smoothing and over-squashing issues.

The remainder of the paper is structured as follows: Section 2 reviews related work. Section 3 introduces mathematical notation and preliminary concepts. Section 4 presents the relationship between over-smoothing and over-squashing. Section 5 describes the SJLR algorithm. Finally, we report our experimental results in Section 6 and present concluding remarks in Section 7.

2 RELATED WORK

Over-smoothing, which refers to the problem of node embeddings of distinct classes becoming indistinguishable when stacking multiple layers in GNNs, was first discussed in [33]. Since then, several methods have been proposed to alleviate over-smoothing, which can be classified into the following categories:

(1) **Graph rewiring methods:** Klicpera *et al.* [18, 19] proposed an improved propagation scheme based on PageRank for rewiring the graph as a pre-processing step. Similarly, Rong *et al.* [42], Huang *et al.* [25], and Liu *et al.* [35] introduced methods that drop edges from the graph when training the GNN. Finally, Chen *et al.* [8] presented an approach that adaptively changes the graph topology.
(2) **Normalization techniques:** Zhao *et al.* [56] and Zhou *et al.* [57] proposed node-embeddings normalization techniques to address over-smoothing directly, *i.e.*, they tried to avoid nodes becoming indistinguishable. Similarly, Oono and Suzuki [39] presented a procedure that normalizes the weights of the GNN architecture.
(3) **Architectural changes:** Li *et al.* [32] proposed dilated convolutions and residual/dense connections in GNNs to create deep architectures. Chen *et al.* [9] introduced an iteratively learning graph structure and graph embedding such that their method learns a better graph structure based on better node embeddings, and vice versa. Chien *et al.* [10] presented a new graph convolutional filter inspired by the graph signal processing literature [44] to avoid over-smoothing.
(4) **Subgraphs approaches:** Zeng *et al.* [55] proposed to train GNNs of arbitrary depths with localized subgraphs.

Recent studies have highlighted over-squashing as a critical issue that hampers the ability of GNNs to effectively propagate information between distant nodes in a graph [1]. Alon and Yahav [1] proposed a rewiring method in which a fully-adjacent matrix is added in the last GNN layer to mitigate this problem. Topping *et al.* [48] and Di Giovanni *et al.* [12] further contributed to the understanding of over-squashing, offering theoretical insights into its origins, topological alleviation, and the impact of GNN design choices. Specifically, Topping *et al.* [48] introduced a rewiring method based on concepts from Ricci flow curvature in differential geometry [22]. However, these studies [1, 48] solely focused on addressing over-squashing and did not consider the trade-off between over-smoothing and over-squashing.

More recently, Karhadkar *et al.* [27] and Liu *et al.* [35] proposed empirical methods to alleviate both over-smoothing and over-squashing. Karhadkar *et al.* [27] acknowledged the trade-off between over-squashing and over-smoothing but did not provide formal proof of its existence. Furthermore, their method solely involved edge addition. Liu *et al.* [35] also presented a method to address both problems, but they only focused on edge removal based on a curvature metric. Differently to these works, 1) we formally prove the existence of the trade-off, 2) we argue that both edge addition and removal are necessary, and 3) we aim to explore the relationship between these two issues, providing a more comprehensive understanding of their interplay.

Transformers offer an alternative approach to address over-smoothing and over-squashing and have gained increasing interest in the graph and computer vision domains, with several studies exploring their effectiveness [4, 7, 54]. Notably, Ying *et al.* [53] observed that transformer architectures are less susceptible to over-smoothing compared to GNNs. Additionally, Kreuzer *et al.* [30] explained that transformers avoid over-squashing due to the presence of direct paths between distant nodes. However, transformers have

significant computational and memory limitations as each node attends to all other vertices, making them less suitable for large-scale graph applications. Moreover, improper training of transformers can result in a mixture of local and non-local interactions. In this work, we propose a novel approach to simultaneously address both over-smoothing and over-squashing. We utilize JLC and node-embedding metrics to rewire the graph, without relying on attention mechanisms, offering potential advantages for large-scale graph problems.

3 NOTATION AND BACKGROUND

In this paper, calligraphic letters like $\mathcal{V}$ designate sets, and $|\mathcal{V}|$ represents their cardinality. Uppercase boldface letters such as $\mathbf{A}$ represent matrices, and lowercase boldface letters like $\mathbf{x}$ denote vectors. diag($\mathbf{x}$) is a diagonal matrix with entries $x_1, x_2, \ldots, x_n$. $\|\cdot\|$ is the ℓ_2-norm of a vector and $(\cdot)^\top$ represents transposition. $\mathbf{I}$ is the identity matrix, and $\mathbf{1}$ is a vector of ones with appropriate dimensions. Finally, $\mathcal{N}_i$ is the set of neighbors of node i.

3.1 Preliminaries

A graph is a mathematical entity represented as $G = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{1, \ldots, N\}$ is the set of N nodes and the set of edges is $\mathcal{E} \subseteq \{(i,j) \mid i,j \in \mathcal{V} \text{ and } i \neq j\}$ such that (i,j) is an edge between the vertices i and j. In this paper, we consider undirected, connected, and unweighted graphs. $\mathbf{A} \in \{0,1\}^{N\times N}$ is the adjacency matrix of G such that $\mathbf{A}(i,j) = 1$ if $(i,j) \in \mathcal{E}$ and $\mathbf{A}(i,j) = 0$ otherwise. Moreover, $\mathbf{D} \in \mathbb{R}^{N\times N}$ is the diagonal degree matrix of G such that $\mathbf{D}(i,i) = \sum_{j=1}^{N} \mathbf{A}(i,j)\ \forall\ i = 1, \ldots, N$, and $d_i = \mathbf{D}(i,i)$. $\mathbf{L} = \mathbf{D} - \mathbf{A}$ is the positive semi-definite combinatorial Laplacian operator. Similarly, $\mathbf{L}_{\text{sym}} = \mathbf{D}^{-\frac{1}{2}}\mathbf{L}\mathbf{D}^{-\frac{1}{2}} = \mathbf{I} - \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$ is the symmetrically normalized Laplacian matrix with eigenvalues $0 = \lambda_1 < \lambda_2 \leq \ldots \leq \lambda_N \leq 2$ and corresponding eigenvectors $\{\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_N\}$.

3.2 Cheeger Inequality and Cheeger Constant

Definition 1. *Let $\mathcal{S} \subset \mathcal{V}$ be a subset of nodes of G. Let $\partial\mathcal{S}$ be the set of edges going from a node in $\mathcal{S}$ to a node in $\mathcal{V} \setminus \mathcal{S}$, i.e., $\partial\mathcal{S} \triangleq \{\{u,v\} \in \mathcal{E} : u \in \mathcal{S}, v \in \mathcal{V} \setminus \mathcal{S}\}$. Therefore, we can define the Cheeger constant h_G of G as $h_G \triangleq \min_{\mathcal{S}} h_G(\mathcal{S})$, where $h_G(\mathcal{S}) = |\partial\mathcal{S}|/\min(\text{vol}(\mathcal{S}), \text{vol}(\mathcal{V} \setminus \mathcal{S}))$, and $\text{vol}(\mathcal{S}) = \sum_{i\in\mathcal{S}} d_i$.*

Intuitively, the Cheeger constant in Definition 1 is small when there exists a *bottleneck* in G, *i.e.*, when there are two sets of nodes with few edges between them. Similarly, we know that $h_G > 0$ iff G is a connected graph [11]. We can relate the Cheeger constant h_G with the first non-zero eigenvalue of $\mathbf{L}_{\text{sym}}$ through the Cheeger inequality:

$$2h_G \geq \lambda_2 \geq \frac{h_G^2}{2}. \tag{1}$$

We notice from (1) that for having less "bottleneckness" in the graph, we need to promote big values of h_G, *i.e.*, having large values of λ_2 will increase h_G since $h_G \geq \lambda_2/2$.

3.3 Message Passing Neural Networks (MPNNs)

Let G be a graph with a set of input features $\mathbf{X} \in \mathbb{R}^{N\times F_1} = [\mathbf{x}_1, \ldots, \mathbf{x}_N]^\top$. The output of a generic MPNN is defined as follows [20]:

$$\mathbf{h}_i^{(l+1)} = \phi_l\left(\mathbf{h}_i^{(l)}, \sum_{j=1}^{N} \hat{\mathbf{A}}(i,j)\psi_l(\mathbf{h}_i^{(l)}, \mathbf{h}_j^{(l)})\right), \tag{2}$$

where $\mathbf{H}^{(l)} \in \mathbb{R}^{N\times F_l} = [\mathbf{h}_1^{(l)}, \ldots, \mathbf{h}_N^{(l)}]^\top$ is the F_l-dimensional embeddings after l layers such that each $\mathbf{h}_i^{(l)} \in \mathbb{R}^{F_l}$ and $\mathbf{H}^{(1)} = \mathbf{X}$, $\psi_l : \mathbb{R}^{F_l} \times \mathbb{R}^{F_l} \to \mathbb{R}^{F'_l}$ is a family of message functions, $\hat{\mathbf{A}}$ is an augmented normalized adjacency matrix, and $\phi_l : \mathbb{R}^{F_l} \times \mathbb{R}^{F'_l} \to \mathbb{R}^{F_{l+1}}$ is an update function.

