Blocking-based Neighbor Sampling for Large-scale Graph Neural Networks

Kai-Lang Yao and **Wu-Jun Li***
National Key Laboratory for Novel Software Technology
Collaborative Innovation Center of Novel Software Technology and Industrialization
Department of Computer Science and Technology, Nanjing University, China
yaokl@lamda.nju.edu.cn, liwujun@nju.edu.cn

Abstract

The exponential increase in computation and memory complexity with the depth of network has become the main impediment to the successful application of graph neural networks (GNNs) on large-scale graphs like graphs with hundreds of millions of nodes. In this paper, we propose a novel neighbor sampling strategy, dubbed blocking-based neighbor sampling (BNS), for efficient training of GNNs on large-scale graphs. Specifically, BNS adopts a policy to stochastically block the ongoing expansion of neighboring nodes, which can reduce the rate of the exponential increase in computation and memory complexity of GNNs. Furthermore, a reweighted policy is applied to graph convolution, to adjust the contribution of blocked and non-blocked neighbors to central nodes. We theoretically prove that BNS provides an unbiased estimation for the original graph convolution operation. Extensive experiments on three benchmark datasets show that, on large-scale graphs, BNS is $2\times \sim 5\times$ faster than state-of-the-art methods when achieving the same accuracy. Moreover, even on the small-scale graphs, BNS also demonstrates the advantage of low time cost.

1 Introduction

Graph has been widely used for describing unstructured data in real applications such as social networks, brain networks, molecular graphs, and knowledge graphs. Edges in graphs depict the complex relationships between samples, and rich relational information between samples is contained in graphs. Making good use of the rich relational information between samples in graphs has great potential in boosting the performance of traditional machine learning methods which are mainly designed for modeling independent and identically distributed (i.i.d.) data. In addition, graph data is now widely available in many applications. Therefore, developing advanced graph learning algorithms is a topic of great interest.

Among various algorithms of representation learning for graphs, graph neural networks (GNNs) [Gori *et al.*, 2005; Bruna *et al.*, 2014] have recently become the most successful and popular ones, due to their powerful ability in modeling complex relationships between samples. Although many advanced GNN models [Kipf and Welling, 2017; Hamilton *et al.*, 2017; Velickovic *et al.*, 2018] have been proposed, most of them are limited to the successful application on small-scale graphs (e.g., graphs with hundreds of thousands of nodes). There are significant challenges in applying existing GNN methods to applications with large-scale graphs (e.g., graphs with hundreds of millions of nodes) because of the expensive computation and memory cost during the training process. Due to the iteratively dependent nature of nodes in GNNs, the number of nodes supporting the computation of output layer exponentially increases with the depth of network. Hence, the computation and memory complexity grow exponentially. Moreover, recent works [Li *et al.*, 2019; Verma and Zhang, 2020; Chen *et al.*, 2020c] show the potential to improve the performance of GNN models as the network becomes deeper, which will undoubtedly exacerbate the problem of expensive cost on large-scale graphs. Nowadays, in order to speed up the training process, it is a dominant trend to perform training on GPUs. However, many GPUs have limited graphics memory, which hinders GNN models from training with large batch size and as a result leads to a sharp increase in time cost for training.

Solutions for the above problem mainly include model simplification methods and sampling-based methods. For model simplification methods [Wu *et al.*, 2019; Klicpera *et al.*, 2019; Chen *et al.*, 2020b], the main idea is to remove the nonlinear transformation between graph convolution layers such that the graph convolution on node features can be preprocessed before training. Although model simplification methods are efficient in training, as stated in [Chen *et al.*, 2020a], it is still an open question whether simplified GNNs' expressive power can match that of the original GNNs. For sampling-based methods, existing works can be broadly categorized into node-wise sampling [Hamilton *et al.*, 2017; Chen *et al.*, 2018a; Cong *et al.*, 2020], layer-wise sampling [Chen *et al.*, 2018b; Huang *et al.*, 2018; Zou *et al.*, 2019], and subgraph sampling [Chiang *et al.*, 2019; Zeng *et al.*, 2020]. For node-wise sampling, the main idea is to sample a number of neighbors for each node of each layer in a top-down

*Corresponding author

manner. For layer-wise sampling, the main idea is to independently sample a number of nodes from a candidate set for each layer based on the importance probabilities of nodes. All connections between the nodes of two adjacent layers are used to perform approximate graph convolution. For subgraph sampling, the main idea is to sample a subgraph and feed it to GNN models before each round of mini-batch training. Although the above sampling strategies are applicable to large-scale GNNs, they have some deficiencies or limitations in terms of accuracy, total time cost, or memory cost. For example, existing node-wise sampling strategies need to sample a large number of neighbors for high accuracy, which will lead to a sharp increase in time cost. Layer-wise sampling strategies have a high time cost of preparing data (including sampling) and may suffer from sparse connection between two adjacent layers. Subgraph sampling strategies may also suffer from sparse connection in subgraphs.

In this paper, we propose a novel node-wise sampling strategy, called blocking-based neighbor sampling (BNS), for large-scale training of GNNs. The contributions of this paper are listed as follows:

- We propose a novel blocking mechanism in BNS to stochastically block the ongoing expansion of neighboring nodes, dramatically reducing the computation and memory complexity.
- We further propose a reweighted policy to adjust the contribution of blocked and non-blocked neighboring nodes to central nodes.
- We theoretically prove that BNS provides an unbiased estimation for the original graph convolution operation.
- Extensive experiments on large-scale graphs show that BNS is $2\times \sim 5\times$ faster than existing state-of-the-art methods when achieving the same accuracy. Even on the small-scale graph, BNS also demonstrates the advantage of low time cost.

2 Notations and Problem Definition

2.1 Notations

We use boldface uppercase letters, such as $\mathbf{B}$, to denote matrices. The ith row and the jth column of a matrix $\mathbf{B}$ are denoted as $\mathbf{B}_{i*}$ and $\mathbf{B}_{*j}$, respectively. B_{ij} denotes the element at the ith row and jth column in $\mathbf{B}$. $\|\mathbf{B}\|_0$ denotes the number of non-zero entries in $\mathbf{B}$. $\|\mathbf{B}\|_F$ denotes the Frobenius norm of $\mathbf{B}$.

2.2 Problem Definition

Suppose we have a graph with N nodes. Let $\mathbf{A} \in \{0,1\}^{N\times N}$ denote the adjacency matrix of the graph. $A_{ij} = 1$ denotes there exists an edge between node i and node j, and $A_{ij} = 0$ denotes there is no edge between them. Let $\mathbf{X} \in \mathbb{R}^{N\times u}$ denote the node feature matrix, where u denotes the dimension of node feature. Suppose the average number of neighbors per node in the graph is s. Suppose the mini-batch size of nodes at output layer is B. We use L to denote the layer number of GNNs.

