Graph Neural Networks with Feature and Structure Aware Random Walk

Wei Zhuo, Chenyun Yu, Guang Tan
Sun Yat-sen University
zhuow5@mail2.sysu.edu.cn, {yuchy35,tanguang}@mail.sysu.edu.cn

1 INTRODUCTION

Graph Neural Networks (GNNs) have received increasing attention for representation learning in various machine learning tasks. However, most existing GNNs applying neighborhood aggregation usually perform poorly on the graph with heterophily where adjacent nodes belong to different classes. In this paper, we show that in typical heterophilous graphs, the edges may be directed, and whether to treat the edges as is or simply make them undirected greatly affects the performance of the GNN models. Furthermore, due to the limitation of heterophily, it is highly beneficial for the nodes to aggregate messages from similar nodes beyond local neighborhood. These motivate us to develop a model that adaptively learns the directionality of the graph, and exploits the underlying long-distance correlations between nodes. We first generalize the graph Laplacian to digraph based on the proposed Feature-Aware PageRank algorithm, which simultaneously considers the graph directionality and long-distance feature similarity between nodes. Then digraph Laplacian defines a graph propagation matrix that leads to a model called DiglacianGCN. Based on this, we further leverage the node proximity measured by commute times between nodes, in order to preserve the nodes’ long-distance correlation on the topology level. Extensive experiments on ten datasets with different levels of homophily demonstrate the effectiveness of our method over existing solutions in the task of node classification.

CCS CONCEPTS
• Mathematics of computing → Graph algorithms; • Computing methodologies → Learning latent representations; Neural networks.

KEYWORDS
Graph neural networks, network representation learning, deep learning

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

Woodstock ’18, June 03–05, 2018, Woodstock, NY © 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-XXXX-X/18/06 . $15.00
https://doi.org/10.1145/1122445.1122456
(digraph), called Diglacian (Digraph Laplacian), based on the transition matrix and its stationary distribution. To make the construction possible for general graphs and simultaneously capable of capturing long-distance feature similarity, we propose a Feature-Aware PageRank algorithm to strengthen the original graph. The algorithm guarantees strong connectivity of the graph, and hence the existence of the stationary distribution; it also gives each node a certain probability to jump to other nodes with similar features. The graph propagation matrix based on the spectral analysis of Diglacian leads to a model called DiglacianGCN.

On top of Diglacian, we further consider capturing nodes’ long-distance correlation in topology, by defining a measure of node proximity based on nodes’ pair-wise commute time. The commute time between node \( v_i \) and \( v_j \) is defined as the expected number of steps of a random walk starting at \( v_i \) reaching \( v_j \) and then returning to \( v_i \). The less commute time from \( v_i \) to \( v_j \), the higher impact of \( v_j \) on \( v_i \), and therefore they should be more similar in the embedding space. For example, when performing a random walk on a social network, the walker starting at an ordinary user can immediately reach a celebrity it follows, but the walker can hardly return to the starting node by the celebrity’s outgoing links. If the walker reaches another ordinary user and returns in a few steps, then the starting node should be more similar to that ordinary user than to the celebrity in the embedding space. Besides, since the random walk is based on the transition matrix of Feature-Aware PageRank, commute times hence are also feature-aware and direction-dependent. We theoretically prove that commute times can be derived by our proposed Diglacian in a sparse manner, and define a propagation matrix based on it. We call this method DiglacianGCN-CT (Commute Time).

Another novelty of our model is a "soft" treatment of the graph’s directionality. Many real-world networks are directed, yet previous methods usually use the direction property in a naive way: the direction property of edges is either used as is, or simply discarded. Table 1 lists several widely used heterophilous graph benchmarks, all exhibiting a directed structure. We use subscripts \( d \) and \( u \) to denote models based on a directed version and an undirected version of the graph, respectively. The accuracy results of node classification are shown in the table, from which we can see that whether one should use the direction property is not obvious and the choice can be quite influential in performance. To avoid the problem, an additional propagation matrix based on the symmetrized adjacency matrix, which corresponds to the undirected version of the graph, is integrated into our model. The model can automatically learn the weight of each version of the graph, thus best utilizing the direction information for the benefit of considered task.

We conduct comprehensive experiments on ten graph datasets, including seven disassortative (heterophilous) graphs and three assortative (homophilous) graphs. The empirical evaluations demonstrate the universality and superiority of our models.

## 2 RELATED WORK

GNNs have achieved tremendous success on homophilous (assortative) graphs [11, 14, 17, 28, 33, 34]. However, recent work [26, 39] shows that traditional MPNN-based GNNs perform poorly on heterophilous (disassortative) graphs, and provide two metrics to measure the degree of homophily on the node level and edge level.

| % Directed edges | Texas | Wisconsin | Cornell | Chameleon |
|------------------|-------|-----------|---------|-----------|
| GCN_d            | 51.97 | 50.85     | 61.93   | 67.96     |
| GCN_u            | 61.08 | 53.14     | 57.84   | 63.82     |
| GraphSAGE_u      | 83.92 | 85.07     | 78.65   | 58.73     |
| GraphSAGE_d      | 77.03 | 83.33     | 81.89   | 69.69     |
| GAT_u            | 58.92 | 55.95     | 60.69   | 60.69     |
| GAT_d            | 52.73 | 61.37     | 62.73   | 57.66     |

To tackle the limitation of heterophily, the early method Geom-GCN [26] precomputes unsupervised node embeddings and uses neighborhoods defined by geometric relationships in the resulting latent space to define graph convolution. H2GCN [39] proposes to make full use of high order neighborhoods, and combine self-embeddings and neighbor embeddings using concatenation. CPGNN [38] integrates the compatibility matrix as a set of learnable parameters into GNN, which it initializes with an estimated class compatibility matrix. PPNP [18] and GDC [19] redefine the node proximity using PageRank and Heat Kernel PageRank. GPRGNN [6] performs feature aggregation for multiple steps to capture long-range information and then linearly combines the features aggregated with different steps, where the weights of the linear combination are learned during the model training. Some other methods like FAGCN [3] use an attention mechanism and learn the weight of an edge as the difference in the proportion of low-frequency and high-frequency signals. AdaGNN [9] leverages a trainable filter that spans across multiple layers to capture the varying importance of different frequency components for node representation learning.

