Graph convolution is the core of most Graph Neural Networks (GNNs) and usually approximated by message passing between direct (one-hop) neighbors. In this work, we remove the restriction of using only the direct neighbors by introducing a powerful, yet spatially localized graph convolution: Graph diffusion convolution (GDC). GDC leverages generalized graph diffusion, examples of which are the heat kernel and personalized PageRank. It alleviates the problem of noisy and often arbitrarily defined edges in real graphs. We show that GDC is closely related to spectral-based models and thus combines the strengths of both spatial (message passing) and spectral methods. We demonstrate that replacing message passing with graph diffusion convolution consistently leads to significant performance improvements across a wide range of models on both supervised and unsupervised tasks and a variety of datasets. Furthermore, GDC is not limited to GNNs but can trivially be combined with any graph-based model or algorithm (e.g. spectral clustering) without requiring any changes to the latter or affecting its computational complexity. Our implementation is available online.

1 Introduction

When people started using graphs for evaluating chess tournaments in the middle of the 19th century they only considered each player’s direct opponents, i.e. their first-hop neighbors. Only later was the analysis extended to recursively consider higher-order relationships via \( A^2, A^3, \) etc. and finally generalized to consider all exponents at once, using the adjacency matrix’s dominant eigenvector \[ \Lambda \] \[ 38 \], \[ 75 \]. The field of Graph Neural Networks (GNNs) is currently in a similar state. Graph Convolutional Networks (GCNs) \[ 33 \] are referred to as Message Passing Neural Networks (MPNNs) \[ 24 \] are the prevalent approach in this field but they only pass messages between neighboring nodes in each layer. These messages are then aggregated at each node to form the embedding for the next layer. While MPNNs do leverage higher-order neighborhoods in deeper layers, limiting each layer’s messages to one-hop neighbors seems arbitrary. Edges in real graphs are often noisy or defined using an arbitrary threshold \[ 70 \], so we can clearly improve upon this approach.

Since MPNNs only use the immediate neighborhood information, they are often referred to as spatial methods. On the other hand, spectral-based models do not just rely on first-hop neighbors and capture more complex graph properties \[ 16 \]. However, while being theoretically more elegant, these methods are routinely outperformed by MPNNs on graph-related tasks \[ 33 \], \[ 74 \], \[ 81 \] and do not generalize to previously unseen graphs. This shows that message passing is a powerful framework worth extending upon. To reconcile these two separate approaches and combine their strengths we propose a novel technique of performing message passing inspired by spectral methods: Graph diffusion convolution (GDC). Instead of aggregating information only from the first-hop neighbors, GDC aggregates information from a larger neighborhood. This neighborhood is constructed via a new graph generated by sparsifying a generalized form of graph diffusion. We show how graph diffusion
is expressed as an equivalent polynomial filter and how GDC is closely related to spectral-based models while addressing their shortcomings. GDC is spatially localized, scalable, can be combined with message passing, and generalizes to unseen graphs. Furthermore, since GDC generates a new sparse graph it is not limited to MPNNs and can trivially be combined with any existing graph-based model or algorithm in a plug-and-play manner, i.e. without requiring changing the model or affecting its computational complexity. We show that GDC consistently improves performance across a wide range of models on both supervised and unsupervised tasks and various homophilic datasets. In summary, this paper’s core contributions are:

1. Proposing graph diffusion convolution (GDC), a more powerful and general, yet spatially localized alternative to message passing that uses a sparsified generalized form of graph diffusion. GDC is not limited to GNNs and can be combined with any graph-based model or algorithm.
2. Analyzing the spectral properties of GDC and graph diffusion. We show how graph diffusion is expressed as an equivalent polynomial filter and analyze GDC’s effect on the graph spectrum.
3. Comparing and evaluating several specific variants of GDC and demonstrating its wide applicability to supervised and unsupervised learning on graphs.