We take GCN [Kipf and Welling, 2017] as an example to describe the problem of the exponential increase in computation and memory complexity. Let $\mathbf{A}' = \mathbf{A} + \mathbf{I}$ and $\hat{\mathbf{A}} = \mathbf{D}^{-\frac{1}{2}}\mathbf{A}'\mathbf{D}^{-\frac{1}{2}}$, where $\mathbf{D}$ denotes the diagonal degree matrix of $\mathbf{A}'$ and $D_{ii} = \sum_{j=1}^{n} A'_{ij}$. Then GCN can be formulated as follows:

$$\mathbf{Z}_{i*}^{(\ell)} = \sum\nolimits_{j\in\mathcal{N}(i)} \hat{A}_{ij}\mathbf{H}_{j*}^{(\ell-1)}, \quad \mathbf{H}_{i*}^{(\ell)} = f(\mathbf{Z}_{i*}^{(\ell)}\mathbf{W}^{(\ell)}), \quad (1)$$

where $\mathbf{H}^{(0)} = \mathbf{X}$, $f(\cdot)$ is the activation function, $\mathcal{N}(i)$ denotes the set of neighbors of node i. $\mathbf{W}^{(\ell)} \in \mathbb{R}^{r\times r}$ is a learnable parameter.

From (1), we can see that the output of a node at the Lth layer iteratively depends on the information of its $1, \cdots, L$-hop neighbors. Such an iteratively dependent nature of nodes leads to the exponential increase in computation and memory complexity with the depth of network. Let r denote the feature dimension of hidden layer. Then, the computation and memory complexity during a mini-batch training are $\mathcal{O}(s^{L-1}\cdot(sBr + Br^2))$ and $\mathcal{O}(Lr^2 + s^L\cdot Br)$, respectively.

3 Blocking-based Neighbor Sampling

In this section, we present the details of BNS. Firstly, we sample a fixed number of neighbors for each node at the current layer ℓ. Secondly, we adopt a policy to stochastically block the ongoing expansion of neighboring nodes at the preceding layers $\{1, \cdots, \ell-1\}$. Note that once a node is blocked, all its ways out to all other nodes are blocked, and it is trapped at its current position. Thirdly, after sampling finishes, reweighted graph convolution is performed to obtain the outputs, in which a reweighted policy is adopted to adjust the contribution of blocked and non-blocked neighbors to central nodes. A visual illustration of BNS is presented in Figure 1.

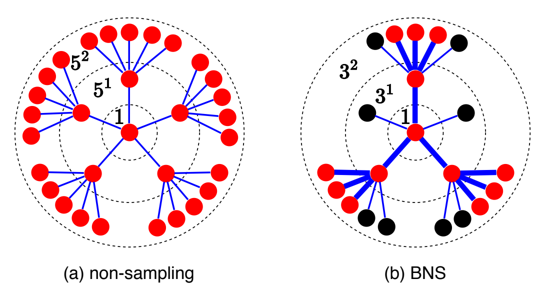


Figure 1: A visual illustration of BNS. Solid circles refer to nodes. The node within the inner dashed circle refers to the node of output layer. (a) We assume each node has 5 neighbors. (b) Black solid circles refer to blocked neighbors. The thickness of solid lines that connect two nodes indicates the magnitude of weights of nodes in reweighted graph convolution.

3.1 Sampling Algorithm

The entire sampling process is performed in a top-down manner, and it is summarized in Algorithm 1. Suppose $\mathcal{V}_{in}$ denote a mini-batch of nodes in the output layer. Firstly, we sample s_n neighbors for each node i at layer ℓ in line 5. $\text{Sample}(\mathcal{N}(i), s_n)$ in line 5 is an operation that uniformly samples s_n elements from $\mathcal{N}(i)$. Then, we randomly select $s_n \times \delta$ $(0 \leq \delta \leq 1)$ nodes from $\mathcal{N}^\ell(i)$ in line 6, and stop sampling neighbors for them at the preceding layers $\{1, \cdots, \ell - 1\}$ via the operations in line 7 and line 13. $\mathrm{Block}(\mathcal{N}^\ell(i), \delta)$ in line 6 is an operation that uniformly samples $|\mathcal{N}^\ell(i)| \times \delta$ elements from $\mathcal{N}^\ell(i)$ as blocked neighbors. $\mathcal{V}_b^\ell$ records the blocked nodes at the ℓth layer, which are used in subsequent processing steps. The operations in line 10 and line 11 ensure that the blocked nodes are mapped to the same feature space as non-blocked nodes.

Algorithm 1 Sampling Algorithm

Require: Mini-batch of nodes $\mathcal{V}_{in}$, the number of neighbors sampled for each node s_n, ratio of blocked neighbors per node δ.

Ensure: $\{(\mathcal{V}_{nb}^\ell, \mathcal{V}_b^\ell, \{(\mathcal{N}_{nb}^\ell(i), \mathcal{N}_b^\ell(i))\}_{i=1}^N)\}_{\ell=1}^L$

1: $\mathcal{V}_{nb}^L = \mathcal{V}_{in}, \mathcal{V}_b = \emptyset$
2: *Sample in a top-down manner:*
3: **for** $\ell = L : 1$ **do**
4:     **for** $i \in \mathcal{V}_{nb}^\ell$ **do**
5:         $\mathcal{N}^\ell(i) = \mathrm{Sample}(\mathcal{N}(i), s_n)$
6:         $\mathcal{N}_b^\ell(i) = \mathrm{Block}(\mathcal{N}^\ell(i), \delta)$
7:         $\mathcal{N}_{nb}^\ell(i) = \mathcal{N}^\ell(i) \backslash \mathcal{N}_b^\ell(i)$
8:     **end for**
9:     **for** $i \in \mathcal{V}_b$ **do**
10:         $\mathcal{N}^\ell(i) = \mathcal{N}^\ell(i) \cup \{i\}$
11:         $\mathcal{N}_b^\ell(i) = \mathcal{N}_b^\ell(i) \cup \{i\}$
12:     **end for**
13:     $\mathcal{V}_{nb}^{\ell-1} = \bigcup_{i \in \mathcal{V}_{nb}^\ell} \mathcal{N}_{nb}^\ell(i)$
14:     $\mathcal{V}_b^{\ell-1} = \bigcup_{i \in \mathcal{V}_{nb}^\ell} \mathcal{N}_b^\ell(i)$
15:     $\mathcal{V}_b = \mathcal{V}_b \cup \mathcal{V}_b^{\ell-1}$
16: **end for**

3.2 Reweighted Graph Convolution

We first reformulate $\mathbf{Z}_{i*}^{(\ell)}$ in Equation (1) to an expectation form:

$$\mathbf{Z}_{i*}^{(\ell)} = |\mathcal{N}(i)| \cdot \mathbb{E}_{j \sim p(j \in \mathcal{N}(i)|i)} \hat{A}_{ij} \mathbf{H}_{j*}^{(\ell-1)}, \tag{2}$$

where $p(j \in \mathcal{N}(i)|i)$ is a uniform distribution over the neighbors of node i. $|\mathcal{N}(i)|$ denotes the number of elements in $\mathcal{N}(i)$.

For blocked nodes, since their neighbor expansions are blocked, their estimation for Equation (2) is less precise (having large variance) than non-blocked nodes. We can see that representations of blocked nodes carry little information about the input graph. Therefore, it is reasonable to increase the contribution of non-blocked nodes to the central nodes. We perform reweighted graph convolution to achieve this goal.

