SCIENCE CHINA
Information Sciences

• RESEARCH PAPER •

December 2023, Vol. 66 222101:1–222101:11
https://doi.org/10.1007/s11432-021-3443-y

# Learning discrete adaptive receptive fields for graph convolutional networks

Xiaojun MA[1†], Ziyao LI[1†], Guojie SONG[1*] & Chuan SHI[2]

[1]*School of Intelligence Science and Technology, Peking University, Beijing* 100871*, China;*
[2]*School of Computer Science, Beijing University of Posts and Telecommunications, Beijing* 100876*, China*

Received 10 April 2021/Revised 30 November 2021/Accepted 16 February 2022/Published online 8 November 2023

**Abstract** Different nodes in a graph neighborhood generally yield different importance. In previous work of graph convolutional networks (GCNs), such differences are typically modeled with attention mechanisms. However, as we prove in our paper, soft attention weights suffer from undesired smoothness large neighborhoods (not to be confused with the oversmoothing effect in deep GCNs). To address this weakness, we introduce a novel framework of conducting graph convolutions, where nodes are discretely selected among multi-hop neighborhoods to construct adaptive receptive fields (ARFs). ARFs enable GCNs to get rid of the smoothness of soft attention weights, as well as to efficiently explore long-distance dependencies in graphs. We further propose GRARF (GCN with reinforced adaptive receptive fields) as an instance, where an optimal policy of constructing ARFs is learned with reinforcement learning. GRARF achieves or matches state-of-the-art performances on public datasets from different domains. Our further analysis corroborates that GRARF is more robust than attention models against neighborhood noises.

**Keywords** graph neural networks, receptive field, reinforcement learning

**Citation** Ma X J, Li Z Y, Song G J, et al. Learning discrete adaptive receptive fields for graph convolutional networks. Sci China Inf Sci, 2023, 66(12): 222101, https://doi.org/10.1007/s11432-021-3443-y

## 1 Introduction

After a series of explorations and modifications [1–6], graph convolutional networks (GCNs)[1)] have gained considerable attention in the machine learning community. Typically, a graph convolutional model can be abstracted as a message-passing process [7] — nodes in the neighborhood of a central node are regarded as contexts, who individually pass their messages to the central node via convolutional layers. The central node then weighs and transforms these messages. This process is recursively conducted as the depth of network increases. We use the term contexts to denote the neighbor nodes, and receptive field to denote the set of contexts that the convolutions refer to.

Neighborhood convolutions proved to be widely useful on various graph data. However, some inconveniences also exist in current GCNs. While different nodes may yield different importance in the neighborhood, early GCNs [2, 8] did not discriminate contexts in their receptive fields. These models either treated contexts equally, or used normalized edge weights as the weights of contexts. As a result, such implementations failed to capture critical contexts — contexts that pose greater influences on the central node, close friends among acquaintances, for example. Graph attention networks (GATs) [3] resolved this problem with attention mechanisms [9, 10]. Soft attention weights were used to discriminate importance of contexts, which allowed the model to better focus on relevant contexts to make decisions. With impressive performances, GATs became widely used in later generations of GCNs including [5, 11]. However, we observe that using soft attention weights in hierarchical convolutions does not fully solve the problem. Firstly, we will show as Proposition 1 that under common conditions, soft attention weights

* Corresponding author (email: gjsong@pku.edu.cn)
† Authors Ma X J and Li Z Y have the same contribution to this work.
1) We use the name GCN for a class of deep learning approaches where information is convolved among graph neighborhoods, including but not limited to the vanilla GCN [2].

 

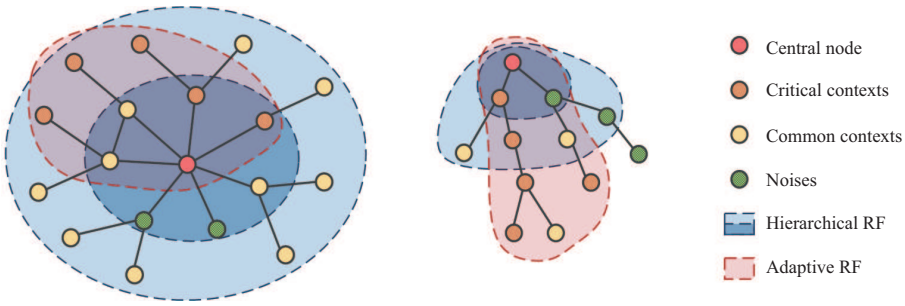


**Figure 1** (Color online) Comparison between hierarchical convolutions and convolutions with ARF. Left: GCNs with ARFs better focus on critical nodes and filter out noises in large neighborhoods. Right: ARFs more efficiently explore LDDs.

almost surely approach 0 as the neighborhood sizes increase. This smoothness hinders the discrimination of context importance in large neighborhoods. Secondly, we will show by experiments in Subsection 4.2 that GATs cannot well distinguish true graph nodes from artificial noises: attention weights assigned to true nodes and noises are almost identical in distribution, which further leads to a dramatic drop of performance.

Meanwhile, an ideal GCN architecture is often expected to exploit information on nodes with various distances. Most existing GCNs use hierarchical convolutional layers, in which only one-hop neighborhoods are convolved. As a result, one must increase the model depth to detect long-distance dependencies (LDDs) (informative nodes that are distant from the central nodes). This is particularly an issue in large graphs, as the complexity of the graph convolutions is exponential to the model depth[2)]. In large graphs, the model depths are often set as 1, 2 or 3 [3, 8]. Accordingly, no dependencies longer than 3 hops are exploited in these models.

Motivated by the discussions above, we propose the idea of adaptive receptive fields (ARFs). Figure 1 illustrates the differences between hierarchical convolutions and convolutions with ARFs. An ARF is defined as a subset of contexts that are most informative for a central node, and is constructed via selecting contexts among the neighborhood. Nodes in an ARF can be at various distances from the central node. The discrete selection process of contexts gets rid of the undesired smoothness of soft weights (see Section 2). In addition, by allowing ARFs to choose contexts on different hops from the central node, one can efficiently explore dependencies with longer distances. Experiments also show that ARFs are more robust to noises (see Section 4). We further propose GRARF (GCNs with reinforced adaptive receptive fields) as an instance for using ARFs in node-level tasks. In GRARF, an optimal policy of constructing ARFs is learned with reinforcement learning (RL). An RL agent (constructor) successively expands the ARF via a two-stage process: a contact node in the intermediately-constructed ARF is firstly selected; a context among the direct neighbors of the contact node is then added to the ARF. The reward of the constructor is defined as the performance of a trained GCN (evaluator) on the constructed ARF.

GRARF is validated on datasets from different domains including three citation networks, one social network, and an inductive protein-protein interaction dataset. GRARF matches or improves performances on node classification tasks compared with strong baselines[3)]. Moreover, we design two tasks to test the models' abilities in focusing on informative contexts and leveraging LDDs by injecting node noises in graphs with different strategies.

