# Message-Passing on Directed Acyclic Graphs Prevents Over-Smoothing

Andreas Roth[1](✉), Franka Bause[2,3], Nils M. Kriege[2,4], and Thomas Liebig[1]

[1] TU Dortmund University, 44227 Dortmund, Germany
{andreas.roth,thomas.liebig}@tu-dortmund.de
[2] Faculty of Computer Science, University of Vienna, Vienna, Austria
[3] UniVie Doctoral School Computer Science, University of Vienna, Vienna, Austria
[4] Research Network Data Science, University of Vienna, Vienna, Austria
{franka.bause,nils.kriege}@univie.ac.at

**Abstract.** Despite the rising popularity of message-passing neural networks (MPNNs), their ability to fit complex functions over graphs is limited as node representations become more similar with increasing depth—a phenomenon known as over-smoothing. Most approaches to mitigate over-smoothing extend common message-passing schemes, e.g., the graph convolutional network, by utilizing residual connections, gating mechanisms, normalization, or regularization techniques. Our work contrarily proposes to operate MPNNs on multiple computational graphs. We show that operating on a graph with no ergodic components, i.e., a directed acyclic graph (DAG), prevents over-smoothing. Each DAG amplifies a different signal in the data, allowing their combination to amplify multiple signals simultaneously and prevent representational rank collapse. Based on these insights, we propose DA-MPNNs, a general framework that splits any given graph into three computational graphs based on a strict partial order of the nodes. We conduct comprehensive experiments that confirm the computational benefits of DA-MPNNs, leading to further improvements of state-of-the-art MPNNs.

**Keywords:** graph neural networks · message-passing neural networks · representation learning.

## 1 Introduction

Many challenging tasks, such as drug discovery [25] and predictions for social networks [14], involve graph-structured data. Message-passing neural networks (MPNNs) [20] have found success in many of these areas. However, MPNNs did not see the same level of improvements against classical methods, such as graph kernels [28], that were achieved for computer vision [24] and natural language processing tasks [56]. MPNNs struggle to achieve satisfying performance for challenging tasks, such as large-scale heterophilic node classification. Computational issues like over-squashing [2] and over-smoothing [30] limit the ability of MPNNs to fit complex continuous functions. Recent works identified structural properties of given graphs, such as bottlenecks [2], to amplify computational challenges.

Methods such as graph rewiring [3] propose to operate on a modified computational graph that is less prone to over-squashing. However, it is less clear which structural properties lead to over-smoothing and how beneficial computational graphs for this phenomenon can be constructed.

Our work identifies the ergodicity of a graph as the structural reason behind over-smoothing. We show that operating on a graph with no ergodic components, a directed acyclic graph (DAG), cannot lead to over-smoothing. Importantly, each DAG amplifies a unique signal in the data depending on its structure, while all ergodic graphs amplify the same smooth signal. When operating on two DAGs as two computational graphs, multiple signals can be amplified simultaneously, preventing representational rank collapse. To transform any given graph into DAGs for these computational benefits, we propose to split the given edges into three computational graphs using a strict partial ordering of the nodes. This leads to DA-MPNNs, for which messages are processed on each computational graph using distinct feature transformations and then aggregated into a joint state. This framework can be directly combined with any MPNN. In our experiments, we evaluate several choices for the partial order and identify the node degree as a powerful general choice. We empirically confirm that DA-MPNN cannot over-smooth and that its ability to amplify multiple signals in the data benefits the learning process for several methods. When the DA framework is combined with Dir-GNN, a state-of-the-art method for node classification on heterophilic directed graphs, we further improve their results on all five considered datasets. We summarize our main contributions as follows:

- We show that the ergodicity of a graph leads to over-smoothing in MPNNs, and that graphs without ergodic components do not over-smooth. Our theory further shows the benefits of operating on multiple DAGs as this also avoids representational rank collapse (Section 4).
- We propose DA-MPNNs, a framework that splits any graph into three computational graphs based on a strict partial ordering of the nodes. This framework can be applied to any MPNN by constructing different messages depending on the order between nodes (Section 5).
- Our experiments confirm our theory by demonstrating that DA-MPNNs cannot over-smooth and show improved optimization properties (Section 6).

## 2 Preliminaries

Let $G = (\mathcal{V}, \mathcal{E})$ be a graph, where $V = \{v_1, \dots, v_n\}$ is its set of $n$ nodes and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ its set of edges. We refer to $\mathbf{A} \in \mathbb{R}^{n \times n}$ as the adjacency matrix for which $\mathbf{A}_{ij} = 1$ if $(i, j) \in \mathcal{E}$, otherwise the entry is 0. The nodes with an edge ending at $v_i$ are defined as its neighbors $N_i = \{k \mid (k, i) \in \mathcal{E}\}$. Based on each node's incoming edges, we define the degree matrix $\mathbf{D} \in \mathbb{R}^{n \times n}$ as a diagonal matrix with $d_{ii} = |N_i|$. A graph is strongly connected if a path exists between each pair of nodes. It is aperiodic if the lengths of its cycles do not have a common divisor $> 1$. A graph that is strongly connected and aperiodic is ergodic. We

utilize directed acyclic graphs (DAGs) as a special graph type that do not contain any ergodic parts, which we formally define as follows:

**Definition 1.** *(DAG) A graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is called a directed acyclic graph if there exists a strict partial order $\prec$ on $\mathcal{V}$ such that for each edge $(i, j) \in \mathcal{E} \Rightarrow i \prec j$.*

**Message-Passing Neural Networks** Given a graph $G$ and $d$-dimensional features $\mathbf{X} \in \mathbb{R}^{n \times d}$ for each node, message-passing neural networks (MPNNs) aim to obtain expressive node representations capturing both structural properties and connections between node features. Most MPNNs follow an iterative message-passing scheme that updates each node's representations

$$\mathbf{x}_i^{(k+1)} = \phi^{(k)} \left( \mathbf{x}_i^{(k)}, \bigoplus_{j \in N_i} \psi^{(k)} \left( \mathbf{x}_i^{(k)}, \mathbf{x}_j^{(k)} \right) \right) \tag{1}$$

using a message function $\psi^{(k)}$, a permutation invariant aggregation function $\oplus$, and a combine function $\phi^{(k)}$ for each iteration $k$. State-of-the-art MPNNs add additional components such as residual connections [7,46], restart terms [7,43], or gating mechanisms [32,47]. However, for exchanging messages, most methods follow a simple scheme that can be expressed in matrix notation

$$\bigoplus_{j \in N_i} \psi^{(k)} \left( \mathbf{x}_i^{(k)}, \mathbf{x}_j^{(k)} \right) = \left[ \tilde{\mathbf{A}}^{(k)} \mathbf{X}^{(k)} \mathbf{W}^{(k)} \right]_i \tag{2}$$

where $\tilde{\mathbf{A}} \in \mathbb{R}^{n \times n}$ corresponds to the aggregation function. $\tilde{\mathbf{A}}$ may be the symmetrically normalized adjacency matrix, the mean aggregation, the sum aggregation, or contain attention coefficients. It may also include self-loops. Most models apply a linear feature transformation $\mathbf{W}^{(k)} \in \mathbb{R}^{d_k \times d_{k+1}}$.

**Over-Smoothing and Rank Collapse** For many MPNNs, node representations tend to become more similar as more update iterations are performed, limiting the learnable functions over graphs. This results from Equation (2) amplifying a single signal in the data, corresponding to the dominant eigenvector of $\tilde{\mathbf{A}}$ [30,6,21,44]. In the limit, this signal dominates the representations, and all features become linearly dependent, i.e., resulting in a rank one matrix. This phenomenon is referred to as representational rank collapse [44]. As a special case of rank collapse, over-smoothing occurs when the dominant eigenvector of $\tilde{\mathbf{A}}$ is a smooth vector, resulting in representations becoming dominated by that smooth signal [30,39]. In particular, averaging aggregation matrices have the constant state $\mathbf{1}$ as their dominant eigenvector, and the symmetrically normalized adjacency matrix $\mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$ has $\mathbf{D}^{1/2}\mathbf{1}$ as its dominant eigenvector [36]. To quantify the degree of over-smoothing, the Dirichlet energy

$$E\left(\frac{\mathbf{X}}{||\mathbf{X}||_F}\right) = \operatorname{tr}\left(\frac{\mathbf{X}}{||\mathbf{X}||_F}^T \boldsymbol{\Delta} \frac{\mathbf{X}}{||\mathbf{X}||_F}\right) = \frac{1}{2} \sum_{(i,j) \in E} \left\| \frac{\mathbf{x}_i}{||\mathbf{X}||_F} - \frac{\mathbf{x}_j}{||\mathbf{X}||_F} \right\|_2^2 \tag{3}$$

is typically employed, for which convergence to 0 indicates over-smoothing [6]. Depending on the type of smoothing the model performs, the unnormalized graph Laplacian $\mathbf{\Delta} = \mathbf{D} - \mathbf{A}$ or the symmetrically normalized Laplacian $\mathbf{\Delta} = \mathbf{I} - \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$ is utilized.

