Appendix: Multi-hop Attention Graph Neural Network

A Attention Diffusion Approximation Proposition

As mentioned in Section 2.2, we use the following equation and proposition to efficiently approximate the attention diffused feature aggregation $\mathcal{A}\boldsymbol{H}^{(l)}$.

$$\begin{aligned} \boldsymbol{Z}^{(0)} &= \boldsymbol{H}^{(l)} \\ \boldsymbol{Z}^{(k+1)} &= (1-\alpha)\boldsymbol{A}\boldsymbol{Z}^{(k)} + \alpha\boldsymbol{Z}^{(0)} \end{aligned} \tag{1}$$

Proposition 1. $\lim_{K\to\infty} Z^{(K)} = \mathcal{A}\boldsymbol{H}^{(l)}$

Proof. Let $K > 0$ be the total number of iterations (i.e., hop number for graph attention diffusion) and we approximate $\hat{\boldsymbol{H}}^{(l)}$ by $\boldsymbol{Z}^{(K)}$. After K-th iteration, we can get

$$\boldsymbol{Z}^{K} = ((1-\alpha)^K \boldsymbol{A}^K + \alpha \sum_{i=0}^{K-1} (1-\alpha)^i \boldsymbol{A}^i)\boldsymbol{H}^{(l)} \tag{2}$$

The term $(1-\alpha)^K \boldsymbol{A}^K$ converges to 0 as $\alpha \in (0,1]$ and $\boldsymbol{A}^K_{i,j} \in (0,1]$ when $K \to \infty$, and thus $\lim_{K\to\infty} Z^{(K)} = (\sum_{i=0}^{\infty} \alpha(1-\alpha)^i \boldsymbol{A}^i)\boldsymbol{H}^{(l)} = \mathcal{A}\boldsymbol{H}^{(l)}$. $\square$

B Connection to Transformer

Given a sequence of tokens, the Transformer architecture makes uses of multi-head attention between all pairs of tokens, and can be viewed as performing message-passing on a fully connected graph between all tokens. A naïve application of Transformer on graphs would require computation of all pairwise attention values. Such approach, however, would not make effective use of the graph structure, and could not scale to large graphs. In contrast, Graph Attention Network (Veličković et al., 2018) leverages the graph structure and only computes attention values and perform message passing between direct neighbors. However, it has a limited receptive field (restricted to one-hop neighborhood) and the attention score (Figure 1) that is independent of the multi-hop context for prediction.

Transformer consists of self-attention layer followed by feed-forward layer. We can organize the self-attention layer in transformer as the following:

$$\text{Attention}(\boldsymbol{Q},\boldsymbol{K},\boldsymbol{V}) = \underbrace{\text{softmax}(\frac{\boldsymbol{Q}\boldsymbol{K}^T}{\sqrt{d}})}_{\text{Attention matrix}}\boldsymbol{V} \tag{3}$$

where $\boldsymbol{Q} = \boldsymbol{K} = \boldsymbol{V}$. The softmax part can be demonstrated as an attention matrix computed by scaled dot-product attention over a complete graph[1] with self-loop. Computation of attention over complete graph is expensive, Transformers are usually limited by a fixed-length context (e.g., 512 in BERT (Devlin et al., 2019)) in the setting of language modeling, and thus cannot handle large graphs. Therefore direct application of the transformer model cannot capture the graph structure in a scalable way.

In the past, graph structure is usually encoded implicitly by special position embeddings (Zhang et al., 2020) or well-designed attention computation (Shiv & Quirk, 2019; Wang et al., 2019; Nguyen et al., 2020). However, none of the methods can compute attention between any pair of nodes at each layer.

In contrast, essentially MAGNA places a prior over the attention values via Personalized PageRank, allowing it to compute the attention between any pair of two nodes via attention diffusion, without any impact on its scalability. In particular, MAGNA can handle large graphs as they are usually quite sparse and the graph diameter is usually quite smaller than graph size in practice, resulting in very efficient attention diffusion computation.

[1]All nodes are connected with each other.

