SLOG: An Inductive Spectral Graph Neural Network Beyond Polynomial Filter

Haobo Xu*[1] Yuchen Yan*[2] Dingsu Wang[2] Zhe Xu[2]
Zhichen Zeng[2] Tarek F. Abdelzaher[2] Jiawei Han[2] Hanghang Tong[2]

Abstract

Graph neural networks (GNNs) have exhibited superb power in many graph related tasks. Existing GNNs can be categorized into spatial GNNs and spectral GNNs. The spatial GNNs primarily capture the local information around each node, while the spectral GNNs are able to operate on the frequency signals of the entire graph. However, most, if not all, existing spectral GNNs are faced with two limitations: (1) the *polynomial limitation* that for most spectral GNNs, the expressive power in the spectral domain is limited to polynomial filters; and (2) the *transductive limitation* that for the node-level task, most spectral GNNs can only be applied on relatively small-scale graphs in transductive setting. In this paper, we propose a novel spectral graph neural network named SLOG to solve the above two limitations. For the *polynomial limitation*, SLOG proposes a novel filter with real-valued order with geometric interpretability, mathematical feasibility and adaptive filtering ability to go beyond polynomial. For the *transductive limitation*, SLOG combines the subgraph sampling technique in spatial GNNs and the signal processing technique in spectral GNNs together to make itself tailored to the inductive node-level tasks on large-scale graphs. Extensive experimental results on 16 datasets demonstrate the superiority of SLOG in inductive homophilic and heterophilic node classification task.

1. Introduction

In the era of big data and AI (Ban et al., 2021; 2023; Wei et al., 2023; Liu et al., 2020a;b; Roach et al., 2020; Du et al., 2021; Lin et al., 2024; Wei et al., 2024), graph neural networks (Hamilton et al., 2017; Veličković et al., 2018) (GNNs) have demonstrated strong learning ability on graph related tasks such as node classification (Kipf & Welling, 2016; Wu et al., 2019; He et al., 2021; Yan et al., 2022a; Liu et al., 2023; Fu et al., 2024), link prediction (Zhang & Chen, 2018; Yan et al., 2024a;b; Wang et al., 2023a), network alignment (Yan et al., 2021a;b; 2022b; Zeng et al., 2023a; 2024), node clustering (Fu et al., 2020; Jing et al., 2022; 2024; Li et al., 2022a; Zeng et al., 2023b; Fu et al., 2023) and knowledge graph reasoning (Wang et al., 2018; Vashishth et al., 2019; Liu et al., 2021; 2022; Wang et al., 2022; 2023b).

Most of existing GNNs can be divided into two main categories: spatial GNNs and spectral GNNs. The spatial GNNs are designed based on the *subgraph sampling technique* and the message-passing mechanism in the spatial domain.[1] The spatial GNNs *sample* the *local topological information* around each node. For example, GraphSAGE (Hamilton et al., 2017) samples a two-hop subgraph around the target node for message-passing and obtains the embedding for the node. GAT (Veličković et al., 2018) adopts the self-attention technique to assign different weights to different edges in the sampled subgraph. Different from the spatial GNNs focusing on capturing the local information around each node, the spectral GNNs pay more attention to the frequency signals of the whole graph. Most spectral GNNs are developed based on the *graph signal processing technique* (Wang & Zhang, 2022) in the spectral domain, which can capture different frequency signals of the graph. To name a few, ChebNet (Defferrard et al., 2016) utilizes a Chebyshev polynomial filter, GNN-LF/HF (Zhu et al., 2021b) embraces a rational function, and BernNet (He et al., 2021) applies Bernstein polynomials as the filter.

However, most, if not all, existing spectral GNNs are faced with with two limitations: the *polynomial limitation* and the *transductive limitation*. Specifically, the *polynomial limitation* means that for most spectral GNNs, the expressive power in the spectral domain is limited to polynomial filters, i.e., the orders of the filters are integer-valued.[2] The order

*Equal contribution [1]Tsinghua University [2]University of Illinois Urbana-Champaign. Correspondence to: Yuchen Yan <yucheny5@illinois.edu>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024. Copyright 2024 by the author(s).

[1]Dealing with small-scale graphs, the spatial GNNs may directly aggregate node information from all neighbors, which can also be seen as a full neighborhood sampling.

[2]The filters in GNN-LF/HF (Zhu et al., 2021b), ARMA (Bianchi et al., 2021), CayleyNet (Levie et al., 2018), and so

of the polynomial filters, with the only exception of (Yan et al., 2023), needs to be fixed as a hyper-parameter before training and lacks flexibility. For the *transductive limitation*, it refers to the fact (Kipf & Welling, 2016; Liu et al., 2022; Yan et al., 2023) that in terms of node-level tasks, most spectral GNNs can only be applied to the transductive setting on relatively small-scale graphs and can not accommodate to the inductive setting in real-world large-scale graphs, where new nodes keep emerging. This is because these existing spectral GNNs have to precompute the graph signal processing operators such as multiplication on the adjacency matrix of the entire graph.

To address the above two limitations of existing spectral GNNs, in this paper, we propose a novel model named SLOG with three sub-models with different components/layers: SLOG(B) (Base), SLOG(N) (Nolinear), and SLOG(L) (Local). The key idea of SLOG(B) is two-fold: firstly, to solve the *polynomial limitation*, SLOG(B) proposes a novel filter with real-valued order to go beyond polynomial. In detail, we elucidate that the filter with real-valued order of SLOG(B) enjoys (1) good geometric interpretability in spatial domain; (2) mathematical feasibility in spectral domain; and (3) adaptive filtering ability for different frequency signals (e.g., low/high/band-pass and band-stop) (Section 3.1). Secondly, to resolve the *transductive limitation*, SLOG(B) creatively combines the subgraph sampling technique in spatial GNNs and the signal processing technique in spectral GNNs together, which renders SLOG(B) the inductive ability on large-scale graphs (Section 3.2). Since SLOG(B) only has one linear layer filter, we further propose two sub-models: SLOG(N) and SLOG(L). SLOG(N) extends SLOG(B) to multiple layers and adds more non-linearity into the model (Section 3.2) and SLOG(L) interpolates the global uniform filter in SLOG(B) with local adaptive filter for each subgraph (Section 3.3). Through extensive empirical evaluations on 16 real-world datasets in the node classification task, we corroborate the effectiveness of the proposed SLOG. To summarize, our contributions are three-fold:

- **Insight.** The key idea of our paper is two-fold: (1) designing a real-valued order filter with geometric interpretability, mathematical feasibility and adaptive filtering ability; and (2) combining the subgraph sampling technique in spatial GNNs and the graph signal processing technique in spectral GNNs together.
- **Model.** We propose a large-scale inductive spectral GNN beyond polynomial filter named SLOG, which includes three sub-models: SLOG(B) as the base sub-model, SLOG(N) with non-linearity and SLOG(L) for interpolating the global uniform filter in SLOG(B) with the local adaptive filter for each subgraph. While the three sub-models are differentiated by unique components/layers, they belong to the unified SLOG model.
- **Experiments.** We conduct extensive experiments on 16 datasets and empirically find that the proposed SLOG achieves comparable or better performance than the state-of-the-arts in the inductive homophilic and heterophilic node classification task, which demonstrates the superiority of SLOG.

on, are still built upon polynomial filters.

2. Preliminaries

Notations. We utilize bold uppercase letters for matrices (e.g., $\mathbf{A}$), bold lowercase letters for column vectors (e.g., $\mathbf{u}$) and lowercase letters for scalars (e.g., α). We use the superscript $\top$ for the transpose of matrices and vectors (e.g., $\mathbf{A}^\top$ and $\mathbf{u}^\top$). Consider a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{v_i\}_{i=1}^n$ represents the set of n nodes, and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ denotes the set of edges. We use $\mathbf{X} \in \mathbb{R}^{n \times f}$ to represent the node features of a graph with feature dimension f. The adjacency matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ is defined such that $\mathbf{A}_{ij} = 1$ if $(v_i, v_j) \in \mathcal{E}$, and $\mathbf{A}_{ij} = 0$ otherwise. The degree matrix is $\mathbf{D} = \text{diag}(\{\sum_j \mathbf{A}_{ij}\}_{i=1}^n)$. We introduce $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ as the adjacency matrix augmented with self-loop for each node, where $\mathbf{I}$ is the identity matrix, and $\tilde{\mathbf{D}}$ is the degree matrix of $\tilde{\mathbf{A}}$.

Spectral Graph Theory. The graph Laplacian matrix is defined as $\mathbf{L} = \mathbf{D} - \mathbf{A}$. We use $\mathbf{L}_{\text{sym}} = \mathbf{I} - \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$ for the symmetrically normalized graph Laplacian matrix. Let $\mathbf{L}_{\text{sym}} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^\top$ represents the eigen-decomposition of $\mathbf{L}_{\text{sym}}$, where $\mathbf{U} = [\mathbf{u}_1, \cdots, \mathbf{u}_n]$ is the matrix of eigenvectors, and $\mathbf{\Lambda} = \text{diag}(\{\lambda_i\}_{i=1}^n)$ is the diagonal matrix of eigenvalues. The eigenvalues of $\mathbf{L}_{\text{sym}}$ are bounded by $\lambda_i \in [0, 2)$ (Chung, 1997)[3], which also applies to the eigenvalues $\{\tilde{\lambda}_i\}_{i=1}^n$ of $\tilde{\mathbf{L}}_{\text{sym}}$. In addition, applying a function $g(\cdot)$, also known as a filter, to $\mathbf{L}_{\text{sym}}$ is equivalent to applying $g(\cdot)$ to its eigenvalues (Shuman et al., 2013):

$$g(\mathbf{L}_{\text{sym}}) = \sum_{i=1}^n g(\lambda_i)\mathbf{u}_i\mathbf{u}_i^\top. \tag{1}$$

This is also applicable to $\tilde{\mathbf{L}}_{\text{sym}}$. For a l-th layer simplified graph convolutional network (SGC) (Wu et al., 2019) without non-linear activation function, the final node representation matrix $\mathbf{Z}$ can be formulated as:

$$\mathbf{Z} = (\mathbf{I} - \tilde{\mathbf{L}}_{\text{sym}})^l \mathbf{X}\mathbf{W}, \tag{2}$$

where $\mathbf{X}$ is the node feature matrix and $\mathbf{W}$ is the trainable parameter matrix. Here, the linear graph convolutional layer is equivalent to applying a polynomial function

[3]In this work, we only consider connected graph without bipartite components (i.e., a component which is a bipartite graph).

$g(\mathbf{L}_{\text{sym}}) = (\mathbf{I} - \mathbf{L}_{\text{sym}})^l (l \in \mathbb{N}^+)$ to the graph Laplacian matrix, functioning as a graph filter.

Graph Homophily and Heterophily. The concept of homophily/heterophily addresses the tendency of nodes to connect with others of the same or different classes/labels respectively. There are several interpretations of homophily/heterophily in existing literature, including perspectives at the edge scale (Abu-El-Haija et al., 2019; Zhu et al., 2020; Luan et al., 2021), node scale (Pei et al., 2020), and graph scale (Lim et al., 2021). This paper specifically addresses edge heterophily, defined as the proportion of edges connecting nodes of the different types relative to the total number of edges: $h(\mathcal{G}) = \frac{\left|(v_i, v_j) \in \mathcal{E} | y_i \neq y_j\right|}{|\mathcal{E}|}$, where y_i represents the type of node v_i.

3. Model

In this section, we present the details of the proposed SLOG model. We first introduce the specially designed filter with real-valued order, which is the key component of SLOG to solve the *polynomial limitation* (Section 3.1). Equipped with the filter with real-valued order, we present how the one linear layer filter base model SLOG(B) solves the *transductive limitation* by combining the subgraph sampling technique in spatial GNNs and the frequency signal processing technique in spectral GNNs together (Section 3.2). Then, we enhance the one layer linear SLOG(B) to multi-layer nonlinear SLOG(N) (Section 3.2). To handle the varying degree of heterophily across different parts of the graph[4], we further propose the SLOG(L), which interpolates the global uniform filter from SLOG(B)/SLOG(N) with the local adaptive filter for each subgraph in corresponding SLOG(LB) and SLOG(LN) (Section 3.3). Moreover, a complexity analysis can be found in Appendix A.4.

3.1. Filter Beyond Polynomial

The filter in most existing spectral GNNs (e.g., ChebNet (Defferrard et al., 2016), APPNP (Gasteiger et al., 2018), and SGC (Wu et al., 2019)) can be summarized as follows:

$$g(\mathbf{L}_{\text{sym}}) = \sum_{k=1}^{K} \alpha_k \mathbf{L}_{\text{sym}}^k, \tag{3}$$

where $\mathbf{L}_{\text{sym}}$ is the symmetrically normalized graph Laplacian and K is order of the polynomial. To map the filter in the spectral domain to the K-hop subgraph in the spatial domain, usually, the order K is a fixed positive integer hyper-parameter, which lacks flexibility and can not be optimized as a variable during the training process. Recently, TeDGCN (Yan et al., 2023) redefines the depth/layer of GNNs and successfully builds a filter with real-valued order as: $g(\mathbf{L}_{\text{sym}}) = (\mathbf{I} - \frac{1}{2}\mathbf{L}_{\text{sym}})^d$, where d is a real number and a trainable parameter. Unfortunately, this filter can only capture low/high frequency signals and is not able to function as a band-pass/band-stop filter (Balcilar et al., 2021). Furthermore, TeDGCN has to conduct the eigen-decomposition on $\mathbf{L}_{\text{sym}}$, which means that it is still faced with the *transductive limitation*.