## 3 Related Work

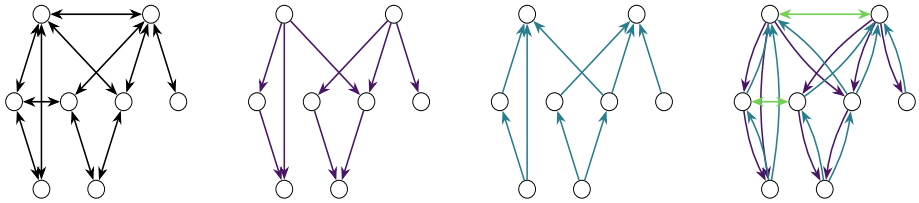
**Dealing with Over-Smoothing** Various methods to reduce over-smoothing have been proposed. These include combining the output of Equation (2) with previous states, e.g., by utilizing residual connections [5,7] or restart terms [19,7,43]. Gating mechanisms were proposed to stop updating node states after varying numbers of iterations [47,16]. Normalization operations that reduce the similarity between representations were introduced [62,29]. Another line of research proposes regularization terms to punish smooth representations during optimization [63]. Various methods propose aggregation functions that do not amplify smooth signals, including negative edge weights [4,59] and combining multiple aggregation functions [10,51]. However, all these methods are built on top of the base message-passing scheme (Equation (2)), which operates on the original graph and amplifies a single signal.

**Disconnecting the Computational Graph from the Input Graph** Typically, MPNNs operate directly on the input graph $G = (\mathcal{V}, \mathcal{E})$, which may cause computational issues like over-squashing [2] as information cannot flow over large distances. Sparsely connected regions, called bottlenecks [2], and related graph properties were identified as causes [55,11]. Several methods propose to perform the message-passing on a computational graph $G' = (\mathcal{V}, \mathcal{E}')$ that has better-suited structural properties, e.g., by graph rewiring techniques [1,3]. Dense connectivity, including higher node degrees [59] and a larger curvature [38] were identified to amplify over-smoothing. However, further structural properties leading to over-smoothing and how to construct a beneficial computational graph remain unclear. To the best of our knowledge, the benefits of operating on multiple computational graphs have not been studied.

**Operating on Directed Acyclic Graphs** The given graph may already satisfy specific properties. Related to our work, several methods for operating on trees [65,26] and on DAGs [48,60,52] were proposed. Datasets satisfying these properties are rare. These methods' main application areas are source code and neural architectures [52]. As the given graph is typically not a DAG, these methods are not broadly applicable. The connection between the computational benefits of these data structures has not been studied.

## 4 DAGs Solve Over-Smoothing and Rank Collapse

While theoretical studies on over-smoothing are available for both undirected and directed graphs, they assume ergodicity of the underlying graph, i.e., strongly



(a) Original graph. (b) DAG. (c) Reverse DAG. (d) DA-MPNNs.

Fig. 1: Different computational graphs that we consider.

connected and aperiodic (see Section 2). This assumption is related to random walks on ergodic graphs, averaging over incoming neighbors, converging to the uniform distribution [17]. We want to investigate how removing this assumption affects message-passing and over-smoothing. A graph that is not ergodic and does not contain ergodic subgraphs is a DAG (Definition 1). We refer to nodes with no incoming edges as root nodes and nodes with no outgoing edges as leaf nodes. Instead of nodes becoming more similar to each other, all signals get pushed away from the root nodes towards the leaf nodes. However, the signal is lost at the leaf nodes as they do not send any messages. Adding self-loops to all leaf nodes to maintain the signals in the graph provably prevents over-smoothing:

**Proposition 1.** *(DAGs do not over-smooth.) Let* $\mathbf{A} \in \mathbb{R}^{n \times n}$ *represent a DAG,* $\mathbf{A}_+$ *the DAG with added self-loops for all leaf nodes, and* $\tilde{\mathbf{A}}_+ = \mathbf{D}_+^{-1}\mathbf{A}_+$ *its normalized adjacency matrix. Further let* $\mathbf{X}^{(k+1)} = \tilde{\mathbf{A}}\mathbf{X}^{(k)}\mathbf{W}^{(k)}$ *for a.e.* $\mathbf{W}^{(0)} \in \mathbb{R}^{d_0 \times d_1}, \dots, \mathbf{W}^{(k)} \in \mathbb{R}^{d_k \times d_{k+1}}$*. Then, for all* $l > 0$ *and any* $\mathbf{X}^{(0)} \neq \mathbf{0} \in \mathbb{R}^{n \times d_0}$

$$E\left(\frac{\mathbf{X}^{(l)}}{||\mathbf{X}^{(l)}||_F}\right) \geq \frac{1}{n \cdot d_l}\,.$$

We provide the proof in Appendix A. Its benefit is still limited, as the representations of all non-leaf nodes become the zero vector. Similar to over-smoothing, where the smooth signal gets amplified in each iteration, here, only the signal corresponding to the particular direction gets amplified. Amplifying a single signal still leads to the more general rank collapse, which limits the performance of MPNNs [44].

We exploit the property that each DAG amplifies different signals based on their particular direction. This is contrary to any two ergodic graphs, for which random walks averaging over incoming nodes amplify the same smooth signal. Combining multiple DAGs with distinct partial orderings allows us to amplify multiple signals simultaneously. The DAG that amplifies the most different signal in the data is its reverse DAG, which we define as follows:

**Definition 2.** *(Reverse DAG) For a given DAG* $\mathcal{G} = (\mathcal{V}, \mathcal{E})$*, its reverse DAG is defined as* $\mathcal{G}' = (V, \mathcal{E}')$ *with* $\mathcal{E}' = \{(j, i) \mid (i, j) \in \mathcal{E}\}$.

When operating on these two computational graphs, we amplify multiple signals in the data simultaneously as long as we use different feature transforma-

tions. This provably prevents the rank collapse of feature representations and ensures that all representations are non-zero:

**Theorem 1.** *(Combining DAGs prevents rank collapse.) Let* $\mathbf{A}_1 \in \mathbb{R}^{n \times n}$ *represent a DAG and* $\mathbf{A}_2 \in \mathbb{R}^{n \times n}$ *represent its reverse DAG. We assume each node to have at least one incoming edge in either* $\mathbf{A}_1$ *or* $\mathbf{A}_2$*. The row-wise vector inequality is denoted as* $\neq_{rw}$*. Let* $\mathbf{X}^{(k+1)} = \mathbf{A}_1\mathbf{X}^{(k)}\mathbf{W}_1^{(k)} + \mathbf{A}_2\mathbf{X}^{(k)}\mathbf{W}_2^{(k)}$*. Then, for all* $l > 0$ *and almost every* $\mathbf{W}_1^{(k)}, \mathbf{W}_2^{(k)} \in \mathbb{R}^{d_k \times d_{k+1}}$ *and* $\mathbf{X}^{(0)} \neq_{rw} \mathbf{0} \in \mathbb{R}^{n \times d_k}$ *we have*

$$\mathbf{X}^{(l)} \neq_{rw} \mathbf{0} \wedge \operatorname{rank}(\mathbf{X}^{(l)}) > 1\,. \tag{4}$$

The proof shows the critical importance of employing different feature transformations for each computational graph. We avoid rank collapse as long as both $\mathbf{W}_1^{(k)}$ and $\mathbf{W}_2^{(k)}$ map a vector to two linearly independent vectors, which explains that the statement holds for almost every $\mathbf{W}_1^{(k)}, \mathbf{W}_2^{(k)}$ with respect to the Lebesgue measure. This even holds in the non-linear case, as long as this property holds for the employed activation functions, e.g., Leaky ReLU or Sigmoid:

*Remark 1.* Theorem 1 holds for any activation function $\phi$ that does not map two linearly independent vectors onto linearly dependent vectors, i.e., $\forall \mathbf{x}, \mathbf{y} \in \mathbb{R}^d, \forall a, b \in \mathbb{R}, \mathbf{x} \neq a \cdot \mathbf{y} \colon \phi(\mathbf{x}) \neq b \cdot \phi(\mathbf{y})$.

Our theory shows clear benefits of operating on DAGs as the computational graphs. This will be particularly valuable for tasks that require models that can capture complex dynamics, e.g., heterophilic graphs. However, for most tasks, the given graph is not a DAG. Thus, we will next propose a method to convert any graph into DAGs to benefit from the computational advantages.