## 3 METHODOLOGY

### 3.1 Diglacian

In the general setting, \( G = (V, E, X) \) is used to denote an unweighted directed graph, where \( V = \{v_i\}_{i=1}^{N} \) is the node set, \( E \subseteq (V \times V) \) is the edge set, \( X \in \mathbb{R}^{N \times d} \) is the node feature matrix with the number of features \( d \) per node. Let \( A \in \mathbb{R}^{N \times N} \) be the adjacency matrix and \( D = \text{diag}(\text{deg}(v_1), \cdots, \text{deg}(v_N)) \in \mathbb{R}^{N \times N} \) be the degree matrix of \( A \), where \( \text{deg}(v_i) = \sum_{v_j \in V} A_{ij} \) is the out-degree of \( v_i \). Let \( \bar{A} = A + I \) and \( \bar{D} = D + I \) denote the augmented adjacency and degree matrix with self-loops, respectively. The transition probability matrix of the Markov chain associated with random walks on \( G \) can be defined as \( P = D^{-1}A \), where \( P_{ij} = A_{ij}/\text{deg}(v_i) \) is the probability of a 1-step random walk starting from \( v_i \) to \( v_j \). Graph Laplacian formulated as \( L = D - A \) is defined on the undirected adjacency graph whose adjacency matrix is symmetric. The augmented symmetrically normalized Laplacian with self-loop [34] is defined as \( \tilde{L} = \bar{D}^{-\frac{1}{2}}L\bar{D}^{-\frac{1}{2}} \), where \( \tilde{L} = \bar{L} - \bar{A} \).

Some spectral graph neural networks [8, 17] simply make a given directed graph undirected by adding reverse edges to node pairs connected by single-directed edges using \( A_{uv} = \frac{A_{vu} + A_{vu}}{2} \). Although it helps explain GNNs in terms of spectral analysis, the original
graph structure is disturbed due to the forced use of a symmetrized adjacency matrix, which can make different graphs share the same Laplacian and many properties of random walks hard to obtain, such as hitting times, cover times and commute times. It is known that the Laplacian is defined as the divergence of the gradient of a signal on an undirected graph. For a signal \( f \in \mathbb{R}^N \), \( (Lf)(i) = \sum_{j \in N_i} A_{ij}(f_i - f_j) \). In effect, the Laplacian on \( f \) acts as a local averaging operator which is a node-wise measure of local smoothness. Following the meaning of the undirected graph Laplacian, we now generalize the existing spectral graph theory defined for undirected graphs to digraphs by defining Diglacian \( T \) acting on \( f \):
\[
(Tf)(i) = \sum_{v_j \in N_i^+} P_{ij}(f_i - f_j) = ((D^{-1} - P)f)(i)
\]

Here, we replace \( A \) with its row-normalization \( P \). \( N_i^+ \) is the set of \( v_j \)’s outgoing neighbors. The Diglacian can be defined as \( T = D^{-1} - P \). Nevertheless, Diglacian \( T \) cannot directly capture the unique nature of random walks on the digraph. Leveraging the inherent equivalence between digraph and Markov chain [27], we can solve this problem by computing the stationary probability distribution and integrating it into Diglacian.

First we assume the digraph \( G \) is irreducible and aperiodic. A fundamental result from [27] is that \( G \) has a unique stationary probability distribution \( \pi \) (i.e., Perron vector), satisfying the balance equation \( \pi_i = \sum_{v_j \in N_i^-} \pi_j P_{ji} \), where \( N_i^- \) is the set of incoming neighbors of \( v_i \). The stationary distribution \( \pi \) can be computed by recurrence and it converges to the left eigenvector \( \bar{\pi} \) of the dominant eigenvalue of the transition matrix \( P \). \( \pi \) satisfies \( \sum_i \pi_i = 1 \) and the \( i \)-th element of \( \pi_i \) is strictly positive. It can be interpreted as the limiting probability of finding a \( i \)-length random walk starting at any other nodes and ending at node \( v_i \), i.e., \( \lim_{n\to\infty} P^k(i) = \pi_i \). Thus, \( \pi_i \) can be used to measure the global importance of \( v_i \), and we further redefine eq. (1) as:
\[
(Tf)(i) = \sum_{v_j \in N_i^+} \pi_j P_{ij}(f_i - f_j) = \pi_i ((D^{-1} - P)f)(i),
\]

where \( \Pi = \text{diag}(\pi_1, \ldots, \pi_N) \) and Diglacian \( T = \Pi(D^{-1} - P) \). However, our proposed Diglacian is based on a strong assumption, that is, \( G \) is irreducible and aperiodic, which does not necessarily hold for general graphs. The input digraph may contain multiple connected components or absorbing nodes that make the graph reducible; or it may contain cyclic structures that make the graph periodic. In these cases, there is no guarantee for the positive, existence and uniqueness of the stationary distribution \( \pi \). The solution in [31] adds a teleporting probability distribution over all the nodes to address this issue, with the help of PageRank [25] that amends the transition matrix as \( P_{pr} = \alpha P + (1 - \alpha) \frac{1}{N} \), where \( \alpha \in (0, 1) \) and \( \frac{1}{N} \) is the all-one column vector. \( P_{pr} \) means that the walker can randomly choose a non-neighbor node as the next step with probability \( 1 - \alpha \). It is obvious that \( P_{pr} \) is irreducible and aperiodic, so it has a unique \( \pi \). However, this solution yields a complete graph with a dense matrix \( P_{pr} \), which is extremely unfriendly to subsequent operations.

Noticing that the input graph is attributed, i.e., every node has a feature vector \( x \in \mathbb{R}^d \) (node degree, centrality or shortest path matrix can be node features if \( X \) is absent), we therefore provide an alternative method based on both features and topological structures of the graph, namely Feature-Aware PageRank, which yields a sparse, irreducible and aperiodic transition matrix. In addition, for graphs with strong heterophily, the original topological structure is unreliable for MPNN-based GNNs, while the global feature-wise similarity between nodes provides an opportunity for the model to mitigate the effect of heterophilous structure.

