Resource-Efficient Training for Large Graph Convolutional Networks with Label-Centric Cumulative Sampling

Mingkai Lin, Wenzhong Li, Ding Li, Yizhou Chen and Sanglu Lu
State Key Laboratory for Novel Software Technology, Nanjing University
Nanjing, China
mingkai@smail.nju.edu.cn,lwz@nju.edu.cn

ABSTRACT

Graph Convolutional Networks (GCNs) are popular for learning representation of graph data and have a wide range of applications in social networks, recommendation systems, etc. However, training GCN models for large networks is resource intensive and time consuming, which hinders them from real deployment. The existing GCN training methods intended to optimize the sampling of mini-batches for stochastic gradient descent to accelerate training process, which did not reduce the problem size and had limited reduction in computation complexity. In this paper, we argue that a GCN can be trained with a sampled subgraph to produce approximate node representations, which inspires us a novel perspective to accelerate GCN training via network sampling. To this end, we propose a label-centric cumulative sampling (LCS) framework for training GCNs for large graphs. The proposed method constructs a subgraph cumulatively based on probabilistic sampling, and trains the GCN model iteratively to generate approximate node representations. The optimality of LCS is theoretically guaranteed to minimize the bias during node aggregation procedure in GCN training. Extensive experiments based on four real-world network datasets show that the LCS framework accelerates the training for the state-of-the-art GCN models up to 17x without causing noteworthy model accuracy drop.

CCS CONCEPTS

• **Information systems** → **Social networks**; • **Computing methodologies** → **Neural networks**.

KEYWORDS

Graph Convolutional Network, Network Sampling, Model Training Acceleration

ACM Reference Format:
Mingkai Lin, Wenzhong Li, Ding Li, Yizhou Chen and Sanglu Lu. 2022. Resource-Efficient Training for Large Graph Convolutional Networks with Label-Centric Cumulative Sampling. In *Proceedings of the ACM Web Conference 2022 (WWW '22), April 25–29, 2022, Virtual Event, Lyon, France.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3485447.3512165

WWW '22, April 25–29, 2022, Virtual Event, Lyon, France

ACM ISBN 978-1-4503-9096-5/22/04…$15.00
https://doi.org/10.1145/3485447.3512165

1 INTRODUCTION

Graph Convolutional Networks (GCNs) [16, 17, 34] are powerful models for learning representation of nodes and have achieved great success in dealing with graph-related applications with data that contains rich relational information among objects, which are widely applied in social networks, knowledge graphs, recommendation systems, etc. Despite the great potential of GCNs, training them efficiently on large graphs remains a challenging task. To train a GCN, the embedding representation of a node depends recursively on all its neighbors' embedding. *The inter-dependencies among nodes' representations grow exponentially with respect to the number of neural network layers, which is a fundamental problem in GCN training that could lead to long training time, huge resource consumption, and high computation cost due to the neighbor explosion phenomenon* [39].

To alleviate the computational burden of training GCN models, many acceleration methods were proposed [4, 5, 10, 35, 36, 40]. These training methods mainly focused on optimizing the choice of mini-batch samples for stochastic gradient descent to accelerate the training process, which did not reduce the problem size actually. Applying such methods on large graphs is still time-consuming and resource-expensive. For instance, the GraphSage [10] method that was optimized for large GCNs, took hours to train a model on the Amazon co-purchasing network (2M nodes) with a Tesla P100 GPU [36], and the VRGCN [5] method even suffered from out of memory error when training GCN on a similar dataset with a Tesla V100 GPU (16 GB memory) [6].

We argue that a GCN can be trained with a sampled subgraph to produce approximate node representations for the full graph. By taking a small subgraph to train a GCN (without sacrificing too much model accuracy), the problem size can be significantly reduced, and the resource requirements (CPU and memory usage) are alleviated accordingly, thereby it enables a GCN to be trained on large graphs with high resource-efficiency. *The idea of training a GCN with a sampled graph to approximate the results of a full graph provides a novel perspective on GCN training acceleration, which is fundamentally different from the existing studies and can be seamlessly integrated with the state-of-the-art mini-batch sampling techniques to further improve the training efficiency.*

Designing such a network sampling method for GCN training acceleration is non-trivial and encounters several challenges. Firstly, training a GCN with a sampled subgraph inevitably causes bias during model training. The key challenge is to seek an efficient subgraph sampling algorithm to reduce the bias for the learning objective as possible. Secondly, the trade-off between sampling size and model accuracy drop should be addressed. Intuitively, a smaller subgraph would be more helpful for accelerating model training,

but it also causes larger accuracy drop. It is critical to sample a subgraph with proper size without sacrificing too much accuracy.

To address these challenges, this paper proposes a scalable label-centric cumulative sampling (LCS) framework for GCN training acceleration. The LCS framework samples nodes from the original graph round by round in a cumulative way to form a subgraph for model training. In each round, it selects a set of informative *anchor nodes* from the original graph according to the uncertainty sampling strategy which is commonly used in the field of active learning. Based on the selected anchor nodes, it further develops a probabilistic sampling approach to select a set of *support nodes* aiming at minimizing the bias of node aggregation in different GCN layers. The anchor nodes and support nodes are combined together to form a subgraph, which is used as input to train a GCN model to produce approximate node representations based on an unbiased estimator that is theoretically guaranteed to be optimized. The process repeats iteratively, where the sampled subgraph is constructed cumulatively and the GCN is trained to gradually improve its performance until accuracy converges.

The contributions of our work are summarized as follows.

- We propose the novel idea of training a GCN with a small sampled subgraph to approximate the results of a full graph, which can significantly reduce the problem size and alleviate the resource requirements for GCN training. To the best of our knowledge, the problem of network sampling based GCN training acceleration with input size reduction has not yet been explored in the literature.
- We propose a scalable label-centric cumulative sampling (LCS) method to accelerate the training of GCNs for large graphs. The proposed framework constructs a sampled subgraph cumulatively to minimize node aggregation bias, and trains the GCN model iteratively to generate approximate node representations without causing noteworthy model accuracy drop. The performance of the proposed LCS framework is theoretically guaranteed (refer to Theorem 4.1-4.3 for more details).
- The efficiency and feasibility of the proposed framework is verified by extensive experiments based on four real-world network datasets. It is shown that LCS achieves up to 17x speed up on the state-of-the-art GCN training methods, meanwhile the model's accuracy drop is less than 3%.

2 RELATED WORKS

We introduce the related works in two aspects: GCN training techniques and network sampling methods.

2.1 GCN Training Techniques

Graph convolutional network (GCN) [16] has become increasingly popular in recent years [8, 25, 30, 33, 35, 37]. Despite the potential of GCNs, training GCNs on large graphs remains a big challenge. Many previous works propose different methods to accelerate GCN training process via optimizing mini-batch sampling. We roughly divide them into four categories: node-wise, layer-wise, subgraph-wise, and history-wise methods.

Node-wise method (e.g. GraphSage [10], PinSage[35]) samples the neighbors of each node by random walk to aggregate their representation. Although it reduces the computation complexity, it is still hard to scale for deep networks, because the number of nodes to be sampled grows exponentially by layers. *Layer-wise sampling method* was first proposed in FastGCN [4]. It treats GCN layers independently for neighbor nodes sampling to avoid recursive expansion of neighborhoods across layers. LADIES [40] made further improvement to reduce the sample size by restricting the candidate nodes in the union of the neighborhoods of the sampled nodes in the upper layer. *Subgraph-wise method* (e.g., ClusterGCN [6]) partitions graph into densely connected clusters during pre-processing, and construct mini-batches by randomly selecting subsets of clusters during training. GraphSAINT [36] sampled high influential nodes to form a subgraph for each mini-batch to avoid full-batch exploration. *History-wise method* was proposed in VRGCN [5]. It used the historical activations of nodes as a control variate to develop a stochastic approximation algorithm for GCN subsampling neighbors with theoretical guarantee to convergence. Later, MVS-GNN [7] employed gradient information to adaptively sample neighbors to reduce the variance introduced by embedding approximation and achieved faster convergence.

While plenty of efforts have been made to optimize mini-batch sampling for GCN model training, hardly any attention has been paid on reducing the input size. All the existing works took the original graph as input to train the GCN model, which is still inefficient and expensive in processing large graphs. Different from them, our work makes the first attempt to develop a novel network sampling mechanism that takes a pre-processed subgraph as input to reduce GCN training costs with comparable results. The proposed sampling method can be combined with the existing mini-batch sampling methods to achieve further efficiency improvement.

2.2 Network Sampling Methods

Network sampling methods aim to obtain a smaller graph which can represent the original network so that some useful results can be concluded from the subgraph. There has been a rich literature in statistics, data mining, and physics on estimating graph properties using graph sampling methods [2, 19, 22, 26]. We categorize network sampling methods into three groups: node sampling, edge sampling, and traversal based sampling.

Node sampling method constructs a subgraph by random node selection from the original graph, typically using a uniformly random selection approach called Random Node (RN) sampling [19] or its non-uniform variants such as Random Degree Node (RDN) sampling [19] and Random PageRank Node (RPN) sampling [24]. *Edge sampling method* randomly samples edges from the graph independently and uniformly, which is known as Random Edge (RE) sampling [18]. *Traversal based sampling method* constructs a subgraph by exploring the network from a set of initial nodes. The Random Walk (RW) sampling [27] picks nodes by randomly walking through the network. The Snow-Ball (SB) sampling [15] adds nodes and edges using Breadth-First-Search (BFS) from a randomly selected seed but stops early once it reaches a particular size. Extensions of Snow-Ball sampling include the Forest Fire Sampling (FFS) [19] and Respondent Driven Sampling (RDS) [9].

Unlike the above general network sampling methods, our work propose an adaptive label-centric cumulative network sampling methods, which is particularly tailored for accelerating GCN training.

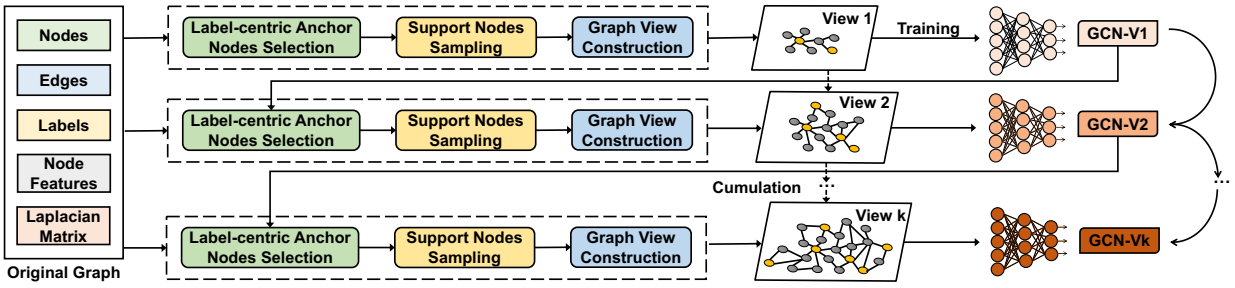


Figure 1: The proposed label-centric cumulative sampling (LCS) framework for GCN training.