## 5 DA-MPNNs

Given a graph $G = (\mathcal{V}, \mathcal{E})$, our goal is to transform it into a DAG $G_1 = (\mathcal{V}, \mathcal{E}_1)$ as our first computational graph. This is a well-known algorithmic challenge in graph theory, which is connected to topological sorting [9] and finding feedback arc sets [18]. In this work, we transform any graph into a DAG by defining a strict partial order $\prec$ on the nodes. Using only those edges $(i, j)$ for which $i \prec j$ results in a DAG, i.e., $\mathcal{E}_1 = \{(i, j) \in \mathcal{E} \mid i \prec j\}$. We define our second computational DAG using the converse relation, i.e., $\mathcal{E}_2 = \{(i, j) \in \mathcal{E} \mid j \prec i\}$. This corresponds to the reverse DAG for an undirected graph. As we defined a strict partial order, neither $i \prec j$ nor $j \prec i$ may be satisfied for some edges. These are not contained in $\mathcal{E}_1$ or $\mathcal{E}_2$. To not lose these edges for our message-passing, we propose a third computational graph $G_3 = (\mathcal{V}, \mathcal{E}_3)$ that contains all the remaining edges, i.e., $\mathcal{E}_3 = \mathcal{E} \setminus (\mathcal{E}_1 \cup \mathcal{E}_2)$. We provide an example for these computational graphs in Figure 1. As this third computational graph amplifies a signal different to $G_1$ and $G_2$, potentially a smooth signal, it is mathematically beneficial to also operate

on $G_3$. We perform the message construction part separately for each edge based on their relation

$$f(i,j) = \begin{cases} 1, & \text{if } i \prec j \\ 2, & \text{if } j \prec i \\ 3, & \text{otherwise} \end{cases} \tag{5}$$

within the strict partial order. The general form of our framework for DA-MPNNs

$$\mathbf{x}_i^{(k+1)} = \left[\text{DA-MPNN}\left(\mathbf{X}^{(k)}, \mathcal{E}\right)\right]_i = \phi^{(k)}\left(\mathbf{x}_i^{(k)}, \bigoplus_{j \in N_i} \psi_{f(i,j)}^{(k)}\left(\mathbf{x}_i^{(k)}, \mathbf{x}_j^{(k)}\right)\right) \tag{6}$$

is similar to the general form of MPNNs, with $\psi_1^{(k)}, \psi_2^{(k)}, \psi_3^{(k)}$ being functions generating different messages per computational graph, $\bigoplus$ a permutation invariant aggregation operator, and $\phi^{(k)}$ a combine function. As examples, we introduce two common models and a state-of-the-art method in the DA framework.

**DA-GCN** We restate the GCN [27] in our framework, as it is commonly used and often serves as the message-passing component within complex models. We define DA-GCN as

$$\begin{aligned} \left[\text{DA-GCN}\left(\mathbf{X}^{(k)}, \mathcal{E}\right)\right]_i &= \left[\tilde{\mathbf{A}}_1 \mathbf{X}^{(k)} \mathbf{W}_1^{(k)} + \tilde{\mathbf{A}}_2 \mathbf{X}^{(k)} \mathbf{W}_2^{(k)} + \tilde{\mathbf{A}}_3 \mathbf{X}^{(k)} \mathbf{W}_3^{(k)}\right]_i \\ &= \sum_{j \in N_i} \frac{1}{\sqrt{d_i}\sqrt{d_j}} \mathbf{W}_{f(i,j)}^{(k)} \mathbf{x}_j^{(k)}, \end{aligned} \tag{7}$$

where $\mathbf{W}_1^{(k)}, \mathbf{W}_2^{(k)}, \mathbf{W}_3^{(k)} \in \mathbb{R}^{d_k \times d_{k+1}}$ are feature transformations and $\tilde{\mathbf{A}}_1, \tilde{\mathbf{A}}_2, \tilde{\mathbf{A}}_3 \in \mathbb{R}^{n \times n}$ contain the edge weights of the corresponding computational graph, i.e., $\tilde{\mathbf{A}}_1 + \tilde{\mathbf{A}}_2 + \tilde{\mathbf{A}}_3 = \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$.

**DA-SAGE** We define the DA-SAGE version of the SAGE convolution [23] as

$$\left[\text{DA-SAGE}\left(\mathbf{X}^{(k)}, \mathcal{E}\right)\right]_i = \mathbf{W}^{(k)} \mathbf{x}_i^{(k)} + \sum_{j \in N_i} \frac{1}{d_i} \mathbf{W}_{f(i,j)}^{(k)} \mathbf{x}_j^{(k)}, \tag{8}$$

where $\mathbf{W}^{(k)} \in \mathbb{R}^{d_k \times d_{k+1}}$ is the additional feature transformation of the previous state $\mathbf{x}_i^{(k)}$.

**State-of-the-Art Methods** State-of-the-art methods utilize complex architectures that may include residual connections, restart terms, normalization layers, or other improvements. However, most methods utilize a basic MPNN as part of their framework. Our approach is thus orthogonal to these methods. When applying DA-MPNNs to these methods, we replace the basic message-passing operation with its DA version. As an example, the current state-of-the-art MPNN

on directed graphs, Dir-GNN [42], utilizes both $\mathcal{E}$ and the transposed version $\mathcal{E}^T = \{(j,i) \mid (i,j) \in \mathcal{E}\}$. They employ two basic MPNNs, like the GCN, which we replace with its DA versions. This leads to our definition

$$\text{DA-Dir-GNN}\left(\mathbf{X}^{(k)}, \mathcal{E}\right) = \alpha \cdot \text{DA-MPNN}\left(\mathbf{X}^{(k)}, \mathcal{E}\right) + (1-\alpha) \cdot \text{DA-MPNN}\left(\mathbf{X}^{(k)}, \mathcal{E}^T\right) \tag{9}$$

where $\alpha \in [0,1]$ is a hyperparameter as proposed in [42]. Combining the DA framework with other methods is equivalent.

### 5.1 Bottleneck Transformations

Using the same dimensions for each $\mathbf{W}_1, \mathbf{W}_2, \mathbf{W}_3 \in \mathbb{R}^{d \times d'}$ in DA-MPNNs as for $\mathbf{W} \in \mathbb{R}^{d \times d'}$ in an MPNN, would cause the DA-MPNN to contain three times as many parameters. While reducing $d$ and $d'$ for the DA-MPNN would be straightforward, this similarly creates issues, as each message would consist of fewer features. Instead, we propose bottleneck transformations that are inspired by bottleneck convolutions in image processing [24]. We reduce the feature dimension using a shared encoding transformation $\mathbf{T} \in \mathbb{R}^{d \times s}$, after which distinct transformations $\mathbf{U}_1, \mathbf{U}_2, \mathbf{U}_3 \in \mathbb{R}^{s \times s}$ are applied in this low-dimensional space. The transformed states are then mapped back to the target dimension using a shared decoding transformation $\mathbf{V} \in \mathbb{R}^{s \times d'}$. The three transformations combined result in the proposed bottleneck transformations $\mathbf{W}_i = \mathbf{T}\mathbf{U}_i\mathbf{V} \in \mathbb{R}^{d \times d'}$. By matching the total number of parameters $d \cdot s + 3 \cdot s^2 + s \cdot d' = d \cdot d'$ and setting

$$s = \left\lfloor \frac{-d - d' + \sqrt{d^2 + 14 \cdot d \cdot d' + (d')^2}}{6} \right\rfloor, \tag{10}$$

we ensure that the feature dimension is unchanged and the total number of parameters of the DA-GCN and DA-SAGE is equal to their respective GCN and SAGE version or slightly lower due to rounding.

### 5.2 Computational Complexity

The main computational differences between a DA-MPNN and its MPNN version are computing the strict partial ordering and applying three transformations instead of one. However, for the bottleneck transformations and our choice of $s$ (Equation (10)), the number of multiplications and additions also matches or is slightly lower for the DA-MPNN. Thus, $\prec$ is the main additional computational cost of the DA framework, for which we will now discuss potential options.

### 5.3 Choosing a partial order

Many graph theoretical algorithms, such as graph traversal algorithms or centrality measures like the node degree, can provide a strict partial ordering for nodes. We construct the strict partial order based on a single real node value $r_i$

Table 1: Mean values and standard deviations over three runs on the ZINC dataset (best results marked in **bold**). Step times are in milliseconds (ms), and values for the mean absolute error (MAE) are multiplied by 100 for clarity. Train MAE is the overall minimum, test MAE is based on the best validation MAE.

| Metrics | Step Time (ms) | | MAE | |
|---|---|---|---|---|
| | Train | Val | Train | Test |
| GCN | $3.3 \pm 0.1$ | $2.1 \pm 0.1$ | $15.9 \pm 0.4$ | $20.1 \pm 0.3$ |
| DA-GCN (random) | $\mathbf{4.5} \pm 0.2$ | $\mathbf{2.6} \pm 0.1$ | $21.3 \pm 0.2$ | $21.5 \pm 0.4$ |
| DA-GCN (Features) | $4.6 \pm 0.1$ | $\mathbf{2.6} \pm 0.1$ | $\mathbf{12.2} \pm 0.3$ | $\mathbf{15.5} \pm 0.7$ |
| DA-GCN (PPR) | $5.7 \pm 0.2$ | $3.7 \pm 0.2$ | $16.3 \pm 0.1$ | $19.7 \pm 0.2$ |
| DA-GCN (Degree) | $4.7 \pm 0.2$ | $2.7 \pm 0.1$ | $12.7 \pm 0.2$ | $16.6 \pm 0.4$ |
| DA-GCN (Degree) \ Bottleneck | $4.1 \pm 0.1$ | $2.4 \pm 0.1$ | $10.9 \pm 0.1$ | $16.3 \pm 0.4$ |

Table 2: Train losses reported are the overall minimum. Test scores are based on the best validation score. The cross-entropy (CE), mean absolute error (MAE), and the average precision (AP) are reported. Mean and standard deviation are reported (pairwise best results marked in **bold**).