### 3.2 Feature-Aware PageRank

Applying teleportations in classical PageRank aims to solve two problems in the web graph, i.e., dead ends and spider traps [25]. Specifically, dead ends mean some nodes do not have any outgoing neighbors, which leads to PageRank scores converging to 0 on all nodes. Spider traps mean there exist absorbing nodes in the graph, which leads to PageRank scores converging to 0 except the absorbing nodes. Thus, we only need to construct an irreducible graph, i.e., a strongly connected graph, that can solve the above two problems. Meanwhile, the graph need to be aperiodic to guarantee the PageRank transition matrix has a unique stationary distribution.

Instead of using \( P_{pr} \) as the transition matrix of the graph, we propose Feature-Aware PageRank (FPR), which aims to construct an irreducible graph based on node feature similarity, with its transition matrix denoted by \( P_{fpr} \). The main difference from \( P_{pr} \) is that \( P_{fpr} \) gives teleport probabilities to all nodes, while \( P_{pr} \) only gives teleport probabilities to \( k \) nearest neighbors in the feature space. In other words, the walker at a node not only has a probability to transit to its outgoing neighbors, but is also subject to a probability of teleporting to \( k \) nodes with most similar features to the current node.

It is easy to construct a kNND graph based on \( X \). However, a kNND graph is not necessarily irreducible, and has \( O(N^2) \) time complexity. Thus, we propose an approximate similarity sorting method to solve the problem. The idea behind is the transitivity of similarity. We first assume node features are non-negative (to be relaxed later on), then all node features are located in one orthant. Let \( \hat{X} = \{ \hat{x}_i \}_{i=1}^N \) be the \( l_2 \)-normalization over each row of \( X \), i.e., \( \hat{x}_i = LN_{\hat{X}}(x_i) = \frac{x_i}{\|x_i\|} \).

Let \( \hat{x} \) be the \( l_2 \)-normalization of \( \frac{1}{N} \sum_i \hat{x}_i \). By this means, \( \hat{x} \) and all node features in \( \hat{X} \) are shrunk to magnitude 1. We give an example in 2-dimensional Euclidean space in the upper part of Figure 1. To compare the similarity between node features in \( \hat{X} \), we introduce an auxiliary vector \( a \) to transit similarity, i.e., if two nodes have a similar similarity score with \( a \), they are similar and more likely to be connected. In this way, we do not need to calculate all pair-wise similarity scores. In order to make dissimilarity vectors have different similarity scores to \( a \), \( a \) is better not in the same or diagonally opposite orthant with \( \hat{x} \). Specifically, we first randomly initialize \( a \) to a \( d \)-dimensional vector, where \( a \neq \hat{x} \) and \( a \neq 0 \), then the auxiliary vector \( a \) is defined as:
\[
a = LN_{\hat{X}}(a - (a^T \hat{x}) \hat{x}).
\]

It is obvious that \( a \) is orthogonal to \( \hat{x} \). Therefore, features with high similarity in \( \hat{X} \) will be mapped to the close positions on \( a \). The goal of the approximate similarity sorting is to construct a graph.
based on an ordering of nodes, where nodes that are similar with respect to the auxiliary vector are connected with each other by undirected edges. For example, if the similarity score between $a$ and $\hat{x}_i$ is similar to it between $a$ and $\hat{x}_j$, $v_i$ and $v_j$ are similar and connected. An illustration of the similarity sorting is shown in Figure 1.

![Figure 1: An illustration of the approximate similarity sorting.](image)

The similarity sorting graph $S = (A_S, X)$ represent that each node connects to it top-$k$ most similar nodes, where $k = 2 = \text{window size}$ in Figure 1. However, since we use one auxiliary vector, the similarity sorting is completely accurate only when $d = 2$. Considering similar features are not mapped far away, we hence add skip connection edges on $S$ and use the hyperparameter $\text{window size}$ to control the number of neighbors. $\text{window size}$ can be set to any positive integer, for example we show $\text{window size} = 2$, i.e., $k = 4$ by adding dummy line edges to $S$ as shown in Figure 1. To guarantee sparsity, $k$ is usually set to a small value. The final combinatorial graph is denoted $G = (\mathcal{A}, X)$, where $\mathcal{A} = \max(A, A_u)$ is the element-wise maximum. The transition matrix of the combinatorial digraph $G$ is $P_{pfpr} = D^{-1}\mathcal{A}$, where $D$ is the diagonal degree matrix of $G$. Performing random walk on $G$ is called Feature-Aware PageRank, since in each step the walker has a possibility of teleporting to other nodes who have similar features to the current node. Compared with PageRank transition matrix $P_{pr}$, which uses all nodes as teleporters, our Feature-Aware PageRank yields a sparse transition matrix while ensuring irreducibility. Besides, another necessary condition is aperiodicity. To satisfy it, we add a restart probability at each step of a random walk, i.e., jumping to the starting node, namely Personalized Feature-Aware PageRank (PFPR). Thus, the transition matrix of PFPR is $P_{pfpr} = \tilde{D}^{-1}\tilde{A}$, where $\tilde{A} = A + I$ and $\tilde{D}_{ij} = \sum_j \tilde{A}_{ij}$. Adding a self-loop for each node with a non-zero probability makes $\tilde{G}$ aperiodic, because of the greatest common divisor of the lengths of its cycles is one.

The transition matrix $P_{pfpr}$ of PFPR is associated with an irreducible and aperiodic graph $\tilde{G}$ with self-loops. According to Perron-Frobenius Theory [27] for non-negative matrix, $P_{pfpr}$ has a positive left eigenvector $\pi$ corresponding to its dominate eigenvalue 1 with algebraic multiplicity 1, which guarantees the random walk on $\tilde{G}$ converge to a unique positive stationary distribution equal to $\pi$. Therefore, the Diglacian in eq. (2) can be rewritten as:

$$\tilde{T} = \Pi(\tilde{D}^{-1} - P_{pfpr}).$$

The remaining question is how to deal with the feature matrix $X$ with negative entries. We solve it by decomposing $X$ into $X = X^+ + X^-$, where $X^+$ and $X^-$ only contain the positive and negative entries respectively. Based on $X^+$ and $X^-$, we will get two similarity sorting graphs and combine them into the original graph to construct $G$.