In this subsection, we introduce the key component of SLOG: a filter with real-valued order, $\omega(\cdot)$, to go beyond the polynomial expressiveness in the spectral domain as follows:

$$\omega(\mathbf{L}_{\text{sym}}) = (\mathbf{I} - \frac{1}{2}\mathbf{L}_{\text{sym}})^p (\mathbf{I} + (\mathbf{L}_{\text{sym}} - \mathbf{I})^2)^q, \tag{4}$$

where p and q are two trainable real-valued parameters. We opt for the filter design in Eq. (4) for the following three key properties: *geometric interpretability*, *mathematical feasibility* and *adaptive filtering*:

P1. Geometric Interpretability in the Spatial Domain.

Proposition 3.1. *The* SLOG*'s filter with real-valued order, $\omega(\cdot)$, in the spectral domain can be regarded as the combination of two linear graph convolutional networks in the spatial domain: $\omega(\mathbf{L}_{sym}) = \mathbf{S}_1^p \cdot \mathbf{S}_2^q$, where $\mathbf{S}_1 = \frac{1}{2}(\mathbf{I} + \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}})$ and $\mathbf{S}_2 = \mathbf{I} + (\mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}})^2$.*

The proof can be found in Appendix A.1. Based on Proposition 3.1, the filter in Eq. (4) essentially represents the combination of two linear graph convolutional networks with trainable real-valued depths, which is composed of a series of operations applied to the adjacency matrix $\mathbf{A}$. This process involves symmetric normalization of $\mathbf{A}$, and addition of self-loops to both one-hop and two-hop adjacency matrices.

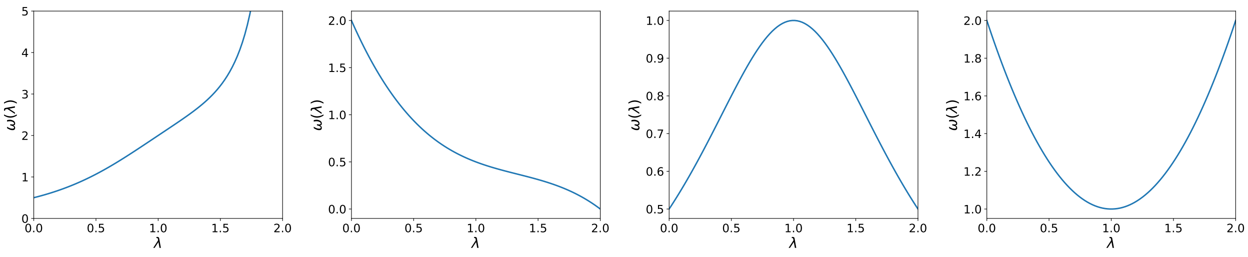
P2. Mathematical Feasibility. The eigenvalues of $\mathbf{L}_{\text{sym}}$ are confined within the range $\lambda_i \in [0, 2)$. And the filter $\omega(\mathbf{L}_{\text{sym}})$ comprises two components: $(\mathbf{I} - \frac{1}{2}\mathbf{L}_{\text{sym}})^p$ and $(\mathbf{I} + (\mathbf{L}_{\text{sym}} - \mathbf{I})^2)^q$. The first part, a polynomial function of $\mathbf{L}_{\text{sym}}$, is positive-definite, leading to the mathematical feasibility to compute its exponentiation by any real number p [5]. Likewise, the second part is also positive-definite and enjoys similar mathematical feasibility.

P3. Adaptive Filtering. The frequency response of the proposed filter under various parameter configurations is illustrated in Figure 1. This demonstrates the filter's ability to be transformed into high-pass, low-pass, band-pass, or band-stop filters by optimizing the parameters p and q in the real number domain.

Given the filter with real-valued order in Eq. (4), we next introduce the details of SLOG and how it can be applied

[4]Please refer to Figure 3 and Section 3.3 for details.

[5]For a positive-definite matrix, computing the real-valued order of the matrix is equivalent to conducting eigen-decomposition and computing the power of the eigenvalues. (Shuman et al., 2013)



(a) High pass $(p = -1, q = -1)$ (b) Low pass $(p = 1, q = 1)$ (c) Band pass $(p = 0, q = -1)$ (d) Band stop $(p = 0, q = 1)$

Figure 1. The frequency responses of the proposed filter with different parameters.

to the inductive setting and run on large-scale graphs in Section 3.2.

3.2. SLOG(B) and SLOG(N)

Algorithm 1 SLOG(B)

```
1: Input: Graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$; node feature $\mathbf{X}$; node $v \in \mathcal{V}$; hop number of subgraphs $K$; maximum neighbor numbers of each depth $\{N_i\}$.
2: Output: Vector representations $\mathbf{z}_v$ for node $v$.
   # Step 1: Sampling $K$-hop subgraph
3: $\mathcal{G}_v(\mathcal{V}_{\mathcal{G}_v}, \mathcal{E}_{\mathcal{G}_v}) \leftarrow SAMPLE(\mathcal{G}, \{v\}, K, \{N_i\})$
   # Step 2: Calculation and Filtering
4: Compute node feature $\mathbf{X}_{\mathcal{G}_v}$ and symmetrically normalized Laplacian matrix $\mathbf{L}_{\mathcal{G}_v}$ of $\mathcal{G}_v$
   # Step 3: Obtaining node representation
5: $\mathbf{Z}_{\mathcal{G}_v} \leftarrow \omega(\mathbf{L}_{\mathcal{G}_v})\mathbf{X}_{\mathcal{G}_v}\mathbf{W}$
6: $\mathbf{z}_v \leftarrow \mathbf{Z}_{\mathcal{G}_v}(v)$
7: return $\mathbf{z}_v$

   # Method: Subgraph sampling strategy
8: function $SAMPLE(\mathcal{G}, \mathcal{V}_i, K, \{N_i\})$
9:     $\mathcal{V}^{(0)} \leftarrow \mathcal{V}_i$
10:    for $k = 1$ to $K$ do
11:        Sample $\mathcal{S}_u \subseteq \mathcal{N}(u)$, s.t. $|\mathcal{S}_u| \leq N_k$ for each $u \in \mathcal{V}^{(k-1)}$, where $\mathcal{N} : v \to 2^{\mathcal{V}}$ is the neighbor function of $\mathcal{G}$.
12:        $\mathcal{V}^{(k)} \leftarrow (\bigcup_{u \in \mathcal{V}^{(k-1)}} \mathcal{S}_u) \cup \mathcal{V}^{(k-1)}$
13:    end for
14:    return $\mathcal{G}(\mathcal{V}^{(K)}, \mathcal{E}^{(K)})$
15: end function
```

In this subsection, we present the details of SLOG(B) and SLOG(N). To handle the inductive setting and large-scale graphs, the key idea of SLOG is to integrate the *subgraph sampling technique* from spatial GNNs with the proposed filter with real-valued order in Section 3.1. In other words, for a given node v, SLOG applies the filter on a sampled subgraph centered on v, whose size is much smaller than the whole graph. In this way, SLOG fits into the inductive setting and avoids the eigen-decomposition of the whole graph, which enables it to be run on large-scale graphs.

Concretely, for a target node v, as shown in Algorithm 1, SLOG(B) contains 3 parts: (1) Firstly, we sample a subgraph $\mathcal{G}_v$ around v: starting from v, we sample a K-hop graph with random node mask. All the sampled nodes and the edges between them form the subgraph $\mathcal{G}_v$; (2) Based on $\mathcal{G}_v$, we calculate the symmetrically normalized Laplacian matrix $\mathbf{L}_{\mathcal{G}_v}$ of $\mathcal{G}_v$ and apply the filter $\omega(\cdot)$ in Eq. (4) on $\mathbf{L}_{\mathcal{G}_v}$; (3) Finally, the representations of nodes in $\mathcal{G}_v$ is obtained as follows:

$$\mathbf{Z}_{\mathcal{G}_v} = \omega(\mathbf{L}_{\mathcal{G}_v})\mathbf{X}_{\mathcal{G}_v}\mathbf{W}, \tag{5}$$

where $\mathbf{W}$ is the parameter matrix. An illustrative example of SLOG(B) is shown in Figure 2(a).

From the above introduction of SLOG(B) and the filter $\omega(\cdot)$ in Section 3.1, we can find that SLOG(B) is able to solve the *polynomial limitation* and the *transductive limitation* with the help of (1) the filter with real-valued order; and (2) the combination of the subgraph sampling technique in spatial GNNs and the frequency signal processing technique in spectral GNNs.

Nevertheless, this simple linear filter in SLOG(B) does not possess non-linearity. To address this, we further propose an enhanced sub-model, SLOG(N), which includes L layers of SLOG(B). In addition, it not only incorporates non-linear activation functions between filters but also introduces residual connections (He et al., 2016). Details of the structure are illustrated in a figure provided in Appendix A.2. The residual connections make the output become an interpolation of the embeddings from previous layer and the embeddings after transformation by Eq. (4). Specifically, in each layer, the representation matrix is updated by:

$$\mathbf{H}_{\mathcal{G}_v}^{(l)} = \sigma(\omega^{(l)}(\mathbf{L}_{\mathcal{G}_v})\mathbf{H}_{\mathcal{G}_v}^{(l-1)}\mathbf{W}_1^{(l)} + \mathbf{H}_{\mathcal{G}_v}^{(l-1)}\mathbf{W}_2^{(l)}), \tag{6}$$

where $\sigma(\cdot)$ denotes the activation function, $\omega^{(l)}(\cdot)$ denotes the filter at the l-th layer, $\mathbf{H}_{\mathcal{G}_v}^{(l)}$ is the representation matrix at the l-th layer, and $\mathbf{W}_1^{(l)}$ and $\mathbf{W}_2^{(l)}$ are the parameter matrices at the l-th layer.

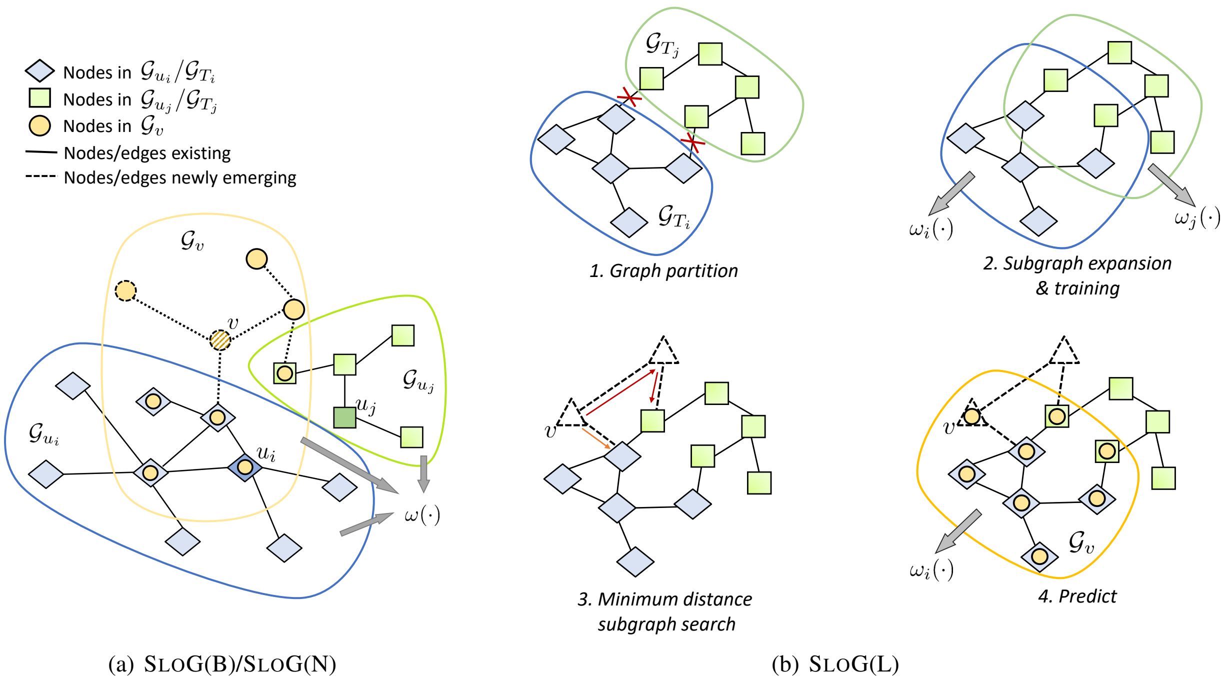


Figure 2. An overview of the proposed SLOG. (a) In SLOG(B)/SLOG(N), nodes u_i and u_j are associated with K-hop subgraphs $\mathcal{G}_{u_i}$ and $\mathcal{G}_{u_j}$ respectively. The filter with real-valued order, $\omega(\cdot)$, is applied to these subgraphs during model training. New nodes, depicted with dashed outlines, are processed similarly: sampled subgraphs are generated and the established filter is utilized for prediction. For node v, its corresponding subgraph $\mathcal{G}_v$ is employed for prediction. (b) In SLOG(L), the graph is partitioned into M subgraphs; for instance, two such subgraphs are $\mathcal{G}_{T_i}$ and $\mathcal{G}_{T_j}$. Each subgraph is expanded to restore disrupted edges, and a combination of global and respective local filters ($\omega_i(\cdot)$ for $\mathcal{G}_{T_i}$, $\omega_j(\cdot)$ for $\mathcal{G}_{T_j}$) is applied during training. Newly added nodes are assigned to the nearest subgraph, exemplified by node v being matched with $\mathcal{G}_{T_i}$, and predictions are made using the corresponding filter $\omega_i(\cdot)$.

