Relative and Absolute Location Embedding for Few-Shot Node Classification on Graph

Zemin Liu,[1] **Yuan Fang,**[1] **Chenghao Liu,**[1,2] **Steven C.H. Hoi**[1,2]
[1] Singapore Management University, Singapore; [2] Salesforce Research Asia, Singapore
{zmliu, yfang}@smu.edu.sg, {chenghao.liu, shoi}@salesforce.com

Abstract

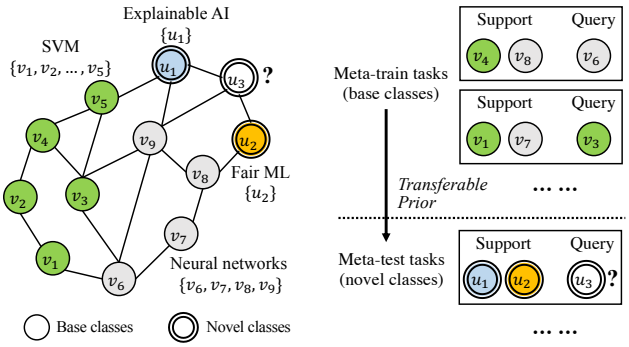
Node classification is an important problem on graphs. While recent advances in graph neural networks achieve promising performance, they require abundant labeled nodes for training. However, in many practical scenarios there often exist *novel classes* in which only one or a few labeled nodes are available as supervision, known as few-shot node classification. Although meta-learning has been widely used in vision and language domains to address few-shot learning, its adoption on graphs has been limited. In particular, graph nodes in a few-shot task are not independent and relate to each other. To deal with this, we propose a novel model called Relative and Absolute Location Embedding (RALE) hinged on the concept of *hub* nodes. Specifically, RALE captures the *task-level* dependency by assigning each node a *relative location* within a task, as well as the *graph-level* dependency by assigning each node an *absolute location* on the graph to further align different tasks toward learning a transferable prior. Finally, extensive experiments on three public datasets demonstrate the state-of-the-art performance of RALE.

Introduction

Real-world entities often interact with each other and connect into complex graphs, such as e-commerce graphs and citation networks. Many analytical tasks on such networks can be formulated as instances of node classification, *e.g.*, predicting user intent on an e-commerce graph, and inferring paper topic on a citation network. As a consequence, the problem of node classification on graphs has attracted a surge of research interest.

Traditional approaches for network analysis attempt to engineer and exploit informative features from the graph (Backstrom and Leskovec 2011), yet facing high cost. Recently, graph neural networks (GNNs) (Wu et al. 2020) emerge as a powerful family of graph representation learning techniques, which resort to the recursive aggregation of information from neighboring nodes. The recursive neighborhood aggregation is able to effectively exploit both graph structures and node features, and attains state-of-the-art performance on node classification.

Problem. Although effective in node classification on graphs, GNNs often require a significant amount of labeled data in order to achieve satisfactory performance. In a typical setting for node classification, we are given a set of classes (*e.g.*, topics) fixed in advance, where each node (*e.g.*, paper) belongs to one of the classes. The goal is then to predict the classes of the remaining unlabeled nodes. However, in many scenarios, we often need to deal with *novel classes* that we have not seen before. On one hand, there are some existing classes known as the *base classes* where a sufficient number of nodes per class are labeled. On the other hand, the novel classes only have one or very few labeled nodes per class as we just begin to tackle them. For instance, on a toy citation network in Fig. 1(a), while there are abundant labeled nodes for established topics like "SVM" and "Neural networks" (*i.e.*, base classes), very few labeled nodes are available for emerging topics like "Explainable AI" and "Fair ML" (*i.e.*, novel classes). In this paper, we study the problem of node classification for the novel classes, where each novel class only has one or a few labeled nodes, known as *few-shot node classification*.



(a) Base and novel classes on graph (b) Few-shot node classification

Figure 1: Illustration of few-shot node classification.

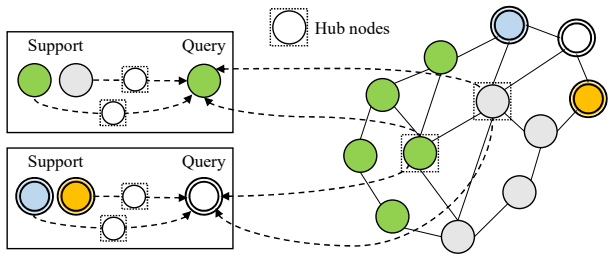
Prior work. Unfortunately, manually labeling a large number of papers in the novel classes is not only expensive but also infeasible as there may not be enough papers in these emerging topics. To rapidly enable novel classes without a large number of examples, *meta-learning* (Vinyals et al. 2016; Snell, Swersky, and Zemel 2017; Finn, Abbeel, and Levine 2017) has become a popular solution, which learns a transferable prior from the base classes with sufficient examples, and adapts the prior to the novel classes.

While meta-learning has demonstrated promising results for few-shot learning in vision and language domains, its adoption in graph data has still been limited. Some recent studies (Zhou et al. 2019; Yao et al. 2020) on few-shot node classification follow the meta-learning paradigm. Specifically, they formulate few-shot node classification as a series of classification tasks. In each task, they aim to predict the class of the *query* node after adapting the transferable prior on only a few *support* nodes, as shown in Fig. 1(b). The tasks are further split into meta-train and meta-test tasks that encompass the base and novel classes, respectively. However, in existing approaches, the dependencies between the nodes in a task are not explicitly modeled and incorporated into the transferable prior. While this is not a concern in other domains as their instances are often taken as i.i.d, graph nodes relate to each other and the inter-dependency is crucial to learning node representations.

Challenges and present work. As discussed, it is crucial to learn from the base classes how the nodes in a task depend on each other as part of the transferable prior, *i.e.*, to exploit additional dependencies based on graph structures to enhance the limited support set. However, there are two major challenges to be addressed.

First, *how do we capture the potentially long-ranged dependencies between nodes within a task?* Support and query nodes may distribute far away from each other across the graph, but state-of-the-art graph models do not capture long-range dependencies between nodes residing at distant locations on the graph. While it is natural to consider the paths between the nodes, it becomes extremely inefficient to sample paths between two distant nodes as the vast majority of random walks from one node will not reach the other within a given number of steps. Second, *how do we align the dependencies across tasks to converge on a common transferable prior?* Each task encapsulates its unique task-specific dependencies between nodes. Thus, to extract common patterns from different tasks, a global coordination is needed to align the tasks w.r.t. some invariant on the graph.