| Method | ZINC (MAE) | | Peptides-Func | | Peptides-Struct (MAE) | |
|---|---|---|---|---|---|---|
| | Train | Test | Train (CE) | Test (AP) | Train | Test |
| GCN | $15.9 \pm 0.4$ | $20.1 \pm 0.3$ | $3.4 \pm 0.5$ | $57.4 \pm 0.8$ | $14.6 \pm 3.2$ | $\mathbf{30.2} \pm 0.5$ |
| DA-GCN | $\mathbf{12.2} \pm 0.1$ | $\mathbf{16.6} \pm 0.2$ | $\mathbf{0.3} \pm 0.0$ | $\mathbf{59.5} \pm 0.3$ | $\mathbf{10.9} \pm 0.7$ | $30.4 \pm 0.8$ |
| SAGE | $7.9 \pm 0.2$ | $13.2 \pm 0.3$ | $1.3 \pm 1.5$ | $56.5 \pm 1.8$ | $16.8 \pm 0.4$ | $31.3 \pm 0.5$ |
| DA-SAGE | $\mathbf{5.7} \pm 0.0$ | $\mathbf{10.7} \pm 0.3$ | $\mathbf{0.3} \pm 0.0$ | $\mathbf{59.8} \pm 1.0$ | $\mathbf{7.2} \pm 0.6$ | $\mathbf{30.8} \pm 0.7$ |

per node $i$, i.e., $i \prec j \Leftrightarrow r_i < r_j$. The choice of partial order adds an inductive bias to the model and removes invariances of messages to their direction, i.e., removing isotropy [57]. The three computational graphs should benefit from different feature transformations for favorable optimization properties. Messages within each computational graph should be more similar to each other than to messages of other computational graphs. For example, when choosing the node degree as our partial ordering, messages from higher-degree nodes to lower-degree nodes should benefit from being transformed differently from their reverse messages. In molecular data, higher-degree nodes correspond to other atoms than lower-degree nodes [58], which may benefit from sending different messages. Finding the optimal ordering per task is a separate algorithmic challenge.

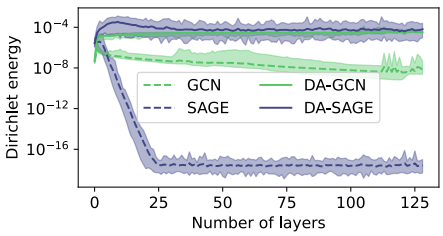



Fig. 2: Comparison of the Dirichlet energy when increasing the number of layers. Mean values over 50 random seeds in bold. Minimum and maximum values as semi-transparent areas.

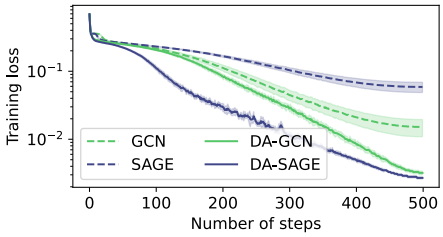



Fig. 3: Training loss (Cross-Entropy) during optimization on Peptides-Func for MPNNs with four layers. Mean values over five runs with standard deviations as semi-transparent areas.

## 6 Experiments

We now investigate the ability of DA-MPNNs to improve learning for complex tasks. We provide additional details in Appendix B and reproducible code for all experiments as supplementary material[5].

### 6.1 Improving the Learning Process

We consider three challenging tasks for molecular prediction, namely, Peptides-Func and Peptides-Struct from the long-range graph benchmark (LRGB) [12] and the ZINC dataset. The Peptides dataset consists of 15 535 graphs. For Peptides-Func, the task is a multi-label graph classification to predict ten molecular properties. For Peptides-Struct, the targets are five geometric properties in a multi-label graph regression setting. ZINC is a single-label graph regression task that consists of 249 456 graphs representing chemical compounds [50]. All considered models perform $k$ iterations of message-passing, with ReLU as a non-linear activation function, followed by a linear prediction layer. We do not use additional techniques like positional encodings, residual connections, normalization layers, or regularization to compare the differences between the message-passing operators independently. We reuse the standard optimization process from [13] using the AdamW optimizer [34]. See Appendix B for more details. We integrate our models into the implementation of [54].

**Evaluating Node Orderings** We compare different strategies for constructing an ordering using a single value per node. We consider random values, the sum of initial node features, Personalized PageRank (PPR) scores, and the node degree. The strategies are evaluated in terms of execution time per training and validation step, minimal training loss, and test loss corresponding to the best validation

[5] `https://anonymous.4open.science/r/da-mpnns-46E9/README.md`

performance. We tune the learning rate and number of layers for each experiment and repeat for three random seeds. All orderings are applied to DA-GCNs.

Results are presented in Table 1. Apart from the random ordering, all orderings improve the performance, with degree and feature-based orderings achieving the best test scores. The DA-GCNs are able to reduce the training loss, which leads to test improvements. Runtime is increased by around 35% for our implementation. DA-GCN performs slightly better without the bottleneck transformation. While this experiment shows potential for studying further orderings, we utilize the node degree as our ordering and bottleneck transformations for all other experiments due to their general applicability and better comparability.

**Preventing Over-Smoothing** To empirically validate that DA-MPNNs cannot cause over-smoothing, we apply MPNN layers and ReLU activations over 128 layers and track the Dirichlet energy after each iteration. We use the Peptides-Func dataset [13] as our initial representations and graph structure. We compare the GCN with DA-GCN and SAGE with DA-SAGE in Figure 2. The Dirichlet energy for the GCN and SAGE converges to zero, with SAGE reaching floating point imprecision after around 25 layers and GCN after around 110 layers. Contrarily, it remains constant across all 128 layers for the DA versions.

**Combination with other Methods** We investigate how DA-MPNNs can improve performance. We present training losses and achieved test scores in Table 2. The training loss is reduced in all cases, with the largest improvement being by more than 90%. As GCN can only amplify the smooth signal in the data, it severely underfits the task. As DA-MPNN amplifies multiple signals, it fits the target function better, which leads to improvements in test scores in most cases. To emphasize this, we display the training loss during optimization in Figure 3. DA-MPNNs visibly improve the optimization process. However, test scores are not improved in all cases. This indicates overfitting of DA-MPNNS to the training data, given their improved approximation abilities. Successfully applying DA-MPNNs thus requires sufficient amounts of data or strong regularization.

### 6.2 Comparison with State-of-the-Art

Improved optimization of the target function suggests that DA-MPNNs will be particularly valuable for challenging tasks. We consider large-scale heterophilic graphs, namely Squirrel and Chameleon [40], Arxiv-Year and Snap-Patents [33], and Roman-Empire [41]. Our implementation is based on Dir-GNN [42], a state-of-the-art method. We replace the MPNN modules of Dir-GNN with the corresponding DA-MPNN modules. Accordingly, DA-SAGE is used for Roman-Empire, while DA-GCN is used for the other datasets. Experiments for Squirrel, Chameleon, and Roman-Empire are repeated for ten fixed splits into training, validation, and test sets and for Arxiv-Year and Snap-Patents for five fixed splits. The baseline results are reused from our reference implementation [42]. Based on their hyperparameter values, we tune the learning rate, number of layers, and

Table 3: Mean accuracy and standard deviation for five directed benchmark graphs (best result marked in **bold** and second-best underlined).

| Method | Squirrel | Chameleon | Arxiv-year | Snap-patents | Roman-Empire |
|---|---|---|---|---|---|
| MLP | $28.77 \pm 1.56$ | $46.21 \pm 2.99$ | $36.70 \pm 0.21$ | $31.34 \pm 0.05$ | $64.94 \pm 0.62$ |
| GCN | $53.43 \pm 2.01$ | $64.82 \pm 2.24$ | $46.02 \pm 0.26$ | $51.02 \pm 0.06$ | $73.69 \pm 0.74$ |
| GPR-GNN | $54.35 \pm 0.87$ | $62.85 \pm 2.90$ | $45.07 \pm 0.21$ | $40.19 \pm 0.03$ | $64.85 \pm 0.27$ |
| LINKX | $61.81 \pm 1.80$ | $68.42 \pm 1.38$ | $56.00 \pm 0.17$ | $61.95 \pm 0.12$ | $37.55 \pm 0.36$ |
| ACM-GCN | $67.40 \pm 2.21$ | $74.76 \pm 2.20$ | $47.37 \pm 0.59$ | $55.14 \pm 0.16$ | $69.66 \pm 0.62$ |
| GloGNN | $57.88 \pm 1.76$ | $71.21 \pm 1.84$ | $54.79 \pm 0.25$ | $62.09 \pm 0.27$ | $59.63 \pm 0.69$ |
| Grad. Gating | $64.26 \pm 2.38$ | $71.40 \pm 2.38$ | $63.30 \pm 1.84$ | $69.50 \pm 0.39$ | $82.16 \pm 0.78$ |
| DiGCN | $37.74 \pm 1.54$ | $52.24 \pm 3.65$ | OOM | OOM | $52.71 \pm 0.32$ |
| MagNet | $39.01 \pm 1.93$ | $58.22 \pm 2.87$ | $60.29 \pm 0.27$ | OOM | $88.07 \pm 0.27$ |
| Dir-GNN | $\underline{75.31} \pm 1.92$ | $\underline{79.71} \pm 1.26$ | $\underline{64.08} \pm 0.30$ | $\underline{73.95} \pm 0.05$ | $\underline{91.23} \pm 0.32$ |
| DA-Dir-GNN | $\mathbf{75.49} \pm 1.70$ | $\mathbf{80.13} \pm 1.53$ | $\mathbf{65.70} \pm 0.23$ | $\mathbf{74.51} \pm 0.07$ | $\mathbf{91.46} \pm 0.53$ |

dropout ratio using a grid search. Best-performing hyperparameters are reused for five repetitions of each split, for which we report the average test scores. We compare to seven state-of-the-art methods for heterophilic graphs and three for directed graphs. Further details are provided in Appendix B. We present the test results in Table 3. DA-MPNNs slightly improve the performance for all five datasets, with more significant gains for the larger datasets, i.e., Arxiv-Year and Snap-Patents. As with our other experiments, we observe larger improvements in the training loss (see Figure 4). This confirms the benefits of DA-MPNNs for optimization, while generalization properties can be further improved.