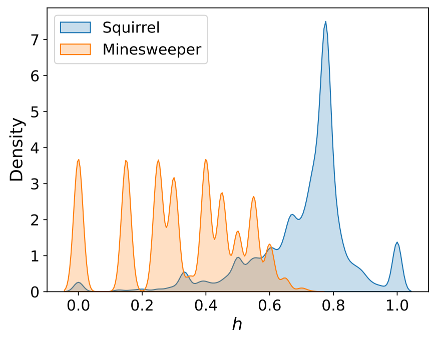


Figure 3. The one hop edge heterophily density of two real-world datasets: Squirrel (Rozemberczki et al., 2021) and Minesweeper (Platonov et al., 2023). The x-axis represents the edge heterophily (h), and the y-axis shows the corresponding density distribution. For all nodes in the graph, we sample 1-hop ego-graph for each node, and the density of the distribution can be defined as $\rho(h) = \Delta N(h)/\Delta h$, where $N(h)$ represents the proportion of the subgraphs that have an edge heterophily equal to h. The edge heterophily distribution of Squirrel is uniform, while that of Minesweeper is varying.

3.3. SLOG(L)

Actually, SLOG(B), SLOG(N), and various other heterophilic graph-oriented methods, such as GPRGNN (Chien et al., 2020), H$_2$GCN (Zhu et al., 2020), and BernNet (He et al., 2021) employ one single uniform filter to capture the frequency information of the whole graph. Nevertheless, heterophily in graphs is not uniformly distributed and can exhibit significant variation across different graph regions, which has been exemplified by Figure 3.

Therefore, one uniform filter with the same parameters (e.g., same p and q in Eq. (4)) for all subgraphs can not effectively capture the varying local heterophily. To address this issue, the proposed SLOG(L)'s filter contains two parts: one global uniform filter from SLOG(B)/SLOG(N) and one local adaptive filter for each subgraph, which is tailored to capture the local frequency signals. The filter for SLOG(L) is outlined in Eq. (7).

$$\omega_i(\mathbf{L}_{\mathcal{G}_v}) = (\mathbf{I} - \frac{1}{2}\mathbf{L}_{\mathcal{G}_v})^{p_i'}(\mathbf{I} + (\mathbf{L}_{\mathcal{G}_v} - \mathbf{I})^2)^{q_i'}, \tag{7}$$

where $p_i' = \beta p_{\text{glo}} + (1-\beta)p_i$ and $q_i' = \beta q_{\text{glo}} + (1-\beta)q_i$ represent the weighted combinations of global and local parameters. Here, β is a hyper-parameter that modulates the balance between global and local filters. The parameters p_{glo} and q_{glo} are associated with the global uniform filter, while p_i and q_i are adaptive to the i-th subgraph. By interpolating the parameters of the global filter with those of the local filter, SLOG(L) can address the varying edge heterophily distribution across different subgraphs.

As illustrated in Figure 2(b), SLOG(L) contains the following steps: (1) Graph partition: when the number of nodes in the graph is extremely large, it is infeasible for SLOG(L) to adopt the same sampling strategy as SLOG(B)/SLOG(N) due to the growing number of parameters (i.e., one (p_i, q_i) for one node v_i). Thus, we employ the METIS graph partition method (Karypis & Kumar, 1998) to partition the training graph into M subgraphs, $\{\mathcal{G}_{T_i}(\mathcal{V}_{T_i}, \mathcal{E}_{T_i})\}_{i=1}^{M}$. More details about METIS are presented in Appendix C.3; (2) Subgraph expansion and training: for each subgraph $\mathcal{G}_{T_i}$, the graph partition may disrupt some edge connections, thus we augment it with nodes and edges within K-hop distance from $\mathcal{G}_{T_i}$. This process can restore the disrupted edges in $\mathcal{G}_{T_i}$. After this step, we pursue the same procedure in SLOG(B)/SLOG(N) to apply the filter $\omega_i(\cdot)$ in Eq. (7) on the subgraph. The filter $\omega_i(\cdot)$ is a combination of global and corresponding local filter. We then train the model; (3) Minimum distance subgraph search & prediction: in the inductive setting, when a new node v emerges, SLOG(L) calculates its distance to each subgraph, and finds out the nearest subgraph. The subgraph with the minimum distance and its corresponding filter are selected to obtain the representation of v, which can be used for prediction. The detailed algorithm of SLOG(L) is attached in Appendix A.3 due to the page limit.

4. Experiment

In this section, we evaluate SLOG on the semi-supervised node classification task. We first introduce the datasets, baselines and settings in Section 4.1. Next, in Section 4.2, we experiment with SLOG(B) and SLOG(N). In Section 4.3, we conduct experiments on SLOG(L). In addition, We conduct additional experiments on SLOG's adaptive filtering ability in Section 4.4. Due to the page limit, some additional experimental results are attached in Appendix: (1) ablation studies (Appendix B.1); (2) results of additional baselines (Appendix B.2); (3) experimental results on synthetic datasets (Appendix B.3); (4) a convergence study (Appendix B.4); and (5) experimental results using two alternative optimization methods (Appendix B.5).[6]

[6]Our code is available at `https://github.com/Hsu1023/SLOG`.

4.1. Experiment Setup

Datasets. We adopt 16 datasets for evaluation, including 13 small-scale datasets and 3 large-scale datasets. The small-scale datasets, sourced from (Kipf & Welling, 2016; Bojchevski & Günnemann, 2017; Shchur et al., 2018; Rozemberczki et al., 2021; Platonov et al., 2023), include two categories: the heterophilic datasets include Chameleon, Squirrel, Squirrel-filtered, Chameleon-filtered, Minesweeper, Tolokers, Amazon-ratings, and Questions; the homophilic datasets include Cora, Citeseer, DBLP, Coauthor-CS, and Coauthor-Physics. The large-scale datasets, sourced from (Hamilton et al., 2017; Zeng et al., 2019; Hu et al., 2020), are Flickr, Ogbn-arxiv, Reddit. These datasets are diverse, varying in scale, domain, and heterophilic/homophilic ratios. Detailed statistics of datasets are presented in Appendix C.1.

Baselines. We compare our method against 13 baselines, including (1) a non-topology method: MLP; (2) general GNN methods including GCN (Kipf & Welling, 2016), ChebNet (Defferrard et al., 2016), GraphSAGE (Hamilton et al., 2017), GAT (Veličković et al., 2018), APPNP (Gasteiger et al., 2018), SGC (Wu et al., 2019), GATv2 (Brody et al., 2021); (3) heterophilic graph oriented methods including GPRGNN (Chien et al., 2020), H_2GCN (Zhu et al., 2020), FAGCN (Bo et al., 2021), BernNet (He et al., 2021), JacobiConv (Wang & Zhang, 2022).

Settings. For small-scale datasets, we employ a random split of 60%/20%/20% for train/validation/test sets and conduct experiments in the inductive setting[7]. For large-scale datasets, we keep the same split and the same transductive/inductive setting as those used in the original papers. It is important to note that in the inductive setting, the models are not exposed to validation or test nodes during training. For evaluation, we use accuracy (ACC) with standard deviation (std) as the metric, averaging the results over 5 runs.

4.2. SLOG(B) & SLOG(N)

The performance comparison on small-scale datasets is detailed in Table 1 and Table 2. Our method demonstrates a notable superiority over all baselines in most datasets, achieving the best performance in 11 out of 13 small-scale datasets. Specifically, for *heterophilic* datasets (Table 1), our method surpasses every baseline across all datasets. This superiority is attributed to the method's capability to effectively discern the graph's heterophily, thereby flexibly adjusting its filter to capture a diverse range of frequency signals depending on the dataset. In contrast, for *homophilic* datasets (Table 2), our method shows excellence in 3 out of 5 datasets and ranks second in the remaining two. Notably,

[7]For spectral methods, we build a new Laplacian matrix $\mathbf{L}_{\text{sym}}$ when new nodes emerge in the inductive setting and inherit the old parameters from training.

Table 1. Evaluation results on heterophilic datasets in the inductive setting.

Datasets	Squirrel	Chameleon	Squirrel-filt.	Chameleon-filt.	Minesweeper	Tolokers	Amazon-ratings	Questions
MLP	0.336±0.014	0.469±0.004	0.366±0.021	0.380±0.021	0.788±0.000	0.775±0.000	0.449±0.005	**0.972±0.000**
GCN	0.374±0.007	0.532±0.012	0.329±0.020	0.411±0.031	0.788±0.000	0.784±0.001	0.420±0.002	0.970±0.000
ChebNet	0.350±0.004	0.535±0.005	0.333±0.019	0.372±0.025	0.823±0.001	0.783±0.003	0.393±0.001	0.969±0.001
GraphSAGE	0.387±0.011	0.246±0.043	0.349±0.013	0.360±0.041	0.810±0.002	0.794±0.003	0.436±0.005	0.970±0.000
GAT	0.306±0.006	0.484±0.020	0.329±0.017	0.344±0.024	0.787±0.001	0.776±0.000	0.392±0.001	0.970±0.000
APPNP	0.314±0.008	0.410±0.010	0.312±0.019	0.381±0.020	0.788±0.000	0.778±0.001	0.429±0.002	0.970±0.000
SGC	0.371±0.005	0.486±0.002	0.320±0.016	0.357±0.021	0.786±0.000	0.782±0.000	0.398±0.002	0.970±0.000
GATv2	0.310±0.006	0.468±0.009	0.350±0.013	0.394±0.026	0.788±0.002	0.775±0.001	0.394±0.002	0.970±0.000
GPRGNN	0.343±0.009	0.472±0.020	0.364±0.019	0.394±0.038	0.791±0.000	0.775±0.001	0.414±0.004	0.970±0.000
H_2GCN	0.359±0.005	0.454±0.007	0.335±0.025	0.381±0.026	0.824±0.001	0.788±0.001	0.442±0.002	0.971±0.000
FAGCN	0.332±0.008	0.412±0.026	0.350±0.030	0.369±0.027	0.789±0.001	0.784±0.002	0.433±0.009	0.970±0.000
BernNet	0.361±0.007	0.578±0.007	0.361±0.020	0.374±0.030	0.788±0.000	0.772±0.007	0.398±0.002	0.969±0.001
JacobiConv	0.221±0.017	0.309±0.015	0.295±0.012	0.348±0.035	0.788±0.000	0.704±0.100	0.355±0.010	0.877±0.176
SLOG(B)	**0.392±0.006**	**0.581±0.024**	**0.427±0.013**	0.420±0.023	0.822±0.009	0.796±0.005	0.451±0.007	**0.972±0.001**
SLOG(N)	0.355±0.010	0.520±0.022	0.376±0.025	**0.431±0.026**	**0.844±0.008**	**0.810±0.006**	**0.456±0.006**	**0.972±0.001**

Table 2. Evaluation results on homophilic datasets in the inductive setting.

Datasets	Cora	Citeseer	DBLP	Co.-CS	Co.-Phys.
MLP	0.695±0.017	0.680±0.016	0.769±0.004	0.925±0.002	0.962±0.000
GCN	0.863±0.005	0.746±0.010	0.847±0.008	0.905±0.001	0.958±0.001
ChebNet	0.804±0.004	0.740±0.009	0.840±0.000	0.640±0.001	0.958±0.000
GraphSAGE	0.835±0.005	0.724±0.010	0.840±0.003	0.907±0.002	**0.970±0.001**
GAT	0.852±0.010	0.739±0.008	0.848±0.005	0.938±0.001	0.958±0.001
APPNP	0.839±0.004	0.748±0.008	0.835±0.008	0.918±0.001	0.961±0.000
SGC	0.859±0.010	0.754±0.008	0.845±0.003	0.938±0.001	0.958±0.000
GATv2	0.863±0.008	0.741±0.012	0.845±0.006	0.905±0.001	0.959±0.001
GPRGNN	**0.874±0.010**	0.756±0.003	0.848±0.004	0.942±0.001	0.966±0.000
H_2GCN	0.815±0.004	0.757±0.010	0.840±0.001	0.940±0.001	0.966±0.000
FAGCN	0.845±0.007	0.751±0.014	0.835±0.007	0.932±0.013	0.963±0.003
BernNet	0.865±0.006	0.745±0.015	0.849±0.004	0.938±0.001	0.959±0.000
JacobiConv	0.584±0.034	0.559±0.098	0.455±0.041	0.882±0.008	0.924±0.010
SLOG(B)	0.865±0.011	**0.766±0.026**	**0.850±0.005**	0.934±0.003	0.959±0.002
SLOG(N)	0.761±0.010	0.675±0.026	0.839±0.003	**0.944±0.005**	0.966±0.001

Table 3. Evaluation results on large-scale datasets.