To address the two challenges, we leverage the concept of *hub* nodes (Jeh and Widom 2003; Zhu et al. 2013). Hubs are structurally important nodes on the graph, as measured by network centrality scores such as degree or PageRank (Page et al. 1999). Intuitively, hubs serve as the "backbone" of the graph that encode crucial graph information. They play *dual roles* in our solution. On one hand, to capture the dependencies between distant nodes in a task, we utilize hubs for the efficient sampling of important paths. As shown in Fig. 2(a), the dotted lines indicate hub-passing paths from the support nodes to a query node, which can be easily constructed and also carry elevated significance due to the high reachability and importance of hubs on the graph. These paths identify the dependency of a query node on the task-specific support nodes, which intuitively describe the *relative location* of the query within the task. On the other hand, to align the dependencies across tasks, we utilize hubs as a set of global references. As illustrated in Fig. 2(b), we consider the paths between a query node and the hubs over the whole graph. These paths identify the dependency of a query in any task on the graph-invariant hub nodes, which intuitively describe the *absolute location* of the query on the graph. While the task-level relative location captures node dependencies within a task, the graph-level absolute location facilitates task alignment toward a common transferable prior.



(a) Task-level relative location (b) Graph-level absolute location

Figure 2: Hub-based relative and absolute locations.

Contributions. In summary, we propose *Relative and Absolute Location Embedding* (RALE) for few-shot node classification on graphs, to effectively learn a transferable prior capturing the dependencies between nodes in few-shot tasks. Our contributions are three-fold: (1) We introduce a new use of hubs as a key enabler for the relative and absolute locations; (2) We propose a novel model RALE to explicitly learn a dependency-aware prior through location embedding at both task and graph levels; (3) We conduct extensive experiments on three public datasets, and achieve promising results over state-of-the-art approaches.

Related Work

Graph representation learning. Graph embedding has been extensively studied due to its ability to learn low-dimensional and structure-preserving representations (Perozzi, Al-Rfou, and Skiena 2014; Grover and Leskovec 2016; Cao, Lu, and Xu 2015; Yang, Cohen, and Salakhutdinov 2016; Wang, Cui, and Zhu 2016; Yang et al. 2017; Zhang et al. 2018; Liu et al. 2017, 2018; Wang et al. 2017). They typically employ a direct embedding lookup to associate nodes through various local structures such as skip-grams (Mikolov et al. 2013) and proximity (Tang et al. 2015). More recently, GNNs (Kipf and Welling 2017; Hamilton, Ying, and Leskovec 2017; Veličković et al. 2018) utilize a powerful scheme of recursive neighborhood aggregation to exploit deeper graph structures and node contents jointly. State-of-the-art GNNs often consider more complex structures (Ying et al. 2018; Morris et al. 2019; Ma et al. 2019b; Pei et al. 2020; Xu et al. 2020), various forms of convolution (Ma et al. 2019a; Jiang, Ji, and Li 2019), and graph isomorphism (You, Ying, and Leskovec 2019; Xu et al. 2019).

Meta-learning. Meta-learning (Santoro et al. 2016; Vinyals et al. 2016) has been widely used in various domains to address few-shot problems. For example, GPN (Liu et al. 2019a) resorts to prototypical meta-learning (Snell, Swersky, and Zemel 2017) for few-shot image classification; HiCE (Hu et al. 2019) leverages MAML-based meta-learning (Finn, Abbeel, and Levine 2017) for few-shot re-

gression of embedding vectors for out-of-vocabulary words. Recent studies also explore meta-learning on graphs (Garcia and Bruna 2018; Liu et al. 2019b,a; Xiong et al. 2018; Zhang et al. 2020), but they typically only use the graphs as auxiliary input to complement the original data.

Few-shot learning on graphs. While GNNs are generally semi-supervised (Kipf and Welling 2017), there exist efforts to reduce labeling requirement (Sun, Lin, and Zhu 2020) or even adopt an unsupervised paradigm (Hamilton, Ying, and Leskovec 2017; Velickovic et al. 2019). However, they do not address few-shot node classification, where novel node classes are encountered in the testing phase. Among recent few-shot learning on graphs, meta-tail2vec (Liu et al. 2020) formulates a few-shot regression problem to learn robust tail node embeddings, GSM (Chauhan, Nathani, and Kaul 2020) focuses on the few-shot graph-level classification of novel graphs, and GFL (Yao et al. 2020) explores few-shot classification on novel graphs for the same set of node classes. Finally, Meta-GNN (Zhou et al. 2019) adopts the same few-shot node classification setting in our paper, but it does not model the crucial node dependencies in each task.

Preliminaries

In this section, we introduce the problem definition and episodic meta-learning.

Few-shot node classification. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{X}, \mathcal{C}, \ell)$ denote a graph, where $\mathcal{V}$ is the set of nodes, $\mathcal{E}$ is the set of edges, $\mathbf{X} \in \mathbb{R}^{|\mathcal{V}| \times d_X}$ is the feature matrix with d_X as the number of features, $\mathcal{C}$ is the set of node classes, and $\ell : \mathcal{V} \rightarrow \mathcal{C}$ is a label function to map a node $v \in \mathcal{V}$ to its class $\ell(v) \in \mathcal{C}$.

In few-shot node classification, we have $\mathcal{C} = \mathcal{C}_b \cup \mathcal{C}_n$ where $\mathcal{C}_b$ and $\mathcal{C}_n$ are the set of base and novel classes respectively, such that $\mathcal{C}_b \cap \mathcal{C}_n = \emptyset$. A sufficient number of nodes in the base classes are labeled, *i.e.*, their label mapping by ℓ is known. Subsequently, given any subset of m novel classes, we aim to train a classifier with only a few (say, k) labeled nodes for each novel class, in order to predict the classes of the remaining unlabeled nodes among the m classes. This is called m-way k-shot node classification.

Episodic meta-learning. We adopt the episodic paradigm, which has shown great promise in few-shot learning. We employ the base classes $\mathcal{C}_b$ and novel classes $\mathcal{C}_n$ as the meta-train and meta-test sets, respectively. Both of them are formulated as a series of tasks (*i.e.*, episodes).