3.4 Over-smoothing

Graph convolutions usually use smoothing functions on each layer. Therefore, when we apply several graph convolution layers, the performance can suffer from over-smoothing, where node embeddings from different clusters become mixed up. Over-smoothing lacks a formal definition in the literature [56]. However, we can think of over-smoothing as a random walk transition matrix that is repeatedly applied to a node feature, thus converging to a stationary distribution and washing away all the feature information (this convergence is explained in (3), Section 4.1).

3.5 Over-squashing

Over-squashing is a more recent and less understood problem than over-smoothing. A graph learning problem has *long-range dependencies* when the outputs of GNNs depend on features of interacting distant nodes. In that scenario, information from non-adjacent nodes should be propagated through the network without distortion. Let $\mathcal{B}_r \triangleq \{j \in \mathcal{V} : d_G(i,j) \leq r\}$ be the receptive field of an r-layer GNN, where d_G is the shortest-path distance and $r \in \mathbb{N}$. Let $\partial\mathbf{h}_i^{(r)}/\partial\mathbf{x}_j$ be the *Jacobian* of a node embedding $\mathbf{h}_i^{(r)}$ with respect to some input feature $\mathbf{x}_j$ in node j. Over-squashing can be understood as the inability of $\mathbf{h}_i^{(r)}$ to be affected by $\mathbf{x}_j$ at a distance r. Topping *et al.* [48] proved that $\left|\partial\mathbf{h}_i^{(r+1)}/\partial\mathbf{x}_j\right| \leq (\alpha\beta)^{r+1}\hat{\mathbf{A}}^{r+1}(i,j)$, if $|\nabla\phi_l| \leq \alpha$ and $|\nabla\psi_l| \leq \beta$ for $0 \leq l \leq r$, with ϕ_l, ψ_l differentiable functions. In many graphs, $|\mathcal{B}_r|$ grows exponentially with r, and then representations of an exponential amount of neighboring nodes should be compressed into fixed-size vectors. For example, if $d_G(i,j) = r+1$ in a binary tree, we have that $\hat{\mathbf{A}}^{r+1}(i,j) = 2^{-1}3^{-r}$, which gives an exponential decay of the node dependence on input features at distance r [48]. This phenomenon is referred to as over-squashing of information [1, 12, 48].

4 UNDERSTANDING THE OVER-SMOOTHING VS. OVER-SQUASHING TRADE-OFF

4.1 The Stationary Distribution on Graphs

Let $\mathbf{P} = \mathbf{D}^{-1}\mathbf{A}$ be the random walk transition matrix. For any initial distribution $f : \mathcal{V} \to \mathbb{R}$ with $\sum_{v\in\mathcal{V}} f(v) = 1$, the distribution after k steps is given by $\mathbf{f}^\top\mathbf{P}^k$, where $\mathbf{f} \in \mathbb{R}^{N\times 1}$ is the vector of initial distributions such that $\mathbf{f}(i)$ is the function evaluated on the ith node. The random walk is *ergodic* when there is a unique *stationary distribution* $\boldsymbol{\pi}$ satisfying that $\lim_{s\to\infty} \mathbf{f}^\top\mathbf{P}^s = \boldsymbol{\pi}$ [11].

Lemma 1 (Chung [11]). *Let $\mathbf{P}$ be an ergodic random walk transition matrix, where G is connected and non-bipartite, let $\boldsymbol{\pi}$ be its*

stationary distribution, and let $\mathbf{f}$ *be any initial distribution. For* $s \in \mathbb{N}^+$, *we have:*

$$\|\mathbf{f}^\top \mathbf{P}^s - \boldsymbol{\pi}\| \leq e^{-s\lambda'} \frac{\max_i \sqrt{d_i}}{\min_j \sqrt{d_j}}, \tag{3}$$

where $\lambda' = \lambda_2$ *if* $1 - \lambda_2 \geq \lambda_N - 1$, *and* $2 - \lambda_N$ *otherwise. Therefore, we can compute the value of* s *such that* $\|\mathbf{f}^\top \mathbf{P}^s - \boldsymbol{\pi}\| \leq \epsilon$ *as follows:*

$$s \geq \frac{1}{\lambda' \log\left(\max_i \sqrt{d_i}/\epsilon \min_j \sqrt{d_j}\right)}. \tag{4}$$

Notice that λ' is either λ_2 or $2 - \lambda_N$ in Lemma 1. However, we can show that only λ_2 is crucial. Suppose that $\lambda_N - 1 > 1 - \lambda_2$, so that $\lambda' = 2 - \lambda_N$. We can consider the lazy walk on the graph G' formed by adding a loop of weight d_i to each node i, *i.e.*, $\mathbf{A} + \mathbf{D}$. Therefore, the new graph Laplacian has eigenvalues $\tilde{\lambda}_i = \lambda_i/2 \leq 1$ [11], and then $1 - \tilde{\lambda}_2 \geq 1 - \tilde{\lambda}_N \geq 0$.

The key message of Lemma 1 is a simplified version of the same observations of [39, 42], *i.e.*, GNNs converge exponentially to a stationary distribution when stacking several layers. However, we show that the convergence of this exponential function depends on the spectral gap λ_2. We use this result later to show the underlying relationship between over-smoothing and over-squashing. Similarly, we can have a simplified explanation of why sparsification methods in GNNs, like DropEdge [42], can alleviate over-smoothing.

Lemma 2 (Chung [11]). *Let* G *be a graph with diameter* $Di \geq 4$, *then:*

$$\lambda_2 \leq 1 - 2\frac{\sqrt{(\max_i d_i) - 1}}{\max_i d_i}\left(1 - \frac{2}{Di}\right) + \frac{2}{Di}. \tag{5}$$

Lemma 2 shows an upper bound of λ_2 so that reducing the maximum degree of G promotes small values of λ_2, *i.e.*, having a sparser graph promotes low values of λ_2 and thus high values of s in (4).

4.2 Over-smoothing and Over-squashing

We can establish a link between the Cheeger constant h_G and the parameter s in (4) as follows:

Theorem 3. *Let* h_G *be the Cheeger constant of* G, *and let* s *be the number of required steps such that the* ℓ_2 *distance between* $\mathbf{f}^\top \mathbf{P}^s$ *and* $\boldsymbol{\pi}$ *is at most* ϵ. *Therefore, we have that:*

$$2h_G \geq \frac{1}{s} \log\left(\frac{\max_i \sqrt{d_i}}{\epsilon \min_j \sqrt{d_j}}\right). \tag{6}$$

Proof: see Appendix A.

From Theorem 3, we have that if $s \to 0$ then $h_G \to \infty$, *i.e.*, we can promote less "bottleneckness" in the graph if we accelerate the convergence to the stationary distribution. Similarly, if $h_G \to 0$ then $s \to \infty$, *i.e.*, we can avoid converging to the stationary distribution if we promote a bottleneck-kind structure in the graph. In other words, we can reduce over-squashing by accelerating the convergence to the stationary distribution that worsens over-smoothing. Correspondingly, we can avoid over-smoothing by promoting having bottlenecks that worsen over-squashing. We also prove in Appendix B that 1) if $s \to \infty$ then $h_G \to 0$, and 2) if $h_G \to \infty$ then $s \to 0$, so $s \to \infty \iff h_G \to 0$ and $h_G \to \infty \iff s \to 0$.

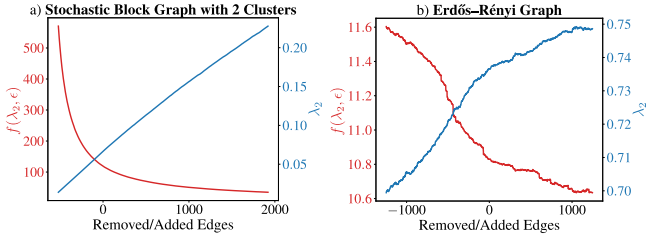

Figure 2: Mixing steps $f(\lambda_2, \epsilon)$ **for** $\epsilon = 5 \times 10^{-4}$ **vs. number of removed or added edges for a) one stochastic block model graph with two clusters, and b) one Erdős-Rényi graph.**

We can make the connection between over-smoothing and over-squashing more precisely using Theorem 3, the Simple Graph Convolution (SGC) model [51], and the developments in [48]. In other words, we can use an SGC with a random walk kernel to show how the node embeddings converge to the stationary distribution when we stack several layers according to Theorem 3. Similarly, Topping *et al.* [48] explained how reducing bottlenecks in the graphs can alleviate over-squashing, *i.e.*, the receptive field of each node in a deep GNN will be polynomial in the hop-distance rather than exponential. SGC is a simplified version of the Graph Convolutional Network (GCN) [29], where we remove all projection parameters and all non-linear activation functions between layers. We use SGC as a proxy of GNNs, as in [56], to study the relationship between over-smoothing and over-squashing. As a consequence, our theoretical results are only available for linear-based GNNs like SGC [51] or simple spectral graph convolution [58] (experimental results on other GNNs are presented in Section 6). We leave for future work the analysis of the relationship between over-smoothing and over-squashing for more complex GNNs architectures.