Datasets	Flickr	Ogbn-arxiv	Reddit
nodes	89,250	169,343	232,965
edges	899,756	1,166,243	114,615,892
setting	inductive	transductive	inductive
MLP	0.474±0.001	0.539±0.001	0.702±0.001
GraphSAGE	0.502±0.002	0.717±0.002	0.944±0.001
GAT	0.509±0.001	0.676±0.003	0.944±0.002
GATv2	0.517±0.001	0.675±0.001	0.957±0.000
GPRGNN	0.508±0.002	0.684±0.002	0.950±0.000
H_2GCN	0.516±0.002	0.677±0.000	OOM
SLOG(B)	0.509±0.001	**0.723±0.001**	0.954±0.000
SLOG(N)	**0.520±0.003**	0.719±0.002	**0.962±0.001**

in the Cora and Coauthor-Physics datasets, our method's performance is marginally lower than the best baseline, by only 1.4% and 0.4%, respectively. This underscores our method's effectiveness in homophilic datasets as well.

For large-scale dataset evaluation, results in Table 3 illustrate that our method consistently outperforms others. A notable aspect is the size of the Reddit dataset, which contains 115M edges and is significantly larger than those in most related studies. Due to the high computational cost, some baselines meet the out-of-memory (OOM) problem in our machine[8]. However, thanks to the sampling technique, our method is able to directly run on the graph, achieving the best performance, confirming its scalability to large-scale graphs. It is also important to note that these experiments adhere to the same transductive/inductive settings as used in the original papers, further evidencing our method's robustness across various settings.

In addition, it is observed that SLOG(N) outperforms SLOG(B) on numerous datasets, despite its more complex architecture and increased number of parameters. This improvement is attributed to the additional non-linearity in SLOG(N), which enhances the model's expressiveness. Moreover, the incorporation of residual connections in SLOG(N) helps preserve the node embedding from previous layer.

4.3. SLOG(L)

The SLOG(L) model, as introduced in Section 3.3, addresses varying distributions of homophily/heterophily ratios across a graph. In order to quantify the heterophily/homophily balance, we introduce a metric, locality. We sample a fixed number of nodes in the graph, obtain their 1-hop ego-graphs, and calculate the edge heterophily of these ego-graphs. The locality, defined as the standard deviation of all 1-hop ego-graphs' edge heterophily, inversely indicates the balance level of local edge heterophily across the whole graph.

As shown in Table 4, SLOG(LB)/SLOG(LN) enhances performance in datasets with high *locality* (e.g., Chameleon, Chameleon-filt., Minesweeper, Tolokers), suggesting its effectiveness in contexts with imbalanced ho-

[8]Since some spectral GNNs can not be run on such large-scale datasets, we do not include them in the comparison.

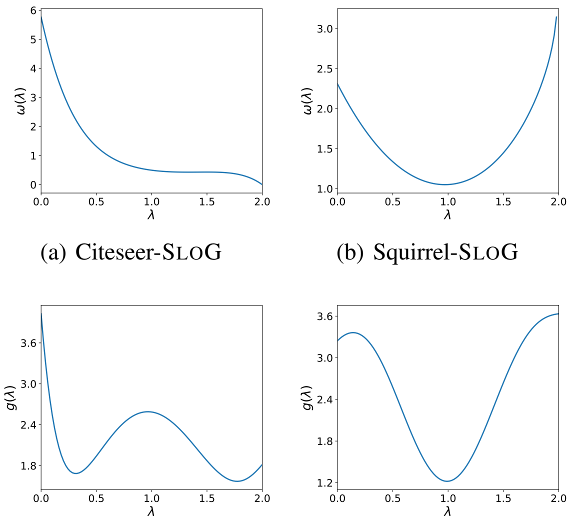
Table 4. Performance of SLOG(L) on datasets with different *locality*.

Datasets	Squirrel	Squirrel-filt.	Chameleon	Chameleon-filt.	Minesweeper	Tolokers
Locality/10^{-2}	1.85	1.66	4.16	3.03	4.08	3.84
Best of baselines	0.387±0.011	0.366±0.021	0.578±0.007	0.411±0.031	0.824±0.001	0.794±0.003
SLOG(B)	**0.392±0.006**	**0.427±0.013**	0.581±0.024	0.420±0.023	0.822±0.009	0.796±0.005
SLOG(N)	0.355±0.010	0.376±0.025	0.520±0.022	0.431±0.026	0.844±0.008	0.810±0.006
SLOG(LB)	0.387±0.008	0.409±0.010	**0.605±0.043**	0.443±0.029	0.807±0.006	0.785±0.008
SLOG(LN)	0.355±0.012	0.375±0.028	0.535±0.017	**0.453±0.041**	**0.848±0.007**	**0.814±0.004**

mophily/heterophily distributions. However, in datasets with low *locality* (e.g., Squirrel, Squirrel-filt.), the performance gain is not observed, likely due to the already balanced local edge heterophily of these graphs. Here, the addition of a local component unnecessarily complicates the model, potentially hindering the effectiveness of the model.

4.4. Adaptive Filtering to Broad Frequency Signals

In this subsection, we present the learned filters of our method on real datasets. The learned filters of our method SLOG(B) on real datasets are shown in Figure 4(a) and 4(b). To compare with BernNet, we also show the learned filters of BernNet on the same datasets in Figure 4(c) and Figure 4(d). For the homophilic Citeseer graph, SLOG(B) functions as a low-pass filter, capturing homophilic information (Figure 4(a)). Conversely, for the heterophilic Squirrel graph, it selectively filters out medium-frequency signals and preserves high-frequency ones (Figure 4(b)). However, BernNet's performance on Citeseer includes not only low-frequency signals but also some medium-frequency noise (Figure 4(c)), indicating a potential for overfitting due to its complex coefficients. Though BernNet learns filters similar to SLOG(B) on Squirrel, the fluctuating signal curves imply the capture of some extraneous signals (Figure 4(d)).



(a) Citeseer-SLOG (b) Squirrel-SLOG

(c) Citeseer-BernNet (d) Squirrel-BernNet

Figure 4. Learned filters on real dataset.

5. Related Work

Graph neural networks (GNN). GNN models can be roughly divided into two categories, i.e. spectral-based methods and spatial-based methods (Zhang et al., 2020). Spectral methods are based on the spectral graph theory, aiming to establish graph convolutional kernel in the spectral domain. The notable attempt is reported in (Bruna et al., 2013), which firstly introduces graph convolutional kernel. After that, ChebNet (Defferrard et al., 2016) utilizes Chebychev polynomials to form a convolutional kernel. GCN (Kipf & Welling, 2016) takes the first-order approximation to simplify the kernel. SGC (Wu et al., 2019) further changes multi-layer design to one linear transformation. Other spectral-based methods include (Levie et al., 2018; Li et al., 2018; Zhu et al., 2021b; Bianchi et al., 2021) and so on. Spatial-based methods mainly focus on aggregating the information of neighboring nodes. GraphSAGE (Hamilton et al., 2017) studies three different aggregators to aggregate the information of neighbors. In GAT (Veličković et al., 2018), attention mechanism is introduced as the aggregator. GIN (Xu et al., 2018) deploys MLPs to model injective functions in order to enhance the discriminative power of the GNN. We refer readers to (Zhou et al., 2020; Wu et al., 2020) for more details.

Heterophilic graph learning. While GNNs are mostly based on the homophily assumption that neighboring nodes are inclined to share the same labels, there are many real-world graphs that do not satisfy this assumption (McPherson et al., 2001). These graphs, which are called heterophilic graphs, have gained an increasing attention recently. CayleyNet (Levie et al., 2018) defines a complex Cayley filter and utilize Jacobi iteration to optimize it, while ARMA (Bianchi et al., 2021) uses auto-regressive moving average (ARMA) filter to capture the global graph structure. Geom-GCN (Pei et al., 2020) defines the geometric relationship in a latent space to use neighborhood information. FAGCN (Bo et al., 2021) utilizes the attention mechanism to seperately learn low-frequency and high-frequency signals. ACM-

GCN (Luan et al., 2021) adopts a linear combination of low/high pass filters and adaptively mix the generated node information from the two filters. CPGNN (Zhu et al., 2021a) utilizes a compatibility matrix to model the heterophilic/ homophilic relationships between nodes. TeDGCN (Yan et al., 2023) utilizes a filter with real-valued order, with its learnable parameter as the depth of graph convolutional layers, expressed as a real number. Other heterophilic graph learning methods include (Li et al., 2022b; Wang & Zhang, 2022; He et al., 2022; Zheng et al., 2023; Xu et al., 2023; Guo & Wei, 2023; Geng et al., 2023; Guo et al., 2023). We refer readers to (Zheng et al., 2022) for more details.

6. Conclusion and Limitations

In this paper, we propose an inductive spectral graph neural network named SLOG with the expressive power beyond a polynomial filter. Specifically, SLOG includes three sub-models: the base model SLOG(B), the non-linear model SLOG(N) and the local model SLOG(L). SLOG(B) is equipped with a filter with real-valued order, which enjoys geometric interpretability, mathematical feasibility and adaptive filtering. SLOG(N) adds non-linearity and residual connections into SLOG(B). To better capture the varying heterophily distribution, SLOG(L) conducts an interpolation between the global uniform filter and the local adaptive filter. Extensive experiments on 16 real-world datasets corroborate the effectiveness, scalability and robustness of SLOG in the inductive semi-supervised homophilic/heterophilic node classification task. One potential limitation of SLOG is that it only focuses on node classification problem, and we leave its extension to other tasks including link prediction as future work.

Impact Statement

This paper presents work whose goal is to advance the field of Graph Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

Acknowledgements

This work is supported by NSF (IIS-19-56151, 2134079, 2316233, and 2324770), DARPA (HR001121C0165), AFOSR (FA9550-24-1-0002). The content of the information in this document does not necessarily reflect the position or the policy of the Government, and no official endorsement should be inferred. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on.

References

Abu-El-Haija, S., Perozzi, B., Kapoor, A., Alipourfard, N., Lerman, K., Harutyunyan, H., Ver Steeg, G., and Galstyan, A. Mixhop: Higher-order graph convolutional architectures via sparsified neighborhood mixing. In international conference on machine learning, pp. 21–29. PMLR, 2019.

Balcilar, M., Guillaume, R., Héroux, P., Gaüzère, B., Adam, S., and Honeine, P. Analyzing the expressive power of graph neural networks in a spectral perspective. In Proceedings of the International Conference on Learning Representations (ICLR), 2021.

Ban, Y., Yan, Y., Banerjee, A., and He, J. Ee-net: Exploitation-exploration neural networks in contextual bandits. arXiv preprint arXiv:2110.03177, 2021.

Ban, Y., Yan, Y., Banerjee, A., and He, J. Neural exploitation and exploration of contextual bandits. arXiv preprint arXiv:2305.03784, 2023.

Bianchi, F. M., Grattarola, D., Livi, L., and Alippi, C. Graph neural networks with convolutional arma filters. IEEE transactions on pattern analysis and machine intelligence, 44(7):3496–3507, 2021.

Bo, D., Wang, X., Shi, C., and Shen, H. Beyond low-frequency information in graph convolutional networks. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 35, pp. 3950–3957, 2021.

Bojchevski, A. and Günnemann, S. Deep gaussian embedding of graphs: Unsupervised inductive learning via ranking. arXiv preprint arXiv:1707.03815, 2017.

Brody, S., Alon, U., and Yahav, E. How attentive are graph attention networks? arXiv preprint arXiv:2105.14491, 2021.

Bruna, J., Zaremba, W., Szlam, A., and LeCun, Y. Spectral networks and locally connected networks on graphs. arXiv preprint arXiv:1312.6203, 2013.

Chien, E., Peng, J., Li, P., and Milenkovic, O. Adaptive universal generalized pagerank graph neural network. In International Conference on Learning Representations, 2020.

Chung, F. R. Spectral graph theory, volume 92. American Mathematical Soc., 1997.

Defferrard, M., Bresson, X., and Vandergheynst, P. Convolutional neural networks on graphs with fast localized spectral filtering. Advances in neural information processing systems, 29, 2016.

Du, B., Zhang, S., Yan, Y., and Tong, H. New frontiers of multi-network mining: Recent developments and future trend. In Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining, pp. 4038–4039, 2021.

Fey, M. and Lenssen, J. E. Fast graph representation learning with pytorch geometric. arXiv preprint arXiv:1903.02428, 2019.

Frazier, P. I. A tutorial on bayesian optimization. arXiv preprint arXiv:1807.02811, 2018.

Fu, D., Zhou, D., and He, J. Local motif clustering on time-evolving graphs. In KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, CA, USA, August 23-27, 2020, pp. 390–400. ACM, 2020. doi: 10.1145/3394486.3403081. URL https://doi.org/10.1145/3394486.3403081.