3 NOTATIONS AND PRELIMINARIES

To simplify the presentation, we focus on undirected graphs. Let $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$ denote the undirected graph with nodes $\mathcal{V}$ and edges $\mathcal{E}$. Let N denote the number of nodes in the graph. The adjacency matrix $A \in \mathbb{R}^{N\times N}$ represents the weight associated to edge (v, u) by each element $A(v, u)$. To describe the attributes of the nodes, we use a feature matrix $X \in \mathbb{R}^{N\times K}$, where $X = \{x_1, x_2, ...x_N\}$ with x_i denoting the K-dimensional features of node v_i.

We adopted the well-known GCN model developed by Kipf and Welling [16], which is one of the most successful convolutional networks for graph representation learning. Given an L-layer GCN, the l-th graph convolution layer is represented by $H^l = \sigma(\tilde{A}H^{l-1}W^{l-1}) \in \mathbb{R}^{N\times K^l}$, where $\sigma(\cdot)$ is the activation function (e.g., ReLU); H^l means the representation matrix for nodes in the l-th layer with K^l dimensions; $\tilde{A}$ is the normalized Laplacian matrix [6, 10, 36] of A (e.g., $\tilde{A} = D^{-1}A$ where D is the diagonal degree matrix associated with $\mathcal{G}$); and W^l is the weight matrix with shape $K^{l-1} \times K^l$ which is to be learned during the training procedure. We consider the general scenario that a GCN is applied in the semi-supervised tasks where partial nodes in the graph are attached with ground-truth labels. We denote the set of labels as $Y = \{y_1, y_2, ..., y_{|Y|}\}$. With the above notations, the layer-wise latent representation [4] of a node v can be written by:

$$\begin{aligned} \xi^l(v) &= \sum_{u\in\mathcal{V}} \tilde{A}(v, u)h^{l-1}(u)W^{l-1}, \\ h^l(v) &= \sigma(\xi^l(v)), \quad (v \in \mathcal{V}, 1 \le l \le L) \end{aligned} \tag{1}$$

where $\xi^l(v)$ (resp. $h^l(v)$) is the representation of node v in the l-th layer before (resp. after) activation $\sigma(\cdot)$, and $h^0(u) = x_u$. The first equation in Eq. (1) represents the aggregation procedure in GCN where each node aggregates information from its direct neighbors. In order to ease the notation as well as analysis, we make $h^{l-1}(u)W^{l-1}$ denote as $\tilde{h}^{l-1}(u)$ and have:

$$\xi^l(v) = \sum_{u\in\mathcal{V}} \tilde{A}(v, u)\tilde{h}^{l-1}(u), \quad (v \in \mathcal{V}, 1 \le l \le L). \tag{2}$$

4 LABEL-CENTRIC CUMULATIVE SAMPLING FOR GCN MODEL TRAINING

In this section, we propose a label-centric cumulative sampling (LCS) method to accelerate GCN model training for large graphs. The overall framework is illustrated in Figure 1. Given a graph and a GCN model, the sampling and training process consists of several rounds. In the i-th round, it samples a subgraph from the original graph to form a *view*, which is used to train the i-th version of the GCN model. The views are sampled accumulatively based on the previous views with a label-centric probabilistic sampling methods to reduce sampling bias, and the GCN model is trained incrementally to evolving versions that minimize a predefined objection function. The training process converges after a limit number of sampling and training rounds, and the well-trained GCN model can be applied to generate approximate node presentations with the benefit of exploring only a small subgraph sampled from the original large network.

Each sampling round consists of three steps: *label-centric anchor nodes selection*, *support nodes sampling*, and *view construction*. Firstly, we select a small number of representative and informative anchor nodes from the labeled nodes in the graph; Secondly, on the basis of the selected anchor nodes, we sample a set of support nodes from the graph with a derived inclusion probability distribution targeting at reducing the bias of GCN model training; Finally, we combine the anchor nodes and support nodes together to construct a graph view, which is used as input to train the GCN model. The details are introduced in the following.

4.1 Label-Centric Anchor Nodes Selection

On training a GCN model, the original input graph typically consists of labeled nodes (the set of nodes that were annotated beforehand) and unlabeled nodes (the set of nodes whose labels to be derived by the downstream machine learning tasks). As the first step of LCS sampling, we select a small number of labeled nodes called *anchor nodes* from the graph, which are expected to be representative and informative for GCN training. Specifically, we select anchor nodes based on the principle of active learning [1, 20] that aims to achieve high accuracy using as few labeled instances as possible.

Following the most commonly used active learning query strategy called *uncertainty sampling* [1, 20], we select the nodes which the current model is least certain with regarding to classification prediction. Specifically we use the general uncertainty measure *information entropy* as our informativeness metric. The information entropy of a labeled node v is calculated as:

$$\phi_{entropy}(v) = -\sum_{c=1}^{C} P(Y_{vc} = 1|x_v; \theta) \log P(Y_{vc} = 1|x_v; \theta), \tag{3}$$

where $P(Y_{vc} = 1|x_v; \theta)$ is the probability of node v belonging to class c predicted by the current GCN model. The larger $\phi_{entropy}(v)$ is, the more uncertain the current model is regarding to v.

However, it is unrealistic to evaluate the information entropy of all labeled nodes in the graph since it is time consuming. We propose the following method to select informative nodes via uncertainty

sampling. Assume we want to select N_a anchor nodes. We firstly uniformly sample αN_a ($\alpha \geq 1$ & $\alpha \in \mathbb{R}$) random nodes from the set of labeled nodes as candidates, and evaluate their information entropy values. Then we select the top-N_a nodes with highest information entropy from the candidates as anchor nodes. The chosen value of the scaling factor α is discussed in our experiments.

In the cumulative sampling framework, the set of anchor nodes is sampled round by round. In the k-th round, the set of anchor nodes $\mathcal{V}_a^k$ combines the anchor nodes from the current round and previous rounds, i.e., $\mathcal{V}_a^k = \mathcal{V}_a^{k-1} \cup \{the\ anchor\ nodes\ sampled\ by\ the\ current\ round\}$. Unless explicitly mentioned, we omit the superscript k and simply refer to anchor nodes set as $\mathcal{V}_a$ in the following sections without causing confusion.

4.2 Support Nodes Sampling

The set of anchor nodes $\mathcal{V}_a$ is informative and contains higher information entropy for GCN training. However, directly applying $\mathcal{V}_a$ in Eq. (2) to estimate the representation aggregation of GCN could cause severe bias, and the aggregation error can not be theoretically bounded. To solve the problem, we further sample a set of *support nodes* $\mathcal{V}_s$ that are combined with the anchor nodes to achieve unbiased estimation of the aggregated representation in Eq. (2). Note that we analyze the aggregation of each layer independently and omit the non-linear activations, which is similar to the treatment of layers independently by prior works [4, 14, 36].

Specifically, we propose a probabilistic sampling approach that samples the support nodes with a probabilistic distribution $\Pi = \{\pi_v | (v \in \mathcal{V})\}$, where π_v is the probability that a node v is included in $\mathcal{V}_s$. Since the support nodes are sample based on the anchor nodes, we consider that the size of $\mathcal{V}_s$ is proportional to that of $\mathcal{V}_a$, i.e., $|\mathcal{V}_s| = \beta|\mathcal{V}_a|$, where $\beta \in \mathbb{R}$ is a scaling factor in the system.

Applying the GCN node aggregation process based on the sampled anchor nodes and support nodes, the approximate node representation in Eq. (2) can be written by

$$\tilde{\xi}^l(v) = \sum_{u \in \mathcal{V}_a \cup \mathcal{V}_s} \tilde{A}(v,u)\tilde{h}^{l-1}(u), \quad (v \in \mathcal{V}_a \cup \mathcal{V}_s, 1 \leq l \leq L). \quad (4)$$

However, the Laplacian matrix $\tilde{A}(v,u)(v,u \in \mathcal{V}_a \cup \mathcal{V}_s)$ derived from the sampled nodes is clearly different from that derived from the original graph. Directly applying Eq. (4) to approximate the representations in Eq. (2) is inaccurate. Next, we will derive the optimal inclusion probabilities to sample the support nodes and propose an unbiased estimator to approximate the aggregated node representation based on label-centric cumulative sampling.

4.2.1 Optimization Objective. We consider support node sampling as an optimization problem that minimize the bias during GCN node aggregation in independent layers. The bias of using a sampled subgraph to train a GCN to approximate the node representation comes from two parts: the *estimation error* E_e and the *sampling error* E_s. The estimation error E_e refers to the difference between the estimated node representation and their corresponding ideal representation (e.g., using $\tilde{\xi}^l(v)$ to approximate $\xi^l(v)$ in different GCN layers). The sampling error E_s refers to the variance of using the sampled graph to estimate the representation on unseen nodes (e.g., applying the trained GCN to infer the embedding of unseen nodes). Therefore the objective of support node sampling is to determine the optimal inclusion probability for each node to minimize the bias, which can be casted as an optimization problem:

$$\begin{aligned} &\arg\min_{\Pi} \sum_{\mathcal{V}_s \sim \Pi} p(\mathcal{V}_s)[E_e(\mathcal{V}_s) + E_s(\mathcal{V}_s)], \\ &\textbf{subject to} \sum_{v \in \mathcal{V}} \pi_v = |\mathcal{V}_s|,\ \pi_v \in (0,1]. \end{aligned} \quad (5)$$

where $p(\mathcal{V}_s)$ is the probability of sampling a set of specific support nodes $\mathcal{V}_s$, and Π is the optimal inclusion probability distribution to be derived. Both $E_e(\mathcal{V}_s)$ and $E_e(\mathcal{V}_s)$ are associated with the sampled supports nodes set $\mathcal{V}_s$. Note that the anchor nodes are established in advance, thus they are not taken into consideration for bias minimization.

4.2.2 Theoretical Analysis. In this section, we provide theoretical analysis to derive the optimal inclusion probability distribution for support node sampling and theoretical guarantee for unbiased estimation. The proofs of the theorems are in the Appendix.