C Spectral Analysis Background and Proof for Proposition 2

Graph Fourier Transform. Suppose $\boldsymbol{A}_{N\times N}$ represents the attention matrix of graph $\mathcal{G}$ with $\boldsymbol{A}_{i,j} \geq 0$, and $\sum_{j=1}^{N} \boldsymbol{A}_{i,j} = 1$. Let $\boldsymbol{A} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^{-1}$ be the Jordan's decomposition of graph attention matrix $\boldsymbol{A}$, where $\boldsymbol{V}$ is the square $N \times N$ matrix whose i-th column is the eigenvector $\boldsymbol{v}_i$ of $\boldsymbol{A}$, and $\boldsymbol{\Lambda}$ is the diagonal matrix whose diagonal elements are the corresponding eigenvalues, i.e., $\boldsymbol{\Lambda}_{i,i} = \lambda_i$. Then, for a given vector $\boldsymbol{x}$, its *Graph Fourier Transform* (Sandryhaila & Moura, 2013a) is defined as

$$\hat{\boldsymbol{x}} = \boldsymbol{V}^{-1}\boldsymbol{x}, \tag{4}$$

where $\boldsymbol{V}^{-1}$ is denoted as graph Fourier transform matrix. The *Inverse Graph Fourier Transform* is defined as $\boldsymbol{x} = \boldsymbol{V}\hat{\boldsymbol{x}}$, which reconstructs the signal from its spectrum. Based on the graph Fourier transform, we can define a graph convolution operation on $\mathcal{G}$ as $\boldsymbol{x} \otimes_{\mathcal{G}} \boldsymbol{y} = \boldsymbol{V}(\boldsymbol{V}^{-1}\boldsymbol{x} \odot \boldsymbol{V}^{-1}\boldsymbol{y})$, where $\odot$ denotes the element-wise product.

Graph Attention Diffusion Acts as A Polynomial Graph Filter. A graph filter (Tremblay et al., 2018; Sandryhaila & Moura, 2013a;b) $\boldsymbol{h}$ acts on $\boldsymbol{x}$ as $\boldsymbol{h}(\mathit{\Lambda})\boldsymbol{x} = \boldsymbol{V}\boldsymbol{h}(\mathit{\Lambda})\boldsymbol{V}^{-1}$, where $\boldsymbol{h}(\mathit{\Lambda}) = diag(\boldsymbol{h}(\lambda_1)\cdots\boldsymbol{h}(\lambda_N))$. A common choice for $\boldsymbol{h}$ in the literature is a polynomial filter of order M, since it is linear and shift invariant (Sandryhaila & Moura, 2013a;b).

$$\boldsymbol{h}(\boldsymbol{A}) = \sum_{\ell=0}^{M} \beta_\ell \boldsymbol{A}^\ell = \sum_{\ell=0}^{M} \beta_\ell (\boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^{-1})^\ell = \boldsymbol{V}(\sum_{\ell=0}^{M} \beta_\ell \mathit{\Lambda}^\ell)\boldsymbol{V}^{-1}. \tag{5}$$

Comparing to the graph attention diffusion $\mathcal{A} = \sum_{i=0}^{\infty} \alpha(1-\alpha)^i \boldsymbol{A}^i$, if we set $\beta_\ell = \alpha(1-\alpha)^\ell$, we can view **graph attention diffusion as a polynomial filter**.

Spectral Analysis. The eigenvectors of the power matrix $\boldsymbol{A}^2$ are same as $\boldsymbol{A}$, since $\boldsymbol{A}^2 = (\boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^{-1})(\boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^{-1}) = \boldsymbol{V}\boldsymbol{\Lambda}(\boldsymbol{V}^{-1}\boldsymbol{V})\boldsymbol{\Lambda}\boldsymbol{V}^{-1} = \boldsymbol{V}\boldsymbol{\Lambda}^2\boldsymbol{V}^{-1}$. By that analogy, we can get that $\boldsymbol{A}^n = \boldsymbol{V}\boldsymbol{\Lambda}^n\boldsymbol{V}^{-1}$. Therefore, the summation of the power series of $\boldsymbol{A}$ has the same eigenvectors as $\boldsymbol{A}$. Therefore by properties of eigenvectors and Equation 3, we obtain:

Proposition 2. *The set of eigenvectors for $\mathcal{A}$ and $\boldsymbol{A}$ are the same.*

Lemma 1. *Let λ_i and $\hat{\lambda}_i$ be the i-th eigenvalues of $\boldsymbol{A}$ and $\mathcal{A}$, respectively. Then, we have*

$$\hat{\lambda}_i = \sum_{\ell=0}^{\infty} \beta_\ell \lambda_i^\ell = \sum_{\ell=1}^{\infty} \alpha(1-\alpha)^\ell \lambda_i^\ell = \frac{\alpha}{1-(1-\alpha)\lambda_i} \tag{6}$$

