Graph-MLP Sampling

Tobias Kalmbach
tobias.kalmbach@uni-ulm.de
Ulm University
Germany

Fabian Karl
fabian.karl@uni-ulm.de
Ulm University
Germany

Jan-Lucas Störkmann
jan-lucas.stoerkmann@uni-ulm.de
Ulm University
Germany

ABSTRACT

Graph sampling is often crucial when training graph neural networks (GNNs) on large graphs. The default approach is to use random batch sampling, which may not be optimal for all datasets or tasks. In this paper, we investigate the effects of various graph sampling strategies on the performance of Graph-MLP [10]. Specifically, we evaluate thirteen different sampling methods on six benchmark datasets and compare the accuracy and runtime. Our experiments show that the performance of different sampling strategies significantly varies and that no single approach performs best on all the datasets. We therefore argue that the graph sampling method should be considered a hyperparameter and carefully optimized based on the specific needs of the task at hand.

CCS CONCEPTS

• **Mathematics of computing** → **Graph algorithms**; • **Information systems** → **Clustering and classification**; • **Computing methodologies** → **Supervised learning by classification**; *Neural networks*; • **Theory of computation** → Sample complexity and generalization bounds.

KEYWORDS

graph-mlp, sampling, node classification, graph analytics

1 INTRODUCTION

Node classification is a popular field of study, and many approaches attempt to tackle this challenge. Most commonly, Graph Neural Networks (GNNs) are employed to predict the class of a node in a graph. These GNNs use message passing to explicitly learn the structure of a given graph using feed-forward propagation. The drawback of message passing is the requirement that the adjacency matrix must be present to aggregate node features. This reduces the efficiency of these models drastically.

In contrast, Graph-MLP is a novel, inductive approach to node classification that uses a simple multi-layer perceptron to learn node representation without message passing [10]. To utilize the graph structure regardless, Hu et al. [10] add a neighboring contrastive (NContrast) loss that is applied to the training run sample. This NContrast loss aims to draw the feature representations of similar nodes together while distancing representations of dissimilar nodes. In Graph-MLP, it is assumed that connected nodes are similar and unconnected nodes are not. With this similarity measure, the NContrast loss approximates the graph structure.

To further improve efficiency and scalability, Graph-MLP uses Random Batch Sampling. This means that B randomly sampled nodes are selected in every run, and their corresponding adjacency information and node features are aggregated [10]. However, this random sampling can cause an unrepresentative sample. As the NContrast loss heavily relies on neighborhoods, the loss for nodes not connected to any other sampled nodes cannot be calculated. Graph-MLP explicitly removes the loss for any such nodes. The classification results can be shifted or distorted by removing the loss for these nodes. This problem may be avoided altogether by trying different sampling strategies, e. g., strategies that preserve the graph structure well or focus on sampling neighborhoods. We try several sampling strategies to investigate what difference sampling has on the performance of Graph-MLP.

In our work, we make the following contributions:

- We adapt Graph-MLP to use a variety of sampling strategies. Graph-MLP uses Random Batch Sampling to select a subset of nodes during training uniformly randomly. This can often lead to unconnected nodes or unrepresentative sub-graphs. Using an alternate sampling strategy might improve this.
- We further extend Graph-MLP to three new datasets: Facebook, ogbn-arxiv, and Reddit2. These are introduced to analyze the performance of Graph-MLP and the new sampling strategies on mid-sized and large datasets.
- We evaluate the selected sampling strategies on a range of graphs with different sizes and densities. This shows us which sampling strategy is the best and most impervious to the choice of the graph.

Below, we summarize the related works. Section 3 further explains the functionality of Graph-MLP and introduces our sampling strategies. The experimental apparatus is described in Section 4. Section 5 reports an overview of the achieved results, and Section 6 discusses the results before we conclude.

2 RELATED WORK

In the following, we present past work that concern GNNs that use sampling, as well as a comparison of sampling strategies of graphs. We first present GNN models that use sampling strategies. Next, we present several papers that theoretically and practically analyze graph sampling strategies. Lastly, we briefly summarize.

2.1 Sampling in Graph Neural Networks

Graph-MLP [10], the focus of our work, uses Random Batch Sampling to extract a (representative) sub-graph. Graph-MLP is not the first or only model that uses sampling. Many GNNs sample a sub-graph, which generally improves scalability and makes applying models to larger datasets possible. We present some of these in the following.

GraphSAGE [7] (**Graph SA**mple and aggre**G**at**E**) aims to learn an embedding of node features using the node's neighborhood. To get the neighborhood of a node, the authors use uniform fixed-size sampling (see also Section 3.2). This keeps the computational impact constant in every iteration and does not explode on dense graphs. E-GraphSAGE [19], an extension of GraphSAGE, does not regard the node features and samples from edges instead to aggregate edge features. Another popular approach similar to GraphSAGE is FastGCN [3]. The authors compare two sampling strategies and apply both to a Graph Convolutional Network (GCN). They argue and analyze that uniform sampling (e. g., Random Batch Sampling) may introduce high variance and

thus diminish the overall accuracy. To combat high sampling variance, Chen et al. [3] introduce an importance sampling, which alters the sampling distribution and sample nodes on every convolutional layer. GraphSAINT [38], a **Graph SA**mpling based **IN**ductive learning Me**T**hod, employs three sampling techniques to sample a sub-graph of (potentially) non-neighboring nodes. The authors find that sampling by randomly selecting edges or using random walks achieves the best results.

Both GraphSAGE [7] and GraphSAINT [38] have been extended or used as baselines in a variety of other papers and research [4, 5, 17, 20, 35, 40].

The finding that using Random Walk sampling yields good results is further supported by Chen et al. [4]. Chen et al. [4] present a model using a generalized PageRank [2] matrix created through a bidirectional propagation algorithm. As part of this propagation, Chen et al. [4] use Random Walk sampling to create an unbiased estimator for every step. Aggregating the estimators for every step results in a good approximation of a PageRank matrix, according to Chen et al. [4].

In contrast, Rossi et al. [26] specifically decide against using sampling. They argue that sampling can introduce optimization bias and instead achieve scalability of GCN using a less computationally expensive training procedure.

2.2 Comparison of sampling strategies

As seen in the section above, many GNN-based models employ sampling. Which sampling strategy they use is mostly not that carefully considered. For example, Hu et al. [10] reference trying other sampling strategies as future work. Nonetheless, many papers look into and compare sampling strategies. These analyses are presented in the following.

2.2.1 Theoretical comparison. Hu and Lau [9] provide a comprehensive survey on graph sampling. The authors also provide a taxonomy for sampling objectives, approaches, and graph types. The sampling objectives include gaining a representative subset or preserving one graph property in particular. The approaches are categorized into node, edge, and traversal-based sampling. We also use this categorization in the following. The sampling strategies are analyzed theoretically to gain a deeper understanding of graph feature preservation.

Focusing mainly on large graphs, Leskovec and Faloutsos [14] analyze various sampling strategies. The sampling strategies are evaluated using metrics describing the graph itself, e. g., in- and out-degree or the clustering coefficient distribution. The authors find that exploration sampling, e. g., random walks, preserves graph features best.

Similarly, Yousuf et al. [36] look into how five different sampling techniques affect the preservation of the original graph structure across 15 datasets. The authors find that priority-based neighborhood exploration yields the best results overall.

2.2.2 Practical comparison. The previously mentioned papers only analyze the sampling methods theoretically. Although this is an important aspect, practically applying sampling is especially interesting for future research and model improvement. For example, Wei and Hu [33] evaluate the performance of six different GNNs using four different graph sampling methods. They examine which sampling method and which network to choose for a specific task. Generally, they find that sampling does improve performance. Such improvement, however, may not always be seen depending on the dataset being used.

To investigate the accuracy and computational cost of GNNs more generally, Liu et al. [18] conduct extensive experiments using a vanilla 2-layer GCN and a variety of sampling techniques on widely used benchmark datasets. They conclude that sampling techniques enhance GCN training by lowering the cost of computation and storage.