We firstly propose the following estimator for $\tilde{\xi}^l(v)$ based on the Horvitz-Thompson theory [12] to address the estimation bias:

$$\begin{aligned} \tilde{\xi}^l(v) &= \sum_{u \in \mathcal{V}_s} \frac{\tilde{A}(v,u)}{\pi_{u,v}/\pi_v}\tilde{h}^{l-1}(u)\ \ (v \in \mathcal{V}_s, 1 \leq l < L), \\ \tilde{\xi}^L(v) &= \sum_{u \in \mathcal{V}_s} \frac{\tilde{A}(v,u)}{\pi_u}\tilde{h}^{L-1}(u)\ \ (v \in \mathcal{V}_a). \end{aligned} \quad (6)$$

In the formulation, π_u is the first order inclusion probability for u being included in the support node set $\mathcal{V}_s$; $\pi_{u,v}$ is the second order inclusion probability for node pair (u,v), namely the joint probability of both nodes u,v are included in $\mathcal{V}_s$. The following theorem holds for unbiased estimation.

Theorem 4.1. *Given an inclusion probability distribution $\Pi = \{\pi_v(v \in \mathcal{V})\}$, and a set of support nodes $\mathcal{V}_s$ that is sampled regarding to Π, Eq. (6) provides an unbiased estimation for the corresponding ideal aggregation $\xi^l(v)$, i.e.,*

$$\begin{aligned} \mathbb{E}[\tilde{\xi}^l(v)] &= \sum_{u \in \mathcal{V}} \tilde{A}(v,u)\tilde{h}^{l-1}(u) = \xi^l(v),\ (v \in \mathcal{V}_s, 1 \leq l < L) \\ \mathbb{E}[\tilde{\xi}^L(v)] &= \sum_{u \in \mathcal{V}} \tilde{A}(v,u)\tilde{h}^{L-1}(u) = \xi^L(v)\ \ (v \in \mathcal{V}_a). \end{aligned} \quad (7)$$

Theorem 4.1 shows that Eq. (6) is an unbiased estimation for Eq. (2). Based on the estimators, in order to describe the relations between ideal aggregations and estimated aggregations, we define the following three variables:

$$\begin{aligned} F &= \sum_{l<L}\sum_{v \in \mathcal{V}} \xi^l(v) + \sum_{v \in \mathcal{V}_a} \xi^L(v), \\ \widehat{F}(\mathcal{V}_s) &= \sum_{l<L}\sum_{v \in \mathcal{V}_s} \frac{\xi^l(v)}{\pi_v} + \sum_{v \in \mathcal{V}_a} \xi^L(v), \\ \widetilde{F}(\mathcal{V}_s) &= \sum_{l<L}\sum_{v \in \mathcal{V}_s} \frac{\tilde{\xi}^l(v)}{\pi_v} + \sum_{v \in \mathcal{V}_a} \tilde{\xi}^L(v). \end{aligned} \quad (8)$$

In the above equation, F is the sum of the ideal node aggregations in all layers with the original full graph. Since the last layer (i.e., the L-th layer) of GCN computes the loss function for model training, we associate it with only anchor nodes with labels. $\widehat{F}(\mathcal{V}_s)$ is the sum for the original aggregations regarding the sampled graph, where

π_v serves as a weight factor to compensate the sum of unseen nodes. $\widetilde{F}(\mathcal{V}_s)$ has the same form as $\widehat{F}(\mathcal{V}_s)$ excepts that it aggregates the estimated node representations $\tilde{\xi}^l(v)$.

With the above variables, the estimation error in Eq. (5) can be calculated by

$$E_e(\mathcal{V}_s) = \|\widetilde{F}(\mathcal{V}_s) - \widehat{F}(\mathcal{V}_s)\|^2, \tag{9}$$

and the sampling error can be calculated by

$$E_s(\mathcal{V}_s) = \|\widehat{F}(\mathcal{V}_s) - F\|^2. \tag{10}$$

Then we can further derive the following theorem.

Theorem 4.2. *If Eq. (6) is adopted to estimate $\xi^l(v)$, then the sum of the estimated aggregations $\widetilde{F}(\mathcal{V}_s)$ is also an unbiased estimator for the sum of the ideal aggregations F, i.e.,*

$$\mathbb{E}[\widetilde{F}(\mathcal{V}_s)] = \sum_{l<L}\sum_{v\in\mathcal{V}} \xi^l(v) + \sum_{v\in\mathcal{V}_a} \xi^L(v) = F. \tag{11}$$

The above theorem builds a bridge between the estimated aggregations in the sampled graph and the aggregations in the original graph. And it is easy to derive $\mathbb{E}[\widehat{F}(\mathcal{V}_s)] = F$, which means that $\widehat{F}(\mathcal{V}_s)$ is also an unbiased estimator for F. With Eqs. (9) (10), we can rephrase the objective in Eq. (5) to an equivalent form:

$$\begin{aligned}
&\textstyle\sum_{\mathcal{V}_s\sim\Pi} p(\mathcal{V}_s)[E_e(\mathcal{V}_s) + E_s(\mathcal{V}_s)] \\
=&\textstyle\sum_{\mathcal{V}_s\sim\Pi} p(\mathcal{V}_s)[\|\widetilde{F}(\mathcal{V}_s) - \widehat{F}(\mathcal{V}_s)\|^2 + \|\widehat{F}(\mathcal{V}_s) - F\|^2] \\
=&\mathbb{E}[\|\widetilde{F}(\mathcal{V}_s) - \widehat{F}(\mathcal{V}_s)\|^2 + \|\widehat{F}(\mathcal{V}_s) - F\|^2] = \mathbb{E}[\|\widetilde{F}(\mathcal{V}_s) - F\|^2] \\
=&\mathbb{E}[\|\widetilde{F}(\mathcal{V}_s) - \mathbb{E}(\widetilde{F}(\mathcal{V}_s))\|^2] = \mathrm{Var}[\widetilde{F}(\mathcal{V}_s)].
\end{aligned} \tag{12}$$

The optimization problem transforms to minimizing the variance of $\widetilde{F}(\mathcal{V}_s)$ subjected to $\sum_{v\in\mathcal{V}} \pi_v = |\mathcal{V}_s|, \pi_v \in (0, 1]$. The formulation of $\mathrm{Var}[\widetilde{F}(\mathcal{V}_s)]$ is provided by the following theorem.

Theorem 4.3. *If we denote*

$$\Psi_v = \sum_{u\in\mathcal{V}_a} \tilde{A}(u, v) + (L-1)\sum_{u\in\mathcal{V}} [\tilde{A}(u, v) + \tilde{A}(v, u)], \tag{13}$$

the variance of $\widetilde{F}(\mathcal{V}_s)$ admits:

$$\mathrm{Var}[\widetilde{F}(\mathcal{V}_s)] \propto \sum_{v\in\mathcal{V}} \frac{\Psi_v^2}{\pi_v} - \sum_{v\in\mathcal{V}} \Psi_v^2. \tag{14}$$

Based on the above theorem, the optimal inclusion probability can be derived by minimizing $\mathrm{Var}[\widetilde{F}(\mathcal{V}_s)]$. By Cauchy-Schwarz inequality [11], the right part of formula (14) is minimized on the condition that π_v is proportional to $|\Psi_v|$. Thus with the constraint of $\sum_{v\in\mathcal{V}} \pi_v = |\mathcal{V}_s|$, it is easy to get the optimal inclusion probability for $v \in \mathcal{V}_s$:

$$\pi_v = \frac{|\mathcal{V}_s|}{\sum_{v'\in\mathcal{V}} |\Psi_{v'}|} |\Psi_v|. \tag{15}$$

Through Eqs. (13)(15), we can intuitively infer that the nodes directly connected to the anchor nodes and the nodes with the ability to pass more information among GCN layers are more probable to be selected as support nodes. In the implementation, we sample the set of support nodes with Poisson sampling method [29] according to Eq. (15). Specifically, we sample a sequence of random variable $\{\psi_v\}_{v\in\mathcal{V}}$, where $\psi_v \sim$ Bernoulli(π_v), and $\psi_v = 1$ indicates that node v is sampled. With such Poisson process, the inclusion probabilities of nodes are independent with each other, and the second order inclusion probability in Eq. (6) can be computed by $\pi_{v,u} = \pi_v\pi_u$, which enables us to obtain the support nodes based on the selected anchor nodes easily.

4.3 View Construction

A *view* is defined as a sampled subgraph that is used to train a GCN model. In the k-th sampling round of the LCS framework, after obtaining the accumulative anchor nodes $\mathcal{V}_a$ and support nodes $\mathcal{V}_s$, the subgraph view $\mathcal{G}'$ with node set $\mathcal{V}'$ and edge set $\mathcal{E}'$, i.e., $\mathcal{G}' = \{\mathcal{V}', \mathcal{E}'\}$, can be constructed with $\mathcal{V}' = \mathcal{V}_a \cup \mathcal{V}_s$ and $\mathcal{E}' = \{(u, v)|\forall u, v \in \mathcal{V}', (u, v) \in \mathcal{E}\}$.

After acquiring the view $\mathcal{G}'$, we can use it to train a new version of the GCN for the k-th round. The training is based on the model parameters in the previous round, and it applies Eq. (6) for node aggregation since it is an unbiased estimation of the original aggregation according to Theorem 4.1.

4.4 Overall Algorithm

Algorithm 1 Label-Centric Cumulative Sampling (LCS) Algorithm

Input: Original graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}, \tilde{A}\}$; Threshold τ; Number of anchor nodes N_a per round; Scaling factor α; Scaling factor β
Output: The well trained model *GCN*

1: Accuracy gain $AG = 1$, Accuracy score $score = 0$, Round number $round = 0$
2: Initialize $\mathcal{V}_a = \emptyset$, initialize GCN model as *GCN*
3: **while** $AG > \tau$ **do**
4: # **Label-centric Anchor Nodes Selection**
5: Randomly sample αN_a labeled nodes (without replacement) as candidates set D_a from the labeled nodes
6: Estimate the information entropy for nodes in D_a
7: Get N_a nodes with top information entropy from D_a as T_a
8: $\mathcal{V}_a = \mathcal{V}_a \cup T_a$
9: # **Support Nodes Sampling**
10: Calculate inclusion probability $\Pi = \{\pi_v(v \in \mathcal{V})\}$ based on Eq. (15) with $|\mathcal{V}_s| = \beta|\mathcal{V}_a|$
11: According to Π, conduct Poisson sampling in $\mathcal{V}$ to get $\mathcal{V}_s$
12: # **View Construction**
13: Construct a sampled subgraph $\mathcal{G}'$ with $\mathcal{V}' = \mathcal{V}_s \cup \mathcal{V}_a$, $\mathcal{E}' = \{(u, v)|\forall u, v \in \mathcal{V}', (u, v) \in \mathcal{E}\}$
14: Construct renormalized Laplacian Matrix $\tilde{A}'$ through Eq. (6)
15: # **GCN Training**
16: Prepare embeddings $X_s = X[\mathcal{V}']$ and Labels $Y_a = Y[\mathcal{V}_a]$
17: *GCN* = Training GCN($\tilde{A}', X_s, Y_a$),
18: Evaluate GCN to get *new_score*
19: $AG = new\_score - score$, $score = new\_score$
20: **end while**
21: **return** *GCN*

The pseudo-code for the overall LCS method is shown in Algorithm 1. Line 5-8 is the part of label-centric adaptive anchor nodes selection. Line 10-11 is the part of support nodes sampling which samples support nodes according to the optimal inclusion probabilities. Line 13-14 is the procedure to construct a subgraph view. In Line 17 a new version of the GCN model is trained based on the corresponding input subgraph. Later line 18 is the procedure

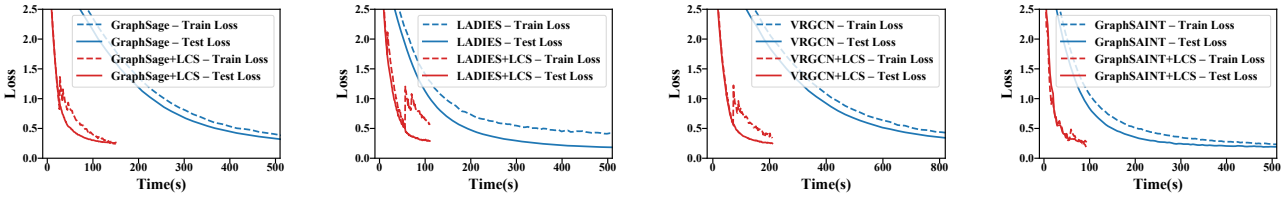


Figure 2: Loss curves for different GCN training methods with/without LCS in Reddit.