Proof. The symmetric normalized graph Laplacian of $\mathcal{G}$ is $\boldsymbol{L}_{sym} = \boldsymbol{I} - \boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{A}\boldsymbol{D}^{-\frac{1}{2}}$, where $\boldsymbol{D}$ = $diag([d_1, d_2, \cdots, d_N])$, and $d_i = \sum_{j=1}^{\boldsymbol{A}_{i,j}}$. As $\boldsymbol{A}$ is the attention matrix of graph $\mathcal{G}$, $d_i = 1$ and thus $\boldsymbol{D} = \boldsymbol{I}$. Therefore, $\boldsymbol{L}_{sym} = \boldsymbol{I} - \boldsymbol{A}$. Let λ_i be the eigenvalues of $\boldsymbol{A}$, the eigenvalues of the symmetric normalized Laplacian of $\mathcal{G}$ $\boldsymbol{L}_{sym}$ is $\bar{\lambda}_i = 1 - \lambda_i$. Meanwhile, for every eigenvalue $\bar{\lambda}_i$ of the normalized graph Laplacian $\boldsymbol{L}_{sym}$, we have $0 \leq \bar{\lambda}_i \leq 2$ (Mohar et al., 1991), and thus $-1 \leq \lambda_i \leq 1$. As $0 < \alpha < 1$ and thus $|(1-\alpha)\lambda_i| \leq (1-\alpha) < 1$. Therefore, $((1-\alpha)\lambda_i)^K \to 0$ when $K \to \infty$, and $\hat{\lambda}_i = \lim_{K\to\infty} \sum_{\ell=0}^{K} \alpha(1-\alpha)^\ell \lambda_i^\ell = \lim_{K\to\infty} \frac{\alpha(1-((1-\alpha)\lambda_i)^K)}{1-(1-\alpha)\lambda_i} = \frac{\alpha}{1-(1-\alpha)\lambda_i}$. □

Section 3 Further defines the eigenvalues of the Laplacian matrices, $\hat{\lambda}_i^g$ and λ_i^g respectively. They satisfy: $\hat{\lambda}_i^g = 1 - \hat{\lambda}_i$ and $\lambda_i^g = 1 - \lambda_i$, and $\lambda_i^g \in [0, 2]$ (proved by (Ng et al., 2002)).

D Graph Learning Tasks

Node classification and knowledge graph link prediction are two representative and common tasks in graph learning. We first define the task of node classification:

Definition 1. ***Node classification*** *Suppose that $\boldsymbol{X} \in \mathbb{R}^{N_n \times d}$ represents the node input features, where each row $\boldsymbol{x}_i = \boldsymbol{X}_{i:}$ is a d-dimensional vector of attribute values of node $v_i \in \mathcal{V}$ $(1 \leq i \leq N)$. $\mathcal{V}_l \subset \mathcal{V}$ consists of a set of labeled nodes, and the labels are from $\mathcal{T}$, node classification is to learn the map function $f : (\boldsymbol{X}, \mathcal{G}) \to \mathcal{T}$, which predicts the labels of the remaining un-labeled nodes $\mathcal{V}/\mathcal{V}_l$.*

Knowledge graph (KG) is a heterogeneous graph describing entities and their typed relations to each other. KG is defined by a set of entities (nodes) $v_i \in \mathcal{V}$, and a set of relations (edges) $e = (v_i, r_k, v_j)$ connecting nodes v_i and v_j via relation r_k. We then define the task of knowledge graph completion:

Definition 2. ***KG completion*** *refers to the task of predicting an entity that has a specific relation with another given entity (Wang et al., 2017), i.e., predicting head h given a pair of relation and entity (r, t) or predicting tail t given a pair of head and relation (h, r).*

E Dataset Statistics

Node classification. We show the dataset statistics of the node classification benchmark datasets in Table 1.

Table 1: Statistical Information on Node Classification Benchmarks

Name	Nodes	Edges	Classes	Features	Train/Dev/Test
Cora	2,708	5,429	7	1,433	140/500/1,000
Citeseer	3,327	4,732	6	3,703	120/500/1,000
Pubmed	19,717	88,651	3	500	60/500/1,000
ogbn-arxiv[†]	169,343	1,166,243	40	128	90,941/29,799/48,603

[†] The data is available at `https://ogb.stanford.edu/docs/nodeprop/`.