2.2.3 Sampling in other domains. The sampling strategies above mainly focus on the use in graph analysis tasks. Many other domains also use graph sampling to achieve various goals.

In visual computing, Wu et al. [34] conduct three user studies covering five different sampling methods on how the sampling strategies influence node-link visualizations of graphs. The authors show that preserving graph characteristics strongly depends on the initial graph for most sampling methods. Similarly, Tang et al. [30] propose a visual analytics framework to measure uncertainty in different graph sampling methods. The framework's goal is to help a user better understand the uncertainty produced by different sampling methods and make an informed decision about which algorithm to use for their use case. Sampling has also been used in data mining. Iwasaki and Shudo [11] analyze different graph sampling strategies focusing on optimizing the number of queries made to an API for mining social network data.

Although many of the above studies show the potential impact of graph sampling, some graph classification models, like Graph-MLP [10], still use random node sampling due to its simplicity. A similar observation can be made in reinforcement learning. Here, random noise is commonly used despite the benefits of exploratory techniques [13].

2.3 Summary

Sampling is a common strategy used in GNNs to improve scalability and enable the use of larger datasets. This is because sampling allows the GNN to focus on a representative sub-graph of the entire graph rather than processing the whole graph at once. This can significantly reduce the computational cost and make it possible to apply the GNN to larger graphs. However, different sampling algorithms have different effects on the distinctive qualities of the graph and may not accurately represent it. This highlights the importance of carefully selecting the appropriate sampling algorithm to ensure that it accurately represents the graph and does not negatively impact the model's prediction accuracy.

Overall, the main learnings from the related work on sampling in GNNs are that it can improve scalability and enable using larger datasets. Still, it can also introduce bias in the optimization process.

3 METHODS

In the following, we describe the methods we use in our experiments. First, we describe Graph-MLP and then we describe the sampling strategies we implement and test on Graph-MLP. Next, we formally describe the problem and briefly summarize this section.

3.1 Graph-MLP

Graph-MLP is the main focus of our work. Hu et al. [10] present Graph-MLP as a more efficient alternative to GNNs. Traditionally, GNNs use message passing among neighbors to learn the given graph structure through feed-forward propagation explicitly. In contrast, Graph-MLP uses a simple multi-layer perceptron (MLP)

to circumvent explicit message passing. The specific MLP structure can be seen in Figure 1. Our MLP uses six layers, namely a linear layer, an activation layer, a normalization layer, and a dropout layer, followed by two linear layers. The last two linear layers are the prediction heads. The output of the second-to-last layer (node feature) is used for a novel contrastive loss (see below), and the output of the last layer (classification logits) is used for the cross-entropy loss ($loss_{CE}$). Graph-MLP uses Gelu [8] as an activation function, and to improve training stability, batch normalization is replaced with layer normalization [1]. The dropout [29] layer is introduced to prevent overfitting.

As this simple MLP does not model graphs well, the MLP-structure is combined with the aforementioned contrastive loss, the **N**eighboring **Contrast**ive loss (NContrast). NContrast loss aims to group positive samples (here, the r-hop neighbors) closer to the target node and push negative samples further away from the target node. The NContrast loss for the i-th node is calculated with

$$l_i = -\log \frac{\sum_{j=1}^{B} \mathbf{1}_{[j \neq i]} \gamma_{ij} \exp(\text{sim}(\mathbf{z}_i, \mathbf{z}_j)/\tau)}{\sum_{k=1}^{B} \mathbf{1}_{[k \neq i]} \exp(\text{sim}(\mathbf{z}_i, \mathbf{z}_k)/\tau)} \tag{1}$$

Here, *sim* is the cosine similarity, τ is the temperature parameter, and γ_{ij} indicates the strength of the connection between nodes i and j. Generally, γ_{ij} is non-zero if and only if node j is a r-hop neighbor of node i. γ_{ij} is calculated using the r-th power of the adjacency matrix. If no positive samples exist for a target node, i. e., no r-hop neighbors are in the sample, the NContrast loss would be negative infinity. To prevent this, the loss for this node is ignored.

The total loss for Graph-MLP can be computed as follows:

$$loss_{Graph-MLP} = loss_{CE} + \underbrace{\alpha \frac{1}{B} \sum_{i=1}^{B} l_i}_{loss_{NC}} \tag{2}$$

Due to the simple structure, the training process for Graph-MLP is also simple. The graphical structure, i. e., adjacency matrix, is only needed to compute the loss and not in the feed-forward network during training. For every training iteration, B nodes are randomly sampled, and their node feature matrix and adjacency matrix are aggregated. This sampling is required, as calculating the adjacency matrix for the entire graph requires too much memory when using large datasets. The NContrast and the cross-entropy loss are then calculated for the sample, and the model parameters are updated via back-propagation. Additionally, as adjacency information is not required for training, Graph-MLP can still provide better results than GCNs if any adjacency information is missing.

3.2 Sampling Strategies

In this section, we explore various graph sampling algorithms that can be used to create a representative subset of a large graph. These algorithms can be broadly classified into three categories: node selection, edge selection, and exploration selection.

3.2.1 Node Selection Strategies. Node selection algorithms focus on selecting a subset of nodes from a graph to be included in the sample. When selecting nodes to include in the sample, these algorithms may consider various factors, such as the degree of the nodes, their centrality, or their community membership.

A simple yet widely-used example is *Random Batch Sampling* [10, 14]. Here, N uniformly randomly sampled nodes are selected and their corresponding adjacency information and node features are aggregated.

In contrast, *Random Degree Node* [14] does not sample nodes with a uniform probability. Instead, the higher a node's degree, the higher the probability of being sampled. The authors note that this may lead to very dense samples, as Leskovec and Faloutsos [14] relate a higher degree to a higher sampling probability. In our work, we also try the inverse, i. e., the higher a node's degree, the lower the sampling probability is.

A similar approach is *Rank Degree* presented by Voudigari et al. [31] and extended by Yousuf et al. [36]. *Rank Degree* creates a set S of k nodes by ranking them according to their degree and selecting the k nodes with the highest degree. A probability value p is used to define the top-k nodes in each case. From the existing set S, a random node is selected and its neighbors are ranked as before. The selected node is added to the sub-graph and the newly ranked neighbors serve as the new set S for the following iteration. After running some tests for several values on Cora, we settled for $|S| = 3$, so three nodes are randomly selected initially. For the probability value, we decided on $p = 0.35$, meaning the top 35% of nodes are selected at each iteration.

A different approach is *Negative Sampling* [24]. In *Negative Sampling*, n source-destination pairs that do not share an edge are sampled. Due to this constraint, it is possible that in small or very dense graphs, less than n pairs are sampled because less than n pairwise-unconnected nodes exist in the graph.

3.2.2 Edge Selection Strategies. Edge selection algorithms focus on selecting a subset of edges from a graph to be included in the sample. These algorithms may consider various factors such as the edges' weight, the nodes' connectivity, or the nodes' community membership when selecting edges to include in the sample.

Leskovec and Faloutsos [14] present three edge selection algorithms. First, *Random Edge* [14] uniformly randomly selects edges and adds the connected nodes to the sample. Second, *Random Node Edge* [14] selects a starting node at random. From this node, an outgoing edge is selected, and the connected node is added to the sample. This procedure is then repeated with a new node selected at random. Leskovec and Faloutsos [14] note that this method has less bias towards high-degree nodes than *Random Edge*. Lastly, *Hybrid Edge* [14] combines both *Random Edge* and *Random Node Edge*. A single iteration of *Random Node Edge* is done with a predefined and fixed probability p (set to 0.8 in the paper and our work) and one iteration of *Random Edge* otherwise.

3.2.3 Exploration Selection Strategies. Exploration selection algorithms focus on selecting a subset of nodes and edges to be included in the sample to maximize the new information discovered during the sampling process. These algorithms may use techniques such as random walks to explore the graph and select nodes and edges for inclusion in the sample.