2 Generalized graph diffusion

We consider an undirected graph \( G = (\mathcal{V}, \mathcal{E}) \) with node set \( \mathcal{V} \) and edge set \( \mathcal{E} \). We denote with \( N = |\mathcal{V}| \) the number of nodes and \( A \in \mathbb{R}^{N \times N} \) the adjacency matrix. We define generalized graph diffusion via the diffusion matrix

\[
S = \sum_{k=0}^{\infty} \theta_k T^k, \tag{1}
\]

with the weighting coefficients \( \theta_k \), and the generalized transition matrix \( T \). The choice of \( \theta_k \) and \( T^k \) must at least ensure that Eq. (1) converges. In this work we will consider somewhat stricter conditions and require that \( \sum_{k=0}^{\infty} \theta_k = 1 \), \( \theta_k \in [0, 1] \), and that the eigenvalues of \( T \) are bounded by \( \lambda_i \in [0, 1] \), which together are sufficient to guarantee convergence. Note that regular graph diffusion commonly requires \( T \) to be column- or row-stochastic.

**Transition matrix.** Examples for \( T \) in an undirected graph include the random walk transition matrix \( T_{rw} = AD^{-1} \) and the symmetric transition matrix \( T_{sym} = D^{-1/2}AD^{-1/2} \), where the degree matrix \( D \) is the diagonal matrix of node degrees, i.e. \( D_{ii} = \sum_{j=1}^{N} A_{ij} \). Note that in our definition \( T_{rw} \) is column-stochastic. We furthermore adjust the random walk by adding (weighted) self-loops to the original adjacency matrix, i.e. use \( T_{sym} = (w_{loop}I + D)^{-1/2}\left((w_{loop}I + A)(w_{loop}I + D)^{-1/2}\right) \), with the self-loop weight \( w_{loop} \in \mathbb{R}_+ \). This is equivalent to performing a lazy random walk with a probability of staying at node \( i \) of \( p_{stay,i} = w_{loop}/D_i \).

**Special cases.** Two popular examples of graph diffusion are personalized PageRank (PPR) \([57]\) and the heat kernel \([30]\). PPR corresponds to choosing \( T = T_{rw} \) and \( \theta_k^{PPR} = \alpha(1 - \alpha)^k \), with teleport probability \( \alpha \in (0, 1) \) \([13]\). The heat kernel uses \( T = T_{rw} \) and \( \theta_k^{HK} = e^{-\frac{t}{\theta^k}}, \) with the diffusion time \( t \) \([14]\). Another special case of generalized graph diffusion is the approximated graph convolution introduced by Kipf & Welling \([33]\), which translates to \( \theta_1 = 1 \) and \( \theta_k = 0 \) for \( k \neq 1 \) and uses \( T = T_{sym} \) with \( w_{loop} = 1 \).

**Weighting coefficients.** We compute the series defined by Eq. (1) either in closed-form, if possible, or by restricting the sum to a finite number \( K \). Both the coefficients defined by PPR and the heat kernel give a closed-form solution for this series that we found to perform well for the tasks considered. Note that we are not restricted to using \( T_{rw} \) and can use any generalized transition matrix along with the coefficients \( \theta_k^{PPR} \) or \( \theta_k^{HK} \) and the series still converges. We can furthermore choose \( \theta_k \) by repurposing the graph-specific coefficients obtained by methods that optimize coefficients analogous to \( \theta_k \) as part of their training process. We investigated this approach using label propagation \([8, 13]\) and node embedding models \([1]\). However, we found that the simple coefficients defined by PPR or the heat kernel perform better than those learned by these models (see Fig. 4 in Sec. 6).
3 Graph diffusion convolution

Essentially, graph diffusion convolution (GDC) exchanges the normal adjacency matrix $A$ with a sparsified version $\tilde{S}$ of the generalized graph diffusion matrix $S$, as illustrated by Fig. 1. This matrix defines a weighted and directed graph, and the model we aim to augment is applied to this graph instead. We found that the calculated edge weights are beneficial for the tasks considered. However, we even found that GDC works when ignoring the weights after sparsification. This enables us to use GDC with models that only support unweighted edges such as the degree-corrected stochastic block model (DCSBM). If required, we make the graph undirected by using $(\tilde{S} + \tilde{S}^T)/2$, e.g. for spectral clustering. With these adjustments GDC is applicable to any graph-based model or algorithm.