Fu, D., Zhou, D., Maciejewski, R., Croitoru, A., Boyd, M., and He, J. Fairness-aware clique-preserving spectral clustering of temporal graphs. In Proceedings of the ACM Web Conference 2023, WWW 2023, Austin, TX, USA, 30 April 2023 - 4 May 2023, pp. 3755–3765. ACM, 2023. doi: 10.1145/3543507.3583423. URL https://doi.org/10.1145/3543507.3583423.

Fu, D., Hua, Z., Xie, Y., Fang, J., Zhang, S., Sancak, K., Wu, H., Malevich, A., He, J., and Long, B. Vcr-graphormer: A mini-batch graph transformer via virtual connections. CoRR, abs/2403.16030, 2024. doi: 10.48550/ARXIV.2403.16030. URL https://doi.org/10.48550/arXiv.2403.16030.

Gasteiger, J., Bojchevski, A., and Günnemann, S. Predict then propagate: Graph neural networks meet personalized pagerank. arXiv preprint arXiv:1810.05997, 2018.

Geng, H., Chen, C., He, Y., Zeng, G., Han, Z., Chai, H., and Yan, J. Pyramid graph neural network: A graph sampling and filtering approach for multi-scale disentangled representations. In Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, pp. 518–530, 2023.

Ghosh, A. and Kannan, R. Alternating minimization converges super-linearly for mixed linear regression. In International Conference on Artificial Intelligence and Statistics, pp. 1093–1103. PMLR, 2020.

Guo, K., Cao, X., Liu, Z., and Chang, Y. Taming over-smoothing representation on heterophilic graphs. Information Sciences, 647:119463, 2023.

Guo, Y. and Wei, Z. Graph neural networks with learnable and optimal polynomial bases. In International Conference on Machine Learning, pp. 12077–12097. PMLR, 2023.

Hamilton, W., Ying, Z., and Leskovec, J. Inductive representation learning on large graphs. Advances in neural information processing systems, 30, 2017.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 770–778, 2016.

He, M., Wei, Z., Xu, H., et al. Bernnet: Learning arbitrary graph spectral filters via bernstein approximation. Advances in Neural Information Processing Systems, 34:14239–14251, 2021.

He, M., Wei, Z., and Wen, J.-R. Convolutional neural networks on graphs with chebyshev approximation, revisited. Advances in neural information processing systems, 35:7264–7276, 2022.

Hu, W., Fey, M., Zitnik, M., Dong, Y., Ren, H., Liu, B., Catasta, M., and Leskovec, J. Open graph benchmark: Datasets for machine learning on graphs. Advances in neural information processing systems, 33:22118–22133, 2020.

Jing, B., Yan, Y., Zhu, Y., and Tong, H. Coin: Co-cluster infomax for bipartite graphs. In NeurIPS 2022 Workshop: New Frontiers in Graph Learning, 2022.

Jing, B., Yan, Y., Ding, K., Park, C., Zhu, Y., Liu, H., and Tong, H. Sterling: Synergistic representation learning on bipartite graphs. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 38, pp. 12976–12984, 2024.

Karypis, G. and Kumar, V. Parallel multilevel k-way partitioning scheme for irregular graphs. In Proceedings of the 1996 ACM/IEEE Conference on Supercomputing, pp. 35–es, 1996.

Karypis, G. and Kumar, V. A fast and high quality multilevel scheme for partitioning irregular graphs. SIAM Journal on scientific Computing, 20(1):359–392, 1998.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014.

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. In International Conference on Learning Representations, 2016.

Levie, R., Monti, F., Bresson, X., and Bronstein, M. M. Cayleynets: Graph convolutional neural networks with complex rational spectral filters. IEEE Transactions on Signal Processing, 67(1):97–109, 2018.

Li, J., Shao, H., Sun, D., Wang, R., Yan, Y., Li, J., Liu, S., Tong, H., and Abdelzaher, T. Unsupervised belief representation learning with information-theoretic variational graph auto-encoders. In Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 1728–1738, 2022a.

Li, Q., Zhu, Z., and Tang, G. Alternating minimizations converge to second-order optimal solutions. In International Conference on Machine Learning, pp. 3935–3943. PMLR, 2019.

Li, R., Wang, S., Zhu, F., and Huang, J. Adaptive graph convolutional neural networks. In Proceedings of the AAAI conference on artificial intelligence, volume 32, 2018.

Li, X., Zhu, R., Cheng, Y., Shan, C., Luo, S., Li, D., and Qian, W. Finding global homophily in graph neural networks when meeting heterophily. In International Conference on Machine Learning, pp. 13242–13256. PMLR, 2022b.

Lim, D., Hohne, F., Li, X., Huang, S. L., Gupta, V., Bhalerao, O., and Lim, S. N. Large scale learning on non-homophilous graphs: New benchmarks and strong simple methods. Advances in Neural Information Processing Systems, 34:20887–20902, 2021.

Lin, H., Bai, H., Liu, Z., Hou, L., Sun, M., Song, L., Wei, Y., and Sun, Z. Mope-clip: Structured pruning for efficient vision-language models with module-wise pruning error metric. arXiv preprint arXiv:2403.07839, 2024.

Liu, L., Du, B., Fung, Y. R., Ji, H., Xu, J., and Tong, H. Kompare: A knowledge graph comparative reasoning system. In Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining, pp. 3308–3318, 2021.

Liu, L., Du, B., Xu, J., Xia, Y., and Tong, H. Joint knowledge graph completion and question answering. In Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, pp. 1098–1108, 2022.

Liu, Z., Cao, W., Gao, Z., Bian, J., Chen, H., Chang, Y., and Liu, T.-Y. Self-paced ensemble for highly imbalanced massive data classification. In 2020 IEEE 36th international conference on data engineering (ICDE), pp. 841–852. IEEE, 2020a.

Liu, Z., Wei, P., Jiang, J., Cao, W., Bian, J., and Chang, Y. Mesa: boost ensemble imbalanced learning with meta-sampler. Advances in neural information processing systems, 33:14463–14474, 2020b.

Liu, Z., Zeng, Z., Qiu, R., Yoo, H., Zhou, D., Xu, Z., Zhu, Y., Weldemariam, K., He, J., and Tong, H. Topological augmentation for class-imbalanced node classification. arXiv preprint arXiv:2308.14181, 2023.

Loshchilov, I. and Hutter, F. Decoupled weight decay regularization. arXiv preprint arXiv:1711.05101, 2017.

Luan, S., Hua, C., Lu, Q., Zhu, J., Zhao, M., Zhang, S., Chang, X.-W., and Precup, D. Is heterophily a real nightmare for graph neural networks to do node classification? arXiv preprint arXiv:2109.05641, 2021.

McPherson, M., Smith-Lovin, L., and Cook, J. M. Birds of a feather: Homophily in social networks. Annual review of sociology, 27(1):415–444, 2001.

Pei, H., Wei, B., Chang, K. C.-C., Lei, Y., and Yang, B. Geom-gcn: Geometric graph convolutional networks. arXiv preprint arXiv:2002.05287, 2020.

Platonov, O., Kuznedelev, D., Diskin, M., Babenko, A., and Prokhorenkova, L. A critical look at the evaluation of gnns under heterophily: are we really making progress? arXiv preprint arXiv:2302.11640, 2023.

Rasmussen, C. E., Williams, C. K., et al. Gaussian processes for machine learning, volume 1. Springer, 2006.

Roach, S., Ni, C., Kopylov, A., Lu, T.-C., Xu, J., Zhang, S., Du, B., Zhou, D., Wu, J., Liu, L., et al. Canon: Complex analytics of network of networks for modeling adversarial activities. In 2020 IEEE International Conference on Big Data (Big Data), pp. 1634–1643. IEEE, 2020.

Rozemberczki, B., Allen, C., and Sarkar, R. Multi-scale attributed node embedding. Journal of Complex Networks, 9(2):cnab014, 2021.

Shchur, O., Mumme, M., Bojchevski, A., and Günnemann, S. Pitfalls of graph neural network evaluation. arXiv preprint arXiv:1811.05868, 2018.

Shuman, D. I., Narang, S. K., Frossard, P., Ortega, A., and Vandergheynst, P. The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. IEEE signal processing magazine, 30(3):83–98, 2013.

Vashishth, S., Sanyal, S., Nitin, V., and Talukdar, P. Composition-based multi-relational graph convolutional networks. arXiv preprint arXiv:1911.03082, 2019.

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

Wang, D., Yan, Y., Qiu, R., Zhu, Y., Guan, K., Margenot, A., and Tong, H. Networked time series imputation via position-aware graph enhanced variational autoencoders. In Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, pp. 2256–2268, 2023a.

Wang, R., Yan, Y., Wang, J., Jia, Y., Zhang, Y., Zhang, W., and Wang, X. Acekg: A large-scale knowledge graph for academic data mining. In Proceedings of the 27th ACM international conference on information and knowledge management, pp. 1487–1490, 2018.

Wang, R., Li, Z., Sun, D., Liu, S., Li, J., Yin, B., and Abdelzaher, T. Learning to sample and aggregate: Few-shot reasoning over temporal knowledge graphs. Advances in Neural Information Processing Systems, 35:16863–16876, 2022.

Wang, R., Li, B., Lu, Y., Sun, D., Li, J., Yan, Y., Liu, S., Tong, H., and Abdelzaher, T. F. Noisy positive-unlabeled learning with self-training for speculative knowledge graph reasoning. arXiv preprint arXiv:2306.07512, 2023b.

Wang, X. and Zhang, M. How powerful are spectral graph neural networks. In International Conference on Machine Learning, pp. 23341–23362. PMLR, 2022.

Wei, T., Guo, Z., Chen, Y., and He, J. Ntk-approximating mlp fusion for efficient language model fine-tuning. In International Conference on Machine Learning, pp. 36821–36838. PMLR, 2023.

Wei, T., Jin, B., Li, R., Zeng, H., Wang, Z., Sun, J., Yin, Q., Lu, H., Wang, S., He, J., et al. Towards unified multimodal personalization: Large vision-language models for generative recommendation and beyond. arXiv preprint arXiv:2403.10667, 2024.

Wu, F., Souza, A., Zhang, T., Fifty, C., Yu, T., and Weinberger, K. Simplifying graph convolutional networks. In International conference on machine learning, pp. 6861–6871. PMLR, 2019.

Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., and Philip, S. Y. A comprehensive survey on graph neural networks. IEEE transactions on neural networks and learning systems, 32(1):4–24, 2020.

Xu, K., Hu, W., Leskovec, J., and Jegelka, S. How powerful are graph neural networks? In International Conference on Learning Representations, 2018.

Xu, Z., Chen, Y., Zhou, Q., Wu, Y., Pan, M., Yang, H., and Tong, H. Node classification beyond homophily: Towards a general solution. In Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, pp. 2862–2873, 2023.

Yan, Y., Liu, L., Ban, Y., Jing, B., and Tong, H. Dynamic knowledge graph alignment. In Proceedings of the AAAI conference on artificial intelligence, volume 35, pp. 4564–4572, 2021a.

Yan, Y., Zhang, S., and Tong, H. Bright: A bridging algorithm for network alignment. In Proceedings of the web conference 2021, pp. 3907–3917, 2021b.

Yan, Y., Chen, Y., Das, M., Yang, H., and Tong, H. Red-gcn: Revisit the depth of graph convolutional network. 2022a.

Yan, Y., Zhou, Q., Li, J., Abdelzaher, T., and Tong, H. Dissecting cross-layer dependency inference on multi-layered inter-dependent networks. In Proceedings of the 31st ACM International Conference on Information & Knowledge Management, pp. 2341–2351, 2022b.

Yan, Y., Chen, Y., Chen, H., Xu, M., Das, M., Yang, H., and Tong, H. From trainable negative depth to edge heterophily in graphs. In Thirty-seventh Conference on Neural Information Processing Systems, 2023.

Yan, Y., Hu, Y., Zhou, Q., Liu, L., Zeng, Z., Chen, Y., Pan, M., Chen, H., Das, M., and Tong, H. Pacer: Network embedding from positional to structural. In Proceedings of the ACM on Web Conference 2024, pp. 2485–2496, 2024a.

Yan, Y., Jing, B., Liu, L., Wang, R., Li, J., Abdelzaher, T., and Tong, H. Reconciling competing sampling strategies of network embedding. Advances in Neural Information Processing Systems, 36, 2024b.

Zeng, H., Zhou, H., Srivastava, A., Kannan, R., and Prasanna, V. Graphsaint: Graph sampling based inductive learning method. arXiv preprint arXiv:1907.04931, 2019.

Zeng, Z., Zhang, S., Xia, Y., and Tong, H. Parrot: Position-aware regularized optimal transport for network alignment. In Proceedings of the ACM Web Conference 2023, pp. 372–382, 2023a.

Zeng, Z., Zhu, R., Xia, Y., Zeng, H., and Tong, H. Generative graph dictionary learning. In International Conference on Machine Learning, pp. 40749–40769. PMLR, 2023b.

Zeng, Z., Du, B., Zhang, S., Xia, Y., Liu, Z., and Tong, H. Hierarchical multi-marginal optimal transport for network alignment. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 38, pp. 16660–16668, 2024.

Zhang, M. and Chen, Y. Link prediction based on graph neural networks. Advances in neural information processing systems, 31, 2018.

Zhang, Z., Cui, P., and Zhu, W. Deep learning on graphs: A survey. IEEE Transactions on Knowledge and Data Engineering, 34(1):249–270, 2020.