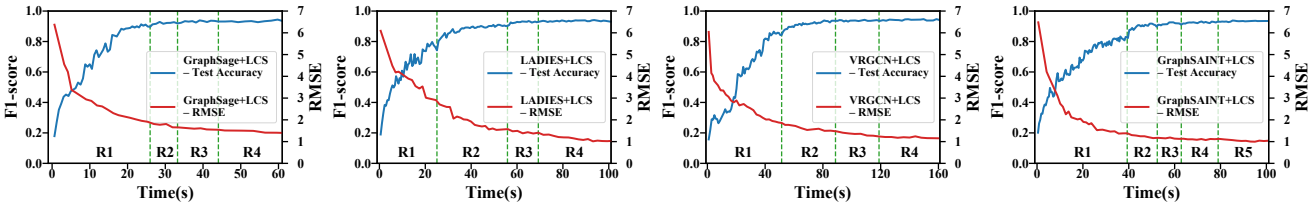


Figure 3: RMSE and test accuracy curves for different GCN training methods with LCS in Reddit. In the figure, R_i represents the intervals of different sampling rounds.

Table 1: Datasets statistics.

Datasets	Nodes	Edges	Feat.	Clas.	Train / Val / Test
Pubmed	19,717	44,338	500	3	0.92 / 0.03 / 0.05
Reddit	232,965	11,606,919	602	41	0.66 / 0.10 / 0.24
Yelp	716,847	6,977,410	300	100	0.75 / 0.10 / 0.15
Amazon	1,598,960	132,169,734	200	107	0.85 / 0.05 / 0.10

Table 2: Size of the sampled graph with LCS method (AN: Anchor nodes; TSN: Total sampled nodes; TSE: Total sampled edges).

Methods	Pubmed			Reddit			Yelp			Amazon		
	AN	TSN	TSE	AN	TSN	TSE	AN	TSN	TSE	AN	TSN	TSE
GraphSage+LCS	4.8%	9.3%	2.5%	4.0%	7.8%	1.1%	5.8%	11.2%	5.5%	6.2%	11.7%	8.9%
LADIES+LCS	5.2%	10.1%	2.8%	4.2%	8.2%	1.2%	6.2%	11.9%	6.1%	6.0%	11.4%	8.5%
VRGCN+LCS	4.8%	9.4%	2.6%	3.8%	7.5%	1.0%	5.2%	10.1%	4.7%	6.0%	11.4%	8.4%
GraphSAINT+LCS	5.8%	11.2%	3.4%	4.8%	9.4%	1.6%	5.8%	11.2%	5.5%	5.8%	11.0%	8.1%

to evaluate the trained GCN model in order to derive a new test accuracy score. If the performance gain (the difference of the test accuracy scores between the consecutive GCN versions) is below a pre-defined threshold τ, the training process terminates and the latest GCN model is returned as the final output.

5 EXPERIMENTS

In this section, we conduct experiments to evaluate the efficiency of the proposed LCS framework for training GCNs on various graphs for node classification task[1].

5.1 Experimental Settings

5.1.1 Dataset. The experiments are conducted on four real-world networks: (1) **Pubmed [23]**: A citation network that consists of scientific publications and their citations from the PubMed database pertaining to diabetes classes. (2) **Reddit [10]**: A graph dataset from the Reddit posts with 50 large communities being sampled to build a post-to-post graph. (3) **Yelp [36]**: A graph formed by users of the Yelp website that uses customer reviewers and friendship to categorize types of businesses. (4) **Amazon [36]**: A graph obtained from the buyer reviews and interactions in Amazon website that is used to classify product categories.

The detailed statistics of these datasets are summarized in Table 1. Note that the split for training, validation and test sets are according to the setup in [4, 7, 10, 36, 40].

5.1.2 Baseline Algorithms and Default Parameters. We implement the proposed LCS framework on four state-of-the-art GCNs: a node-wise method GraphSage [10], a layer-wise method LADIES [40], a subgraph-wise method GraphSAINT [36], and a history-wise method VRGCN [5]. The implementation of the baselines algorithms is based on the source code from [7].

[1] The implementation of algorithm is available here.

By default, we utilize the vanilla two-layer GCNs with the Laplacian multiplication aggregation defined in [10]. We set the hidden state dimension 128 for Pubmed and Reddit, and 512 for Yelp and Amazon according to the work [36]. In the implementation of LCS, we sample 1% nodes of the whole network as anchor nodes in each round, and the scaling factors are set to $\alpha = 5$ and $\beta = 1$ by default. The performance gain threshold τ is set to 0.01.

The batch size is set to 1024 for all baselines and it is proportionally scaled for LCS regarding the sampled subgraph. The models are updated with Adam optimizer with a learning rate of 0.001. We report the model accuracy in terms of Micro-F1 score. For each settings, we conduct training for 5 times and take the mean of the evaluation results. The experiments are implemented with Pytorch in Python 3.6.8 and conducted on a personal computer with Intel Xeon E5-2620 v2 2.10GHz CPU, GeForce RTX 2070 8G GPU and 64GB memory, running 64-bit CentOS Linux 7.2.

5.2 Numerical Results

5.2.1 Convergence Analysis. We first compare the train/test loss of different GCNs with/without applying LCS, and Fig. 2 shows the results on Reddit dataset. The curves in the other three datasets are similar and they are omitted due to page limit. After applying LCS for GCN training, both train loss and test loss decrease rapidly, which implies that the corresponding models converge much faster than that without LCS. In the figure, LCS shows different acceleration levels for different GCN models in the convergence of loss functions.

To study the ability of LCS in bias reduction for GCN node aggregation, we compute the Root Mean Square Error (RMSE) of node representations compared to that of a GCN trained with the full graph. The results of RMSE and test accuracy during different sampling rounds are illustrated in Fig. 3. With the increasing

Table 3: Comparison of memory usage and total time for training GCN models with/without LCS.

Pubmed	CPU Mem. (MB)	GPU Mem. (MB)	Samp. rounds	Samp. time per round(s)	Epochs	Time per epoch(s)	Total time (min)	Speed up
GraphSage	41.59	54.93	–		121	0.78	1.56	8.03
GraphSage+LCS	3.84	6.27	5	0.01	122	0.09	0.19	
LADIES	41.54	45.05	–		139	0.52	1.20	8.04
LADIES+LCS	3.80	6.57	5	0.01	133	0.07	0.15	
VRGCN	93.04	101.62	–		150	0.75	1.87	6.19
VRGCN+LCS	8.56	8.02	5	0.02	130	0.14	0.30	
GraphSAINT	41.04	30.82	–		138	0.46	1.07	7.19
GraphSAINT+LCS	4.37	4.41	6	0.01	126	0.07	0.15	

Reddit	CPU Mem. (MB)	GPU Mem. (MB)	Samp. rounds	Samp. time per round(s)	Epochs	Time per epoch(s)	Total time (min)	Speed up
GraphSage	1530.69	216.91	–		160	16.80	44.80	16.88
GraphSage+LCS	96.65	20.22	4	0.49	122	1.29	2.65	
LADIES	1417.66	51.81	–		164	6.55	17.89	10.57
LADIES+LCS	91.54	6.60	4	0.52	136	0.73	1.69	
VRGCN	2321.08	1366.81	–		177	22.05	64.73	17.25
VRGCN+LCS	146.91	59.39	4	0.48	163	1.38	3.75	
GraphSAINT	1410.92	39.11	–		183	5.13	15.61	9.42
GraphSAINT+LCS	91.31	5.69	5	0.50	156	0.62	1.66	

Yelp	CPU Mem. (MB)	GPU Mem. (MB)	Samp. rounds	Samp. time per round(s)	Epochs	Time per epoch(s)	Total time (min)	Speed up
GraphSage	2050.95	139.74	–		228	33.53	127.30	10.07
GraphSage+LCS	220.62	22.82	6	0.78	208	3.62	12.65	
LADIES	1914.73	88.83	–		244	20.81	84.55	8.17
LADIES+LCS	207.98	18.06	6	0.80	209	2.95	10.35	
VRGCN	9808.69	803.23	–		313	34.01	177.63	7.99
VRGCN+LCS	906.57	54.26	5	0.77	273	4.85	22.22	
GraphSAINT	1905.23	80.47	–		272	16.07	72.75	7.88
GraphSAINT+LCS	212.49	16.82	6	0.80	217	2.53	9.24	

Amazon	CPU Mem. (MB)	GPU Mem. (MB)	Samp. rounds	Samp. time per round(s)	Epochs	Time per epoch(s)	Total time (min)	Speed up
GraphSage	7927.63	109.82	–		228	103.62	394.96	11.59
GraphSage+LCS	648.45	21.30	6	4.89	234	8.61	34.08	
LADIES	5032.83	71.54	–		227	51.52	194.88	10.34
LADIES+LCS	371.56	16.01	6	5.13	263	4.18	18.86	
VRGCN	–				Memory Error			–
VRGCN+LCS	2951.54	174.53	6	4.91	378	15.24	96.70	
GraphSAINT	4685.25	66.16	–		171	50.70	144.49	7.06
GraphSAINT+LCS	429.98	14.99	6	5.30	218	5.48	20.48	

sampling rounds, the RMSE gradually decreases below 1, which means the approximate node representation approaches the ideal representation with the full graph. According to the figure, most GCN models converge in 4 or 5 sampling rounds with their test accuracy higher than 0.9.