### 3.3 DiglacianGCN

For an undirected graph, its Perron vector $\pi$ of transition matrix can be computed by $\pi_i = \frac{d_i}{\sum_4 d_j}$, so the vector of node degrees $A \cdot e$ is a scalar multiple of $\pi$. Besides, since $\pi_i$ is the sum of all incoming probabilities from the neighbors of $v_i$ in a digraph, it hence plays the same role of degree vector in undirected graph that reflects the connectivity between nodes [12]. Based on these properties, [7] and [37] define the normalized graph Laplacian as a Hermitian matrix by using $\Pi$ to normalize transition matrix instead of using the symmetrized adjacency matrix $A_u$. Following the same spirit, we define the augmented normalized Diglacian as:

$$\mathcal{T} = \frac{1}{2}(\Pi^\dagger(\tilde{D}^{-1} - P_{pfpr})\Pi^{-\frac{1}{2}} + \Pi^{-\frac{1}{2}}(\tilde{D}^{-1} - P_{pfpr}^T)\Pi^\dagger),$$

which is a symmetric, real-valued Laplacian with a full set of real eigenvalues. Recall that GCN [17] uses 1-th order polynomials with specific coefficients to approximate the normalized graph Laplacian $D^{-\frac{1}{2}}(D-A)D^{-\frac{1}{2}}$ to $D^{-\frac{1}{2}}(D+A)D^{-\frac{1}{2}}$, which serves as a convolution support to do neighbor weighted aggregation for each node. Thus, following the same purpose, we can directly modify the augmented normalized Diglacian as:

$$\mathcal{T} = \frac{1}{2}(\Pi^\dagger(\tilde{D}^{-1} + P_{pfpr})\Pi^{-\frac{1}{2}} + \Pi^{-\frac{1}{2}}(\tilde{D}^{-1} + P_{pfpr}^T)\Pi^\dagger).$$

The detailed spectral analysis can be found in Appendix A. Considering direction structure of the graph may be not always useful in some cases as stated in Section 1, we take the symmetrized adjacent matrix into consideration, so that the directionality of the graph can be learned. Hence, the $l$-th layer of DiglacianGCN is defined as:

$$H^{(l)}(i) = \sigma(H^{(l-1)}W_0 + (\alpha\tilde{D}^{-1}\mathcal{A}_u + \beta\tilde{T})H^{(l-1)}W_1^{(l)})$$

where $\alpha$ and $\beta$ are the mixing coefficients.
where $\sigma$ is the activation function, $W_0$ and $W_1$ the trainable matrices. Motivated by the good performance of MLP on some heterophilous graphs, we separate the self-embedding and its neighbor embedding, which is an effective design for networks with heterophily \[39\]. We use $\mathcal{D}_u^{-1}\mathcal{A}_u$ and $\bar{T}$ as the neighbor weighted matrices under undirected and directed setting respectively, and $\alpha, \beta \in \mathbb{R}^1$ are trainable scalers to balance them to learn a “soft” directionality. We employ power iteration \[27\] to approximate the stationary matrix, which is an effective design for networks with heterophily \[39\].

Theorem 3.1. Given a combinatorial digraph $\mathcal{G}$ and its Personalized PageRank transition matrix $P_{pfpr}$, the Diglacian of $\mathcal{G}$ is defined as $\bar{T} = \bar{\Pi}(\bar{D}^{-1} - P_{pfpr})$. Then the fundamental matrix $Z$ of $\mathcal{G}$ can be solved by:

$$Z = \bar{\Pi}^{-\frac{1}{2}}(\bar{\Pi}^{-\frac{1}{2}}\bar{\Pi}^{-\frac{1}{2}} - \bar{D}^{-1} + I)^{\frac{1}{2}}\bar{\Pi}^{-\frac{1}{2}},$$ (10)

where the superscript $\dagger$ means Moore–Penrose pseudoinverse of the matrix.

The proof is given in Appendix B. The pseudoinverse of the sparse matrix in eq. (10) can be calculated via low-rank SVD. In particular, we use ARPACK \[20\], an iteration method based on the restarted Lanczos algorithm, as an eigensolver. ARPACK depends on matrix-vector multiplication. Usually a small number of iterations is enough, so if the matrix is sparse and the matrix-vector multiplication can be done in $O(N)$ time parallelly, then the eigenvalues are found in $O(N)$ time as well.

Based on the fundamental matrix defined in eq. (10), we can compute the hitting time matrix $H$ by eq. (9). The commute time matrix can be obtained by $C = H + H^T$, and we set $C_{ij} = 0$ for $i = 1, \cdots, N$. Intuitively, the fewer steps a random walk takes to go from $v_i$ to $v_j$ and then returns to $v_i$, the higher the importance of $v_j$ to $v_i$. Also, since $\mathcal{G}$ is irreducible, all entries in $C$ are positive, which yields a fully connected symmetric graph. It makes the subsequent propagation step computationally expensive and makes little sense for most downstream tasks. Hence, we sparsify $C$ with a certain ratio $\mu \in (0, 1)$. For each row of $C$, we set $\mu N$ largest entries to 0. In doing so, we obtain a sparse non-negative matrix $\mathcal{C}$. Then, we define the normalized graph propagation matrix based on $\mathcal{C}$ as:

$$\tilde{C} = D_s^{-1} \exp(-\tilde{C}^\top).$$ (11)

where $\mathcal{C}^\top$ means positive entries in $\mathcal{C}$, $D_s$ a diagonal matrix, $[D_s]_{ii} = \sum_j |\tilde{C}|_{ij}$. $\mathcal{C}$ is the probability of a node moving to others based on the commute time of a random walk. Since the commute time matrix $C$ is based on the Diglacian $\bar{T}$ of the combinatorial graph $\mathcal{G}$, $\bar{C}$ is also feature-aware. We replace $\bar{T}$ in eq. (7) with $\bar{C}$ and define the $l$-th layer of DiglacianGCN-CT as:

$$H^{(l)} = \sigma(H^{(l-1)}W_0^{(l)} + (\alpha D_u^{-1}A_u + \beta \bar{C})H^{(l-1)}W_1^{(l)}),$$ (12)

where $\alpha$ and $\beta$ are trainable scalers.