Intuition. The general intuition behind GDC is that graph diffusion smooths out the neighborhood over the graph, acting as a kind of denoising filter similar to Gaussian filters on images. This helps with graph learning since both features and edges in real graphs are often noisy. Previous works also highlighted the effectiveness of graph denoising. Berberidis & Giannakis [7] showed that PPR is able to reconstruct the underlying probability matrix of a sampled stochastic block model (SBM) graph. Kloumann et al. [35] and Ragain [64] showed that PPR is optimal in recovering the SBM and DCSBM clusters in the space of landing probabilities under the mean field assumption. Li et al. [40] generalized this result by analyzing the convergence of landing probabilities to their mean field values. These results confirm the intuition that graph diffusion-based smoothing indeed recovers meaningful neighborhoods from noisy graphs.

Sparsification. Most graph diffusions result in a dense matrix $S$. This happens even if we do not sum to $k = \infty$ in Eq. 1 due to the “four/six degrees of separation” in real-world graphs [5]. However, the values in $S$ represent the influence between all pairs of nodes, which typically are highly localized [54]. This is a major advantage over spectral-based models since the spectral domain does not provide any notion of locality. Spatial localization allows us to simply truncate small values of $S$ and recover sparsity, resulting in the matrix $\tilde{S}$. In this work we consider two options for sparsification: 1. top-$k$: Use the $k$ entries with the highest mass per column, 2. Threshold $\epsilon$: Set entries below $\epsilon$ to zero. Sparsification would still require calculating a dense matrix $S$ during preprocessing. However, many popular graph diffusions can be approximated efficiently and accurately in linear time and space. Most importantly, there are fast approximations for both PPR [3,77] and the heat kernel [34], with which GDC achieves a linear runtime $O(N)$. Furthermore, top-$k$ truncation generates a regular graph, which is amenable to batching methods and solves problems related to widely varying node degrees [15]. Empirically, we even found that sparsification slightly improves prediction accuracy (see Fig. 5 in Sec. 6). After sparsification we calculate the (symmetric or random walk) transition matrix on the resulting graph via $T_{\text{sym}} = D^{-1/2} S D^{-1/2}$.

Limitations. GDC is based on the assumption of homophily, i.e. “birds of a feather flock together” [49]. Many methods share this assumption and most common datasets adhere to this principle. However, this is an often overlooked limitation and it seems non-straightforward to overcome. One way of extending GDC to heterophily, i.e. “opposites attract”, might be negative edge weights.
[17, 24]. Furthermore, we suspect that GDC does not perform well in settings with more complex edges (e.g., knowledge graphs) or graph reconstruction tasks such as link prediction. Preliminary experiments showed that GDC indeed does not improve link prediction performance.

4 Spectral analysis of GDC

Even though GDC is a spatial-based method it can also be interpreted as a graph convolution and analyzed in the graph spectral domain. In this section we show how generalized graph diffusion is expressed as an equivalent polynomial filter and vice versa. Additionally, we perform a spectral analysis of GDC, which highlights the tight connection between GDC and spectral-based models.

Spectral graph theory. To employ the tools of spectral theory to graphs we exchange the regular Laplace operator with either the unnormalized Laplacian $L_{un} = D - A$, the random-walk normalized $L_{rw} = I_N - T_{rw}$, or the symmetric normalized graph Laplacian $L_{sym} = I_N - T_{sym}$ [76]. The Laplacian’s eigendecomposition is $L = U \Lambda U^T$, where both $U$ and $\Lambda$ are real-valued. The graph Fourier transform of a vector $x$ is then defined via $\hat{x} = U^T x$ and its inverse as $x = U \hat{x}$. Using this we define a graph convolution on $G$ as $x \ast_G y = U((U^T x) \odot (U^T y))$, where $\odot$ denotes the Hadamard product. Hence, a filter $g_\xi$ with parameters $\xi$ acts on $x$ as $g_\xi(L)x = U G_\xi(\Lambda) U^T x$, where $G_\xi(\Lambda) = \text{diag}(g_\xi(\Lambda), \ldots, g_\xi(\Lambda))$. A common choice for $g_\xi$ in the literature is a polynomial filter of order $J$, since it is localized and has a limited number of parameters [16, 28]:

$$g_\xi(L) = \sum_{j=0}^J \xi_j L^j = U \left( \sum_{j=0}^J \xi_j \Lambda^j \right) U^T.$$ (2)