## 7 Conclusion

In this work, we propose to operate MPNNs on multiple computational graphs. By removing all ergodic components from a graph, which results in a DAG, over-smoothing is prevented, and different signals are amplified depending on the direction. Operating on multiple DAGs amplifies multiple signals in the data simultaneously, which additionally prevents rank collapse. To utilize the computational benefits, we propose splitting any given graph into three computational graphs based on a strict partial order of the nodes. Messages are constructed for each computational graph and combined into a joint state. This framework, which we call DA-MPNN, can be applied to any MPNN. Our experiments confirm that DA-MPNNs cannot cause over-smoothing and can improve learning.

We anticipate further opportunities to study the benefits of multiple computational graphs, potentially operating on more than three graphs. Task-specific knowledge and invariances for graph splitting can provide additional benefits. Limitations of DA-MPNNs are worsened generalization capabilities, potentially different optimal partial orderings per task, and slower runtime.

**Acknowledgments.** Part of this research has been funded by the Federal Ministry of Education and Research of Germany and the state of North-Rhine Westphalia as part of the Lamarr-Institute for Machine Learning and Artificial Intelligence and by the Federal Ministry of Education and Research of Germany under grant no. 01IS22094E WEST-AI. This work was supported by the Vienna Science and Technology Fund (WWTF) [10.47379/VRG19009].

**Disclosure of Interests.** The authors have no competing interests to declare that are relevant to the content of this article.

## References

1. Abboud, R., Dimitrov, R., Ceylan, I.I.: Shortest path networks for graph property prediction. In: Learning on Graphs Conference. pp. 5–1. PMLR (2022)
2. Alon, U., Yahav, E.: On the bottleneck of graph neural networks and its practical implications. In: International Conference on Learning Representations (2021)
3. Barbero, F., Velingker, A., Saberi, A., Bronstein, M.M., Giovanni, F.D.: Locality-aware graph rewiring in GNNs. In: The Twelfth International Conference on Learning Representations (2024)
4. Bo, D., Wang, X., Shi, C., Shen, H.: Beyond low-frequency information in graph convolutional networks. In: Proceedings of the AAAI conference on artificial intelligence. pp. 3950–3957 (2021)
5. Bresson, X., Laurent, T.: Residual gated graph convnets. arXiv preprint arXiv:1711.07553 (2017)
6. Cai, C., Wang, Y.: A note on over-smoothing for graph neural networks. arXiv preprint arXiv:2006.13318 (2020)
7. Chen, M., Wei, Z., Huang, Z., Ding, B., Li, Y.: Simple and deep graph convolutional networks. In: International conference on machine learning. pp. 1725–1735. PMLR (2020)
8. Chien, E., Peng, J., Li, P., Milenkovic, O.: Adaptive universal generalized pagerank graph neural network. In: International Conference on Learning Representations (2021)
9. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to algorithms. MIT press (2022)
10. Corso, G., Cavalleri, L., Beaini, D., Liò, P., Veličković, P.: Principal neighbourhood aggregation for graph nets. Advances in Neural Information Processing Systems **33**, 13260–13271 (2020)
11. Di Giovanni, F., Giusti, L., Barbero, F., Luise, G., Lio, P., Bronstein, M.M.: On over-squashing in message passing neural networks: The impact of width, depth, and topology. In: International Conference on Machine Learning. pp. 7865–7885. PMLR (2023)
12. Dwivedi, V.P., Luu, A.T., Laurent, T., Bengio, Y., Bresson, X.: Graph neural networks with learnable structural and positional representations. In: International Conference on Learning Representations (2022)
13. Dwivedi, V.P., Rampášek, L., Galkin, M., Parviz, A., Wolf, G., Luu, A.T., Beaini, D.: Long range graph benchmark. In: Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (2022)
14. Fan, W., Ma, Y., Li, Q., He, Y., Zhao, E., Tang, J., Yin, D.: Graph neural networks for social recommendation. In: The world wide web conference. pp. 417–426 (2019)

15. Fey, M., Lenssen, J.E.: Fast graph representation learning with PyTorch Geometric. In: ICLR Workshop on Representation Learning on Graphs and Manifolds (2019)
16. Finkelshtein, B., Huang, X., Bronstein, M., Ceylan, İ.İ.: Cooperative graph neural networks. arXiv preprint arXiv:2310.01267 (2023)
17. Gallager, R.G.: Discrete stochastic processes. Journal of the Operational Research Society **48**(1), 103–103 (1997)
18. Garey, M.R., Johnson, D.S.: Computers and intractability, vol. 174. freeman San Francisco (1979)
19. Gasteiger, J., Bojchevski, A., Günnemann, S.: Combining neural networks with personalized pagerank for classification on graphs. In: International Conference on Learning Representations (2019)
20. Gilmer, J., Schoenholz, S.S., Riley, P.F., Vinyals, O., Dahl, G.E.: Neural message passing for quantum chemistry. In: International conference on machine learning. pp. 1263–1272. PMLR (2017)
21. Giovanni, F.D., Rowbottom, J., Chamberlain, B.P., Markovich, T., Bronstein, M.M.: Understanding convolution on graphs via energies. Transactions on Machine Learning Research (2023)
22. Hall, B.H., Jaffe, A.B., Trajtenberg, M.: The nber patent citation data file: Lessons, insights and methodological tools (2001)
23. Hamilton, W., Ying, Z., Leskovec, J.: Inductive representation learning on large graphs. Advances in neural information processing systems **30** (2017)
24. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 770–778 (2016)
25. Igashov, I., Stärk, H., Vignac, C., Schneuing, A., Satorras, V.G., Frossard, P., Welling, M., Bronstein, M., Correia, B.: Equivariant 3d-conditional diffusion model for molecular linker design. Nature Machine Intelligence pp. 1–11 (2024)
26. Kiperwasser, E., Goldberg, Y.: Easy-first dependency parsing with hierarchical tree lstms. Transactions of the Association for Computational Linguistics **4**, 445–461 (2016)
27. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. In: International Conference on Learning Representations (2016)
28. Kriege, N.M.: Weisfeiler and leman go walking: Random walk kernels revisited. In: Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., Oh, A. (eds.) Advances in Neural Information Processing Systems. vol. 35, pp. 20119–20132 (2022)
29. Li, G., Xiong, C., Thabet, A., Ghanem, B.: Deepergcn: All you need to train deeper gcns. arXiv preprint arXiv:2006.07739 (2020)
30. Li, Q., Han, Z., Wu, X.: Deeper insights into graph convolutional networks for semi-supervised learning. In: Proceedings of the AAAI conference on artificial intelligence (2018)
31. Li, X., Zhu, R., Cheng, Y., Shan, C., Luo, S., Li, D., Qian, W.: Finding global homophily in graph neural networks when meeting heterophily. In: International Conference on Machine Learning. pp. 13242–13256. PMLR (2022)
32. Li, Y., Tarlow, D., Brockschmidt, M., Zemel, R.: Gated graph sequence neural networks. arXiv preprint arXiv:1511.05493 (2015)
33. Lim, D., Hohne, F., Li, X., Huang, S.L., Gupta, V., Bhalerao, O., 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)