On one hand, to construct a meta-train task t, a subset of m classes is sampled from the base classes $\mathcal{C}_b$. t consists of the support set S_t and query set Q_t. The support set samples k nodes from each of the m classes (*i.e.*, m-way k-shot), denoted as $S_t = \{(v_{t,1}, \ell(v_{t,1})), ..., (v_{t,m\times k}, \ell(v_{t,m\times k}))\}$, while the query set $Q_t = \{(v^*_{t,1}, \ell(v^*_{t,1})), ..., (v^*_{t,n}, \ell(v^*_{t,n}))\}$ includes n different nodes from the same m classes. The support set S_t serves as the labeled training set in task t, on which the model is trained to ultimately minimize the loss of its predictions on the query set Q_t. Given a series of meta-train tasks, the goal is to learn a transferable prior that can be applied to meta-test tasks.

On the other hand, a meta-test task t' is sampled from the novel classes $\mathcal{C}_n$ in a similar m-way k-shot fashion. There is also a support set $S_{t'}$ consisting of k labeled nodes from each of the m novel classes, as well as a query set $Q_{t'} = \{v^*_{t',1}, ..., v^*_{t',n}\}$ consisting of n *unlabeled* nodes from the same m classes. The prior learned from meta-train is updated on the support set $S_{t'}$ for adapting to the m novel classes, and the adapted model is used for prediction on the query set $Q_{t'}$.

Proposed Model: RALE

We first give an overview of the proposed model RALE, as illustrated in Fig. 3. Given a graph and few-shot task in Fig. 3(a), we resort to relative and absolute location embedding to capture and align node dependencies in Fig. 3(b). Specifically, we sample a set of paths based on hubs at both the task and graph levels, and employ a graph encoder and a path encoder to model the dependencies through the sampled paths. Finally, in Fig. 3(c), dependency-aware few-shot node classification is performed in a meta-learning setup.

In the following, we introduce an abstraction of location embedding, which is then materialized at the task and graph levels using hubs. Finally, we present the overall dependency-aware meta-objective.

Abstraction of Location Embedding

We design the general methodology of embedding the location of a node $v \in \mathcal{V}$ w.r.t. an arbitrary set of reference nodes $\mathcal{R} \subset \mathcal{V}$. Let $\mathbf{e}^{\mathcal{R}}_v \in \mathbb{R}^{d_l}$ denotes the d_l-dimensional location embedding vector of v w.r.t. $\mathcal{R}$, which captures the dependency of v on $\mathcal{R}$ in the context of the graph.

Naturally, a *graph encoder* such as state-of-the-art GNNs is able to contextualize nodes on the graph. However, GNNs only explicitly preserve the structures between nodes that are within a small number of hops of each other, as deep GNNs often cause the "over-smoothing" effect (Li, Han, and Wu 2018; Xu et al. 2018; Pei et al. 2020). In other words, GNNs cannot directly model the dependency between two potentially distant nodes on the graph, which motivated us to further exploit a *path encoder*. More specifically, the dependency between any two nodes v and u can be derived from the paths connecting them. Let $\mathcal{P}_{u,v}$ be the set of paths, subject to some maximum length, between u and v. Then, the location embedding can be produced by

$$\mathbf{e}^{\mathcal{R}}_v = \phi(\{\mathcal{P}_{u,v} : u \in \mathcal{R}\}; \theta_g, \theta_p), \tag{1}$$

where ϕ is the embedding function. Specifically, ϕ requires as input a set of path sets between the target node v and each reference node $u \in \mathcal{R}$, and is parameterized by the graph encoder weights θ_g and path encoder weights θ_p. In the following, we introduce the details of the encoders.

Graph encoder. We employ GNNs (Wu et al. 2020) to contextualize nodes on a graph. Let $\phi_g(\cdot; \theta_g)$ denote a GNN encoder parameterized by θ_g. Most GNNs follow a neighborhood aggregation scheme: each node receives messages from its neighbors recursively in multiple layers. Specifically, in each layer of $\phi_g(v; \theta_g)$,

$$\mathbf{h}^i_v = \mathcal{M}(\mathbf{h}^{i-1}_v, \{\mathbf{h}^{i-1}_u, \forall u \in \mathcal{N}_v\}; \theta_g), \tag{2}$$

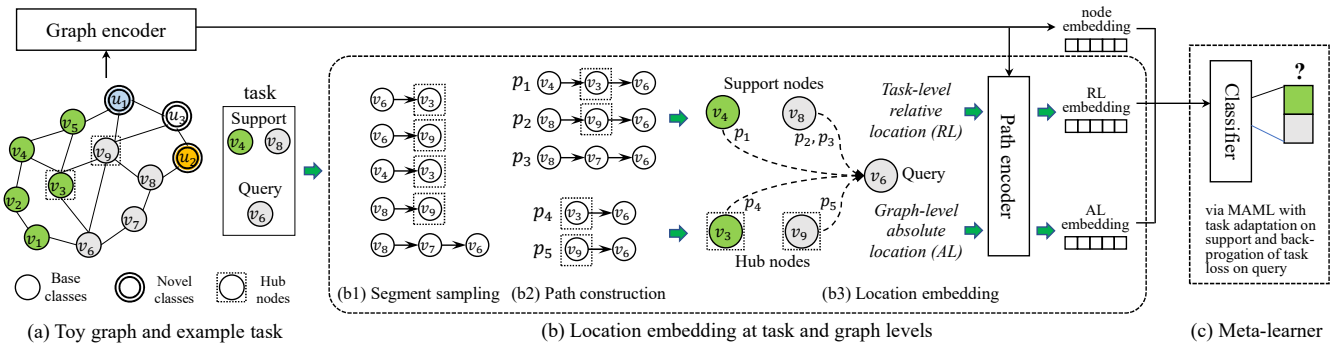


Figure 3: Overview of the proposed model RALE.

where $\mathbf{h}_v^i \in \mathbb{R}^{d_i}$ is the d_i-dimensional embedding vector of node v in the i-th layer, $\mathcal{N}_v$ is the set of neighbors of v, and $\mathcal{M}(\cdot)$ is the message passing function for neighborhood aggregation. Note that the input layer is given by the raw node features, *i.e.*, $\mathbf{h}_v^0 \equiv \mathbf{x}_v$; for the last layer, we simply write the output as $\phi_g(v;\theta_g) = \mathbf{h}_v \in \mathbb{R}^{d_g}$.