## 2 Preliminaries and theories

**Notations.** In our paper, we consider node-level supervised learning tasks on attributed graphs. An attributed graph $G$ is generally represented as $G = (V, A, X)$, where $V = \{v_1, \ldots, v_n\}$ denotes the set of nodes, $A \in \{0, 1\}^{n \times n}$ denotes the (binary) adjacency matrix, and $X \in \mathbb{R}^{n \times d_0}$ denotes the input node features, $x_v \in \mathbb{R}^{d_0}$ the features of node $v$. $E$ is used as the set of edges. We use $N(v_i)$ to denote the one-hop neighborhood of node $v_i$, with $v_i$ itself included. We use $H^{(l)} \in \mathbb{R}^{n \times d_l}$ as the matrix containing

2) With sparse adjacency matrices, the average complexity of graph convolutions is $O(d^L)$, where $L$ is the model depth and $d$ is the graph degree (or the neighborhood-sampling sizes in [8]).

3) We mainly show the results of node classification tasks in our paper, whereas GRARF is intrinsically adapted to all node-level supervised learning tasks.

$d_l$-dimensional hidden representations of nodes in the $l$-th layer, $h_v^{(l)}$ that of node $v$. $\hat{A}$ denotes the symmetrically normalized adjacency matrix with $\hat{A} = D^{-1/2}(A + I_n)D^{-1/2}$ and $D = \mathrm{diag}(d), d_i = \sum_j (A + I_n)_{ij}$. We use bold letters for neural network parameters.

**The smoothness of GATs.** As a pioneering work of simplifying architectures of graph neural networks, the vanilla GCN layers in [2] were defined as follows:

$$H^{(l+1)} = \sigma(\hat{A}H^{(l)}\boldsymbol{W}^{(l)}) = \sigma\left(\sum_{j\in N(v_i)} \hat{A}_{ij} h_j^{(l)} \boldsymbol{W}^{(l)}\right), \quad l = 0, 1, \ldots. \tag{1}$$

In each layer, the node representations in one-hop neighborhoods were transformed with $\boldsymbol{W}$ and averaged by normalized edge weights $\hat{A}_{ij}$. GATs [3] elaborated the average scheme in (1) with attention mechanisms [9,10]. Instead of using $\hat{A}_{ij}$, an attention weight $\alpha_{ij}$ between node $v_i$ and $v_j$ was calculated in GAT layers as

$$e_{ij} = f_\theta\left(h_i, h_j\right), \quad \alpha_{ij} = \mathrm{softmax}_j\left(e_{ij}\right) = \frac{\exp(e_{ij})}{\sum_{k\in N(v_i)} \exp(e_{ik})}, \tag{2}$$

where $f_\theta(\cdot)$ is often called the energy function with parameter $\theta$. GATs implicitly enabled specifying different weights in a neighborhood. However, under some common assumptions and as the neighborhood size increases, these attention weights, normalized with the softmax function, suffer from undesired smoothness: all attention weights approach 0 as the neighborhood size increases. We formally introduce and prove this claim as Lemma 1 and Proposition 1.

**Lemma 1** (Smoothness of softmax)**.** If random variables $X_1, X_2, \ldots$ are uniformly bounded with probability 1, that is, for any $i$ and some $C$, $P(|X_i| > C) = 0$, then the softmax values taking $\{X_i\}_{i=1}^n$ as inputs approach 0 almost surely when $n \to \infty$, i.e.,

$$\mathrm{e}^{X_i} \Big/ \sum_{j=1}^{n} \mathrm{e}^{X_j} \to 0 \quad \text{a.s.} \tag{3}$$

*Proof.* The proof is simple noting that $\mathrm{e}^{X_i}/\sum_{j=1}^n \mathrm{e}^{X_j} > 0$, and that with probability 1,

$$\mathrm{e}^{X_i} \Big/ \sum_{j=1}^{n} \mathrm{e}^{X_j} < \mathrm{e}^{C} \Big/ n\mathrm{e}^{-C} \to 0, \quad n \to \infty.$$

**Proposition 1** (Smoothness of attention weights)**.** If the representation of nodes (random vectors) $H_1, H_2, \ldots \in \mathbb{R}^d$ are uniformly bounded with probability 1 (for any $i$ and some $C$, $P(\|H_1\| > C) = 0$), and for any (fixed) node $v_i$, the energy function $f_\theta(h_i, \cdot)$ is continuous on any closed set $D \in \mathbb{R}^d$, then the attention weights in the neighborhood of $v_i$ approach 0 almost surely when $n \to \infty$, i.e.,

$$\alpha_{ij} = \frac{\exp(f_\theta(h_i, H_j))}{\sum_{k\in N(v_i)} \exp(f_\theta(h_i, H_k))} \to 0 \quad \text{a.s.} \tag{4}$$

*Proof.* Following the a.s. boundedness $\{H_i\}$s and the continuity condition on $f_\theta(\cdot)$, the random energies $E_{ij} = f_\theta(h_i, H_j)$ are also bounded a.s.. The desired result then follows Lemma 1.

Note that the continuity condition on $f_\theta(h_i, \cdot)$ in Proposition 1 can be satisfied with almost any commonly used non-linear functions and (regularized) parameters in deep learning, specifically, those in the official version of GATs ($e_{ij} = a^{\mathrm{T}}[\boldsymbol{W}h_i \| \boldsymbol{W}h_j]$, where $\|$ is the operator of concatenation). Also, the boundedness of inputs is trivial in deep learning.

**GCNs with ARFs overcome smoothness.** What Proposition 1 shows is that in large neighborhoods, attention weights are smoothed to 0, thus hindering the discrimination of context importance. In addition, such smoothness can be immediately generated to any other form of normalized weights as long as $\alpha_{ij} > 0$ uniformly and $\sum_{j\in N(v_i)} \alpha_{ij} = 1$. We alleviate the smoothness with ARFs by incorporating discreteness. Specifically, let us denote the convolution in the evaluator as

$$h_i' = \sigma\left(\sum_{j\in N_a(u)} \eta_{ij} h_j W\right) = \sigma\left(\sum_{j\in N^k(u)} \tilde{\eta}_{ij} h_j W\right), \quad \tilde{\eta}_{ij} = \begin{cases} \eta_{ij}, & j \in N_a(u), \\ 0, & j \notin N_a(u), \end{cases} \tag{5}$$

where $N_a(u)$ is an ARF, $k$ is the maximum hop that the ARF explores, and $N^k$ is the entire $k$-hop neighborhood. Accordingly, $\tilde{\eta}_{ij}$ is not subjected to smoothness: if $\eta_{ij}$ has a uniform lower bound $D > 0$, then for $p \in N_a(u)$ and $q \notin N_a(u)$, we have $\tilde{\eta}_{ip} - \tilde{\eta}_{iq} > D$, regardless of the sizes of $N^{(k)}$. Note that $\eta_{ij} > 0$ uniformly can be guaranteed in most cases when the maximum ARF size is limited, for example, with uniform weights or softmax weights of bounded energies.