A popular exploration selection strategy is *Random Walk* [14]. In *Random Walk*, a random node is selected, and a 'walk' is started by randomly choosing outgoing edges, adding connected nodes to the sample. With a probability of 0.15 (used by Leskovec and Faloutsos [14] and in our work), the algorithm returns to the first node and restarts the walk. It is possible that the walk gets stuck or cannot find new nodes. To prevent this, it is possible to use an iteration counter to break and select a new start node. However, we do not use this iteration counter. Instead, we only return the nodes that were sampled until the walk gets stuck. An adaptation of this algorithm is *Random Jump* [14]. The functionality is

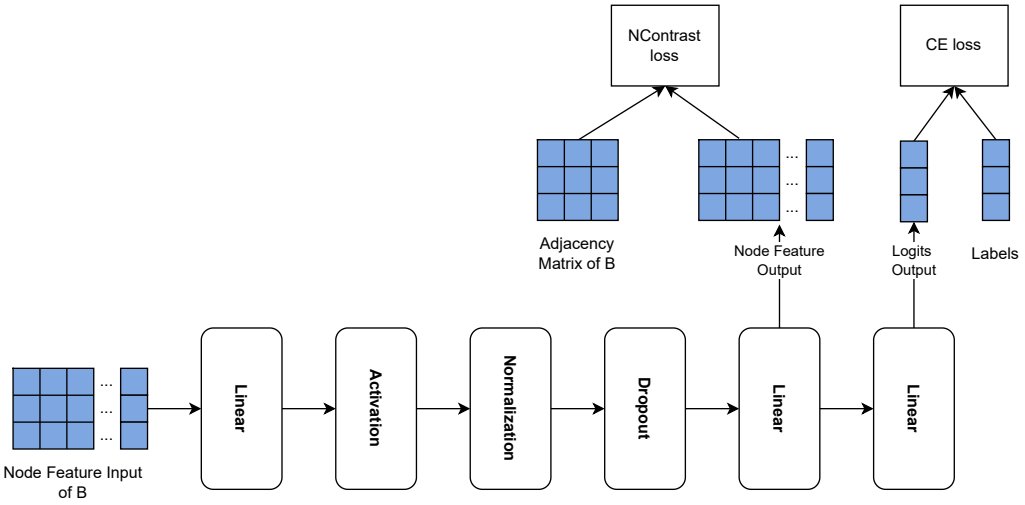

Figure 1: The general structure of Graph-MLP. B refers to the currently sampled batch. Image from Hu et al. [10].

similar to *Random Walk*. However, instead of returning to the first node, a new random node is selected with the set probability (set to 0.15 in our work).

A different approach is sampling based on a node's neighborhood. As a first example, *Random Node Neighbor* [14] selects a node at random and adds it and all its neighbors to the sample. This process is repeated until the desired number of nodes is reached. A similar approach is the *Fixed-size Neighbor Sampling* used in GraphSAGE [7]. Same as before, a node is selected at random, however, only a fixed number k of its neighbors are added to the sample, and for each initially chosen node n many layers are sampled. This process gets recursively repeated for each sampled neighbor n-times. This provides the benefit of a constant computational expense for every batch. After testing several values on Cora, we settled on the value $k = 3$ and $n = 2$ for our experiments.

Furthermore, we implement the more complex sampling strategy *Frontier Sampling*. In *Frontier Sampling* [25, 36], m-dimensional dependent random walks sample the data by keeping track of a list of m nodes. We settled on the value of $m = 100$ for our experiments. Initially, these nodes should be chosen to represent the graph well. We start with random initialization and add these nodes to the sample. The m-dimensional dependent random walks choose a node u from the list where the probability is determined by the node degree. A neighbor v of the chosen node is randomly selected (i. e., a step of a random walk), and u is replaced by v in the list. u is then added to the sample set. Our adapted implementation can be seen in Algorithm 1.

Algorithm 1 Frontier Sampling [25]

1: Initialize $L = (v_1, \ldots, v_m)$ with m randomly chosen nodes
2: Initialize result set r and add all nodes from L
3: **repeat**
4: Select $u \in L$ with probability $deg(u)/\sum_{\forall v \in L} deg(v)$
5: Select an outgoing edge of u, (u,v) uniformly at random
6: Replace u by v in L and add v to r
7: **until** $|r| \geq Batchsize$

We also investigated other sampling strategies in more detail, see Appendix B. However, we did not use them in our experiments due to under-specification and computational limitations. For example, we implemented Snowball Expansion Sampling, but running our implementation on Cora was unfeasible as a single epoch took 30 minutes.

3.3 Summary

Given the challenges of processing graph data with Graph-MLP due to memory limitations and the need to work with batches, sampling strategies must be used to improve the efficiency and performance of the model. There are a variety of sampling strategies with different properties that can be selected depending on the use case. Overall, sampling strategies can play an important role in processing graph data with Graph-MLP and maybe even help to improve model efficiency and performance.

4 EXPERIMENTAL APPARATUS

In the following, we present the setup for our experiments. We begin with a summary of the datasets we use, followed by the preprocessing we use. Next, we describe our experimental procedure. Finally, we present the metrics we use in our evaluation.

4.1 Datasets

We use the common citation datasets *Cora* [22], *CiteSeer* [6] and *Pubmed* [28]. We use three additional datasets to cover other domains and have a more diverse set of graphs. These addtional datasets are *obgn-arxiv* [32], *Reddit2* [38] and *Facebook* [27]. *obgn-arxiv* is a dataset containing computer science arXiv papers. Nodes are arXiv papers, and edges indicate that one paper cites the other. Every node has a 128-dimensional feature vector. Those 128 features are extracted from the title and abstract of the corresponding arXiv paper the node represents. The challenge is to predict the primary categories of the arXiv papers out of the 40 different subject areas. *Reddit2* is a sparser version of the original Reddit dataset [7]. It is a graph of Reddit posts sampled from 41 different communities. Nodes are posts, and two nodes are connected via an edge if the same Reddit user comments on both posts the nodes represent. Each node has a 602-dimensional feature vector. The features are extracted from each post title, the comments on the post, the post score, and the number of comments it has. The task is predicting to which

of the 41 communities a post got posted. The *Facebook* dataset is a graph containing official Facebook sites. Nodes represent verified Facebook pages. Two nodes are connected via an edge if the two pages share mutual likes between them. Every node has a 14,000-dimensional feature vector. The features are each extracted from the respective site descriptions. The objective is to predict a page belonging to one of the four total page categories.

See Table 1 for characteristics of our datasets.

4.2 Preprocessing

For every training run, we load the PyTorch Geometric[1] dataset and convert it into the structure used in Graph-MLP. In particular, this means extracting the adjacency matrix, node labels, features, and the train/val/test split as an array containing the respective node ids. The adjacency matrix is then normalized using the *renormalization trick* presented by Kipf and Welling [12]. Additionally, the r-th power of the adjacency matrix is computed to utilize the r-hop neighborhoods. Since multiple sampling techniques use the node degree, we also compute it in advance for all nodes.

4.3 Procedure

After preprocessing, our procedure is the same for every dataset and sampling strategy. In total, we apply Graph-MLP to 6 datasets and 13 sampling strategies. Our datasets are described in Section 4.1, and our sampling strategies are found in Section 3.2. Every run consists of 400 epochs, and every run is repeated five times for Cora, CiteSeer, Pubmed, and Facebook, and three times for ogbn-arxiv and Reddit2. Due to computational limitations, we did not run every sampling strategy on ogbn-arxiv and Reddit2. In particular, we use Random Batch Sampling, Random Degree Node (higher and lower), Random Edge, and Random Node Neighbor for ogbn-arxiv and Reddit2. Additionally, we run Graph-MLP on ogbn-arxiv using Random Node Edge sampling. For every run, we measure the runtime, cross-entropy, and NContrast loss, as well as the test accuracy in our evaluation. For more information on our measures, see Section 4.4.