Path encoder. Given a path $p = (v_1, v_2, ..., v_s)$ on the graph, the graph encoder outputs a corresponding sequence of representations $P = \phi_g(p;\theta_g) = (\mathbf{h}_{v_1}, \mathbf{h}_{v_2}, ..., \mathbf{h}_{v_s})$, noting that the function ϕ_g is applied to each node in the sequence. The path encoder takes in the representation sequence P and produces a path embedding $\mathbf{p} \in \mathbb{R}^{d_p}$ for path p, as follows.

$$\mathbf{p} = \phi_p(P;\theta_p), \tag{3}$$

where $\phi_p(\cdot;\theta_p)$ is a sequence model parameterized by θ_p, such as recurrent neural networks (Hochreiter and Schmidhuber 1997; Chung et al. 2014), and self-attention with positional encoding (Vaswani et al. 2017).

Location embedding. Using the graph and path encoders, we generate the location embedding for node v w.r.t. a single reference node u based on the set of paths $\mathcal{P}_{u,v}$, as follows.

$$\mathbf{e}_v^{\{u\}} = \textsc{Aggr}(\{\phi_p(\phi_g(p;\theta_g);\theta_p) : \forall p \in \mathcal{P}_{u,v}\}), \tag{4}$$

where $\textsc{Aggr}(\cdot)$ is an aggregation function such as mean-pooling. Subsequently, given a set of reference nodes $\mathcal{R}$, the overall location embedding of v can be computed as

$$\begin{aligned}\mathbf{e}_v^{\mathcal{R}} &= \phi(\{\mathcal{P}_{u,v} : u \in \mathcal{R}\};\theta_g,\theta_p) \\ &= \textsc{Aggr}(\{\mathbf{e}_v^{\{u\}} : \forall u \in \mathcal{R}\}).\end{aligned} \tag{5}$$

Task and Graph-level Dependencies with Hubs

A unique nature of graph data is the dependencies between nodes, which do not follow the usual i.i.d. assumption. In a few-shot task, the support and query nodes may reside at distant locations on the graph. To capture long-range dependencies between distant nodes, We resort to location embedding materialized at both the task and graph levels by exploiting *hub* nodes (Jeh and Widom 2003; Zhu et al. 2013). Hubs are nodes with high network centrality scores such as degree or PageRank (Page et al. 1999), and thus are important nodes underpinning the graph structure. Let $\mathcal{H} \subset \mathcal{V}$ denote the set of hubs consisting of the top $|\mathcal{H}|$ nodes ranked by some centrality measure. In particular, hubs play two crucial roles at the task and graph levels, respectively.

Task-level relative location. Consider a few-shot task $t = (S_t, Q_t)$. To capture the dependencies between the nodes in task t, we assign each node v a relative location (RL) within the task, taking the task-specific support nodes S_t as the reference nodes. Specifically, $\forall v \in S_t \cup Q_t$, its RL embedding w.r.t. S_t is denoted $\mathbf{e}_v^{S_t} \in \mathbb{R}^{d_r}$, given by

$$\mathbf{e}_v^{S_t} = \phi(\{\mathcal{P}_{s,v} : s \in S_t\};\theta_g,\theta_p). \tag{6}$$

The set of paths $\mathcal{P}_{s,v}$ can be sampled by performing random walks starting from the support node s and ending at the target node v, subject to a maximum path length l_p. However, when s and v are distant to each other on the graph, such naïve random walks can be remarkably inefficient, as the vast majority of walks starting from s will fail to reach v in l_p steps. To improve the sampling efficacy, we consider paths passing through hubs, which have two major benefits. Firstly, they can be *efficiently* constructed, since hubs are easily reachable on the graph due to their high network centrality. Secondly, they tend to have elevated *significance*, due to the presence of structurally important hubs.

Specifically, we first sample path segments up to length $l_p/2$, starting from s and v, respectively. For instance, given a support node $s = v_8$ and target node $v = v_6$, in Fig. 3(b1) we sample path segments from v_8 (*e.g.*, $v_8 \to v_9$, $v_8 \to v_7 \to v_6$), as well as from v_6 (*e.g.*, $v_6 \to v_3$ and $v_6 \to v_9$). While very few of these segments would end at v_6 or v_8 directly, a significant number of them would end at a hub such as v_3 or v_9. Among the segments ending at a hub, those starting from s and from v can be further joined on the common hubs, which readily results in more paths between s and v up to length l_p. In our example, as v_3 and v_9 are hubs, the segments can join into a path p_2: $v_8 \to v_9 \to v_6$ between v_8 and v_6, as shown in Fig. 3(b2).

Graph-level absolute location. While RL embedding models node dependencies specific to each task, in order to learn a common transferable prior, the task-level dependencies must be aligned across tasks at the graph level. To this end, we use the hubs $\mathcal{H}$ as the set of global references which are invariant on the graph, and assign every node an absolute location (AL) on the graph. Specifically, $\forall v \in \mathcal{V}$, its AL embedding w.r.t. $\mathcal{H}$ is denoted $\mathbf{e}_v^{\mathcal{H}} \in \mathbb{R}^{d_a}$, given by

$$\mathbf{e}_v^{\mathcal{H}} = \phi(\{\mathcal{P}_{h,v} : h \in \mathcal{H}\};\theta_g,\theta_p). \tag{7}$$

In particular, we need to sample paths between each hub h and the target node v as input for the path encoder. As hubs

are easily reachable from most parts of the graph, there is no efficiency concern using straightforward random walks. In fact, we can simply re-use the path segments that end at a hub, which are already available from RL embedding.

Dependency-aware Meta-Objective

We adopt the episodic meta-learning framework of MAML (Finn, Abbeel, and Levine 2017) for few-shot node classification. MAML is flexible to work with any model with gradient-based optimization, and learns a prior to enable rapid adaptation to new tasks with only a few gradient updates. More specifically, the prior is adapted on each task t by updating a few gradient steps w.r.t. the loss on its support nodes S_t, and the task loss calculated on its query nodes Q_t is then backpropagated to optimize the prior.

Dependency-aware classification layer. As illustrated in Fig. 3(c), for node classification, we utilize not only the node embedding vector to capture graph context, but also the RL and AL embedding vectors to capture long-range dependencies. Consider a target node v in task t; v can be either a support or query node whose loss is to be computed. Let $\psi(\cdot;\Theta)$ denote the classification layer, as follows.

$$\psi(v;\Theta) = \textsc{Softmax}\left(\sigma(\mathbf{W}\left[\mathbf{h}_v \| \mathbf{e}_v^{S_t} \| \mathbf{e}_v^{\mathcal{H}}\right])\right), \quad (8)$$

where a softmax classifier is used for m-way classification in task t. Here $\|$ is the concatenation operator to augment the node embedding $\mathbf{h}_v \in \mathbb{R}^{d_g}$ with RL embedding $\mathbf{e}_v^{S_t} \in \mathbb{R}^{d_r}$ and AL embedding $\mathbf{e}_v^{\mathcal{H}} \in \mathbb{R}^{d_a}$, $\mathbf{W} \in \mathbb{R}^{m\times(d_g+d_r+d_a)}$ is a weight matrix, $\sigma(\cdot)$ is an activation function (*e.g.*, ELU), and $\Theta = \{\mathbf{W}, \theta_g, \theta_p\}$ denotes all the learnable parameters.