34. Loshchilov, I., Hutter, F.: Decoupled weight decay regularization. In: International Conference on Learning Representations (2019)
35. Luan, S., Hua, C., Lu, Q., Zhu, J., Zhao, M., Zhang, S., Chang, X.W., Precup, D.: Revisiting heterophily for graph neural networks. Advances in neural information processing systems **35**, 1362–1375 (2022)
36. Luxburg, U.: A tutorial on spectral clustering. Statistics and computing **17**(4), 395–416 (2007)
37. Maurya, S.K., Liu, X., Murata, T.: Improving graph neural networks with simple architecture design. arXiv preprint arXiv:2105.07634 (2021)
38. Nguyen, K., Hieu, N.M., Nguyen, V.D., Ho, N., Osher, S., Nguyen, T.M.: Revisiting over-smoothing and over-squashing using ollivier-ricci curvature. In: Proceedings of the 40th International Conference on Machine Learning. pp. 25956–25979 (2023)
39. Oono, K., Suzuki, T.: Graph neural networks exponentially lose expressive power for node classification. In: international conference on learning representations (ICLR) (2020)
40. Pei, H., Wei, B., Chang, K.C.C., Lei, Y., Yang, B.: Geom-gcn: Geometric graph convolutional networks. In: International Conference on Learning Representations (2020)
41. Platonov, O., Kuznedelev, D., Diskin, M., Babenko, A., Prokhorenkova, L.: A critical look at the evaluation of GNNs under heterophily: Are we really making progress? In: The Eleventh International Conference on Learning Representations (2023)
42. Rossi, E., Charpentier, B., Di Giovanni, F., Frasca, F., Günnemann, S., Bronstein, M.M.: Edge directionality improves learning on heterophilic graphs. In: The Second Learning on Graphs Conference (2023)
43. Roth, A., Liebig, T.: Transforming pagerank into an infinite-depth graph neural network. In: Joint European conference on machine learning and knowledge discovery in databases. pp. 469–484. Springer (2022)
44. Roth, A., Liebig, T.: Rank collapse causes over-smoothing and over-correlation in graph neural networks. In: The Second Learning on Graphs Conference (2023)
45. Rozemberczki, B., Allen, C., Sarkar, R.: Multi-scale attributed node embedding. Journal of Complex Networks **9**(2), cnab014 (2021)
46. Rusch, T.K., Chamberlain, B., Rowbottom, J., Mishra, S., Bronstein, M.: Graph-coupled oscillator networks. In: International Conference on Machine Learning. pp. 18888–18909. PMLR (2022)
47. Rusch, T.K., Chamberlain, B.P., Mahoney, M.W., Bronstein, M.M., Mishra, S.: Gradient gating for deep multi-rate learning on graphs. In: The Eleventh International Conference on Learning Representations (2023)
48. Shuai, B., Zuo, Z., Wang, B., Wang, G.: Dag-recurrent neural networks for scene labeling. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 3620–3629 (2016)
49. Singh, S., Chaudhary, K., Dhanda, S.K., Bhalla, S., Usmani, S.S., Gautam, A., Tuknait, A., Agrawal, P., Mathur, D., Raghava, G.P.: Satpdb: a database of structurally annotated therapeutic peptides. Nucleic acids research **44**(D1), D1119–D1126 (2016)
50. Sterling, T., Irwin, J.J.: Zinc 15 – ligand discovery for everyone. Journal of Chemical Information and Modeling **55**(11), 2324–2337 (2015). https://doi.org/10.1021/acs.jcim.5b00559
51. Tailor, S.A., Opolka, F., Lio, P., Lane, N.D.: Adaptive filters for low-latency and memory-efficient graph neural networks. In: International Conference on Learning Representations (2022)

52. Thost, V., Chen, J.: Directed acyclic graph neural networks. In: International Conference on Learning Representations (2021)
53. Tong, Z., Liang, Y., Sun, C., Li, X., Rosenblum, D., Lim, A.: Digraph inception convolutional networks. Advances in neural information processing systems **33**, 17907–17918 (2020)
54. Tönshoff, J., Ritzert, M., Rosenbluth, E., Grohe, M.: Where did the gap go? reassessing the long-range graph benchmark. In: The Second Learning on Graphs Conference (2023)
55. Topping, J., Giovanni, F.D., Chamberlain, B.P., Dong, X., Bronstein, M.M.: Understanding over-squashing and bottlenecks on graphs via curvature. In: International Conference on Learning Representations (2022)
56. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. Advances in neural information processing systems **30** (2017)
57. Weickert, J., et al.: Anisotropic diffusion in image processing, vol. 1. Teubner Stuttgart (1998)
58. Wells, A.F.: Structural inorganic chemistry. Oxford University Press, USA (2012)
59. Yan, Y., Hashemi, M., Swersky, K., Yang, Y., Koutra, D.: Two sides of the same coin: Heterophily and oversmoothing in graph convolutional neural networks. In: 2022 IEEE International Conference on Data Mining (ICDM). pp. 1287–1292. IEEE (2022)
60. Zhang, M., Jiang, S., Cui, Z., Garnett, R., Chen, Y.: D-vae: A variational autoencoder for directed acyclic graphs. Advances in neural information processing systems **32** (2019)
61. Zhang, X., He, Y., Brugnone, N., Perlmutter, M., Hirn, M.: Magnet: A neural network for directed graphs. Advances in neural information processing systems **34**, 27003–27015 (2021)
62. Zhao, L., Akoglu, L.: Pairnorm: Tackling oversmoothing in gnns. In: International Conference on Learning Representations (2020)
63. Zhou, K., Huang, X., Zha, D., Chen, R., Li, L., Choi, S.H., Hu, X.: Dirichlet energy constrained learning for deep graph neural networks. Advances in Neural Information Processing Systems **34**, 21834–21846 (2021)
64. Zhu, J., Yan, Y., Zhao, L., Heimann, M., Akoglu, L., Koutra, D.: Beyond homophily in graph neural networks: Current limitations and effective designs. Advances in neural information processing systems **33**, 7793–7804 (2020)
65. Zhu, X., Sobihani, P., Guo, H.: Long short-term memory over recursive structures. In: Bach, F., Blei, D. (eds.) Proceedings of the 32nd International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 37, pp. 1604–1612. PMLR (07–09 Jul 2015)

# A Mathematical Details

*Proof for Proposition 1*

**Proposition 2** *(DAGs do not over-smooth.) Let* $\mathbf{A} \in \mathbb{R}^{n \times n}$ *represent a DAG,* $\mathbf{A}_+$ *the DAG with added self-loops for all leaf nodes, and* $\tilde{\mathbf{A}_+} = \mathbf{D}_+^{-1}\mathbf{A}_+$ *its normalized adjacency matrix. Further let* $\mathbf{X}^{(k+1)} = \tilde{\mathbf{A}}\mathbf{X}^{(k)}\mathbf{W}^{(k)}$ *for a.e.* $\mathbf{W}^{(0)} \in \mathbb{R}^{d_0 \times d_1}, \ldots, \mathbf{W}^{(k)} \in \mathbb{R}^{d_k \times d_{k+1}}$*. Then, for any* $\mathbf{X}^{(0)} \neq \mathbf{0} \in \mathbb{R}^{n \times d_0}$

$$E\left(\frac{\mathbf{X}^{(l)}}{||\mathbf{X}^{(l)}||_F}\right) \geq \frac{1}{n \cdot d_l}$$

*for all* $l > 0$*.*

*Proof.* For $l > 0$, the representations of the root nodes are the zero vector. As we added self-loops to the leaf nodes, there exists a node representation that is non-zero unless $\mathbf{W}^{(k)}$ maps a representation to the zero vector. Thus, this holds for almost every $\mathbf{W}^{(k)}$ with respect to the Lebesgue measure. Consider the node representation $\mathbf{x}_i$ with the maximal norm $\max \|\mathbf{x}_i\|_1$. There exists a path $r, q_1, \ldots, q-1, i$ between a root node $r$ and node $i$. The Dirichlet energy is bounded by the Dirichlet energy along this path:

$$\begin{aligned} E\left(\frac{\mathbf{X}^{(l)}}{\|\mathbf{X}^{(l)}\|_F}\right) &\geq \left\| \frac{\mathbf{x}_i^{(l)}}{\|\mathbf{X}^{(l)}\|_F} - \frac{\mathbf{x}_{q-1}^{(l)}}{\|\mathbf{X}^{(l)}\|_F} \right\|_2^2 + \cdots + \left\| \frac{\mathbf{x}_{q_1}^{(l)}}{\|\mathbf{X}^{(l)}\|_F} - \frac{\mathbf{x}_r^{(l)}}{\|\mathbf{X}^{(l)}\|_F} \right\|_2^2 \\ &\geq \left\| \frac{\mathbf{x}_i^{(l)}}{\|\mathbf{X}^{(l)}\|_F} \right\|_2^2 . \end{aligned} \tag{11}$$

As the representation of the root node $\mathbf{x}_r^{(l)} = \mathbf{0}$ is zero, we apply the triangle inequality to obtain the last inequality.

We utilize the known norm equivalence of the Frobenius norm

$$\|\mathbf{X}^{(l)}\|_F^2 \leq n \cdot d_l \cdot \|\mathbf{X}^{(l)}\|_\infty^2 \tag{12}$$

and the vector norm equivalence $\|\mathbf{x}_i^{(l)}\|_2^2 \leq \|\mathbf{x}_i^{(l)}\|_1^2$. We thus obtain the lower bound

$$E\left(\frac{\mathbf{X}^{(l)}}{\|\mathbf{X}^{(l)}\|_F}\right) \geq \left\| \frac{\mathbf{x}_i^{(l)}}{\|\mathbf{X}^{(l)}\|_F} \right\|_2^2 \geq \frac{\|\mathbf{x}_i^{(l)}\|_1^2}{n \cdot d \cdot \|\mathbf{X}^{(l)}\|_\infty^2} = \frac{1}{n \cdot d_l} \tag{13}$$

as $\mathbf{x}_i^{(l)}$ has the maximal 1-norm by definition which is equal to $\|\mathbf{X}^{(l)}\|_\infty$.