5.2.2 Size of Sampled Graph. Table 2 shows the sampled graph size when applying LCS for GCN training. We report the size for anchor nodes, total sampled nodes, and total sampled edges by percentage of the original graph. From the table, we can find that the scale of the cumulative subgraph is in general about 10% out of the original graph except for Yelp and Amazon datasets which nearly reach 12%. We notice that GraphSAINT often needs larger cumulative subgraph (e.g. Pubmed and Reddit). The Amazon graph is more dense in edge, therefore the percentage of sampled edges is relatively higher than that of the other datasets. It shows that LCS method can reduce the data usage (as well as the input size) for GCN training significantly.

5.2.3 Comparison of Resource Usage and Training Time. We compare the resource usage during training process and the training time of different methods with/without LCS. The results are illustrated in Table 3. We have the following takeaways. (1) Resource usage in terms of CPU memory and GPU memory during training process is significantly reduced after applying LCS for model training, where up to 90% memory reduction is observed for different training methods on various of datasets. (2) The time for training a GCN with LCS consists of both sampling time (for several rounds) and training time (for several epochs). The sampling time is much less than the training time in each epoch, which is almost negligible. (3) With LCS, the training time per epoch is dramatically reduced, thanks to the reduction of resource usage in training with a sampled subgraph. (4) The training time of a GCN is significantly accelerated with LCS, and a speed up of 17x is observed for VRGCN+LCS in Reddit dataset.

5.2.4 Comparison of Accuracy with Baselines. We report the accuracy (F1-score) for baseline training methods and the corresponding LCS versions. From the results in Table 4, we can see that the accuracies of LCS-based methods are slightly lower than that of the original ones. It is normal since the sampling based method inevitably causes information loss. From the results, the accuracy drops from the baseline methods are slight and below 3% (except GraphSage+LCS, VRGCN+LCS in Pubmed and GraphSage+LCS in Amazon). Most drops are within 2% like GraphSage, VRGCN and GraphSAINT in Reddit. It shows that the proposed LCS method achieves comparable model accuracy as the baseline methods.

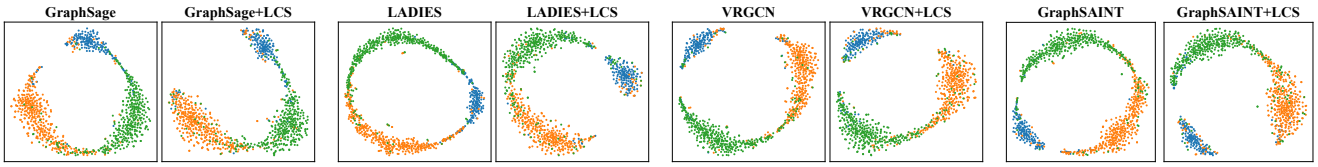
Table 4: F1-score (%) for different GCN training methods.

Methods	Pubmed	Reddit	Yelp	Amazon
GraphSage	90.42 ± 0.29	96.74 ± 0.03	64.46 ± 0.07	80.78 ± 0.04
GraphSage+LCS	87.36 ± 0.14	94.99 ± 0.09	61.71 ± 0.15	77.16 ± 0.05
LADIES	90.48 ± 0.44	96.50 ± 0.06	64.39 ± 0.07	80.65 ± 0.13
LADIES+LCS	87.88 ± 0.28	93.92 ± 0.16	61.26 ± 0.09	77.91 ± 0.16
VRGCN	90.66 ± 0.12	96.69 ± 0.06	63.12 ± 0.07	Memory Error
VRGCN+LCS	87.12 ± 0.19	95.20 ± 0.10	60.17 ± 0.08	76.49 ± 0.16
GraphSAINT	90.98 ± 0.20	95.45 ± 0.03	63.85 ± 0.25	78.66 ± 0.03
GraphSAINT+LCS	88.04 ± 0.29	93.89 ± 0.11	61.21 ± 0.10	76.16 ± 0.07

Table 5: F1-score (%) of different sampling methods

Datasets	Methods	RN	RE	RW	SB	LCS
Reddit	GraphSage	93.81	89.80	93.59	82.52	**94.99**
	LADIES	91.50	89.74	88.83	71.82	**93.92**
	VRGCN	93.89	89.91	93.34	79.09	**95.20**
	GraphSAINT	90.56	89.77	90.44	76.12	**93.89**
Amazon	GraphSage	73.45	73.64	71.23	72.08	**77.16**
	LADIES	73.62	73.43	74.92	75.27	**77.91**
	VRGCN	73.24	73.84	71.96	72.31	**76.49**
	GraphSAINT	71.67	72.10	71.92	70.69	**76.16**

5.2.5 Comparison of Accuracy with Different Network Sampling Methods. We further compare the proposed LCS method with several mainstream network sampling methods, which include Random Node (RN) [19], Random Edge (RE) [18], Random Walk (RW) [27] and Snowball (SB) sampling [27]. The implementation of these methods are from the work [32]. We compare them with the same size of sampled subgraph. Due to page limit, we only report the results of Reddit and Amazon datasets in Table 5. According to the results, RN and RW perform better than RE and SB methods in Reddit while RW and SB with LADIES perform well in Amazon. But our proposed LCS outperforms all other network sampling methods with significant improvement in F1-score, thanks to its



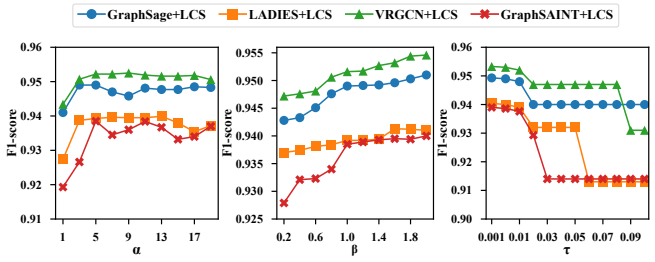
(a) GraphSage (b) LADIES (c) VRGCN (d) GraphSAINT

Figure 4: Visualization of node representation of different GCN models in Pubmed.

adaptivity of choosing informative labeled nodes and the ability of minimizing the bias during sampling.

5.2.6 Visualization of Node Representations. To intuitively understand the effectiveness of LCS based GCN training, we visualize the node representation of different GCN models with and without LCS. We use TSNE [31], a widely used dimension reduction algorithm, to visualize the nodes representation (with different color representing different classes) in a two-dimensional plane, and the results are shown in Fig. 4. It can be seen that the latent embeddings derived from the LCS method are very similar with the original embeddings from the full graph, where they have similar shapes in both the classification clusters and the relative positions in space. It implies that the LCS method can well approximate the node representations, and it can achieve good performance for graph representation learning in node classification task.

5.2.7 Hyperparameter Analysis. The LCS framework refers to three hyperparameters which should be tuned. They are respectively the scaling factor α for selecting anchor node, the scaling factor β for selecting support node, and the performance gain threshold τ for training terminate. In implementation we take 1% for anchor nodes as the fixed basic scale unit to expand the subgraph in each round. A more flexible subgraph expansion method can be adapted from [13], which will be considered in our future work. We study the influence of the hyperparameters based on the Reddit dataset, and the results are shown in Fig. 5.



(a) Parameter α (b) Parameter β (c) Parameter τ

Figure 5: Hyperparameter analysis for α, β and τ in Reddit.

Fig. 5(a) shows the result of α tuning from 1 to 20. When α increasing from 1 to 5, the F1-score increases sharply, which means that compared with selecting anchor nodes randomly, choosing nodes by information entropy from more candidates is effective. When α is larger than 5, the F1-score drops remains stable and the curve keeps fluctuation slightly. Hence, a suitable value for α is 5.

Fig. 5(b) shows the performance with β varying from 0.1 to 2. It is no doubt that with the increase of β, the scale of the sampled subgraph gets larger, and the F1-score gets higher. However, larger β also means larger problem size and higher computational complexity. It is observed that the performance increase is more visible when β is tuned from 0.1 to 1, and slows down thereafter. We set $\beta = 1$ considering the trade off between model accuracy drop and computation efficiency.

Fig. 5(c) shows the performance with τ varying from 0.001 to 0.09. Parameter τ controls the terminate of GCN training, and a smaller τ means more convergent of the GCN model, which typically requires more sampling rounds. When τ increases from 0.001 to 0.01, the F1-score remains stable, which means all GCN models converge well. When τ is larger than 0.03, dramatic performance drop are observed for all GCN models. Therefore we set $\tau = 0.01$ that keeps high accuracy without causing too many sampling rounds.

Another important hyperparameter is the number of GCN layers l, and we show the F1-score vs. model layers in Fig. 6. When GCN models go deeper, the changes of accuracies are not significant except VRGCN. The visible accuracy drop for VRGCN may be due to over-smooth [21]. But for VRGCN+LCS, such phenomenon is weakened. For most GCN models, increasing l from 2 to 6 does not improve accuracy significantly, which is in line with the findings in [3, 28, 38].

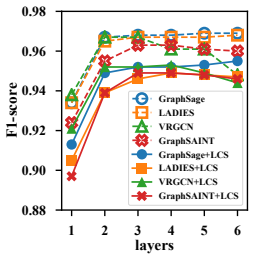


Figure 6: Analysis of GCN layers l in Reddit.

6 CONCLUSION

Training GCN models for large networks is a challenging task. In this paper, we introduced a novel idea of training a GCN with a small sampled subgraph to approximate the results at scale, which can significantly reduce the problem size and alleviate the resource requirements. We proposed a label-centric cumulative sampling (LCS) method to construct a sampled subgraph, which was used to train the GCN model iteratively to generate approximate node representations. The LCS method was theoretically guaranteed to be unbiased and convergent, and it could be combined with the existing mini-batch sampling methods to achieve further efficiency improvement. Extensive experiments based on real-world network datasets showed that the LCS framework significantly reduced resource usage, accelerated GCN training up to 17x, and achieved comparable model accuracy compared to the state-of-the-arts.

ACKNOWLEDGMENTS

This work was partially supported by the National Natural Science Foundation of China (Grant Nos. 61972196, 61832008, 61832005), the Collaborative Innovation Center of Novel Software Technology and Industrialization, and the Sino-German Institutes of Social Computing. The corresponding author is Wenzhong Li (lwz@nju.edu.cn).

REFERENCES

[1] Charu C Aggarwal, Xiangnan Kong, Quanquan Gu, Jiawei Han, and S Yu Philip. 2014. Active learning: A survey. In *Data Classification: Algorithms and Applications*. CRC Press, 571–605.