**Deep RL on graphs.** As the discrete context selection process is non-differentiable, we apply deep RL approaches to learn the policy of construting ARFs in GRARF, specifically, the deep $Q$-learning (DQN) [12] algorithm. DQN uses deep neural networks to approximate the action value function ($Q$-function), and chooses the action that maximizes it in each step. The $Q$-function is defined iteratively as follows:

$$Q^*(s_t, a_t) = R(s_t, a_t, s_{t+1}) + \gamma \max_{a \in \mathcal{A}} Q^*(s_{t+1}, a), \tag{6}$$

where $R(\cdot)$ is the reward function, $\mathcal{A}$ is the action space, and $\gamma$ is a discount factor. A reward shaping technique [13] is also used in GRARF to alleviate the sparsity of rewards, which decorates the original reward $R(\cdot)$ with a potential energy $F(\cdot)$, yielding an immediate reward $\hat{R}(\cdot)$. Denoted in formula,

$$F(s, a, s') = \Phi(s') - \Phi(s), \quad \hat{R}(s, a, s') = R(s, a, s') + F(s, a, s'), \tag{7}$$

where $\Phi(\cdot)$ is a fixed potential function of states that does not change during training. Ref. [13] proved that the optimal policies of Markov decision processes (MDPs) remain invariant if $R(\cdot)$ is replaced by $\hat{R}(\cdot)$.

There are other recent studies implementing RL on graphs. For example, graph convolutional policy network (GCPN) [14] proposed an RL agent for generating graph representations of biomedical molecules, and graph convolutional reinforcement learning (DGN) [15] introduced a multi-agent RL approach where the agents in the system formed a dynamic network. The successive molecule generation process in GCPN inspired us in designing the ARF constructor in GRARF, whereas the two models are of different motivations.

**ARFs and neighborhood sampling.** It should be noted that GRARF can also be interpreted as a neighborhood sampling approach. Neighborhood sampling was proposed as a necessary process to apply GCNs to large graphs with arbitrarily large neighborhoods. GraphSAGE [8] proposed a general framework of neighborhood sampling and aggregation, where contexts were uniformly sampled. Later work improved the sampling strategy with importance sampling [16] and explicit variance reduction [8, 17]. Sub-graphs instead of subsets of neighborhoods were directly sampled in [18]. Indeed, selecting ARF nodes takes a specific form of neighborhood sampling. However, the aim of constructing ARFs is to ignore trivial information and to focus on critical contexts, rather than to estimate the neighborhood average as is the primary target of neighborhood sampling. Therefore, despite the similarity, the two approaches are in different directions.

## 3 Proposed method: GRARF

### 3.1 ARF construction as MDP

An ARF is defined as a set of nodes $N_a(u)$ with regard to a central node $u$. Nodes in $N_a(u)$ can be at various distances from $u$, and $u$ itself should be contained in its ARF. We also assume that the induced subgraph of an ARF must be (weakly) connected, under the motivation that if a far context poses great influence on the central node, then at least one path connecting it to the central node should be included in the ARF. The ARF construction process is modeled as an MDP $M = (\mathcal{S}, \mathcal{A}, P, R, \gamma)$, where $\mathcal{S} = \{s_i\}$ is the state space of all possible ARFs; $\mathcal{A} = \{a_i\}$ describes all possible (two-stage) actions $a = (a^1, a^2)$; $P$ is the transition dynamics $p(s_{t+1}|s_t, a_t)$ which describes how nodes are added to ARFs; $R$ is the reward function of an ARF, and $\gamma$ is the discount factor.

Figure 2 introduces the general structure of our model. GRARF is composed of a constructor and an evaluator. The constructor implements an RL agent to learn an optimal policy of the MDP with DQN, and the evaluator conducts graph convolutions on constructed ARFs. Specifically, in the training phase, the constructor and the evaluator are trained alternately, where rewards of the constructor are derived from the performances of the evaluator; in the prediction phase, the evaluator convolves over the

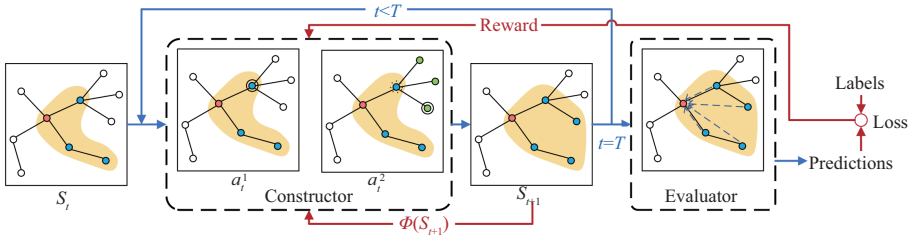


**Figure 2** (Color online) General architecture of GRARF. Red and blue arrows together demonstrate the training process. Blue arrows along demonstrate the prediction process.

ARFs constructed for the target node by the constructor and then predicts the result. No gradient flows between the constructor and the evaluator.

### 3.2 Model architecture

We hereby specify the MDP of ARF construction, the evaluator, and the training scheme.

**States.** At step $t$, the state $s_t$ is defined as the intermediately generated ARF, which is encoded as a vector representation $\boldsymbol{s}_t$ by a function $f_s$, and then observed by the constructor. Formally,

$$s_t \triangleq N_a^{(t)}(u), \quad \boldsymbol{s}_t \triangleq f_s(N_a^{(t)}(u)). \tag{8}$$

We initialize the ARF as $N_a^{(0)}(u) = \{u\}$. The transition dynamics in GRARF is deterministic: nodes selected by the RL agent is added to the ARF, i.e., $N_a^{(t+1)}(u) = N_a^{(t)}(u) \cup a_t^2$.

**Actions.** For each action, the constructor chooses a node to add to the ARF among all nodes adjacent to the ARF. The average complexity of directly choosing among all adjacent candidates is $O(n_t \times d)$, where $n_t$ is the ARF size at step $t$ and $d$ is the graph degree. We reduce the complexity to $O(n_t + d)$ by decomposing the action into two stages, denoted as $a_t = (a_t^1, a_t^2)$. In the first stage, the constructor chooses a contact node $a_t^1 \in N_a^{(t)}(u)$, who limits the candidates of the next stage in its own direct neighborhood. In the second stage, the constructor searches among the neighborhood of $a_t^1$ for a node $a_t^2$ to add to the ARF. The optimal $Q$-function can accordingly be rewritten as

$$Q_1^*(s_t, a_t^1) = \max_{a_2 \in N(a_t^1)} Q_2^*(s_t, a_2), \tag{9}$$

$$Q_2^*(s_t, a_t^2; a_t^1) = R(s_t, a_t^2, s_{t+1}) + \gamma \max_{a_1 \in N_a^{(t+1)}(u)} Q_1^*(s_{t+1}, a_1). \tag{10}$$