Knowledge Graph Link Prediction. We show the dataset statistics of the knowledge graph benchmark datasets in Table 2.

Table 2: Statistical Information on Benchmarks

Dataset	**#Entities**	**#Relations**	**#Train**	**#Dev**	**#Test**	**#Avg. Degree**
WN18RR	40,943	11	86,835	3034	3134	2.19
FB15k-237	14,541	237	272,115	17,535	20,466	18.17

F Knowledge Graph Training and Evaluation

Training. The standard knowledge graph completion task training procedure is as follows. We add the reverse-direction triple (t, r^{-1}, h) for each triple (h, r, t) to construct an undirected knowledge graph $\mathcal{G}$. Following the training procedure introduced in (Balazevic et al., 2019; Dettmers et al., 2018), we use 1-N scoring, i.e. we simultaneously score entity-relation pairs (h, r) and (t, r^{-1}) with all entities, respectively. We explore KL diversity loss with label smoothing as the optimization function.

Inference time procedure. For each test triplet (h, r, t), the head h is removed and replaced by each of the entities appearing in KG. Afterward, we remove from the corrupted triplets all the ones that appear either in the training, validation or test set. Finally, we score these corrupted triplets by the link prediction models and then sorted by descending order; the rank of (h, r, t) is finally scored. This whole procedure is repeated while removing the tail t instead of h. And averaged metrics are reported. We report mean reciprocal rank (MRR), mean rank (MR) and the proportion of correct triplets in the top K ranks (Hits@K) for K = 1, 3 and 10. Lower values of MR and larger values of MRR and Hits@K mean better performance.

Experimental Setup. We use the multi-layer MAGNA as encoder for both FB15k-237 and WN18RR. We randomly initialize the entity embedding and relation embedding as the input of the encoders, and set the dimensionality of the initialized entity/relation vector as 100 used in DistMult (Yang et al., 2015). We select other MAGNA model hype-parameters, including number of layers, hidden dimension, head number, top-k, learning rate, hop number, teleport probability α and dropout ratios (see the settings of these parameter in Table 5 of Appendix), by a random search during the training.

G Hyper-parameter Settings for Node Classification

The best models are selected according to the classification accuracy on the validation set by early stopping with window size 200.

For each data set, the hyper-parameters are determined by a random search (Bergstra & Bengio, 2012), including learning rate, hop number, teleport probability α and dropout ratios. The hyper-parameter search space is show in Tables 3 (for Cora, Citeseer and Pubmed) and 4 (for ogbn-arxiv).

Table 3: Hyper-parameter search space used for node classification on Cora, Citeseer and Pubmed

Hyper-parameters	Search Space	Type
Hidden Dimension	512	Fixed$^{\vdash}$
Head Number	8	Fixed
Layer Number	6	Fixed
Learning rate	$[5 \times 10^{-5}, 10^{-3}]$	Range*
Hop Number	$[2, 3, \cdots, 10]$	Choice$^{\diamond}$
Teleport probability α	$[0.05, 0.6]$	Range
Dropout (attention, feature)	$[0.1, 0.6]$	Range
Weight Decay	$[10^{-6}, 10^{-5}]$	Range
Optimizer	Adam	Fixed

$^{\vdash}$ Fixed: a constant value;

* Range: a value range with lower bound and higher bound;

$^{\diamond}$ Choice: a set of values.

Table 4: Hyper-parameter search space used for node classification on ogbn-arxiv

Hyper-parameters	Search Space	Type
Hidden Dimension	128	Fixed
Head Number	8	Fixed
Layer Number	2	Fixed
Learning rate	$[0.001, 0.01]$	Range
Hop Number	$[3, 4, 5, 6]$	Choice
Teleport probability α	$[0.05, 0.6]$	Range
Dropout (attention, feature)	$[0.1, 0.6]$	Range
Weight Decay	$[10^{-5}, 10^{-4}]$	Range
Optimizer	Adam	Fixed

H Hyper-parameter Setting for Link Prediction on KG

For each KG, the hyper-parameters are determined by a random search (Bergstra & Bengio, 2012), including number of layers, learning rate, hidden dimension, batch-size, head number, hop number, teleport probability α and dropout ratios. The hyper-parameter search space is show in Table 5.