Meta-objective. To adapt to task t, we employ the cross-entropy loss on its support set S_t among the m classes in the task. Mapping the m classes to indices $\{1, 2, \ldots, m\}$, we calculate the support loss $L(S_t;\Theta)$ as

$$L(S_t;\Theta) = -\sum_{v\in S_t}\sum_{i=1}^{m} \mathbb{I}_{\ell(v)=i} \ln(\psi(v;\Theta)[i]), \quad (9)$$

where $\mathbb{I}$ is an indicator function, and $\psi(v;\Theta)[i]$ is the i-th element of $\psi(v;\Theta) \in \mathbb{R}^m$. By updating the model parameters Θ with one (or a few) gradient step w.r.t. the support loss, we obtain the parameters Θ' that is adapted to the task:

$$\Theta' = \Theta - \alpha\frac{\partial L(S_t;\Theta)}{\partial\Theta}, \quad (10)$$

where α is the learning rate of the adaptation.

The task loss can be further calculated on the query set Q_t using the updated model Θ', *i.e.*, $L(Q_t;\Theta')$. Therefore, given meta-train tasks $\mathcal{T}_{tr}$, the overall meta-objective can be optimized as follows.

$$\Theta^* = \arg\min_\Theta \sum_{t\in\mathcal{T}_{tr}} L(Q_t;\Theta - \alpha\frac{\partial L(S_t;\Theta)}{\partial\Theta}). \quad (11)$$

The learned prior Θ^* is further adapted to support set $S_{t'}$ of each meta-test task $t' \in \mathcal{T}_{ts}$, before testing on query set $Q_{t'}$.

Algorithm and complexity. The training algorithm consists of two stages. The first stage involves the sampling of path segments and the construction of paths, which only needs to be done once as precomputation. The second stage optimize the meta-objective over the meta-train tasks, which can reuse the paths constructed earlier. The pseudocode and complexity analysis are included in the supplementary.

Table 1: Summary of datasets.

	Nodes	Edges	Features	Classes (Train/Val/Test)
Amazon	13,381	245,778	767	10 (5/2/3)
Email	909	13,733	128	28 (15/6/7)
Reddit	231,371	11,606,876	602	41 (25/6/10)

Experiments

In this section, we conduct a comprehensive empirical evaluation on RALE, including a comparison with state-of-the-art approaches and model analysis.

Experimental Setup

Datasets. We employ three public real-world datasets in our experiments. (1) ***Amazon*** (Hou et al. 2020) is an e-commerce network, in which each node is an item and each edge denotes the co-purchasing relationship by a common user. (2) ***Email*** (Yin et al. 2017) is a communication network between members from a large research institution. Each node is a member and each edge denotes an email exchange between two members. (3) ***Reddit*** (Hamilton, Ying, and Leskovec 2017) is a social network, in which each node is a discussion post and each edge denotes that two posts are commented by a common user. We summarize the datasets in Table 1. More details and any processing on the datasets are described in the supplementary.

Training and testing. We randomly split the node classes into training (*i.e.*, the base classes), validation and testing (*i.e.*, the novel classes), as presented in Table 1. In particular, the base classes are used to construct meta-train tasks to learn a transferable prior, whereas the novel classes are used to construct meta-test tasks for evaluation. The random splitting is repeated five times, and for each split four random initializations are used for training. We report the average classification accuracy over these repetitions.

Baselines. We compare with competitive baselines from the following three categories.

- GNNs: We experiment with three established GNN architectures, namely, ***GCN*** (Kipf and Welling 2017), ***GraphSAGE*** (Hamilton, Ying, and Leskovec 2017) and ***GAT*** (Veličković et al. 2018). For each architecture, we follow the same train and test strategy. During training, we optimize a GNN model for node classification on all base classes; during testing, we deploy the same GNN reusing and fixing the trained parameters of neighborhood aggregation layers, but updating the parameters of the classification layer on the support set of each new task before predicting the classes of the query set.
- GNN+'s: We consider the same three architectures, resulting in ***GCN+***, ***GraphSAGE+***, ***GAT+***. Each architecture also employs a GNN model pre-trained on the base classes; during testing, we use all trained parameters as initialization only, and fine-tune them on the support set before making predictions on the query set.
- Meta-learning models: We compare with ***Meta-GNN*** (Zhou et al. 2019), which trains a GNN for few-shot

Table 2: Accuracy (percent) of RALE and baselines. In each row, the best result is bolded and the second best is underlined. RALE's improvement is calculated relative to the best baseline, with the corresponding p-value under two-tail paired t-test.

		GCN	GraphSAGE	GAT	GCN+	GraphSAGE+	GAT+	Meta-GNN	Proto-GNN	RALE	impr.	p-value
Amazon (2-way)	1-shot	71.97	63.19	72.61	70.86	68.36	66.99	<u>73.20</u>	68.91	**78.07**	+6.65%	0.007
	3-shot	<u>78.67</u>	63.15	78.18	72.66	69.75	76.07	74.45	76.41	**84.17**	+6.99%	0.021
	5-shot	79.46	64.71	<u>85.18</u>	79.84	68.44	**85.45**	76.58	79.67	84.50	-1.11%	0.437
Email (5-way)	1-shot	43.15	40.25	43.11	47.15	43.48	40.82	<u>47.21</u>	35.47	**51.82**	+9.76%	< 0.001
	3-shot	53.29	45.56	41.06	50.51	44.28	35.83	<u>55.64</u>	40.93	**59.47**	+6.88%	0.002
	5-shot	58.19	48.38	37.24	58.37	47.01	33.52	<u>58.76</u>	42.38	**65.46**	+11.40%	< 0.001
Reddit (5-way)	1-shot	20.02	41.61	19.95	20.13	35.89	19.99	<u>46.42</u>	44.95	**48.63**	+4.76%	< 0.001
	3-shot	20.16	47.00	20.15	20.21	49.11	20.02	51.28	<u>51.60</u>	**56.85**	+10.17%	< 0.001
	5-shot	20.40	50.53	20.12	20.41	51.30	20.14	<u>53.33</u>	52.57	**57.45**	+7.73%	0.029

node classification under the MAML-based meta-learning framework. We also implement prototypical networks (Snell, Swersky, and Zemel 2017) on graphs, called ***Proto-GNN***. Both models do not capture long-range node dependencies in tasks. For all meta-learning models including our RALE, we choose GraphSAGE as the GNN architecture as it performs better than other architectures when coupled with these models. For RALE, we use self-attention (Vaswani et al. 2017) as the path encoder.