Graph diffusion as a polynomial filter. Comparing Eq. 1 with Eq. 2 shows the close relationship between polynomial filters and generalized graph diffusion since we only need to exchange $L$ by $T$ to go from one to the other. To make this relationship more specific and find a direct correspondence between polynomial filters and generalized graph diffusion since we only need to exchange $L$ and a polynomial filter with parameters $\xi_j$ we need to find parameters that solve

$$\sum_{j=0}^J \xi_j L^j = \sum_{k=0}^K \theta_k T^k.$$ (3)

To find these parameters we choose the Laplacian corresponding to $L = I_n - T$, resulting in (see App. A)

$$\xi_j = \sum_{k=j}^{K} \binom{k}{j} (-1)^j \theta_k, \quad \theta_k = \sum_{j=k}^{J} \binom{j}{k} (-1)^k \xi_j,$$ (4)

which shows the direct correspondence between graph diffusion and spectral methods. Note that we need to set $J = K$. Solving Eq. 3 for the coefficients corresponding to the heat kernel $\theta_k^{HK}$ and PPR $\theta_k^{PPR}$ leads to

$$\xi_j^{HK} = \frac{(-t)^j}{j!}, \quad \xi_j^{PPR} = \left(1 - \frac{1}{\alpha}\right)^j,$$ (5)

showing how the heat kernel and PPR are expressed as polynomial filters. Note that PPR’s corresponding polynomial filter converges only if $\alpha > 0.5$. This is caused by changing the order of summation when deriving $\xi_j^{PPR}$, which results in an alternating series. However, if the series does converge it gives the exact same transformation as the equivalent graph diffusion.

Spectral properties of GDC. We will now extend the discussion to all parts of GDC and analyze how they transform the graph Laplacian’s eigenvalues. GDC consists of four steps: 1. Calculate the transition matrix $T$, 2. take the sum in Eq. 1 to obtain $\tilde{S}$, 3. sparsify the resulting matrix by truncating small values, resulting in $S$, and 4. calculate the transition matrix $T_\tilde{S}$.

1. Transition matrix. Calculating the transition matrix $T$ only changes which Laplacian matrix we use for analyzing the graph’s spectrum, i.e. we use $L_{sym}$ or $L_{rw}$ instead of $L_{un}$. Adding self-loops to obtain $\tilde{T}$ does not preserve the eigenvectors and its effect therefore cannot be calculated precisely. Wu et al. [78] empirically found that adding self-loops shrinks the graph’s eigenvalues.
2. Sum over $T^k$. Summation does not affect the eigenvectors of the original matrix, since $T^k v_i = \lambda_i T^{k-1} v_i = \lambda_i^k v_i$ for the eigenvector $v_i$ of $T$ with associated eigenvalue $\lambda_i$. This also shows that the eigenvalues are transformed as
\[
\tilde{\lambda}_i = \sum_{k=0}^\infty \theta_k \lambda_i^k.
\] (6)

Since the eigenvalues of $T$ are bounded by 1 we can use the geometric series to derive a closed-form expression for PPR, i.e. $\tilde{\lambda}_i = \alpha \sum_{k=0}^\infty (1 - \alpha)^k \lambda_i^k = \frac{\alpha}{1 - (1 - \alpha) \lambda_i}$.

For the heat kernel we use the exponential series, resulting in $\tilde{\lambda}_i = e^{-t} \sum_{k=0}^\infty \frac{\lambda_i^k}{k!} = e^{t(\lambda_i - 1)}$. How this transformation affects the corresponding Laplacian’s eigenvalues is illustrated in Fig. 2a. Both PPR and the heat kernel act as low-pass filters. Low eigenvalues corresponding to large-scale structure in the graph (e.g. clusters) are amplified, while high eigenvalues corresponding to fine details but also noise are suppressed.