We do not design explicit stop actions, and the process stops after a fixed number of steps ($T$). Meanwhile, we do not require that $a_t^2 \notin N_a^{(t)}(u)$, so a node may be selected multiple times in an ARF. Note that if a node already in the ARF is selected, the state ceases to change, and no new nodes will be selected. Therefore, various ARF sizes are implicitly allowed. The actions (i.e., candidate nodes) in both stages are encoded with $f_a$, and the approximated $Q$-function in GRARF is parameterized as

$$Q_1(s_t, a_t^1) = \boldsymbol{w}_1^{\mathrm{T}} \left[ f_a(a_t^1) \| f_s(s_t) \right] + \boldsymbol{b}_1, \tag{11}$$

$$Q_2(s_t, a_t^2; a_t^1) = \boldsymbol{w}_2^{\mathrm{T}} \left[ f_a(a_t^1) \| f_a(a_t^2) \| f_s(s_t) \right] + \boldsymbol{b}_2. \tag{12}$$

**Rewards.** The reward of the constructor is defined as the performance of the GCN evaluator. As conducting step-wise evaluations is much time-costly, we sample the reward once the ARF is fully constructed, that is,

$$R(s_t, a_t^2, s_{t+1}) \triangleq \begin{cases} 0, & t < T, \\ \mathrm{eval}(N_a^{(t)}(u)) = -\mathrm{loss}(\mathrm{GCN}(N_a^{(t)}(u))), & t = T. \end{cases} \tag{13}$$

The loss in (13) is specified by the downstream supervised node-level (or node-pair-level) tasks, such as node classification and link prediction. We further adopt the reward shaping technique [13] to guide and accelerate the training process. Considering the desired properties of critical contexts and ARFs, we propose the following heuristic potential functions: (i) $\Phi_1(s) = |N_a^{(t)}(u)|$, the sizes of ARFs, to encourage

the variety of contexts; (ii) $\Phi_2(s) = \sum_{v\in N_a(u)} \deg(v)$, the sum of degrees in the ARF, as nodes with higher degrees are intuitively more informative; (iii) $\Phi_3(s) = \sum_{v\in N_a(u)} \mathrm{sim}(v,u) = \sum_{v\in N_a(u)} x_v \cdot x_u$, the inner product of input features between the central node and the ARF nodes, to empirically encourage more relevant contexts. According to (7) with so-defined $\Phi$s, the immediate rewards of the constructor is

$$\hat{R}(s_t, a_t^2, s_{t+1}) = \begin{cases} 0, & t < T \quad \text{and} \quad a_t^2 \in N_a^{(t)}(u), \\ 1 + \deg(a_t^2) + x_{a_t^2} \cdot x_u, & t < T \quad \text{and} \quad a_t^2 \notin N_a^{(t)}(u), \\ -\mathrm{loss}(\mathrm{GCN}(N_a^{(t)}(u))), & t = T. \end{cases} \tag{14}$$

**Evaluator.** Given a central node $u$ and the constructed ARF $N_a(u)$, the evaluator takes the ARF as neighborhood and convolves over it, generating the representation of the central node. An example of evaluator would be the one defined in (5). The representation is then used to conduct downstream tasks. Theoretically, any graph convolutional layers can be used as the evaluator. It is worth noting that although in our experiments we only perform a one-layer graph convolution on the constructed ARF, multiple convolutional layers can be applied on the subgraph induced by the ARF node set. We leave this as future work.

**Training.** In order to mutually train the constructor and the evaluator in GRARF, we propose an alternate training strategy, somehow analogous to the training of GAN [19]. Specifically, the evaluator is first pre-trained for the given downstream task, taking direct neighborhoods as receptive fields. We then fix the evaluator to derive constant task-aware rewards for the training of the constructor. The alternate process goes recursively until convergence. In details, as the training of the evaluator converges much faster than the constructor, we train the constructor with more steps in the alternate process. Empirically, 10:1 is a promising choice of the ratio.

## 4 Experiments

### 4.1 Experiment setup

**Datasets.** We evaluate GRARF on public real-world datasets including 3 citation networks, a social network, and a protein-protein interaction dataset. Some interesting statistics of the datasets are shown in Table 1. In the citation networks (`cora`, `citeseer`, and `pubmed`)[4], nodes correspond to publications in disjoint fields, and edges to (undirected) citation relationships. The social network (`github`)[5] consists of website users (nodes) and their friendships (edges). We reduce the number of input features of social networks to the figures in Table 1 by selecting most frequent ones (all features binary and sparse). The protein-protein interaction (`ppi`) dataset[6] contains 24 graphs, each representing a human tissue, where nodes denote different proteins and edges denotes the interactions in between. We use the preprocessed data of GraphSAGE [8].

**Baselines.** We pick up 5 GCN baselines to compare GRARF against, including vanilla GCN [2], GraphSAGE [8], GAT [3], GIN [4], MixHop [6], SGAT [20], GeniePath [11], and NeuralSparse [21]. For fairness, the dimensions of all hidden representations are set as $d = 128$ and an identical two-layer setup is used in all baselines. Hyper-parameters such as learning rates in all models are tuned to achieve best performances on the validation sets. All neural models are trained with adequate epochs with the early-stopping strategy. More details of baselines are included in Appendix A.

**GRARF implementation.** In the constructor, all graph nodes are first encoded with a GraphSAGE layer ($h_u^c = \mathrm{elu}(\boldsymbol{W}_c[x_u \| \frac{1}{n}\sum_{v\in N(u)} x_v] + \boldsymbol{b}_c)$). For the state encoder $f_s$, we use linear transformations of concatenations of central node representations and ARF-averaged node representations, i.e., $f_s(N_a(u)) = \boldsymbol{W}_s[h_u^c \| \frac{1}{n}\sum_{v\in N_a(u)} h_v^c] + \boldsymbol{b}_s$. For the action encoder $f_a$, we directly use the hidden representations of nodes, i.e., $f_a(v) = h_v^c$. The discount factor is chosen as $\gamma = 0.9$. In the evaluator, a simple GraphSAGE layer convolves over constructed ARFs as $h_u^e = \mathrm{elu}(\boldsymbol{W}_e[x_u \| \frac{1}{n}\sum_{v\in N_a(u)} x_v] + \boldsymbol{b}_e)$. The hidden representations $h_u^e$ are later used in node-level tasks with only one fully-connected layer. The same as all baselines, dimensions of all hidden representations in GRARF ($h_u^c$, $h_u^e$, and $\boldsymbol{s}_t$) are 128. To demonstrate the effectiveness of ARFs, we also conduct experiments with a random baseline, Random RF, where actions

4) https://linqs.soe.ucsc.edu/data.
5) http://snap.stanford.edu/data/.
6) http://snap.stanford.edu/graphsage/ppi.zip.