[2] Nesreen K Ahmed, Jennifer Neville, and Ramana Kompella. 2014. Network sampling: From static to streaming graphs. *ACM Transactions on Knowledge Discovery from Data (TKDD 2014)* 8, 2 (2014), 7.

[3] Deli Chen, Yankai Lin, Wei Li, Peng Li, Jie Zhou, and Xu Sun. 2020. Measuring and relieving the over-smoothing problem for graph neural networks from the topological view. In *Proceedings of the AAAI Conference on Artificial Intelligence(AAAI 2020)*, Vol. 34. 3438–3445.

[4] Jie Chen, Tengfei Ma, and Cao Xiao. 2018. FastGCN: Fast Learning with Graph Convolutional Networks via Importance Sampling. In *International Conference on Learning Representations (ICLR 2018)*.

[5] Jianfei Chen, Jun Zhu, and Le Song. 2018. Stochastic training of graph convolutional networks with variance reduction. In *Proceedings of the 35th International Conference on Machine Learning (ICML 2018)*. 1–9.

[6] Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. 2019. Cluster-GCN: An efficient algorithm for training deep and large graph convolutional networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD 2019)*. 257–266.

[7] Weilin Cong, Rana Forsati, Mahmut Kandemir, and Mehrdad Mahdavi. 2020. Minimal variance sampling with provable guarantees for fast training of graph neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD 2020)*. 1393–1403.

[8] Songgaojun Deng, Huzefa Rangwala, and Yue Ning. 2019. Learning Dynamic Context Graphs for Predicting Social Events. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD 2019)*. 1007–1016.

[9] Minas Gjoka, Maciej Kurant, Carter T Butts, and Athina Markopoulou. 2010. Walking in facebook: A case study of unbiased sampling of osns. In *2010 Proceedings IEEE International Conference on Computer Communications (INFOCOM 2010)*. Ieee, 1–9.

[10] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *Advances in neural information processing systems (NIPS 2017)*. 1024–1034.

[11] Godfrey Harold Hardy, John Edensor Littlewood, and George Pólya. 1952. *Inequalities. By GH Hardy, JE Littlewood, G. Pólya.*. University Press.

[12] Daniel G Horvitz and Donovan J Thompson. 1952. A generalization of sampling without replacement from a finite universe. *Journal of the American statistical Association* 47, 260 (1952), 663–685.

[13] Yifan Hou, Jian Zhang, James Cheng, Kaili Ma, Richard TB Ma, Hongzhi Chen, and Ming-Chang Yang. 2020. Measuring and improving the use of graph information in graph neural networks. In *International Conference on Learning Representations(ICLR 2020)*.

[14] Wenbing Huang, Tong Zhang, Yu Rong, and Junzhou Huang. 2018. Adaptive Sampling Towards Fast Graph Representation Learning. *Advances in Neural Information Processing Systems* 31 (2018), 4558–4567.

[15] Long Jin, Yang Chen, Pan Hui, Cong Ding, Tianyi Wang, Athanasios V Vasilakos, Beixing Deng, and Xing Li. 2011. Albatross sampling: robust and effective hybrid vertex sampling for social graphs. In *Proceedings of the 3rd ACM international workshop on MobiArch (MobiArch 2011)*. ACM, 11–16.

[16] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *International Conference on Learning Representations (ICLR 2017)*.

[17] Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. 2019. Predict then propagate: Graph neural networks meet personalized pagerank. In *International Conference on Learning Representations (ICLR 2019)*.

[18] Sang Hoon Lee, Pan-Jun Kim, and Hawoong Jeong. 2006. Statistical properties of sampled networks. *Physical Review E* 73, 1 (2006), 016102.

[19] Jure Leskovec and Christos Faloutsos. 2006. Sampling from large graphs. In *Proceedings of the 12th international conference on Knowledge discovery and data mining (KDD 2006)*. ACM, 631–636.

[20] David D Lewis. 1995. A sequential algorithm for training text classifiers: Corrigendum and additional data. In *the ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 1995)*, Vol. 29. ACM, 13–19.

[21] Qimai Li, Zhichao Han, and Xiao-Ming Wu. 2018. Deeper insights into graph convolutional networks for semi-supervised learning. In *Thirty-Second AAAI conference on artificial intelligence(AAAI 2018)*.

[22] Arun S Maiya and Tanya Y Berger-Wolf. 2011. Benefits of bias: Towards better characterization of network sampling. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD 2011)*. 105–113.

[23] Galileo Namata, Ben London, Lise Getoor, Bert Huang, and UMD EDU. 2012. Query-driven active surveying for collective classification. In *10th International Workshop on Mining and Learning with Graphs (MLG 2012)*, Vol. 8.

[24] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. 1999. *The PageRank citation ranking: Bringing order to the web.* Technical Report. Stanford InfoLab.

[25] Jiezhong Qiu, Jian Tang, Hao Ma, Yuxiao Dong, Kuansan Wang, and Jie Tang. 2018. DeepInf: Modeling influence locality in large social networks. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2018)*.

[26] Alireza Rezvanian, Behnaz Moradabadi, Mina Ghavipour, Mohammad Mehdi Daliri Khomami, and Mohammad Reza Meybodi. 2019. Social network sampling. In *Learning Automata Approach for Social Networks*. Springer, 91–149.

[27] Bruno Ribeiro and Don Towsley. 2010. Estimating and sampling graphs with multidimensional random walks. In *Proceedings of the 10th SIGCOMM conference on Internet measurement (SIGCOMM 2010)*. ACM, 390–403.

[28] Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. 2019. DropEdge: Towards Deep Graph Convolutional Networks on Node Classification. In *International Conference on Learning Representations(ICLR 2019)*.

[29] Carl-Erik Särndal, Bengt Swensson, and Jan Wretman. 2003. *Model assisted survey sampling*. Springer Science & Business Media.

[30] Jianing Sun, Wei Guo, Dengcheng Zhang, Yingxue Zhang, Florence Regol, Yaochen Hu, Huifeng Guo, Ruiming Tang, Han Yuan, Xiuqiang He, et al. 2020. A Framework for Recommending Accurate and Diverse Items Using Bayesian Graph Convolutional Neural Networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD 2020)*. 2030–2039.

[31] Laurens Van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-SNE. *Journal of machine learning research* 9, 11 (2008).

[32] Claudia Wagner, Philipp Singer, Fariba Karimi, Jürgen Pfeffer, and Markus Strohmaier. 2017. Sampling from Social Networks with Attributes. In *Proceedings of the 26th International Conference on World Wide Web (WWW 2017)*. 1181–1190.

[33] Hao Wang, Tong Xu, Qi Liu, Defu Lian, Enhong Chen, Dongfang Du, Han Wu, and Wen Su. 2019. MCNE: An end-to-end framework for learning multiple conditional network representations of social network. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD 2019)*. 1064–1072.

[34] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. 2019. Simplifying graph convolutional networks. In *International conference on machine learning (ICML 2019)*. PMLR, 6861–6871.

[35] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. 2018. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD 2018)*. 974–983.

[36] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. 2019. GraphSAINT: Graph Sampling Based Inductive Learning Method. In *International Conference on Learning Representations (ICLR 2019)*.

[37] Muhan Zhang and Yixin Chen. 2018. Link prediction based on graph neural networks. In *Advances in Neural Information Processing Systems (NIPS 2018)*. 5165–5175.

[38] Lingxiao Zhao and Leman Akoglu. 2019. PairNorm: Tackling Oversmoothing in GNNs. In *International Conference on Learning Representations(ICLR 2019)*.

[39] Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. 2018. Graph neural networks: A review of methods and applications. *arXiv preprint arXiv:1812.08434* (2018).

[40] Difan Zou, Ziniu Hu, Yewen Wang, Song Jiang, Yizhou Sun, and Quanquan Gu. 2019. Layer-dependent importance sampling for training deep and large graph convolutional networks. In *Advances in Neural Information Processing Systems (NIPS 2019)*. 11249–11259.

A PROOF OF THEOREM 4.1

The proof the theorem is based on the Horvitz-Thompson estimation theory [12]. We use $\mathrm{I}(v)$ to denote the indicator function for v being sampled in $\mathcal{V}_s$, and $\mathrm{I}(u|v)$ to indicate that u is sampled in $\mathcal{V}_s$ given that v is also in $\mathcal{V}_s$. For $v \in \mathcal{V}_s, 1 \le l < L$, according to Horvitz-Thompson estimation theory:

$$\mathbb{E}[\tilde{\xi}^l(v)] = \mathbb{E}[\sum_{u\in\mathcal{V}_s} \frac{\tilde{A}(v,u)}{\pi_{u,v}/\pi_v}\tilde{h}^{l-1}(u)] = \sum_{u\in\mathcal{V}} \frac{\tilde{A}(v,u)}{\pi_{u,v}/\pi_v}\tilde{h}^{l-1}(u)\mathbb{E}[\mathrm{I}(u|v)] = \sum_{u\in\mathcal{V}} \frac{\tilde{A}(v,u)}{\pi_{u,v}/\pi_v}\tilde{h}^{l-1}(u)\pi_{u|v} = \sum_{u\in\mathcal{V}} \frac{\tilde{A}(v,u)}{\pi_{u,v}/\pi_v}\tilde{h}^{l-1}(u)\frac{\pi_{u,v}}{\pi_v} = \xi^l(v).$$

Similarly, for $v \in \mathcal{V}_a, l = L$,

$$\mathbb{E}[\tilde{\xi}^L(v)] = \mathbb{E}[\sum_{u\in\mathcal{V}_s} \frac{\tilde{A}(v,u)}{\pi_u}\tilde{h}^{L-1}(u)] = \sum_{u\in\mathcal{V}} \frac{\tilde{A}(v,u)}{\pi_u}\tilde{h}^{L-1}(u)\mathbb{E}[\mathrm{I}(u)] = \sum_{u\in\mathcal{V}} \frac{\tilde{A}(v,u)}{\pi_u}\tilde{h}^{L-1}(u)\pi_u = \sum_{u\in\mathcal{V}} \frac{\tilde{A}(v,u)}{\pi_{u,v}/\pi_v}\tilde{h}^{l-1}(u)\frac{\pi(u,v)}{\pi(v)} = \xi^L(v).$$

Note that in the equation for the estimation of $\tilde{\xi}^l(v)(v \in \mathcal{V}_s, 1 \le l < L)$, it refers to the second order inclusion probability since the calculation implies the objective node v should be also included in $\mathcal{V}_s$. While the estimation of $\tilde{\xi}^L(v)(v \in \mathcal{V}_a)$ refers to the first order inclusion probability for the reason that the existence of $\mathcal{V}_a$ is an established fact and it is not depended on $\mathcal{V}_s$.