3. Sparsification. Sparsification changes both the eigenvalues and the eigenvectors, which means that there is no direct correspondence between the eigenvalues of $S$ and $\tilde{S}$ and we cannot analyze its effect analytically. However, we can use eigenvalue perturbation theory (Stewart & Sun [69], Corollary 4.1.3) to derive the upper bound
\[
\sqrt{\sum_{i=1}^N (\tilde{\lambda}_i - \lambda_i)^2} \leq ||E||_F \leq N \epsilon \quad \text{max} \leq N \epsilon,
\] (7)

with the perturbation matrix $E = \tilde{S} - S$ and the threshold $\epsilon$. This bound significantly overestimates the perturbation since PPR and the heat kernel both exhibit strong localiztion on real-world graphs and hence the change in eigenvalues empirically does not scale with $N$ (or, rather, $\sqrt{N}$). By ordering the eigenvalues we see that, empirically, the typical thresholds for sparsification have almost no effect on the eigenvalues, as shown in Fig. 2b and in the close-up in Fig. 11 in App. B.2. We find that the small changes caused by sparsification mostly affect the highest and lowest eigenvalues. The former correspond to very large clusters and long-range interactions, which are undesired for local graph smoothing. The latter correspond to spurious oscillations, which are not helpful for graph learning either and most likely affected because of the abrupt cutoff at $\epsilon$.

4. Transition matrix on $\tilde{S}$. As a final step we calculate the transition matrix on the resulting graph $\tilde{S}$. This step does not just change which Laplacian we consider since we have already switched to using the transition matrix in step 1. It furthermore does not preserve the eigenvectors and is thus again best investigated empirically by ordering the eigenvalues. Fig. 2c shows that, empirically, this step slightly dampens low eigenvalues. This may seem counterproductive. However, the main purpose of using the transition matrix is ensuring that sparsification does not cause nodes to be treated differently by losing a different number of adjacent edges. The filtering is only a side-effect.

Limitations of spectral-based models. While there are tight connections between GDC and spectral-based models, GDC is actually spatial-based and therefore does not share their limitations. Similar to polynomial filters, GDC does not compute an expensive eigenvalue decomposition, preserves locality on the graph and is not limited to a single graph after training, i.e. typically the same coefficients $\theta_k$.
can be used across graphs. The choice of coefficients \( \theta_k \) depends on the type of graph at hand and does not change significantly between similar graphs. Moreover, the hyperparameters \( \alpha \) of PPR and \( t \) of the heat kernel usually fall within a narrow range that is rather insensitive to both the graph and model (see Fig. 8 in Sec. 6).

5 Related work

Graph diffusion and random walks have been extensively studied in classical graph learning [13] [14] [36] [37], especially for clustering [34], semi-supervised classification [12] [22], and recommendation systems [44]. For an overview of existing methods see Masuda et al. [46] and Fouss et al. [22].

The first models similar in structure to current Graph Neural Networks (GNNs) were proposed by Sperduti & Starita [63] and Baskin et al. [60], and the name GNN first appeared in [25] [63]. However, they only became widely adopted in recent years, when they started to outperform classical models in many graph-related tasks [19] [23] [42] [82]. In general, GNNs are classified into spectral-based models [11] [16] [29] [33] [41], which are based on the eigendecomposition of the graph Laplacian, and spatial-based methods [24] [27] [43] [52] [56] [62] [74], which use the graph directly and form new representations by aggregating the representations of a node and its neighbors. However, this distinction is often rather blurry and many models can not be clearly attributed to one type or the other. Deep learning also inspired a variety of unsupervised node embedding methods. Most models use random walks to learn node embeddings in a similar fashion as word2vec [51] [26] [61] and have been shown to implicitly perform a matrix factorization [63]. Other unsupervised models learn Gaussian distributions instead of vectors [10], use an auto-encoder [32], or train an encoder by maximizing the mutual information between local and global embeddings [73].

There have been some isolated efforts of using extended neighborhoods for aggregation in GNNs and graph diffusion for node embeddings. PPNP [23] propagates the node predictions generated by a neural network using personalized PageRank, DCNN [4] extends node features by concatenating features aggregated using the transition matrices of \( k \)-hop random walks, GraphHeat [29] uses the heat kernel and PAN [45] the transition matrix of maximal entropy random walks to aggregate over nodes in each layer. PinSage [82] uses random walks for neighborhood aggregation, and MixHop [2] concatenates embeddings aggregated using the transition matrices of \( k \)-hop random walks before each layer. VERSE [71] learns node embeddings by minimizing KL-divergence from the PPR matrix to a low-rank approximation. Attention walk [1] uses a similar loss to jointly optimize the node embeddings and diffusion coefficients \( \theta_k \). None of these works considered sparsification, generalized graph diffusion, spectral properties, or using preprocessing to generalize across models.