Parameters and settings. For each method, we tune and select their hyperparameters and settings based on validation and guidance from the literature.

For all GNNs and GNN+'s, we employ two layers, and further tune the dropout probability to 0.6, the dimension of the hidden layers to 32, as well as the learning rates for base class training and novel class updating or fine-tuning to 0.001 and 0.01, respectively. For specific architectures, we use 8 heads for the multi-head attention mechanism in GAT, and apply the mean aggregator for GraphSAGE.

For all meta-learning methods, we use the same parameters in the above for the base GNN architecture, with a meta-learning rate of 0.001. We also tune the dropout rate, and find that Meta-GNN and Proto-GNN is optimal when no dropout is used, whereas for RALE we use the dropout rate of 0.4 on the Amazon dataset and 0.5 on others. For MAML-based approaches, we further tune the number of gradient updates when adapting to the support set, which is set to 2 for Meta-GNN and 1 for RALE, as well as the learning rate of adaptation α, which is set to 5.0 for both on the Amazon and Reddit datasets, and 0.2 on the Email dataset.

Finally, we report the default settings for hubs and random walks in our model RALE. We rank all nodes by their PageRank scores in descending order, and choose the top 5% nodes as hubs. The 5% here is called the *hub ratio*, *i.e.*, $|\mathcal{H}|/|\mathcal{V}|$. To sample paths for a given task, we perform $w = 200$ random walks of length $l = 50$ starting from each node in the task (Perozzi, Al-Rfou, and Skiena 2014). A sliding window of $l_p/2$ is applied on each sampled walk to extract path segments. Segments ending with a hub are further joined to form paths up to length $l_p = 6$. We will further investigate the impact of these parameters in our experiments.

Performance Comparison

Table 2 shows the performance comparison with baselines on few-shot node classification, using 1, 3 and 5 shots on each datasets. We make the following observations.

Firstly, between each GNN and its GNN+ counterpart, no consistent winner emerge. While GNN+'s with additional fine-tuning on the neighborhood aggregation layers can potentially benefit from the support set, they may overfit to the support set due to the small number of shots. The results imply that straightforward fine-tuning cannot work very well in few-shot settings.

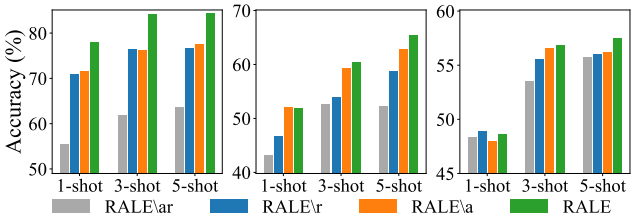
Secondly, Meta-GNN and Proto-GNN can only achieve comparable or slightly better performance than GNNs and GNN+'s. This demonstrates that, although meta-learning is promising in few-shot learning, without modeling node dependencies its advantage is rather limited on graph data due to the non-i.i.d. nodes.

Thirdly, RALE outperforms all baselines with statistical significance, demonstrating the benefit of capturing and aligning node dependencies. The only exception is on the Amazon dataset with 5 shots, where RALE is worse than GAT/GAT+, although the difference is *not* statistically significant ($p = 0.437$). The possible reason is that the co-purchasing ties between diverse items on a large e-commerce platform like Amazon are weaker than email exchanges between users or topical relatedness between posts participated in by the same user on social networks. Thus, it becomes less important to capture the dependencies, and the attention mechanism in GAT/GAT+ is more suited to weighing diverse neighbors especially when given more shots.

Lastly, as we use more shots for the support set, all meta-learning approaches steadily perform better too. In contrast, the performance of GNNs and GNNs+ fluctuates without a clear pattern. The difference implies again that GNNs and GNN+'s are not designed to work on few-shot setting, where a modest increase in the number of shots does not help much to improve their performance.

Model Analysis

We analyze the behavior of our model RALE in several aspects, including an ablation study to show the contribution from different modules, an investigation on the hubs and random walks, as well as an analysis of parameter sensitivity.



(a) Amazon (b) Email (c) Reddit

Figure 4: Ablation study.

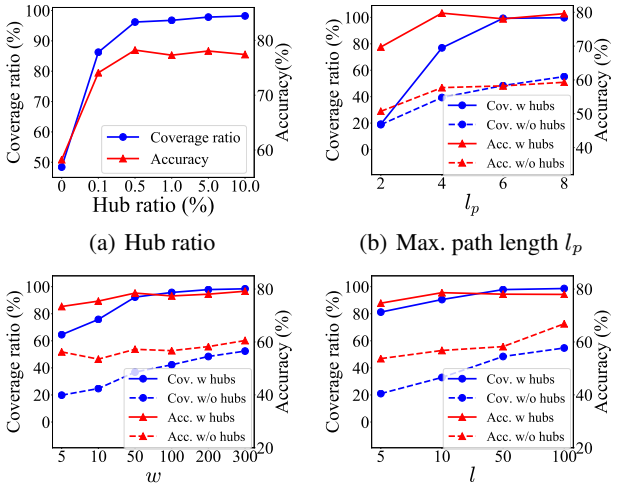
Ablation study. We compare RALE with its three degenerate versions, including the version without the task-level RL embedding (RALE\r), without graph-level AL embedding (RALE\a), and without both (RALE\ar).

From Fig. 4, several observations can be made. Firstly, RALE\ar has the lowest accuracy, which is not surprising as it does not model any long-range dependency. Secondly, both RALE\r and RALE\a perform better than RALE\ar, and RALE\a is often better than RALE\r. Intuitively, there is little for AL embedding to align across tasks without RL embedding within tasks, whereas RL embedding captures task-specific dependencies which can still be distilled to some extent even without explicit global alignment. Lastly, the full model RALE consistently achieves the best performance, showing the necessity to not only capture dependencies in individual tasks but also align them on the graph.