Prediction. The class prediction $\hat{Y}$ of a $L$-layer DiglacianGCN or DiglacianGCN-CT is based on node embeddings in the last layer:

$$\hat{Y} = \text{softmax}(H^{(L)}),$$ (13)

where $W_0^{(L)}, W_1^{(L)} \in \mathbb{R}^{d(L-1) \times m}$, $d(L-1)$ the dimension of the $(L-1)$-th layer and $m$ the number of classes.

3.5 Time Complexity

In the stage of constructing the combinatorial graph $\mathcal{G}$, the time complexity of the approximate similarity sorting including cosine similarity computation $O(N)$ and similarity sorting $O(N \log N)$. Then we compute the stationary distribution $\bar{\pi}$ using power iteration, whose time complexity is $O(t N)$, where $t \ll N$ is usually small. The total time complexity of an $L$-layer DiglacianGCN is therefore $O(N \log N + L N d^2)$. Further, the time complexity of computing the hitting matrix by the fundamental matrix based on eq. (10) is $O(N)$. The time complexity of row-wise sparsification of $\mathcal{C}$ is $O(N \log N)$. In total, the DiglacianGCN-CT has the same time complexity as DiglacianGCN.
4 EXPERIMENTS

4.1 Datasets

We conduct experiments on seven disassortative graph datasets and three assortative graph datasets, which are widely used in previous work. Specifically, the disassortative datasets including three web page graphs from the WebKB dataset [26] (Texas, Wisconsin and Cornell), three webpage graphs from Wikipedia [26] (Actor, Chameleon and Squirrel) and a social network of European Deezer [23] (deeezer). On the other hand, the assortative datasets including two citation networks (CoraML [4] and Citeseer [10]) and a coauthor network (CoauthorCS [29]). In order to distinguish assortative and disassortative graph datasets, Zhu et al. [39] propose the edge homophily ratio as a metric to measure the homophily of a graph \( h = \frac{|\{(v_i, v_j) \in E | y_i = y_j\}|}{|E|} \), where \( y_i \) is the label of \( v_i \). This metric is defined as the proportion of edges that connect two nodes of the same class. The datasets that we used have edge homophily ratio ranging from low to high.

For all disassortative datasets except deezer, we use the feature vectors, class labels, and 10 fixed splits (48%/32%/20% of nodes per class for train/validation/test) from [26]. For deezer, we use 5 fixed splits (50%/25%/25% for train/validation/test) provided by [23]. For all assortative datasets, we use the same split as [31], i.e., 20 labels per class for the training set, 500 labels for validation set and the rest for test set. The detailed information and statistics of these datasets are shown in Table 2.

Table 2: Statistics of the ten datasets used in our experiments.

| Dataset       | \( N \) | \# \( E \) | # Feat. | # Classes | Digraph |
|---------------|---------|-----------|--------|----------|---------|
| Texas         | 183     | 309       | 1,703  | 5        | ✓       |
| Wisconsin     | 251     | 499       | 1,703  | 5        | ✓       |
| Cornell       | 183     | 295       | 1,703  | 5        | ✓       |
| Chameleon     | 2,277   | 36,101    | 2,325  | 5        | ✓       |
| Squirrel      | 5,201   | 217,073   | 2,089  | 5        | ✓       |
| Actor         | 7,600   | 33,544    | 931    | 5        | ✓       |
| deezer        | 28,281  | 92,752    | 31,241 |          |         |
| Cora-ML       | 2,995   | 8,416     | 2,879  | 7        | ✓       |
| Citeseer      | 3,312   | 4,715     | 3,703  | 6        | ✓       |
| CoauthorCS    | 18,333  | 81,894    | 6,805  | 15       |         |

4.2 Baselines

We compare our DiglacianGCN and DiglacianGCN-CT against 19 benchmark methods. The full list of methods are:

- Structure-independent: 2-layer MLP.
- General GNNs: GCN [17], ChebyNet [8], GAT [33], GraphSAGE (Mean aggregation) [14], APPNP [18], and jumping knowledge networks (GCN-JK, GCN+r) [35], where GCN+r only integrates knowledge from the input features.
- Digraph GNNs: DGCN [32], DiGCN and DiGCN-IB [31].
- Non-homophilous GNNs: MixHop [1], H2GCN-1/2 [39], FANGCN [3], CPRGCN [38], GPRGNN [6], and GCNN [24].

Besides, to evaluate the effectiveness of our proposed graph propagation matrices \( \mathcal{F} \) in DiglacianGCN and \( \mathcal{G} \) in DiglacianGCN-CT, we propose a variant of GraphSAGE, i.e., AdaGraphSAGE, as an additional baseline:

\[
H^{(l)} = \sigma(H^{(l-1)}W_0^{(l)} + (\alpha D^{-1} A u + \beta D^{-1} A)H^{(l-1)}W_1^{(l)}). \tag{14}
\]

For all baselines, we report their performance based on their official implementations after careful hyperparameter tuning.

4.3 Experimental Setting

For undirected graph datasets (deeezer and CoauthorCS), i.e., all edges are bidirectional in the provided raw data, we directly use the original adjacency matrix in all baselines. In the experiments of digraph datasets, for ChebyNet as a spectral method, we use the symmetrized adjacency matrix. For other baselines, we apply both the symmetrized and asymmetric adjacency matrix for node classification. The results reported are the better of the two results. Note that GCN is a spectral method, but it can be interpreted from the spatial perspective, i.e., outgoing neighbor aggregation with specific weights \( \frac{1}{\sqrt{d_i}} \). Hence, we view GCN as a spatial method. Other implementation details including running environment, hyperparameter settings and corresponding search space are presented in Appendix C.