After Algorithm 1 is performed, reweighted graph convolution is formulated as follows. For readability, we denote $n_{i,1}^\ell = |\mathcal{N}_b^\ell(i)|$, $n_{i,2}^\ell = |\mathcal{N}_{nb}^\ell(i)|$ and $n_i = |\mathcal{N}(i)|$.

$$\rho_{i,1}^\ell = \rho \cdot \frac{n_{i,1}^\ell + n_{i,2}^\ell}{n_{i,1}^\ell}, \quad \rho_{i,2}^\ell = (1-\rho) \cdot \frac{n_i^\ell + \tilde{n}_i^\ell}{n_{i,2}^\ell}, \tag{3}$$

$$\tilde{A}_{ij}^\ell = \rho_{i,1}^\ell \cdot \frac{n_i}{n_{i,1}^\ell + n_{i,2}^\ell} \cdot \hat{A}_{ij}, \quad \forall j \in \mathcal{N}_{nb}^\ell(i) \,\text{and}\, i \in \mathcal{V}_{nb}^\ell,$$

$$\tilde{A}_{ij}^\ell = \rho_{i,2}^\ell \cdot \frac{n_i}{n_{i,1}^\ell + n_{i,2}^\ell} \cdot \hat{A}_{ij}, \quad \forall j \in \mathcal{N}_b^\ell(i) \,\text{and}\, i \in \mathcal{V}_{nb}^\ell,$$

$$\tilde{A}_{ii}^\ell = n_i \cdot \hat{A}_{ii}, \quad i \in \mathcal{V}_b^\ell \backslash \mathcal{V}_{nb}^\ell,$$

$$\mathbf{Z}_{i*}^{(\ell)} \approx \sum_{j \in \mathcal{N}^{(\ell)}(i)} \tilde{A}_{ij} \mathbf{H}_{j*}^{(\ell-1)} := \tilde{\mathbf{Z}}_{i*}^{(\ell)}, \quad \forall i \in \mathcal{V}_{nb}^\ell \cup \mathcal{V}_b^\ell, \tag{4}$$

$$\mathbf{H}_{i*}^{(\ell)} = f(\tilde{\mathbf{Z}}_{i*}^{(\ell)} \mathbf{W}^{(\ell)}), \tag{5}$$

where $\rho \in [0, 1]$. Compared with $(|\mathcal{N}(i)|/|\mathcal{N}^\ell(i)|) \cdot \hat{A}_{ij}$ in Equation (2), $\tilde{A}_{ij}$ adopts a different weights, $\rho_{i,1}^\ell$ and $\rho_{i,2}^\ell$, to adjust the contribution of non-blocked and blocked nodes to node i. In the following proposition, we prove that $\tilde{\mathbf{Z}}^{(\ell)}$ is an unbiased estimation of $\mathbf{Z}_{i*}^{(\ell)}$, which makes our proposed reweighted graph convolution theoretically sound. In experiments, ρ is set to 0.5 for convenience.

Proposition 1. *Suppose $\mathbf{H}^{(\ell-1)}$ is given. If $\mathcal{N}^\ell(i)$ is uniformly sampled from $\mathcal{N}(i)$, $\mathcal{N}_b^\ell(i)$ is uniformly sampled from $\mathcal{N}^\ell(i)$ and $\rho \in [0, 1]$, then $\tilde{\mathbf{Z}}_{i*}^{(\ell)}$ defined in Equation (4) is an unbiased estimation of $\mathbf{Z}_{i*}^{(\ell)}$.*

Proof. The proof can be found in the Appendix [1]. □

3.3 Objective Function

Let $\mathcal{W} = \{\mathbf{W}^{(1)}, \cdots, \mathbf{W}^{(L)}\}$ denote the learnable parameters defined in Equation (5). $\hat{\mathbf{Y}} = \mathbf{H}^{(L)}$ denotes the output of GNN models. For multi-class classification, $f(\cdot)$ in the last layer denotes the softmax function, while it denotes the sigmoid function for multi-label classification. The objective function for BNS is formulated as follows:

$$\min_{\mathcal{W}} \sum_{i \in \mathcal{V}'} \sum_c -Y_{ic} \log \hat{Y}_{ic} + \lambda/2 \cdot \sum_\ell \|\mathbf{W}^{(\ell)}\|_F^2, \tag{6}$$

where λ is a hyper-parameter for the regularization term of parameters $\mathcal{W}$, $\mathcal{V}'$ denotes the set of nodes in training set.

3.4 Complexity Analysis

In this subsection, we compare the computation and memory complexity of different methods with those of BNS in a mini-batch training step, which is summarized in Table 1. For existing node-wise sampling methods NS [Hamilton *et al.*, 2017], VRGCN [Chen *et al.*, 2018a] and MVS-GNN [Cong *et al.*, 2020], they reduce the growth rate from s to s_n, where s_n is much smaller than s. In particular, VRGCN and MVS-GNN show that they can achieve comparable accuracy to NS with smaller s_n. For layer-wise sampling method LADIES [Zou *et al.*, 2019] and subgraph sampling method GraphSAINT [Zeng *et al.*, 2020], they reduce the computation and memory complexity to the level that is linear with the depth of network.

Although the above methods can achieve good performance in terms of accuracy, time cost, and memory cost on small-scale graphs (e.g., graphs with hundreds of thousands of nodes), they are not efficient or even not applicable for large-scale graphs (e.g., graphs with millions of nodes and hundreds of millions of nodes). Some problems and drawbacks existing in these methods are overlooked due to the

[1] The Appendix can be found in https://cs.nju.edu.cn/lwj/.

Method	Computation complexity	Memory complexity
Non-sampling [Kipf and Welling, 2017]	$\mathcal{O}\big(s^{L-1}(sBr+Br^2)\big)$	$\mathcal{O}\big(Lr^2+s^L\cdot Br\big)$
NS [Hamilton *et al.*, 2017]	$\mathcal{O}\big(s_n^{L-1}\cdot(s_nBr+Br^2)\big)$	$\mathcal{O}\big(Lr^2+s_n^L\cdot Br\big)$
VRGCN [Chen *et al.*, 2018a]	$\mathcal{O}\big(s_n^{L-1}\cdot((s_n+s)\cdot Br+Br^2)\big)$	$\mathcal{O}\big(Lr^2+s_n^{L-1}\cdot(s+s_n)Br\big)$
MVS-GNN [Cong *et al.*, 2020]	$\mathcal{O}\big(s_n^{L-1}\cdot((s_n+s)\cdot Br+Br^2)\big)$	$\mathcal{O}\big(Lr^2+s_n^{L-1}\cdot(s+s_n)Br\big)$
LADIES [Zou *et al.*, 2019]	$\mathcal{O}\big(L\cdot(s_l/N)^2\cdot\|\mathbf{A}\|_0+Ls_l\cdot r^2\big)$	$\mathcal{O}\big(Lr^2+Ls_l\cdot r\big)$
GraphSAINT [Zeng *et al.*, 2020]	$\mathcal{O}\big(L\cdot(s_g/N)^2\cdot\|\mathbf{A}\|_0+Ls_g\cdot r^2\big)$	$\mathcal{O}\big(Lr^2+Ls_g\cdot r\big)$
BNS (ours)	$\mathcal{O}\big(\tilde{s}_n^{L-1}\cdot(s_nBr+(\delta/(1-\delta)+1)\cdot Br^2)\big)$	$\mathcal{O}\big(Lr^2+\tilde{s}_n^{L-1}\cdot s_nBr\big)$

Table 1: Computation and memory complexity. s denotes the average number of neighbors per node in $\mathbf{A}$. s_n denotes the average number of neighbors sampled for each node. $\tilde{s}_n = s_n \times (1-\delta)$, where δ denotes the ratio of blocked nodes in BNS. s_l denotes the average number of nodes per layer in layer-wise sampling. s_g denotes the average number of nodes per subgraph in subgraph sampling. $B = |\mathcal{V}_{in}|$ denotes the mini-batch size of output layer. L is the number of layers in GNN models. r is the hidden dimension of networks.

lack of systematically experimental analysis on large-scale graphs. For example, even with low computation complexity, VRGCN, MVS-GNN and LADIES have a high time cost of preparing data (including sampling) before each round of mini-batch training. In addition, VRGCN brings a huge burden to the memory for storing all nodes' historical representations at each layer. MVS-GNN has the same complexity as the non-sampling method at the outer iteration, which might make the training infeasible on large-scale graphs because of running out of graphics memory. GraphSAINT faces the problem of sparse connection in subgraphs. Moreover, GraphSAINT adopts the non-sampling strategy at the evaluation and testing stage, which is also inefficient on large-scale graphs.