Hubs and random walks. In the following, we vary one parameter at a time and fix the others as their default values given in experimental setup. We only show the 1-shot results on Amazon datasets, as other datasets show similar trends.

In RALE, hubs facilitate path sampling within tasks and global coordination across tasks. With no or few hubs, it becomes unlikely to sample enough paths of significance between two distant nodes, causing low *coverage ratio* (*i.e.*, fraction of node pairs within tasks that are connected by sampled paths). In Fig. 5(a), when we increase the hub ratio, the coverage ratio and the corresponding classification accuracy both increase and converge around hub ratio of 1–5%. The strong correlation between the coverage ratio and accuracy show that our hub-based paths is the key to effectively capturing dependencies.

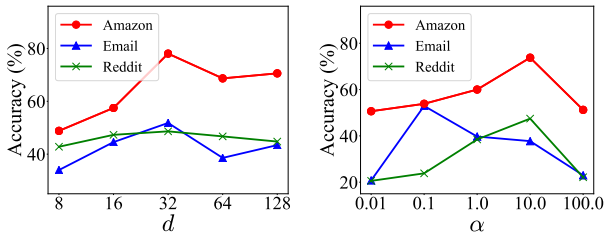
Next, we study the impact of random walk parameters. We also compare to the alternative strategy without hubs, *i.e.*, direct path sampling instead of constructing from hub-ending segments. As shown in Figs. 5(b)–(d), increasing maximum path length l_p, number of walks per node w and walk length l all lead to consistently higher coverage ratio and classification accuracy whether using hubs or not, due to more random walk samples. However, under the same settings, using hubs give much higher coverage and accuracy than without hubs, showing the efficacy of our hub-based strategy. In particular, with hubs the coverage and accuracy will converge much quicker (*e.g.*, $l_p \sim 6$, $w \sim 100$, $l \sim 50$), whereas without using hubs much more random walks (*i.e.*, much larger l_p, w, l) are needed to achieve similar coverage and accuracy. Note that more random walks require more computational time, as further discussed in the supplementary.



(a) Hub ratio (b) Max. path length l_p

(c) No. of walks per node w (d) Walk length l

Figure 5: Analysis of hubs and random walks.



(a) Embedding dimension d (b) Adaptation learning rate α

Figure 6: Analysis of parameter sensitivity.

Parameter sensitivity. We study the sensitivity of two hyperparameters in 1-shot setting. In Fig. 6(a), we vary the embedding dimension d of our encoders between 8 and 128 (all encoders use the same d). While a very small or large d results in inferior performance, $d = 32$ appears to be a robust setting across datasets. In Fig. 6(b), we vary the learning rate of adaptation α between 0.01 and 100. We observe that $[0.1, 10]$ is generally a good range for α on our datasets, although they may have different optimal values that could be attributed to the various relationships between tasks.

Conclusion

In this paper, we studied few-shot node classification on graph, to deal with novel classes with limited labeled nodes. While meta-learning has been widely used on few-shot visual and language tasks, nodes on a graph are fundamentally different for their non-i.i.d. nature. Unfortunately, existing meta-learning approaches do not explicitly capture the potentially long-ranged dependencies between nodes in a task. Thus, we propose RALE, a novel meta-learning model for few-shot node classification through hub-based relative and absolute location embedding. While relative locations capture the task-level dependency between nodes within a task, absolute locations capture the graph-level dependency to align the tasks. Extensive experiments show that RALE significantly outperforms state-of-the-art baselines.

Acknowledgments

This research/project is supported by the National Research Foundation, Singapore under its AI Singapore Programme (AISG Award No: AISG-RP-2018-001).

References

Backstrom, L.; and Leskovec, J. 2011. Supervised random walks: predicting and recommending links in social networks. In *WSDM*, 635–644.

Cao, S.; Lu, W.; and Xu, Q. 2015. GraRep: Learning graph representations with global structural information. In *CIKM*, 891–900.

Chauhan, J.; Nathani, D.; and Kaul, M. 2020. Few-Shot Learning on Graphs via Super-Classes based on Graph Spectral Measures. In *ICLR*.

Chung, J.; Gulcehre, C.; Cho, K.; and Bengio, Y. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. In *NeurIPS Deep Learning and Representation Learning Workshop*.

Finn, C.; Abbeel, P.; and Levine, S. 2017. Model-agnostic meta-learning for fast adaptation of deep networks. In *ICML*, 1126–1135.

Garcia, V.; and Bruna, J. 2018. Few-shot learning with graph neural networks. In *ICLR*.

Grover, A.; and Leskovec, J. 2016. node2vec: Scalable feature learning for networks. In *KDD*, 855–864.

Hamilton, W.; Ying, Z.; and Leskovec, J. 2017. Inductive representation learning on large graphs. In *NeurIPS*, 1024–1034.

Hochreiter, S.; and Schmidhuber, J. 1997. Long short-term memory. *Neural computation* 9(8): 1735–1780.

Hou, Y.; Zhang, J.; Cheng, J.; Ma, K.; Ma, R. T.; Chen, H.; and Yang, M.-C. 2020. Measuring and Improving the Use of Graph Information in Graph Neural Networks. In *ICLR*.

Hu, Z.; Chen, T.; Chang, K.-W.; and Sun, Y. 2019. Few-Shot Representation Learning for Out-Of-Vocabulary Words. In *ACL*, 4102–4112.

Jeh, G.; and Widom, J. 2003. Scaling personalized web search. In *WWW*, 271–279.

Jiang, X.; Ji, P.; and Li, S. 2019. CensNet: Convolution with Edge-Node Switching in Graph Neural Networks. In *IJCAI*, 2656–2662.

Kipf, T. N.; and Welling, M. 2017. Semi-supervised classification with graph convolutional networks. In *ICLR*.

Li, Q.; Han, Z.; and Wu, X.-M. 2018. Deeper insights into graph convolutional networks for semi-supervised learning. In *AAAI*, 3538–3545.

Liu, L.; Zhou, T.; Long, G.; Jiang, J.; and Zhang, C. 2019a. Learning to propagate for graph meta-learning. In *NeurIPS*, 1039–1050.

Liu, Y.; Lee, J.; Park, M.; Kim, S.; Yang, E.; Hwang, S. J.; and Yang, Y. 2019b. Learning to propagate labels: Transductive propagation network for few-shot learning. *ICLR* .

Liu, Z.; Zhang, W.; Fang, Y.; Zhang, X.; and Hoi, S. C. 2020. Towards locality-aware meta-learning of tail node embeddings on networks. In *CIKM*, 975–984.