Let $f(\lambda_2, \epsilon) = \frac{1}{\lambda_2} \log\left(\max_i \sqrt{d_i}/\epsilon \min_j \sqrt{d_j}\right)$ be the mixing steps of our graph, *i.e.*, the lower bound in the maximum number of layers of an SGC such that the difference between the initial and stationary distribution is at most ϵ. Figure 2 shows, for $\epsilon = 5 \times 10^{-4}$, how $f(\lambda_2, \epsilon)$ and λ_2 change when we add or remove edges in one artificial stochastic block model graph and one Erdős-Rényi graph. We can increase the mixing steps by removing edges, *i.e.*, we can alleviate over-smoothing by making the graph more "bottleneckness". This partially explains why DropEdge [42] can alleviate over-smoothing. On the other hand, we increase λ_2 by adding edges as shown in Fig. 2, so we promote higher values of h_G, *i.e.*, we can alleviate over-squashing by making the graph less "bottleneckness". This can partially explain why the methodology by Topping *et al.* [48] can alleviate over-squashing. However, there is a trade-off between $f(\lambda_2, \epsilon)$ and λ_2 from a topological point of view, *i.e.*, we can increase $f(\lambda_2, \epsilon)$ by removing key edges but λ_2 will decrease, and vice versa. The algorithm to add and remove edges is explained in Section 5.

5 CURVATURE REWIRING ALGORITHM

Topping *et al.* [48] proposed a method to alleviate over-squashing using concepts of Ricci flow curvature. We can understand curvature-based methods using the following analysis:

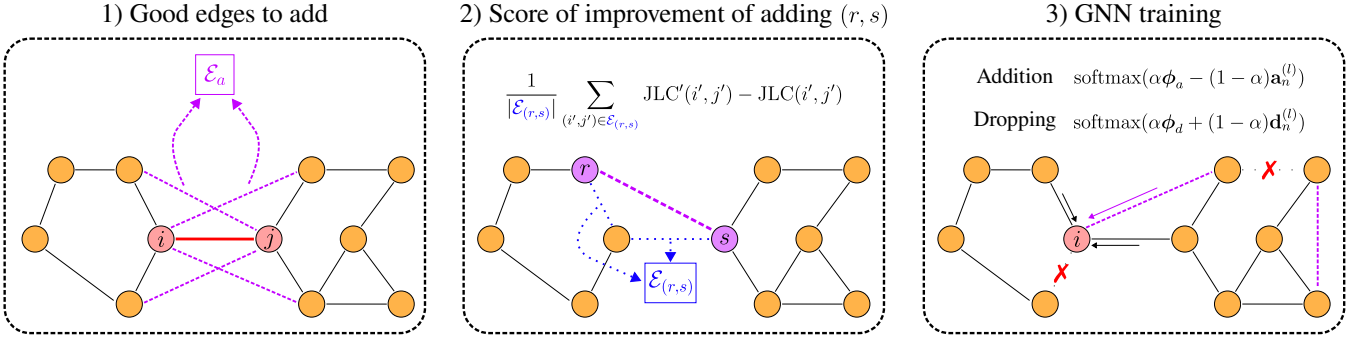

Figure 3: The pipeline of the proposed Stochastic Jost and Liu curvature Rewiring (SJLR) algorithm. SJLR first saves potential good edges $\mathcal{E}_a$ to add to the graph, based on the JLC metric. Secondly, SJLR computes the score of improvement of adding these new edges as the average of the JLC improvement that concerns these edges. Thirdly, edges are added and removed according to some probability distribution during the graph neural network training. The graph remains unchanged during testing.

Theorem 4 (Lin et al. [34]). *Let G be a finite graph, let λ_2 be its spectral gap, and let $\kappa(i, j)$ be the Ricci curvature as defined in [34]. If for any edge (i, j), $\kappa(i, j) \geq \kappa > 0$, then $\lambda_2 \geq \kappa$.*

Corollary 5. *If $\kappa(i, j) \geq \kappa > 0$ for any edge (i, j), then $2h_G \geq \kappa$.*
Proof: Using the Cheeger inequality and Theorem 4, we have that $2h_G \geq \lambda_2 \geq \kappa$, and then $2h_G \geq \kappa$.

From Theorem 4 and Corollary 5, we can conclude that if we have positive Ricci curvature everywhere, then $2h_G \geq \kappa$. Therefore, increasing curvature will make the graph less "bottleneckness". In this paper, we use a bound of the Ollivier's Ricci curvature [38] presented in [26].

Definition 2 (Jost and Liu Curvature (JLC) [26]). *For any edge (i, j) in a finite graph:*

$$\text{JLC}(i,j) = -\left(1 - \frac{1}{d_i} - \frac{1}{d_j} - \frac{\#(i,j)}{d_i \wedge d_j}\right)_+ - \left(1 - \frac{1}{d_i} - \frac{1}{d_j} - \frac{\#(i,j)}{d_i \vee d_j}\right)_+ + \frac{\#(i,j)}{d_i \vee d_j}, \tag{7}$$

where $\#(i, j)$ is the number of triangles which include (i, j) as nodes, $c_+ \triangleq \max(c, 0)$, $c \vee t \triangleq \max(c, t)$, and $c \wedge t \triangleq \min(c, t)$.

Theorem 6 (Jost and Liu [26]). *On a locally finite graph we have that $\kappa(i, j) \geq \text{JLC}(i, j)$.*

Corollary 7. *If $\text{JLC}(i, j) \geq \kappa > 0$ for any edge (i, j), then $\kappa(i, j) \geq \kappa > 0$.*
Proof: Using Theorems 4 and 6 we have that $\kappa(i, j) \geq \text{JLC}(i, j) \geq \kappa \geq 0$, then $\kappa(i, j) \geq \kappa \geq 0$.

From Corollary 7, we can conclude that if we have positive JLC everywhere in G, $\kappa(i, j)$ will also be positive everywhere. As a result, having positive JLC ensures that the receptive field of each node in a deep GNN will be polynomial in the hop-distance rather than exponential (see Corollary 3 in [48]). Topping *et al.* [48] defined the Balanced Forman Curvature (BFC) metric to solve over-squashing. However, BFC requires counting triangles, 4-cycles, and the maximal number of 4-cycles traversing a common node for each edge. This process makes BFC very computationally intensive and unsuitable for practical applications. Thus, JLC is less computationally complex than BFC while keeping the same theoretical properties about the polynomial receptive field growth.

5.1 Stochastic Jost and Liu Curvature Rewiring Details

Our algorithm uses JLC and node feature information to perform rewiring. Topping *et al.* [48] proposed a method where edges are added and removed as a pre-processing step. However, their reason for removing edges was not properly justified. We argue that deleting edges is also important to alleviate over-smoothing, according to the developments in Section 4. SJLR also adds and removes edges but is fundamentally different. We stochastically add and remove edges only during training, so we can alleviate over-squashing and over-smoothing without modifying the initial graph, maintaining its original properties. We define a set of hyperparameters for SJLR: 1) let p_A be the percentage of added edges, 2) let p_D be the percentage of dropped edges, and 3) let $\alpha \in [0, 1]$ be a variable controlling how important is the JLC metric against the embedding information while dropping or adding edges. We optimize the hyperparameters in the validation set so that SJLR can choose either if the specific dataset requires more addition or removal of edges, *i.e.*, SJLR tries to find the "sweet point" in the trade-off between over-smoothing and over-squashing[1].

Figure 3 shows the pipeline of the proposed Stochastic Jost and Liu curvature Rewiring (SJLR) algorithm, where edges are added and removed while training the GNN using JLC and node-embeddings information. Algorithm 1 presents our SJLR approach in detail. The algorithm has as input an initial graph G and a given GNN architecture. SJLR first computes a bank of potential good edges to add $\mathcal{E}_a$ for improving the JLC curvature (first part in Fig. 3), where the JLC metric is calculated (for all $(i, j) \in \mathcal{E}$) and sorted (in ascending order). To this end, we look at every edge (i', j') from the sorted JLC vector, and we compute $\mathcal{A} = \{\{(\mathcal{N}_{i'} \setminus j') \times j'\} \cup \{(\mathcal{N}_{j'} \setminus i') \times i'\} : \mathcal{A} \notin \mathcal{E}\}$, which is the set of edges that form triangles with (i', j') and are not in $\mathcal{E}$. We append this set $\mathcal{A}$ to $\mathcal{E}_a$ until we have at least $2p_A|\mathcal{E}|$ edges in $\mathcal{E}_a$. Therefore, we associate a score $\sigma(m)$ to every edge $(r, s) \in \mathcal{E}_a$, which is computed as the

[1]For further details about the "sweet point" see Appendix C.

Table 1: Statistics of the datasets in the experimental framework of this work.