6 Experimental results

Experimental setup. We take extensive measures to prevent any kind of bias in our results. We optimize the hyperparameters of all models on all datasets with both the unmodified graph and all GDC variants separately using a combination of grid and random search on the validation set. Each result is averaged across 100 data splits and random initializations for supervised tasks and 20 random initializations for unsupervised tasks, as suggested by Gasteiger et al. [23] and Shchur et al. [67]. We report performance on a test set that was used exactly once. We report all results as averages with 95% confidence intervals calculated via bootstrapping.

We use the symmetric transition matrix with self-loops \( \tilde{T}_{sym} = (I_N + D)^{-1/2}(I_N + A)(I_N + D)^{-1/2} \) for GDC and the column-stochastic transition matrix \( \tilde{T}_{sym} = \tilde{S}D^{-1}_\tilde{S} \) on \( \tilde{S} \). We present two simple and effective choices for the coefficients \( \theta_k \): The heat kernel and PPR. The diffusion matrix \( \tilde{S} \) is sparsified using either an \( \epsilon \)-threshold or top-\( k \).

Datasets and models. We evaluate GDC on six datasets: The citation graphs Citeseer [66], Cora [48], and Pubmed [53], the co-author graph Coauthor CS [67], and the co-purchase graphs Amazon Computers and Amazon Photo [47] [62]. We only use their largest connected components. We show how GDC impacts the performance of 9 models: Graph Convolutional Network (GCN) [33], Graph Attention Network (GAT) [74], jumping knowledge network (JK) [80], Graph Isomorphism Network (GIN) [81], and ARMA [9] are supervised models. The degree-corrected stochastic block model (DCSBM) [31], spectral clustering (using \( L_{sym} \)) [55], DeepWalk
Figure 3: Node classification accuracy of GNNs with and without GDC. GDC consistently improves accuracy across models and datasets. It is able to fix models whose accuracy otherwise breaks down.

Figure 4: Clustering accuracy with and without GDC. GDC consistently improves the accuracy across a diverse set of models and datasets.

[61] and Deep Graph Infomax (DGI) [73] are unsupervised models. Note that DGI uses node features while other unsupervised models do not. We use k-means clustering to generate clusters from node embeddings. Dataset statistics and hyperparameters are reported in App. B.

**Semi-supervised node classification.** In this task the goal is to label nodes based on the graph, node features $X \in \mathbb{R}^{N \times F}$ and a subset of labeled nodes $y$. The main goal of GDC is improving the performance of MPNN models. Fig. 3 shows that GDC consistently and significantly improves the accuracy of a wide variety of state-of-the-art models across multiple diverse datasets. Note how GDC is able to fix the performance of GNNs that otherwise break down on some datasets (e.g. GAT). We also surpass or match the previous state of the art on all datasets investigated (see App. B.2).

**Clustering.** We highlight GDC’s ability to be combined with any graph-based model by reporting the performance of a diverse set of models that use a wide range of paradigms. Fig. 4 shows the unsupervised accuracy obtained by matching clusters to ground-truth classes using the Hungarian algorithm. Accuracy consistently and significantly improves for all models and datasets. Note that spectral clustering uses the graph’s eigenvectors, which are not affected by the diffusion step itself. Still, its performance improves by up to 30 percentage points. Results in tabular form are presented in App. B.2.
Figure 5: GCN+GDC accuracy (using PPR and top-\(k\)). Lines indicate original accuracy and degree. GDC surpasses original accuracy at around the same degree independent of dataset. Sparsification often improves accuracy.

Figure 6: Difference in GCN+GDC accuracy (using PPR and top-\(k\), percentage points) compared to the symmetric \(T_{\text{sym}}\) without self-loops. \(T_{\text{rw}}\) performs worse and self-loops have no significant effect.