It should be noted that we train ogbn-arxiv and Reddit2 in an inductive setting (i. e., we only sample nodes from the train split) instead of the usual transductive setting in Graph-MLP.

4.4 Measures and Metrics

As metrics for our datasets, we consider homophily and the clustering coefficient.

In general, homophily describes the tendency of similar nodes (i. e., nodes with the same label) to be connected. We focus on edge-insensitive homophily and do not regard node homophily [23].

Edge-insensitive homophily [16] (Equation 4) normalizes the edge homophily measure across the classes in the dataset. Edge homophily [39], calculates the ratio of edges that connect two nodes with the same label to all edges, see Equation 3.

$$\frac{(u,v) \in E : y_v = y_u}{|E|} \tag{3}$$

In Equation 4, C refers to the number of classes, C_k to the number of nodes of class k. h_k refers to the edge homophily (Equation 3) only regarding nodes of class k.

$$\frac{1}{C-1} \sum_{k=1}^{C} \max\left(0, h_k - \frac{|C_k|}{|V|}\right) \tag{4}$$

[1] https://pytorch-geometric.readthedocs.io/en/latest/modules/datasets.html

Edge-insensitive homophily is our main focus, as the datasets (see Table 1) are heavily imbalanced regarding the nodes to class ratio. The normalization across the number of classes weakens this imbalance.

The clustering coefficient measures the degree to which nodes in a graph tend to cluster together. It is defined as the fraction of pairs of neighbors of a node that are also neighbors of each other.

Formally, let $G = (V, E)$ be a graph, where V is the set of nodes, and E is the set of edges. The clustering coefficient of a node $v \in V$, denoted $c(v)$, is given by:

$$c(v) = \frac{2 \cdot e_v}{d_v(d_v - 1)} \tag{5}$$

where $e_v \in E$ is the number of edges between the neighbors of v, and d_v is the degree of v (i. e., the number of neighbors of v) [36]. The clustering coefficient can be computed for each node in a graph, and the average of these values gives the global clustering coefficient of the graph.

We measure accuracy during every epoch on our train split as well as the test and validation split. Also, we plot the cross entropy and neighboring contrastive loss for the train split. Here we plot the losses separately and also in sum. On the test split, we only plot the cross entropy loss. The specific train/val/test splits for every dataset are in Table 2.

5 RESULTS

The outcomes of our experiments are presented in this section. In Section 5.1, we present the classification task results. In Section 5.2, we show the learning curves for the various datasets.

5.1 Accuracy and Runtime

Table 3 presents the results of our experiments, where we compare the average accuracy of different sampling strategies for node classification.

Random Batch Sampling achieved the highest accuracy on ogbn-arxiv with 52.26% and performed second best on Reddit2, Pubmed, and Facebook with an accuracy of 59.26%, 80.12%, and 62.07%, respectively. Although Random Batch Sampling is not part of the top three on Cora and CiteSeer, it still performed similarly to the best strategies with an accuracy difference of less than 1%.

Random Node Edge achieved the highest accuracies on Cora (80.22%), Pubmed (80.32%), and Facebook (62.73%). Random Degree Node (lower) achieved the highest accuracy on CiteSeer with 72.8% and generally performed well, reaching the top three on four of the six applied datasets. On Reddit2, an accuracy of 59.29% was reached, the same as Random Batch Sampling, therefore sharing the second place, only outperformed by Random Degree Node (higher) with an accuracy of 61.68%. On every other dataset Random Degree Node (higher) was consistently one of the three worst performing strategies and the worst on CiteSeer, Facebook, and ogbn-arxiv, reaching accuracies of 70.24%, 57.14%, and 47.98% respectively.

Negative Sampling was the second best strategy for Cora and ogbn-arxiv in terms of accuracy, while Random Jump achieved good results on Cora and CiteSeer. Rank Degree, Hybrid Edge, and Fixed-size Neighbor Sampling performed close to average on all datasets. At the same time, Frontier Sampling, Random Node Neighbor, and Random Edge generally performed worse than average. Our implementation of Random Walk was the worst

	Cora	CiteSeer	Pubmed	Facebook	ogbn-arxiv	Reddit2
Nodes	2,708	3,327	19,717	22,470	169,343	232,965
Edges	5,429	4,732	44,338	171,002	1,166,243	23,213,838
Edges/Nodes	2.00	1.42	2.25	7.61	6.89	99.65
Classes	7	6	3	4	40	41
Features	1,433	3,703	500	14,000	128	602
Edge-Insensitive Homophily in % [16]	76.57	62.67	66.41	81.98	44.45	69.14
Average Clustering Coefficient [36]	0.24	0.06	0.14	0.36	0.12	0.17

Table 1: General information for the datasets we use. Datasets used in an inductive training setting are underlined.

	Cora	CiteSeer	Pubmed	Facebook	ogbn-arxiv	Reddit2
train split	140	120	60	80	90,941	153,932
validation split	500	500	500	500	29,799	23,699
test split	1,000	1,000	1,000	21,890	48,603	55,334

Table 2: The train/val/test split we use during training for each dataset. Datasets used in an inductive training setting are underlined.

	Cora	CiteSeer	Pubmed	Facebook	ogbn-arxiv	Reddit2
Random Batch Sampling	79.56 ± 0.84	72.06 ± 0.40	80.12 ± 0.83	62.07 ± 0.30	**52.26** ± 0.12	59.29 ± 0.26
Random Degree Node (higher)	77.90 ± 0.52	70.24 ± 0.96	78.14 ± 0.26	57.14 ± 2.06	47.98 ± 0.03	**61.68** ± 0.53
Random Degree Node (lower)	79.32 ± 1.11	**72.80** ± 0.99	80.10 ± 0.79	58.19 ± 3.04	52.07 ± 0.24	59.29 ± 0.39
Rank Degree	79.32 ± 0.32	72.52 ± 0.98	77.62 ± 0.68	59.01 ± 3.20	-	-
Negative Sampling	79.94 ± 1.25	72.26 ± 0.45	79.48 ± 0.28	60.86 ± 2.19	52.24 ± 0.11	-
Random Edge	78.16 ± 1.12	72.16 ± 0.90	79.18 ± 0.41	58.46 ± 1.86	51.86 ± 0.12	58.60 ± 0.50
Random Node Edge	**80.22** ± 0.99	72.30 ± 0.60	**80.32** ± 0.50	**62.73** ± 2.80	-	-
Hybrid Edge	78.28 ± 1.15	72.26 ± 0.82	78.78 ± 0.44	61.85 ± 2.65	-	-
Random Walk	75.02 ± 0.62	71.24 ± 1.94	75.42 ± 2.34	59.27 ± 3.56	-	-
Random Jump	79.86 ± 1.05	72.72 ± 0.33	79.48 ± 0.53	61.13 ± 3.35	-	-
Random Node Neighbor	78.96 ± 0.34	71.56 ± 0.68	79.30 ± 0.65	57.71 ± 0.60	50.74 ± 0.32	54.62 ± 0.36
Fixed-size Neighbor Sampling	79.22 ± 0.77	71.66 ± 1.03	79.34 ± 0.40	60.86 ± 2.69	-	-
Frontier Sampling	78.20 ± 0.64	71.12 ± 0.37	79.08 ± 0.62	59.75 ± 0.86	-	-

Table 3: Test accuracy (in %) of node classification in Graph-MLP with different sampling strategies. The best accuracy for each dataset is shown in bold. Datasets used in an inductive training setting are underlined. Transductive datasets are averaged over five runs, while inductive datasets are averaged over three runs.