Zheng, X., Liu, Y., Pan, S., Zhang, M., Jin, D., and Yu, P. S. Graph neural networks for graphs with heterophily: A survey. arXiv preprint arXiv:2202.07082, 2022.

Zheng, Y., Zhang, H., Lee, V., Zheng, Y., Wang, X., and Pan, S. Finding the missing-half: Graph complementary learning for homophily-prone and heterophily-prone graphs. arXiv preprint arXiv:2306.07608, 2023.

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

Zhu, J., Yan, Y., Zhao, L., Heimann, M., Akoglu, L., and Koutra, D. Beyond homophily in graph neural networks: Current limitations and effective designs. Advances in neural information processing systems, 33:7793–7804, 2020.

Zhu, J., Rossi, R. A., Rao, A., Mai, T., Lipka, N., Ahmed, N. K., and Koutra, D. Graph neural networks with heterophily. In Proceedings of the AAAI conference on artificial intelligence, volume 35, pp. 11168–11176, 2021a.

Zhu, M., Wang, X., Shi, C., Ji, H., and Cui, P. Interpreting and unifying graph neural networks with an optimization framework. In Proceedings of the Web Conference 2021, pp. 1215–1226, 2021b.

Appendices

The contents of the appendices are organized as follows:

A. Details of Proposed Model

A.1. Proof of Proposition 3.1

Proposition 3.1. The SLOG's filter with real-valued order, $\omega(\cdot)$, in the spectral domain can be regarded as the combination of two linear graph convolutional networks in the spatial domain: $\omega(\mathbf{L}_{\text{sym}}) = \mathbf{S}_1^p \cdot \mathbf{S}_2^q$, where $\mathbf{S}_1 = \frac{1}{2}(\mathbf{I} + \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}})$ and $\mathbf{S}_2 = \mathbf{I} + (\mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}})^2$.

Proof. The filter with real-valued order, $\omega(\cdot)$, is defined as follows:

$$\omega(\mathbf{L}_{\text{sym}}) = (\mathbf{I} - \frac{1}{2}\mathbf{L}_{\text{sym}})^p(\mathbf{I} + (\mathbf{L}_{\text{sym}} - \mathbf{I})^2)^q, \tag{8}$$

We define $\mathbf{S}_1 = \frac{1}{2}(\mathbf{I} + \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}})$, $\mathbf{S}_2 = \mathbf{I} + (\mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}})^2$. Since $\mathbf{L}_{\text{sym}} = \mathbf{I} - \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$, we have:

$$\begin{aligned}
\mathbf{S}_1 &= \mathbf{I} - \frac{1}{2}\mathbf{L}_{\text{sym}} = \mathbf{U}(\mathbf{I} - \frac{1}{2}\mathbf{\Lambda})\mathbf{U}^\top \\
&= \mathbf{I} - \frac{1}{2}(\mathbf{I} - \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}) = \frac{1}{2}(\mathbf{I} + \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}), \\
\mathbf{S}_2 &= \mathbf{I} + (\mathbf{L}_{\text{sym}} - \mathbf{I})^2 = \mathbf{I} + (\mathbf{I} - \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}} - \mathbf{I})^2 \\
&= \mathbf{I} + (\mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}})^2.
\end{aligned}$$

Then, we have:

$$\omega(\mathbf{L}_{\text{sym}}) = \mathbf{S}_1^p \cdot \mathbf{S}_2^q. \tag{9}$$

Similar to d in TeDGCN (Yan et al., 2023), p and q can be explained as two real-valued depths of the graph convolution networks $\mathbf{S}_1$ and $\mathbf{S}_2$. □

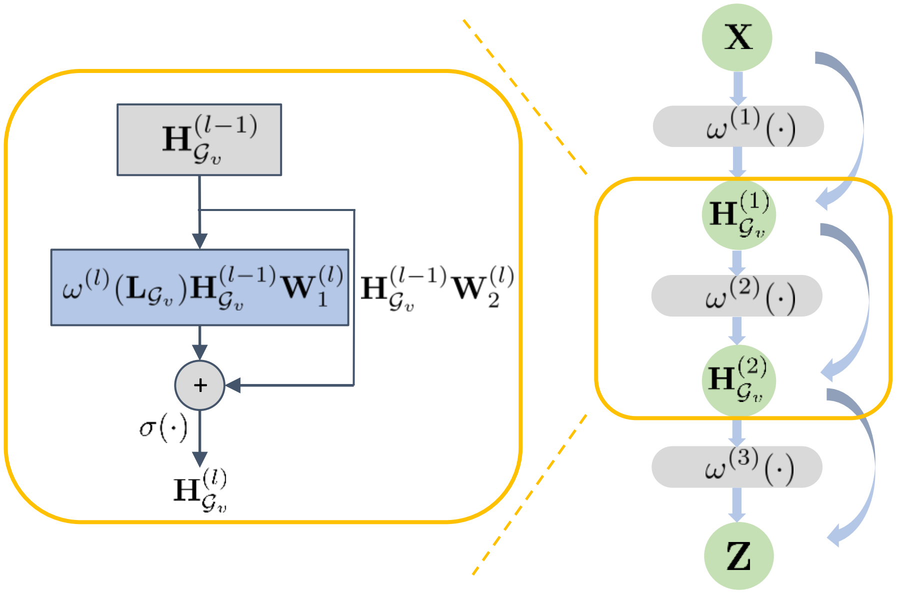

$\mathbf{H}_{\mathcal{G}_v}^{(l-1)}$
$\omega^{(l)}(\mathbf{L}_{\mathcal{G}_v})\mathbf{H}_{\mathcal{G}_v}^{(l-1)}\mathbf{W}_1^{(l)}$
$\mathbf{H}_{\mathcal{G}_v}^{(l-1)}\mathbf{W}_2^{(l)}$
+
$\sigma(\cdot)$
$\mathbf{H}_{\mathcal{G}_v}^{(l)}$
$\mathbf{X}$
$\omega^{(1)}(\cdot)$
$\mathbf{H}_{\mathcal{G}_v}^{(1)}$
$\omega^{(2)}(\cdot)$
$\mathbf{H}_{\mathcal{G}_v}^{(2)}$
$\omega^{(3)}(\cdot)$
$\mathbf{Z}$

Figure 5. Residual connection in SLOG(N).

A.2. Illustration of Residual Connections in SLOG(N)

In Section 3.2, we introduce the SLOG(N), which adds more non-linearity and residual connections. In each layer, the representation of nodes is updated by Eq. (6). Here, we present a figure to further illustrate the residual connection structure, as shown in Figure 5.

A.3. Pseudo Code of SLOG(L)

Algorithm 2 SLOG(LB) Algorithm

```
1: Input: Training Graph $\mathcal{G}_T(\mathcal{V}_T, \mathcal{E}_T)$; evaluating nodes $\mathcal{V}_E$, corresponding edges $\mathcal{E}_E$; input features $\mathbf{X}_T, \mathbf{X}_E$; hop number of subgraphs $K$; maximum neighbor numbers of each depth $\{N_i\}$
2: Output: Vector representations $\mathbf{z}_v$ for node $v \in \mathcal{V}_E$
   # Step 1: Graph partition
3: Utilize METIS to partition $\mathcal{G}_T$ into $M$ subgraphs $\{\mathcal{G}_{T_i}(\mathcal{V}_{T_i}, \mathcal{E}_{T_i})\}_{i=1}^{M}$.
4: Initialize parameters of global filter $(p_{\text{glo}}, q_{\text{glo}})$ and local filters $\{(p_i, q_i)\}_{i=1}^{M}$
   # Step 2: Subgraph expansion and training
5: for $i = 1$ to $M$ do
6:     $\mathcal{G}'_{T_i} \leftarrow SAMPLE(\mathcal{G}_{T_i}, \mathcal{V}_{T_i}, K, \{N_i\})$
7:     Obtain corresponding filter $\omega_i$ by Eq. (7)
8:     Obtain nodes representations $\mathbf{Z}_{\mathcal{G}'_{T_i}}$ of $\mathcal{G}'_{T_i}$ by Eq. (5)
9:     Normalize representations in $\mathcal{G}_{T_i}$ with Softmax function and optimize Cross-entropy loss
10: end for
   # Step 3: Minimum distance subgraph search & prediction
11: $\mathcal{G} \leftarrow (\mathcal{V}_T \cup \mathcal{V}_E, \mathcal{E}_T \cup \mathcal{E}_E)$
12: for $v \in \mathcal{V}_E$ do
13:     Calculate distance from $v$ to each $\mathcal{G}_i$ and get $\{dist_i\}$
14:     $\mathcal{G}_v \leftarrow \mathcal{G}_{T_i}$, s.t. $dist_i$ is the minimum
15:     $\mathcal{G}_v \leftarrow SAMPLE(\mathcal{G}, \{v\} \cup \mathcal{V}_{T_v}, K, \{N_i\})$
16:     Obtain corresponding filter $\omega_i$ by Eq. (7)
17:     Obtain nodes representations $\mathbf{Z}_{\mathcal{G}_v}$ of $\mathcal{G}_v$ by Eq. (5)
18:     $\mathbf{z}_v \leftarrow \mathbf{Z}_{\mathcal{G}_v}(v)$
19: end for
20: return $\{\mathbf{z}_v | v \in \mathcal{V}_E\}$
```

A.4. Complexity Analysis

In this subsection, we conduct a brief complexity analysis of SLOG(B) and SLOG(N). We base our analysis on a one-hop neighbor sampling strategy, setting the maximum number of sampled neighbors (k) empirically to no more than 25. This setup leads to a time complexity of $O(k)$ per batch. For SLOG(B), if we utilize a single node for one batch, we have N batches to train the model, where N is the number of nodes. The computational procedures for each batch can be categorized into two main steps: (1) eigenvalue decomposition and (2) message passing. Regarding (1), the eigenvalue decomposition of $\mathbf{L}_{\text{sym}}$ takes $O(k^3)$ time. In terms of (2), by Eq. (1), the computation of $\omega(\mathbf{L}_{\text{sym}})$ requires $O(k^3)$ time. Then, matrix multiplications cost $O(k^2f + kfc)$ time, where f is the feature dimension, and c is the class number. Consequently, the time complexity for a single batch in SLOG(B) is $O(k^3 + k^2f + kfc)$, and for the entire graph, it escalates to $O(N(k^3 + k^2f + kfc))$. For SLOG(N) with L layers, the time complexity is $O(NL(k^3 + k^2f + kfc))$.

B. Additional Experiment

B.1. Ablation studies

We conduct an ablation study on SLOG(B), which is detailed in Table 5, and assess the impact of each component. Results of the first two rows reveal that omitting any term from Eq. (4) reduces performance, emphasizing the importance of each element. Additionally, the sampling strategy plays a crucial role in enhancing performance. Without this strategy, the model would not only encounter a drop in performance but also become unsuitable for application to large-scale datasets, including Flickr, Ogbn-arxiv, and Reddit.

Table 5. Ablation Study for SLOG(B) Model.

Datasets	Cora	Chameleon	Minesweeper
SLOG(B)	**0.865±0.011**	**0.581±0.024**	**0.822±0.009**
w/o. $\mathbf{S}_1$	0.854±0.016	0.570±0.015	0.803±0.004
w/o. $\mathbf{S}_2$	0.838±0.013	0.576±0.030	0.818±0.008
w/o. sampling	0.856±0.017	0.568±0.013	0.802±0.008

To test the effectiveness of subgraph sampling strategy on other baseline methods, we equip ChebNet with subgraph sampling. The results are shown in Table 6. We can observe that the subgraph sampling strategy can indeed be beneficial to other baseline methods as well. However, the proposed SLOG still outperforms the baseline methods with subgraph sampling strategy like ChebNet.

Table 6. Evaluation results on different datasets in the inductive setting.

Dataset	Cora	Citeseer	Squirrel	Chameleon	Tolokers
ChebNet	0.804±0.004	0.740±0.009	0.350±0.004	0.535±0.005	0.783±0.003
w. sampling	0.829±0.013	0.754±0.010	0.356±0.013	0.467±0.000	0.784±0.003
SLOG(B)	0.865±0.011	0.766±0.026	0.392±0.006	0.581±0.024	0.796±0.005

B.2. Additional Baselines

In inductive setting, we compare SLOG(B)/SLOG(N) with CayleyNet (Levie et al., 2018), GIN (Xu et al., 2018), ARMA (Bianchi et al., 2021), ChebNetII (He et al., 2022), TeDGCN (Yan et al., 2023), PyGNN (Geng et al., 2023), FavardGNN (Guo & Wei, 2023), OptBasisGNN (Guo & Wei, 2023) with the same random split in Section 4. The results can be found in Table 7. It is observed that the latest baseline methods, such as ChebNetII, TeDGCN, can produce competitive results. However, SLOG still outperforms all the baselines among most of the datasets. It reveals the effectiveness of our proposed method.

B.3. Synthetic Datasets

To demonstrate the robustness of SLOG under different homophily/heterophily ratios, we generate synthetic datasets with varying heterophily ratios (h) from 0.1 to 0.9, following the method described in (Zhu et al., 2020). The higher the h value, the more heterophilic the graph is. Node features are sampled from the Ogbn-arxiv dataset (Hu et al., 2020).