4.4 Node Classification

Table 3 reports the results of node classification on all ten graph datasets. We can see that our method DiglacianGCN and DiglacianGCN-CT achieve new state-of-the-art results on 8 out of 10 datasets, and comparable results on two other datasets. Specifically, on datasets with strong heterophily (\( h < 0.5 \)), DiglacianGCN-CT achieves the best results on 5 out of 6 datasets and DiglacianGCN achieves the best result on 1 dataset and second-best results on 4 datasets. In particular, on Texas, Squirrel, Chameleon and Cornell, we achieve 87.84%, 54.22%, 71.33% and 87.30% accuracies respectively, which are 2.98%, 3.79%, 3.37% and 5.14% relative improvements over previous state-of-the-art. This demonstrates that considering the direction of edges can help the target nodes aggregate useful information, and commute time is an effective weighting scheme to help preserve long-distance dependency between nodes. On the dataset with intermediate heterophily (\( h \approx 0.5 \)), our methods are competitive. Specifically, on deezer, \( H_2 \) GCN and APPNP achieve the top two results, and DiglacianGCN-CT is the third. Our methods still have strong adaptability to homophilous graphs (\( h > 0.7 \)). Specifically, DiglacianGCN achieves the best results on Citeseer and CoauthorCS, and the second-best result on CoraML. DiglacianGCN-CT is the second-best on CoauthorCS and the third on CoraML. In addition, AdaGraphSAGE outperforms GraphSAGE in almost all datasets, that demonstrates adaptively learning the directionality of the graph is useful.

4.5 Adversarial Defense

Adversarial attacks on graphs aim to introduce minor modifications to the graph structure or node features that lead to a significant drop in the performance of GNNs. Recall that our methods provide two strategies, which are considering direction structure of the graph and underlying long-distance correlations between nodes, to greatly improve the performance on node classification tasks when graph structure is not reliable. This motivates us to examine
Table 3: Classification accuracy (with standard deviation) in percent. The "*" represents the best results among all variants of the model. For all baselines except MLP and ChebyNet, we respectively conduct experiments on symmetrized and original adjacency matrices and report the better results. Bold : best; Underline: runner-up.

| Graph | Texas | Wisconsin | Actor | Squirrel | Chameleon | Cornell | dezer | Citepeer | CoraML | CoauthorCS |
|-------|-------|----------|-------|----------|-----------|---------|-------|----------|--------|------------|
| MixHop | 77.84±(7.73) | 75.88±(5.49) | 32.22±(2.34) | 43.80±(1.48) | 60.50±(2.53) | 73.51±(6.34) | 66.80±(0.58) | 65.09±(2.08) | 65.89±(1.50) | 89.97±(0.04) |
| FAGCN | 82.43±(6.89) | 82.93±(7.95) | 34.87±(1.25) | 42.50±(0.79) | 55.22±(3.19) | 79.19±(9.79) | 65.88±(0.31) | 68.29±(1.17) | 84.00±(0.05) | 91.07±(1.28) |
| HLG-CN-1 | 84.86±(6.77) | 86.67±(6.49) | 35.28±(0.03) | 36.42±(1.89) | 57.11±(1.58) | 82.16±(4.80) | 67.49±(1.18) | 64.75±(0.26) | 80.66±(0.97) | 84.48±(0.97) |
| HLG-CN-2 | 82.16±(2.28) | 85.86±(2.22) | 55.65±(1.30) | 37.96±(2.02) | 59.96±(0.98) | 82.16±(6.00) | 65.04±(0.73) | 67.15±(9.99) | 78.39±(2.29) | 88.53±(0.38) |
| CPGNN* | 82.63±(6.88) | 84.58±(2.72) | 35.80±(0.92) | 29.25±(4.17) | 65.17±(3.17) | 79.93±(6.12) | 58.26±(0.71) | 66.19±(1.74) | 81.02±(0.77) | 89.20±(0.91) |
| FGPRGN | 84.43±(6.30) | 83.73±(0.02) | 34.99±(0.45) | 50.56±(1.51) | 66.31±(0.25) | 79.27±(0.03) | 66.90±(0.50) | 67.14±(1.87) | 73.31±(1.37) | 91.49±(0.39) |
| GCNII | 77.57±(3.83) | 80.39±(4.09) | 34.52±(1.23) | 38.47±(1.58) | 63.86±(3.04) | 77.86±(3.79) | 66.08±(1.93) | 58.32±(1.93) | 64.72±(2.85) | 84.13±(1.91) |
| DGCN | 71.58±(7.22) | 65.52±(4.71) | 37.94±(0.25) | 37.16±(1.72) | 50.77±(3.31) | 68.32±(4.30) | 62.11±(2.14) | 66.37±(1.93) | 78.36±(1.41) | 88.41±(0.68) |
| DGCN̂ | 65.18±(6.09) | 66.08±(3.82) | 34.25±(0.78) | 34.76±(1.24) | 50.35±(3.38) | 67.76±(6.11) | 58.37±(0.89) | 63.77±(2.27) | 79.51±(1.34) | OOM |
| DGCN̂-RR | 66.97±(3.72) | 64.16±(0.01) | 30.32±(0.68) | 33.44±(2.07) | 50.37±(3.31) | 60.03±(1.33) | 55.94±(2.88) | 64.99±(1.72) | 81.07±(1.14) | 91.90±(3.32) |
| DiglacianGCN | 87.84±(3.25) | 86.86±(4.21) | 35.37±(1.10) | 54.58±(3.30) | 70.70±(2.07) | 83.41±(5.36) | 66.52±(0.17) | 68.97±(1.38) | 82.36±(0.81) | 91.87±(0.73) |
| DiglacianGCN-CT | 87.50±(3.66) | 87.96±(4.03) | 36.10±(1.02) | 56.22±(2.09) | 71.33±(1.48) | 87.50±(7.18) | 66.90±(5.56) | 67.85±(1.66) | 81.52±(0.64) | 91.60±(3.34) |

the potential benefit of our models on adversarial defense. In this paper, we focus on perturbing the structure by adding or deleting edges, and evaluating the robustness of our methods on the node classification task. Specifically, we use metattack [40] to perform non-targeted attack, and follow the same experimental setting as [15], i.e., the ratio of changed edges, from 0 to 25% with a step of 5%. We use GCN, GAT, APPNP and FAGCN as baselines and use the default hyperparameter settings in the authors’ implementations. The hyperparameter of our methods are the same with Section 4.4. We conduct the experiments on CoraML, Citepeer and Chameleon and report results in Figure 2. From Figure 2a and Figure 2b, we can observe that all methods have similar downward trends. On Citepeer, APPNP and our DiglacianGCN achieve the best defensive effect under 5%~15% perturbation rate. FAGCN and APPNP achieve the best results under 20% and 25% perturbation rates respectively. On CoraML, DiglacianGCN achieves the best results under 10%~25% perturbation rate, and shows comparable performance to FAGCN under 5% and 25%. We can find that the class labels of CoraML are highly related to node feature according to the results of MLP in Table 3, that is the reason that Feature-Aware PageRank can help improve the robustness of our methods. From Figure 2c, we can find that DiglacianGCN-CT has more than 10% improvement over DiglacianGCN and other baselines under 15% perturbation rate, which demonstrates that the strategy of considering node proximity based on mixture graphs can boost model robustness on heterogeneous graphs.