Similar to existing node-wise sampling methods, BNS reduces the growth rate from s to a small $\tilde{s}_n$, where $\tilde{s}_n$ denotes the number of non-blocked neighbors per node. We will show that with a small $\tilde{s}_n$, BNS can achieve comparable accuracy to NS with a large s_n, while BNS has lower computation and memory complexity. Moreover, BNS has a low time cost of preparing data before each round of mini-batch training.

4 Experiments

In this section, we compare BNS with other baselines on five node-classification datasets. BNS is implemented on the Pytorch platform [Paszke *et al.*, 2019] with Pytorch-Geometric Library [Fey and Lenssen, 2019]. All experiments are run on a NVIDIA TitanXP GPU server with 12 GB graphics memory.

4.1 Datasets

Ogbn-products, ogbn-papers100M and ogbn-proteins[2] are publicly available [Hu *et al.*, 2020]. Ogbn-products is a large-scale dataset with millions of nodes. Ogbn-papers100M is a large-scale dataset with hundreds of millions of nodes. Ogbn-proteins is a small-scale dataset with hundreds of thousands of nodes. Amazon and Yelp in GraphSAINT, are also used for evaluation. Due to space limitation, the information and results on Amazon and Yelp are moved to the Appendix. The statistics of datasets can be found in the Appendix.

[2]https://ogb.stanford.edu/docs/nodeprop/

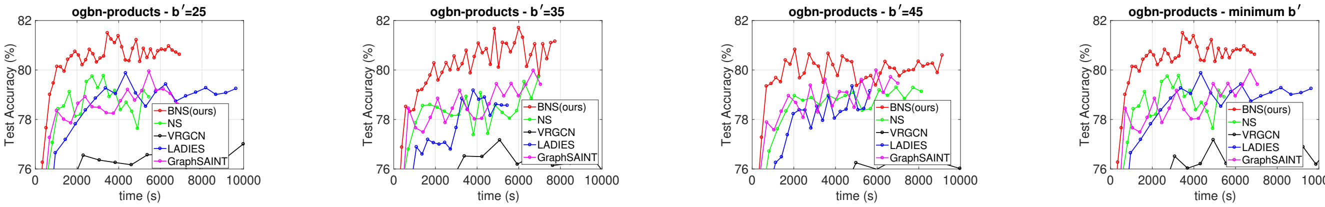
4.2 Baselines and Settings

We compare BNS with VRGCN [Chen *et al.*, 2018a], LADIES [Zou *et al.*, 2019] and GraphSAINT [Zeng *et al.*, 2020], which are the state-of-the-art methods with node-wise sampling, layer-wise sampling and subgraph sampling, respectively. Additionally, we compare BNS with the classical node-wise sampling method NS [Hamilton *et al.*, 2017]. We do not compare BNS with MVS-GNN since MVS-GNN adopts the non-sampling strategy for training at the outer iteration, which leads to the problem of running out of graphics memory. Besides, comparisons with model simplification methods are moved to the Appendix due to space limitation. Since the original implementations of the above baselines cannot directly scale to the benchmark datasets in this paper, we re-implement them according to the corresponding authors' codes. For a fair comparison, implementations of all methods, including BNS, only differ in the sampling process. For all methods, GNN model is instantiated with GraphSAGE [Hamilton *et al.*, 2017], since it can achieve good performance on the benchmark datasets. Note that sampling strategies and settings during inference are the same as those in the training stage for all methods except for GraphSAINT.

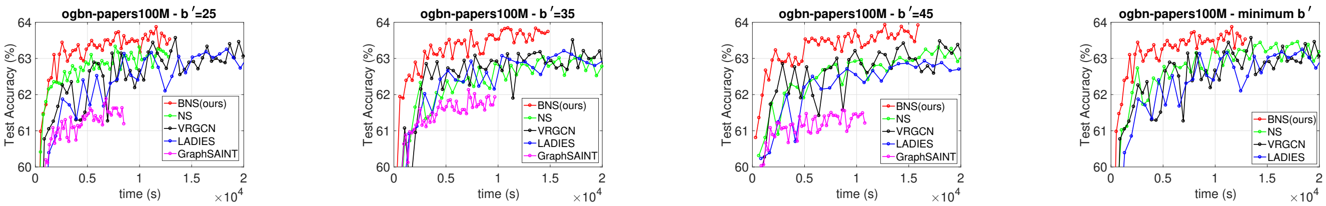
The hyper-parameters r, L, T (maximum epoch), λ and p (probability of dropout) are independent of sampling strategies, and hence they are set to be the same for different sampling strategies on one specific dataset. Empirically, r is set to 128 on all datasets, L is set to 5 on both ogbn-proteins and ogbn-products, and L is set to 3 on ogbn-papers100M. For T, it is set to 100 on both ogbn-products and ogbn-papers100M, and set to 1,000 on ogbn-proteins. For λ and p, the values of them are obtained by tuning with NS on the benchmark datasets. On ogbn-product, $\lambda = 5\times10^{-6}$ and $p = 0.1$. On ogbn-papers100M, $\lambda = 5\times10^{-7}$ and $p = 0.1$. On ogbn-proteins, $\lambda = 0$ and $p = 0$. In BNS, we set ρ to 0.5 for convenience and do not tune it. Adam [Kingma and Ba, 2015] is used to optimize the model and the learning rate η is set to 0.01. For all settings, experiments are run for 10 times with different initialization each time, and the mean results of 10 runs are reported.

4.3 Evaluation Criteria

The ultimate goal of sampling strategies for GNNs is to obtain high accuracy with a low time cost, not just to reduce time and memory cost to extreme cases at the expense of sac-



(a) ogbn-products.



(b) ogbn-papers100M.

Figure 2: Test accuracy curves on ogbn-products and ogbn-papers100M. Methods that need more than one day to obtain the curves are omitted in the figures. $b' = |\mathcal{V}'|/B$, where $|\mathcal{V}'|$ denotes the number of nodes in training data and B is batch size. At each row, the first three figures present the results of the first experimental setting in Section 4.3. The last figure presents the results of the second experimental setting.