Table 7. Evaluation results on different datasets in the inductive setting.

Datasets	Cora	Citeseer	Squirrel-filt.	Chameleon-filt.	Minesweeper	Tolokers	Amazon-ratings
CayleyNet	0.773±0.007	0.685±0.017	0.347±0.027	0.264±0.022	0.798±0.004	0.781±0.008	0.265±0.003
GIN	0.808±0.022	0.700±0.009	0.327±0.009	0.348±0.042	0.788±0.001	0.775±0.002	0.420±0.007
ARMA	0.757±0.026	0.735±0.009	0.364±0.010	0.343±0.034	0.794±0.012	0.779±0.007	0.428±0.004
ChebNetII	**0.868±0.011**	0.717±0.011	0.354±0.013	0.367±0.018	0.759±0.016	0.783±0.004	0.396±0.002
TeDGCN	0.825±0.005	0.751±0.003	$\underline{0.409\pm0.031}$	$\underline{0.422\pm0.022}$	0.793±0.001	$\underline{0.796\pm0.002}$	0.425±0.002
PyGNN	0.863±0.011	0.742±0.007	0.349±0.011	0.397±0.030	0.787±0.003	0.789±0.001	**0.456±0.005**
FavardGNN	0.850±0.007	0.749±0.018	0.383±0.015	0.406±0.019	0.805±0.004	0.786±0.002	0.397±0.004
OptBasisGNN	0.856±0.005	$\underline{0.753\pm0.006}$	0.392±0.007	0.400±0.043	0.789±0.001	0.775±0.000	0.368±0.000
SLOG(B)	$\underline{0.865\pm0.011}$	**0.766±0.026**	**0.427±0.013**	0.420±0.023	$\underline{0.822\pm0.009}$	$\underline{0.796\pm0.005}$	$\underline{0.451\pm0.007}$
SLOG(N)	0.761±0.010	0.675±0.026	0.376±0.025	**0.431±0.026**	**0.844±0.008**	**0.810±0.006**	**0.456±0.006**

To be specific, we initialize a small graph, and recursively add new nodes to the graph. The class assignment for the newly added nodes is determined randomly from a predefined set of class numbers, denoted as C. The probability of an edge between the new node and the existing nodes in the graph is determined by the classes that the nodes belong to, the heterophily ratio h, and the current degrees of existing nodes. Once the graph scale reaches the target scale, we assign each node a feature vector from a real dataset. It is ensured that nodes belonging to the same class in the generated graph have feature vectors originating from nodes in the same class in the real dataset. The statistics of generated datasets are shown in Table 8.

Table 8. The statistics of synthetic datasets.

Datasets	Nodes	Edges	Features	Classes	Heterophily
syn-arxiv	2,000	∼20,000	128	40	0.1 to 0.9

The experiments are conducted under inductive settings across 5 runs for each model on each dataset, and each dataset is randomly splited into 60%/20%/20% train/validation/test split. The results are shown in Table 9.

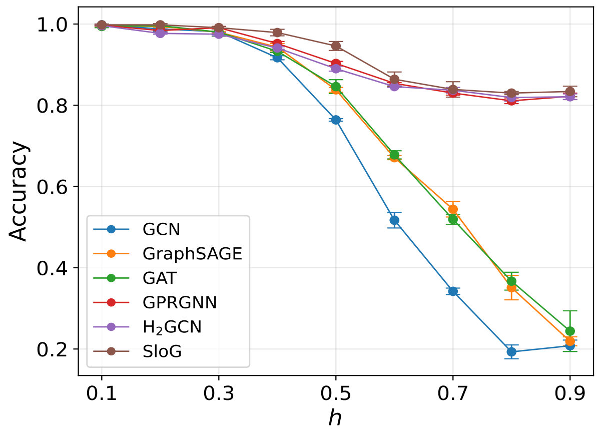


Figure 6. Performance comparison on synthetic datasets with different heterophily ratios h.

Figure 6 shows the performance comparison of SLOG(B) and several baselines on synthetic datasets of different h values. It can be seen that our method demonstrates superior performance on synthetic datasets across different heterophily ratios h, indicating its robustness to varying heterophilic/homophilic balances. Notably, general GNN methods (e.g., GCN, GraphSAGE, GAT) show high sensitivity to h, especially under high h values where their performance drops significantly, suggesting their limitation in heterophilic contexts. In contrast, heterophilic graph-oriented methods (e.g., GPRGNN, H$_2$GCN) show more robustness. Yet our method still outperforms them, demonstrating our method's ability to adaptively

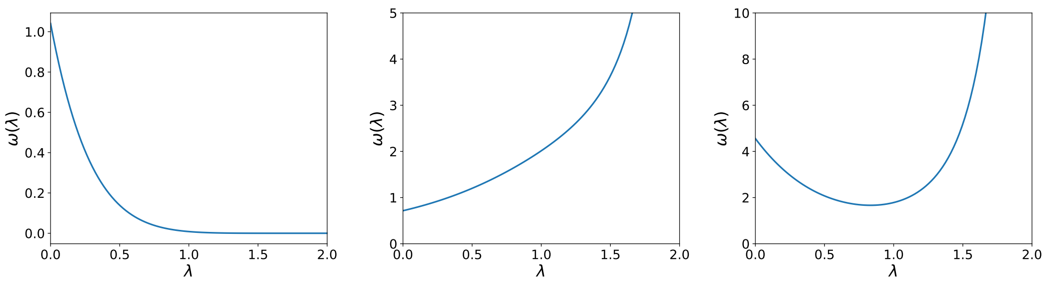
Table 9. Performance comparison on synthetic datasets.

h	0.1	0.2	0.3	0.4	0.5
MLP	0.771±0.018	0.803±0.005	0.826±0.007	0.817±0.004	0.857±0.006
GCN	**0.998±0.002**	0.988±0.005	0.981±0.007	0.917±0.005	0.764±0.003
ChebNet	0.845±0.004	0.823±0.002	0.856±0.019	0.818±0.005	0.817±0.005
GraphSAGE	0.996±0.003	0.994±0.002	0.980±0.002	0.943±0.009	0.838±0.006
GAT	0.994±0.003	0.996±0.002	0.980±0.010	0.933±0.003	0.846±0.017
APPAP	0.996±0.003	0.975±0.004	0.962±0.005	0.930±0.006	0.874±0.005
SGC	0.997±0.002	0.989±0.004	0.970±0.013	0.899±0.009	0.744±0.008
GATv2	0.995±0.001	0.982±0.003	0.983±0.009	0.952±0.007	0.901±0.008
GPRGNN	0.997±0.001	0.984±0.004	**0.991±0.003**	0.952±0.005	0.903±0.005
H_2GCN	0.996±0.001	0.977±0.002	0.975±0.004	0.941±0.003	0.890±0.006
FAGCN	0.993±0.004	0.981±0.004	0.973±0.014	0.952±0.003	0.905±0.004
BernNet	0.830±0.003	0.839±0.006	0.845±0.008	0.823±0.012	0.812±0.004
JacobiConv	0.213±0.022	0.187±0.020	0.213±0.031	0.235±0.039	0.201±0.038
SLOG(B)	**0.998±0.002**	**0.998±0.003**	**0.991±0.003**	**0.979±0.008**	**0.946±0.011**

h	0.6	0.7	0.8	0.9
MLP	0.813±0.005	0.825±0.006	0.828±0.002	0.810±0.006
GCN	0.517±0.019	0.342±0.008	0.193±0.017	0.208±0.014
ChebNet	0.805±0.006	0.814±0.014	0.782±0.007	0.804±0.006
GraphSAGE	0.671±0.005	0.544±0.019	0.351±0.030	0.219±0.011
GAT	0.678±0.010	0.519±0.012	0.367±0.022	0.244±0.050
APPAP	0.831±0.005	0.832±0.005	0.799±0.007	0.751±0.009
SGC	0.481±0.005	0.331±0.013	0.194±0.018	0.195±0.012
GATv2	0.852±0.016	0.776±0.019	0.654±0.068	0.413±0.046
GPRGNN	0.854±0.002	0.830±0.005	0.811±0.007	0.822±0.008
H_2GCN	0.846±0.002	0.837±0.007	0.819±0.009	0.821±0.007
FAGCN	0.846±0.006	0.828±0.002	0.820±0.007	0.822±0.005
BernNet	0.783±0.011	0.822±0.009	0.810±0.007	0.800±0.013
JacobiConv	0.235±0.051	0.341±0.047	0.378±0.080	0.384±0.101
SLOG(B)	**0.864±0.018**	**0.839±0.019**	**0.830±0.004**	**0.834±0.013**

capture both homophilic and heterophilic information.

Figure 7 shows the learned filters of our method on synthetic datasets with different h. Learned filters on synthetic datasets reveal adaptability to the heterophily ratio: a low-pass filter for small h (better for homophilic graphs) and a high-pass filter for large h (suited for heterophilic graphs). Especially, combining datasets with contrasting h values results in a band-stop filter, a hybrid of high-pass and low-pass filters. This is consistent with our intuition that the filter should be able to capture both homophilic and heterophilic information.



(a) h=0.1 (b) h=0.9 (c) h=0.1&0.9 combined

Figure 7. Learned filters on synthetic datasets with different h. Note that Figure 7(c) is the learned filter when two dataset with h=0.1 and h=0.9 are combined.

B.4. Convergence Study

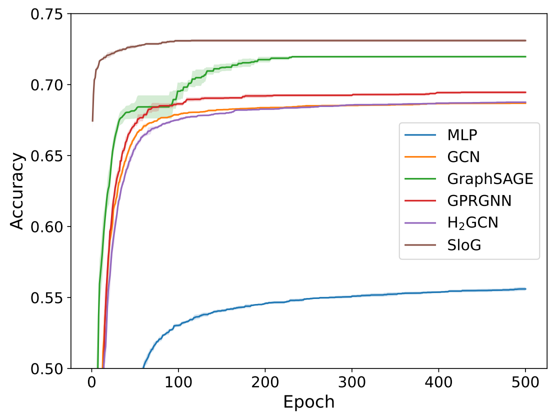


Figure 8. Convergence comparison on Ogbn-arxiv dataset.

To assess the convergence speed of SLOG(B), we performed experiments on the Ogbn-arxiv dataset, comparing it with various baselines. The results, depicted in Figure 8, demonstrate that SLOG(B) converges faster than the other methods. This rapid convergence is due to the method's simplicity, characterized by only two learnable exponents, p and q, along with several weight matrices.

B.5. Optimization Study

SLOG(B) comprises two groups of learnable parameters: those from the filter with real-valued order and those from weight matrices. in Section 4, we utilize Adam to optimize the orders of SLOG filter, p, q. However, the parameters p, q in the filter, being in the exponent position, are challenging to converge using traditional gradient descent methods or other commonly-used gradient-based optimizers. We evaluate two alternative optimization methods, Alternating Minimization in Section B.5.1 and Gaussian Process in Section B.5.2.

B.5.1. ALTERNATING MINIMIZATION

Alternating minimization can partially address this convergence issue, enabling the model to converge more rapidly (Li et al., 2019; Ghosh & Kannan, 2020). Specifically, this method sequentially optimizes one group of variables while fixing the others, then recursively repeats this optimization sequence. We apply alternating minimization to **SLOG(B)**, dividing the parameters into filter-related and weight-related groups to facilitate convergence.

We start by initializing all parameters. Then, we fix p and q in the filter and train all other parameters. After this, we fix all parameters except for p and q and continue the training. This process is repeated until the model converges or reaches the maximum number of iterations. The details are presented in Algorithm 3, where d denotes the tuple of (p, q).

We assess the algorithm using the same settings described in Section 4.2. We set the total number of minimization steps, $2TS = 500$, equal to the maximum iteration number used in Section 4.2. T is maximum iteration number of the outer loop, and S is that of the inner loop. For Squirrel and Squirrel-filt., we use $S = 5$. For Chameleon, we set $S = 10$, and for the remaining datasets, $S = 25$. The results are presented in Table 10.

Table 10. Performance of Gaussian Process optimization.