4.6 Component Analysis

**Directed vs. Undirected.** To verify the importance of considering the direction structure, we compare values of the learnable parameters $\alpha$ and $\beta$ in DiglacianGCN and DiglacianGCN-CT when achieving the best average accuracy on the validation set. It can reflect the tendency of directionality and the influence of direction structure on the classification accuracy on the testing set. As shown in Figure 3, we observe that the learned $\alpha$ and $\beta$ vary across different datasets, and in most cases our proposed graph propagation matrices $\hat{T}$ and $\hat{G}$ have greater contributions to the final representations than that based on the symmetrized adjacency matrix. In addition, the results of AdaGraphSAGE and GraphSAGE in Table 3 indicate that instead of treating the given graph as a directed or undirected graph before training, adaptively learning the "soft" directionality of the graph can improve the performance of models.

**Feature-Aware PageRank vs. PageRank vs. kNN.** In our models, we propose to construct a combinatorial graph $G$ based on Feature-Aware PageRank to guarantee the irreducibility and aperiodicity, so that the Diglacian can be defined. We directly use PageRank transition matrix $P_{pr} = e^T + \left(1 - \alpha\right) \frac{e^T}{N}$ with self-loop for the same purpose, which yields a dense and feature-independent propagation matrix. Correspondingly, we replace $P_{pr}$, $\Phi$ in eq. (6) to $P_{pr}$ and its stationary distribution, and $\hat{D}$ is replaced with the out-degree matrix of the original graph. We use "(w/o feat.)" to represent this variant. This variant can help us verify the effectiveness of considering underlying correlations on the feature level. In addition, our methods are based on the preprocessed $G$, where each node has additional edges to link to several non-neighbor nodes with similar features. The measure of feature similarity is based on the proposed approximate similarity sorting. Although it is not completely accurate, we can apply it to construct an irreducible graph efficiently. Hence, we compare it with accurate $k$NN. Since the $k$NN graph is not necessarily irreducible, we first construct a...
null
A SPECTRAL CONVOLUTION OF THE DIGLACIAN

Since \( P_{pfpr} \in \mathbb{R}^{N \times N} \) is a transition matrix of the combinatorial graph \( G \), following the inherent property of Markov chain [27], the eigenvalues \( \lambda_i \) are bounded in \((-1,1]\). We first analyze the eigenvalues of \( T \). For clarity, we rewrite \( T \) in eq. (5) as:

\[
T = \tilde{\Sigma}^{-1} - \frac{1}{2} \left( \tilde{\Pi}^T P_{pfpr} \tilde{\Pi}^{-\frac{1}{2}} + \tilde{\Pi}^{-\frac{1}{2}} P_{pfpr}^T \tilde{\Pi}^2 \right). \tag{15}
\]

Let \( P_{pfpr} = \tilde{\Pi}^2 P_{pfpr} \tilde{\Pi}^{-\frac{1}{2}} \), then \( P_{pfpr} = \tilde{\Pi}^{-\frac{1}{2}} P_{pfpr} \tilde{\Pi}^2 \). Let \( \lambda \) denote a eigenvalue of \( P_{pfpr} \) with the eigenvector \( x \), i.e., \( P_{pfpr} x = \lambda x \). Then, \( \tilde{\Pi}^{-\frac{1}{2}} P_{pfpr} \tilde{\Pi}^2 x = \lambda x \), and \( P_{pfpr} \tilde{\Pi}^2 x = \lambda \tilde{\Pi}^\frac{1}{2} x \). Therefore, eigenvalues of \( P_{pfpr} \) are the same with \( P_{pfpr} \) ranging in \((-1,1]\), with eigenvectors \( \tilde{\Pi}^\frac{1}{2} x \). Thus, the eigenvalues \( \lambda_i \) of \( T \) is lower bounded by \( \frac{1}{d_{\text{max}}} - 1 \) and upper bounded by \( \frac{1}{d_{\text{min}}} + 1 \).

Since \( T \) is a real symmetric matrix, it has a complete set of orthonormal eigenvectors \( U = (u_1, u_2, \ldots, u_N) \). Let \( A \) be a diagonal matrix of eigenvalues with \( \lambda_{kk} = \lambda_k \). Since \( U \) is unitary, taking the eigenvectors of \( T \) as a set of basis, graph Fourier transform of a signal \( f \in \mathbb{R}^N \) on graph \( G \) is defined as \( \hat{f} = U^T f \), so the inverse graph Fourier transform is:

\[
f = U^T \hat{f} = \sum_{k=1}^{N} \hat{f}(k) u_k. \tag{16}
\]

According to convolution theorem, convolution in Euclidean space corresponds to pointwise multiplication in the Fourier domain. Denoting with \( \hat{g} \) the convolution kernel, the convolution of \( f \) with the filter \( \hat{g} \) in the Fourier domain can be defined by:

\[
f \ast g = U \left( (U^T \hat{g}) \odot (U^T \hat{f}) \right), \tag{17}
\]

where \( \odot \) is the element-wise Hadamard product, and \( \ast \) is the graph convolution operator. Since the filter \( \hat{g} \) is free, the vector \( U^T \hat{g} \) can be any vector. Thus, the filter can be replaced by a diagonal matrix \( \Sigma = \text{diag}(\theta) \) parameterized by \( \theta \in \mathbb{R}^N \). We can rewrite eq. (17) as \( U \Sigma U^T f \).