*Proof for Theorem 1*

**Theorem 3** *(Combining DAGs prevents rank collapse.) Let* $\mathbf{A}_1 \in \mathbb{R}^{n \times n}$ *represent a DAG and* $\mathbf{A}_2 \in \mathbb{R}^{n \times n}$ *represent its reverse DAG. We assume each node to have at least one incoming edge in either* $\mathbf{A}_1$ *or* $\mathbf{A}_2$*. The row-wise vector inequality is denoted as* $\neq_{rw}$*. Let* $\mathbf{X}^{(k+1)} = \mathbf{A}_1\mathbf{X}^{(k)}\mathbf{W}_1^{(k)} + \mathbf{A}_2\mathbf{X}^{(k)}\mathbf{W}_2^{(k)}$*. Then, for all* $l > 0$ *and almost every* $\mathbf{W}_1^{(k)}, \mathbf{W}_2^{(k)} \in \mathbb{R}^{d_k \times d_{k+1}}$ *and* $\mathbf{X}^{(0)} \neq_{rw} \mathbf{0} \in \mathbb{R}^{n \times d_k}$ *we have*

$$\mathbf{X}^{(l)} \neq_{rw} \mathbf{0} \wedge \operatorname{rank}(\mathbf{X}^{(l)}) > 1\,. \tag{14}$$

*Proof.* We will show that this property is maintained in each iteration. Consider one leaf node $v_p$ from $A_1$ and one leaf node $v_q$ from $A_2$. These only have incoming messages from $\mathbf{A}_1$ or $\mathbf{A}_2$, respectively. Thus, $\mathbf{x}_p^{(k+1)} = \mathbf{y}\mathbf{W}_1^{(k)}$ and $\mathbf{x}_q^{(k+1)} = \mathbf{z}\mathbf{W}_2^{(k)}$ for some $\mathbf{y}, \mathbf{z} \in \mathbb{R}^{d_k}$. In particular, $\mathbf{y}$ and $\mathbf{z}$ could be equal or linearly dependent. For $\mathbf{W}_1^{(k)}$ and $\mathbf{W}_2^{(k)}$ almost everywhere with respect to the Lebesgue measure, two vectors will be mapped to linearly independent vectors, as long as $\mathbf{y} \neq \mathbf{0}$ and $\mathbf{z} \neq \mathbf{0}$. Thus, $\text{rank}(\mathbf{X}^{(k+1)}) > 1$. By assumption, each node $i$ has at least one incoming edge in one of the graphs. Thus, $\mathbf{x}_i^{(k+1)} = \mathbf{q}\mathbf{W}_1^{(k)} + \mathbf{r}\mathbf{W}_2^{(k)} \neq 0$ for some $\mathbf{q}, \mathbf{r} \in \mathbb{R}^{d_k}$ and either one being non-zero.

*Computational Complexity* Here, we provide the calculation for the number of multiplications for the bottleneck transformation using the basic matrix multiplication method, i.e., one scalar product per entry:

Performing a matrix multiplication $\mathbf{XW}$ for $\mathbf{X} \in \mathbb{R}^{n \times d}, \mathbf{W} \in \mathbb{R}^{d \times d'}$ costs $n \cdot d' \cdot d$ multiplications, i.e., perform $l$ multiplications for each entry. For the bottleneck transformation, we have $\mathbf{XTUV}$ with $\mathbf{T} \in \mathbb{R}^{d \times s}, \mathbf{U}_i \in \mathbb{R}^{s \times s}, \mathbf{V} \in \mathbb{R}^{s \times d'}$. In total, we have $n \cdot s \cdot d + 3n \cdot s \cdot s + n \cdot s \cdot d'$ multiplications. By equating both terms and canceling $n$, the resulting equation $d' \cdot d = s \cdot d + 3s \cdot s + s \cdot d'$ is equal to the equation for the number of parameters. By definition of $s$, the resulting value for the bottleneck transformation is at most as large as for the regular transformation. Equivalently for the number of additions.

## B Experiments

In this section, we provide additional details regarding our experiments. All experiments were run on an internal cluster and separately on H100 GPUs, each on a single H100 GPU and an Intel Xeon 8468 Sapphire CPU.

### B.1 Improving the Learning Process

Our implementation is built on the Long Range Graph Benchmark (LRGB) [13,54] which is available under the MIT license. It is based on PyTorch Geometric [15]. We add our models while making no changes to the optimization and data construction parts.

**Models and Optimization** All models perform $k$ iterations of message-passing, with ReLU as a non-linear activation function. We reuse the standard optimization process from [13] and use the AdamW [34] optimizer with a cosine learning rate schedule. The cross-entropy loss is used for optimization, and the average precision (AP) as a metric. All DA models use the bottleneck transformation to utilize at most as many parameters as the base version.

**Datasets** We now briefly describe the considered datasets, ZINC, Peptides-Func, and Peptides-Struct.

*ZINC* The ZINC dataset [50] consists of 249 456 chemical compounds that are represented as graphs. Each node represents an atom, and each edge is a bond between two atoms. On average, a graph has around 23 nodes and 50 edges. Node features are given as a single value indicating its corresponding type of heavy atom. We do not utilize edge features. The objective is given as a graph regression task, which corresponds to the prediction of the constrained solubility of the molecule. The mean absolute error (MAE) is used as the loss function and for the score. Each experiment on ZINC is repeated for three random seeds. ZINC is freely commercially available under the license DbCL.

*Peptides-Func* The Peptides dataset consists of 15 535 peptides, which are short molecular chains [49]. As with ZINC, nodes of the graph represent atoms and edges the bonds between them. They are part of the LRGB as peptides have a large diameter while each node has a low average degree of around 2. Thus, it is argued that this dataset requires models that can combine distant information in the graph, i.e., models with many layers. Node and edge features are constructed using molecular SMILES based on the atom types. This dataset was released under license CC BY-NC 4.0.

The task is to predict the molecular properties of each peptide, i.e., a multi-label graph classification task. Each graph belongs, on average, to 1.65 of 10 classes. As proposed by [13], the cross-entropy (CE) loss is used for optimization, and the unweighted mean average precision (AP) as the metric. We follow their same data split, i.e., 70% for training and 15% for validation and testing.

The resulting representations are globally aggregated using the mean and mapped to class probabilities using a linear layer.

*Peptides-Struct* The dataset is the same as used for Peptides-Func. The task is to predict five continuous geometric properties of the peptides, i.e., a multi-label graph regression task. The same data split is utilized. The MAE is used for both optimization and as the target metric.

**Orderings** For all orderings, we compute a single value $r_i$ per node $v_i$ and construct the strict partial ordering $i \prec j \Leftrightarrow r_i < r_j$.

*Random* For the random ordering, we assign each node $v_i \in \mathcal{V}$ a unique random index $r_i \in [0, |\mathcal{V}|]$. This index is consistent across layers and epochs. The edges within each computational graph will be similar to those from different computational graphs. Thus, we expect the optimal solution to be achieved when all transformations are equal.

*Features* This ordering utilizes the initial node features $\mathbf{x}_i \in \mathbb{R}^d$ by summing $r_i = \sum_{c \in [d]} \mathbf{x}_{ic}$ over all features of that node.

*PPR* Here, we perform 15 iterations of Personalized PageRank (PPR) with a restart probability $\alpha = 0.1$. This provides a finer node centrality measure as

Table 4: Best hyperparameters for results in Table 1

| Dataset | | ZINC | |
|---|---|---|---|
| Parameter | Split | LR | Layers |
| GCN | Train | 0.0003 | 16 |
| | Val | 0.0003 | 16 |
| DA-GCN (random) | Train | 0.0003 | 16 |
| | Val | 0.001 | 8 |
| DA-GCN (Features) | Train | 0.0003 | 16 |
| | Val | 0.0003 | 16 |
| DA-GCN (PPR) | Train | 0.0003 | 16 |
| | Val | 0.0003 | 16 |
| DA-GCN (Degree) | Train | 0.0003 | 16 |
| | Val | 0.001 | 8 |
| DA-GCN (Degree) \ Bottleneck | Train | 0.0003 | 16 |
| | Val | 0.0003 | 16 |

opposed to the node degree. A finer ordering will have fewer edges in the third computational graph, but the similarity between edges in each computational graph may be lowered. A node's role in a graph is connected to its centrality, e.g., influential persons in social networks have many connections.

*Degree* As a coarser node centrality measure, we consider the node degree $r_i = d_i$. For molecular graphs, the node degree is closely connected with its role within the molecule. The degree is also much more efficient to compute compared to PPR.