Liu, Z.; Zheng, V. W.; Zhao, Z.; Zhu, F.; Chang, K. C.-C.; Wu, M.; and Ying, J. 2017. Semantic proximity search on heterogeneous graph by proximity embedding. In *AAAI*, 154–160.

Liu, Z.; Zheng, V. W.; Zhao, Z.; Zhu, F.; Chang, K. C.-C.; Wu, M.; and Ying, J. 2018. Distance-Aware DAG Embedding for Proximity Search on Heterogeneous Graphs. In *AAAI*, 2355–2362.

Ma, J.; Cui, P.; Kuang, K.; Wang, X.; and Zhu, W. 2019a. Disentangled Graph Convolutional Networks. In *ICML*, 4212–4221.

Ma, Y.; Wang, S.; Aggarwal, C. C.; and Tang, J. 2019b. Graph Convolutional Networks with EigenPooling. In *KDD*, 723–731.

Mikolov, T.; Sutskever, I.; Chen, K.; Corrado, G. S.; and Dean, J. 2013. Distributed representations of words and phrases and their compositionality. In *NeurIPS*, 3111–3119.

Morris, C.; Ritzert, M.; Fey, M.; Hamilton, W. L.; Lenssen, J. E.; Rattan, G.; and Grohe, M. 2019. Weisfeiler and leman go neural: Higher-order graph neural networks. In *AAAI*, volume 33, 4602–4609.

Page, L.; Brin, S.; Motwani, R.; and Winograd, T. 1999. The PageRank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab.

Pei, H.; Wei, B.; Chang, K. C.-C.; Lei, Y.; and Yang, B. 2020. Geom-GCN: Geometric graph convolutional networks. In *ICLR*.

Perozzi, B.; Al-Rfou, R.; and Skiena, S. 2014. DeepWalk: Online learning of social representations. In *KDD*, 701–710.

Santoro, A.; Bartunov, S.; Botvinick, M.; Wierstra, D.; and Lillicrap, T. 2016. Meta-learning with memory-augmented neural networks. In *ICML*, 1842–1850.

Snell, J.; Swersky, K.; and Zemel, R. 2017. Prototypical networks for few-shot learning. In *NeurIPS*, 4077–4087.

Sun, K.; Lin, Z.; and Zhu, Z. 2020. Multi-Stage Self-Supervised Learning for Graph Convolutional Networks on Graphs with Few Labels. In *AAAI*, 5892–5899.

Tang, J.; Qu, M.; Wang, M.; Zhang, M.; Yan, J.; and Mei, Q. 2015. LINE: Large-scale information network embedding. In *WWW*, 1067–1077.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention is all you need. In *NeurIPS*, 5998–6008.

Veličković, P.; Cucurull, G.; Casanova, A.; Romero, A.; Lio, P.; and Bengio, Y. 2018. Graph attention networks. In *ICLR*.

Velickovic, P.; Fedus, W.; Hamilton, W. L.; Liò, P.; Bengio, Y.; and Hjelm, R. D. 2019. Deep Graph Infomax. In *ICLR*.

Vinyals, O.; Blundell, C.; Lillicrap, T.; Wierstra, D.; et al. 2016. Matching networks for one shot learning. In *NeurIPS*, 3630–3638.

Wang, D.; Cui, P.; and Zhu, W. 2016. Structural deep network embedding. In *KDD*, 1225–1234.

Wang, J.; Zheng, V. W.; Liu, Z.; and Chang, K. C.-C. 2017. Topological recurrent neural network for diffusion prediction. In *ICDM*, 475–484.

Wu, Z.; Pan, S.; Chen, F.; Long, G.; Zhang, C.; and Philip, S. Y. 2020. A comprehensive survey on graph neural networks. *TNNLS* (Early Access).

Xiong, W.; Yu, M.; Chang, S.; Guo, X.; and Wang, W. Y. 2018. One-shot relational learning for knowledge graphs. In *EMNLP*, 1980–1990.

Xu, C.; Cui, Z.; Hong, X.; Zhang, T.; Yang, J.; and Liu, W. 2020. Graph Inference Learning for Semi-supervised Classification. In *ICLR*.

Xu, K.; Hu, W.; Leskovec, J.; and Jegelka, S. 2019. How powerful are graph neural networks? In *ICLR*.

Xu, K.; Li, C.; Tian, Y.; Sonobe, T.; Kawarabayashi, K.-i.; and Jegelka, S. 2018. Representation learning on graphs with jumping knowledge networks. In *ICML*, 5453–5462.

Yang, C.; Sun, M.; Liu, Z.; and Tu, C. 2017. Fast network embedding enhancement via high order proximity approximation. In *IJCAI*, 3894–3900.

Yang, Z.; Cohen, W. W.; and Salakhutdinov, R. 2016. Revisiting semi-supervised learning with graph embeddings. In *ICML*, 40–48.

Yao, H.; Zhang, C.; Wei, Y.; Jiang, M.; Wang, S.; Huang, J.; Chawla, N. V.; and Li, Z. 2020. Graph Few-shot Learning via Knowledge Transfer. In *AAAI*, 6656–6663.

Yin, H.; Benson, A. R.; Leskovec, J.; and Gleich, D. F. 2017. Local higher-order graph clustering. In *KDD*, 555–564.

Ying, Z.; You, J.; Morris, C.; Ren, X.; Hamilton, W.; and Leskovec, J. 2018. Hierarchical graph representation learning with differentiable pooling. In *NeurIPS*, 4800–4810.

You, J.; Ying, R.; and Leskovec, J. 2019. Position-aware Graph Neural Networks. In *ICML*, 7134–7143.

Zhang, C.; Yao, H.; Huang, C.; Jiang, M.; Li, Z.; and Chawla, N. V. 2020. Few-Shot Knowledge Graph Completion. In *AAAI*, 3041–3048.

Zhang, Z.; Cui, P.; Wang, X.; Pei, J.; Yao, X.; and Zhu, W. 2018. Arbitrary-order proximity preserved network embedding. In *KDD*, 2778–2786.

Zhou, F.; Cao, C.; Zhang, K.; Trajcevski, G.; Zhong, T.; and Geng, J. 2019. Meta-GNN: On Few-shot Node Classification in Graph Meta-learning. In *CIKM*, 2357–2360.

Zhu, F.; Fang, Y.; Chang, K. C.; and Ying, J. 2013. Incremental and Accuracy-Aware Personalized PageRank through Scheduled Approximation. *PVLDB* 6(6): 481–492.