To reduce the number of trainable parameters to prevent overfitting and avoid explicit diagonalization of the matrix \( T \), following ChebyNet [8], which restricts spectral convolution kernel \( \Sigma \) to a polynomial expansion of \( \Lambda \) and approximates the graph spectral convolutions by a truncated expansion in terms of Chebyshev polynomials up to \( K \)-th order, we define a normalized eigenvalue matrix, with entries in \((-1,1]\), by \( \Lambda = \frac{2}{d_{\text{max}}} - 1 \), which can be verified by \( \frac{1}{d_{\text{max}}} - 1 \leq \lambda_i \leq \frac{1}{d_{\text{min}}} + 1 \). Then the graph convolution can be approximated by

\[
U \Sigma U^T f \approx \sum_{k=0}^{K} \theta_k T_k \left( \frac{T - \tilde{\Sigma}^{-1}}{2} \right) f, \tag{18}
\]

where \( T_k(\cdot) \) is the \( k \)-th order matrix Chebyshev polynomial defined by \( T_0(x) = 1, T_1(x) = x \), and \( T_k(x) = 2xT_{k-1}(x) + T_{k-2}(x) \) for \( k \geq 2 \), \( \theta_0, \theta_1, \ldots, \theta_K \) real-valued parameters. Let \( Y = U \Sigma U^T \), we employ an affine approximation \((K=1)\) with coefficients \( \theta_0 = \theta \) and \( \theta_1 = 2\theta \), from which we attain the graph convolution operation:

\[
Y f \approx \theta \left( I + \frac{1}{2} \left( \tilde{\Pi}^\frac{1}{2} P_{pfpr} \tilde{\Pi}^{-\frac{1}{2}} + \tilde{\Pi}^{-\frac{1}{2}} P_{pfpr}^T \tilde{\Pi}^\frac{1}{2} \right) \right) f. \tag{19}
\]

In our final form, we replace identity matrix \( I \) in eq. (19) with \( \tilde{\Sigma}^{-1} \) as shown in eq. (6). The reason is that in eq. (19) the transition matrix is \( P_{pfpr} + I \), which is equivalent to giving each node a large probability to move to itself in a random walk on \( G \). It makes the model hard to capture enough information from neighbors. Hence, we assume that the probability of the node moving to itself is the same as moving to its neighbors in each step, and it can be satisfied by eq. (6).

B PROOF OF THEOREM 3.1

Proof. As \( P_{pfpr}^t J = T \), we have \( ZJ = 0_{NXT} \). Let \( \tilde{Z} = \tilde{\Pi}^{-\frac{1}{2}} \tilde{T} \tilde{\Pi}^{-\frac{1}{2}} = \tilde{\Pi}^{-\frac{1}{2}}(\tilde{T} - P_{pfpr}) \tilde{\Pi}^{-\frac{1}{2}}, J = \tilde{T}^{-1} \tilde{T}^{-1} \), and \( Z = \tilde{T}^{-1} Z \tilde{T}^{-1} \), and in eq. (8) we have:

\[
Z + J = (\tilde{Z} + J) (\tilde{T}^{-1} + \tilde{T}^{-1} + I) \tag{20}
\]

then multiplying from the right by \((\tilde{Z} + J) (\tilde{T}^{-1} + \tilde{T}^{-1} + I)\), we have:

\[
l = (\tilde{Z} + J) (\tilde{T}^{-1} + \tilde{T}^{-1} + I) (\tilde{Z} + J) \tag{21}
\]

Therefore, eq. (22) can be simplified to:

\[
\tilde{Z} + J = (\tilde{Z} + J) (\tilde{T}^{-1} + \tilde{T}^{-1} + I) \tag{23}
\]

Finally, recovering \( Z \) and \( \tilde{Z} \), which concludes the proof.

C IMPLEMENTATION DETAILS

Hardware infrastructures. The experiments are conducted on Linux servers installed with a NVIDIA Quadro RTX8000 GPU and ten Intel(R) Xeon(R) Silver 4210R CPUs.

Hyperparameter Specifications. All parameters of our DiglaciaGCN and DiglaciaGCN-CT are initialized with Glorot initialization [13], and trained using Adam optimizer [16] with learning rate \( lr \) selected from \([0.01, 0.005]\). The activation function \( \sigma \) is ReLU. The \( l_2 \) weight decay is selected from \([0, 5e-4, 8e-4, 1e-3]\), and the dropout rate [30] is selected from \([0.5, 0.6, 0.7]\). The hyperparameter \( k \) of Feature-aware PageRank is set to a non-zero even number, which selected from \([2, 4, 6]\). The number of iterations \( t \) of the power method to compute the stationary distribution \( \pi \) is set to 30 for all datasets. The dimension of hidden layers is selected from \([48, 64, 96]\) for all datasets and the number of layers is set to 2 for all datasets. For DiglaciaGCN-CT, we set \( \mu = 0.97 \) for all datasets to ensure the sparsity of the graph propagation matrix. In addition, we use an early stopping strategy on accuracies on the validation nodes, with a patience 500 epochs. All dataset-specific hyperparameter configurations are summarized in Table 5.
| Dataset   | $k$  | $lr$ | Weight decay | Hidden dimension | Dropout |
|-----------|------|------|--------------|------------------|---------|
| Texas     | 2    | 0.01 | 8e-4         | 48               | 0.6     |
| Wisconsin | 4    | 0.01 | 1e-3         | 96               | 0.7     |
| Actor     | 2    | 0.01 | 1e-3         | 64               | 0.5     |
| Squirrel  | 2    | 0.005| 5e-4         | 64               | 0.5     |
| Chameleon | 2    | 0.01 | 5e-4         | 64               | 0.5     |
| Cornell   | 6    | 0.01 | 1e-3         | 64               | 0.5     |
| deezer    | 2    | 0.01 | 5e-4         | 64               | 0.5     |
| Citeeseer | 4    | 0.01 | 8e-4         | 64               | 0.5     |
| CoraML    | 6    | 0.01 | 5e-4         | 96               | 0.5     |
| CoauthorCS| 2    | 0.01 | 5e-4         | 64               | 0.5     |