	Cornell	Texas	Wisconsin	Chameleon	Squirrel	Actor	Cora	Citeseer	Pubmed
$H(G)$	0.11	0.06	0.16	0.25	0.22	0.24	0.83	0.72	0.79
Nodes	140	135	184	832	2,186	4,388	2,485	2,120	19,717
Edges	219	251	362	12,355	65,224	21,907	5,069	3,679	44,324
Features	1,703	1,703	1,703	2,325	2,089	932	1,433	3,703	500
Classes	5	5	5	5	5	5	7	6	3
Directed	✓	✓	✓	✓	✓	✓			

Algorithm 1 Stochastic Jost and Liu Curvature Rewiring (SJLR)

Input: graph G, GNN architecture
Initialization: hyperparameters p_A, p_D, α, $\rho = 1$
1: $\mathcal{E}_a$, JLC = BANK_EDGES(G)
2: $\boldsymbol{\phi}_d, \boldsymbol{\phi}_a$ = SCORES_IMPROVEMENT($G, \mathcal{E}_a$, JLC)
3: **for** each layer l in GNN while training **do**
4: Compute $\mathbf{d}^{(l)}(p) = \|\mathbf{h}_i^{(l)} - \mathbf{h}_j^{(l)}\| \; \forall \; (i,j) \in \mathcal{E}, \; 1 \leq p \leq |\mathcal{E}|$
5: Drop $p_D|\mathcal{E}|$ edges from $\mathcal{E}$ according to the probability distribution softmax($\alpha\boldsymbol{\phi}_d + (1-\alpha)\mathbf{d}_n^{(l)}$)
6: Compute $\mathbf{a}^{(l)}(q) = \|\mathbf{h}_r^{(l)} - \mathbf{h}_s^{(l)}\| \; \forall \; (r,s) \in \mathcal{E}_a, \; 1 \leq q \leq |\mathcal{E}_a|$
7: Add $p_A|\mathcal{E}|$ edges from $\mathcal{E}_a$ according to the probability distribution softmax($\alpha\boldsymbol{\phi}_a - (1-\alpha)\mathbf{a}_n^{(l)}$)

8: **function** BANK_EDGES(Graph G)
9: Compute JLC$(i,j) \; \forall \; (i,j) \in \mathcal{E}$
10: $\text{JLC}_{\text{sorted}}$ = sort(JLC), sort the JLC scores in ascending order
11: Create an empty set of potential good edges to add $\mathcal{E}_a = \emptyset$
12: **while** $|\mathcal{E}_a| < 2p_A|\mathcal{E}|$ **do**
13: (i', j') is the ρth edge in $\text{JLC}_{\text{sorted}}$
14: $\mathcal{A} = \{\{(\mathcal{N}_{i'} \setminus j') \times j'\} \cup \{(\mathcal{N}_{j'} \setminus i') \times i'\} : \mathcal{A} \notin \mathcal{E}\}$
15: $\mathcal{E}_a = \mathcal{E}_a \cup \mathcal{A}$, $\rho = \rho + 1$
16: **return** $\mathcal{E}_a$, JLC$(i,j) \; \forall \; (i,j) \in \mathcal{E}$

17: **function** SCORES_IMPROVEMENT(Graph G, $\mathcal{E}_a$, JLC values)
18: Create a vector $\boldsymbol{\sigma} \in \mathbb{R}^{|\mathcal{E}_a|}$
19: **for** $m = 1$ until $m = |\mathcal{E}_a|$ **do**
20: Compute the set of edges $\mathcal{E}_{(r,s)} \subset \mathcal{E}$ that form a triangle with (r,s), where (r,s) is the mth edge in $\mathcal{E}_a$
21: Create a new graph $G' = (\mathcal{V}, \{\mathcal{E} \cup (r,s)\})$
22: Compute JLC$'(i',j') \; \forall \; (i',j') \in \mathcal{E}_{(r,s)}$ in graph G'
23: $\boldsymbol{\sigma}(m) = \frac{1}{|\mathcal{E}_{(r,s)}|} \sum_{(i',j') \in \mathcal{E}_{(r,s)}} \text{JLC}'(i',j') - \text{JLC}(i',j')$
24: Normalize JLC(i,j) and $\boldsymbol{\sigma}(m)$ to be in $[0,1]$, $\forall \; (i,j) \in \mathcal{E}$ and $1 \leq m \leq |\mathcal{E}_a|$
25: **return** $\boldsymbol{\phi}_d, \boldsymbol{\phi}_a$

average improvement of curvature from adding that edge (r,s) to the graph (second part in Fig. 3). Let $\boldsymbol{\sigma} \in \mathbb{R}^{|\mathcal{E}_a|}$ be the vector of JLC improvements, such that:

$$\boldsymbol{\sigma}(m) = \frac{1}{|\mathcal{E}_{(r,s)}|} \sum_{(i',j') \in \mathcal{E}_{(r,s)}} \text{JLC}'(i',j') - \text{JLC}(i',j'), \tag{8}$$

where JLC$'(i',j')$ is the JLC metric of edge (i',j') computed in the augmented graph $G' = (\mathcal{V}, \{\mathcal{E} \cup (r,s)\})$, and $\mathcal{E}_{(r,s)} \subset \mathcal{E}$ is the set of edges that form a triangle with the edge $(r,s) \in \mathcal{E}_a$. Before the GNN training loop, we normalize JLC(i,j) and $\boldsymbol{\sigma}(m)$ as shown in line 24, and save them in two vectors $\boldsymbol{\phi}_d$ and $\boldsymbol{\phi}_a$. During the GNN training for each layer l, 1) we compute the euclidean distances $\mathbf{d}^{(l)}$ and $\mathbf{a}^{(l)}$ as shown in lines 4 and 6, 2) we normalize $\mathbf{d}^{(l)}$ and $\mathbf{a}^{(l)}$ to be in $[0,1]$ and get $\mathbf{d}_n^{(l)}$ and $\mathbf{a}_n^{(l)}$, and 3) we drop and add edges according to the probabilities distribution in lines 5 and 7 (third part in Fig. 3). It is worth noting that the addition of potential good edges could be partially parallelized since there is not a sequential procedure. This is an important difference regarding previous methods [48], where edges cannot be added in parallel. SJLR is agnostic to the GNN architecture. However, we only test SJLR using SGCs [51] and GCNs [29] in Section 6.

6 EXPERIMENTAL FRAMEWORK AND RESULTS

6.1 Experiments

We perform a set of experiments to compare SJLR with several approaches in the literature. SJLR is compared to eight state-of-the-art methods to alleviate over-smoothing or over-squashing, including Residual/Dense Connections (RDC) [32], Graph Diffusion Convolution (GDC) with personalized PageRank kernel [19], DropEdge (DE) [42], PairNorm (PN) [56], Differentiable Group Normalization (DGN) [57], Fully-Adjacent (FA) layers [1], Stochastic Discrete Ricci Flow (SDRF) [48], and First-order Spectral Rewiring FoSR [27]. Our GNN base models are SGC [51] and GCN [29] for all experiments. We do not test RDC and FA in SGC because this GNN model only has one graph convolutional layer. We evaluate all methods in nine datasets: Cornell, Texas, and Wisconsin from the WebKB project[2], Chameleon [43], Squirrel [43], Actor [47], Cora [36], Citeseer [45], and Pubmed [37]. We consider the largest connected component of the graph for each dataset as in [19, 48]. Table 1 shows the statistics of the datasets tested in this work, where $H(G)$ is the homophily of the graph as defined in [40]. We split the data into a development set and a test set, ensuring that the test set is not used during the hyperparameter optimization process. We follow the same experimental framework as in [19, 48], *i.e.*, we optimize the hyperparameters for all dataset-preprocessing combinations separately by random search over 100 data splits. Furthermore, we report average accuracies on the test set accompanied by 95% confidence intervals calculated by bootstrapping with 1,000 samples. For Cora, Citeseer, and Pubmed, the development set contains 1,500 nodes and the rest of the nodes are used for testing. Similarly, the

[2] http://www.cs.cmu.edu/afs/cs.cmu.edu/project/theo-11/www/wwkb/

Table 2: Comparison results of the proposed SJLR algorithm with several state-of-the-art methods to alleviate over-smoothing and over-squashing with the SGC model as backbone.