Table 5: Hyper-parameter search space used for link prediction on KG

Hyper-parameters	Search Space	Type
Initial Entity/Relation Dimension	100	Fixed
Number of layers	$[2, 3]$	Choice
Learning rate	$[10^{-4}, 5 \times 10^{-3}]$	Range
Hidden Dimension	$[256, 512, 768]$	Choice
Batch size	$[1024, 2048, 3072]$	Choice
Head Number	$[4, 8]$	Choice
Hop Number	$[2, 3, 4, 5, 6]$	Choice
Teleport probability α	$[0.05, 0.6]$	Range
Dropout (attention, feature)	$[0.1, 0.6]$	Range
Weight Decay	$[10^{-10}, 10^{-8}]$	Range
Optimizer	Adam	Fixed

Table 6: Comparison of Diffusion GCN and MAGNA for Node classification accuracy on Cora, Citeseer, Pubmed.

Model	Cora	Citeseer	Pubmed
Diffusion-GCN (PPR) (Klicpera et al., 2019)	83.6 ± 0.2	73.4 ± 0.3	$78.7 \pm 0.4^{\star}$
Diffusion-GCN (PPR) + LayerNorm + FF$^{\diamond}$	83.4 ± 0.4	72.3 ± 0.4	78.1 ± 0.5
MAGNA	**85.4** ± 0.6	**73.7** ± 0.5	**81.4** ± 0.2

$^{\diamond}$: FF: Feed Forward (FF) layer, and our implementation is based on Diffusion GCN in pytorch geometric (`https://github.com/rusty1s/pytorch_geometric`).

*: The best number derived from Diffusion GCN with "PPR". This is different from the number in Table 1 of main body, which comes from Diffusion GCN with "Heat".

I RESULTS

Comparison to Diffusion GCN. NEW

Diffusion-GCN (Klicpera et al., 2019) is also based on Personal Page-Rank (PPR) propagation. Specifically, it performs propagation over the adjacent matrix. Comparing to MAGNA, there is no LayerNorm and Feed Forward layers in standard Diffusion GCN. To clarify how much of the gain in performance depends on the page-rank-based propagation compared to the attention propagation in MAGNA, we add the two modules: LayerNorm and Feed Forward layer, into Diffusion-GCN, and conduct experiments over Cora, Citeseer and Pubmed, respectively. Table 6 shows the comparison results. From this table, we observe that adding layer normalization and feed forward layer gets the similar GNN structure to MAGNA, but does not benefit for node classification. And this implies that the PPR based attention propagation in MAGNA is more effective than PPR propagation over adjacent matrix in Diffusion GCN. NEW

Meanwhile, we also conduct depth analysis over Diffusion-GCN in Fig. 1 over the dataset "Cora" to compare with MAGNA and GAT. From this figure, we observe that deep Diffusion GCN (even with residual connection) suffers from degrading performance as well, due to the over-smoothing problem (Li et al., 2018). This is because of that, Diffusion GCN first applies Personal Page-Rank over the adjacent matrix to get a dense connected graph, and then performs top-k or threshold selection to a sparse graph. Finally, it makes use of the standard GCN over the sparse graph for node classification. In other words, the sparse graph construction is independent of GCN learning, and thus Diffusion GCN still suffers from over-smoothing.

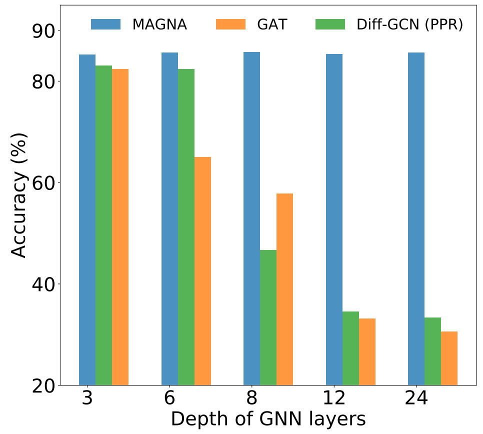

Figure 1: Model Depth Analysis on Cora dataset w.r.t. MAGNA, GAT and Diffusion-GCN (with residual connection), respectively.