Methods	ogbn-products			ogbn-papers100M		
	Accuracy (%) ↑	Time (s) ↓	T1+T2 (s)	Accuracy (%) ↑	Time (s) ↓	T1+T2 (s)
NS	78.64 ± 0.17	5.5×10^3	$4.5 \times 10^3 + 9.6 \times 10^2$	63.61 ± 0.13	2.5×10^4	$8.0 \times 10^3 + 1.7 \times 10^4$
VRGCN	77.07 ± 0.49	1.2×10^4	$1.1 \times 10^4 + 1.5 \times 10^3$	63.34 ± 0.12	2.2×10^4	$7.0 \times 10^3 + 1.5 \times 10^4$
LADIES	78.96 ± 0.50	4.7×10^3	$4.5 \times 10^3 + 2.5 \times 10^2$	63.25 ± 0.21	2.5×10^4	$1.2 \times 10^4 + 1.3 \times 10^4$
GraphSAINT	78.95 ± 0.41	7.1×10^3	$4.5 \times 10^3 + 2.6 \times 10^3$	61.60 ± 0.12	2.1×10^4	$8.0 \times 10^3 + 1.3 \times 10^4$
BNS (ours)	$\mathbf{80.14 \pm 0.27}$	$\mathbf{9.1 \times 10^2}$	$7.3 \times 10^2 + 1.8 \times 10^2$	$\mathbf{63.88 \pm 0.12}$	$\mathbf{1.2 \times 10^4}$	$4.3 \times 10^3 + 7.7 \times 10^3$

Table 2: Results on ogbn-products and ogbn-papers100M. Boldface letters denote the best results. Time presented in tables denotes the total training time of one run. "T1" refers to the time cost of preparing data. "T2" refers to the time cost of performing forward and backward propagation. The results in tables are obtained under the second experimental setting in the Section 4.3.

rificing accuracy. In most cases, reducing memory can also reduce the time cost since GNN model can perform training with a larger batch size when the graphics memory cost is lower. Hence, we omit the comparison of memory cost in experiments. In a nutshell, the accuracy of GNN model and time cost during training are presented to evaluate the performance of different methods.

One reasonable way to evaluate the performance of different methods is to compare time cost when achieving the same accuracy. Since batch size has an important impact on time cost and accuracy, we design two kinds of experiments for fair comparison:

- The first experimental setting: On each dataset, for different methods, we train GNN model with the same batch size. All methods are run with the best setting that can achieve the best accuracy in this case.
- The second experimental setting: On each dataset, for different methods, we train GNN model with the maximum batch size that can achieve the best accuracy. All methods are run with the best setting that can achieve the best accuracy.

Detailed settings of each method can be found in the Appendix.

4.4 Results

Results on ogbn-products and ogbn-papers100M are summarized in Figure 2 and Table 2, from which we can draw the following conclusions. Firstly, when achieving the same accuracy under different settings, BNS is faster than all other methods. For example, from Figure 2(a), we can see that BNS is approximately $4\times \sim 5\times$ faster than GraphSAINT (second-best) when achieving the accuracy of 80% on ogbn-products. From Figure 2(b), we can see that BNS is approximately $2\times$ faster than NS (second-best) when achieving the accuracy of 63.5% on ogbn-papers100M. Secondly, compared with other methods, BNS can achieve the best performance in accuracy with the minimum time cost. This point can be drawn from Table 2, which is consistent with the results in Figure 2. Thirdly, VRGCN and LADIES have a high time cost in preparing data, which is even higher than the time cost in performing forward and backward propagation. Finally, from Table 2, we observe an interesting phenomenon, i.e., the accuracy of BNS does not decrease compared to that of NS, and is even higher than that of NS. This phenomenon can be explained by the observations in JK-Net [Xu *et al.*, 2018], i.e., it is important to enhance the influence of local neighborhoods on the central nodes; otherwise the local information of the central nodes in the input graphs will be washed out in a few

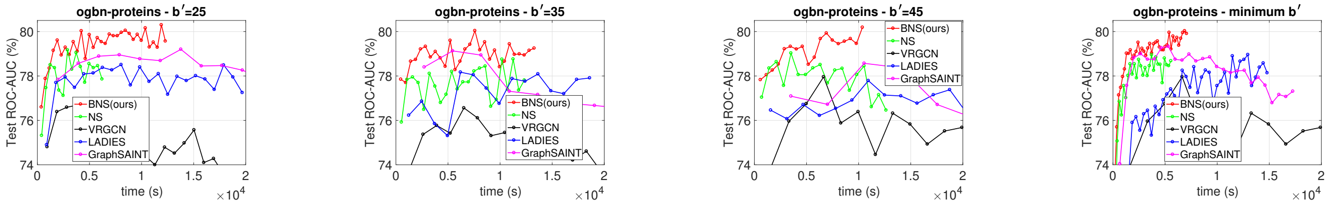


Figure 3: Test ROC-AUC curves on ogbn-proteins.

Method	Accuracy (%) ↑	Time (s) ↓	T1+T2 (s)
NS	78.84 ± 0.32	1.1×10^4	$3.6 \times 10^3 + 7.4 \times 10^3$
VRGCN	78.26 ± 0.64	1.9×10^4	$8.0 \times 10^3 + 1.1 \times 10^4$
LADIES	79.49 ± 0.37	1.9×10^4	$1.9 \times 10^4 + 3.0 \times 10^2$
GraphSAINT	78.73 ± 0.45	3.3×10^4	$1.1 \times 10^4 + 2.2 \times 10^4$
BNS (ours)	$\mathbf{79.60 \pm 0.29}$	$\mathbf{9.3 \times 10^3}$	$8.6 \times 10^3 + 7.4 \times 10^2$

Table 3: Results on ogbn-proteins.

steps. We can see that the stochastically blocking policy is helpful for BNS to preserve local information around central nodes.

Results on ogbn-proteins, a relative small graph, are summarized in Figure 3 and Table 3, from which we can draw the following conclusions. Firstly, BNS is faster than all other methods when achieving the same accuracy under different settings. However, the gap of the time cost for achieving the same accuracy between BNS and other methods is small. The main reason is that neighboring expansions can easily cover the entire graph within a few layers or steps on small-scale graphs. Therefore, these methods have the same order of computation and memory complexity $\mathcal{O}(Nsr + Nr^2)$. Secondly, BNS can achieve the best performance in terms of accuracy with the fastest speed. This point can be drawn from Table 3, which is consistent with results in Figure 3. Thirdly, once again, we observe that LADIES has a high time cost in preparing data. Finally, we observe that GraphSAINT (non-sampling strategy) achieves lower accuracy than NS, LADIES and BNS. This may be caused by the over-smoothing problem of GNNs [Li *et al.*, 2018; Xu *et al.*, 2018; Oono and Suzuki, 2020]. This observation, in turn, shows that the stochasticity introduced by sampling can alleviate the over-smoothing problem of GNNs.

Summary. First, on large-scale graphs, BNS is $2\times \sim 5\times$ faster than existing state-of-the-art methods when achieving the same accuracy. Compared with BNS, other methods have some deficiencies or limitations. For example, NS needs a large number of s_n to achieve high accuracy. VRGCN and LADIES have a high time cost of preparing data, which are more expensive than performing forward and backward propagation. Second, even on the small-scale graph, BNS demonstrates the advantage of low time cost. Third, compared with other methods, BNS can achieve the best performance in accuracy with the minimum time cost.

4.5 Ablation Study

We study the effectiveness of reweighted policy by setting $\rho_{i,1}^{\ell} = 1$ and $\rho_{i,2}^{\ell} = 1$ in Equation (3). With $\rho_{i,1}^{\ell} = 1$ and $\rho_{i,2}^{\ell} = 1$ in Equation (3), Equation (5) is a plain Monte-Carlo approximation of Equation (2). The results are presented in Table 4. From Table 4, we can conclude that reweighted policy enhances the ability of BNS in utilizing the information of blocked neighbors.