Method	Cornell	Texas	Wisconsin	Chameleon	Squirrel	Actor	Cora	Citeseer	Pubmed	*Overall*
Baseline	$53.40_{\pm 2.11}$	$56.69_{\pm 1.78}$	$47.90_{\pm 1.73}$	$38.40_{\pm 0.69}$	$40.52_{\pm 0.54}$	$29.93_{\pm 0.16}$	$76.94_{\pm 1.31}$	$67.45_{\pm 0.80}$	$71.79_{\pm 2.13}$	53.67
GDC [19]	$58.65_{\pm 1.43}$	$57.42_{\pm 0.74}$	$45.93_{\pm 1.05}$	$38.13_{\pm 0.55}$	$36.63_{\pm 0.31}$	**$32.25_{\pm 0.17}$**	$76.02_{\pm 1.70}$	$66.22_{\pm 1.13}$	$71.91_{\pm 2.30}$	53.68
DE [42]	*$61.99_{\pm 1.04}$*	*$57.88_{\pm 0.81}$*	*$54.78_{\pm 0.89}$*	$40.38_{\pm 0.47}$	$41.28_{\pm 0.32}$	$30.62_{\pm 0.17}$	$80.59_{\pm 0.80}$	**$68.63_{\pm 0.51}$**	$74.47_{\pm 1.65}$	*56.74*
PN [56]	$53.11_{\pm 1.36}$	$50.47_{\pm 1.04}$	$48.72_{\pm 1.65}$	**$41.49_{\pm 0.68}$**	$39.72_{\pm 0.33}$	$22.58_{\pm 0.29}$	$75.55_{\pm 0.42}$	$64.16_{\pm 0.41}$	$73.81_{\pm 0.52}$	52.18
DGN [57]	$55.68_{\pm 1.32}$	$57.42_{\pm 2.59}$	$50.67_{\pm 2.08}$	*$40.99_{\pm 0.62}$*	*$41.72_{\pm 0.29}$*	$29.53_{\pm 0.18}$	*$80.65_{\pm 0.48}$*	$67.65_{\pm 0.59}$	*$74.95_{\pm 0.59}$*	55.47
SDRF [48]	$54.68_{\pm 1.29}$	$55.36_{\pm 1.48}$	$47.81_{\pm 1.51}$	$38.07_{\pm 0.77}$	$39.94_{\pm 0.53}$	$30.04_{\pm 0.17}$	$76.04_{\pm 1.69}$	$67.60_{\pm 0.80}$	$69.62_{\pm 2.35}$	53.24
FoSR [27]	$53.73_{\pm 1.75}$	$56.33_{\pm 1.37}$	$47.82_{\pm 2.14}$	$38.01_{\pm 0.73}$	$40.68_{\pm 0.42}$	$30.11_{\pm 0.18}$	$78.24_{\pm 0.98}$	$67.04_{\pm 0.83}$	$72.76_{\pm 2.35}$	53.86
SJLR (ours)	**$67.37_{\pm 1.64}$**	**$58.40_{\pm 1.48}$**	**$55.42_{\pm 0.92}$**	$40.17_{\pm 0.49}$	**$41.91_{\pm 0.34}$**	*$30.81_{\pm 0.18}$*	**$81.24_{\pm 0.77}$**	*$68.39_{\pm 0.69}$*	**$76.28_{\pm 0.96}$**	**57.78**

The best and second-best performing methods on each dataset are shown in **red** and *blue*, respectively.

Table 3: Comparison results of the proposed SJLR algorithm with several state-of-the-art methods to alleviate over-smoothing and over-squashing for the GCN model as backbone.

Method	Cornell	Texas	Wisconsin	Chameleon	Squirrel	Actor	Cora	Citeseer	Pubmed	*Overall*
Baseline	*$67.34_{\pm 1.50}$*	$58.05_{\pm 0.96}$	$52.10_{\pm 0.95}$	$40.35_{\pm 0.48}$	**$42.12_{\pm 0.29}$**	$28.62_{\pm 0.36}$	$81.81_{\pm 0.26}$	$68.35_{\pm 0.35}$	$78.25_{\pm 0.37}$	*57.44*
RDC [32]	$63.78_{\pm 1.68}$	$59.47_{\pm 1.00}$	$50.89_{\pm 1.00}$	$40.33_{\pm 0.51}$	*$41.98_{\pm 0.31}$*	$28.97_{\pm 0.33}$	$81.54_{\pm 0.26}$	$68.70_{\pm 0.35}$	$78.42_{\pm 0.39}$	57.12
GDC [19]	$64.18_{\pm 1.36}$	$56.43_{\pm 1.15}$	$49.61_{\pm 0.95}$	$38.49_{\pm 0.51}$	$33.20_{\pm 0.29}$	**$31.08_{\pm 0.27}$**	**$82.63_{\pm 0.23}$**	$69.15_{\pm 0.30}$	**$79.04_{\pm 0.37}$**	55.98
DE [42]	$63.39_{\pm 1.29}$	$57.41_{\pm 0.93}$	$47.84_{\pm 0.86}$	*$40.80_{\pm 0.55}$*	$41.68_{\pm 0.39}$	*$29.99_{\pm 0.21}$*	$81.90_{\pm 0.24}$	$68.99_{\pm 0.36}$	$78.53_{\pm 0.26}$	56.73
PN [56]	$64.44_{\pm 1.39}$	**$60.93_{\pm 1.15}$**	$51.78_{\pm 0.95}$	$40.37_{\pm 0.59}$	$40.92_{\pm 0.31}$	$28.21_{\pm 0.21}$	$78.89_{\pm 0.32}$	$66.95_{\pm 0.40}$	$76.60_{\pm 0.41}$	56.57
DGN [57]	$65.19_{\pm 1.79}$	$58.91_{\pm 0.93}$	$50.76_{\pm 0.92}$	$40.06_{\pm 0.60}$	$41.30_{\pm 0.32}$	$28.32_{\pm 0.36}$	$81.34_{\pm 0.31}$	$69.25_{\pm 0.35}$	$78.06_{\pm 0.42}$	57.02
FA [1]	$53.57_{\pm 0.00}$	$59.26_{\pm 0.00}$	$43.02_{\pm 0.49}$	$27.76_{\pm 0.29}$	$31.51_{\pm 0.00}$	$26.69_{\pm 0.50}$	$29.85_{\pm 0.00}$	$23.23_{\pm 0.00}$	$39.24_{\pm 0.00}$	37.13
SDRF [48]	$63.88_{\pm 1.68}$	$56.40_{\pm 0.89}$	$40.99_{\pm 0.62}$	$40.74_{\pm 0.45}$	$41.44_{\pm 0.37}$	$28.95_{\pm 0.33}$	$81.42_{\pm 0.26}$	*$69.37_{\pm 0.31}$*	$77.74_{\pm 0.42}$	55.66
FoSR [27]	$56.65_{\pm 0.93}$	$50.01_{\pm 1.37}$	*$53.73_{\pm 1.08}$*	$40.26_{\pm 0.50}$	$41.83_{\pm 0.28}$	$28.80_{\pm 0.35}$	$81.79_{\pm 0.26}$	$67.99_{\pm 0.37}$	$78.26_{\pm 0.39}$	55.48
SJLR (ours)	**$71.75_{\pm 1.50}$**	*$60.13_{\pm 0.89}$*	**$55.16_{\pm 0.95}$**	**$41.19_{\pm 0.46}$**	$41.86_{\pm 0.29}$	$29.89_{\pm 0.20}$	*$81.95_{\pm 0.25}$*	**$69.50_{\pm 0.33}$**	*$78.60_{\pm 0.33}$*	**58.89**

train set contains 20 nodes of each class while the rest of the nodes are used for validation. As for the other datasets, we use a 60/20/20 split of the nodes, meaning that 60% of the nodes are assigned for training, 20% for validation, and 20% for testing.

6.2 Implementation Details

All methods are implemented using PyTorch and PyG [15]. We use the same architectural components in all techniques for a fair comparison. We use SGC [51] or GCN [29] as graph convolutional layers. We implemented SDRF [48] at our best understanding because there was not an available implementation of the method at the time of conducting the experiments. However, we use JLC instead of BFC in our implementation of SDRF [48] because of the significant computational resources required to run the hyperparameter optimization using BFC. The hyperparameter search space for each method is defined as follows: 1) learning rate $lr \in [0.005, 0.02]$; 2) weight decay $wd \in [0.0001, 0.001]$; 3) hidden units of each graph convolutional layer $hu \in \{16, 32, 64, 128\}$; 4) dropout $d \in [0.3, 0.7]$; 5) the number of layers $L \in \{2, 3, 4\}$; 6) percentage of added and dropped edges $p_A, p_D \in [0, 1]$; 7) $\alpha \in [0, 1]$; 8) scale $s \in \{0.1, 1, 10, 50, 100\}$ for PN; 9) number of clusters $c \in \{3, 4, \ldots, 10\}$ and balancing factor $bf \in [0.0005, 0.05]$ for DGN; 10) $\alpha_{\text{GDC}} \in [0.01, 0.2]$ and $k \in \{16, 32, 64, 128\}$ for GDC; 10) stochasticity level $\tau \in [1, 500]$, iterations $it \in \{20, 21, \ldots, 4000\}$, and Ricci curvature upper-bound $C^+ \in [0.1, 40]$ for SDRF; and 11) number of SoFR iterations $itF \in \{1, 2, \ldots, 150\}$. We use Rectified Linear Unit (ReLU) and log-softmax as activation functions in our GNN architectures. For GDC, we apply weight decay regularization only in the first graph convolutional layer, otherwise we do not get comparable results as in [19]. All methods are trained for 1,000 epochs using Adam optimizer [28]. We do not use early stopping or learning schedulers for any method. We make all graphs undirected, and we also remove all the self-loops from the input graph. The code is publicly available[3] under the MIT license.