**Table 1** Interesting statistics of the datasets used in this paper[a)]

| | cora | citeseer | pubmed | github | ppi* |
|---|---|---|---|---|---|
| # Nodes | 2708 | 3327 | 19717 | 37700 | 56944 |
| # Links | 5429 | 4732 | 44338 | 289003 | 818716 |
| # Classes | 7 | 6 | 3 | 2 | 121* |
| # Features | 1433 | 3703 | 500 | 600 | 50 |
| Average degree | 2.00 | 1.42 | 2.25 | 7.67 | 28.8 |

a) * represents multi-label task.

**Table 2** Performances of GRARF and baselines on node classification tasks (% micro-f1)

| | cora | citeseer | pubmed | github | ppi |
|---|---|---|---|---|---|
| Vanilla GCN [2] | $87.40 \pm 0.36$ | $78.09 \pm 0.35$ | $86.18 \pm 0.10$ | $81.72 \pm 0.14$ | $59.26 \pm 0.03$ |
| GraphSAGE [8] | $82.02 \pm 0.31$ | $70.23 \pm 0.51$ | $86.26 \pm 0.33$ | $78.09 \pm 1.23$ | $56.97 \pm 0.94$ |
| GAT [3] | $87.80 \pm 0.11$ | $76.43 \pm 0.73$ | $85.39 \pm 0.12$ | $81.78 \pm 0.52$ | $47.29 \pm 2.30$ |
| GIN [4] | $85.08 \pm 0.83$ | $73.87 \pm 0.52$ | $84.88 \pm 0.17$ | $79.05 \pm 0.88$ | $60.35 \pm 1.14$ |
| MixHop [6] | $\mathbf{88.70 \pm 0.19}$ | $74.59 \pm 0.27$ | $85.60 \pm 0.10$ | $79.68 \pm 1.00$ | $58.55 \pm 0.14$ |
| SGAT [20] | $87.96 \pm 0.99$ | $76.55 \pm 0.45$ | $85.96 \pm 0.44$ | $82.07 \pm 1.35$ | $60.73 \pm 0.87$ |
| GeniePath [11] | $85.75 \pm 0.21$ | $76.26 \pm 2.35$ | $85.59 \pm 0.35$ | $83.51 \pm 2.44$ | $62.44 \pm 0.92$ |
| NeuralSparse [21] | $87.39 \pm 0.35$ | $77.24 \pm 0.48$ | $86.23 \pm 1.35$ | $85.38 \pm 2.36$ | $61.25 \pm 0.34$ |
| Random RF | $86.93 \pm 1.22$ | $77.31 \pm 1.37$ | $86.87 \pm 1.30$ | $79.81 \pm 7.82$ | $63.27 \pm 2.38$ |
| GRARF | $88.34 \pm 0.82$ | $\mathbf{79.24 \pm 0.88}$ | $\mathbf{88.20 \pm 0.33}$ | $\mathbf{87.30 \pm 2.54}$ | $\mathbf{66.54 \pm 1.12}$ |

are chosen completely randomly, but strictly following the setup of GRARF. More details of GRARF are included in Appendix B.

### 4.2 Results

**Performances of node classification tasks.** Table 2 shows the micro-f1s of GRARF and baselines on the node classification tasks including means and standard deviations across 10 replicas. The datasets are uniformly split to 5:2:3 as training, validation and test sets, except for `ppi`, where 20 graphs are used as training set, 2 as validation set, and 2 as test set, identical to that in [8]. Tasks are transductive on citation and social network and inductive on `ppi`. GRARF shows very competitive performances on both types of tasks: on citation graphs where neighborhoods are small (degree $\approx 2$), GRARF matches or improves by margin the performances of other baselines; on `github` and `ppi` datasets with larger neighborhoods (degree $> 7$), GRARF displays a significant advantages over other baselines. This is in concordance with our expectations, as we have shown that GRARF overcomes the undesired smoothness and thus better focuses on informative contexts in larger neighborhoods.

**Capturing informative contexts and LDDs.** To demonstrate the benefits of using ARFs, we design two experiments through adding artifical noises in `cora` with different strategies. In the denoising setup, we generate $r \times |V|$ noise nodes and randomly connects them to true nodes. An amount of $r \times |E|$ edges between true nodes and noises are added, so that the proportion of noises in the neighborhood is approximately $\frac{r}{1+r}$. In the LDD setup, we assign a split number $k$ drawn from Poisson($\lambda$) on each edge, and then split each edge to a $(k+1)$-hop path (do nothing if $k = 0$) by inserting $k$ noises. The one-hop dependencies are hence stretched longer and more difficult for the models to detect. In both experiments, features of noises are drawn from marginal distributions of individual dimensions. We conduct experiments with 5 replicas per model and report the averaged performances and 95% confidence intervals. Figure 3 shows examples and performances of GRARF, vanilla GCNs under two setups. We also show the performances of raw-feature baseline for comparison (which fully ignores the noises).

In the denoising experiment, GRARF displays a significantly better ability in capturing informative nodes while ignoring noises. The performances of vanilla GCNs and GATs drop dramatically as $r$ increases and even below the raw baseline when $r > 1$, whereas the performances of GRARF drops only marginally. A similar phenomenon is observed in the LDD experiment, which corroborates that the constructed ARFs in GRARF better collect the information from far-hop neighbors through exploring the neighborhood in a more flexible manner.

We further analyzed the behavior of GATs and GRARF on real and noisy data. Figure 4(a) shows the distributions of attention weights that GATs assign to neighbors of central nodes (with degree $d$s) on

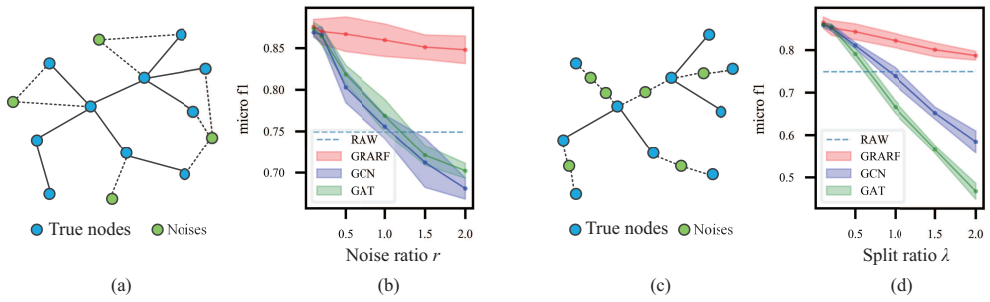


**Figure 3** (Color online) Examples and results of two designed experiments. (a), (b) The denoising experiment. (c), (d) The long-distance dependency experiment. GRARF displays robustness to noises in both experiments, whereas performances of vanilla GCNs and GATs drop dramatically.