	Cora	CiteSeer	Pubmed	Facebook	ogbn-arxiv	Reddit2
Random Batch Sampling	12.2 ± 0.45	13.0 ± 0.71	22.4 ± 1.14	14.8 ± 0.84	33307.0 ± 114.82	95852.00 ± 3075.82
Random Degree Node (higher)	12.8 ± 0.45	12.6 ± 0.89	21.8 ± 0.45	16.2 ± 0.84	34281.67 ± 2669.53	97688.67 ± 2162.66
Random Degree Node (lower)	12.6 ± 1.34	14.6 ± 0.55	23.4 ± 0.55	18.4 ± 0.55	40372.67 ± 1485.21	91896.67 ± 1475.63
Rank Degree	558.4 ± 11.06	537.6 ± 1.82	295.2 ± 1.48	166.8 ± 7.46	-	-
Negative Sampling	12.8 ± 2.17	12.6 ± 0.89	16.6 ± 1.95	22.6 ± 0.89	53717.33 ± 2216.18	-
Random Edge	10.4 ± 0.55	11.0 ± 0.71	13.6 ± 0.89	18.8 ± 7.05	32818.00 ± 407.18	98982.33 ± 763.02
Random Node Edge	64.8 ± 0.45	64.6 ± 1.14	94.2 ± 0.45	68.2 ± 1.30	-	-
Hybrid Edge	65.0 ± 0.71	64.4 ± 1.34	96.2 ± 0.84	70.2 ± 1.64	-	-
Random Walk	146.8 ± 1.48	140.4 ± 0.89	227.8 ± 2.05	160.8 ± 0.84	-	-
Random Jump	696.0 ± 7.35	665.0 ± 4.18	564.0 ± 5.10	326.8 ± 2.49	-	-
Random Node Neighbor	36.4 ± 0.55	42.8 ± 0.45	55.6 ± 0.55	22.8 ± 0.45	36845.33 ± 1854.01	96408.00 ± 3715.10
Fixed-size Neighbor Sampling	59.2 ± 1.30	60.8 ± 0.45	95.8 ± 0.45	64.2 ± 1.64	-	-
Frontier Sampling	253.0 ± 2.55	254.8 ± 1.10	372.2 ± 3.03	259.8 ± 3.63	-	-

Table 4: Average runtime in seconds of node classification in Graph-MLP with different sampling strategies. Datasets used in an inductive training setting are underlined. Transductive datasets are averaged over five runs, while inductive datasets are averaged over three runs.

performing strategy on Cora and Pubmed and also did poorly in comparison on most other datasets. Moreover, except for Cora, it produced the highest standard deviation on all datasets it was tested on.

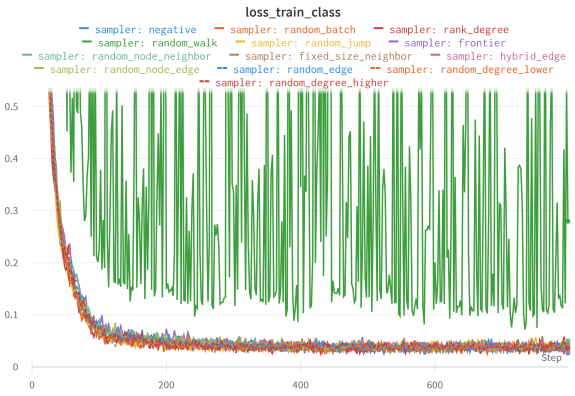

Figure 2: Cross-Entopy Loss on Cora

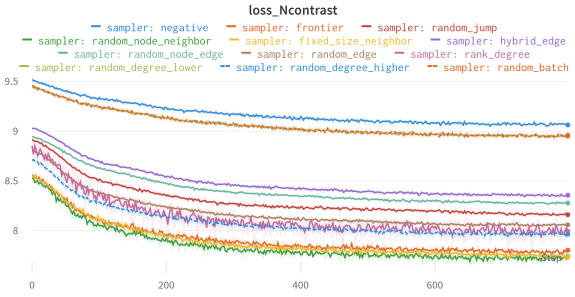

Figure 3: NContrast Loss on Pubmed

Besides accuracy, we also tracked the average runtime of each sampling method (Table 4) and found that overall Random Edge, Random Batch Sampling, Random Degree Node (higher and lower), Negative Sampling, and Random Node Neighbor were the fastest methods, therefore allowing testing on ogbn-arxiv and Reddit2. Note that Negative Sampling could only be tested on ogbn-arxiv because of not scaling well enough. Random Edge was the fastest on most datasets, while Random Batch Sampling was, on average, the fastest on Facebook and Random Degree Node (lower) on Reddit2. Our implementation of Random Jump performed the worst in terms of runtime on all datasets.

5.2 Impact on the Learning Curve

Choosing the right sampling strategy is critical to building accurate models, but it is equally important to evaluate the model's performance as it gains knowledge from the data. Learning curves, which show how the model's performance evolves during training, can be used to assess the model's learning progress.

The experimental results show some interesting insights into the influence of different graph sampling strategies on the learning curve. Although there is no significant difference in the cross-entropy loss between the various sampling strategies, we found that the Neighboring Contrastive Loss depends on the sampling strategy.

The loss values obtained from Random Walk sampling were found to be highly volatile and have little helpful information for the following analysis. So, we did not include the outcomes of Random Walk sampling in the following figures and analysis. Figure 2 demonstrates this effect on the cross-entropy learning curve for Cora.

Our experiments showed that, except on Pubmed, all sampling strategies converged equally fast but to different values. According to our analysis, compared to other sampling strategies on Pubmed, Negative Sampling, Random Batch, and Random Degree (lower) showed slower convergence and produced higher NContrast loss values, see Figure 3.

Among the smaller datasets, Fixed-size Neighbor achieved the lowest NContrast loss on Cora and CiteSeer, and also performed well on Pubmed. For larger datasets, Random Node Neighbor emerged as the best sampling strategy on Pubmed and Facebook and ranked first on Reddit2 while also performing strongly on other large datasets. On ogbn-arxiv, Random Degree (higher) achieved the best loss.

The learning curves of the different sampling strategies for the NContrast loss are provided in Appendix C, while a complete view of all learning curves can be found in our Learning-Curves-Report[2].

6 DISCUSSION

Although no single sampling strategy performs best overall, our research still provides interesting insights. The first observation is that Random Batch Sampling performs well on most datasets and seems less susceptible to the worsening effects of lower homophily.

Random Degree Node (lower) sampling also performs well on most datasets but worse on Facebook. A probable cause for this is the high average clustering coefficient and homophily of the Facebook graph. This sampling strategy prioritizes nodes with a lower degree which may help to capture the behavior of less connected nodes, especially in graphs with less homophily. But in graphs with high homophily, the information from connected nodes with higher degrees may also be important. Interestingly though, Random Degree Node (higher), the strategy with the opposite selection preference, prioritizing nodes with a higher degree, also does poorly on the Facebook graph, suggesting that high and low degree nodes are essential for the Graph-MLP model to achieve higher accuracy.

It should be noted that our implementation of Random Walk, which does not select a new start node once stuck, performs poorly. This performance contrasts the presented research in Section 2 but is most likely due to this change.

Another interesting observation is the poor performance of Random Degree Node (higher) on most datasets but the best performance on Reddit2. One possible explanation is the degree distribution of nodes in Reddit2 differs from the other datasets. The nodes may have a broader range of degrees and a more significant number of nodes with high degrees compared to other datasets. As a result, sampling nodes with higher degrees may be more effective in capturing important information. It is also possible that the graph structure includes more hub-like nodes that play a crucial role compared to other datasets.

Overall, we find that many factors, such as the homophily of the graph, the average clustering coefficient, or the degree distribution of nodes, can influence the performance of individual sampling strategies. From this, understanding the graph's specific properties can help select an appropriate sampling strategy for the model to rely on.

[2]https://api.wandb.ai/links/graph-mlp-sampling/eqam7jrc

Similar findings can also be seen in the learning curve of the NContrast loss. Again, the results are highly dependent on the dataset, so there is no clear best sampling strategy. However, different sampling strategies converge to different low values. A lower value means that the model is better at solving the problem posed by the loss function on the sampled graphs. In the case of the NContrast loss, this means a good representation of the neighborhood. The effects of choosing an appropriate sampling strategy are particularly striking in the case of Pubmed. Since the convergence behavior is worse for some samplers, this suggests that for some datasets, the choice of a suitable sampler even impacts the training time.