Methods	Accuracy (%) or ROC-AUC (%)		
	ogbn-products	ogbn-papers100M	ogbn-proteins
BNS w/o rew	79.12 ± 0.19	62.54 ± 0.14	79.40 ± 0.21
BNS	80.14 ± 0.27	63.88 ± 0.12	79.60 ± 0.29

Table 4: Ablation study on reweighted policy. 'w/o rew' means BNS runs without reweighted policy.

5 Conclusions

On large-scale graphs (e.g., graphs with hundreds of millions of nodes), existing sampling strategies have deficiencies or limitations in accuracy, time cost, or memory cost. Hence, designing an effective sampling strategy for efficient training of GNNs on large-scale graphs is still challenging. In this paper, we propose a novel neighbor sampling strategy, dubbed blocking-based neighbor sampling (BNS), for training GNNs on large-scale graphs. The main idea is to adopt a policy to stochastically block the ongoing expansion of neighbors, by which computation and memory complexity can be significantly reduced. Furthermore, reweighted graph convolution is proposed to adjust the contribution of blocked and non-blocked neighbors to central nodes. Extensive experiments on large-scale graphs show that, when achieving the same accuracy, BNS is $2\times \sim 5\times$ faster than state-of-the-art methods. Experiments on the small-scale graph also demonstrate the advantage of BNS in terms of time cost.

Acknowledgments

This work is supported by the NSFC-NRF Joint Research Project (No. 61861146001) and NSFC Project (No. 61921006).

References

[Bruna *et al.*, 2014] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. In *International Conference on Learning Representations*, 2014.

[Chen *et al.*, 2018a] Jianfei Chen, Jun Zhu, and Le Song. Stochastic training of graph convolutional networks with variance reduction. In *International Conference on Machine Learning*, 2018.

[Chen *et al.*, 2018b] Jie Chen, Tengfei Ma, and Cao Xiao. FastGCN: fast learning with graph convolutional networks via importance sampling. In *International Conference on Learning Representations*, 2018.

[Chen *et al.*, 2020a] Lei Chen, Zhengdao Chen, and Joan Bruna. On graph neural networks versus graph-augmented MLPs. *CoRR*, abs/2010.15116, 2020.

[Chen *et al.*, 2020b] Ming Chen, Zhewei Wei, Bolin Ding, Yaliang Li, Ye Yuan, Xiaoyong Du, and Ji-Rong Wen. Scalable graph neural networks via bidirectional propagation. In *Advances in Neural Information Processing Systems*, 2020.

[Chen *et al.*, 2020c] Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. Simple and deep graph convolutional networks. In *International Conference on Machine Learning*, 2020.

[Chiang *et al.*, 2019] Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. Cluster-GCN: an efficient algorithm for training deep and large graph convolutional networks. In *ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019.

[Cong *et al.*, 2020] Weilin Cong, Rana Forsati, Mahmut T. Kandemir, and Mehrdad Mahdavi. Minimal variance sampling with provable guarantees for fast training of graph neural networks. In *ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020.

[Fey and Lenssen, 2019] Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *International Conference on Learning Representations Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[Gori *et al.*, 2005] M. Gori, G. Monfardini, and F. Scarselli. A new model for learning in graph domains. In *IEEE International Joint Conference on Neural Networks*, 2005.

[Hamilton *et al.*, 2017] William L. Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Adavances in Neural Information Processing Systems*, 2017.

[Hu *et al.*, 2020] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: datasets for machine learning on graphs. In *Advances in Neural Information Processing Systems*, 2020.

[Huang *et al.*, 2018] Wen-bing Huang, Tong Zhang, Yu Rong, and Junzhou Huang. Adaptive sampling towards fast graph representation learning. In *Adavances in Neural Information Processing Systems*, 2018.

[Kingma and Ba, 2015] Diederik P. Kingma and Jimmy Ba. Adam: a method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

[Kipf and Welling, 2017] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017.

[Klicpera *et al.*, 2019] Johannes Klicpera, Stefan Weißenberger, and Stephan Günnemann. Diffusion improves graph learning. In *Adavances in Neural Information Processing Systems*, 2019.

[Li *et al.*, 2018] Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *AAAI Conference on Artificial Intelligence*, 2018.

[Li *et al.*, 2019] Guohao Li, Matthias Müller, Ali K. Thabet, and Bernard Ghanem. DeepGCNs: can GCNs go as deep as CNNs? In *IEEE/CVF International Conference on Computer Vision*, 2019.

[Oono and Suzuki, 2020] Kenta Oono and Taiji Suzuki. Graph neural networks exponentially lose expressive power for node classification. In *International Conference on Learning Representations*, 2020.

[Paszke *et al.*, 2019] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: an imperative style, high-performance deep learning library. In *Adavances in Neural Information Processing Systems*, 2019.

[Velickovic *et al.*, 2018] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018.

[Verma and Zhang, 2020] Saurabh Verma and Zhi-Li Zhang. Towards deeper graph neural networks. In *ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020.

[Wu *et al.*, 2019] Felix Wu, Amauri H. Souza Jr., Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Q. Weinberger. Simplifying graph convolutional networks. In *International Conference on Machine Learning*, 2019.

[Xu *et al.*, 2018] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. In *International Conference on Machine Learning*, 2018.

[Zeng *et al.*, 2020] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor K. Prasanna. GraphSAINT: graph sampling based inductive learning method. In *International Conference on Learning Representations*, 2020.

[Zou *et al.*, 2019] Difan Zou, Ziniu Hu, Yewen Wang, Song Jiang, Yizhou Sun, and Quanquan Gu. Layer-dependent importance sampling for training deep and large graph convolutional networks. In *Adavances in Neural Information Processing Systems*, 2019.

A Proofs

Proposition 1. *Suppose $\mathbf{H}^{(\ell-1)}$ is given. If $\mathcal{N}^\ell(i)$ is uniformly sampled from $\mathcal{N}(i)$, $\mathcal{N}_b^\ell(i)$ is uniformly sampled from $\mathcal{N}^\ell(i)$ and $\rho \in [0,1]$, then $\tilde{\mathbf{Z}}_{i*}^{(\ell)}$ defined in Equation (4) is an unbiased estimator of $\mathbf{Z}_{i*}^{(\ell)}$.*

Proof. Case 1. If $i \in \mathcal{V}_b^\ell \backslash \mathcal{V}_{nb}^\ell$, then we have

$$\mathbb{E}[\tilde{\mathbf{Z}}_{i*}^{(\ell)}] = \mathbb{E}[|\mathcal{N}(i)| \cdot \hat{A}_{ii}\mathbf{H}_{i*}^{(\ell-1)}] = \mathbf{Z}_{i*}^{(\ell)}.$$

Case 2. If $i \in \mathcal{V}_{nb}^\ell$, let $\mathcal{N}_{nb}^\ell(i)$ denote the set of non-blocked neighbors of node i at layer ℓ. Then we have

$$\begin{aligned}
\mathbb{E}[\tilde{\mathbf{Z}}_{i*}^{(\ell)}] &= \mathbb{E}\left[\sum_{j\in\mathcal{N}_{nb}^\ell(i)} \tilde{A}_{ij}\mathbf{H}_{j*}^{(\ell-1)} + \sum_{t\in\mathcal{N}_b^\ell(i)} \tilde{A}_{ij}\mathbf{H}_{t*}^{(\ell-1)}\right] \\
&= \mathbb{E}\left[\sum_{j\in\mathcal{N}_{nb}^\ell(i)} \tilde{A}_{ij}\mathbf{H}_{j*}^{(\ell-1)}\right] + \mathbb{E}\left[\sum_{t\in\mathcal{N}_b^\ell(i)} \tilde{A}_{ij}\mathbf{H}_{t*}^{(\ell-1)}\right].
\end{aligned}$$