Comparison over Random Splitting. NEW

We also randomly split nodes in each graph for training, validation and testing following the same ratio in the standard data splits. We conduct experiments with MAGNA, Diffusion GCN (PPR) and GAT over 10 different random splits over Cora, Citeseer and Pubmed, respectively. For each algorithm, we apply random search (Bergstra & Bengio, 2012) for hype-parameter tuning. We report the

Table 7: Results of MAGNA on Citeseer, Cora and Pubmed with random splitting. The baselines are Diffusion GCN with PPR and GAT.

Model	Cora	Citeseer	Pubmed
Diffusion GCN (PPR) (Klicpera et al., 2019)	83.6 $\pm$ 1.8	71.7 $\pm$ 1.1	79.6 $\pm$ 2.5
Diffusion GCN (PPR) + (LayerNorm & FF)	83.3 $\pm$ 1.9	69.7 $\pm$ 1.2	79.7 $\pm$ 2.3
GAT (Veličković et al., 2018)	82.2 $\pm$ 1.4	71.0 $\pm$ 1.2	78.3 $\pm$ 3.0
GAT + (LayerNorm & FF)	82.5 $\pm$ 2.2	69.5 $\pm$ 1.1	78.2 $\pm$ 2.0
MAGNA	**83.6** $\pm$ 1.2	**72.1** $\pm$ 1.2	**80.3** $\pm$ 2.4

average classification accuracy with standard deviation over 10 different splits in Table 7. From this table, we observe that MAGNA gets the best average accuracy over all data sets. However, it is noted that the standard deviation is quite large comparing to the that from the standard split in Table 6. This means that the performance is quite sensitive to the graph split. This observation is consistent wit the conclusion in (Weihua Hu, 2020): random splitting is often problematic in graph learning as they are impractical in real scenarios, and can lead to large performance variation. Therefore, in our main results, we use the standard split and also record standard deviation to demonstrate significance of the results.

REFERENCES

Ivana Balazevic, Carl Allen, and Timothy Hospedales. Tucker: Tensor factorization for knowledge graph completion. In *EMNLP*, 2019.

James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *JMLR*, pp. 281–305, 2012.

Tim Dettmers, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. Convolutional 2d knowledge graph embeddings. In *AAAI*, 2018.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL*, 2019.

Johannes Klicpera, Stefan Weißenberger, and Stephan Günnemann. Diffusion improves graph learning. In *NeurIPS*, 2019.

Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *AAAI*, 2018.

Bojan Mohar, Y Alavi, G Chartrand, and OR Oellermann. The laplacian spectrum of graphs. *Graph theory, combinatorics, and applications*, pp. 871–898, 1991.

Andrew Y Ng, Michael I Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. In *NeurIPS*, 2002.

Xuan-Phi Nguyen, Shafiq Joty, Steven CH Hoi, and Richard Socher. Tree-structured attention with hierarchical accumulation. In *ICLR*, 2020.

Aliaksei Sandryhaila and José MF Moura. Discrete signal processing on graphs: Graph fourier transform. In *ICASSP*, 2013a.

Aliaksei Sandryhaila and José MF Moura. Discrete signal processing on graphs. *TSP*, pp. 1644–1656, 2013b.

Vighnesh Shiv and Chris Quirk. Novel positional encodings to enable tree-based transformers. In *NeurIPS*, 2019.

Nicolas Tremblay, Paulo Gonçalves, and Pierre Borgnat. Design of graph filters and filterbanks. In *Cooperative and Graph Signal Processing*, pp. 299–324. Elsevier, 2018.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. In *ICLR*, 2018.

Quan Wang, Zhendong Mao, Bin Wang, and Li Guo. Knowledge graph embedding: A survey of approaches and applications. *TKDE*, pp. 2724–2743, 2017.

Yaushian Wang, Hung-Yi Lee, and Yun-Nung Chen. Tree transformer: Integrating tree structures into self-attention. In *EMNLP*, 2019.

Marinka Zitnik Yuxiao Dong Hongyu Ren Bowen Liu Michele Catasta Jure Leskovec Weihua Hu, Matthias Fey. Open graph benchmark: Datasets for machine learning on graphs. *arXiv preprint arXiv:2005.00687*, 2020.

Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. Embedding entities and relations for learning and inference in knowledge bases. In *ICLR*, 2015.

Jiawei Zhang, Haopeng Zhang, Li Sun, and Congying Xia. Graph-bert: Only attention is needed for learning graph representations. *arXiv preprint arXiv:2001.05140*, 2020.