B PROOF OF THEOREM 4.2

The proof of the theorem is also based on the Horvitz-Thompson estimation theory [12]. What's more it applies the conclusion of Theorem 4.1.

$$\begin{aligned}
\mathbb{E}[\widetilde{F}(\mathcal{V}_s)] =& \mathbb{E}[\sum_{l<L}\sum_{v\in\mathcal{V}_s}\frac{\tilde{\xi}^l(v)}{\pi_v} + \sum_{v\in\mathcal{V}_a}\tilde{\xi}^L(v)] = \mathbb{E}[\sum_{l<L}\sum_{v\in\mathcal{V}}\frac{\tilde{\xi}^l(v)}{\pi_v}\mathrm{I}(v) + \sum_{v\in\mathcal{V}_a}\tilde{\xi}^L(v)] = \sum_{l<L}\sum_{v\in\mathcal{V}}\mathbb{E}[\frac{\tilde{\xi}^l(v)}{\pi_v}\mathrm{I}(v)] + \sum_{v\in\mathcal{V}_a}\mathbb{E}[\tilde{\xi}^L(v)] \\
=& \sum_{l<L}\sum_{v\in\mathcal{V}}\sum_{u\in\mathcal{V}}\mathbb{E}[\frac{\tilde{A}(v,u)\tilde{h}^{l-1}(u)}{\pi_{u,v}/\pi_v}\frac{\mathrm{I}(v)\mathrm{I}(u|v)}{\pi_v}] + \sum_{v\in\mathcal{V}_a}\xi^L(v) = \sum_{l<L}\sum_{v\in\mathcal{V}}\sum_{u\in\mathcal{V}}\frac{\tilde{A}(v,u)\tilde{h}^{l-1}(u)}{\pi_{u,v}}\mathbb{E}[\mathrm{I}(v)\mathrm{I}(u|v)] + \sum_{v\in\mathcal{V}_a}\xi^L(v) \\
=& \sum_{l<L}\sum_{v\in\mathcal{V}}\sum_{u\in\mathcal{V}}\frac{\tilde{A}(v,u)\tilde{h}^{l-1}(u)}{\pi_{u,v}}\pi_{u,v} + \sum_{v\in\mathcal{V}_a}\xi^L(v) = \sum_{l<L}\sum_{v\in\mathcal{V}}[\sum_{u\in\mathcal{V}}\tilde{A}(v,u)\tilde{h}^{l-1}(u)] + \sum_{v\in\mathcal{V}_a}\xi^L(v) = \sum_{l<L}\sum_{v\in\mathcal{V}}\xi^l(v) + \sum_{v\in\mathcal{V}_a}\xi^L(v) = F.
\end{aligned}$$

C PROOF OF THEOREM 4.3

We ignore the possible self-connections in the graph and $\widetilde{F}(\mathcal{V}_s)$ can be presented as:

$$\widetilde{F}(\mathcal{V}_s) = \sum_{l<L}\sum_{v\in\mathcal{V}_s}\frac{\tilde{\xi}^l(v)}{\pi_v} + \sum_{v\in\mathcal{V}_a}\tilde{\xi}^L(v) = \sum_{l<L}\sum_{v\in\mathcal{V}}\sum_{u\in\mathcal{V}}\frac{\tilde{A}(v,u)\tilde{h}^{l-1}(u)}{\pi_{u,v}}\mathrm{I}(u,v) + \sum_{v\in\mathcal{V}_a}\sum_{u\in\mathcal{V}}\frac{\tilde{A}(v,u)\tilde{h}^{L-1}(u)}{\pi_u}\mathrm{I}(u).$$

For convenience, we denote $\mathcal{H} = \sum_{l<L}\sum_{v\in\mathcal{V}_s}\frac{\tilde{\xi}^l(v)}{\pi_v}$, $\mathcal{R} = \sum_{v\in\mathcal{V}_a}\tilde{\xi}^L(v)$, and $t^l_{u,v} = \tilde{A}(v,u)\tilde{h}^{l-1}(u)$ $(1 \le l \le L)$. We have:

$$\mathcal{H} = \sum_{l<L}\sum_{v\in\mathcal{V}}\sum_{u\in\mathcal{V}}\frac{t^l_{u,v}}{\pi_{u,v}}\mathrm{I}(u,v), \quad \mathcal{R} = \sum_{v\in\mathcal{V}_a}\sum_{u\in\mathcal{V}}\frac{t^L_{u,v}}{\pi_u}\mathrm{I}(u).$$

The variance for $\widetilde{F}(\mathcal{V}_s)$ can be formulated as: $\mathrm{Var}[\widetilde{F}(\mathcal{V}_s)] = \mathrm{Var}(\mathcal{H}+\mathcal{R}) = \mathrm{Var}(\mathcal{H}) + \mathrm{Var}(\mathcal{R}) + 2\mathrm{Cov}(\mathcal{R},\mathcal{H})$.

(1) For the calculation of $\mathrm{Var}(\mathcal{R})$, we have: $\mathrm{Var}(\mathcal{R}) = \sum_{v_1\in\mathcal{V}_a}\sum_{v_2\in\mathcal{V}_a}\sum_{u_1\in\mathcal{V}}\sum_{u_2\in\mathcal{V}}\frac{t^L_{u_1,v_1}t^L_{u_2,v_2}}{\pi_{u_1}\pi_{u_2}}\mathrm{Cov}[\mathrm{I}(u_1),\mathrm{I}(u_2)]$.

The term $\mathrm{Cov}[\mathrm{I}(u_1),\mathrm{I}(u_2)]$ is non-zero only when nodes u_1 and u_2 are the same. Hence $\frac{\mathrm{Cov}[\mathrm{I}(u_1),\mathrm{I}(u_2)]}{\pi_{u_1}\pi_{u_2}} = \frac{1}{\pi_{u_1}} - 1$, and we have:

$$\mathrm{Var}(\mathcal{R}) = \sum_{u\in\mathcal{V}}(\sum_{v\in\mathcal{V}_a}\frac{t^L_{u,v}}{\sqrt{\pi_u}})^2 - \sum_{u\in\mathcal{V}}(\sum_{v\in\mathcal{V}_a}t^L_{u,v})^2.$$

(2) For the calculation of $\mathrm{Var}(\mathcal{H})$, we make $\tilde{t}^l_{u,v} = t^l_{u,v} + t^l_{v,u}$ and thus, the value of variable $\tilde{t}^l_{u,v}$ has nothing to do with the order of u, v, namely $\tilde{t}^l_{u,v} = \tilde{t}^l_{v,u}$. In this way, $\mathcal{H}$ can be denoted as: $\mathcal{H} = \frac{1}{2}\sum_{l<L}\sum_{v\in\mathcal{V}}\sum_{u\in\mathcal{V}}\frac{\tilde{t}^l_{u,v}}{\pi_{u,v}}\mathrm{I}(u,v)$.

Therefore, the variance of $\mathcal{H}$ can be calculated as:

$$\mathrm{Var}(\mathcal{H}) = \mathrm{Var}[\frac{1}{2}\sum_{l<L}\sum_{v\in\mathcal{V}}\sum_{u\in\mathcal{V}}\frac{\tilde{t}^l_{u,v}}{\pi_{u,v}}\mathrm{I}(u,v)] = \frac{1}{4}\sum_{l_1<L}\sum_{l_2<L}\sum_{u_1\in\mathcal{V}}\sum_{u_2\in\mathcal{V}}\sum_{v_1\in\mathcal{V}}\sum_{v_2\in\mathcal{V}}\frac{\tilde{t}^{l_1}_{u_1,v_1}\tilde{t}^{l_2}_{u_2,v_2}}{\pi_{u_1,v_1}\pi_{u_2,v_2}}\mathrm{Cov}[\mathrm{I}(u_1,v_1),\mathrm{I}(u_2,v_2)].$$

The term $\frac{\mathrm{Cov}[\mathrm{I}(u_1,v_1),\mathrm{I}(u_2,v_2)]}{\pi_{u_1,v_1}\pi_{u_2,v_2}}$ means the covariance between $\mathrm{I}(u_1,v_1)$ and $\mathrm{I}(u_2,v_2)$ divided by their existence probabilities. We approximate this term as follows:

$$\frac{\mathrm{Cov}[\mathrm{I}(u_1,\,v_1),\,\mathrm{I}(u_2,\,v_2)]}{\pi_{u_1,v_1}\pi_{u_2,v_2}} = \frac{\mathrm{Cov}[\mathrm{I}(u_1)\mathrm{I}(v_1),\,\mathrm{I}(u_2)\mathrm{I}(v_2)]}{\pi_{u_1,v_1}\pi_{u_2,v_2}} \doteq \frac{\mathrm{Cov}[\pi_{u_1}\mathrm{I}(v_1)+\pi_{v_1}\mathrm{I}(u_1)-\pi_{u_1}\pi_{v_1},\,\pi_{u_2}\mathrm{I}(v_2)+\pi_{v_2}\mathrm{I}(u_2)-\pi_{u_2}\pi_{v_2}]}{\pi_{u_1}\pi_{v_1}\pi_{u_2}\pi_{v_2}}$$
$$= \frac{\mathrm{Cov}[\mathrm{I}(u_1),\,\mathrm{I}(u_2)]}{\pi_{u_1}\pi_{u_2}} + \frac{\mathrm{Cov}[\mathrm{I}(u_1),\,\mathrm{I}(v_2)]}{\pi_{u_1}\pi_{v_2}} + \frac{\mathrm{Cov}[\mathrm{I}(v_1),\,\mathrm{I}(u_2)]}{\pi_{v_1}\pi_{u_2}} + \frac{\mathrm{Cov}[\mathrm{I}(v_1),\,\mathrm{I}(v_2)]}{\pi_{v_1}\pi_{v_2}}.$$

The denominator is approximated by assuming the nodes in the second order inclusion probability independent with each other and thus $\pi_{u,v} \doteq \pi_u\pi_v$. Note that if we use Poisson sampling method, such formula is strictly equal. While for the numerator, we make use of binary Taylor Expansion in point (π_u, π_v), namely

$$\mathrm{I}(u)\mathrm{I}(v) = \pi_u\pi_v + [\mathrm{I}(u)-\pi_u]\pi_v + [\mathrm{I}(v)-\pi_v]\pi_u + R_1 \doteq \pi_v\mathrm{I}(u) + \pi_u\mathrm{I}(v) - \pi_u\pi_v.$$

The term $\mathrm{Cov}[\mathrm{I}(u_1,v_1),\mathrm{I}(u_2,v_2)]$ is non-zero only on the following general conditions:

- When $u_1 = u_2, v_1 \neq v_2$:
$$\frac{\mathrm{Cov}[\mathrm{I}(u_1,\,v_1),\,\mathrm{I}(u_2,\,v_2)]}{\pi_{u_1,v_1}\pi_{u_2,v_2}} \doteq \frac{\mathrm{Cov}[\mathrm{I}(u_1),\,\mathrm{I}(u_2)]}{\pi_{u_1}\pi_{u_2}} = \frac{1}{\pi_{u_1}} - 1.$$
- When $u_1 = u_2, v_1 = v_2$:
$$\frac{\mathrm{Cov}[\mathrm{I}(u_1,\,v_1),\,\mathrm{I}(u_2,\,v_2)]}{\pi_{u_1,v_1}\pi_{u_2,v_2}} \doteq \frac{\mathrm{Cov}[\mathrm{I}(u_1),\,\mathrm{I}(u_2)]}{\pi_{u_1}\pi_{u_2}} + \frac{\mathrm{Cov}[\mathrm{I}(v_1),\,\mathrm{I}(v_2)]}{\pi_{v_1}\pi_{v_2}} = \frac{1}{\pi_{u_1}} + \frac{1}{\pi_{v_1}} - 2.$$

Therefore we can derive:

$$\mathrm{Var}(\mathcal{H}) \doteq \sum_{u\in\mathcal{V}}\Big(\sum_{l<L}\sum_{v\in\mathcal{V}}\frac{\tilde{t}^l_{u,v}}{\sqrt{\pi_u}}\Big)^2 - \sum_{u\in\mathcal{V}}\Big(\sum_{l<L}\sum_{v\in\mathcal{V}}\tilde{t}^l_{u,v}\Big)^2.$$

(3) For the calculation of $2\mathrm{Cov}(\mathcal{R},\mathcal{H})$, similarly we have:

$$2\mathrm{Cov}(\mathcal{R},\,\mathcal{H}) = 2\mathrm{Cov}\Big[\sum_{v\in\mathcal{V}_a}\sum_{u\in\mathcal{V}}\frac{t^L_{u,v}}{\pi_u}\mathrm{I}(u),\,\frac{1}{2}\sum_{l<L}\sum_{v\in\mathcal{V}}\sum_{u\in\mathcal{V}}\frac{\tilde{t}^l_{u,v}}{\pi_{u,v}}\mathrm{I}(u,\,v)\Big] = \sum_{v_1\in\mathcal{V}_a}\sum_{u_1\in\mathcal{V}}\sum_{l<L}\sum_{v_2\in\mathcal{V}}\sum_{u_2\in\mathcal{V}}\frac{t^L_{u_1,v_1}\tilde{t}^l_{u_2,v_2}}{\pi_{u_1}\pi_{u_2,v_2}}\mathrm{Cov}[\mathrm{I}(u_1),\,\mathrm{I}(u_2,\,v_2)].$$

The term $\mathrm{Cov}[\mathrm{I}(u_1),\mathrm{I}(u_2,v_2)]$ is non-zero only on two general situations that $u_1 = u_2$ or $u_1 = v_2$. In such cases, it is easy to derive $\frac{\mathrm{Cov}[\mathrm{I}(u_1),\mathrm{I}(u_2,v_2)]}{\pi_{u_1}\pi_{u_2,v_2}} = \frac{1}{\pi_{u_1}} - 1$. Since $\frac{\tilde{t}^l_{u_2,v_2}}{\pi_{u_2,v_2}} = \frac{\tilde{t}^l_{v_2,u_2}}{\pi_{v_2,u_2}}$, we can derive the following formulation:

$$2\mathrm{Cov}(\mathcal{R},\,\mathcal{H}) = 2\sum_{u\in\mathcal{V}}\sum_{l<L}\sum_{v_1\in\mathcal{V}_a}\sum_{v_2\in\mathcal{V}}\frac{t^L_{u,v_1}\tilde{t}^l_{u,v_2}}{\pi_u} - 2\sum_{u\in\mathcal{V}}\sum_{l<L}\sum_{v_1\in\mathcal{V}_a}\sum_{v_2\in\mathcal{V}}t^L_{u,v_1}\tilde{t}^l_{u,v_2}$$

Combining the calculation of (1)(2)(3), we can finally derive the formulation of variance $\widetilde{F}(\mathcal{V}_s)$ as:

$$\begin{aligned}
&\mathrm{Var}[\widetilde{F}(\mathcal{V}_s)] = \mathrm{Var}(\mathcal{H}) + \mathrm{Var}(\mathcal{R}) + 2\mathrm{Cov}(\mathcal{R},\,\mathcal{H})\\
&\doteq \sum_{u\in\mathcal{V}}\Big(\sum_{v\in\mathcal{V}_a}\frac{t^L_{u,v}}{\sqrt{\pi_u}}\Big)^2 - \sum_{u\in\mathcal{V}}\Big(\sum_{v\in\mathcal{V}_a}t^L_{u,v}\Big)^2 + \sum_{u\in\mathcal{V}}\Big(\sum_{l<L}\sum_{v\in\mathcal{V}}\frac{\tilde{t}^l_{u,v}}{\sqrt{\pi_u}}\Big)^2 - \sum_{u\in\mathcal{V}}\Big(\sum_{l<L}\sum_{v\in\mathcal{V}}\tilde{t}^l_{u,v}\Big)^2 + 2\sum_{u\in\mathcal{V}}\sum_{l<L}\sum_{v_1\in\mathcal{V}_a}\sum_{v_2\in\mathcal{V}}\frac{t^L_{u,v_1}\tilde{t}^l_{u,v_2}}{\pi_u} - 2\sum_{u\in\mathcal{V}}\sum_{l<L}\sum_{v_1\in\mathcal{V}_a}\sum_{v_2\in\mathcal{V}}t^L_{u,v_1}\tilde{t}^l_{u,v_2}\\
&= \sum_{u\in\mathcal{V}}\Big(\sum_{v\in\mathcal{V}_a}\frac{t^L_{u,v}}{\sqrt{\pi_u}}\Big)^2 + \sum_{u\in\mathcal{V}}\Big(\sum_{l<L}\sum_{v\in\mathcal{V}}\frac{\tilde{t}^l_{u,v}}{\sqrt{\pi_u}}\Big)^2 + 2\sum_{u\in\mathcal{V}}\sum_{l<L}\sum_{v_1\in\mathcal{V}_a}\sum_{v_2\in\mathcal{V}}\frac{t^L_{u,v_1}\tilde{t}^l_{u,v_2}}{\pi_u} - \Big[\sum_{u\in\mathcal{V}}\Big(\sum_{v\in\mathcal{V}_a}t^L_{u,v}\Big)^2 + \sum_{u\in\mathcal{V}}\Big(\sum_{l<L}\sum_{v\in\mathcal{V}}\tilde{t}^l_{u,v}\Big)^2 + 2\sum_{u\in\mathcal{V}}\sum_{l<L}\sum_{v_1\in\mathcal{V}_a}\sum_{v_2\in\mathcal{V}}t^L_{u,v_1}\tilde{t}^l_{u,v_2}\Big]\\
&= \sum_{u\in\mathcal{V}}\Big[\Big(\sum_{v\in\mathcal{V}_a}\frac{t^L_{u,v}}{\sqrt{\pi_u}}\Big)^2 + \Big(\sum_{l<L}\sum_{v\in\mathcal{V}}\frac{\tilde{t}^l_{u,v}}{\sqrt{\pi_u}}\Big)^2 + 2\sum_{l<L}\sum_{v_1\in\mathcal{V}_a}\sum_{v_2\in\mathcal{V}}\frac{t^L_{u,v_1}\tilde{t}^l_{u,v_2}}{\pi_u}\Big] - \Big\{\sum_{u\in\mathcal{V}}\Big[\Big(\sum_{v\in\mathcal{V}_a}t^L_{u,v}\Big)^2 + \Big(\sum_{l<L}\sum_{v\in\mathcal{V}}\tilde{t}^l_{u,v}\Big)^2 + 2\sum_{l<L}\sum_{v_1\in\mathcal{V}_a}\sum_{v_2\in\mathcal{V}}t^L_{u,v_1}\tilde{t}^l_{u,v_2}\Big]\Big\}\\
&= \sum_{u\in\mathcal{V}}\Big(\sum_{v\in\mathcal{V}_a}\frac{t^L_{u,v}}{\sqrt{\pi_u}} + \sum_{l<L}\sum_{v\in\mathcal{V}}\frac{\tilde{t}^l_{u,v}}{\sqrt{\pi_u}}\Big)^2 - \sum_{u\in\mathcal{V}}\Big(\sum_{v\in\mathcal{V}_a}t^L_{u,v} + \sum_{l<L}\sum_{v\in\mathcal{V}}\tilde{t}^l_{u,v}\Big)^2 = \sum_{u\in\mathcal{V}}\frac{1}{\pi_u}\Big(\sum_{v\in\mathcal{V}_a}t^L_{u,v} + \sum_{l<L}\sum_{v\in\mathcal{V}}\tilde{t}^l_{u,v}\Big)^2 - \sum_{u\in\mathcal{V}}\Big(\sum_{v\in\mathcal{V}_a}t^L_{u,v} + \sum_{l<L}\sum_{v\in\mathcal{V}}\tilde{t}^l_{u,v}\Big)^2.
\end{aligned}$$

Therefore if we denote:

$$\Psi'_u = \sum_{v\in\mathcal{V}_a}t^L_{u,v} + \sum_{l<L}\sum_{v\in\mathcal{V}}\tilde{t}^l_{u,v} = \sum_{v\in\mathcal{V}_a}\tilde{A}(v,\,u)\tilde{h}^L(u) + \sum_{l<L}\sum_{v\in\mathcal{V}}[\tilde{A}(u,\,v)\tilde{h}^l(v) + \tilde{A}(v,\,u)\tilde{h}^l(u)]$$

the variance of $\widetilde{F}(\mathcal{V}_s)$ admits:

$$\mathrm{Var}[\widetilde{F}(\mathcal{V}_s)] \doteq \sum_{u\in\mathcal{V}}\frac{\Psi'^2_u}{\pi_u} - \sum_{u\in\mathcal{V}}\Psi'^2_u.$$

For the reason that the representations $\tilde{h}^l(u)$ in the formulation can't be obtained in advance and they are continues changing during model training. As a reasonable simplification, in practical use, we ignore $\tilde{h}^l(u)$ to make the variance dependent on the graph topology only. Therefore we have

$$\mathrm{Var}[\widetilde{F}(\mathcal{V}_s)] \propto \sum_{u\in\mathcal{V}}\frac{\Psi^2_u}{\pi_u} - \sum_{u\in\mathcal{V}}\Psi^2_u \quad \text{where} \quad \Psi_u = \sum_{v\in\mathcal{V}_a}\tilde{A}(v,\,u) + (L-1)\sum_{v\in\mathcal{V}}[\tilde{A}(u,\,v) + \tilde{A}(v,\,u)]$$

Thus the theorem is proved.