For $\mathbb{E}\left[\sum_{j\in\mathcal{N}_{nb}^\ell(i)} \tilde{A}_{ij}\mathbf{H}_{j*}^{(\ell-1)}\right]$, we have

$$\begin{aligned}
\mathbb{E}\left[\sum_{j\in\mathcal{N}_{nb}^\ell(i)} \tilde{A}_{ij}\mathbf{H}_{j*}^{(\ell-1)}\right] &= \mathbb{E}\left[\sum_{j\in\mathcal{N}_{nb}^\ell(i)} \rho_{i,1}^\ell \cdot \frac{|\mathcal{N}(i)|}{|\mathcal{N}^\ell(i)|} \cdot \hat{A}_{ij}\mathbf{H}_{j*}^{(\ell-1)}\right] \\
&= \mathbb{E}\left[\sum_{j\in\mathcal{N}_{nb}^\ell(i)} \rho \cdot \frac{|\mathcal{N}(i)|}{|\mathcal{N}_{nb}^\ell(i)|} \hat{A}_{ij}\mathbf{H}_{j*}^{(\ell-1)}\right] \\
&= \rho\mathbf{Z}_{i*}^{(\ell)}.
\end{aligned}$$

For $\mathbb{E}\left[\sum_{j\in\mathcal{N}_b^\ell(i)} \tilde{A}_{ij}\mathbf{H}_{j*}^{(\ell-1)}\right]$, we have

$$\begin{aligned}
\mathbb{E}\left[\sum_{j\in\mathcal{N}_b^\ell(i)} \tilde{A}_{ij}\mathbf{H}_{j*}^{(\ell-1)}\right] &= \mathbb{E}\left[\sum_{j\in\mathcal{N}_b^\ell(i)} \rho_{i,2}^\ell \cdot \frac{|\mathcal{N}(i)|}{|\mathcal{N}^\ell(i)|} \cdot \hat{A}_{ij}\mathbf{H}_{j*}^{(\ell-1)}\right] \\
&= \mathbb{E}\left[\sum_{j\in\mathcal{N}_b^\ell(i)} (1-\rho) \cdot \frac{|\mathcal{N}(i)|}{|\mathcal{N}_b^\ell(i)|} \hat{A}_{ij}\mathbf{H}_{j*}^{(\ell-1)}\right] \\
&= (1-\rho)\mathbf{Z}_{i*}^{(\ell)}.
\end{aligned}$$

Summing up the above two parts, we have $\mathbb{E}[\tilde{\mathbf{Z}}_{i*}^{(\ell)}] = \mathbf{Z}_{i*}^{(\ell)}$. Combining case 1 and case 2 ends the proof. □

B Experimental Settings

B.1 Statistics of Datasets

The statistics of datasets are presented in Table B.1.

Datasets	ogbn-products	ogbn-papers100M	ogbn-proteins
#Nodes	2,449,029	111,059,956	132,534
#Edges	61,859,140	1,615,685,872	39,561,252
Features/Node	100	128	8
#Classes	47	172	112
#Training Nodes	196,615	1,207,179	86,619
#Validation Nodes	39,323	125,265	21,236
#Test Nodes	2,213,091	214,338	24,679
Task Type	Multi-class	Multi-class	Multi-label
Metric	Accuracy	Accuracy	ROC-AUC

Table B.1: Statistics of benchmark datasets.

B.2 Detailed Experimental Settings

Detailed experimental settings, mainly including hyper-parameters for reproducing results, are presented in Table B.2, Table B.3 and Table B.4.

Dataset	L	r	T	λ	p	η
ogbn-products	5	128	100	5×10^{-6}	0.1	0.01
ogbn-papers100M	3	128	100	5×10^{-7}	0.1	0.01
ogbn-proteins	5	128	1000	0	0	0.01

Table B.2: Hyper-parameters that are independent of different methods. L is the layer number. r is the feature dimension of hidden layer. T is the maximum epoch. λ is the hyper-parameter of the regularization term of parameters. p is the probability of dropout. η is the learning rate of Adam optimizer.

Dataset	NS	VRGCN	LADIES	GraphSAINT	BNS
ogbn-products	$s_n = 5$	-	$k = 7$	$w = 50$	$\tilde{s}_n = 3, \delta = 1/2$
ogbn-papers100M	$s_n = 10$	$s_n = 6$	$k = 15$	$w = 20$	$\tilde{s}_n = 6, \delta = 2/3$
ogbn-proteins	$s_n = 9$	$s_n = 3$	$k = 0.5$	-	$\tilde{s}_n = 4, \delta = 2/3$

Table B.3: Method-dependent hyper-parameters for generating test accuracy curves of $b' = 25, 35, 45$ in Figure 2 and Figure 3. For each method and each dataset, the settings of different b' are the same.

Dataset	NS	VRGCN	LADIES	GraphSAINT	BNS
ogbn-products	$s_n = 5, b' = 15$	$s_n = 2, b' = 15$	$k = 7, b' = 15$	$w = 50, b' = 15$	$\tilde{s}_n = 3, \delta = 1/2, b' = 5$
ogbn-papers100M	$s_n = 15, b' = 50$	$s_n = 6, b' = 30$	$k = 15, b' = 15$	$w = 50, b' = 70$	$\tilde{s}_n = 6, \delta = 2/3, b' = 25$
ogbn-proteins	$s_n = 9, b' = 8$	$s_n = 3, b' = 8$	$k = 0.5, b' = 8$	full graph, $b' = 1$	$\tilde{s}_n = 4, \delta = 2/3, b' = 8$

Table B.4: Method-dependent hyper-parameters for generating test accuracy curves of maximum b' in Figure 2 and Figure 3. Hyper-parameters in this table are the same as those for generating results in Table 3 and Table 4.

C Experiments on Amazon and Yelp

To have a fair comparison with GraphSAINT [Zeng *et al.*, 2020], we conduct experiments on Amazon and Yelp. However, our experimental results show that differences in accuracy and time cost between different methods are small. For example, NS [Hamilton *et al.*, 2017] only needs to sample two neighbors for each node to achieve the best performance in accuracy and time cost. Hence, Amazon and Yelp are not suitable for analyzing the differences between different sampling strategies.

C.1 Datasets and Settings

Datasets. Amazon has millions of nodes. Yelp has hundreds of thousands of nodes. Details of Amazon and Yelp can refer to [Zeng *et al.*, 2020].

Settings All hyper-parameters independent of sampling strategies are set the same as in [Zeng *et al.*, 2020]. For all sampling strategies, they are evaluated with the same GNN model as GraphSAINT does in [Zeng *et al.*, 2020]. Other detailed settings are presented in Table C.1.