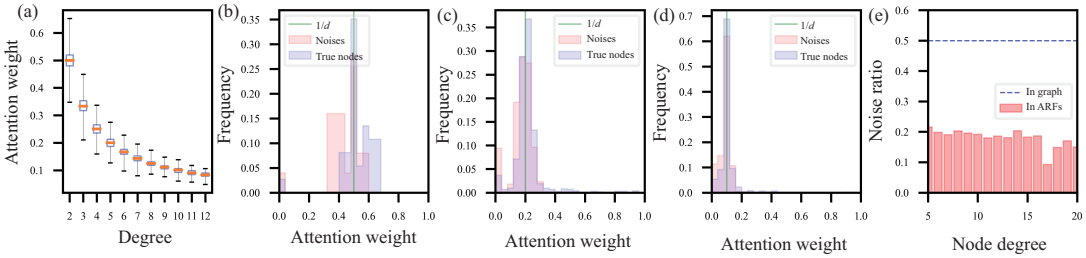


**Figure 4** (Color online) Analysis of behaviors in GAT and GRARF. (a) A box-plot of attention weights on `cora` with regard to node degrees, where medians, Q1s, Q3s, and a 95% intervals are displayed. (b)–(d) Histograms of attention weights assigned to true nodes and noises in the denoising experiments ($r = 1$), with different node degrees ($d = 2, 5, 10$). (e) The ratios of noises in constructed ARFs with regard to node degrees, in the denoising experiments ($r = 1$).

original `cora`. The majority of attention weights lie in a very thin interval around $1/d$ which continues to shrink as $d$ increases. This is empirical evidence of Proposition 1. Figures 4(b)–(d) show distributions of attention weights assigned to true nodes and noises in the denoising experiment with $r = 1$ and $d = 2, 5, 10$ (calculated in true nodes' neighborhoods only). Attention weights assigned to true nodes and noises are almost identical, especially with larger $d$. This indicates that GATs cannot well distinguish noises from true nodes with attention weights. We also report the ratios of noises in constructed ARFs in the same experiment. The noise ratios in the ARFs stay far below the noise ratio in the graph, and remain almost invariant to the sizes of neighborhoods. This suggests that the ARF constructor learns to ignore noises in the graph neighborhoods. More analysis of GRARF are included in Appendix C.

## 5 Conclusion & future work

In this paper, we proposed the idea of ARFs and GRARF as an instance. We showed both theoretically and empirically that GCNs with ARFs address the smoothness of attention weights, and hence better focus on critical contexts instead of common ones or noises. Meanwhile, as nodes in ARFs can be at various distances from the central node, GCNs with ARFs explore LDDs more efficiently. Nevertheless, the ARFs in our paper are simplified as sets of nodes, whose structures are mostly ignored. For future work, a straight direction is to learn an optimal convolutional structures on constructed ARFs, or to jointly learn both. Finding fast approximations of RL is also attracting. Another promising prospect is to provide the constructor with more global-level information, for currently constructor only observes local neighborhoods.

### References

1 Bruna J, Zaremba W, Szlam A, et al. Spectral networks and locally connected networks on graphs. In: Proceedings of the 2nd International Conference on Learning Representations, Banff, 2014
2 Kipf T N, Welling M. Semi-supervised classification with graph convolutional networks. In: Proceedings of the 5th International Conference on Learning Representations, Toulon, 2017
3 Velickovic P, Cucurull G, Casanova A, et al. Graph attention networks. 2017. ArXiv:1710.10903
4 Xu K, Hu W, Leskovec J, et al. How powerful are graph neural networks? In: Proceedings of the 7th International Conference on Learning Representations, New Orleans, 2019
5 Li Z, Zhang L, Song G. GCN-LASE: towards adequately incorporating link attributes in graph convolutional networks. In: Proceedings of the 28th International Joint Conference on Artificial Intelligence, Macao, 2019

6 Abu-El-Haija S, Perozzi B, Kapoor A, et al. Mixhop: higher-order graph convolutional architectures via sparsified neighborhood mixing. In: Proceedings of the 36th International Conference on Machine Learning, Long Beach, 2019
7 Gilmer J, Schoenholz S S, Riley P F, et al. Neural message passing for quantum chemistry. In: Proceedings of the 34th International Conference on Machine Learning, Sydney, 2017. 1263–1272
8 Hamilton W L, Ying Z, Leskovec J. Inductive representation learning on large graphs. In: Proceedings of Annual Conference on Neural Information Processing Systems, Long Beach, 2017. 1024–1034
9 Bahdanau D, Cho K, Bengio Y. Neural machine translation by jointly learning to align and translate. In: Proceedings of the 3rd International Conference on Learning Representations, San Diego, 2015
10 Vaswani A, Shazeer N, Parmar N, et al. Attention is all you need. In: Proceedings of Annual Conference on Neural Information Processing Systems, Long Beach, 2017. 5998–6008
11 Liu Z, Chen C, Li L, et al. Geniepath: graph neural networks with adaptive receptive paths. In: Proceedings of the 33rd AAAI Conference on Artificial Intelligence, Honolulu, 2019. 4424–4431
12 Mnih V, Kavukcuoglu K, Silver D, et al. Human-level control through deep reinforcement learning. Nature, 2015, 518: 529–533
13 Ng A Y, Harada D, Russell S J. Policy invariance under reward transformations: theory and application to reward shaping. In: Proceedings of the 16th International Conference on Machine Learning, Bled, 1999. 278–287
14 You J, Liu B, Ying Z, et al. Graph convolutional policy network for goal-directed molecular graph generation. In: Proceedings of Annual Conference on Neural Information Processing Systems, Montréal, 2018. 6412–6422
15 Jiang J, Dun C, Huang T, et al. Graph convolutional reinforcement learning. In: Proceedings of the 8th International Conference on Learning Representations, Addis Ababa, 2020
16 Chen J, Ma T, Xiao C. FastGCN: fast learning with graph convolutional networks via importance sampling. In: Proceedings of the 6th International Conference on Learning Representations, Vancouver, 2018
17 Huang W, Zhang T, Rong Y, et al. Adaptive sampling towards fast graph representation learning. In: Proceedings of Annual Conference on Neural Information Processing Systems, Montréal, 2018. 4563–4572
18 Zeng H, Zhou H, Srivastava A, et al. GraphSAINT: graph sampling based inductive learning method. In: Proceedings of the 8th International Conference on Learning Representations, Addis Ababa, 2020
19 Goodfellow I J, Pouget-Abadie J, Mirza M, et al. Generative adversarial networks. 2014. ArXiv:1406.2661
20 Ye Y, Ji S. Sparse graph attention networks. IEEE Trans Knowl Data Eng, 2021, doi: 10.1109/TKDE.2021.3072345
21 Zheng C, Zong B, Cheng W, et al. Robust graph representation learning via neural sparsification. In: Proceedings of International Conference on Machine Learning, 2020. 11458–11468


## Appendix A Details on experimental setup of baselines

**Code preparation.** We downloaded the code of all baselines (vanilla GCN[7)], GraphSAGE[8)], GAT[9)], GIN[10)], and MixHop[11)]) from corresponding official GitHub repositories. The original version of GIN only worked on graph classification tasks and we adapted it to node classification tasks. The multi-layer perceptrons in GIN have 2 layers.