6.3 Results

Tables 2 and 3 show the results for SGC and GCN, respectively. SJLR shows the overall best performance in both cases. We notice two general trends: 1) rewiring methods such as DE and SJLR dominate in almost all datasets for the experiment with SGC, and 2) GDC leads in the homophilous datasets Cora and Pubmed with GCN. Our theoretical results are based on the assumption that there are no non-linear activation functions, so perhaps some nuances are missed for GNNs like GCN. Similarly, we notice that SJLR outperforms SDRF [48] and FoSR [27] in all datasets. For SJLR and SDRF, both methods use the same JLC metric in Tables 2 and 3, and therefore we are assessing their performance based on how the edges are added or removed. We argue that SJLR is a critical improvement over SDRF regarding the practical adoption of curvature-based methods

[3] https://github.com/jhonygiraldo/SJLR

Table 4: Ablation study about dropping and adding edges in SJLR with the SGC model as backbone.

Dropping	Adding	Cornell	Texas	Wisconsin	Chameleon	Squirrel	Actor	Cora	Citeseer	Pubmed
✗	✗	$53.40_{\pm 2.11}$	$56.69_{\pm 1.78}$	$47.90_{\pm 1.73}$	$38.40_{\pm 0.69}$	$40.52_{\pm 0.54}$	$29.93_{\pm 0.16}$	$76.94_{\pm 1.31}$	$67.45_{\pm 0.80}$	$71.79_{\pm 2.13}$
✓	✗	$59.26_{\pm 1.61}$	$56.16_{\pm 0.85}$	$53.64_{\pm 0.97}$	$39.95_{\pm 0.81}$	$41.27_{\pm 0.39}$	$30.04_{\pm 0.19}$	$80.86_{\pm 0.76}$	$\mathbf{68.48}_{\pm 0.51}$	$75.19_{\pm 1.51}$
✗	✓	$54.07_{\pm 2.36}$	$56.79_{\pm 1.93}$	$45.44_{\pm 1.95}$	$38.20_{\pm 0.58}$	$41.57_{\pm 0.32}$	$29.86_{\pm 0.19}$	$73.90_{\pm 2.01}$	$66.20_{\pm 1.05}$	$69.56_{\pm 2.36}$
✓	✓	$\mathbf{67.37}_{\pm 1.64}$	$\mathbf{58.40}_{\pm 1.48}$	$\mathbf{55.42}_{\pm 0.92}$	$\mathbf{40.17}_{\pm 0.49}$	$\mathbf{41.91}_{\pm 0.34}$	$\mathbf{30.81}_{\pm 0.18}$	$\mathbf{81.24}_{\pm 0.77}$	$68.39_{\pm 0.69}$	$\mathbf{76.28}_{\pm 0.96}$

The best result on each dataset are shown in **bold**.

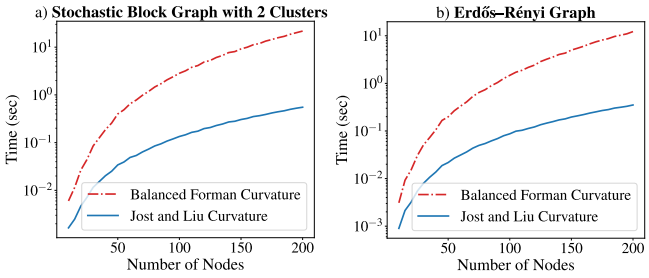

Figure 4: Average running time for BFC and JLC for variations in the number of nodes.

in GNNs. Finally, we remark that some methods like GDC [19] and FA [1] require specific architectural changes to work properly. For example, we achieve the results of GDC only when applying weight decay in the first graph convolutional layer.

Figure 4 shows the average running time over ten repetitions to compute the BFC and JLC metrics in a stochastic block graph and one Erdős-Rényi graph. We notice the large gap between the computation time of the JLC and BFC, which makes JLC more suitable in practice.

6.4 Ablation Study

We conduct an ablation study to examine the influence of the addition and removal of edges in SJLR, employing SGC as the backbone model in alignment with our theoretical findings. To explore this, we perform hyperparameter optimization as outlined in Section 6.1, and we obtain the results summarized in Table 4. The findings suggest that the addition and removal of nodes are complementary and additive in performance. For example, for nearly all datasets, incorporating the addition and removal of edges leads to better performance than performing one strategy alone. This is an important difference regarding recent works [27, 35] where only edges are added or removed. We theoretically (Theorem 3) and empirically (Tables 2, 3, and 4) show that both, removing and adding edges is required to find a good compromise in the over-smoothing over-squashing trade-off.

6.5 Limitations

One of the limitations of SJLR is the optimization of hyperparameters. Due to the expanded search space, finding an optimal set of hyperparameters for SJLR through random search becomes more challenging compared to simpler methods. Another significant limitation is that the current implementation of SJLR relies on a bank $\mathcal{E}_a$ of good edges to add, which is based solely on the triangles of edges with the most negative curvature. As a result, it is not possible to have edges that directly connect long-distance nodes. A potential solution to address this limitation is to explore more sophisticated curvature metrics that are weighted with the shortest path distance between nodes, as proposed in [35]. However, implementing such an approach would introduce a high computational cost, making it impractical for large-scale graph applications. These limitations underscore the need for further research in developing efficient curvature metrics that can effectively account for larger distances in the graph.

7 CONCLUSIONS

In this work, we have established a connection between the challenges of over-smoothing and over-squashing in GNNs. We showed how both issues are intrinsically related to the spectral gap of the normalized Laplacian matrix. Through the application of the Cheeger inequality, we have revealed the existence of a trade-off between over-smoothing and over-squashing, highlighting the inherent limitations of simultaneously addressing both problems from a topological perspective. To tackle these challenges, we have introduced the SJLR algorithm, which utilizes a bound of the Ollivier's Ricci curvature. SJLR offers a computationally efficient solution compared to previous methods, such as SDRF, while still preserving essential theoretical properties. SJLR outperformed previous methods in homophilous and heterophyllous graph datasets for node classification. Most importantly, this work presented a crucial yet simple theoretical contribution to the fundamental problems of over-smoothing and over-squashing in GNNs.

This work opens several research directions. For example, integrating neural architecture search [5] into curvature-based methods could facilitate the challenge of hyperparameter optimization in the context of SJLR.

Acknowledgements. This work was supported by the DATAIA Institute as part of the "Programme d'Investissement d'Avenir", (ANR-17-CONV-0003) operated by CentraleSupélec, and by ANR (French National Research Agency) under the JCJC project GraphIA (ANR-20-CE23-0009-01).

A PROOF OF THEOREM 3

Proof. Let $\mathbf{P} = \mathbf{D}^{-1}\mathbf{A}$ be the random walk transition matrix, and let $f : \mathcal{V} \to \mathbb{R}$ be any initial distribution with vector representation $\mathbf{f} \in \mathbb{R}^{N\times 1}$. If we want to measure the distance between $\mathbf{f}^\top\mathbf{P}^s$ and the stationary distribution in the ℓ_2 norm we need to compute $\|\mathbf{f}^\top\mathbf{P}^s - \boldsymbol{\pi}\| = \|\mathbf{f}^\top\mathbf{P}^s - \frac{\mathbf{1}^\top\mathbf{D}}{\mathrm{vol}(G)}\|$. Let $\boldsymbol{\phi}_i$ be the orthonormal eigenfunction

associated with λ_i. We know that $\boldsymbol{\phi}_1 = \frac{\mathbf{1}^\top \mathbf{D}^{\frac{1}{2}}}{\sqrt{\text{vol}(G)}}$. Therefore, we have:

$$a_1 \boldsymbol{\phi}_1 \mathbf{D}^{\frac{1}{2}} = \frac{1}{\sqrt{\text{vol}(G)}} \frac{\mathbf{1}^\top \mathbf{D}^{\frac{1}{2}}}{\sqrt{\text{vol}(G)}} \mathbf{D}^{\frac{1}{2}} = \frac{\mathbf{1}^\top \mathbf{D}}{\text{vol}(G)},$$
$$\text{where } a_1 = \frac{\langle \mathbf{f}^\top \mathbf{D}^{-\frac{1}{2}}, \mathbf{1}^\top \mathbf{D}^{\frac{1}{2}} \rangle}{\|\mathbf{1}^\top \mathbf{D}^{\frac{1}{2}}\|} = \frac{1}{\sqrt{\text{vol}(G)}}. \tag{9}$$