Datasets	Cora	Citeseer	Squirrel	Chameleon	Squirrel-filt.	Chameleon-filt.	Minesweeper
SLOG(B)	0.865±0.011	0.766±0.026	0.392±0.006	0.581±0.024	0.427±0.013	0.420±0.023	0.822±0.009
w. AM	**0.876±0.014**	**0.771±0.004**	**0.412±0.012**	**0.589±0.010**	**0.428±0.025**	**0.424±0.028**	**0.825±0.004**

The results indicate that alternating minimization enhances the performance of the original model, even with the same number of iterations. This suggests that the filter parameters, though challenging to train, can be effectively optimized with

Algorithm 3 SLOG(B) with Alternating Minimization

```
1:  Input: Model $\mathcal{M}$; dataset $\mathcal{D}$; parameter space $\mathcal{X}_d$ and $\mathcal{X}_\theta$; loss function $\mathcal{L}$; optimizer $\mathcal{O}$; learning rate $\eta$; maximum alternation iteration number $T$; maximum optimization step number $S$.
2:  Output: Optimal $d^*$ and $\theta^*$.
3:  Initialize $\theta^{(0,0)}, d^{(0,0)}$ randomly
4:  for $t = 0$ to $T - 1$ do
5:      # Step 1: optimization for $\theta$
6:      for $s = 1$ to $S$ do
7:          $\theta^{(t,s)} \leftarrow \arg\min_{\theta \in \mathcal{X}_\theta} \mathcal{L}(\mathcal{M}(d^{(t,0)}, \theta^{(t,s-1)}), \mathcal{D})$ using Optimizer $\mathcal{O}$
8:      end for
9:      $\theta^{(t+1,0)} \leftarrow \theta^{(t,s)}$
10:     # Step 2: optimization for $d$
11:     for $s = 1$ to $S$ do
12:         $d^{(t,s)} \leftarrow \arg\min_{d \in \mathcal{X}_d} \mathcal{L}(\mathcal{M}(d^{(t,s-1)}, \theta^{(t+1,0)}), \mathcal{D})$ using Optimizer $\mathcal{O}$
13:     end for
14:     $d^{(t+1,0)} \leftarrow d^{(t,s)}$
15: end for
16: $d^*, \theta^* \leftarrow d^{(T,0)}, \theta^{(T,0)}$
17: return $d^*, \theta^*$
```

appropriate methods.

B.5.2. GAUSSIAN PROCESS

SLOG(B) includes parameters p, q (as in Eq. (4)) and other weight matrices. The placement of p, q in the exponent position makes training difficult. Large values of p, q can lead to excessively large gradients, risking gradient explosion, while small values may cause vanishing gradients. Both scenarios present challenges in achieving stable convergence. Therefore, we explore an alternative optimization method to tackle this issue.

We denote the tuple of (p, q) as d and all weight matrices as θ. The optimization problem for SLOG(B) can be formulated as:

$$d^*, \theta^* = \underset{d \in \mathcal{X}_d, \theta \in \mathcal{X}_\theta}{\arg\min} \mathcal{L}(\mathcal{M}(d, \theta), \mathcal{D}). \tag{10}$$

We approach this optimization using a bi-level strategy:

$$\begin{aligned} d^* &= \underset{d \in \mathcal{X}_d}{\arg\min} \mathcal{L}(\mathcal{M}(d, \theta^*), \mathcal{D}) \\ s.t.\ \theta^* &= \underset{\theta \in \mathcal{X}_\theta}{\arg\min} \mathcal{L}(\mathcal{M}(d, \theta), \mathcal{D}). \end{aligned} \tag{11}$$

The lower-level optimization (optimization of θ) can be solved using traditional methods such as gradient descent or advanced optimizers like Adam (Kingma & Ba, 2014), AdamW (Loshchilov & Hutter, 2017).

The high-level optimization resembles a hyper-parameter search problem. We employ Bayesian Optimization methods, such as Gaussian Process, for this purpose.

Here, we briefly introduce Gaussian Process. Gaussian Process is a stochastic search process, in which every data point searched at different timestamp obeys a multi-variant normal distribution. A key factor of the process is the covariance functions K, also known as Gaussian kernels, which determine the covariance of the variable collection. The most common covariance function is squared exponential kernel, defined as follows:

$$K(\mathbf{d}, \mathbf{d}') = \exp(-\frac{1}{2\ell^2} ||\mathbf{d} - \mathbf{d}'||^2), \tag{12}$$

where ℓ is the characteristic length scale. Therefore, if we set the mean value of the multi-variant normal distribution as 0, then we can derive the prediction of newly searched data point $\mathbf{d}$ as:

$$\begin{bmatrix} \mathbf{y} \\ \mathbf{y}' \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} K(\mathbf{d}, \mathbf{d}) & K(\mathbf{d}, \mathbf{d}') \\ K(\mathbf{d}', \mathbf{d}) & K(\mathbf{d}', \mathbf{d}') \end{bmatrix}\right), \tag{13}$$

where $y = \mathcal{L}(\mathcal{M}(d, \theta), \mathcal{D})$ indicates the loss (also the performance) given the model parameters. From this, we can derive the following equation (Rasmussen et al., 2006):

$$\mathbf{y}'|\mathbf{y} \sim \mathcal{N}(K(\mathbf{d}', \mathbf{d})K(\mathbf{d}, \mathbf{d})^{-1}\mathbf{y}, K(\mathbf{d}', \mathbf{d}') - K(\mathbf{d}', \mathbf{d})K(\mathbf{d}, \mathbf{d})^{-1}K(\mathbf{d}, \mathbf{d}')). \tag{14}$$

Therefore, the most likely value of y' is given by:

$$\mathbf{y}' = K(\mathbf{d}', \mathbf{d})K(\mathbf{d}, \mathbf{d})^{-1}\mathbf{y}. \tag{15}$$

Thus, we alternate the optimization of model parameters θ and hyper-parameters d. For further details on Bayesian optimization and Gaussian Process, see (Rasmussen et al., 2006; Frazier, 2018). The optimization algorithm is detailed in Algorithm 4.

Algorithm 4 SLOG(B) with Gaussian Process optimization

```
1: Input: Model $\mathcal{M}$; dataset $\mathcal{D}$; parameter space $\mathcal{X}_d$ and $\mathcal{X}_\theta$; loss function $\mathcal{L}$; optimizer $\mathcal{O}$; learning rate $\eta$; maximum iteration number $T$.
2: Output: Optimal $d^*$ and $\theta^*$.
3: Initialize $\theta^{(0)}, d^{(0)}$ randomly
4: $\mathcal{Y} \leftarrow \emptyset$
5: for $t = 0$ to $T - 1$ do
6:     # Lower-level: optimization for $\theta$
7:     $\theta^* \leftarrow \arg\min_{\theta \in \mathcal{X}_\theta} \mathcal{L}(\mathcal{M}(d^{(t)}, \theta), \mathcal{D})$ using Optimizer $\mathcal{O}$
8:     $y^{(t)} \leftarrow \mathcal{L}(\mathcal{M}(d^{(t)}, \theta^*), \mathcal{D})$
9:     $\mathcal{Y} \leftarrow \mathcal{Y} \cup \{d^{(t)}, y^{(t)}\}$
10:    # Upper-level: optimization for $d$
11:    $\mathcal{C} \leftarrow \{d_i \sim \mathcal{X}_d\}_{i=1}^N$
12:    Calculate $y_i$ according to Eq. (15) for each $d_i \in \mathcal{C}$
13:    $idx \leftarrow \arg\max_{i \in |\mathcal{C}|} y_i$
14:    $d^{(t+1)} \leftarrow d_{idx} \in \mathcal{C}$
15: end for
16: $k \leftarrow \arg\min_i \{y^{(i)} | (d^{(i)}, y^{(i)}) \in \mathcal{Y}\}$
17: return $d^{(k)}, \theta^{(k)}$
```

We evaluate the algorithm using the same settings described in Section 4.2, setting the maximum iteration number $T = 20$. The results are shown in Table 11.

Table 11. Performance of Gaussian Process optimization.

Datasets	Cora	Citeseer	Squirrel	Chameleon	Squirrel-filt.	Chameleon-filt.	Minesweeper
SLOG(B)	**0.865±0.011**	0.766±0.026	0.392±0.006	0.581±0.024	**0.427±0.013**	0.420±0.023	0.822±0.009
w. GP	0.853±0.022	**0.778±0.010**	**0.417±0.005**	**0.586±0.021**	0.402±0.017	**0.426±0.044**	**0.826±0.010**

This evaluation demonstrates that Gaussian Process optimization significantly enhances the performance of SLOG(B). As shown in Table 11, the model optimized with Gaussian Process outperforms that using the original settings across most of the datasets. Note that Gaussian Process will not increase the computational complexity of SLOG, since the total time complexity of Gaussian Process is $O(n^3 + n^2m)$, where n is the number of elements in training set (i.e., $|\mathcal{Y}|$ in Algorithm 4) and m is the amount of samples to be predicted (i.e., $|\mathcal{C}|$ in Algorithm 4). However, due to the rather limited training set scale ($n \sim 10$) and prediction set scale ($m \sim 100$ in the experiments), the complexity of Gaussian Process (Line 10-14, Algorithm 4) is negligible in comparison to that of model training (Line 6-9, Algorithm 4).

C. Experimental Details

C.1. Dataset Statistics

For datasets like Cora, Citeseer, Chameleon, Squirrel, DBLP, Coauthor-CS, and Coauthor-Physics, we collect them from Pytorch Geometric library (Fey & Lenssen, 2019). For Chameleon-filtered, Squirrel-filtered, Minesweeper, Tolokers, Amazon-ratings, Questions, we collect the raw data released from (Platonov et al., 2023)[9]. For Ogbn-arxiv, we collect it from Open Graph Benchmark (Hu et al., 2020). The statistics of these datasets are shown in Table 12.

Table 12. The statistics of the datasets.

Datasets	Nodes	Edges	Features	Classes	Heterophily
Cora	2,708	10,556	1,433	7	0.190
Citeseer	3,327	9,104	3,703	6	0.264
Chameleon	2,277	62,792	2,325	5	0.765
Squirrel	5,201	396,846	2,089	5	0.776
DBLP	17,716	105,734	1,639	4	0.172
Coauthor-CS	18,333	163,788	6,805	15	0.192
Coauthor-Physics	34,493	495,924	8,415	5	0.069
Chameleon-filtered	890	13,584	2,325	5	0.764
Squirrel-filtered	2,223	65,718	2,089	5	0.793
Minesweeper	10,000	39,402	7	2	0.317
Tolokers	11,758	519,000	10	2	0.405
Amazon-ratings	24,492	93,050	300	5	0.620
Questions	48,921	153,540	301	2	0.160
Flickr	89,250	899,756	500	7	0.681
Ogbn-arxiv	169,343	1,166,243	128	40	0.322
Reddit	232,965	114,615,892	602	41	0.244

C.2. Implementation Details

For small-scale datasets, we set the hidden dimension of all methods to 128, the learning rate to 0.01. For large-scale datasets, we set the hidden dimension as 512, and fine-tune the learning rate in the range of $[10^{-2}, 10^{-3}, 10^{-4}]$. We use Adam as the optimizer. For all datasets, we set the weight decay to 0.0005. For all datasets except Reddit, we limit the training epochs to 500. For Reddit, we set maximum training epochs as 30. The layer number L for SLOG(N) is set to 3. We determine the size of the subgraphs partitioned by SLOG(L) to be approximately 512, and the parameter β is set to 0.5. For hyper-parameters of baselines, we use the default configurations if available, otherwise, we use the same hyper-parameter space for SLOG. We run all the experiments on a Tesla-V100 GPU with 32G Memory.

C.3. Graph Partition: METIS Algorithm

In SLOG(L), METIS (Karypis & Kumar, 1998) is used for graph partition. Here, we provide more information about graph partition and METIS.

A k-way graph partition is defined as minimizing the edge-cut to conduct the following node partition (Karypis & Kumar, 1996): given a graph $\mathcal{G} = \{(\mathcal{V}, \mathcal{E})\}$ with $|V| = n$, partition the node set $\mathcal{V}$ into k subsets, $\mathcal{V}_1, \cdots, \mathcal{V}_k$, with the minimal number of edges whose incident verticles belong to different subset, subject to the constraint that $\mathcal{V}_i \cap \mathcal{V}_j = \emptyset$ for $i \neq j$, $\cup_{i=1}^{k} \mathcal{V}_i = \mathcal{V}$ and $|\mathcal{V}_i| = n/k$ for $i = 1, \cdots, k$. To simplify the problem, we introduce bisection graph partition, which is a special case of k-way graph partition with $k = 2$. The METIS algorithm solve the problem with the following steps (Karypis & Kumar, 1998):

Step 1: Coarsening Phase. The original graph $\mathcal{G}_0$ will produce a sequence of graph $\mathcal{G}_1, \cdots, \mathcal{G}_m$, such that $|\mathcal{V}_0| > |\mathcal{V}_1| > \cdots > |\mathcal{V}_m|$. A set of nodes in $\mathcal{G}_i$ can be coarsened to a single *multinode* in $\mathcal{G}_{i+1}$. The coarsening strategy is related to *matching*, which is defined as a set of edges in which no two edges are incident on a same node. There are lots of matching algorithm, and METIS uses *heavy-edge matching* with a slight modification. One can refer to (Karypis & Kumar, 1998) for more details. The coarsening phase is repeated until the graph size is small enough.

Step 2: Partition Phase. We compute the bi-parition for $\mathcal{G}_m$, guaranteeing that each part after partition contains approximately half of the nodes in the original graph $\mathcal{G}_0$. Since the number of nodes in $\mathcal{G}_m$ is small, the time consumed in this step is rather small.

Step 3: Uncoarsening Phase. The partition of $\mathcal{G}_m$ is projected back to $\mathcal{G}_0$ by applying the same partition to the coarser graphs $\mathcal{G}_{m-1}, \cdots, \mathcal{G}_1$. The uncoarsening phase is repeated until the original graph $\mathcal{G}_0$ is reached.

[9]Github link: https://github.com/yandex-research/heterophilous-graphs/tree/main/data.

We implement graph partition with the help of PyMETIS[10] library, which is a python wrapper for METIS.

[10]Github link: https://github.com/inducer/pymetis.