In summary, learning curves provide valuable insight into the behavior of machine learning models and can help select appropriate sampling strategies.

6.1 Key Results

While the differences in accuracy between many sampling algorithms are often small, the choice of sampling strategy can impact the model's performance. Our experiments show that the performance of the different sampling methods varies across datasets, indicating that there is no single method that is optimal for all datasets. Thus, we recommend treating the used sampler as a hyperparameter that should be optimized for the task and dataset.

Our results show that the selection of the graph sampling method plays a significant role in achieving higher accuracies in the Graph-MLP model. Although Random Batch Sampling generally performs well across many datasets, it is outperformed by other sampling methods in terms of accuracy. Nevertheless, Random Batch Sampling remains a solid option for a large variety of datasets, especially when runtime is a concern.

Moving on to the learning curve, another crucial component of machine learning. Our experiments also show that different sampling techniques with a model using the Neighboring Contrastive Loss function also affects the model's learning curve, which in turn is influenced by factors such as dataset complexity, graph size, and hyperparameter choice.

6.2 Threat to Validity and Limitations

We acknowledge that we have a strict upper limit concerning dataset size. The largest dataset is Reddit2, with 232k nodes. However, this is a general limitation to Graph-MLP as the adjacency matrix needs to be computed and loaded into memory during computation. For example, the adjacency matrix for ogbn-arxiv is 100GB large and does not fit on a conventional GPU anymore.

Furthermore, some of the selected sampling strategies are inefficient and were too slow to run on ogbn-arxiv and Reddit2, although there are possible optimizations. Nonetheless, these inefficiencies show an apparent weakness in the sampling strategy, as one has to make a trade-off between higher training and inference time and a potential increase in accuracy.

Also, we average the accuracy on three runs instead of 5 runs on the large datasets (ogbn-arxiv and Reddit2). This is a generally accepted number, as the standard deviation is generally lower for large datasets.

Additionally, four of six of our datasets are citation networks providing little variety. But even though the majority of our used datasets are citation networks, we also cover social media networks (Facebook and Reddit), and the characteristics of all datasets vary widely (see Table 1).

6.3 Generalization

As we have an extensive range of datasets and samplers, we believe the results can be generalized well. Sampling strategies that perform well across several datasets will also perform well on new datasets, and sampling strategies with a mixed performance will continue to perform sub-par.

On the other hand, our results of what sampling strategies work best with Graph-MLP are not necessarily transferable to other models using sampling. As Graph-MLP heavily relies on neighboring nodes existing in a sample and other models might not, the sampling strategies that sample neighbors well may decrease performance on other models.

6.4 Future Work and Impact

In future work, one can explore how the sampling strategies that improve Graph-MLP impact other models using sampling, e. g., GraphSAGE [7] or GraphSAINT [38]. Furthermore, comparing the results of models using full message passing (e. g., GCN or GAT) to the results produced by Graph-MLP is an exciting avenue of exploration. Full message passing models could potentially be extended to use sampling to explore if sampling benefits these models.

Graph-MLP is a highly cited paper and widely used model. It is, therefore, desirable that the performance of Graph-MLP is improved and optimized further. In this paper, we focus on the sampling strategies to sample a sub-graph during Graph-MLP training.

7 CONCLUSION

In this paper, we applied different graph sampling algorithms to the Graph-MLP model and used six benchmark datasets to determine accuracy and runtime for node prediction. Overall, we find that the performance of individual sampling strategies varies, and the optimal sampler to use depends on priorities, the specific context of an application, and the dataset's characteristics. While our research only focused on the Graph-MLP model, the choice of sampling strategy could also affect the performance of other models, methods, and domains of application where graph sampling plays a role. Therefore, the choice of sampler should be considered as a hyperparameter that can be optimized for the specific needs of the task at hand.

For repeatability and further insights, we provide our source code: https://github.com/Tobi2K/Graph-MLP-Sampling

ACKNOWLEDGMENTS

The authors acknowledge support by the state of Baden-Württemberg through bwHPC. The presented research results from the module "Graph Analytics and Deep Learning" taught at the University of Ulm in 2022/23 We also thank Nicolas Lell for providing the hyperparameters optimized on ogbn-arxiv. We use these hyperparameters on both ogbn-arxiv and Reddit2.

REFERENCES

[1] Lei Jimmy Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. 2016. Layer Normalization. *CoRR* abs/1607.06450 (2016). arXiv:1607.06450 http://arxiv.org/abs/1607.06450

[2] Sergey Brin and Lawrence Page. 1998. The Anatomy of a Large-Scale Hypertextual Web Search Engine. *Comput. Networks* 30, 1-7 (1998), 107–117. https://doi.org/10.1016/S0169-7552(98)00110-X

[3] Jie Chen, Tengfei Ma, and Cao Xiao. 2018. FastGCN: Fast Learning with Graph Convolutional Networks via Importance Sampling. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net. https://openreview.net/forum?id=rytstxWAW

[4] Ming Chen, Zhewei Wei, Bolin Ding, Yaliang Li, Ye Yuan, Xiaoyong Du, and Ji-Rong Wen. 2020. Scalable Graph Neural Networks via Bidirectional Propagation. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (Eds.). https://proceedings.neurips.cc/paper/2020/hash/a7789ef88d599b8df86bbee632b2994d-Abstract.html

[5] Chenchen Feng, Yu He, Shiyang Wen, Guojun Liu, Liang Wang, Jian Xu, and Bo Zheng. 2022. DC-GNN: Decoupled Graph Neural Networks for Improving and Accelerating Large-Scale E-commerce Retrieval. In *Companion of The Web Conference 2022, Virtual Event / Lyon, France, April 25 - 29, 2022*, Frédérique Laforest, Raphaël Troncy, Elena Simperl, Deepak Agarwal, Aristides Gionis, Ivan Herman, and Lionel Médini (Eds.). ACM, 32–40. https://doi.org/10.1145/3487553.3524203

[6] C. Lee Giles, Kurt D. Bollacker, and Steve Lawrence. 1998. CiteSeer: An Automatic Citation Indexing System. In *Proceedings of the 3rd ACM International Conference on Digital Libraries, June 23-26, 1998, Pittsburgh, PA, USA*. ACM, 89–98. https://doi.org/10.1145/276675.276685

[7] William L. Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (Eds.). 1024–1034. https://proceedings.neurips.cc/paper/2017/hash/5dd9db5e033da9c6fb5ba83c7a7ebea9-Abstract.html

[8] Dan Hendrycks and Kevin Gimpel. 2016. Bridging Nonlinearities and Stochastic Regularizers with Gaussian Error Linear Units. *CoRR* abs/1606.08415 (2016). arXiv:1606.08415 http://arxiv.org/abs/1606.08415

[9] Pili Hu and Wing Cheong Lau. 2013. A Survey and Taxonomy of Graph Sampling. *CoRR* abs/1308.5865 (2013). arXiv:1308.5865 http://arxiv.org/abs/1308.5865

[10] Yang Hu, Haoxuan You, Zhecan Wang, Zhicheng Wang, Erjin Zhou, and Yue Gao. 2021. Graph-MLP: Node Classification without Message Passing in Graph. *CoRR* abs/2106.04051 (2021). arXiv:2106.04051 https://arxiv.org/abs/2106.04051

[11] Kenta Iwasaki and Kazuyuki Shudo. 2018. Comparing Graph Sampling Methods Based on the Number of Queries. In *IEEE International Conference on Parallel & Distributed Processing with Applications, Ubiquitous Computing & Communications, Big Data & Cloud Computing, Social Computing & Networking, Sustainable Computing & Communications, ISPA/IUCC/BDCloud/SocialCom/SustainCom 2018, Melbourne, Australia, December 11-13, 2018*, Jinjun Chen and Laurence T. Yang (Eds.). IEEE, 1136–1143. https://doi.org/10.1109/BDCloud.2018.00168