As a consequence we have that:

$$\begin{aligned} \|\mathbf{f}^\top \mathbf{P}^s - \frac{\mathbf{1}^\top \mathbf{D}}{\text{vol}(G)}\| &= \|\mathbf{f}^\top \mathbf{P}^s - a_1 \boldsymbol{\phi}_1 \mathbf{D}^{\frac{1}{2}}\| \\ &= \|\mathbf{f}^\top \mathbf{D}^{-\frac{1}{2}} (\mathbf{I} - \mathbf{L}_{\text{sym}})^s \mathbf{D}^{\frac{1}{2}} - a_1 \boldsymbol{\phi}_1 \mathbf{D}^{\frac{1}{2}}\|, \end{aligned} \tag{10}$$

since $\mathbf{P} = \mathbf{D}^{-\frac{1}{2}} (\mathbf{I} - \mathbf{L}_{\text{sym}}) \mathbf{D}^{\frac{1}{2}}$. Suppose we write $\mathbf{f}^\top \mathbf{D}^{-\frac{1}{2}} = \sum_i a_i \boldsymbol{\phi}_i$, then:

$$\begin{aligned} \|\mathbf{f}^\top \mathbf{P}^s - \boldsymbol{\pi}\| &= \|\sum_i a_i \boldsymbol{\phi}_i (\mathbf{I} - \mathbf{L}_{\text{sym}})^s \mathbf{D}^{\frac{1}{2}} - a_1 \boldsymbol{\phi}_1 (1 - \lambda_1)^s \mathbf{D}^{\frac{1}{2}}\| \\ &= \|\sum_{i \neq 1} a_i \boldsymbol{\phi}_i (1 - \lambda_i)^s \mathbf{D}^{\frac{1}{2}}\| = \|\sum_{i \neq 1} (1 - \lambda_i)^s a_i \boldsymbol{\phi}_i \mathbf{D}^{\frac{1}{2}}\| \\ &\leq e^{-s\lambda'} \frac{\max_i \sqrt{d_i}}{\min_j \sqrt{d_j}}, \\ &\text{where } \lambda' = \begin{cases} \lambda_2 & \text{if } 1 - \lambda_2 \geq \lambda_N - 1, \\ 2 - \lambda_N & \text{otherwise.} \end{cases} \end{aligned} \tag{11}$$

Thus, we can compute the value of s such that $\|\mathbf{f}^\top \mathbf{P}^s - \boldsymbol{\pi}\| \leq \epsilon$ as follows:

$$\begin{aligned} s &\geq \frac{1}{\lambda' \log\left(\max_i \sqrt{d_i} / \epsilon \min_j \sqrt{d_j}\right)} \\ \rightarrow \lambda' &\geq \frac{1}{s \log\left(\max_i \sqrt{d_i} / \epsilon \min_j \sqrt{d_j}\right)}. \end{aligned} \tag{12}$$

In general, suppose that we have a weighted graph with weights $w(u, v)$, and so we have eigenvalues λ_i with $\lambda_N - 1 \geq 1 - \lambda_2$. Therefore, we can modify the weights with some constant c as follows:

$$w'(u, v) = \begin{cases} w(v, v) + c d_v & \text{if } u = v, \\ w(u, v) & \text{otherwise.} \end{cases} \tag{13}$$

Notice that the lazy walk is given by $c = 1$. The weighted graph has eigenvalues:

$$\lambda'_k = \frac{\lambda_k}{1 + c} = \frac{2\lambda_k}{\lambda_N + \lambda_2}, \text{ where } c = \frac{\lambda_2 + \lambda_N}{2} - 1 \leq \frac{1}{2}. \tag{14}$$

We thus have $1 - \lambda'_2 = \lambda'_N - 1 = \frac{\lambda_N - \lambda_2}{\lambda_N + \lambda_2}$. Since $c \leq \frac{1}{2}$ and we have that $\lambda'_k \geq \frac{2\lambda_k}{2 + \lambda_k} \geq \frac{2}{3}\lambda_k$ for $\lambda_k \leq 1$. Particularly, we can set:

$$\lambda = \lambda'_2 = \frac{2\lambda_2}{\lambda_N + \lambda_2} \geq \frac{2}{3}\lambda_2. \tag{15}$$

Therefore, the modified random walk corresponding to the weighted function w' has an improved bound for the convergence rate in ℓ_2 distance as follows:

$$\frac{1}{\lambda} \log\left(\frac{\max_i \sqrt{d_i}}{\epsilon \min_j \sqrt{d_j}}\right). \tag{16}$$

Please see [11] for further details. Finally, using the Cheeger inequality in (1) we have that:

$$2h_G \geq \lambda_2 \geq \frac{1}{s} \log\left(\frac{\max_i \sqrt{d_i}}{\epsilon \min_j \sqrt{d_j}}\right) \rightarrow 2h_G \geq \frac{1}{s} \log\left(\frac{\max_i \sqrt{d_i}}{\epsilon \min_j \sqrt{d_j}}\right). \tag{17}$$

□

B MIXING TIME AND CHEEGER CONSTANT

The concepts of random walks, Cheeger constant, and convergence to the stationary distribution are tightly related to Markov chains, conductance (Cheeger constant), and the mixing time. We can represent any Markov chain as a random walk on some weighted directed graph G. The mixing time $\tau(\epsilon)$ of an ergodic Markov chain is the time until the Markov chain is close to its stationary distribution, *i.e.*, s. We can find an upper bound for $\tau(\epsilon)$ as follows:

Theorem 8 (Sinclair [46]). *Let h_G be the Cheeger constant of an ergodic and reversible Markov chain. For $\epsilon > 0$ we have that:*

$$s = \tau(\epsilon) \leq \frac{2}{h_G^2} \left(\log\left(\frac{1}{\epsilon}\right) + \log\left(\frac{1}{\pi_*}\right)\right), \tag{18}$$

Where $\pi_* = \min_{j \in \Omega} \boldsymbol{\pi}(j)$, and Ω is the state space of the Markov chain.

From Theorem 8, and since $h_G > 0$ and $\lambda_2 > 0$ for connected graphs, we have that $\lim_{h_G \to \infty} \frac{2}{h_G^2} \left(\log\left(\frac{1}{\epsilon}\right) + \log\left(\frac{1}{\pi_*}\right)\right) = 0$ so that if $h_G \rightarrow \infty$ then $s \rightarrow 0$. Similarly, we have that $\lim_{s \to \infty} \frac{2}{s} \left(\log\left(\frac{1}{\epsilon}\right) + \log\left(\frac{1}{\pi_*}\right)\right) = 0$, so that if $s \rightarrow \infty$ then $h_G^2 \rightarrow 0$. Finally, considering the results from Theorem 3 we have that $s \rightarrow \infty \iff h_G \rightarrow 0$ and $h_G \rightarrow \infty \iff s \rightarrow 0$.

C SWEET POINT

We can propose an optimization problem to find the "sweet point" in the over-smoothing vs. over-squashing trade-off as follows:

$$\arg\min_{\mathcal{E}' \in \mathcal{V} \times \mathcal{V}} \; h_{G'} + \rho f(\lambda'_2, \epsilon), \tag{19}$$

where $G' = (\mathcal{V}, \mathcal{E}')$, ρ is a regularization parameter, $f(\lambda'_2, \epsilon)$ is the function of mixing steps of G', λ'_2 is the spectral gap of G', and ϵ is a hyperparameter. The optimization problem in (19) poses several challenges:

(1) Since $\mathcal{E}' \in \mathcal{V} \times \mathcal{V}$, the optimal solution could be very different from $\mathcal{E}$, destroying the original structural information of the graph.
(2) Computing the Cheeger constant of a finite graph is an NP-hard problem [17], and the algorithm to solve (19) could require calculating $h_{G'}$ several times.
(3) The optimization of the hyperparameters ρ and ϵ should be data-driven for the specific graph learning task.

As a consequence, solving the combinatorial optimization problem in (19) is practically infeasible. Instead of trying to solve (19) directly, we have proposed SJLR as a heuristic algorithm. In SJLR, we relate to the Cheeger constant indirectly through Corollaries 5 and 7 with the JLC metric. Similarly, we relate to the mixing step $f(\lambda'_2, \epsilon)$ through the Lemma 2 when dropping edges in Algorithm 1.

REFERENCES

[1] Uri Alon and Eran Yahav. 2021. On the bottleneck of graph neural networks and its practical implications. In *International Conference on Learning Representations*.

[2] Adrien Benamira, Benjamin Devillers, Etienne Lesot, Ayush K. Ray, Manal Saadi, and Fragkiskos D. Malliaros. 2019. Semi-Supervised Learning and Graph Neural Networks for Fake News Detection. In *International Conference on Advances in Social Networks Analysis and Mining*.

[3] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. 2014. Spectral networks and locally connected networks on graphs. In *International Conference on Learning Representations*.

[4] Deng Cai and Wai Lam. 2020. Graph transformer for graph-to-sequence learning. In *AAAI Conference on Artificial Intelligence*.

[5] Shaofei Cai, Liang Li, Jincan Deng, Beichen Zhang, Zheng-Jun Zha, Li Su, and Qingming Huang. 2021. Rethinking graph neural architecture search from message-passing. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*.

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

[7] Chaoqi Chen, Yushuang Wu, Qiyuan Dai, Hong-Yu Zhou, Mutian Xu, Sibei Yang, Xiaoguang Han, and Yizhou Yu. 2022. A Survey on Graph Neural Networks and Graph Transformers in Computer Vision: A Task-Oriented Perspective. *arXiv preprint arXiv:2209.13232* (2022).

[8] Deli Chen, Yankai Lin, Wei Li, Peng Li, Jie Zhou, and Xu Sun. 2020. Measuring and Relieving the Over-Smoothing Problem for Graph Neural Networks from the Topological View. In *AAAI Conference on Artificial Intelligence*.

[9] Yu Chen, Lingfei Wu, and Mohammed Zaki. 2020. Iterative deep graph learning for graph neural networks: Better and robust node embeddings. In *Advances in Neural Information Processing Systems*.