**Hyperparameter selection.** For baselines containing recommended hyperparameters in the source code, including GAT on `cora`, MixHop, and Vanilla GCN on `cora`, `citeseer`, and `pubmed`, we directly adopted their settings. Meanwhile, we conducted additional tests to adjust their learning rate and model dimension and confirmed that their official settings are almost the best choices. Specifically, GAT had an initial learning rate of 0.005, a hidden state dimension of 8, and 8 attention heads. MixHop's initial learning rates are 1, 0.5, and 0.25 on `cora`, `citeseer`, and `pubmed`, respectively. Vanilla GCN had an initial learning rate of 0.01 and a hidden state dimension of 16[12)].

When there is no recommended setting of hyperparameters, or the settings are not affordable on our device, we performed hyperparameter selection on learning rate, dropout rate, and model dimension. We tested the initial learning rates 0.1, 0.01, 0.001, 0.0001 and the dropout rates 0.1, 0.3, 0.5, 0.7, 0.9 for all models. For fair comparison, we constrained the number of layers at 2 and the dimension of hidden states at 128. All methods used a batch size of 512. Specifically, for MixHop, we performed dropout rate selection on input and hidden layers and tested the layer capacities 18, 20, 24, 30. For GIN, we adopted GIN-0 at each layer.

## Appendix B Details on GRARF implementation

**Constructor.** We decomposed the actions into two stages. Each stage corresponded to a specific sub-constructor. In both stages, all graph nodes were first encoded with a GraphSAGE layer $h_u^c = \mathrm{elu}(\boldsymbol{W}_c[x_u \| \frac{1}{n}\sum_{v\in N_a(u)} x_v] + \boldsymbol{b}_c)$. For the state encoder $f_s$, we used linear transformations of concatenations of central node representations and ARF-averaged node representations, i.e.,

$$f_s(N_a(u)) = \boldsymbol{W}_s \left[ h_u \Big\| \frac{1}{n} \sum_{v\in N_a(u)} h_v \right] + \boldsymbol{b}_s. \tag{B1}$$

For the action encoder $f_a$, we used the hidden representations of nodes, i.e., $f_a(v) = h_c(v)$. The approximated $Q$-function in GRARF was parameterized as

$$Q_1(s_t, a_t^1) = \boldsymbol{w}_1^{\mathrm{T}}[f_a(a_t^1) \| f_s(s_t)] + \boldsymbol{b}_1, \tag{B2}$$

$$Q_2(s_t, a_t^2; a_t^1) = \boldsymbol{w}_2^{\mathrm{T}}[f_a(a_t^1) \| f_a(a_t^2) \| f_s(s_t)] + \boldsymbol{b}_2, \tag{B3}$$

where $a_t^1$ was the node selected in the first stage. In the training of the constructors, we set the discount rate as $\gamma = 0.9$, and the exploration with linear decay. After 200 steps, the exploration rate decayed to 0.05 and remained unchanged. The size of the memory pool was set as 50000. For each action, the constructor chose a node to add to the ARF among all nodes adjacent to the

7) https://github.com/tkipf/gcn.
8) https://gihub.com/williamleif/GraphSAGE.
9) https://gihub.com/PetarV-/GAT.
10) https://gihub.com/weihua916/powerful-gnns.
11) https://github.com/samihaija/mixhop.
12) We tried the hidden state dimension 128 in Vanilla GCN and the hidden state dimension 16 in GAT, while no improvement was observed.

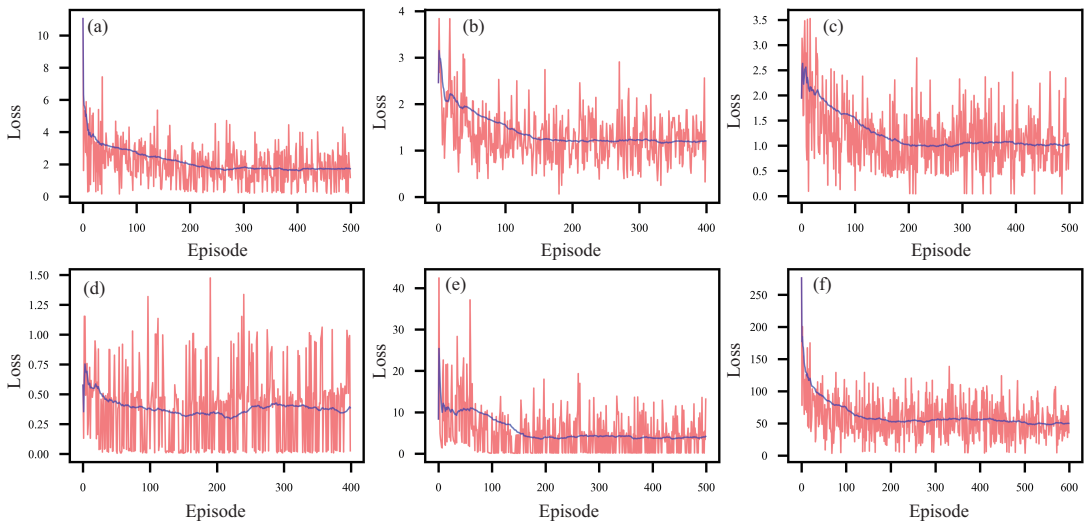


**Figure C1** (Color online) Training loss. (a)–(f) correspond to the training loss curves of `cora`, `citeseer`, `pubmed`, `github`, and `ppi` datasets, respectively. The red lines represent step loss and the blue lines represent sliding mean loss (sliding window size is 25).

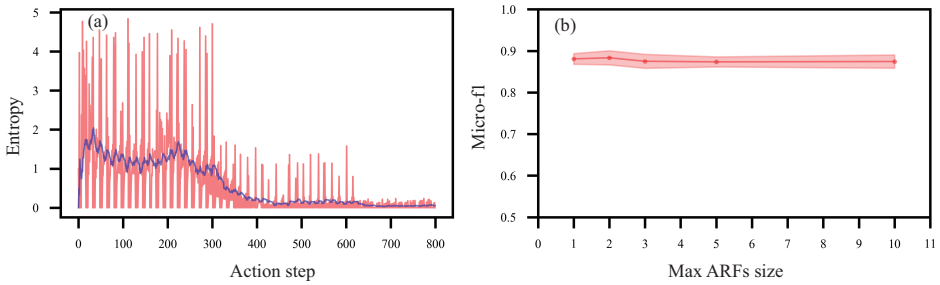


**Figure C2** (Color online) (a) Entropy of each action step. The red line is the entropy of each action step, and the blue line is the sliding mean entropy (sliding window size is 25). (b) Micro-f1s of different maximum ARFs sizes and confidence intervals.