**Comparing Partial Orderings (Table 1)** To compare the performances of the partial orderings we considered, we evaluate them on the ZINC task. All models have a fixed hidden dimension of 64. For DA-GCN without a bottleneck transformation, the hidden dimension is reduced so that the total number of parameters is less than the number of parameters for the GCN with the same number of layers. We tune the learning rate for all models with values $\in \{0.001, 0.0003, 0.0001\}$ and the number of layers $\in \{1, 2, 4, 8, 16\}$. All experiments are repeated for three random seeds. Best hyperparameters are presented in Table 4. We compute the ordering and the split once in each forward pass for all DA methods. Normalizing the edge weights is performed once per forward pass for GCN and DA-GCN. Displayed runtimes are for four-layer models. These experiments run for a total of around 600 hours on one H100 GPU.

**Preventing Over-Smoothing (Figure 2)** We sample 100 random graphs from the Peptides-Func dataset for a total of 15 454 nodes and 31 458 edges. We perform a single linear transformation to change the feature dimension to 16.

Table 5: Best hyperparameters for results in Table 2

| Dataset | | ZINC | | Peptides-Func | | Peptides-Struct | |
|---|---|---|---|---|---|---|---|
| Parameter | Split | LR | Layers | LR | Layers | LR | Layers |
| GCN | Train | 0.003 | 16 | 0.005 | 4 | 0.005 | 8 |
| | Val | 0.003 | 16 | 0.001 | 8 | 0.0005 | 32 |
| DA-GCN | Train | 0.003 | 16 | 0.001 | 8 | 0.001 | 8 |
| | Val | 0.001 | 8 | 0.001 | 16 | 0.0005 | 32 |
| SAGE | Train | 0.003 | 16 | 0.001 | 16 | 0.001 | 16 |
| | Val | 0.003 | 16 | 0.001 | 8 | 0.0005 | 16 |
| DA-SAGE | Train | 0.003 | 16 | 0.001 | 16 | 0.001 | 16 |
| | Val | 0.003 | 16 | 0.001 | 16 | 0.005 | 32 |

We then apply message-passing iterations, each followed by a ReLU activation. Each iteration maintains the feature dimension as 16. Feature transformations are not shared between iterations. The Dirichlet energy is calculated after the ReLU activation. Corresponding to their aggregation functions, the Dirichlet energy for the GCN and the DA-GCN uses the symmetrically normalized graph Laplacian and the unnormalized graph Laplacian for SAGE and DA-SAGE. These experiments run for less than one minute on a CPU.

**Details on Table 2** We tune the number of layers in $\{1, 2, 4, 8, 16\}$ and the base learning rate in $\{0.005, 0.001, 0.0005\}$ for Peptides-Func and Peptides-Struct using a grid search. As proposed in [13], the hidden dimension is set so that the total number of parameters is less than 500k. For ZINC, we tune the number of layers in $\{1, 2, 4, 8, 16\}$ and the base learning rate in $\{0.001, 0.0003, 0.0001\}$. All runs utilize a cosine learning rate schedule with a maximum of 500 epochs for Peptides and 400 for ZINC. As given by our reference implementation [54], a batch size of 200 is used for Peptides. For ZINC, a batch size of 32 is used, as given in [12]. Best hyperparameters for each experiment are displayed in Table 5. These experiments require around 450 hours on an H100 GPU.

### B.2 Comparison with State-of-the-Art

Our models are integrated into the implementation of Dir-GNNs [42]. This implementation is available under the MIT license.

#### Datasets

*Chameleon and Squirrel* These two datasets are based on pages about particular topics in Wikipedia. Nodes represent articles on that topic and edges their links. Node features are constructed as the appearance of particular nouns [45]. The task is to classify each article based on their average monthly traffic [40]. Chameleon

Table 6: Best hyperparameters for the results in Table 3.

| Dataset | Learning Rate | Layers | Dropout Ratio |
|---|---|---|---|
| Chameleon | 0.005 | 6 | 0.2 |
| Squirrel | 0.001 | 6 | 0.0 |
| Roman-Empire | 0.005 | 6 | 0.2 |
| Arxiv-Year | 0.005 | 5 | 0.6 |
| Snap-Patents | 0.01 | 6 | 0.0 |

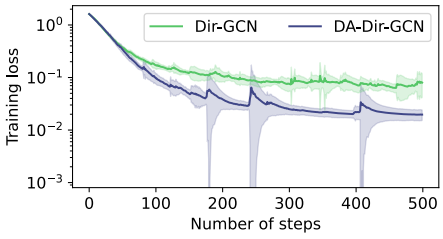
consists of 2277 nodes and 36 101 edges. Squirrel consists of 5201 nodes and 217 073 edges. To the best of our knowledge, the dataset was released without a license.

*Arxiv-Year* In this dataset, nodes correspond to publications, and an edge is constructed when a publication cites another. The task is to classify the publication year into one of five time spans. It consists of 169 343 nodes and 1 166 243 edges. Nodes are given by word 128-dimensional embeddings of the title and abstract of the corresponding publication. Arxiv-Year is released under the ODC-BY license.

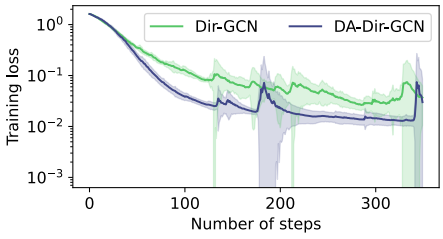
*Snap-Patents* Nodes correspond to patents, for which the year it was granted should be classified into one of five time spans. Edges similarly correspond to citations between patents. This dataset consists of 2 923 922 nodes and 13 975 791 edges. This dataset was released without a license in [22].

**Implementational Details** We compare to several state-of-the-art methods for directed graphs, namely DiGCN [53], MagNet [61], and Dir-GNN [42], and state-of-the-art methods for heterophilic graphs, namely $H_2$GCN [64], GPR-GNN [8], LINKX [33], FSGNN [37], ACM-GCN [35], GloGNN [31], and Gradient Gating [47].

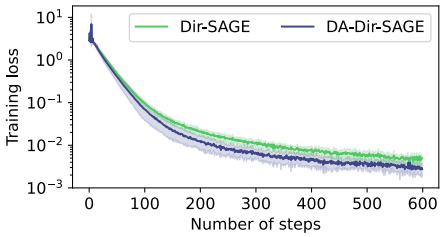
We use the same model as the Dir-GNN [42], with the only change being the DA-MPNN instead of each MPNN, as proposed in Equation 9. The models consist of $k$ layers of message-passing, each followed by ReLU and potentially Dropout. All representations are normalized by the $L2$-norm $\|\mathbf{X}\|_2$. Final representations are obtained by Knowledge Knowledge, either concatenating all intermediate states (cat) or taking the element-wise maximum (max). Bottleneck transformation and degree-based ordering are used for all experiments. Bottleneck transformations are initialized with the same values across the three computational graphs. It thus equals the base model at initialization and helps with generalization. We tune DA-Dir-GNN using the same hyperparameters and their ranges as performed for Dir-GNN using the same implementation: The learning rate $\in \{0.01, 0.005, 0.001, 0.0005\}$, number of layers $\in \{4, 5, 6\}$, jumping knowledge $\in \{\text{cat}, \max\}$, dropout $\in \{0.0, 0.2, 0.4, 0.6\}$. We use their optimal values for



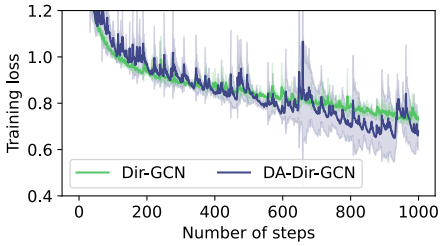

(a) Chameleon.



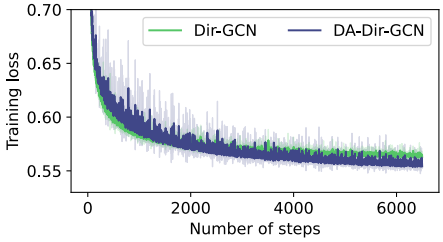

(b) Squirrel.




(c) Roman-Empire.




(d) Arxiv-Year.




(e) Snap-Patents.

Fig. 4: Training loss during optimization. All hyperparameters are set to the best-performing ones of Dir-GNNs.

hidden feature dimension $\in \{32, 64, 128, 256, 512\}$, normalization $\in \{\text{True}, \text{False}\}$ and $\alpha \in \{0., 0.5, 1\}$ for each task. As given in their implementation, the patience of stopping training based on not improving the validation accuracy is set to 200 for Roman-Empire, Snap-Patents, and Arxiv-Year and to 400 for Squirrel and Chameleon. Consequently, DA-SAGE is used for Roman-Empire and DA-GCN for the other datasets. The best-performing hyperparameters are presented in Table 6.

**Runtime** On an H100, each run for Chameleon takes around 10 seconds, for Squirrel 20 seconds, for Roman-Empire 90 seconds, for Arxiv-Year 15 minutes, and for Snap-Patents 30 minutes. In total, these Experiments take around 150 GPU hours. We estimate the time of our preliminary experiments with various versions of DA-MPNNs to additional 1500 hours.

**Training Losses** We present a comparison between training losses for all datasets in Figure 4. For a fair comparison, all runs utilize the best-performing hyperparameters that were identified for the corresponding Dir-GNNs [42]. For all considered datasets, we observe that DA-MPNNs benefit from a smaller learning rate.