[10] Eli Chien, Jianhao Peng, Pan Li, and Olgica Milenkovic. 2021. Adaptive universal generalized PageRank graph neural network. In *International Conference on Learning Representations*.

[11] Fan R. K. Chung. 1997. *Spectral graph theory*. Number 92. American Mathematical Soc.

[12] Francesco Di Giovanni, Lorenzo Giusti, Federico Barbero, Giulia Luise, Pietro Lio, and Michael Bronstein. 2023. On Over-Squashing in Message Passing Neural Networks: The Impact of Width, Depth, and Topology. *arXiv preprint arXiv:2302.02941* (2023).

[13] Alexandre Duval, Victor Schmidt, Alex Hernández-García, Santiago Miret, Fragkiskos D. Malliaros, Yoshua Bengio, and David Rolnick. 2023. FAENet: Frame Averaging Equivariant GNN for Materials Modeling. In *International Conference on Machine Learning*.

[14] Vijay Prakash Dwivedi, Ladislav Rampášek, Mikhail Galkin, Ali Parviz, Guy Wolf, Anh Tuan Luu, and Dominique Beaini. 2022. Long range graph benchmark. In *Advances in Neural Information Processing Systems*.

[15] Matthias Fey and Jan Eric Lenssen. 2019. Fast Graph Representation Learning with PyTorch Geometric. In *International Conference on Learning Representations - Workshops*.

[16] Pablo Gainza, Freyr Sverrisson, Frederico Monti, Emanuele Rodola, D Boscaini, Michael Bronstein, and BE Correia. 2020. Deciphering interaction fingerprints from protein molecular surfaces using geometric deep learning. *Nature Methods* 17, 2 (2020), 184–192.

[17] Michael R Garey, David S Johnson, and Larry Stockmeyer. 1974. Some simplified NP-complete problems. In *ACM Symposium on Theory of Computing*.

[18] Johannes Gasteiger, Aleksandar Bojchevski, and Stephan Günnemann. 2019. Predict then propagate: Graph neural networks meet personalized PageRank. In *International Conference on Learning Representations*.

[19] Johannes Gasteiger, Stefan Weißenberger, and Stephan Günnemann. 2019. Diffusion improves graph learning. In *Advances in Neural Information Processing Systems*.

[20] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. 2017. Neural message passing for quantum chemistry. In *International Conference on Machine Learning*.

[21] Jhony H Giraldo, Sajid Javed, Naoufel Werghi, and Thierry Bouwmans. 2021. Graph CNN for moving object detection in complex environments from unseen videos. In *IEEE/CVF International Conference on Computer Vision*.

[22] R. Hamilton. 1998. The Ricci flow on surfaces. *Mathematics and General Relativity* 71 (1998), 237–262.

[23] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*.

[24] Kai Han, Yunhe Wang, Jianyuan Guo, Yehui Tang, and Enhua Wu. 2022. Vision GNN: An image is worth graph of nodes. In *Advances in Neural Information Processing Systems*.

[25] Wei Huang, Yayong Li, Weitao Du, Jie Yin, Richard Yi Da Xu, Ling Chen, and Miao Zhang. 2022. Towards Deepening Graph Neural Networks: A GNTK-based Optimization Perspective. In *International Conference on Learning Representations*.

[26] Jürgen Jost and Shiping Liu. 2014. Ollivier's Ricci curvature, local clustering and curvature-dimension inequalities on graphs. *Discrete & Computational Geometry* 51, 2 (2014), 300–322.

[27] Kedar Karhadkar, Pradeep Kr Banerjee, and Guido Montúfar. 2023. FoSR: First-order spectral rewiring for addressing oversquashing in GNNs. In *International Conference on Learning Representations*.

[28] Diederik P Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*.

[29] Thomas N Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *International Conference on Learning Representations*.

[30] Devin Kreuzer, Dominique Beaini, Will Hamilton, Vincent Létourneau, and Prudencio Tossou. 2021. Rethinking graph transformers with spectral attention. In *Advances in Neural Information Processing Systems*.

[31] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *Nature* 521, 7553 (2015), 436–444.

[32] Guohao Li, Matthias Muller, Ali Thabet, and Bernard Ghanem. 2019. DeepGCNs: Can GCNs go as deep as CNNs?. In *IEEE/CVF International Conference on Computer Vision*.

[33] Qimai Li, Zhichao Han, and Xiao-Ming Wu. 2018. Deeper Insights into Graph Convolutional Networks for Semi-Supervised Learning. In *AAAI Conference on Artificial Intelligence*.

[34] Yong Lin, Linyuan Lu, and Shing-Tung Yau. 2011. Ricci curvature of graphs. *Tohoku Mathematical Journal, Second Series* 63, 4 (2011), 605–627.

[35] Yang Liu, Chuan Zhou, Shirui Pan, Jia Wu, Zhao Li, Hongyang Chen, and Peng Zhang. 2023. CurvDrop: A Ricci Curvature Based Approach to Prevent Graph Neural Networks from Over-Smoothing and Over-Squashing. In *ACM Web Conference*.

[36] Andrew Kachites McCallum, Kamal Nigam, Jason Rennie, and Kristie Seymore. 2000. Automating the construction of internet portals with machine learning. *Information Retrieval* 3, 2 (2000), 127–163.

[37] Galileo Namata, Ben London, Lise Getoor, Bert Huang, and U Edu. 2012. Query-driven active surveying for collective classification. In *International Workshop on Mining and Learning with Graphs*.

[38] Yann Ollivier. 2009. Ricci curvature of Markov chains on metric spaces. *Journal of Functional Analysis* 256, 3 (2009), 810–864.

[39] Kenta Oono and Taiji Suzuki. 2020. Graph neural networks exponentially lose expressive power for node classification. In *International Conference on Learning Representations*.

[40] Hongbin Pei, Bingzhe Wei, Kevin Chen-Chuan Chang, Yu Lei, and Bo Yang. 2020. Geom-GCN: Geometric graph convolutional networks. In *International Conference on Learning Representations*.

[41] Wieke Prummel, Jhony H Giraldo, Anastasia Zakharova, and Thierry Bouwmans. 2023. Inductive Graph Neural Networks for Moving Object Segmentation. In *IEEE International Conference on Image Processing*.

[42] Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. 2020. DropEdge: Towards deep graph convolutional networks on node classification. In *International Conference on Learning Representations*.

[43] Benedek Rozemberczki, Carl Allen, and Rik Sarkar. 2021. Multi-scale attributed node embedding. *Journal of Complex Networks* 9, 2 (2021), 1–22.

[44] Aliaksei Sandryhaila and Jose M. F. Moura. 2014. Discrete signal processing on graphs: Frequency analysis. *IEEE Transactions on Signal Processing* 62, 12 (2014), 3042–3054.

[45] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. 2008. Collective classification in network data. *AI magazine* 29, 3 (2008), 93–93.

[46] Alistair Sinclair. 2012. *Algorithms for random generation and counting: a Markov chain approach*. Springer Science & Business Media.

[47] Jie Tang, Jimeng Sun, Chi Wang, and Zi Yang. 2009. Social influence analysis in large-scale networks. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

[48] Jake Topping, Francesco Di Giovanni, Benjamin Paul Chamberlain, Xiaowen Dong, and Michael M Bronstein. 2022. Understanding over-squashing and bottlenecks on graphs via curvature. In *International Conference on Learning Representations*.

[49] Werner Uwents, Gabriele Monfardini, Hendrik Blockeel, Marco Gori, and Franco Scarselli. 2011. Neural networks for relational learning: An experimental comparison. *Machine Learning* 82, 3 (2011), 315–349.

[50] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2018. Graph Attention Networks. In *International Conference on Learning Representations*.

[51] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. 2019. Simplifying graph convolutional networks. In *International Conference on Machine Learning*.

[52] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. 2020. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems* (2020).

[53] Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and Tie-Yan Liu. 2021. Do transformers really perform badly for graph representation?. In *Advances in Neural Information Processing Systems*.

[54] Seongjun Yun, Minbyul Jeong, Raehyun Kim, Jaewoo Kang, and Hyunwoo J Kim. 2019. Graph transformer networks. In *Advances in neural information processing systems*.

[55] Hanqing Zeng, Muhan Zhang, Yinglong Xia, Ajitesh Srivastava, Andrey Malevich, Rajgopal Kannan, Viktor Prasanna, Long Jin, and Ren Chen. 2021. Decoupling the Depth and Scope of Graph Neural Networks. In *Advances in Neural Information Processing Systems*.

[56] Lingxiao Zhao and Leman Akoglu. 2020. PairNorm: Tackling oversmoothing in GNNs. In *International Conference on Learning Representations*.

[57] Kaixiong Zhou, Xiao Huang, Yuening Li, Daochen Zha, Rui Chen, and Xia Hu. 2020. Towards deeper graph neural networks with differentiable group normalization. In *Advances in Neural Information Processing Systems*.

[58] Hao Zhu and Piotr Koniusz. 2021. Simple spectral graph convolution. In *International Conference on Learning Representations*.

[59] Marinka Zitnik and Jure Leskovec. 2017. Predicting multicellular function through multi-layer tissue networks. *Bioinformatics* 33, 14 (2017), i190–i198.