ARF. Due to memory constraints, we limited the size of nodes adjacent to the ARF to 300 by sampling. This design was for nodes with large numbers of degrees specifically. The hidden representation of states and actions (i.e., outputs of $f_s$ and $f_a$) were defined to be 128-dimensional.

**Hyperparameter selection.** In all settings, we performed hyperparameter selection on the learning rate, training steps, maximum ARFs size, and update frequency of constructor. In the pretraining phase, we trained the evaluator with the same training nodes as in GRARF. The initial learning rate in the pretraining phase was 0.001. In the training phase for GRARF, we performed a parameter sweep on initial learning rates over $0.01, 0.001, 0.0001$ with step-wise learning rate decay at every 3 steps. The decay $\gamma$ was set as 0.95. For dataset `cora`, `citeseer`, `pubmed`, we updated the target constructors every 20 steps. For dataset `github`, `ppi`, we updated the target constructors every 30 steps. We performed a parameter sweep on training steps over $400, 500, 600$. The specific batch number in each step was decided by the training size of each dataset. For `cora`, `citeseer` datasets, the maximum ARFs size was set as 2. For `pubmed` dataset, the maximum ARFs size was set as 4, and for `github` and `ppi`, the maximum ARFs size was set as 8. To train the evaluator, we used the cross-entropy loss over the softmax output for single-class node classification and the sigmoid output for multi-class node classification; to train the constructor, we used the mean-square-error loss.

## Appendix C Further analysis

**Training curves.** The training loss curves of GRARF is shown in Figure C1. In our implementation, the step loss would not reach near 0, but converge to a constant value. The strategies given by constructors would also become stable as the number of training steps grown. To illustrate the stability of strategy, we visualized the entropy of $q$-values in Figure C2(a) given by constructors. With the number increasing, the entropy of $q$-values of each step decreased in fluctuations, and finally reached a constant approaching 0. This shows GRARF learns stable policies.

**Effects of maximum ARFs sizes.** To explore the relationship between the performances of GRARF and maximum ARFs sizes, we designed experiments on different maximum ARFs sizes on `cora`. Results are shown in Figure C2(b). The performance of GRARF remained consistent as the maximum ARF size varies. We believe that this indicates the neighborhood can indeed be depicted with sparse contexts.

**Distance distribution on LDD experiments.** To further demonstrate the benefits of using ARFs in LDD, we show the dependency length distributions of nodes having the same degree in Figure C3. In these experiments, maximum ARFs size was set as 5. We selected LDD with $\lambda = 2.0$ and divided central nodes according to their degrees. For low-degree central nodes, ARFs tended to include long-dependency nodes to exploit more information. For high-degree central nodes, ARF nodes with dependency

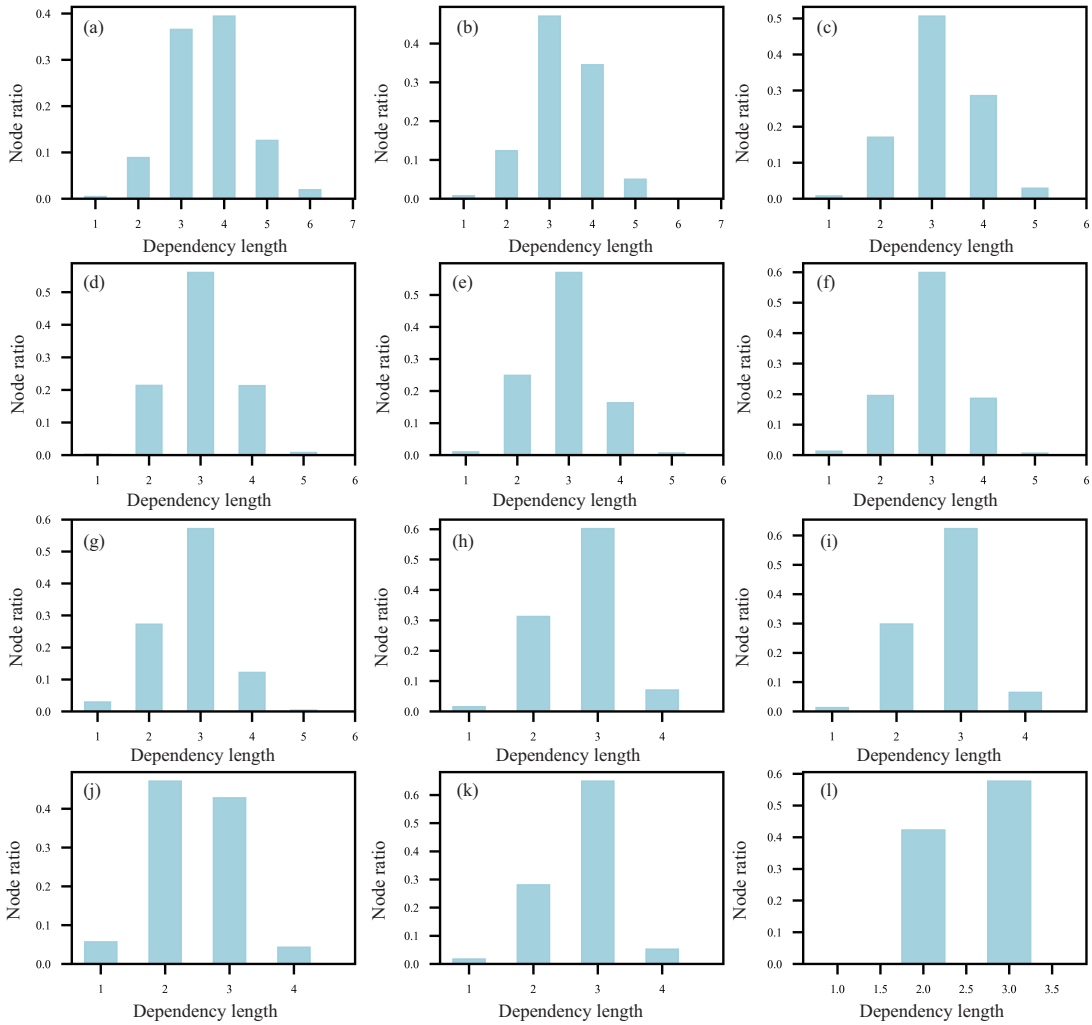


**Figure C3** (Color online) Dependency length of nodes in LDD experiments ($\lambda = 2.0$). (a)–(l) Correspond to central nodes of which degrees range from 2 to 13.

length at 2 and 3 appeared more frequently. Note that in the LDD setup, we assigned a split number $k$ drawn from Poisson($\lambda$) on each edge, and then split each edge to a $(k+1)$-hop path (do nothing if $k = 0$) by inserting $k$ noises. Under this setting, informative nodes are pulled away from central nodes, which means GCNs need to "look" further to aggregate informative nodes.