[12] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net. https://openreview.net/forum?id=SJU4ayYgl

[13] Pawel Ladosz, Lilian Weng, Minwoo Kim, and Hyondong Oh. 2022. Exploration in deep reinforcement learning: A survey. *Information Fusion* 85 (2022), 1–22. https://doi.org/10.1016/j.inffus.2022.03.003

[14] Jure Leskovec and Christos Faloutsos. 2006. Sampling from large graphs. In *Proceedings of the Twelfth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Philadelphia, PA, USA, August 20-23, 2006*, Tina Eliassi-Rad, Lyle H. Ungar, Mark Craven, and Dimitrios Gunopulos (Eds.). ACM, 631–636. https://doi.org/10.1145/1150402.1150479

[15] Jure Leskovec, Jon M. Kleinberg, and Christos Faloutsos. 2005. Graphs over time: densification laws, shrinking diameters and possible explanations. In *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Chicago, Illinois, USA, August 21-24, 2005*, Robert Grossman, Roberto J. Bayardo, and Kristin P. Bennett (Eds.). ACM, 177–187. https://doi.org/10.1145/1081870.1081893

[16] Derek Lim, Felix Hohne, Xiuyu Li, Sijia Linda Huang, Vaishnavi Gupta, Omkar Bhalerao, and Ser-Nam Lim. 2021. Large Scale Learning on Non-Homophilous Graphs: New Benchmarks and Strong Simple Methods. In *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan (Eds.). 20887–20902. https://proceedings.neurips.cc/paper/2021/hash/ae816a80e4c1c56caa2eb4e1819cbb2f-Abstract.html

[17] Jielun Liu, Ghim Ping Ong, and Xiqun Chen. 2022. GraphSAGE-Based Traffic Speed Forecasting for Segment Network With Sparse Data. *IEEE Trans. Intell. Transp. Syst.* 23, 3 (2022), 1755–1766. https://doi.org/10.1109/TITS.2020.3026025

[18] Xin Liu, Mingyu Yan, Lei Deng, Guoqi Li, Xiaochun Ye, and Dongrui Fan. 2022. Sampling Methods for Efficient Training of Graph Convolutional Networks: A Survey. *IEEE CAA J. Autom. Sinica* 9, 2 (2022), 205–234. https://doi.org/10.1109/JAS.2021.1004311

[19] Wai Weng Lo, Siamak Layeghy, Mohanad Sarhan, Marcus Gallagher, and Marius Portmann. 2022. E-GraphSAGE: A Graph Neural Network based Intrusion Detection System for IoT. In *2022 IEEE/IFIP Network Operations and Management Symposium, NOMS 2022, Budapest, Hungary, April 25-29, 2022*. IEEE, 1–9. https://doi.org/10.1109/NOMS54207.2022.9789878

[20] Zihan Luo, Jianxun Lian, Hong Huang, Hai Jin, and Xing Xie. 2022. AdaGNN: Adapting to Local Patterns for Improving Graph Neural Networks. In *WSDM '22: The Fifteenth ACM International Conference on Web Search and Data Mining, Virtual Event / Tempe, AZ, USA, February 21 - 25, 2022*, K. Selcuk Candan, Huan Liu, Leman Akoglu, Xin Luna Dong, and Jiliang Tang (Eds.). ACM, 638–647. https://doi.org/10.1145/3488560.3498460

[21] Arun S. Maiya and Tanya Y. Berger-Wolf. 2010. Sampling community structure. In *Proceedings of the 19th International Conference on World Wide Web, WWW 2010, Raleigh, North Carolina, USA, April 26-30, 2010*, Michael Rappa, Paul Jones, Juliana Freire, and Soumen Chakrabarti (Eds.). ACM, 701–710. https://doi.org/10.1145/1772690.1772762

[22] Andrew McCallum, Kamal Nigam, Jason Rennie, and Kristie Seymore. 2000. Automating the Construction of Internet Portals with Machine Learning. *Inf. Retr.* 3, 2 (2000), 127–163. https://doi.org/10.1023/A:1009953814988

[23] Hongbin Pei, Bingzhe Wei, Kevin Chen-Chuan Chang, Yu Lei, and Bo Yang. 2020. Geom-GCN: Geometric Graph Convolutional Networks. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net. https://openreview.net/forum?id=S1e2agrFvS

[24] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2009. BPR: Bayesian Personalized Ranking from Implicit Feedback. In *UAI 2009, Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence, Montreal, QC, Canada, June 18-21, 2009*, Jeff A. Bilmes and Andrew Y. Ng (Eds.). AUAI Press, 452–461. https://www.auai.org/uai2009/papers/UAI2009_0139_48141db02b9f0b02bc7158819ebfa2c7.pdf

[25] Bruno F. Ribeiro and Donald F. Towsley. 2010. Estimating and sampling graphs with multidimensional random walks. In *Proceedings of the 10th ACM SIGCOMM Internet Measurement Conference, IMC 2010, Melbourne, Australia - November 1-3, 2010*, Mark Allman (Ed.). ACM, 390–403. https://doi.org/10.1145/1879141.1879192

[26] Emanuele Rossi, Fabrizio Frasca, Ben Chamberlain, Davide Eynard, Michael M. Bronstein, and Federico Monti. 2020. SIGN: Scalable Inception Graph Neural Networks. *CoRR* abs/2004.11198 (2020). arXiv:2004.11198 https://arxiv.org/abs/2004.11198

[27] Benedek Rozemberczki, Carl Allen, and Rik Sarkar. 2021. Multi-Scale attributed node embedding. *J. Complex Networks* 9, 2 (2021). https://doi.org/10.1093/comnet/cnab014

[28] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Gallagher, and Tina Eliassi-Rad. 2008. Collective Classification in Network Data. *AI Mag.* 29, 3 (2008), 93–106. https://doi.org/10.1609/aimag.v29i3.2157

[29] Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.* 15, 1 (2014), 1929–1958. https://doi.org/10.5555/2627435.2670313

[30] Tan Tang, Sufei Wang, Yunfeng Li, Bohan Li, and Yingcai Wu. 2017. UNMAT: Visual comparison and exploration of uncertainty in large graph sampling. *J. Vis. Lang. Comput.* 41 (2017), 71–78. https://doi.org/10.1016/j.jvlc.2017.05.006

[31] Elli Voudigari, Nikos Salamanos, Theodore Papageorgiou, and Emmanuel J Yannakoudakis. 2016. Rank degree: An efficient algorithm for graph sampling. In *2016 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*. IEEE, 120–129.

[32] Kuansan Wang, Zhihong Shen, Chiyuan Huang, Chieh-Han Wu, Yuxiao Dong, and Anshul Kanakia. 2020. Microsoft Academic Graph: When experts are not enough. *Quant. Sci. Stud.* 1, 1 (2020), 396–413. https://doi.org/10.1162/qss_a_00021

[33] Qiang Wei and Guangmin Hu. 2022. Evaluating graph neural networks under graph sampling scenarios. *PeerJ Comput. Sci.* 8 (2022), e901. https://doi.org/10.7717/peerj-cs.901

[34] Yanhong Wu, Nan Cao, Daniel Archambault, Qiaomu Shen, Huamin Qu, and Weiwei Cui. 2017. Evaluation of Graph Sampling: A Visualization Perspective. *IEEE Trans. Vis. Comput. Graph.* 23, 1 (2017), 401–410. https://doi.org/10.1109/TVCG.2016.2598867

[35] Lizhong Xiao, Xinke Wu, and Guangzhong Wang. 2019. Social Network Analysis Based on Graph. In *12th International Symposium on Computational Intelligence and Design, ISCID 2019, Hangzhou, China, December 14-15, 2019, Volume 2*. IEEE, 196–199. https://doi.org/10.1109/ISCID.2019.10128

[36] Muhammad Irfan Yousuf, Izza Anwer, and Raheel Anwar. 2021. Empirical Characterization of Graph Sampling Algorithms. *CoRR* abs/2102.07980 (2021). arXiv:2102.07980 https://arxiv.org/abs/2102.07980