Dataset	NS	VRGCN	LADIES	GraphSAINT	BNS
Yelp	$s_n = 2$	2	$k = 1$	$w = 2$	$\tilde{s}_n = 1, \delta = 1/2$
Amazon	$s_n = 2$	-	$k = 1$	$w = 2$	$\tilde{s}_n = 1, \delta = 1/2$

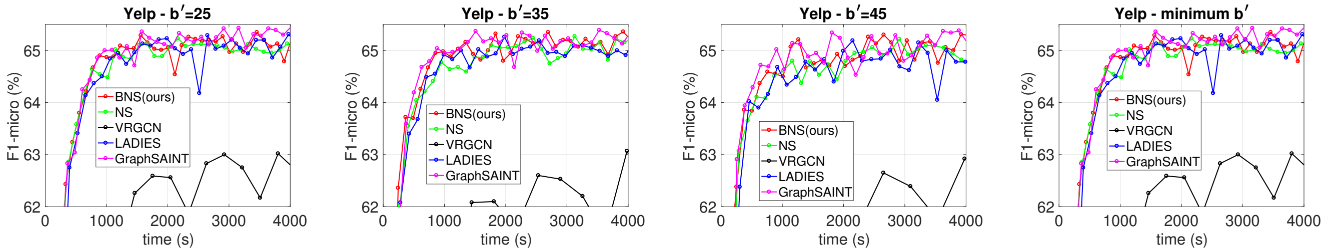
Table C.1: Method-dependent hyper-parameters for generating test accuracy curves of $b' = 25, 35, 45$ in Figure C.1. For each method and each dataset, settings of different b' are the same.

Dataset	NS	VRGCN	LADIES	GraphSAINT	BNS
Yelp	$s_n = 2, b' = 25$	$s_n = 2, b' = 25$	$k = 1, b' = 25$	$w = 2, b' = 25$	$\tilde{s}_n = 1, \delta = 1/2, b' = 25$
Amazon	$s_n = 2, b' = 25$	$s_n = 2, b' = 25$	$k = 1, b' = 25$	$w = 2, b' = 25$	$\tilde{s}_n = 1, \delta = 1/2, b' = 25$

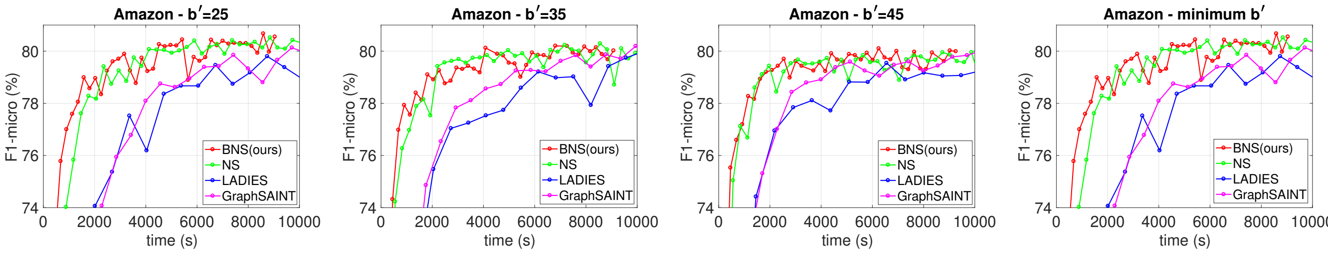
Table C.2: Method-dependent hyper-parameters for generating test accuracy curves of maximum b' in Figure C.1. Hyper-parameters in this table are the same as those for generating results in Table C.3.

C.2 Results on Amazon and Yelp

Results on Yelp and Amazon are summarized in Figure C.1 and Table C.3. On Yelp, we can see that the gap between the time cost of different methods is small when achieving the same accuracy. On Amazon, we can see that NS and BNS are approximately 2$\times$ faster than GraphSAINT and LADIES when achieving the same accuracy. But the gap between NS and BNS is small.



(a) Yelp.



(b) Amazon.

Figure C.1: Test accuracy curves on Yelp and Amazon. Methods that need more than one day to obtain the curves are omitted in the figures. $b' = |\mathcal{V}'|/b$, where $|\mathcal{V}'|$ denotes the number of nodes in training data and b is batch size. At each row, the first three figures present the results of the first experimental setting in subsection Evaluation Criteria. The last figure presents the results of the second experimental setting.

Methods	Yelp			Amazon		
	F1-micro (%) $\uparrow$	Time (s) $\downarrow$	T1+T2 (s)	F1-micro (%) $\uparrow$	Time (s) $\downarrow$	T1+T2 (s)
NS	65.39 ± 0.05	5.1×10^3	$4.0 \times 10^3 + 1.1 \times 10^3$	80.89 ± 0.08	1.1×10^4	$9.0 \times 10^3 + 2.1 \times 10^3$
VRGCN	62.58 ± 0.42	7.9×10^3	$5.1 \times 10^3 + 2.8 \times 10^3$	-	-	-
LADIES	65.36 ± 0.03	5.3×10^3	$4.6 \times 10^3 + 7.2 \times 10^2$	81.05 ± 0.08	2.7×10^4	$2.5 \times 10^4 + 2.4 \times 10^3$
GraphSAINT	65.46 ± 0.05	4.9×10^3	$3.8 \times 10^3 + 1.1 \times 10^3$	80.94 ± 0.07	2.3×10^4	$1.7 \times 10^4 + 5.9 \times 10^3$
BNS (ours)	65.34 ± 0.04	4.5×10^3	$3.5 \times 10^3 + 9.6 \times 10^2$	80.91 ± 0.08	9.0×10^3	$7.1 \times 10^3 + 1.9 \times 10^3$

Table C.3: Results on Yelp and Amazon. Time presented in tables indicates the total training time of one run. "T1" refers to the time cost of preparing data. "T2" refers to the time cost of performing forward and backward propagation. The results in tables are obtained under the second experimental setting in the subsection Evaluation Criteria.

D Comparisons with Model Simplification Methods

Model simplification methods solve the exponential increase problem of GNNs by simplifying the GNN model, which is different from sampling-based methods. Although model simplification methods are efficient in training, as stated in [Chen *et al.*, 2020], it is still an open question whether simplified GNNs' expressive power can match that of deep GNNs. Therefore, we only conduct comparisons with SGC [Wu *et al.*, 2019], and we leave the extensive evaluations on model simplification methods in future works.

The results are summarized in Table D.1. From Table D.1, we can see that SGC performs worse on ogbn-products, ogbn-proteins, Yelp and Amazon.

Methods	ogbn-products	ogbn-papers100M	ogbn-proteins	Yelp	Amazon
	Accuracy (%) $\uparrow$	Accuracy (%) $\uparrow$	ROC-AUC (%) $\uparrow$	F1-micro (%) $\uparrow$	F1-micro (%) $\uparrow$
SGC	78.18 ± 0.31	63.29 ± 0.19	72.45 ± 0.26	40.53 ± 0.04	38.80 ± 0.05
BNS (ours)	80.14 ± 0.27	63.88 ± 0.12	79.60 ± 0.29	65.34 ± 0.04	80.91 ± 0.08

Table D.1: Comparisons with SGC.

References

[Chen *et al.*, 2020] Lei Chen, Zhengdao Chen, and Joan Bruna. On graph neural networks versus graph-augmented MLPs. *CoRR*, abs/2010.15116, 2020.

[Hamilton *et al.*, 2017] William L. Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Adavances in Neural Information Processing Systems*, 2017.

[Wu *et al.*, 2019] Felix Wu, Amauri H. Souza Jr., Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Q. Weinberger. Simplifying graph convolutional networks. In *International Conference on Machine Learning*, 2019.

[Zeng *et al.*, 2020] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor K. Prasanna. GraphSAINT: graph sampling based inductive learning method. In *International Conference on Learning Representations*, 2020.