Figure 7: Accuracy of GDC with coefficients \(\theta_k\) defined by PPR and learned by AdaDIF. Simple PPR coefficients consistently perform better than those obtained by AdaDIF, even with optimized regularization.

Figure 8: Accuracy achieved by using GDC with varying hyperparameters of PPR (\(\alpha\)) and the heat kernel (\(t\)). Optimal values fall within a narrow range that is consistent across datasets and models.

In this work we concentrate on node-level prediction tasks in a transductive setting. However, GDC can just as easily be applied to inductive problems or different tasks like graph classification. In our experiments we found promising, yet not as consistent results for graph classification (e.g. 2.5 percentage points with GCN on the DD dataset [13]). We found no improvement for the inductive setting on PPI [50], which is rather unsurprising since the underlying data used for graph construction already includes graph diffusion-like mechanisms (e.g. regulatory interactions, protein complexes, and metabolic enzyme-coupled interactions). We furthermore conducted experiments to answer five important questions:

**Does GDC increase graph density?** When sparsifying the generalized graph diffusion matrix \(S\) we are free to choose the resulting level of sparsity in \(S\). Fig. 5 indicates that, surprisingly, GDC requires roughly the same average degree to surpass the performance of the original graph independent of the dataset and its average degree (\(c\)-threshold in App. B.2, Fig. 12). This suggests that the sparsification hyperparameter can be obtained from a fixed average degree. Note that CORA and CITESEER are both small graphs with low average degree. Graphs become denser with size [39] and in practice we expect GDC to typically reduce the average degree at constant accuracy. Fig. 5 furthermore shows that there is an optimal degree of sparsity above which the accuracy decreases. This indicates that sparsification is not only computationally beneficial but also improves prediction performance.

**How to choose the transition matrix \(T\)?** We found \(T_{\text{sym}}\) to perform best across datasets. More specifically, Fig. 6 shows that the symmetric version on average outperforms the random walk transition matrix \(T_{\text{rw}}\). This figure also shows that GCN accuracy is largely insensitive to self-loops when using \(T_{\text{sym}}\) – all changes lie within the estimated uncertainty. However, we did find that other models, e.g. GAT, perform better with self-loops (not shown).

**How to choose the coefficients \(\theta_k\)?** We found the coefficients defined by PPR and the heat kernel to be effective choices for \(\theta_k\). Fig. 8 shows that their optimal hyperparameters typically fall within a narrow range of \(\alpha \in [0.05, 0.2]\) and \(t \in [1, 10]\). We also tried obtaining \(\theta_k\) from models that learn analogous coefficients [11] [8] [13]. However, we found that \(\theta_k\) obtained by these models tend to converge to a minimal neighborhood, i.e. they converge to \(\theta_0 \approx 1\) or \(\theta_1 \approx 1\) and all other \(\theta_k \approx 0\).
Our experiments showed no correlation of improvement withWhich nodes benefit from GDC?

We propose graph diffusion convolution (GDC), a method based on sparsified generalized graph diffusion. GDC is a more powerful, yet spatially localized extension of message passing in GNNs, but able to enhance any graph-based model. We show the tight connection between GDC and spectral-based models and analyzed GDC’s spectral properties. GDC shares many of the strengths of spectral methods and none of their weaknesses. We conduct extensive and rigorous experiments that show that GDC consistently improves the accuracy of a wide range of models on both supervised and unsupervised tasks across various homophilic datasets and requires very little hyperparameter tuning. There are many extensions and applications of GDC that remain to be explored. We expect many graph-based models and tasks to benefit from GDC, e.g. graph classification and regression. Promising extensions include other diffusion coefficients $\theta_k$ such as those given by the methods presented in Fouss et al. [22] and more advanced random walks and operators that are not defined by powers of a transition matrix.

**Acknowledgments**

This research was supported by the German Federal Ministry of Education and Research (BMBF), grant no. 01IS18036B, and by the Deutsche Forschungsgemeinschaft (DFG) through the Emmy Noether grant GU 1409/2-1 and the TUM International Graduate School of Science and Engineering (IGSSE), GSC 81. The authors of this work take full responsibilities for its content.
References