[37] Muhammad Irfan Yousuf and Suhyun Kim. 2018. List sampling for large graphs. *Intell. Data Anal.* 22, 2 (2018), 261–295. https://doi.org/10.3233/IDA-163319

[38] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor K. Prasanna. 2020. GraphSAINT: Graph Sampling Based Inductive Learning Method. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net. https://openreview.net/forum?id=BJe8pkHFwS

[39] Jiong Zhu, Yujun Yan, Lingxiao Zhao, Mark Heimann, Leman Akoglu, and Danai Koutra. 2020. Beyond Homophily in Graph Neural Networks: Current Limitations and Effective Designs. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (Eds.). https://proceedings.neurips.cc/paper/2020/hash/58ae23d878a47004366189884c2f8440-Abstract.html

[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 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett (Eds.). 11247–11256. https://proceedings.neurips.cc/paper/2019/hash/91ba4a4478a66bee9812b0804b6f9d1b-Abstract.html

Supplementary Materials

A HYPERPARAMETER OPTIMIZATION

In the following, we describe the hyperparameter optimization we applied in this paper. Following Hu et al. [10], the number of epochs is set to 400 for every dataset. The dropout rate is set to 0.6 and the number of hidden layers to 256. We run the same optimization for the remaining hyperparameters as the original Graph-MLP paper using Random Batch Sampling as the default sampler. We use a grid search over a list of all values seen below:

- impact of NContrast loss α: [0, 1, 10, 100]
- batch size: [2, 000, 3, 000]
- learning rate: [0.001, 0.01, 0.05, 0.1]
- order of the adjacency matrix r: [2, 3, 4]
- temparature parameter τ: [0.5, 1, 2]
- weight decay: [0.0005, 0.005]

We set the hyperparameters of our experiments to the best run for each dataset found by our grid search.

Due to computational limitations, we do not optimize the hyperparameters for ogbn-arxiv and Reddit2. Here we use the hyperparameters provided by Nicolas Lell tuned on ogbn-arxiv. We copy the ogbn-arxiv hyperparameters for Reddit 2 and only decrease the order r to reduce computation time further. For the complete list of parameters used for every dataset, see Table 5.

B ADDITIONAL SAMPLING STRATEGIES

We investigate four other sampling strategies we do not evaluate on Graph-MLP. These can be added in future work. We present a short description of these sampling strategies.

(Snowball) Expansion Sampling [21, 36]. This sampling strategy tries to select nodes from every community in a graph. By maximizing the expansion factor, the algorithm aims to construct the sample greedily. The equation for the expansion factor is seen in Equation 6.

$$\textit{Expansion factor} = |N(\{v\}) - (N(S) \cup S)| \tag{6}$$

S denotes the constructed sample, $N(S)$ the current neighborhood set and v the selected node [21]. See the complete implementation in Algorithm 2.

Algorithm 2 (Snowball) Expansion Sampling (XS) [21]

$S = \emptyset$
v = random chosen Node
$S = S \cup \{v\}$
while $\|S\| \leq$ sample size **do**
    select a new node $v \in N(S)$ based on maximization of:
    $|N(\{v\}) - (N(S) \cup S)|$
    $S = S \cup \{v\}$

Random PageRank Node [14]. A non-uniform sampling strategy where the probability of a node being sampled is relative to its PageRank weight. PageRank is a centrality measure focusing on a node's incoming edges [2]. A node with many incoming and high-quality edges is considered more important (i.e., higher weight) if such a measure exists. The assumption is that nodes connect to important nodes more often than others.

List Sampling (LS) [36, 37]. Same as Rank Degree, but in contrast, we maintain a list of candidate nodes composed of all the neighbors of nodes that have been found but have yet to be sampled. Experiments show that this sampling method retains a better balance between the depth and breadth of the sampled sub-graph.

Forest Fire [14, 15]. Select a start node and start adding (burning) edges with corresponding nodes. When an edge and its corresponding other node are burned, this node can also add its edges. See Leskovec et al. [15] for more information.

C NEIGHBORING CONTRASTIVE LEARNING CURVES

This section contains the plots of the NContrast loss obtained from different graph sampling strategies on our datasets. These figures (figs. 4 to 8) provide a visualization of the learning curves and comparative performance of the different sampling strategies. A complete view of all learning curves can be found in our Learning-Curves-Report[3].

[3] https://api.wandb.ai/links/graph-mlp-sampling/eqam7jrc

	Cora	CiteSeer	Pubmed	Facebook	*ogbn-arxiv*	*Reddit2*
epochs	400	400	400	400	400	400
hidden units	256	256	256	256	2048	2048
dropout rate	0.6	0.6	0.6	0.6	0.15	0.15
learning rate	0.001	0.01	0.001	0.001	0.001	0.001
weight decay	0.005	0.005	0.005	0.0005	0	0
alpha α	100	1	1	1	30	30
batch size $\|B\|$	2000	2000	3000	2000	7000	9000
order r	3	2	3	4	3	2
temperature τ	2	1	2	0.5	15	15

Table 5: Hyperparameters we use during training for each dataset. Datasets used in an inductive training setting are denoted in *italic*.

Figure 7: NContrast Loss on ogbn-arxiv

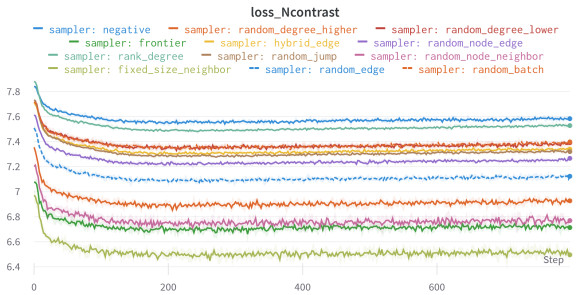

Figure 5: NContrast Loss on CiteSeer

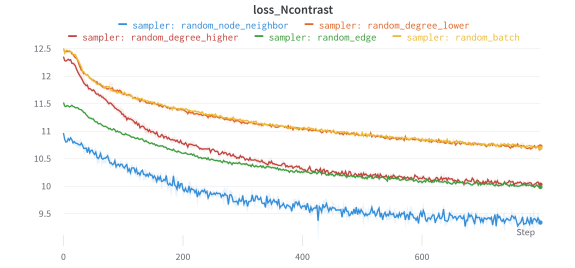

Figure 8: NContrast Loss on Reddit2

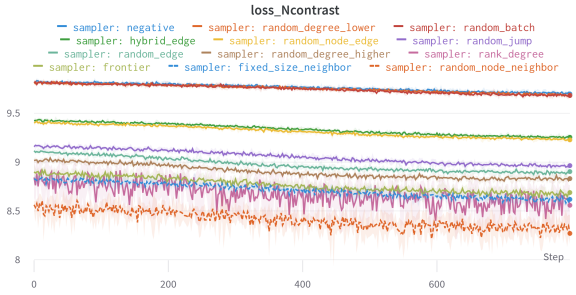

Figure 6: NContrast Loss on Facebook

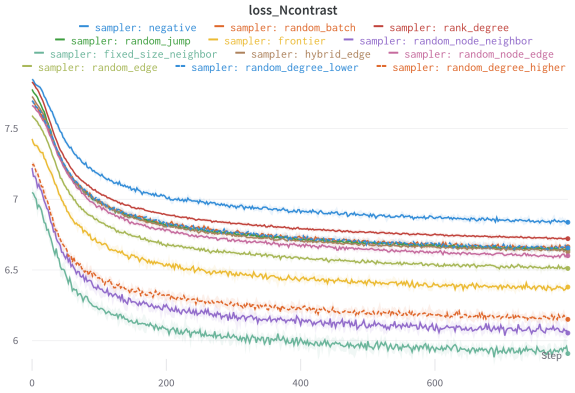

Figure 4: NContrast Loss on Cora